Table of Contents

1. Introduction to Computer Science 26
1.1. What is Computer Science? 26
1.2. Computer Science Applications 27
1.3. Local Job Market 27
1.4. International Job Market 28
1.5. Are you not a student of Computer Science? 28
2. Breadth First Learning 29
2.1. Search Engine Usage Techniques 29
2.2. History of Computing 29
2.3. Data Storage 29
2.4. Data Manipulation 29
2.5. Operating System 29
2.6. Networking and the Internet 29
2.7. Algorithms 29
2.8. Programming Languages 29
2.9. Software Engineering 29
2.10. Data Abstraction 29
2.11. Database Systems 30
2.12. Artificial Intelligence 30
2.13. CS impact on society 30
2.14. Content Filtering, Spam, International laws 30
2.15. Word Processing 30
2.16. Presentations Development 30
2.17. Spreadsheet 30
2.18. Database MS Access 30
2.19. Web Page Development 30
Module 3 31
3. Search Engines 31
3.1. Query 31
3.2. How Google Works 31
3.3. Try Searching on Google 32

Copyright Virtual University of Pakistan

3.4. Use of Microphone 32
3.5. Flip a Coin 32
3.6. Query Formulation 32
3.7. Capitalization 32
Module 4 33
4. Searching Tricks 33
4.1. Weather Searching 33
4.2. Performing Calculations on Google 33
4.3. Currency Conversion 34
Module 5 35
5. Search Operators (1) 35
5.1. Search on Social Media 35
5.2. Search for a price 35
5.3. Searching Hash tags 36
5.4. Exclude words from Query 36
5.5. Exact Match 36
5.6. Wild Card based Searching 36
Module 6 37
6. Search Operators (2) 37
6.1. Searching within the range 37
6.2. Boolean Operators 37
6.3. Search with a Specific Site 37
6.4. Searching Related Websites 37
6.5. Searching a cached version 37
Module 7 38
7. Search Operators (3) 38
7.1. Stocks Operator 38
7.2. Map Operator 38
7.3. Movie Operator 38
7.4. Compare Food 38
7.5. Define Operator 38
7.6. Image Search 38
7.7. Tilt 38

Copyright Virtual University of Pakistan

Module 8 39
8. Advanced Search Operators 39
8.1. Intitle 39
8.2. Allintitle 39
8.3. inurl and allinurl 39
8.4. Intext and allintext 39
8.5. Proximity Search 39
8.6. Let's solve a complex Query 39
Module 9 40
9. What we should not search on Internet 40
9.1. Avoiding Ads: 40
9.2. Dangerous to Search 40
9.3. Avoid Cyber-security attacks 40
9.4. Unpleasant Results 41
Module 10 42
10. Roots of Computing 42
10.1. Abacus 42
10.2. Technology of Gears 42
10.3. Punch Cards 43
10.4. ENIAC 44
10.5. Factsheet of ENIAC 44
10.6. Rapid Advancement 44
Module 11 46
11. Bits 46
11.1. Basics 46
11.2. What Bits can represent: 46
11.3. Bits units 46
11.4. Patterns Using Bits 46
Module 12 48
12. Boolean Operations 48
12.1. AND Boolean Operation 48
12.2. OR Boolean Operation 48
12.3. XOR Boolean Operation 48

Copyright Virtual University of Pakistan

12.4. Not Operation 48
12.5. Boolean Operation Example 49
Module 13 50
13. Hexadecimal Notation 50
13.1. Why we need Hexadecimal Notation 50
13.2. Hexadecimal Representation 50
13.3. Hexadecimal Example 50
Module 14 51
14. Storing a Bit 51
14.1. Main Memory 51
14.2. Main Memory Organization 51
14.3. Byte Size Organization 51
14.4. Memory Address 51
14.5. RAM 52
14.6. DRAM 52
14.7. SDRAM 52
Module 15 53
15. Magnetic Systems 53
15.1. Mass Storage 53
15.2. How it works 53
15.3. Zoned-bit recording 53
15.4. Seek Time 53
15.5. Rotation Delay 53
15.6. Access Time 53
15.7. Transfer rate 54
Module 16 55
16. Optical System 55
16.1. Compact Disk 55
16.2. DVD (Digital Versatile Disks) 55
16.3. BDs (Blue Ray Disks) 55
Module 17 55
17. Flash Drives 55
17.1. Issues in Magnetic and Optical Systems 55

Copyright Virtual University of Pakistan

17.2. Flash Drive Technology 56
17.3. SSDs (Solid State Disks) 56
17.4. SDs (Secure Digital Memory Cards) 56
Module 18 57
18. Representing Text 57
18.1. Representation as Code 57
18.2. ASCII Codes 57
18.3. ASCII code example 57
18.4. Limitation of ASCII codes 57
18.5. Limitations of ASCII-extensions 57
18.6. Unicode 57
18.7. UTF-8 57
Module 19 58
19. Representing Numeric Values 58
19.1. Issues in storing numeric as Unicode 58
19.2. Binary Notation 58
19.3. Binary Notation Variations 58
Module 20 59
20. Representing Images 59
20.1. Pixel 59
20.2. Encoding Method: Pixel to Bitmap 59
20.3. Encoding Method: Handling shades 59
20.4. Encoding Method: Colorful Images 59
20.5. Brightness Chrominance 59
20.6. Image Scaling 59
20.7. Geometric Structures for Image Scaling 59
20.8. Scalable Fonts 59
Module 21 60
21. Representing Sound 60
21.1. Sound Amplitude 60
21.2. How to encode the Sound 60
21.3. How Sound data communication takes place 60
21.4. Sample Intervals 60

Copyright Virtual University of Pakistan

21.5. Alternative Method: MIDI 60
Module 22 61
22. Binary Notation 61
22.1. Power Method 61
22.2. Algorithm for Finding Binary Representation of a Positive Decimal Number 61
Module 23 63
23. Binary Addition 63
Module 24 65
24. Fraction in Binary 65
24.1. Radix Point 65
24.2. Example of Radix Point 65
24.3. Addition in fraction 65
Module 25 66
25. 2's Complement Notation to Store Numbers 66
25.1. Integer Representation in 2's Complement 66
25.2. Conversion between positive and negative representations 66
25.3. Addition in 2's complement notation 66
25.4. Problem of Overflow 67
Module 26 68
26. Excess Notation 68
26.1. Integer Representations in Excess Notation 68
26.2. Excess Four Notation 68
26.3. Excess Four Notation Comparisons 68
Module 27 70
27. Floating Point Notation 70
27.1. Storing Radix 70
27.2. Storing Fractions 70
Module 28 71
28. Truncation Errors in Floating Point Notation 71
28.1. Primitive Methods for Handling Truncation Errors 72
28.2. Intelligent Processing for Handling Truncation Errors 72
Module 29 73
29. Data Compression: Generic Techniques 73

Copyright Virtual University of Pakistan

Module 30 75
30. Data Compression: Compressing Images 75
Module 31 77
31. Data Compression: Compressing Audio and Videos 77
Module 32 78
32. Data Manipulation: CPU Basic 78
Module 33 80
33. Data Manipulation: Stored Program 80
Module 34 81
34. Data Manipulation: CPU Architecture Philosophies 81
Module 35 82
35. Data Manipulation: Machine Instruction Categories 82
35.1. Data Transfer Group 82
35.2. Arithmetic/Logic Group 82
35.3. Control Group 82
Module 36 83
36. Data Manipulation: Program Execution 83
Module 37 84
37. Data Manipulation: Program Execution Example 84
Module 38 87
38. Data Manipulation: Logic Operators 87
Module 39 89
39. Data Manipulation: Rotation and Shift 89
Module 40 90
40. Data Manipulation: Arithmetic Operators 90
Module 41 91
41. Data Manipulation: Role of Controller 91
Module 42 92
42. Data Manipulation: Direct Memory Access and Handshaking 92
Module 43 93
43. Data Manipulation: Communication media and communication rates 93
43.1. Parallel Communication 93
43.2. Serial Communication 93

Copyright Virtual University of Pakistan

Module 44 94
44. Data Manipulation: Pipelining 94
Module 45 95
45. Operating Systems: History 95
Module 46 97
46. Operating Systems: Basic Concepts (I) 97
Module 47 98
47. Operating Systems: Basic Concepts (II) 98
Module 48 99
48. Operating Systems: Software Classification 99
Module 49 100
49. Operating Systems: Components (I) 100
Module 50 102
50. Operating Systems: Components (II) 102
51. Operating Systems: Process of Booting 104
Module 52 106
52. Operating Systems: Process and its Administration 106
Module 53 107
53. Operating Systems: Process and its Administration 107
Module 54 109
54. Operating Systems: Handling Competition between Processes 109
Module 55 110
55. Operating Systems: Semaphores 110
Module 56 112
56. Operating Systems: Deadlock 112
Module 57 114
57. Operating Systems: Security Attacks from outside 114
Module 58 115
58. Operating Systems: Security Attacks from inside 115
Module 59 117
59. Networking and the Internet: Network Classification 117
Module 60 118
60. Networking and the Internet: Protocols 118

Copyright Virtual University of Pakistan

Module 61 120
61. Networking and the Internet: Combining Networks 120
Module 62 122
62. Networking and the Internet: Methods of Process Communication 122
Module 63 124
63. Networking and the Internet: Distributed Systems 124
Module 64 125
64. Networking and the Internet: Internet Architecture 125
Module 65 127
65. Networking and the Internet: Internet Addressing 127
Module 66 129
66. Networking and the Internet: Internet Applications 129
Module 67 130
67. Networking and the Internet: Internet Applications: Email 130
Module 68 131
68. Networking and the Internet: VoIP 131
Module 69 132
69. Networking and the Internet: Internet Multimedia Streaming 132
Module 70 133
70. Networking and the Internet: World Wide Web 133
Module 71 134
71. Networking and the Internet: Web implementations 134
Module 72 135
72. Networking and the Internet: HTML 135
Module 73 136
73. Networking and the Internet: HTML 136
Module 74 138
74. Networking and the Internet: More on HTML 138
Module 75 139
75. Networking and the Internet: XML 139
Module 76 141
76. Networking and the Internet: Client Side and Server Side 141
Module 77 143

Copyright Virtual University of Pakistan

77. Networking and the Internet: Layered Approach to Internet Software (I) 143
Module 78 145
78. Networking and the Internet: Layered Approach to Internet Software (II) 145
Module 79 146
79. Networking and the Internet: Layered Approach to Internet Software (III) 146
Module 80 148
80. Networking and the Internet: TCP /IP Protocol Suite 148
Module 81 150
81. Networking and the Internet: Security (Forms of Attacks) 150
Module 82 152
82. Networking and the Internet: Protection and Cures 152
Module 83 154
83. Networking and the Internet: Encryption 154
Module 84 156
84. Networking and the Internet: Legal Approaches to Network Security 156
Module 85 158
85. Algorithm: an Informal Review 158
Module 86 159
86. Algorithm: Formal Definition of Algorithm 159
Module 87 160
87. Algorithm: Abstract Nature of Algorithms 160
Module 88 161
88. Algorithm: Representation (Primitives) 161
Module 89 163
89. Algorithm: Representation (Pseudocode) 163
Module 90 165
90. Algorithm: Representation (Pseudocode) While-Structure 165
Module 91 166
91. Algorithm: Representation (Pseudocode) Function-Structure 166
Module 92 167
92. Algorithm: Discovery (The Art of Problem Solving) 167
Module 93 169
93. Algorithm: Getting your Foot in the Door 169

Copyright Virtual University of Pakistan

Module 94 170
94. Algorithm: Algorithm Discovery Strategies (I) 170
Module 95 171
95. Algorithm: Algorithm Discovery Strategies (II) 171
Module 96 172
96. Algorithm: Iterative Structures (Sequential Search Algorithm) 172
Module 97 174
97. Algorithm: Iterative Structures (Loop Control) 174
Module 98 175
98. Algorithm: Iterative Structures (Components of Repetitive Control) 175
Module 99 176
99. Algorithm: Iterative Structures: Loop Execution (Examples-1) 176
Module 100 177
100. Algorithm: Iterative Structures: Loop Execution (Examples-II) 177
Module 101 179
101. Algorithm: Iterative Structures (Pretest and Posttest loops) 179
Module 102 180
102. Algorithm: Insertion Sort Algorithm 180
Module 103 181
103. Algorithm: Insertion Sort Algorithm Example 181
Module 104 182
104. Algorithm: Recursive Structure (The Binary Search Algorithm) 182
Module 105 185
105. Algorithm: Recursive Control 185
Module 106 186
106. Algorithm: Algorithm Efficiency 186
Module 107 188
107. Algorithm: Software Verification 188
Module 108 190
108. Algorithm: Software Verification Examples 190
Module 109 190
109. Programming Languages: Early Generations-I 190
Module 110 192

Copyright Virtual University of Pakistan

110. Programming Languages: Early Generations-II 192
Module 111 194
111. Programming Languages: Machine Independence 194
Module 112 195
112. Programming Languages: Imperative Paradigms 195
Module 113 197
113. Programming Languages: Declarative Paradigms 197
Module 114 198
114. Programming Languages: Functional Paradigm 198
Module 115 200
115. Programming Languages: Object Oriented Paradigm 200
Module 116 201
116. Programming Languages: Variable and Data Types 201
Module 117 203
117. Programming Languages: Data Structure 203
Module 118 205
118. Programming Languages: Assignment Statement 205
Module 119 206
119. Programming Languages: Control Structures (if-statement) 206
Module 120 207
120. Programming Languages: Control Structures (if-statement examples) 207
Module 121 208
121. Programming Languages: Control Structures (Loops) 208
Module 122 209
122. Programming Languages: Programming Concurrent Activities 209
Module 123 211
123. Programming Languages: Arithmetic Operators Examples 211
Module 124 211
124. Programming Languages: Relational Operators Examples 211
Module 125 213
125. Programming Languages: Logical Operators Examples 213
Module 126 215
126. Software Engineering: Software Engineering Discipline 215

Copyright Virtual University of Pakistan

Module 127 217
127. Software Engineering: Software Life cycle 217
Module 128 218
128. Software Engineering: Requirement Analysis Phase 218
Module 129 219
129. Software Engineering: Design Phase 219
Module 130 220
130. Software Engineering: Implementation Phase 220
Module 131 221
131. Software Engineering: Testing Phase 221
Module 132 222
132. Software Engineering: Software Engineering Methodologies (I) 222
Module 133 223
133. Software Engineering: Software Engineering Methodologies (II) 223
Module 134 224
134. Software Engineering: Software Engineering Methodologies (II) 224
Module 135 225
135. Software Engineering: Coupling 225
Module 136 227
136. Software Engineering: Cohesion 227
Module 137 228
137. Software Engineering: Information Hiding 228
Module 138 229
138. Software Engineering: Components 229
Module 139 230
139. Software Engineering: Design Patterns 230
Module 140 231
140. Software Engineering: Design Patterns Examples 231
Module 141 233
141. Software Engineering: Scope of Quality Assurance 233
Module 142 234
142. Software Engineering: Software Testing 234
Module 143 236

Copyright Virtual University of Pakistan

143. Software Engineering: Documentation 236
Module 144 238
144. Software Engineering: Human Machine Interface 238
Module 145 241
145. Software Engineering: Software Ownership and Liability 241
Module 146 243
146. Data Abstraction: Arrays and Aggregates 243
Module 147 243
147. Data Abstraction: List, Stacks and Queues 243
Module 148 245
148. Data Abstraction: Trees 245
Module 149 247
149. Data Abstraction: Pointers 247
Module 150 248
150. Database Systems: The Significance of Database Systems 248
Module 151 250
151. Database Systems: Role of Schema 250
Module 152 251
152. Database Systems: Database Management Systems 251
Module 153 253
153. Database Systems: Relational Database model 253
Module 154 254
154. Database Systems: Issues of Relational Designs 254
Module 155 256
155. Database Systems: Relational operators 256
Module 156 257
156. Database Systems: Select Operation 257
Module 157 258
157. Database Systems: project Operation 258
Module 158 259
158. Database Systems: Join Operation 259
Module 159 262
159. Database Systems: Object Oriented Databases 262

Copyright Virtual University of Pakistan

Module 160 264
160. Database Systems: Maintaining DB Integrity 264
Module 161 265
161. Database Systems: The Commit/Rollback Protocol 265
Module 162 266
162. Database Systems: Locking 266
Module 163 268
163. Database Systems: Sequential Files 268
Module 164 270
164. Database Systems: Indexed Files 270
Module 165 271
165. Database Systems: Hash Files 271
Module 166 272
166. Database Systems: Hash File Example 272
Module 167 274
167. Database Systems: Data Mining 274
Module 168 275
168. Database Systems: Data Mining Examples and Implications 275
Module 169 276
169. Database Systems: Social Impact of Database Technology 276
Module 170 278
170. Artificial Intelligence: Introduction and Vision 278
Module 171 281
171. Artificial Intelligence: Intelligent Agents 281
Module 172 284
172. Artificial Intelligence: Research Methodologies 284
Module 173 285
173. Artificial Intelligence: The Turing Test 285
Module 174 286
174. Artificial Intelligence: Understanding Images 286
Module 175 287
175. Artificial Intelligence: Language Processing 287
Module 176 289

Copyright Virtual University of Pakistan

176. CS Impact: CS impact on Society 289
Module 177 290
177. CS Impact: CS Impact on Health 290
Module 178 291
178. CS Impact: CS Impact on Environment 291
Module 179 292
179. CS Impact: Ethical Issues 292
Module 180 293
180. CS Impact: Software Licenses and Information Privacy 293
Module 181 294
181. CS Impact: Intellectual Property 294
Module 182 296
182. CS Impact: Security 296
Module 183 297
183. CS Impact: Privacy 297
Module 184 299
184. CS Impact: Social Issues of IT 299
Module 185 301
185. CS Impact: Content Filtering, Email-Spams and Laws 301
Module 186 302
186. CS Impact: Children Protection and Electronic Theft 302
Module 187 304
187. Word Processing: MS Word 304
Module 188 308
188. Word Processing: MS Word (Quick Access bar) 308
Alternate Method -- Undo & Redo by Using Keys 308
Module 189 309
189. Word Processing: MS Word (Home Ribbon) 309
Module 190 310
190. Word Processing: MS Word (Clipboard Group) 310
I want to move. I am content where I am. 311
Module 191 314
191. Word Processing: MS Word (Font Group) 314

Copyright Virtual University of Pakistan

Module 192 316
192. Word Processing: MS Word (Paragraph Group-I) 316
Module 193 318
193. Word Processing: MS Word (Paragraph Group-II) 318
Module 194 320
194. Word Processing: MS Word (Style Group) 320
Module 195 321
195. Word Processing: MS Word (Editing Group) 321
Module 196 322
196. Word Processing: MS Word (Insert Functionalities) 322
Module 197 323
197. Word Processing: MS Word (page Group) 323
Module 198 325
198. Word Processing: MS Word (Table Group) 325
Module 199 326
199. Word Processing: MS Word (Illustration Group) 326
Module 200 330
200. Word Processing: MS Word (Media and Links Groups) 330
Module 201 332
201. Word Processing: MS Word (Comments and Header & footer) 332
Module 202 333
202. Word Processing: MS Word (Text Group (Part-1)) 333
Module 203 335
203. Word Processing: MS Word (Text Group (Part-II) and Symbols Group) 335
Module 204 336
204. Word Processing: MS Word (Design Ribbon) 336
Module 205 338
205. Word Processing: MS Word (Page Setup Group in Page Layout Ribbon) 338
Module 206 339
206. Word Processing: MS Word (Page Setup Group in Page Layout Ribbon) 339
Module 207 340
207. Word Processing: MS Word (Arrange Group in Page Layout Ribbon) 340
Module 208 341

Copyright Virtual University of Pakistan

208. Word Processing: MS Word (References Ribbon) 341
Module 209 342
209. Word Processing: MS Word (Proofing Group in Review Ribbon) 342
Module 210 343
210. Word Processing: MS Word (Language Group in Review Ribbon) 343
Module 211 344
211. Word Processing: MS Word (Comments Group in Review Ribbon) 344
Module 212 345
212. Word Processing: MS Word (Tracking and Changes Groups in Review Ribbon) 345
Module 213 346
213. Word Processing: MS Word (Compare and Protect Groups in Review Ribbon) 346
Module 214 347
214. Word Processing: MS Word (View Ribbon) 347
Module 215 349
215. Presentations: MS-PowerPoint (Introduction) 349
Module 216 350
216. Presentations: MS-PowerPoint (Slides Group on Home Ribbon) 350
Module 217 350
217. Presentations: MS-PowerPoint (Design Ribbon) 350
Module 218 351
218. Presentations: MS-PowerPoint (Transition Ribbon) 351
Module 219 352
219. Presentations: MS-PowerPoint (Animation Ribbon) 352
Module 220 353
220. Spreadsheets: MS Excel (Introduction) 353
Module 221 355
221. Spreadsheets: MS Excel (Functions) 355
Module 222 358
222. Spreadsheets: MS Excel (Application Scenarios-I) 358
Module 223 359
223. Spreadsheets: MS Excel (Application Scenarios-II) 359
Module 224 361
224. Spreadsheets: MS Excel (Sorting and Filter) 361

Copyright Virtual University of Pakistan

Module 225 363
225. Database: MS Access (Introduction) 363
Module 226 365
226. Database: MS Access (Creating and Managing Tables) 365
Module 227 366
227. Database: MS Access (Creating Forms) 366
Module 228 367
228. Database: MS Access (Creating Reports) 367
Module 229 368
229. Database: MS Access (Query Wizard) 368
Module 230 372
230. Web page Development (Notepad Editor) 372
Module 231 374
231. Web page Development (Introduction to Dreamweaver) 374
Module 232 377
232. Web page Development (Inserting Tables using Dreamweaver) 377
Module 233 380
233. Web page Development (Inserting Lists) 380
Module 234 381
234. Web page Development (Inserting Images) 381

Table of Figures:

Figure 1: Job Opportunities in Pakistan when classified as "By Function" 27

Copyright Virtual University of Pakistan

Figure 2: Job Opportunities in Pakistan when classified as “By Industry” 28
Figure 3: Market share of Search Engines [gs.statcounter.com/search-engine-market-share] 31
Figure 4: Google Search Engine home page 31
Figure 5: Weather searching on Google 33
Figure 6: Calculations on Google 33
Figure 7: Currency conversion on Google 34
Figure 8: Searching in Social media over Google 35
Figure 9: searching for a price at Google 35
Figure 10: Searching Hash tag 36
Figure 11: Abacus 42
Figure 12: Gear Technology 43
Figure 13: Punch Cards 43
Figure 14: ENIAC 44
Figure 15: AND, OR Boolean operation 48
Figure 16: XoR, Not operation 49
Figure 17: Flip flop 51
Figure 18: Byte organization in memory 51
Figure 19: Memory arranged with respect to addresses 52
Figure 20: Disk Storage System 53
Figure 21: Data storage in CD 55
Figure 22: ASCII code example 57
Figure 23: Encoding the amplitude of the sound 60
Figure 24: Algorithm for converting positive decimal to binary number 61
Figure 25: Algorithm working for converting positive decimal to binary 62
Figure 26: Binary Addition basics 63
Figure 27: Example 1 of Binary Addition 63
Figure 28: Example 2 of Binary Addition 63
Figure 29: Example 3 of Binary Addition 64
Figure 30: Radix point example 65
Figure 31: Understanding Fraction represented in Binary 65
Figure 32: Binary addition 65
Figure 33: 2's complement notation for three bits patterns 66
Figure 34: 2's complement notation for four bits patterns 66
Figure 35: Addition in 2's complement notation 67
Figure 36: Excess eight notation 68
Figure 37: Excess Four notation 68
Figure 38: Excess four notation comparisons 69
Figure 39: Floating-point notation components 70
Figure 40: Encoding the value $2^{5/8}$ 72
Figure 41: CPU and main memory connected via a bus 78
Figure 42: Adding values stored in memory 79
Figure 43: CPU and Main memory linkage using Bus 83
Figure 44: Machine Cycle 83
Figure 45: Machine Instructions for adding two numbers 84

Copyright Virtual University of Pakistan

Figure 46: Machine instructions loaded in the main memory 85
Figure 47: Arithmetic Operations Examples 90
Figure 48: Controllers attached to a machine's bus 91
Figure 49: Serial communication 93
Figure 50: Batch Processing 95
Figure 51: Interactive Processing 96
Figure 52: Time Sharing 97
Figure 53: Software Classification 99
Figure 54: Components of OS 101
Figure 55: process of Booting 104
Figure 56: Multiprogramming between process A and process B 107
Figure 57: Resource Allocation Issues 109
Figure 58: Deadlock Situation 112
Figure 59: Two popular network topologies 117
Figure 60: Communication over a bus network 118
Figure 61: The hidden terminal problem 119
Figure 62: Building a large bus network from smaller ones 120
Figure 63: Routers connecting two WiFi networks and an Ethernet network to form an internet 121
Figure 64: The client/server model compared to the peer-to-peer model 122
Figure 65: Internet composition 125
Figure 66: Email procedure 130
Figure 67: A typical URL 134
Figure 68: A simple webpage 136
Figure 69: An enhanced simple webpage 137
Figure 70: <b> tag on w3school 138
Figure 71: Code and output for <b> tag 138
Figure 72: The first two bars of Beethoven's Fifth Symphony 139
Figure 73: Package-shipping example 143
Figure 74: The Internet software layers 144
Figure 75: Following a message through the Internet 145
Figure 76: Choosing between TCP and UDP 148
Figure 77: Public key encryption 155
Figure 78: Folding a bird from a square piece of paper 161
Figure 79: Origami primitives 162
Figure 80: The function Greetings in pseudocode 166
Figure 81: Analyzing the possibilities 168
Figure 82: The sequential search algorithm in pseudocode 173
Figure 83: Components of repetitive control 175
Figure 84: Pretest and Posttest loop 179
Figure 85: Sorting the list Fred, Alex, Diana, Byron, and Carol alphabetically 181
Figure 86: Applying our strategy to search a list for the entry John 183
Figure 87: A first draft of the binary search technique 183
Figure 88: The binary search algorithm in pseudocode 184
Figure 89: Separating the chain using only three cuts 188

Copyright Virtual University of Pakistan

Figure 90: Solving the problem with only one cut 189
Figure 91: Generations of programming languages 195
Figure 92: The evolution of programming paradigms 196
Figure 93: A function for checkbook balancing constructed from simpler functions 199
Figure 94: A two-dimensional array with two rows and nine columns 203
Figure 95: The conceptual layout of the structure Employee 204
Figure 96: Spawning threads 210
Figure 97: The software life cycle 217
Figure 98: The traditional development phase of the software life cycle 218
Figure 99: A simple structure chart 224
Figure 100: The interaction between objects resulting from PlayerA's serve. 225
Figure 101: Logical and functional cohesion within an object 227
Figure 102: Factory Design Pattern 231
Figure 103: Shopping Cart Design Pattern 232
Figure 104: Lists, stacks, and queues 244
Figure 105: An example of an organization chart 245
Figure 106: Tree terminology 245
Figure 107: Novels arranged by title but linked according to authorship 247
Figure 108: A file versus a database organization 248
Figure 109: The conceptual layers of a database implementation 251
Figure 110: A relation containing employee information 253
Figure 111: A relation containing redundancy 254
Figure 112: An employee database consisting of three relations 255
Figure 113: The SELECT operation 257
Figure 114: The PROJECT operation 258
Figure 115: The Join Operation 259
Figure 116: Another example of the JOIN operation 260
Figure 117: An application of the JOIN operation 261
Figure 118: The associations between objects in an object-oriented database 262
Figure 119: The structure of a simple employee file implemented as a text file 268
Figure 120: Opening an indexed file 270
Figure 121: Hashing the key field value 25X3Z to one of 41 buckets 272
Figure 122: The rudiments of a hashing system 273
Figure 123: Robot 278
Figure 124: Self Driving Cars 278
Figure 125: Medical robot helping the patients 279
Figure 126: Receptionist Robot 279
Figure 127: Robot draws blood 280
Figure 128: The eight-puzzle in its solved configuration 282
Figure 129: Our puzzle-solving machine 283
Figure 130: Start Menu to launch MS Word 304
Figure 131: MS Word Home ribbon 304
Figure 132: MS Word home ribbon after click 305
Figure 133: Font Panel 306

Copyright Virtual University of Pakistan

Figure 134: Bold Clicked in Font Panel 306
Figure 135: Font Panel Changing Case 306
Figure 136: Underline Panel 307
Figure 137: Panel Launch Buttons 307
Figure 138: Mini Toolbar 307
Figure 139: Quick Access toolbar 308
Figure 140: Home ribbon 309
Figure 141: Clipboard Group 310
Figure 142: Cut option in clipboard group 311
Figure 143: Pasting in Document 311
Figure 144: Pasting in Document (Alternate Method) 312
Figure 145: Pasting in Document (Alternate Method) 312
Figure 146: Font group 314
Figure 147: Typing your name 314
Figure 148: Changing font 314
Figure 149: Applying bold and underline 315
Figure 150: Changing Font Size 315
Figure 151: Paragraph Group 316
Figure 152: Numbers varriations in paragraph group 316
Figure 153: Bullets variations in paragraph group 317
Figure 154: Bullet selection 318
Figure 155: Adding Numbered Lists 319
Figure 156: Style Group 320
Figure 157: Editing group 321
Figure 158: Inert tab 322
Figure 159: Pages group in the insert tab 323
Figure 160: Inserting Cover Page 323
Figure 161: Removing cover page 324
Figure 162: Tables group 325
Figure 163: Selecting Rows and Columns 325
Figure 164: Illustrations 326
Figure 165: Shapes types in Word 327
Figure 166: Shapes required to make snowman 327
Figure 167: Snowman 327
Figure 168: Smart Art 328
Figure 169: Basic cycle 328
Figure 170: Human Life Cycle 329
Figure 171: Adding and Removing Objects 329
Figure 172: Links group 330
Figure 173: Adding Hyperlink 330
Figure 174: Adding Bookmark 331
Figure 175: Header and Footer 332
Figure 176: Header in edit mode 332
Figure 177: After Header Insertion 332

Copyright Virtual University of Pakistan

Figure 178: Text group 333
Figure 179: Adding Text Box 333
Figure 180: Word Art Gallery 334
Figure 181: Drop cap 'P' 335
Figure 182: Design Ribbon 336
Figure 183: Page background group 336
Figure 184: Page setup group 338
Figure 185: Text selections for 2-columns display 338
Figure 186: Text in two columns 338
Figure 187: Paragraph group in page layout 339
Figure 188: Arrange Group 340
Figure 189: Table of Content 341
Figure 190: Footnotes Group 341
Figure 191: Proofing group 342
Figure 192: Language group 343
Figure 193: Comments group 344
Figure 194: Tracking group 345
Figure 195: Changes group 345
Figure 196: Compare group 346
Figure 197: Protect group 346
Figure 198: Views group 347
Figure 199: Show group 347
Figure 200: Zoom group 347
Figure 201: Windows group 348
Figure 202: MS: Powerpoint 349
Figure 203: Slides group in home ribbon 350
Figure 204: Themes group 350
Figure 205: Transition group 351
Figure 206: Transition timing group 351
Figure 207: Animations group 352
Figure 208: Spread Sheet Example 353
Figure 209: Excel Layout 354
Figure 210: Cell formatting 354
Figure 211: Charts group 355
Figure 212: Chart example in Excel 355
Figure 213: Formula Example 355
Figure 214: Formula ribbon 356
Figure 215: Data for Sum Formula 356
Figure 216: Sum Formula 356
Figure 217: Result of Sum Formula 357
Figure 218: Function Computation 357
Figure 219: Sum of marks 358
Figure 220: Result of sum formula 358
Figure 221: Personal expense sheet 359

Copyright Virtual University of Pakistan

Figure 222: Finding total expense for the January month 359
Figure 223: Result of query from Figure 222 360
Figure 224: Total bill Expense-head wise 360
Figure 225: Sort and Filter 361
Figure 226: Custom Sort 362
Figure 227: MS Access ribbon 363
Figure 228: Create Blank Database 363
Figure 229: Naming and saving database 364
Figure 230: Blank Database 364
Figure 231: External data Ribbon 364
Figure 232: Create tab 365
Figure 2334: Saved Form "studentForm" 366
Figure 234: Insert Report 367
Figure 235: Student Table Report 367
Figure 236: Query Group: Query Wizard 368
Figure 237: Creating Query using wizard 369
Figure 238: Query Result after saving 369
Figure 239: SQL View 370
Figure 240: SQL View Query Syntax 370
Figure 241: Query Design View 371
Figure 242: Create Query Using Design View 371
Figure 243: HTML code in Notepad 373
Figure 244: Saving as html page 373
Figure 245: First web page output 373
Figure 246: Dreamweaver interface 375
Figure 247: Properties panel 376
Figure 248: Add new page 377
Figure 249: Adding HTML page 377
Figure 250: Naming the page 378
Figure 251: Insert Table 378
Figure 252: Rows and columns 378
Figure 253: Adjusting row sizes 379
Figure 254: Adding unordered list 380
Figure 255: List of properties 380
Figure 256: Inserting Image 381
Figure 257: Browse Image 381

Module 1

Copyright Virtual University of Pakistan

Download More Highlighted Handouts From
VUAnswer.com

1. Introduction to Computer Science

1.1. What is Computer Science?

Computer Science is the discipline that seeks to build a scientific foundation for such topics as:

Hardware

Computer hardware is the collection of physical parts of a computer system. This includes the computer case, monitor, keyboard, and mouse. It also includes all the parts inside the computer case, such as the hard disk drive, motherboard, video card, and many others. Computer hardware is what you can physically touch.

Software

Computer software, also called software, is a set of instructions and its documentations that tells a computer what to do or how to perform a task. Software includes all different software programs on a computer, such as applications and the operating system.

Programing

Computer programming is the process of designing and building an executable computer program for accomplishing a specific computing task.

Networks

A computer network is a set of computers connected for the purpose of sharing resources. The most common resource shared today is connection to the Internet. Other shared resources can include a printer or a file server. The Internet itself can be considered a computer network.

Graphics

Computer graphics is the discipline of generating images with the aid of computers. Today, computer graphics is a core technology in digital photography, film, video games, cell phone and computer display, and many specialized applications.

Robots

A robot is a machine—especially one programmable by a computer— capable of carrying out a complex series of actions automatically. Robots can be guided by an external control device or the control may be embedded within

Database

Database, also called electronic database, any collection of data, or information, that is specially organized for rapid search and retrieval by a computer. Databases are structured to facilitate the storage, retrieval, modification, and deletion of data in conjunction with various data-processing operations. A database management system (DBMS) extracts information from the database in response to queries.

Security

Security are those controls that are put in place to provide confidentiality, integrity, and availability for all components of computer systems. These components include data, software, hardware, and firmware.

Algorithmic Solutions

Copyright Virtual University of Pakistan

An algorithm is a set of instructions designed to perform a specific task.

Information Processing

Information processing refers to the manipulation of digitized information by computers and other digital electronic equipment, known collectively as information technology (IT). Information processing systems include business software, operating systems, computers, networks and mainframes.

We will be learning the details of these terms throughout the courses in different modules.

1.2. Computer Science Applications

Furthermore, Computer Science has applications in almost all domains such as:

- ✓ Telecom
- ✓ Banks
- ✓ Hospitals
- ✓ Software Development
- ✓ Service Industry
- ✓ Pak Army
- ✓ Freelancing
- ✓ and many more

Download More Highlighted Handouts From

VUAnswer.com

1.3. Local Job Market

According to famous Job market website in Pakistan, most of the jobs are available in Computer Science, for example Figure 1 shows job opportunities when filtered using "By Function", and Figure 2 represents job opportunities when filtered using "By Industry". In both cases, the job opportunities in Computer Science are higher than rest of the fields.



Figure 1: Job Opportunities in Pakistan when classified as "By Function"

Copyright Virtual University of Pakistan

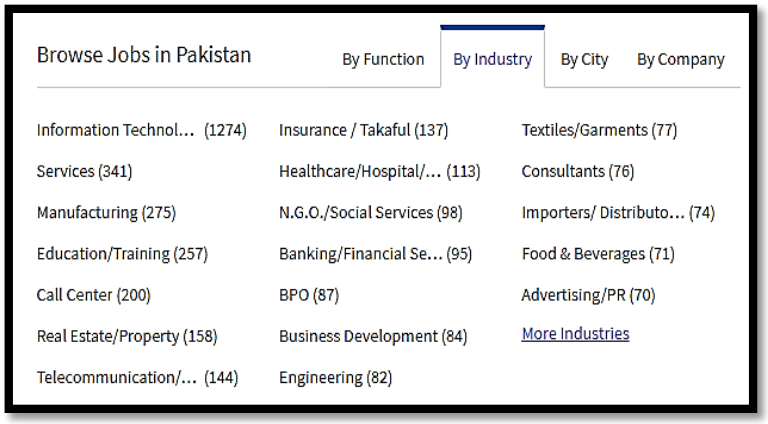


Figure 2: Job Opportunities in Pakistan when classified as "By Industry"

1.4. International Job Market

Similarly, internationally, jobs related to Computer Science are ranked on the top. For example, forbes magazine[1], one of the acclaimed agencies in US claims that Software Developer has been ranked as Number 1 job in the US. In Computer Science, the following areas have been ranked by the Forbes magazine.

- ✓ Artificial Intelligence and Machine Learning
- ✓ Data Science
- ✓ Virtual Reality
- ✓ IoT
- ✓ Back-End Developer
- ✓ Front-End Developer
- ✓ UI Designer
- ✓ Full-Stack Engineer
- ✓ IT Manager
- ✓ Quality Assurance Expert

All of this discussion would be helpful to make you motivated all the time and to be happy with the decision you have made to choose Computer Science and a career.

1.5. Are you not a student of Computer Science?

Those who are not studying CS, even then this course will be helpful for them too. As this course covers all basic concepts of Computer Science which you would require in whatever field of study you are working. You know studying basics of Computer Science is compulsory for everyone even you are studying Business, Engineering, or Sciences. This course has been made very easy and interactive to make sure that you learn properly the basics of Computer Science and can apply them in your own studies.

[1] https://www.forbes.com/sites/richardgano/2018/01/29/the-best-jobs-for-2018-and-beyond/#1940306742c4 accessed on 16th July

Copyright Virtual University of Pakistan

Module 2

2. Breadth First Learning

This course will give basic introduction to almost all courses of Computer Science. Such strategy of learning is known as Breadth First Learning. The other strategy is called Depth First Learning in which one particular course is first covered in details and then next course is covered. The Breadth First Learning helps students to first know what they will be learning in the whole degree program.

We will cover:

1. Abstract view of all major courses in CS
2. Understanding what will be studied in CS
3. Why each course is important
4. Clarifying the bigger Picture

Introduction to the topics will be covered in this course:

2.1. Search Engine Usage Techniques

We will cover the searching techniques in this topic. How can you effectively search over the internet using popular search engine?

2.2. History of Computing

We will cover in this topic that how todays computer evolved from the basic idea.

2.3. Data Storage

Whatever data you enter the computer, this data need to be stored somewhere in the hardware, such storage science will be understood in this topic.

2.4. Data Manipulation

Then the next thing is to study how data is manipulated. For example, how basic arithmetic operations (+,-,*, /) are performed in computer and how some advance operators are performed.

2.5. Operating System

We will study about Operating System which is the overall in-charge of the computer system.

2.6. Networking and the Internet

We will also study how different computers communicate over the internet.

2.7. Algorithms

We will then study the fundamental concept of algorithm in Computer Science. We define algorithm as: "Set of Steps in a sequence to perform a certain task".

2.8. Programming Languages

Then we will study programming tools. How algorithm can be written to achieve the said goal in a computer programming. We will cover just basics of C++.

2.9. Software Engineering

Here we will cover that how a complete software is developed starting from the requirement gathering phase to designing, implementation and testing

2.10. Data Abstraction

In Computer Science, designing a complex and large system becomes a challenge when we discuss design at a great depth, data abstraction hides complexities from the designer and helps to comprehend things with more ease. Such things will be discussed in data abstraction like Arrays, Stack, Queue, and trees.

Copyright Virtual University of Pakistan

2.11. Database Systems

In Computer Science, we store and link data in an organized way using state-of-the-art Database Management Systems (DBMS). We will discuss database design, DBMS functionalities in these modules.

2.12. Artificial Intelligence

You know Computer is a dump device, cannot think of take inferential decisions. However, Artificial Intelligence (AI) is a specialized field of Computer Science that aims at building such Computer Systems which can act intelligently.

2.13. CS impact on society

We will also learn the impact of Computer Science on society, social setups and on humans.

2.14. Content Filtering, Spam, International laws

Here we will cover content filtering, dealing with the spams and international laws related to Computer Science in terms of data and privacy.

2.15. Word Processing

Word processor like Microsoft Word is an application software which helps to build editable word documents, we will cover all major functionalities of Microsoft Word.

2.16. Presentations Development

Presentation are built in specialized tools like Microsoft Power point. We will cover Microsoft PowerPoint tool in details to design and animate the presentation slides.

2.17. Spreadsheet

To perform certain types of calculations on data, there are some specialized software like Spreadsheet. One of such software is Microsoft Excel which will be covered in such modules.

2.18. Database MS Access

Database learnt above will be implemented using DBMS tool like Microsoft Access.

2.19. Web Page Development

We will also learn that how a web page can be made easily using built-in tools like Dreamweaver.

Copyright Virtual University of Pakistan

Module 3

3. Search Engines

Search Engine like Google index web pages and helps you to retrieve the relevant web pages based on your queries. Starting from today, in next few modules, we will focus on techniques and short-cuts to retrieve most relevant information from Google.

There are many search engines like Google, Yahoo, and MSN etc. However, we will focus on Google because it is most widely used search engine and has most of the market share of searching over the internet as illustrated from the Figure 3.



Figure 3: Market share of Search Engines [gs.statcounter.com/search-engine-market-share]

To search on Google, type: https://www.google.com/ on the web browser, you will be shown the screen like shown in the Figure 4.



Figure 4: Google Search Engine home page

3.1. Query

Query is the set of words given to the search engine for searching for example, if you are interested to find "Virtual University" on the internet, you will provide this the text box available in the Figure 4 and you will be given the relevant results.

3.2. How Google Works

When you search in the Google by typing any query in the text box shown in the Figure 4, Google finds all those pages which contains all the terms given in the query. Try the following queries and see the results:

Copyright Virtual University of Pakistan

3.3. Try Searching on Google

1. "Airport"
2. "Where is the closest airport"
3. "Virtual University in Islamabad"

[Note: If you find any query with double quotations like "", you should not give this query in double quotations on Google, I am attaching double quotations just to make sure that you can understand it is a query not a running text. Otherwise posting a query in double quotations has a particular meaning in Google which you will learn in next modules.]

3.4. Use of Microphone

There is another option of microphone as can be seen in the Figure 4 on the right side. If you click on you can search by speaking.

3.5. Flip a Coin

If you want to flip a coin and you do not have the coin to flip, you can use Google to do that. To achieve this just type the following on Google:

"Flip a Coin"

3.6. Query Formulation

Always remember, when you search, Google will try to identify the pages on which the provided term is available. For example, if your head is in pain and you type:

"Head Hurts"

It might not give you relevant results as when head is in pain, then its medical name is headache. So, your query should be:

"Headache "

Similarly, if you are interested to search the name of the head of the virtual university and type the query as:

"Head of the virtual university"

Google can give you wrong result for example the web page which contains the term "head" and "virtual university" that might be the head office of virtual university. The right query could be:

"Rector of virtual university"

As the head of virtual university is called rector, therefore in this way it will give you more chance to search the right person effectively.

3.7. Capitalization

Google does not distinguish the capital letters or small letters. This is called case-insensitive in computer science. For example, search

"COMPUTER SCIENCE" or "computer science"

Will not make a difference.

Copyright Virtual University of Pakistan

Module 4

4. Searching Tricks

In this module we will learn about searching tricks while searching on Google.

4.1. Weather Searching

You can simply add the location with the word "Weather" to form such query for example:

Query "Weather Lahore" will give you a screen shown in the Figure 5.

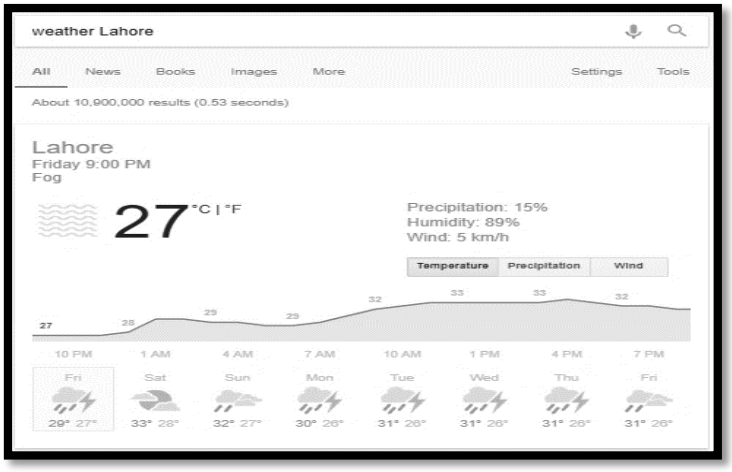


Figure 5: Weather searching on Google

4.2. Performing Calculations on Google

To perform calculations on Google, just type whatever you want to perform like if we want to multiply 12 with 391, we will type the query like:

12*391

and you can see in the screen shown in the Figure 6 that the Google has automatically started the calculator, have understood our query, have translated our query and given this query to calculator application and have shown the result as well.

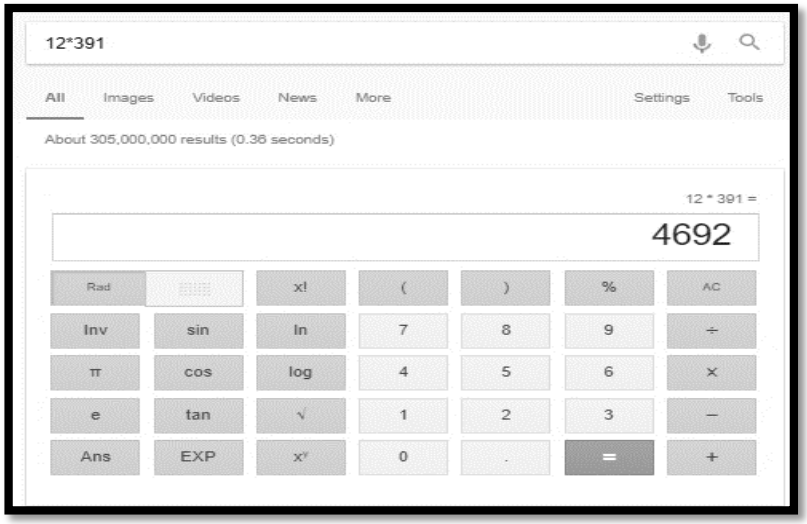


Figure 6: Calculations on Google

Copyright Virtual University of Pakistan

Perform the following queries on your computer:

1. 12-5
2. Sin 90
3. Tan 80
4. 80/100*200
5. Subtract 10 from 30

4.3. Currency Conversion

You can also see latest currency conversion on Google. For example, you want to see how much 100 Euros will be when converted to Pakistani rupees. Here is the query and result are shown in the Figure 7.

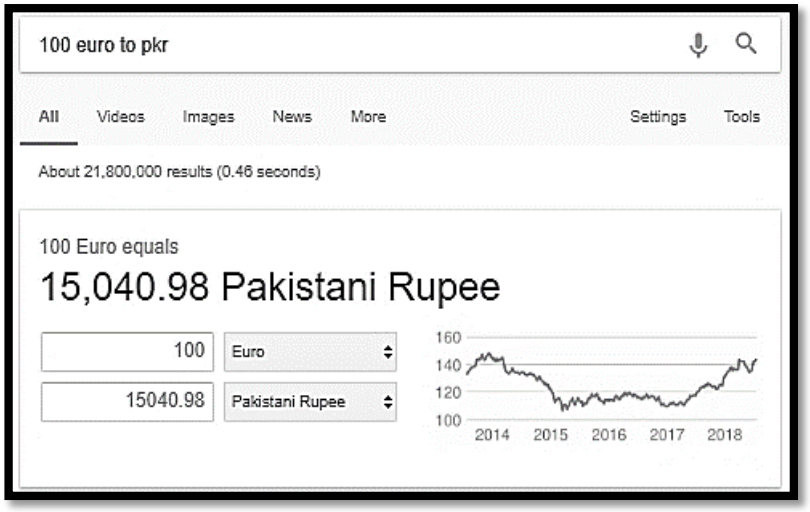


Figure 7: Currency conversion on Google

Try the following queries and see the results:

1. Kph in Mph
2. m in cm
3. Pakistan Cricket team
4. Baadshahi Mosque
5. Minar e Pakistan

Copyright Virtual University of Pakistan

Module 5

5. Search Operators (1)

In this module, we will further learn about different search operators that can be used on the Google to retrieve more relevant and focused results.

5.1. Search on Social Media

USE FOR SEARCHING FB INSTA AND OTHER:@

When you want to use Google and want to search the query on a particular social media, you can add "@" symbol. For example, when we give queries:

"Fifa World cup @facebook" and "Fifa World Cup @Twitter", the results can be seen in the Figure 8.

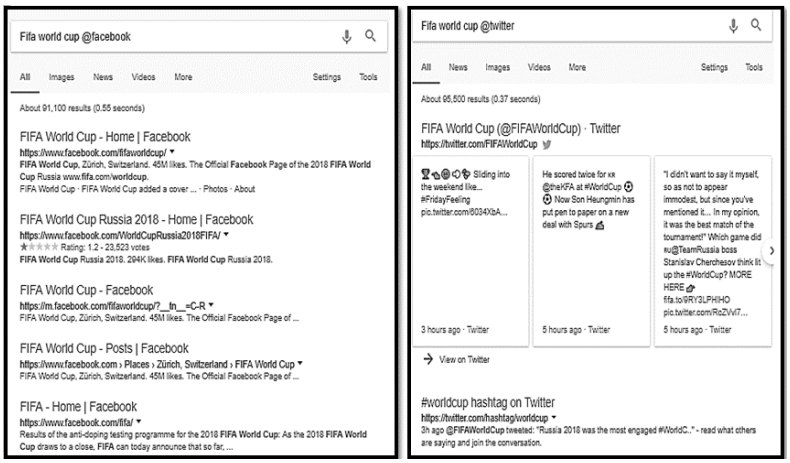


Figure 8: Searching in Social media over Google

5.2. Search for a price

Put pkr in front of a number. For example, give a query like:

"Laptop pkr 50000" it will give you laptops whose price is around 50000 as shown in the Figure 9.

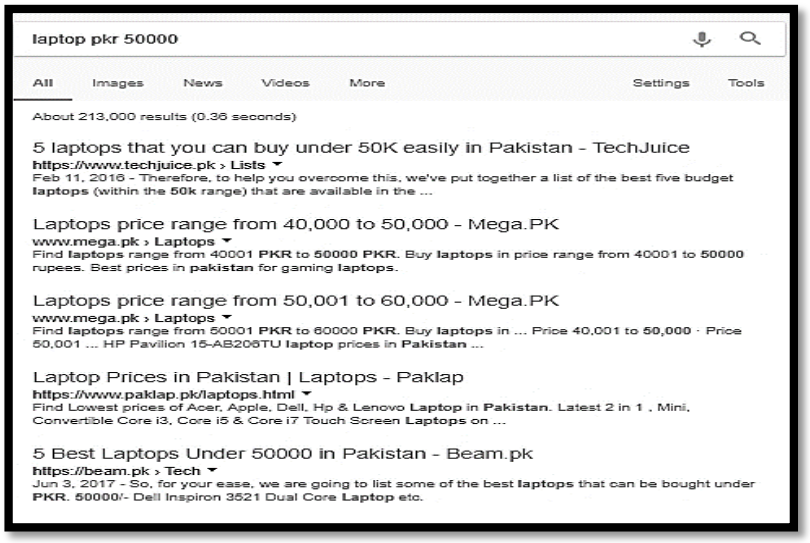


Figure 9: searching for a price at Google

Copyright Virtual University of Pakistan

5.3. Searching Hash tags TWITTER TRENDING:#

On Social media, sometimes, a hash tag is very popular, if you are interested to identify such pages using hash tag, you can use the query like:

"#education"

You will get result like shown in the Figure 10.

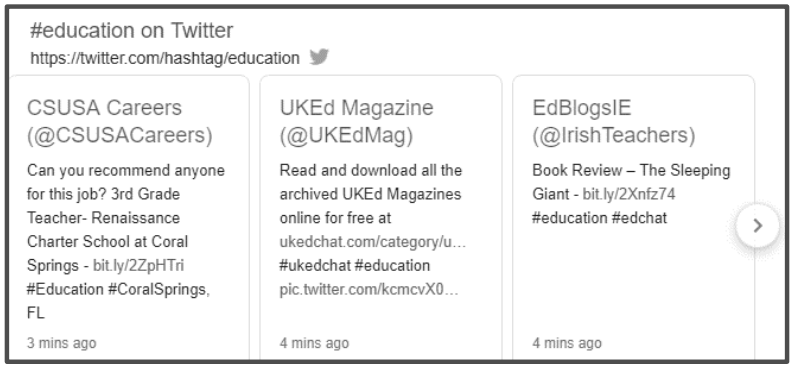


Figure 10: Searching Hash tag

5.4. Exclude words from Query " - "

One term could mean more than one thing for example word: "Jaguar" is used in two meanings such as: animal and car. If you want to exclude all pages containing pages of cars, you can write the query like "Jaguar -cars".

5.5. Exact Match ""

If you want to search for an exact phrase enclose it in double quotes, in this case, the search engine will search where it can find the exact phrase as it is. For example, searching

"Tallest Building in Pakistan" will give only those pages which contains exact this phrase, however, if you give the query like:

Tallest Building in Pakistan

Google will give you all those pages which contains all or any of these words.

5.6. Wild Card based Searching *

You can search a phrase when even you do not know what word would be there at a particular location for example, you want to search all those pages which contains any word in the start and then contains the exact phrase "is thicker than water". You can write:

"* is thicker than water"

This query will give you all those pages which contains any word in the start and then have the exact phrase "is thicker than water"

Please note if you search "is thicker than water", it might give you those pages where such question has been posed or there is no word in the start, but the phrase is written like "is thicker than water".

Copyright Virtual University of Pakistan

Module 6

6. Search Operators (2)

This module will further explore more search operators.

6.1. Searching within the range

If you want to search within the range of number, you can use two dots. For example: "laptop pkr25000. pkr35000" will try to give those pages which contains laptop options between the specified ranges.

6.2. Boolean Operators

We can use Boolean operators specially "And" "Or" to find relevant pages. For example, writing a query:

Computer and Science

Will give you all those pages which have both words but not necessarily they occur together.

Similarly writing a query:

Computer Or science

Will give you those pages in which either the term "computer" is written, or "science" is written. Off course, it will also return those pages in which both terms are also written.

6.3. Search with a Specific Site

If you are interested to search the query in a website, we can write like:

- ✓ "virtual university site: youtube.com"

This will give us those pages of virtual university which are contained within the youtube.com website.

6.4. Searching Related Websites

For example, you like the videos on YouTube and interested to know are there some more websites like YouTube? you can type the query:

- ✓ "related:youtube.com"

Furthermore, you can also search information for a particular website like if you write:

- ✓ "info:youtube.com"

It will provide the information about this website.

6.5. Searching a cached version

If the website is down and you are interested to see the cached version of the website, you can use the following query:

- ✓ cache:youtube.com

Searching on a file type

If you are interested to search a query for a specific file type, you can include such file type in the query. For example, we are interested to search for "virtual university" but all those pages having file type PDF, so you can write a query:

- ✓ "Virtual University" filetype:pdf
- ✓ "Virtual University" ext:pdf

Copyright Virtual University of Pakistan

Module 7

7. Search Operators (3)

In this module we will further learn about some remaining search operators.

7.1. Stocks Operator

Using this operator, we can find the trend of the stock market for a particular company for example, if we want to see what the stock market trend for the company "Apple" is, we can write the following query:

- ✓ stocks:aapl

7.2. Map Operator

If we are interested to see the map of some location, we can use map operator in the following way:

- ✓ map:Lahore

7.3. Movie Operator

Using movie operator, we can find information about any movie. For example, try:

- ✓ movie:steve jobs

7.4. Compare Food

If you want to compare two foods based on their nutrient values, you can go and explore the following website:

- ✓ https://www.myfooddata.com/

7.5. Define Operator

If you are interested to find the definition of a particular term, type:

- ✓ Define:Computer

7.6. Image Search

There is a dedicated web link at Google:

https://images.google.com/

Using this link, you can search images even by giving a query of another image. Google will provide you other images which look quite similar to the queried image.

7.7. Tilt

This is an interesting option available at Google. Using this option, you can perform number of funny things with the shown screen of the Google. For example, you can rotate the Google screen to 360 degree etc. To do it type the following query:

- ✓ Tilt

On the show results, select the appropriate link like: https://elgoog.im/tilt/ and then perform the interesting tricks with the Google screen.

Copyright Virtual University of Pakistan

Module 8

8. Advanced Search Operators

In this module, we will learn some advanced search operators.

8.1. Intitle

This operator will give us only those web pages in which the searched term appears in the title of the web pages. For example, if we are interested to search web pages in which the phrase "iPhone vs. android" appears in the title, we can write the following query.

- ✓ Intitle:"iPhone vs. android"

8.2. Allintitle

The next similar operator is allintitle without double quotation. This will find all those web pages in which any of the word "IPhone" or "android" is found. The query would be:

- ✓ allintitle:iphone vs. android

8.3. inurl and allinurl

The "inurl" operator finds all those web pages which contains the mentioned query in the url of the web pages. For example, if we try the following query:

- ✓ inurl:2018 "virtual university"

This will give us all those web pages in which 2018 is written and the phrase "virtual university" is written.

As we had allintitle above, here again we have allinurl which means it will find any of the those mentioned terms within the url and will fetch all those pages in which any of the mentioned query terms are present.

8.4. Intext and allintext

If we are interested to search anywhere in the body of the document, we can use intext operator, for example:

- ✓ intext:"virtual university admissions 2018"

this will give all those pages in which the whole phrase is present as it is. Furthermore, we can try:

- ✓ allintext:virtual university admissions 2018

Which will return all those pages in which any of those terms are available.

8.5. Proximity Search

Suppose we are interested to search two words:

- Education
- Virtual university

But both words should be collocated on the web page within the margin of three words. To achieve this, we can use the operator around.

- ✓ education AROUND(3) "virtual university"

This query will give all those web pages on which both words are available within the margin of three words.

8.6. Let's solve a complex Query

As we have learnt many operators, now let's solve a bit complex query. The query is:

"We are interested to see how many of pages of a website are not secured?"

- ✓ site:youtube.com –inurl:https

Copyright Virtual University of Pakistan

Module 9

9. What we should not search on Internet

We have learnt different tricks and operators to ease the task of searching on the Google. However, there are many things which we should not search on Google. But Why?

There are three main reasons:

1) Google Adds: Google give you advertisements based on your search queries, history, geographical location etc.
2) Security Agencies: law and enforcement agencies might see your queries and can enquire you or can reach at your location if they see a sensitive term searching again and again.
3) Sometimes, when we search like free music, we can be trapped to provide a virus which might be harmful for our computer.

By keeping in view, the above three points, the followings and related things must not be searched over the internet:

9.1. Avoiding Ads:

Searching the followings can avoid un-necessary ads from the search engines.

- ✓ Your email
- ✓ Medical issues and drugs
- ✓ Your name
- ✓ Your location
- ✓ Your favorite things

9.2. Dangerous to Search

Searching the followings can alert the law enforcement agencies or any organization who is keeping eye on the security threats. There is one typical such Pressure Cooker bomb story. When someone searched this term and the security agency representatives reached to the particular location from where such searching was performed. Therefore, avoid searching the followings:

- ✓ pressure cooker bomb story
- ✓ Attacks
- ✓ Suicide bomb
- ✓ Killers/Underworld
- ✓ Terrifying insects
- ✓ Killing animals
- ✓ Poisons
- ✓ Murder
- ✓ Medical Symptoms
- ✓ How to make computer Virus
- ✓ Hacking
- ✓ About Religion
- ✓ About Politics

9.3. Avoid Cyber-security attacks

Avoid searching the followings:

- ✓ Free Music

The hackers normally add many viruses to such websites as the information seeker of free music etc.

Copyright Virtual University of Pakistan

9.4. Unpleasant Results

You might search the followings or similar thing and might get unpleasant results which might depress you.

- ✓ Smokers Lungs
- ✓ Skin Condition

Copyright Virtual University of Pakistan

Module 10

10. Roots of Computing

Here we will see how we moved from mechanical driven machines to electronics driven machine (Computer).

10.1. Abacus

One of the earlier computing devices was the abacus. History tells us that it probably had its roots in ancient China and was used in the early Greek and Roman civilizations. The machine is quite simple, consisting of beads strung on rods that are in turn mounted in a rectangular frame as shown in the Figure 11.

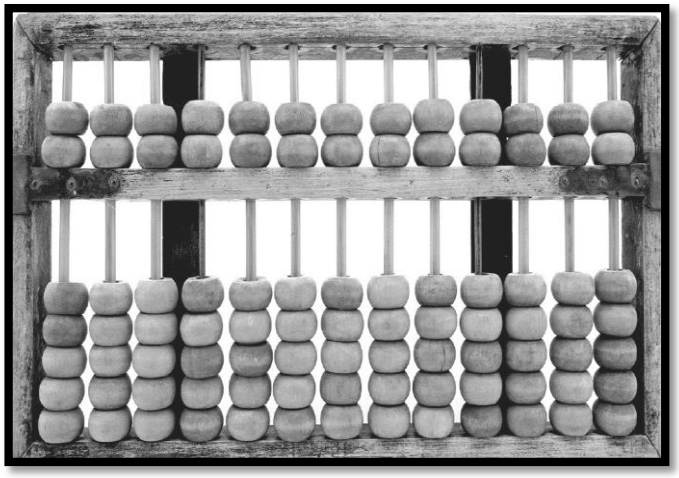


Figure 11: Abacus

10.2. Technology of Gears

In the time period after the Middle Ages and before the Modern Era, the quest for more sophisticated computing machines was seeded. A few inventors began to experiment with the technology of gears. Among these were Blaise Pascal (1623–1662) of France, Gottfried Wilhelm Leibniz (1646–1716) of Germany, and Charles Babbage (1792–1871) of England. These machines represented data through gear positioning, with data being entered mechanically by establishing initial gear positions. Output from Pascal's and Leibniz's machines was achieved by observing the final gear positions. Babbage, on the other hand, envisioned machines that would print results of computations on paper so that the possibility of transcription errors would be eliminated. One typical diagram is shown in the Figure 12.

Copyright Virtual University of Pakistan

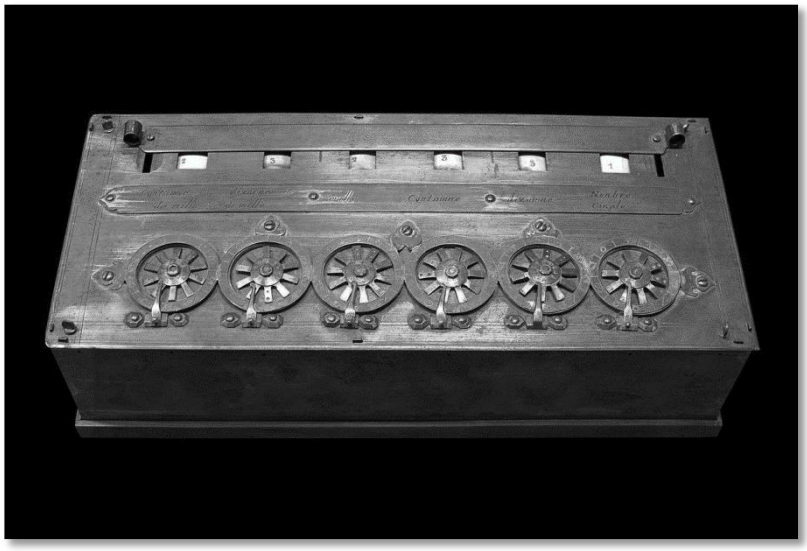


Figure 12: Gear Technology

10.3. Punch Cards

Herman Hollerith (1860–1929), who applied the concept of representing information as holes in paper cards to speed up the tabulation process in the 1890 U.S. census. (It was this work by Hollerith that led to the creation of IBM.) Such cards ultimately came to be known as punched cards and survived as a popular means of communicating with computers well into the 1970s. One typical punch card machine is illustrated in the Figure 13.



Figure 13: Punch Cards

Mechanical-driven to Electronics-driven machines

Nineteenth-century technology was unable to produce the complex gear-driven machines of Pascal, Leibniz, and Babbage cost-effectively. But with the advances in electronics in the early 1900s, this barrier

Copyright Virtual University of Pakistan

was overcome. Examples of this progress include the electromechanical machine of George Stibitz, completed in 1940 at Bell Laboratories, and the Mark I, completed in 1944 at Harvard University by Howard Aiken and a group of IBM engineers. These machines made heavy use of electronically controlled mechanical relays. In this sense they were obsolete almost as soon as they were built, because other researchers were applying the technology of vacuum tubes to construct totally electronic computers.

10.4. ENIAC

More flexible machines, such as the ENIAC (electronic numerical integrator and calculator) developed by John Mauchly and J. Presper Eckert at the Moore School of Electrical Engineering, University of Pennsylvania, soon followed as shown in the Figure 14.

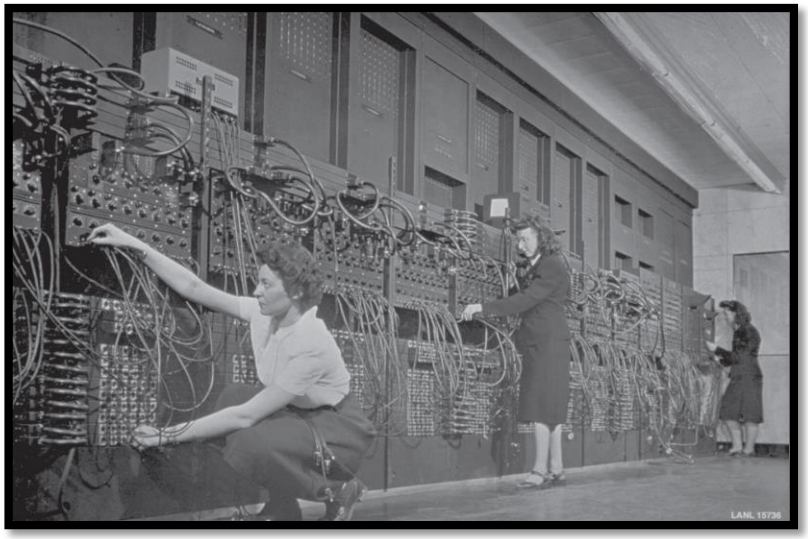


Figure 14: ENIAC

10.5. Factsheet of ENIAC

- ✓ Occupied 1800 square feet
- ✓ 20, 000 vacuum tubes
- ✓ 1500 relays
- ✓ 10, 000 capacitors
- ✓ 70, 000 registers
- ✓ 200 Kilo Watt electricity
- ✓ Weight 30tons
- ✓ Cost = $487, 000
- ✓ PKR = 62.5 millions

Download More Highlighted Handouts From

VUAnswer.com

[https://www.computerhope.com/jargon/e/eniac.htm]

10.6. Rapid Advancement

Since the invention of ENIAC, there was a rapid advancement. Some key milestones are listed below:

- ✓ Transistor
- ✓ Integrated Circuits
- ✓ Size reduction
- ✓ Processing Power doubling in 2 years
- ✓ Desktop computer by Steve Jobs and Stephen Wozniak in 1976
- ✓ IBM launched PC in 1981

Copyright Virtual University of Pakistan

- ✓ Web
- ✓ Smart Phones

Copyright Virtual University of Pakistan

Module 11

11. Bits

In Computer, all kind of information is stored in bits. Bit is the basic unit of storage.

11.1. Basics

- ✓ Information is coded as pattern of 0 or 1
- ✓ Short form of Binary Digits
- ✓ One bit can contain only one value 0 or 1

11.2. What Bits can represent:

- ✓ Representing numbers, text, audio, video, image etc.
- ✓ In Chip electric Charge 0/1

11.3. Bits units

Table 1 shows the bits and their equivalent units.

Table 1: Bits Units

Unit	Equivalent
1 Byte	8 bits
1 Kilo Byte (KB)	1024 Bytes
1 Mega Byte (MB)	1024 KB
1 Giga Byte (GB)	1024 MB
1 Tear Byte (TB)	1024 GB

11.4. Patterns Using Bits

In one bit we can represent two patterns either 0 or 1, if we have 2 bits then 4 different patterns can be representing as shown in the Table 2.

Table 2: bits patterns for two bits

00
01
10
11

Let's comprehend all bit patterns, we will have the Table 3.

Table 3: Bits and their number of patterns

Bits	Number of Patterns
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8 = 1 byte	256

Copyright Virtual University of Pakistan

At page 577 of your book[2], you can find ASCII codes, using which we can represent different characters. For example, representing character "A" we need to represent it using digit 65. The bit pattern of 65 would be:

01000001.

How did we get this, we will learn in details, at the moment you can just try to learn very basic method of converting decimal to binary. As the binary is represented using base 2. To represent any number, please form the representation as shown in the Table 4.

Table 4: representing decimal to Binary

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Now whatever number you want to represent, put 1 the respective bit. For example if you want to represent the decimal 2, just put 1 on the second bit from the right side and put zeros otherwise, you will get the number 2 in binary as shown in the Table 5.

Table 5: Representing Decimal 2 in Binary

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	0	0	0	0	1	0

Similarly, 8 will be represented as: 00001000,

65 will be represented as: 01000001

[2] Computer Science – An Overview by J. Glenn Brookshear and Dennis Brylow

Copyright Virtual University of Pakistan

Module 12

12. Boolean Operations

To understand how individual bits are stored and manipulated inside a computer, it is convenient to imagine that the bit 0 represents the value *false* and the bit 1 represents the value *true*. Operations that manipulate true/false values are called **Boolean operations,** in honor of the mathematician George Boole (1815–1864), who was a pioneer in the field of mathematics called logic. Three of the basic Boolean operations are AND, OR, and XOR (exclusive or). The "AND" operation and "OR" are summarized in the Figure 15.

12.1. AND Boolean Operation

It produces the output as 1 when both of the inputs are 1. Otherwise, it gives output as 0 as shown in the Figure 15.

12.2. OR Boolean Operation

It produces output of 1 when any of the inputs are 1, otherwise 0 as shown in the Figure 15.

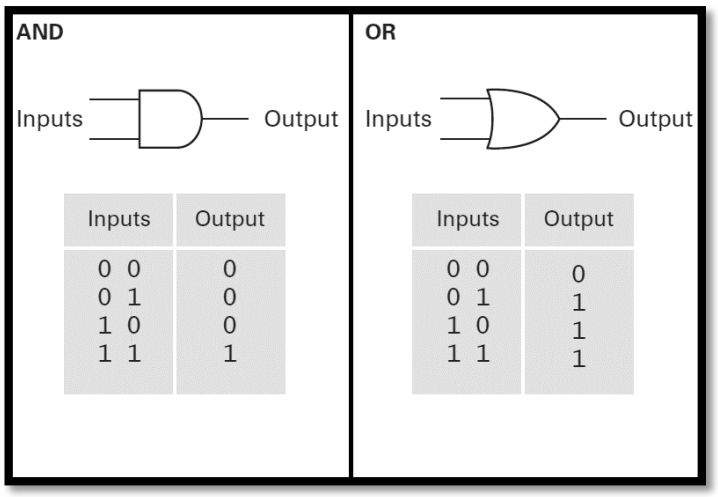


Figure 15: AND, OR Boolean operation

12.3. XOR Boolean Operation

XOR (Exclusive or) is another Boolean operation which produces the output of 1 when both inputs are different. It produces 0 when both inputs are same as shown in the Figure 16.

12.4. Not Operation

Not operation takes one input, produces 1 when input is 0 and produces 0, when the input is 1 as shown in the Figure 16.

Copyright Virtual University of Pakistan

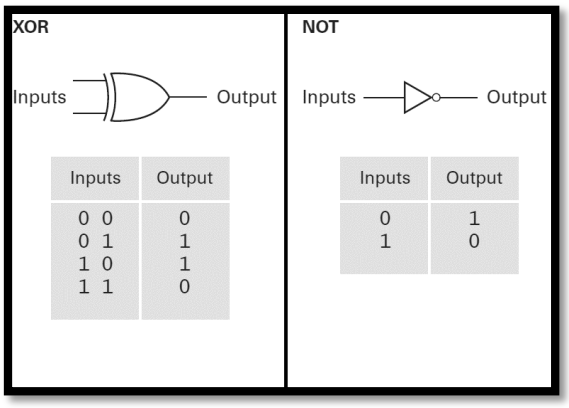


Figure 16: XoR, Not operation

12.5. Boolean Operation Example

To demonstrate the working of the Boolean operations, an example has been shown in the Table 6. First two rows represent the bit patterns as "A" and "B" and next rows are self-explanatory to perform "AND", "OR", "XOR", "Not" on both "A" and "B".

Table 6: Example of Boolean operation

A	0	0	1	1	0	1	1	0
B	1	0	0	1	1	0	0	1
AND	0	0	0	1	0	0	0	0
OR	1	0	1	1	1	1	1	1
XOR	1	0	1	0	1	1	1	1
Not (A)	1	1	0	0	1	0	0	1
Not (B)	0	1	1	0	0	1	1	0

Copyright Virtual University of Pakistan

Module 13

13. Hexadecimal Notation

13.1. Why we need Hexadecimal Notation

When considering the internal activities of a computer, we must deal with patterns of bits, which we will refer to as a string of bits, some of which can be quite long. A long string of bits is often called a **stream.** Unfortunately, streams are difficult for the human mind to comprehend. Merely transcribing the pattern 101101010011 is tedious and error prone. To simplify the representation of such bit patterns, therefore, we usually use a shorthand notation called **hexadecimal notation**, which takes advantage of the fact that bit patterns within a machine tend to have lengths in multiples of four. In particular, hexadecimal notation uses a single symbol to represent a pattern of four bits. For example, a string of twelve bits can be represented by three hexadecimal symbols.

13.2. Hexadecimal Representation

Table 7 presents the hexadecimal notation.

Table 7: Hexadecimal Notation

Bit pattern	Hexadecimal representation
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

13.3. Hexadecimal Example

Let's solve an example from the book, on page 38, Q#5 (b), we want to represent the following bit pattern in hexadecimal.

111010000101010100010111

Our strategy would be to divide it in four bit pattern as shown in the Table 8.

Table 8: hexadecimal example

1110	1000	0101	0101	0001	0111
E	8	5	5	1	7

So the hexadecimal notation for:

111010000101010100010111

Is E85517.

Copyright Virtual University of Pakistan

Module 14

14. Storing a Bit

14.1. Main Memory

For the purpose of storing data, a computer contains a large collection of circuits (such as flip-flops), each capable of storing a single bit. This bit reservoir is known as the machine's **main memory.** A typical Flip-flop is shown in the Figure 17

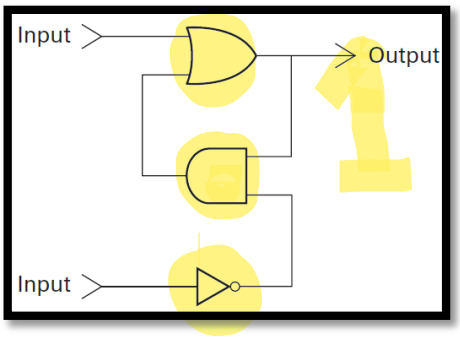


Figure 17: Flip flop

14.2. Main Memory Organization

- ✓ Manageable unit Cells – 8 bits
- ✓ Household devices have few hundreds of cells
- ✓ Large computer may have billions of cells

14.3. Byte Size Organization

In Computer Science, there is no left or right, however, to understand the significance of bits, we represent most significant bits on left side while least significant bits on the right size as shown the Figure 18.

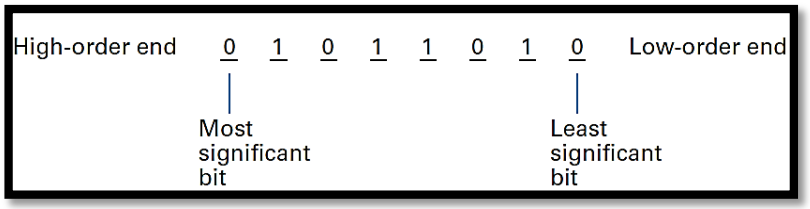


Figure 18: Byte organization in memory

14.4. Memory Address

To identify individual cells in a Computer's main memory, each cell is assigned a unique "name," called its **address.** It has been shown in the Figure 19.

Copyright Virtual University of Pakistan

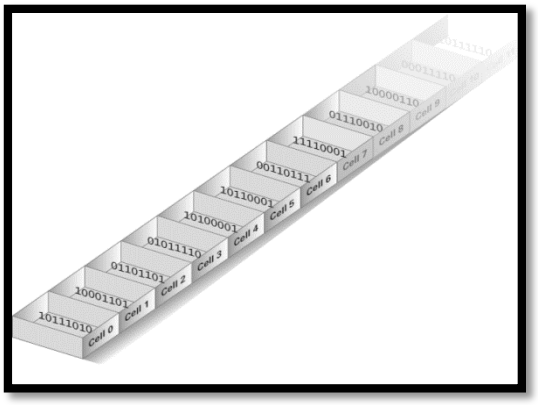


Figure 19: Memory arranged with respect to addresses

14.5. RAM

The cells (represented in Figure 19) can be accessed independently as required. To reflect the ability to access cells in any order, a computer's main memory is often called **random access memory (RAM).**

14.6. DRAM

Dynamic ram – Stores bits as tiny electric Charge, refreshes many times a second.

14.7. SDRAM

Synchronous DRAM is used in reference to DRAM that applies additional techniques to decrease the time needed to retrieve the contents from its memory cells.

Copyright Virtual University of Pakistan

Module 15

15. Magnetic Systems

15.1. Mass Storage

Due to the volatility and limited size of a computer's main memory, most computers have additional memory devices called **mass storage** (or secondary storage) systems, including magnetic disks, CDs, DVDs, magnetic tapes, flash drives, and solid-state disks (all of which we will discuss shortly). The advantages of mass storage systems over main memory include less volatility, large storage capacities, low cost, and in many cases, the ability to remove the storage medium from the machine for archival purposes.

15.2. How it works

One typical device has been shown in the Figure 19.

- ✓ Thin spinning disk with magnetic coating to hold data
- ✓ Read/Write Heads placed above or below disk
- ✓ Each head traverses a circle called track.
- ✓ Normally, each track is divided into equal sectors
- ✓ Sectors have equal number of bits: 512 bytes to few KBs.
- ✓ Outer tracks contain more information.

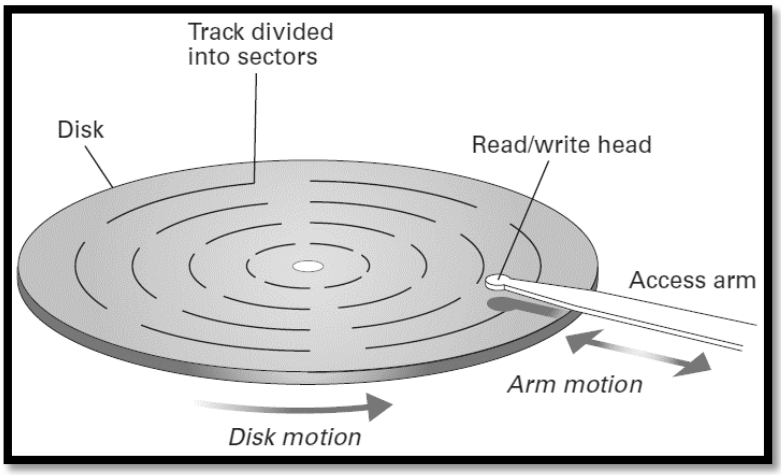


Figure 20: Disk Storage System

15.3. Zoned-bit recording

- ✓ Adjacent tracks form Zones.
- ✓ A typical disk contains 10 zones
- ✓ All tracks within a zone have equal number of sectors.

15.4. Seek Time

Time required to move the read/write heads from one track to another.

15.5. Rotation Delay

Average amount of time required for the desired data to rotate around to the read/write head once the head has been positioned over the desired track.

15.6. Access Time

The sum of seek time and rotation delay

Copyright Virtual University of Pakistan

15.7. Transfer rate

The rate at which data can be transferred to or from the disk.

Download More Highlighted Handouts From

VUAnswer.com

Copyright Virtual University of Pakistan

Module 16

16. Optical System

Another class of mass storage system applies optical technology. An example is the **compact disk (CD).** Digital Versatile Disks (DVDs) and Blu-ray Disks (BDs) are also popular examples of optical systems.

16.1. Compact Disk

Data storage in terms of tracks has been shown in the Figure 21. The characteristics of a CD are:

- ✓ 12 centimeter in diameter
- ✓ Consists of reflective material
- ✓ Information is recorded on them by creating variations in their reflective surfaces.
- ✓ Data is stored on a single-track spiral from inside outside.
- ✓ Each track is divided into sectors,
- ✓ Capacity of a sector is 2KB of data, 1/75 of a second audio music.
- ✓ Retrieval using a Laser that detects the irregularities on the reflective surface of the CD as it spins
- ✓ Initially applied on audio recording using format CD-DA (compact disk-digital audio)
- ✓ Format is being used even today to store computer data
- ✓ Single Track
- ✓ As all sectors are not individually accessible in Optical system having one track, therefore, the data retrieval is faster in magnetic systems than optical systems.
- ✓ Optical System is best suited for long continuous string of data.

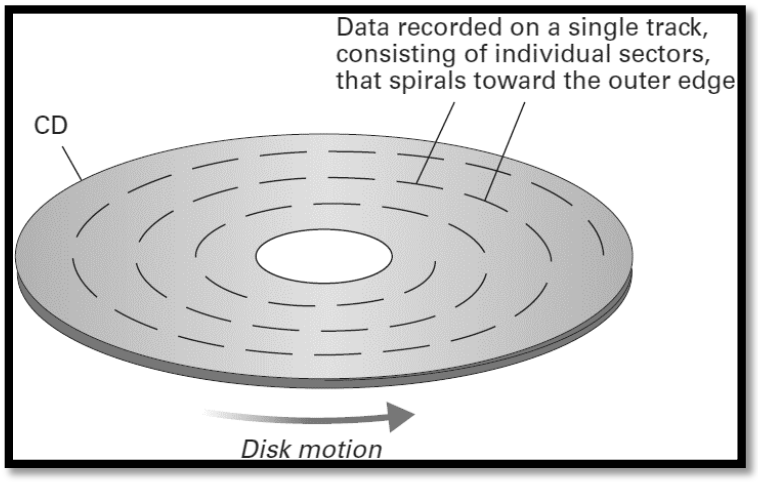


Figure 21: Data storage in CD

16.2. DVD (Digital Versatile Disks)

CDs capacity is 600 to 700 MB; however, DVD has space in GBs as have multiple semi-transparent layers, which can be viewed by precise focused laser. Its shape is like CD.

16.3. BDs (Blue Ray Disks)

BDs use a blue-violet spectrum of light (instead of red), which is able to focus its laser beam with very fine precision. BDs provide five-time capacity than DVDs.

Module 17

17. Flash Drives

17.1. Issues in Magnetic and Optical Systems

- ✓ Physical motion is required

Copyright Virtual University of Pakistan

- ✓ Moving Read/Write heads
- ✓ Aiming Laser beams
- ✓ All this takes lot of time in comparison with electronic circuitry

17.2. Flash Drive Technology

- ✓ Bits are stored by sending electronic signals directly to the storage medium where they cause electrons to be trapped in tiny chambers of silicon dioxide
- ✓ Chambers are able to store data for years without external power.
- ✓ Repeated erasing slowly damages the silicon dioxide chambers
- ✓ Not suitable in place of Main memory
- ✓ Suitable where alterations can be controlled like cameras, smart phones, portable devices
- ✓ Not reliable as optical disks for long term

17.3. SSDs (Solid State Disks)

- ✓ Larger Flash memory devices designed to take place of magnetic disks
- ✓ Quite operations and low access time
- ✓ However, costly than magnetic systems

17.4. SDs (Secure Digital Memory Cards)

- ✓ Provide up to few GBs of storage
- ✓ Available in smaller, mini, and micro sizes
- ✓ SDHC (High Capacity) can provide 32 GBs
- ✓ SDXC (Extendable Capacity) may exceed to TB.
- ✓ Compact physical size, suitable for car navigation, cameras etc.

Download More Highlighted Handouts From

VUAnswer.com

Copyright Virtual University of Pakistan

Module 18

18. Representing Text

This module highlights that how a text is stored with the computer memory.

18.1. Representation as Code

- ✓ Each Textual Symbol is represented with a unique bit pattern
- ✓ Normally 8 bits for each character
- ✓ "Virtual University"
- ✓ 18 characters = 18*8 (144) bits or 18 bytes
- ✓ In 1940's and 1950's many such codes were designed
- ✓ American National Standards Institute (ANSI)
- ✓ American Standard Code for Information Interchange (ASCII)

18.2. ASCII Codes

- ✓ 7 bits for information and most significant bit has zero
- ✓ 27combinations= 128
- ✓ Uppercase, lower case, punctuation marks, digits 0 to 9, line feed, carriage returns, and tabs
- ✓ Available at page 577 of your book.

18.3. ASCII code example

Suppose we want to represent "Hello", by looking at the ASCII table available in your book, the solution is represented in the Figure 22:

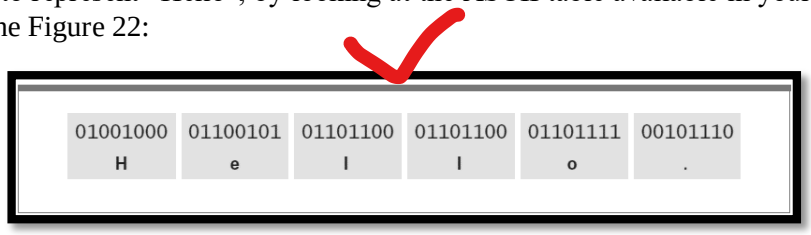


Figure 22: ASCII code example

18.4. Limitation of ASCII codes

- ✓ Only 128 characters
- ✓ International Organization for Standardization (ISO) come-up with many extensions to ASCII
- ✓ One to support western language symbols.

18.5. Limitations of ASCII-extensions

- ✓ 256 are still insufficient to denote all language symbols
- ✓ Document having multiple languages could not be read as it should follow a one standard

18.6. Unicode

- ✓ Internationalization of codes by manufacturers of hardware and software
- ✓ Unique patterns of 21 bits
- ✓ Compliance with ASCII
- ✓ Supporting thousands of character sets of Chinese, Hebrew, Japanese

18.7. UTF-8

- ✓ Uses 24 to 32 bits having large possibilities for expansion
- ✓ 224= 16,777,216 unique symbols
- ✓ File consisting of long symbols encoded with ASCII or Unicode is called text file.

Copyright Virtual University of Pakistan

Module 19

19. Representing Numeric Values

This module explains how the numeric values can be represented.

19.1. Issues in storing numeric as Unicode

- ✓ Inefficient suppose you want to store 12, you would need 16 bits to do that
- ✓ 99 could be stored in 16 bits
- ✓ We will learn 16 bits can store 65,535 numeric values

19.2. Binary Notation

Binary notation is a way of representing numeric values using only the digits 0 and 1 rather than the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Let's represent numeric values using three bits as shown in the Table 9.

Table 9: Three-bit representation of Numeric values

Numeric Value	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

If we add another bit and takes 4 bits then you can observe in the Table 10, we can represent 16 values [ranging from 0 to 15]

Table 10: 4 bits representation of Numeric Values

Numeric	Binary	Numeric	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

If we closely look into the representation, we can represent 2^n numbers if we have n bits. Therefore, if we have 16 bits, we can represent 2^{16} = 65536 numeric values.

19.3. Binary Notation Variations

Binary notation has two following representations, we will learn them in detail in the upcoming modules.

- ✓ Two's complement for storing whole numbers
- ✓ Floating point notation for fractional numbers

Copyright Virtual University of Pakistan

Module 20

20. Representing Images

In this module, we will learn how images are stored in the memory.

Type text here

20.1. Pixel

- ✓ Collection of dots – Pixel short for Picture Element.
- ✓ Appearance of each pixel is encoded to form bit map.
- ✓ Many display devices, printers work on Pixel concept.

20.2. Encoding Method: Pixel to Bitmap

- ✓ In black and white images, each pixel is represented as one bit – e.g. 0 for black and 1 for white
- ✓ Often used in Facsimile

20.3. Encoding Method: Handling shades

- ✓ 8 bits are used instead of 1 bit to store the shades of grayness.

20.4. Encoding Method: Colorful Images

- ✓ RGB encoding
- ✓ One byte for Red, one byte for Green, and one-byte Blue – Three bytes to represent one pixel.

20.5. Brightness Chrominance

- ✓ One brightness component and two-color components.
- ✓ Brightness component= Pixel's luminance (sum of red, green, and blue)
- ✓ Blue Chrominance and Red Chrominance
- ✓ Difference between luminance and amount of blue or red.

20.6. Image Scaling

- ✓ Scaling to a larger size needs more pixels
- ✓ Digital Zoom

20.7. Geometric Structures for Image Scaling

- ✓ Collection of lines and curves
- ✓ Using Analytical Geometry
- ✓ Technique: How geometric structures should be displayed rather Pixel reproduction

20.8. Scalable Fonts

- ✓ TrueType by Microsoft and Apple
- ✓ PostScript by Adobe
- ✓ Also popular in Computer Aided Design (CAD)

Copyright Virtual University of Pakistan

Module 21

21. Representing Sound

21.1. Sound Amplitude

It is the extent to which air particles are displaced, and this **amplitude** of **sound** or **sound amplitude** is experienced as the loudness of **sound**.

21.2. How to encode the Sound

- ✓ Sample the amplitude of sound at regular intervals and record the values as shown in the Figure 23.
- ✓ 8000 samples per second for long distance telephone communication

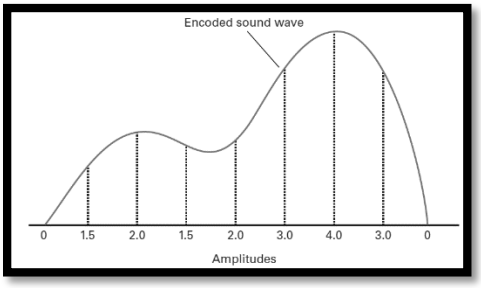


Figure 23: Encoding the amplitude of the sound

21.3. How Sound data communication takes place

- ✓ At one end, it stores amplitude numeric values for each eight thousand of a second
- ✓ Transmitted over the network
- ✓ Received on other end and sound is produced using amplitude.

21.4. Sample Intervals

After how long we should take the sample depends on how accurate and high definition sound recordings are required.

- ✓ 8000 is not enough for high fidelity music recordings
- ✓ 44, 100 samples per second are recorded in today's CD's.
- ✓ Data from each sample is recorded in 16 bits (32 bits for Stereo)
- ✓ 32*44100 = million bits/sec

21.5. Alternative Method: MIDI

- ✓ Musical Instrument Digital Interface.
- ✓ Used in music synthesizers found in electronic keyboards.
- ✓ Encode directions for producing music rather than storing music itself.
- ✓ 2 seconds sound can be stored in 3 bytes rather than 2 million bits.
- ✓ Encoding the "Sheet music" read by performer rather than the performance itself.
- ✓ MIDI recordings could be significantly different when performed on different synthesizer.

Copyright Virtual University of Pakistan

Module 22

22. Binary Notation

Quantity associated with each position is twice the quantity associated with the position to its right.

22.1. Power Method

To understand binary notation first recall decimal numbers we use in our daily life, when we say 375. The representation of 375 in decimal system is available in the Table 11.

Table 11: Decimal System representation

Hundreds	Tens	Units
$10^2 = 100$	$10^1 = 10$	$10^0 = 1$
3	7	5

In each of the position we can use any number from 0 to 9. However, in binary system we can use only 0 or 1 at each position and from right side it represents 2^0 and onwards as illustrated in the Table 12. For example, if we want to represent 32, we will just put 1 in that particular location and rest are 0s.

Table 12: Binary System Representation

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	1	0	0	0	0	0

Similarly, if we want to represent 255, we will put 1 in all locations as:
11111111
If you count (128+64+32+16+8+4+2+1 = 255).

22.2. Algorithm for Finding Binary Representation of a Positive Decimal Number

Remember, algorithm was a set of steps to perform a certain task. To convert a positive decimal number, the set of steps are shown in the Figure 24.

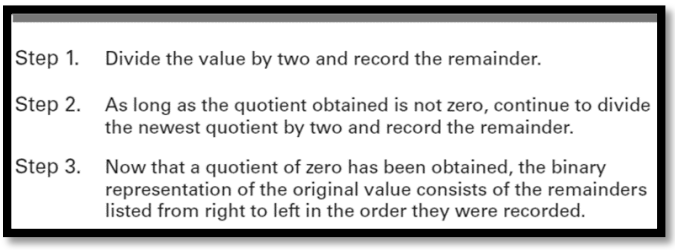


Figure 24: Algorithm for converting positive decimal to binary number

Let's apply this algorithm, suppose we have the number 13 and wants to convert it to the equivalent binary, the working of this algorithm is shown in the Figure 25.

Copyright Virtual University of Pakistan

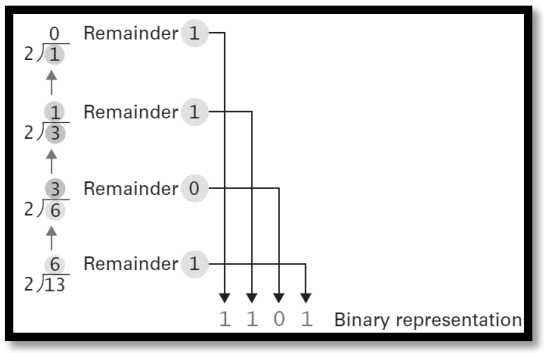


Figure 25: Algorithm working for converting positive decimal to binary

Copyright Virtual University of Pakistan

Module 23

23. Binary Addition

In this module, you will learn the binary addition.

As we are dealing with two bits while adding them, there could be 4 possibilities, firstly whether both bits are zero, secondly first bit is one and second is zero, thirdly first bit is zero and second bit is one, fourthly both are one. All four cases and their sum is illustrated in the Figure 26.

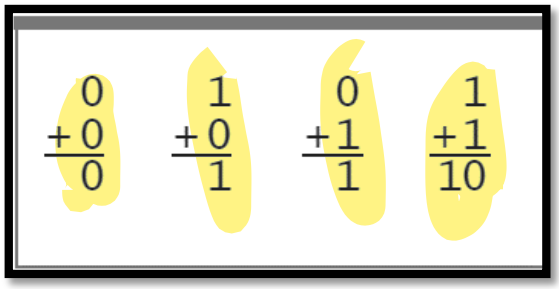


Figure 26: Binary Addition basics

We will apply these rules from right to left as we do the addition of decimal numbers, when we come across 10, we will place 0 and takes 1 as a carry. Different examples have been shown in the Figures 27 to 29.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
0	0	1	1	1	0	1	0	58
0	0	0	1	1	0	1	1	27
0	1	0	1	0	1	0	1	85

Figure 27: Example 1 of Binary Addition

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
0	0	0	0	1	0	1	1	11
0	0	0	0	1	1	1	0	14
0	0	0	1	1	0	0	1	25

Figure 28: Example 2 of Binary Addition

Copyright Virtual University of Pakistan

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
0	1	0	0	0	0	0	0	64
0	1	0	0	0	0	0	0	64
1	0	0	0	0	0	0	0	128

Figure 29: Example 3 of Binary Addition

Copyright Virtual University of Pakistan

Module 24

24. Fraction in Binary

24.1. Radix Point

- ✓ As we have decimal point
- ✓ Digits on left side represent the whole number
- ✓ Digits on right side represent the fractional part

24.2. Example of Radix Point

The example has been shown in the Figure 30.

2^2	2^1	2^0	radix	2^{-1}	2^{-2}	2^{-3}
4	2	1	.	1/2	1/4	1/8
1	0	1		1	0	1

Figure 30: Radix point example

How to read the data represented in the Figure 30, consider the Figure 31.

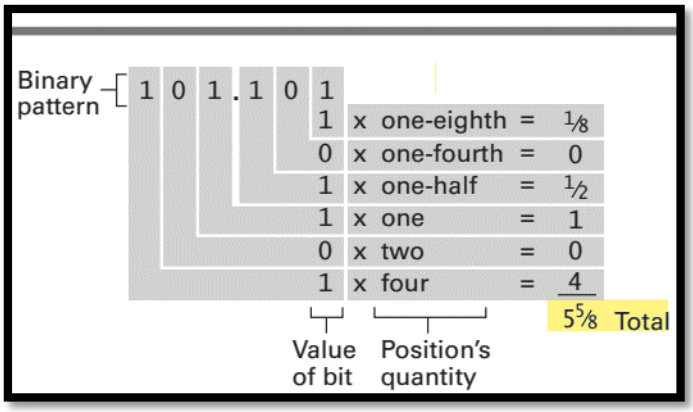


Figure 31: Understanding Fraction represented in Binary

24.3. Addition in fraction

It is same as the addition in decimal system, you need to align the radix point and perform the addition as we did in simple binary or decimal system illustrated in the previous modules. For example, 10.011 added to 100.11 produces 111.001, as shown in the Figure 32.

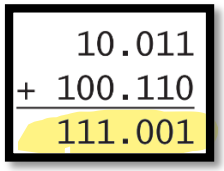


Figure 32: Binary addition

Copyright Virtual University of Pakistan

Module 25

25. 2's Complement Notation to Store Numbers

The most popular system for representing integers within today's computers is **two's complement** notation.

25.1. Integer Representation in 2's Complement

- ✓ Fixed number of bits to represent each value
- ✓ Normally 32 bits are used
- ✓ We will discuss smaller examples for demonstration purpose.
- ✓ For +ve integers, starting from zero going upward until a single zero is reached followed by all 1's
- ✓ For -ve integers, starting from all 1's going downwards until a single 1 is reached followed by all 0's.
- ✓ Left most bit = sign bit.

Integers are represented in the pattern of three lengths in the Figure 33 and patterns of four lengths have been represented in the Figure 34.

a. Using patterns of length three

Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	-1
110	-2
101	-3
100	-4

Figure 33: 2's complement notation for three bits patterns

b. Using patterns of length four

Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

Figure 34: 2's complement notation for four bits patterns

25.2. Conversion between positive and negative representations

One trick to convert between positive and negative numbers represented in 2's complement is as follows:

1. Start from right most bit.
2. Start copying until first 1 arrives
3. After first 1 is found, complement all numbers i.e 0 to 1 and 1 to 0

For example, you know the binary of 7 in 2's complement notation which is 0111, now if you want to find the binary form of -7, apply the above rule and you will get 1001.

25.3. Addition in 2's complement notation

The beauty of 2's complement notation is that it works perfectly fine by using the same method of addition as we studied earlier, for more understanding carefully look at the Figure 35.

Copyright Virtual University of Pakistan

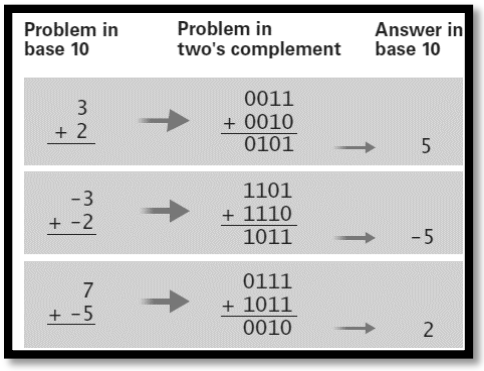


Figure 35: Addition in 2's complement notation

25.4. Problem of Overflow

- ✓ Using 4 bits, maximum positive number 7 can be represented and -8 on the negative side.
- ✓ 5+4 = 9 which cannot be stored in four bits
- ✓ Result would be as -7.
- ✓ Can be deducted using sign bit.
- ✓ Today's computer have longer bit pattern normally 32 bits
- ✓ Maximum positive value = 2,147,483,647
- ✓ If still overflow occurs, we can use even more bits, or changing the units like calculating answer in Kilometer than meter.
- ✓ Previously 16 bits were used to store 2's complement notation, maximum number represented was 32,768
- ✓ On "September 19, 1989", Hospital system malfunctioned as 32, 768 days had been passed after "January 1, 1900", and thus it produced negative value.

Copyright Virtual University of Pakistan

Module 26

26. Excess Notation

This is another method of representing integer values.

26.1. Integer Representations in Excess Notation

- ✓ Fixed number of bits to represent each value
- ✓ Write down all bit patterns of the same length
- ✓ First bit pattern having 1 in the most significant bit is used to represent Zero
- ✓ Following values used to represent positive numbers and Preceding values to represent negative numbers.
- ✓ Each value in Excess notation is Excess of its original value in Binary
- ✓ 1011 represent 11 in Binary but here it is representing 3 (excess of 8).
- ✓ Excess 16 to represent 10000 as Zero
- ✓ Excess 4 to represent 100 as Zero

Excess 8 notation is shown in the Figure 36.

Bit pattern	Value represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	-1
0110	-2
0101	-3
0100	-4
0011	-5
0010	-6
0001	-7
0000	-8

Figure 36: Excess eight notation

26.2. Excess Four Notation

- ✓ Excess 4 to represent 100 as Zero
- ✓ Values are Excess of 4 as shown in the Figure 37.

Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

Figure 37: Excess Four notation

26.3. Excess Four Notation Comparisons

Let's see the differences of bits patterns represented in Binary, Excess 4, and 2's complement. Such comparison is shown in the Figure 38.

Copyright Virtual University of Pakistan

Binary	Decimal	Excess 4	2's Complement
111	7	3	-1
110	6	2	-2
101	5	1	-3
100	4	0	-4
011	3	-1	3
010	2	-2	2
001	1	-3	1
000	0	-4	0

Figure 38: Excess four notation comparisons

Copyright Virtual University of Pakistan

Module 27

27. Floating Point Notation

In contrast to the storage of integers, the storage of a value with a fractional part requires that we store not only the pattern of 0s and 1s representing its binary representation but also the position of the radix point. A popular way of doing this is based on scientific notation and is called floating-point notation.

27.1. Storing Radix

- ✓ Numbers with fractional part have a radix, so its important to store the position of Radix
- ✓ One popular way is floating-point notation
- ✓ For demonstration purpose we will use examples of 8 bits storage system.

27.2. Storing Fractions

We first designate the high-order bit of the byte as the sign bit. Once again, a in the sign bit will mean that the value stored is nonnegative, and a 1 will mean that the value is negative. Next, we divide the remaining 7 bits of the byte into two groups, or fields: the **exponent field** and the **mantissa field**. Let us designate the 3 bits following the sign bit as the exponent field and the remaining 4 bits as the mantissa field. Figure 39 illustrates how the byte is divided. We can explain the meaning of the fields by considering the following example. Suppose a byte consists of the bit pattern 01101011. Analyzing this pattern with the preceding format, we see that the sign bit is 0, the exponent is 110, and the mantissa is 1011. To decode the byte, we first extract the mantissa and place a radix point on its left side, obtaining

.1011

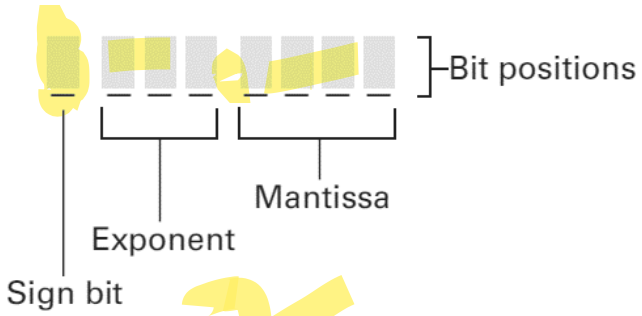


Figure 39: Floating-point notation components

Next, we extract the contents of the exponent field (110) and interpret it as an integer stored using the 3-bit excess method (see again Figure 39). Thus the pattern in the exponent field in our example represents a positive 2. This tells us to move the radix in our solution to the right by 2 bits. (A negative exponent would mean to move the radix to the left.) Consequently, we obtain:

10.11

Which is the binary representation for 2¾. (Recall the representation of binary fractions from Figure 1.18.) Next, we note that the sign bit in our example is 0; the value represented is thus nonnegative. We conclude that the byte 01101011 represents 2¾. Had the pattern been 11101011 (which is the same as before except for the sign bit), the value represented would have been 2¾. As another example, consider the byte 00111100. We extract the mantissa to obtain

.1100

and move the radix 1 bit to the left, since the exponent field (011) represents the value -1. We therefore have

.01100

Copyright Virtual University of Pakistan

Which represents 3/8. Since the sign bit in the original pattern is 0, the value stored is nonnegative. We conclude that the pattern 00111100 represents 3⁄8. To store a value using floating-point notation, we reverse the preceding process. For example, to encode 1 1⁄8, first we express it in binary notation and obtain 1.001. Next, we copy the bit pattern into the mantissa field from left to right, starting with the leftmost 1 in the binary representation. At this point, the byte looks like this:

-	-	-	-	1	0	0	1

We must now fill in the exponent field. To this end, we imagine the contents of the mantissa field with a radix point at its left and determine the number of bits and the direction the radix must be moved to obtain the original binary number. In our example, we see that the radix in .1001 must be moved 1 bit to the right to obtain 1.001. The exponent should therefore be a positive one, so we place 101 (which is positive one in excess four notation as shown in Figure 1.23) in the exponent field. Finally, we fill the sign bit with 0 because the value being stored is nonnegative. The finished byte looks like this:

0 1 0 1 1 0 0 1

There is a subtle point you may have missed when filling in the mantissa field. The rule is to copy the bit pattern appearing in the binary representation from left to right, starting with the leftmost 1. To clarify, consider the process of storing the value 3⁄8, which is .011 in binary notation. In this case the mantissa will be

-	-	-	-	1	1	0	0

It will not be

-	-	-	-	0	1	1	0

This is because we fill in the mantissa field starting with the leftmost 1 that appears in the binary representation. Representations that conform to this rule are said to be in **normalized form**.

Using normalized form eliminates the possibility of multiple representations for the same value. For example, both 00111100 and 01000110 would decode to the value 3⁄8, but only the first pattern is in normalized form. Complying with normalized form also means that the representation for all nonzero values will
have a mantissa that starts with 1. The value zero, however, is a special case; its floating-point representation is a bit pattern of all 0s.

Module 28

28. Truncation Errors in Floating Point Notation

Let us consider the annoying problem that occurs if we try to store the value $2^{5/8}$ with our one-byte floating-point system. We first write $2^{5/8}$ in binary, which gives us 10.101. But when we copy this into the mantissa field, we run out of room, and the rightmost 1 (which represents the last 1⁄8) is lost (Figure 40). If we ignore this
problem for now and continue by filling in the exponent field and the sign bit, we end up with the bit pattern 01101010, which represents $2^{1/2}$ instead of $2^{5/8}$. What has occurred is called a **truncation error,** or **round-off error**—meaning that part of the value being stored is lost because the mantissa field is not large enough.

Copyright Virtual University of Pakistan

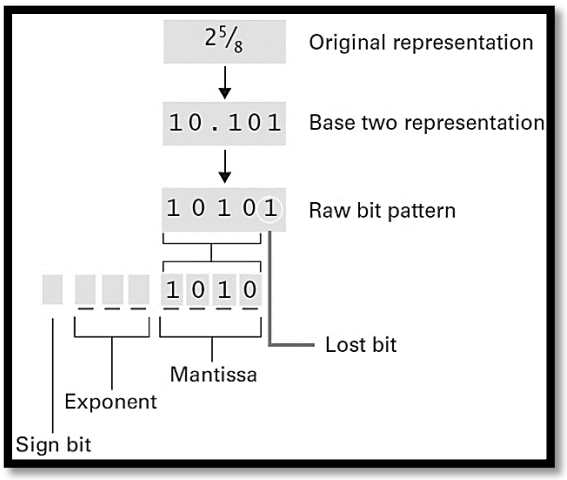


Figure 40: Encoding the value $2^{5/8}$

28.1. Primitive Methods for Handling Truncation Errors

The significance of such errors can be reduced by using a longer mantissa field. In fact, most computers manufactured today use at least 32 bits for storing values in floating-point notation instead of the 8 bits we have used here. This also allows for a longer exponent field at the same time. Even with these longer formats, however, there are still times when more accuracy is required. Another source of truncation errors is a phenomenon that you are already accustomed to in base 10 notation: the problem of nonterminating expansions, such as those found when trying to express 1⁄3 in decimal form. Some values cannot be accurately expressed regardless of how many digits we use. The difference between our traditional base 10 notation and binary notation is that more values have nonterminating representations in binary than in decimal notation. For example, the value 1/10 is nonterminating when expressed in binary. Imagine the problems this might cause the unwary person using floating-point notation to store and manipulate dollars and cents. In particular, if the dollar is used as the unit of measure, the value of a dime could not be stored accurately. A solution in this case is to manipulate the data in units of pennies so that all values are integers that can be accurately stored using a method such as two's complement.

28.2. Intelligent Processing for Handling Truncation Errors

- ✓ Lets suppose we want to store
- ✓ $2^{½}$ + 1/8 + 1/8
- ✓ If we add $2^{½}$ to 1/8 we ends up with $2^{5/8}$ which is 10.101 which cannot be stored in 4 bit mantissa.
- ✓ The result 10.10 would be $2^{½}$ this means the 1/8 is truncated

In this situation, the following can be done:

- ✓ Lets add first 1/8 to 1/8 which is ¼ or .01, can be stored and result would be 00111000
- ✓ Now add this to $2^{½}$ now we got $2^{¾}$ = 01101011 which is accurate.
- ✓ Order is important, adding small quantities together first might give significant quantity to be added to large quantity.

Copyright Virtual University of Pakistan

Module 29

29. Data Compression: Generic Techniques

Data compression schemes fall into two categories. Some are **lossless,** others are **lossy.** Lossless schemes are those that do not lose information in the compression process. Lossy schemes are those that may lead to the loss of information. Lossy techniques often provide more compression than lossless ones and are therefore popular in settings in which minor errors can be tolerated, as in the case of images and audio.

In cases where the data being compressed consist of long sequences of the same value, the compression technique called **run-length encoding,** which is a lossless method, is popular. It is the process of replacing sequences of identical data elements with a code indicating the element that is repeated and the number of times it occurs in the sequence. For example, less space is required to indicate that a bit pattern consists of 253 ones, followed by 118 zeros, followed by 87 ones than to actually list all 458 bits.

Another lossless data compression technique is **frequency-dependent encoding,** a system in which the length of the bit pattern used to represent a data item is inversely related to the frequency of the item's use. Such codes are examples of variable-length codes, meaning that items are represented by patterns of different lengths. David Huffman is credited with discovering an algorithm that is commonly used for developing frequency-dependent codes, and it is common practice to refer to codes developed in this manner as **Huffman codes.** In turn, most frequency-dependent codes in use today are Huffman codes.

As an example of frequency-dependent encoding, consider the task of encoded English language text. In the English language the letters *e, t, a,* and I are used more frequently than the letters *z, q,* and *x*. So, when constructing a code for text in the English language, space can be saved by using short bit patterns to represent the former letters and longer bit patterns to represent the latter ones. The result would be a code in which English text would have shorter representations than would be obtained with uniform-length codes.

In some cases, the stream of data to be compressed consists of units, each of which differs only slightly from the preceding one. An example would be consecutive frames of a motion picture. In these cases, techniques using **relative encoding,** also known as **differential encoding,** are helpful. These techniques record the differences between consecutive data units rather than entire units; that is, each unit is encoded in terms of its relationship to the previous unit. Relative encoding can be implemented in either lossless or lossy form depending on whether the differences between consecutive data units are encoded precisely or approximated.

Still other popular compression systems are based on **dictionary encoding** techniques. Here the term *dictionary* refers to a collection of building blocks from which the message being compressed is constructed, and the message itself is encoded as a sequence of references to the dictionary. We normally think of dictionary encoding systems as lossless systems, but as we will see in our discussion of image compression, there are times when the entries in the dictionary are only approximations of the correct data elements, resulting in a lossy compression system.

Dictionary encoding can be used by word processors to compress text documents because the dictionaries already contained in these processors for the purpose of spell checking make excellent compression dictionaries. In particular, an entire word can be encoded as a single reference to this dictionary rather than as a sequence of individual characters encoded using a system such as UTF-8. A typical dictionary in a word processor contains approximately 25,000 entries, which means an individual entry can be identified by an integer in the range of 0 to 24,999. This means that a particular entry in the dictionary can be identified by a pattern of only 15 bits. In contrast, if the word being referenced consisted of six letters, its character-by-character encoding would require 48 bits using UTF-8.

Copyright Virtual University of Pakistan

A variation of dictionary encoding is **adaptive dictionary encoding** (also known as dynamic dictionary encoding). In an adaptive dictionary encoding system, the dictionary is allowed to change during the encoding process. A popular example is **Lempel-Ziv-Welsh (LZW) encoding** (named after its creators, Abraham Lempel, Jacob Ziv, and Terry Welsh). To encode a message using LZW, one starts with a dictionary containing the basic building blocks from which the message is constructed, but as larger units are found in the message, they are added to the dictionary –meaning that future occurrences of those units can be encoded as single, rather than multiple, dictionary references. For example, when encoding English text, one could start with a dictionary containing individual characters, digits, and punctuation marks. But as words in the message are identified, they could be added to the dictionary. Thus, the dictionary would grow as the message is encoded, and as the dictionary grows, more words (or recurring patterns of words) in the message could be encoded as single references to the dictionary.

The result would be a message encoded in terms of a rather large dictionary that is unique to that particular message. But this large dictionary would not have to be present to decode the message. Only the original small dictionary would be needed. Indeed, the decoding process could begin with the same small dictionary with which the encoding process started. Then, as the decoding process continues, it would encounter the same units found during the encoding process, and thus be able to add them to the dictionary for future reference just as in the encoding process.

To clarify, consider applying LZW encoding to the message

xyx xyx xyx xyx

starting with a dictionary with three entries, the first being *x*, the second being y, and the third being a space. We would begin by encoding *xyx* as 121, meaning that the message starts with the pattern consisting of the first dictionary entry, followed by the second, followed by the first. Then the space is encoded to produce 1213. But, having reached a space, we know that the preceding string of characters forms a word, and so we add the pattern *xyx* to the dictionary as the fourth entry. Continuing in this manner, the entire message would be encoded as 121343434.

If we were now asked to decode this message, starting with the original three entry dictionary, we would begin by decoding the initial string 1213 as *xyx* followed by a space. At this point we would recognize that the string *xyx* forms a word and add it to the dictionary as the fourth entry, just as we did during the encoding process. We would then continue decoding the message by recognizing that the 4 in the message refers to this new fourth entry and decode it as the word *xyx*, producing the pattern

xyx xyx

Continuing in this manner we would ultimately decode the string 121343434 as

xyx xyx xyx xyx

Which is the original message.

Copyright Virtual University of Pakistan

Module 30

30. Data Compression: Compressing Images

Numerous compression schemes have been developed specifically for image representations. One system known as GIF (short for Graphic Interchange Format and pronounced "Giff" by some and "Jiff" by others) is a dictionary encoding system that was developed by CompuServe. It approaches the compression problem by reducing the number of colors that can be assigned to a pixel to only 256. The red-green-blue combination for each of these colors is encoded using three bytes, and these 256 encodings are stored in a table (a dictionary) called the palette. Each pixel in an image can then be represented by a single byte whose value indicates which of the 256 palette entries represents the pixel's color. (Recall that a single byte can contain any one of 256 different bit patterns.) Note that GIF is a lossy compression system when applied to arbitrary images because the colors in the palette may not be identical to the colors in the original image.

GIF can obtain additional compression by extending this simple dictionary system to an adaptive dictionary system using LZW techniques. In particular, as patterns of pixels are encountered during the encoding process, they are added to the dictionary so that future occurrences of these patterns can be encoded more efficiently. Thus, the final dictionary consists of the original palette and a collection of pixel patterns.

One of the colors in a GIF palette is normally assigned the value "transpar- ent," which means that the background is allowed to show through each region assigned that "color." This option, combined with the relative simplicity of the GIF system, makes GIF a logical choice in simple animation applications in which multiple images must move around on a computer screen. On the other hand, its ability to encode only 256 colors renders it unsuitable for applications in which higher precision is required, as in the field of photography.

Another popular compression system for images is **JPEG** (pronounced "JAY-peg"). It is a standard developed by the **Joint Photographic Experts Group** (hence the standard's name) within ISO. JPEG has proved to be an effective stan- dard for compressing color photographs and is widely used in the photography industry, as witnessed by the fact that most digital cameras use JPEG as their default compression technique.

The JPEG standard actually encompasses several methods of image compression, each with its own goals. In those situations that require the utmost in precision, JPEG provides a lossless mode. However, JPEG's lossless mode does not produce high levels of compression when compared to other JPEG options. Moreover, other JPEG options have proven very successful, meaning that JPEG's lossless mode is rarely used. Instead, the option known as JPEG's baseline standard (also known as JPEG's lossy sequential mode) has become the standard of choice in many applications.

Image compression using the JPEG baseline standard requires a sequence of steps, some of which are designed to take advantage of a human eye's limitations. In particular, the human eye is more sensitive to changes in brightness than to changes in color. So, starting from an image that is encoded in terms of luminance and chrominance components, the first step is to average the chrominance values over two-by-two-pixel squares. This reduces the size of the chrominance information by a factor of four while preserving all the original brightness information. The result is a significant degree of compression without a noticeable loss of image quality.

The next step is to divide the image into eight-by-eight-pixel blocks and to compress the information in each block as a unit. This is done by applying a mathematical technique known as the discrete cosine transform, whose details need not concern us here. The important point is that this transformation converts the original eight-by-eight block into another block whose entries reflect how the pixels in the original block relate to each other rather than the actual pixel values. Within this new block, values below a predetermined threshold are then replaced by zeros, reflecting the fact that the changes represented by these values are too subtle to be detected by the human eye. For example, if the original block contained a checkerboard pattern, the new block might reflect a uniform average color. (A typical eight-by-eight-pixel

Copyright Virtual University of Pakistan

block would represent a very small square within the image so the human eye would not identify the checkerboard appearance anyway.)

At this point, more traditional run-length encoding, relative encoding, and variable-length encoding techniques are applied to obtain additional compression. Altogether, JPEG's baseline standard normally compresses color images by a factor of at least 10, and often by as much as 30, without noticeable loss of quality.

Still another data compression system associated with images is **TIFF** (short for **Tagged Image File Format**). However, the most popular use of TIFF is not as a means of data compression but instead as a standardized format for storing photographs along with related information such as date, time, and camera set- tings. In this context, the image itself is normally stored as red, green, and blue pixel components without compression.

The TIFF collection of standards does include data compression techniques, most of which are designed for compressing images of text documents in facsimile applications. These use variations of run-length encoding to take advantage of the fact that text documents consist of long strings of white pixels. The color image compression option included in the TIFF standards is based on techniques similar to those used by GIF and are therefore not widely used in the photography community.

Copyright Virtual University of Pakistan

Module 31

31. Data Compression: Compressing Audio and Videos

The most commonly used standards for encoding and compressing audio and video were developed by the **Motion Picture Experts Group (MPEG)** under the leadership of ISO. In turn, these standards themselves are called MPEG.

MPEG encompasses a variety of standards for different applications. For example, the demands for high definition television (HDTV) broadcast are distinct from those for video conferencing, in which the broadcast signal must find its way over a variety of communication paths that may have limited capabilities. Both of these applications differ from that of storing video in such a manner that sections can be replayed or skipped over.

The techniques employed by MPEG are well beyond the scope of this text, but in general, video compression techniques are based on video being constructed as a sequence of pictures in much the same way that motion pictures are recorded on film. To compress such sequences, only some of the pictures, called I-frames, are encoded in their entirety. The pictures between the I-frames are encoded using relative encoding techniques. That is, rather than encode the entire picture, only its distinctions from the prior image are recorded. The I-frames themselves are usually compressed with techniques similar to JPEG.

The best-known system for compressing audio is **MP3,** which was developed within the MPEG standards. In fact, the acronym MP3 is short for MPEG layer 3. Among other compression techniques, MP3 takes advantage of the properties of the human ear, removing those details that the human ear can- not perceive. One such property, called **temporal masking,** is that for a short period after a loud sound, the human ear cannot detect softer sounds that would otherwise be audible. Another, called **frequency masking,** is that a sound at one frequency tends to mask softer sounds at nearby frequencies. By taking advantage of such characteristics, MP3 can be used to obtain significant compression of audio while maintaining near CD quality sound.

Using MPEG and MP3 compression techniques, video cameras are able to record as much as an hour's worth of video within 128MB of storage, and portable music players can store as many as 400 popular songs in a single GB. But, in contrast to the goals of compression in other settings, the goal of compressing audio and video is not necessarily to save storage space. Just as important is the goal of obtaining encodings that allow information to be transmitted over today's communication systems fast enough to provide timely presentation. If each video frame required a MB of storage and the frames had to be transmitted over a communication path that could relay only one KB per second, there would be no hope of successful video conferencing. Thus, in addition to the quality of reproduction allowed, audio and video compression systems are often judged by the transmission speeds required for timely data communication. These speeds are normally measured in **bits per second (bps).** Common units include **Kbps** (kilo-bps, equal to one thousand bps), **Mbps** (mega-bps, equal to one million bps), and **Gbps** (giga-bps, equal to one billion bps). Using MPEG techniques, video presentations can be successfully relayed over communication paths that provide transfer rates of 40 Mbps. MP3 recordings generally require transfer rates of no more than 64 Kbps.

16/32kbps
32/64kbps
64/128kbps

Copyright Virtual University of Pakistan

Module 32

32. Data Manipulation: CPU Basic

A CPU consists of three parts (Figure 41): the **arithmetic/logic unit,** which contains the circuitry that performs operations on data (such as addition and subtraction); the **control unit,** which contains the circuitry for coordinating the machine's activities; and the **register unit,** which contains data storage cells (similar to main memory cells), called **registers,** that are used for temporary storage of information within the CPU.

Some of the registers within the register unit are considered **general-purpose registers,** whereas others are **special-purpose registers.** For now, we are concerned only with the general purpose registers.

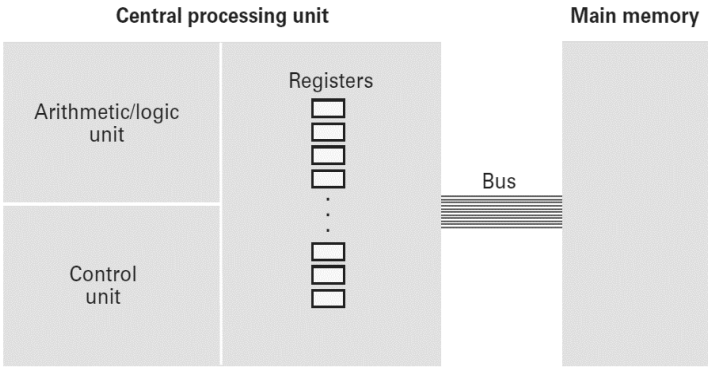


Figure 41: CPU and main memory connected via a bus

General-purpose registers serve as temporary holding places for data being manipulated by the CPU. These registers hold the inputs to the arithmetic/logic unit's circuitry and provide storage space for results produced by that unit. To per- form an operation on data stored in main memory, the control unit transfers the data from memory into the general purpose registers, informs the arithmetic/ logic unit which registers hold the data, activates the appropriate circuitry within the arithmetic/logic unit, and tells the arithmetic/logic unit which register should receive the result.

For the purpose of transferring bit patterns, a machine's CPU and main memory are connected by a collection of wires called a **bus** (see again Figure 41). Through this bus, the CPU extracts (reads) data from main memory by supplying the address of the pertinent memory cell along with an electronic signal telling the memory circuitry that it is supposed to retrieve the data in the indicated cell. In a similar manner, the CPU places (writes) data in memory by providing the address of the destination cell and the data to be stored together with the appropriate electronic signal telling main memory that it is supposed to store the data being sent to it.

Based on this design, the task of adding two values stored in main memory involves more than the mere execution of the addition operation. The data must be transferred from main memory to registers within the CPU, the values must be added with the result being placed in a register, and the result must then be stored in a memory cell. The entire process is summarized by the five steps listed in Figure 42.

Copyright Virtual University of Pakistan

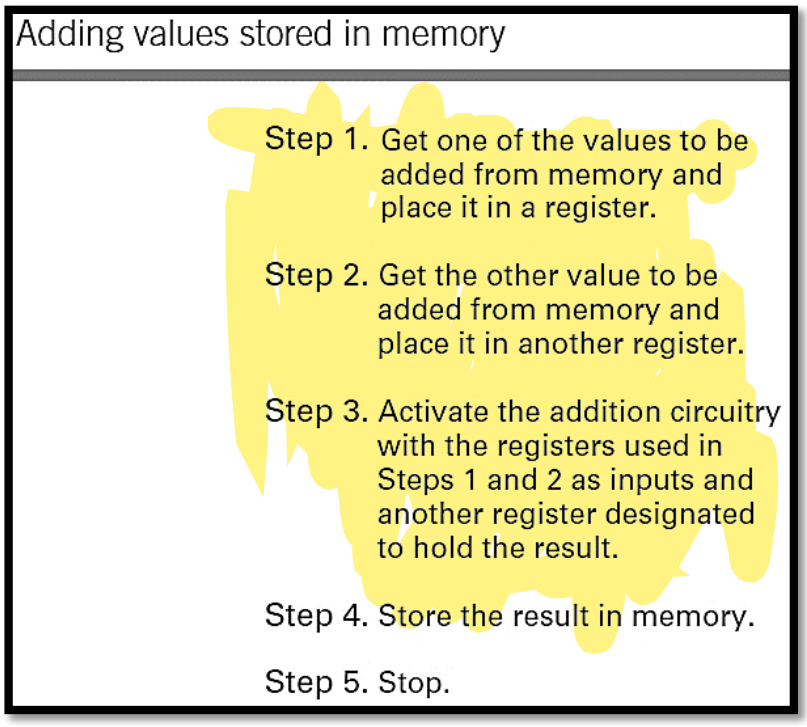


Figure 42: Adding values stored in memory

Copyright Virtual University of Pakistan

Module 33

33. Data Manipulation: Stored Program

Early computers were not known for their flexibility—the steps that each device executed were built into the control unit as a part of the machine. To gain more flexibility, some of the early electronic computers were designed so that the CPU could be conveniently rewired. This flexibility was accomplished by means of a pegboard arrangement similar to old telephone switchboards in which the ends of jumper wires were plugged into holes.

A breakthrough (credited, apparently incorrectly, to John von Neumann) came with the realization that a program, just like data, can be encoded and stored in main memory. If the control unit is designed to extract the program from memory, decode the instructions, and execute them, the program that the machine follows can be changed merely by changing the contents of the computer's memory instead of rewiring the CPU.

The idea of storing a computer's program in its main memory is called the **stored-program concept** and has become the standard approach used today—so standard, in fact, that it seems obvious. What made it difficult originally was that everyone thought of programs and data as different entities: Data were stored in memory; programs were part of the CPU. The result was a prime example of not seeing the forest for the trees. It is easy to be caught in such ruts, and the development of computer science might still be in many of them today without our knowing it. Indeed, part of the excitement of the science is that new insights are constantly opening doors to new theories and applications.

Copyright Virtual University of Pakistan

Module 34

34. Data Manipulation: CPU Architecture Philosophies

To apply the stored program concept, CPUs are designed to recognize instructions encoded as bit patterns. This collection of instructions along with the encoding system is called the machine language. An instruction expressed in this language is called a machine level instruction or, more commonly, a machine instruction.

The list of machine instructions that a typical CPU must be able to decode and execute is quite short. In fact, once a machine can perform certain elementary but well-chosen tasks, adding more features does not increase the machine's theoretical capabilities. In other words, beyond a certain point, additional features may increase such things as convenience but add nothing to the machine's fundamental capabilities.

The degree to which machine designs should take advantage of this fact has led to two philosophies of CPU architecture. One is that a CPU should be designed to execute a minimal set of machine instructions. This approach leads to what is called a reduced instruction set computer (RISC). The argument in favor of RISC architecture is that such a machine is efficient, fast, and less expensive to manufacture. On the other hand, others argue in favor of CPUs with the ability to execute a large number of complex instructions, even though many of them are technically redundant. The result of this approach is known as a complex instruction set computer (CISC). The argument in favor of CISC architecture is that the more complex CPU can better cope with the ever-increasing complexities of today's software. With CISC, programs can exploit a powerful rich set of instructions, many of which would require a multi-instruction sequence in a RISC design. In the 1990s and into the millennium, commercially available CISC and RISC processors were actively competing for dominance in desktop computing. Intel processors, used in PCs, are examples of CISC architecture; PowerPC processors (developed by an alliance between Apple, IBM, and Motorola) are examples of RISC architecture and were used in the Apple Macintosh. As time progressed, the manufacturing cost of CISC was drastically reduced; thus Intel's processors (or their equivalent from AMD—Advanced Micro Devices, Inc.) are now found in virtually all desktop and laptop computers (even Apple is now building computers based on Intel products).

While CISC secured its place in desktop computers, it has an insatiable thirst for electrical power. In contrast, the company Advanced RISC Machine (ARM) has designed a RISC architecture specifically for low power consumption. (Advanced RISC Machine was originally Acorn Computers and is now ARM Holdings.) Thus, ARM based processors, manufactured by a host of vendors including Qualcomm and Texas Instruments, are readily found in game controllers, digital TVs, navigation systems, automotive modules, cellular telephones, smartphones, and other consumer electronics.

Copyright Virtual University of Pakistan

Module 35

35. Data Manipulation: Machine Instruction Categories

Regardless of the choice between RISC and CISC, a machine's instructions can be categorized into three groupings: (1) the data transfer group, (2) the arithmetic/logic group, and (3) the control group.

35.1. Data Transfer Group

The data transfer group consists of instructions that request the movement of data from one location to another. Steps 1, 2, and 4 in Figure 42 fall into this category. We should note that using terms such as transfer or move to identify this group of instructions is actually a misnomer. It is rare that the data being transferred is erased from its original location. The process involved in a transfer instruction is more like copying the data rather than moving it. Thus, terms such as copy, or clone better describe the actions of this group of instructions. While on the subject of terminology, we should mention that special terms are used when referring to the transfer of data between the CPU and main memory. A request to fill a general-purpose register with the contents of a memory cell is commonly referred to as a LOAD instruction; conversely, a request to transfer the contents of a register to a memory cell is called a STORE instruction. In Figure 2.2, Steps 1 and 2 are LOAD instructions, and Step 4 is a STORE instruction.

An important group of instructions within the data transfer category consists of the commands for communicating with devices outside the CPU main memory context (printers, keyboards, display screens, disk drives, etc.). Since these instructions handle the input/output (I/O) activities of the machine, they are called **I/O instructions** and are sometimes considered as a category in their own right.

35.2. Arithmetic/Logic Group

The arithmetic/logic group consists of the instructions that tell the control unit to request an activity within the arithmetic/logic unit. Step 3 in Figure 42 falls into this group. As its name suggests, the arithmetic/logic unit is capable of performing operations other than the basic arithmetic operations.

35.3. Control Group

The control group consists of those instructions that direct the execution of the program rather than the manipulation of data. Step 5 in Figure 42 falls into this category,

Copyright Virtual University of Pakistan

Module 36

36. Data Manipulation: Program Execution

- ✓ Machine Instruction is fetched from main memory to CPU as illustrated in the Figure 43.
- ✓ Each instruction is decoded and obeyed.
- ✓ The order is as the instructions are stored in memory otherwise specified by a JUMP.

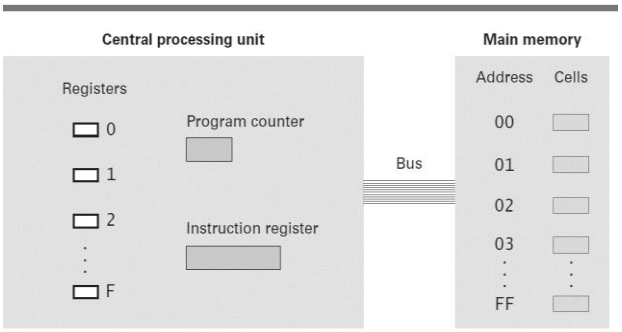


Figure 43: CPU and Main memory linkage using Bus

Special Purpose Registers

- ✓ **Instruction Register:** Holding instruction being executed now.

Program Counter: contains the address of next instruction to be executed.

Machine cycle is shown in the Figure 44.

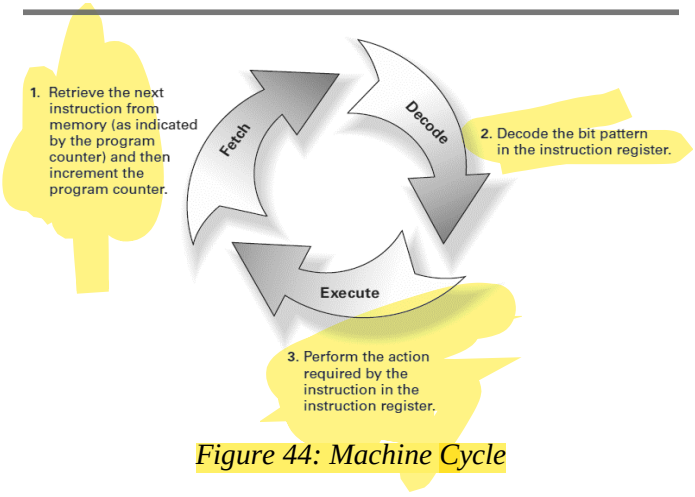


Figure 44: Machine Cycle

Copyright Virtual University of Pakistan

Module 37

37. Data Manipulation: Program Execution Example

In your book an example of machine instructions with their meanings are available in Appendix C at page 581. let's discuss it.

Let's execute a program that reads two numbers from memory, adds them and store in the memory. The instructions for such an activity can be seen in the Figure 45.

Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

Figure 45: Machine Instructions for adding two numbers

When this instruction will be loaded in the memory. The snapshot of the main memory is shown in the Figure 46.

Copyright Virtual University of Pakistan

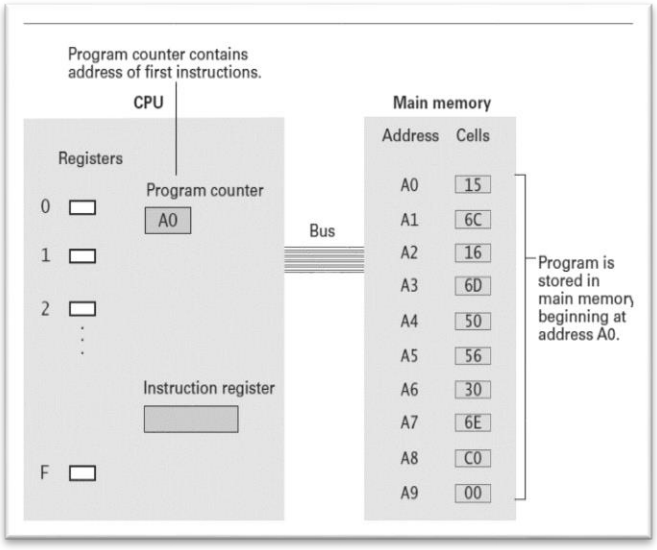


Figure 46: Machine instructions loaded in the main memory

The CPU begins the fetch step of the machine cycle by extracting the instruction stored in main memory at location A0 and placing this instruction (156C) in its instruction register. Notice that, in our machine, instructions are 16 bits (two bytes) long. Thus, the entire instruction to be fetched occupies the memory cells at both address A0 and A1. The CPU is designed to take this into account, so it retrieves the contents of both cells and places the bit patterns received in the instruction register, which is 16 bits long. The CPU then adds two to the program counter so that this register contains the address of the next instruction. At the end of the fetch step of the first machine cycle, the program counter and instruction register contain the following data:

Program Counter: A2

Instruction Register: 156C

Next, the CPU analyzes the instruction in its instruction register and concludes that it is to load register 5 with the contents of the memory cell at address 6C. This load activity is performed during the execution step of the machine cycle, and the CPU then begins the next cycle.

This cycle begins by fetching the instruction 166D from the two memory cells starting at address A2. The CPU places this instruction in the instruction register and increments the program counter to A4. The values in the program counter and instruction register therefore become the following:

Program Counter: A4

Instruction Register: 166D

Now the CPU decodes the instruction 166D and determines that it is to load register 6 with the contents of memory address 6D. It then executes the instruction. It is at this time that register 6 is actually loaded.

Since the program counter now contains A4, the CPU extracts the next instruction starting at this address. The result is that 5056 is placed in the instruction register, and the program counter is incremented to A6. The CPU now decodes the contents of its instruction register and executes it by activating the two's complement addition circuitry with inputs being registers 5 and 6.

Copyright Virtual University of Pakistan

During this execution step, the arithmetic/logic unit performs the requested addition, leaves the result in register 0 (as requested by the control unit), and reports to the control unit that it has finished. The CPU then begins another machine cycle. Once again, with the aid of the program counter, it fetches the next instruction (306E) from the two memory cells starting at memory location A6 and increments the program counter to A8. This instruction is then decoded and executed. At this point, the sum is placed in memory location 6E.

The next instruction is fetched starting from memory location A8, and the program counter is incremented to AA. The contents of the instruction register (C000) are now decoded as the halt instruction. Consequently, the machine stops during the execute step of the machine cycle, and the program is completed.

Copyright Virtual University of Pakistan

Module 38

38. Data Manipulation: Logic Operators

Bitwise operations that combine two strings of bits to produce a single output string by applying the basic operation to individual columns. For example, the result of ANDing the patterns 10011010 and 11001001 results in

```
    10011010
AND 11001001
    10001000
```

where we have merely written the result of ANDing the two bits in each column at the bottom of the column. Likewise, ORing and XORing these patterns would produce

```
    10011010
OR  11001001
    11011011
```

```
    10011010
XOR 11001001
    01010011
```

One of the major uses of the AND operation is for placing 0s in one part of a bit pattern while not disturbing the other part. There are many applications for this in practice, such as filtering certain colors out of a digital image represented in the RGB format, as described in the previous chapter. Consider, for example, what happens if the byte 00001111 is the first operand of an AND operation without knowing the contents of the second operand, we still can conclude that the four most significant bits of the result will be 0s. Moreover, the four least significant bits of the result will be a copy of that part of the second operand, as shown in the following example:

```
    00001111
AND 10101010
    00001010
```

This use of the AND operation is an example of the process called **masking.** Here one operand, called a **mask,** determines which part of the other operand will affect the result. In the case of the AND operation, masking produces a result that is a partial replica of one of the operands, with 0s occupying the non-duplicated positions. One trivial use of the AND operation in this context would be to mask off all of the bits associated with the red component of the pixels in an image, leaving only the blue and green components. This transformation is frequently available as an option in image manipulation software.

AND operations are useful when manipulating other types of **bit map**

besides images, whenever a string of bits is used in which each bit represents the presence or absence of a particular object. As a non-graphical example, a string of 52 bits, in which each bit is associated with a particular playing card, can be used to represent a poker hand by assigning 1s to those five bits associated with the cards in the hand and 0s to all the others. Likewise, a bit map of 52 bits, of which thirteen are 1s, can be used to represent a hand of bridge, or a bit map of 32 bits can be used to represent which of thirty-two ice cream flavors are available.

Suppose, then, that the eight bits in a memory cell are being used as a bit map, and we want to find out whether the object associated with the third bit from the high-order end is present. We merely need to AND the entire byte with the mask 00100000, which produces a byte of all 0s if and only if the third bit from the high-order end of the bit map is itself 0. A program can then act accordingly by following the AND operation with a conditional branch instruction. Moreover, if the third bit from the high-order end of the bit

Copyright Virtual University of Pakistan

map is a 1, and we want to change it to a 0 without disturbing the other bits, we can AND the bit map with the mask 11011111 and then store the result in place of the original bit map.

Where the AND operation can be used to duplicate a part of a bit string while placing 0s in the non-duplicated part, the OR operation can be used to duplicate a part of a string while putting 1s in the non-duplicated part. For this we again use a mask, but this time we indicate the bit positions to be duplicated with 0s and use 1s to indicate the non-duplicated positions. For example, ORing any byte with 11110000 produces a result with 1s in its most significant four bits while its remaining bits are a copy of the least significant four bits of the other operand, as demonstrated by the following example:

```
   11110000
OR 10101010
   11111010
```

Consequently, whereas the mask 11011111 can be used with the AND operation to force a 0 in the third bit from the high-order end of a byte, the mask 00100000 can be used with the OR operation to force a 1 in that position.

A major use of the XOR operation is in forming the complement of a bit string. XORing any byte with a mask of all 1s produces the complement of the byte. For example, note the relationship between the second operand and the result in the following example:

```
    11111111
XOR 10101010
    01010101
```

The XOR operation can be used to invert all of the bits of an RGB bitmap image, resulting in an “inverted” color image in which light colors have been replaced by dark colors, and vice versa.

Copyright Virtual University of Pakistan

Module 39

39. Data Manipulation: Rotation and Shift

The operations in the class of rotation and shift operations provide a means for moving bits within a register and are often used in solving alignment problems. These operations are classified by the direction of motion (right or left) and whether the process is circular. Within these classification guidelines are numerous variations with mixed terminology. Let us take a quick look at the ideas involved.

Consider a register containing a byte of bits. If we shift its contents one bit to the right, we imagine the rightmost bit falling off the edge and a hole appearing at the leftmost end. What happens with this extra bit and the hole is the distinguishing feature among the various shift operations. One technique is to place the bit that fell off the right end in the hole at the left end. The result is a **circular shift,** also called a **rotation.** Thus, if we perform a right circular shift on a byte size bit pattern eight times, we obtain the same bit pattern we started with.

Another technique is to discard the bit that falls off the edge and always fill the hole with a 0. The term **logical shift** is often used to refer to these operations. Such shifts to the left can be used for multiplying two's complement representations by two. After all, shifting binary digits to the left corresponds to multiplication by two, just as a similar shift of decimal digits corresponds to multiplication by ten. Moreover, division by two can be accomplished by shifting the binary string to the right. In either shift, care must be taken to preserve the sign bit when using certain notational systems. Thus, we often find right shifts that always fill the hole (which occurs at the sign bit position) with its original value. Shifts that leave the sign bit unchanged are sometimes called **arithmetic shifts.**

Copyright Virtual University of Pakistan

Module 40

40. Data Manipulation: Arithmetic Operators

Subtraction can be simulated by addition and negation like in 2's complement notation 7 -5 would be 7 + (-5) which means the binary of 7 will be added to binary of -5 as demonstrated in the Figure 47.

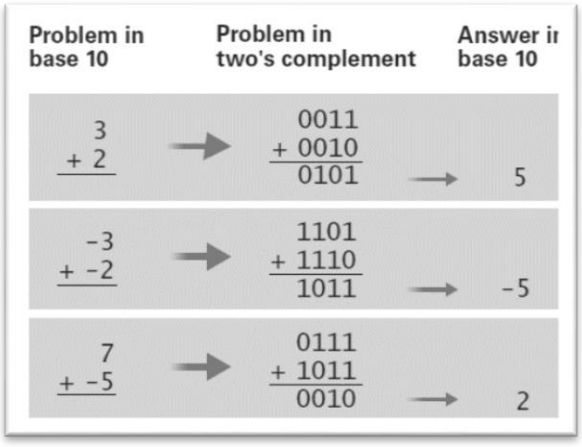


Figure 47: Arithmetic Operations Examples

Multiplication

- ✓ Multiplication is repetitive addition
- ✓ For example, 8 multiplied by 3, we will add 8 three times

Division

- ✓ Can be achieved through subtraction
- ✓ Some small CPUs are designed to have just add, or just add and subtract t do all arithmetic operations.

Remember

- ✓ If its stores as 2's complement, it is straight forward
- ✓ If it save in floating point notation, then you know you need to use mantissa, exponent and sign bit.

Although both are additions, but both have different workout.

Copyright Virtual University of Pakistan

Module 41

41. Data Manipulation: Role of Controller

Communication between a computer and other devices is normally handled through an intermediary apparatus known as a **controller.** In the case of a personal computer, a controller may consist of circuitry permanently mounted on the computer's motherboard or, for flexibility, it may take the form of a circuit board that plugs into a slot on the motherboard. In either case, the controller connects via cables to peripheral devices within the computer case or perhaps to a connector, called a **port,** on the back of the computer where external devices can be attached. These controllers are sometimes small computers themselves, each with its own memory circuitry and simple CPU that performs a program directing the activities of the controller.

A controller translates messages and data back and forth between forms compatible with the internal characteristics of the computer and those of the peripheral device to which it is attached. Originally, each controller was designed for a particular type of device; thus, purchasing a new peripheral device often required the purchase of a new controller as well.

Recently, steps have been taken within the personal computer arena to develop standards, such as the **universal serial bus (USB)** and **FireWire,** by which a single controller is able to handle a variety of devices. For example, a single USB controller can be used as the interface between a computer and any collection of USB-compatible devices. The list of devices on the market today that can communicate with a USB controller includes mice, printers, scanners, mass storage devices, digital cameras, and smartphones.

Each controller communicates with the computer itself by means of connections to the same bus that connects the computer's CPU and main memory (Figure 48). From this position it is able to monitor the signals being sent between the CPU and main memory as well as to inject its own signals onto the bus.

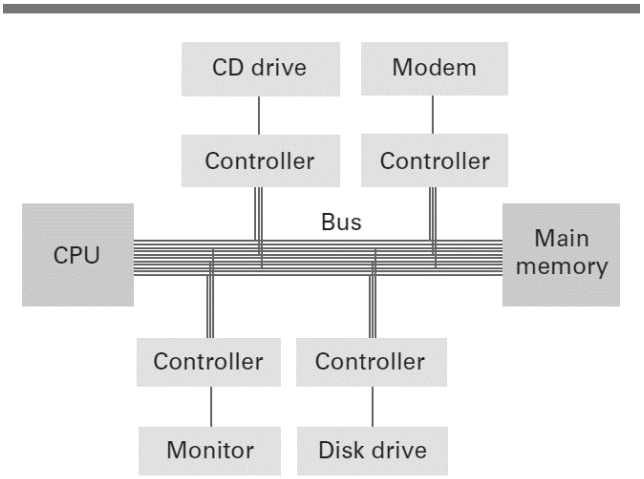


Figure 48: Controllers attached to a machine's bus

With this arrangement, the CPU is able to communicate with the controllers attached to the bus in the same manner that it communicates with main memory. To send a bit pattern to a controller, the bit pattern is first constructed in one of the CPU's general-purpose registers. Then an instruction similar to a STORE instruction is executed by the CPU to "store" the bit pattern in the controller. Likewise, to receive a bit pattern from a controller, an instruction similar to a LOAD instruction is used.

Copyright Virtual University of Pakistan

Module 42

42. Data Manipulation: Direct Memory Access and Handshaking

Since a controller is attached to a computer's bus, it can carry on its own communication with main memory during those nanoseconds in which the CPU is not using the bus. This ability of a controller to access main memory is known as **direct memory access (DMA),** and it is a significant asset to a computer's performance. For instance, to retrieve data from a sector of a disk, the CPU can send requests encoded as bit patterns to the controller attached to the disk asking the controller to read the sector and place the data in a specified area of main memory. The CPU can then continue with other tasks while the controller performs the read operation and deposits the data in main memory via DMA. Thus, two activities will be performed at the same time. The CPU will be executing a program and the controller will be overseeing the transfer of data between the disk and main memory. In this manner, the computing resources of the CPU are not wasted during the relatively slow data transfer.

The use of DMA also has the detrimental effect of complicating the communication taking place over a computer's bus. Bit patterns must move between the CPU and main memory, between the CPU and each controller, and between each controller and main memory. Coordination of all this activity on the bus is a major design issue. Even with excellent designs, the central bus can become an impediment as the CPU and the controllers compete for bus access. This impediment is known as the **von Neumann bottleneck** because it is a consequence of the underlying **von Neumann architecture** in which a CPU fetches its instructions from memory over a central bus.

Copyright Virtual University of Pakistan

Module 43

43. Data Manipulation: Communication media and communication rates

Communication between computing devices is handled over two types of paths:

- ✓ Parallel communication
- ✓ Serial communication

43.1. Parallel Communication

- ✓ Several signals are transferred at the same time, each on a separate line.
- ✓ Data transfer is good
- ✓ Requires complex architecture

It was illustrated in the Figure 48.

43.2. Serial Communication

- ✓ Transfer data one after the other,
- ✓ Requires a simple data path
- ✓ Data transfer rate is relatively slower than parallel communication.
- ✓ USB and FireWire are examples of high-speed data transfer over a short distance.
- ✓ Ethernet connections for slightly longer distances.

It has been shown in the Figure 49.



Figure 49: Serial communication

- ✓ Traditional voice lines dominated the PC arena for many years.
- ✓ Converting bit patterns to audible tones using Modem (Modulator-demodulator)
- ✓ For faster long-distance communication, DSL (Digital Subscriber Line)
- ✓ **Cable Modems:**
- ✓ Fiber Optics and Coaxial cables for high definition TV and computer network.
- ✓ Rate at which bits are transferred from one computing component to another measured in bits per second **(bps)**
- ✓ **Kbps** (Kilo bits)
- ✓ **Mbps** (Million bits)
- ✓ **Gbps** (Billion bits)
- ✓ 8kbps = 1KB per second.
- ✓ USB and Firewire provide several hundred Mbps.

Copyright Virtual University of Pakistan

Module 44

44. Data Manipulation: Pipelining

Electric pulses travel through a wire no faster than the speed of light. Since light travels approximately 1 foot in a nanosecond (one billionth of a second), it requires at least 2 nanoseconds for the CPU to fetch an instruction from a memory cell that is 1 foot away. (The read request must be sent to memory, requiring at least 1 nanosecond, and the instruction must be sent back to the CPU, requiring at least another nanosecond.) Consequently, to fetch and execute an instruction in such a machine requires several nanoseconds—which means that increasing the execution speed of a machine ultimately becomes a miniaturization problem. However, increasing execution speed is not the only way to improve a computer's performance. The real goal is to improve the machine's **throughput,** which refers to the total amount of work the machine can accomplish in a given amount of time.

An example of how a computer's throughput can be increased without requiring an increase in execution speed involves **pipelining,** which is the technique of allowing the steps in the machine cycle to overlap. In particular, while one instruction is being executed, the next instruction can be fetched, which means that more than one instruction can be in "the pipe" at any one time, each at a different stage of being processed. In turn, the total throughput of the machine is increased even though the time required to fetch and execute each individual instruction remains the same. (Of course, when a JUMP instruction is reached, any gain that would have been obtained by prefetching is not realized because the instructions in "the pipe" are not the ones needed after all.)

Modern machine designs push the pipelining concept beyond our simple example. They are often capable of fetching several instructions at the same time and actually executing more than one instruction at a time when those instructions do not rely on each other.

Copyright Virtual University of Pakistan

Module 45

45. Operating Systems: History

Today's operating systems are large, complex software packages that have grown from humble beginnings. The computers of the 1940s and 1950s were not very flexible or efficient. Machines occupied entire rooms. Program execution required significant preparation of equipment in terms of mounting magnetic tapes, placing punched cards in card readers, setting switches, and so on. The execution of each program, called a **job,** was handled as an isolated activity—the machine was prepared for executing the program, the program was executed, and then all the tapes, punched cards, etc. had to be retrieved before the next program preparation could begin. When several users needed to share a machine, sign-up sheets were provided so that users could reserve the machine for blocks of time. During the time period allocated to a user, the machine was totally under that user's control. The session usually began with program setup, followed by short periods of program execution. It was often completed in a hurried effort to do just one more thing ("It will only take a minute") while the next user was impatiently starting to set up.

In such an environment, operating systems began as systems for simplifying program setup and for streamlining the transition between jobs. One early development was the separation of users and equipment, which eliminated the physical transition of people in and out of the computer room. For this purpose, a computer operator was hired to operate the machine. Anyone wanting a program run was required to submit it, along with any required data and special directions about the program's requirements, to the operator and return later for the results. The operator, in turn, loaded these materials into the machine's mass storage where a program called the operating system could read and execute them one at a time. This was the beginning of **batch processing**—the execution of jobs by collecting them in a single batch, then executing them without further interaction with the user.

In batch processing systems, the jobs residing in mass storage wait for execution in a **job queue** (Figure 50). A **queue** is a storage organization in which objects (in this case, jobs) are ordered in **first-in, first-out** (abbreviated FIFO and pronounced "FI-foe") fashion. That is, the objects are removed from the queue in the order in which they arrived. In reality, most job queues do not rigorously follow the FIFO structure, since most operating systems provide for consideration of job priorities. As a result, a job waiting in the job queue can be bumped by a higher-priority job.

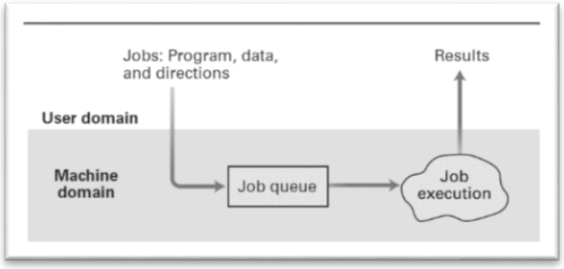


Figure 50: Batch Processing

In early batch-processing systems, each job was accompanied by a set of instructions explaining the steps required to prepare the machine for that particular job. These instructions were encoded, using a system known as a job control language (JCL), and stored with the job in the job queue. When the job was selected for execution, the operating system printed these instructions at a printer where they could be read and followed by the computer operator. This communication between the operating system and the computer operator is still seen today, as witnessed by PC operating systems that report such errors as "net- work not available" and "printer not responding."

Copyright Virtual University of Pakistan

A major drawback to using a computer operator as an intermediary between a computer and its users is that the users have no interaction with their jobs once they are submitted to the operator. This approach is acceptable for some applications, such as payroll processing, in which the data and all processing decisions are established in advance. However, it is not acceptable when the user must interact with a program during its execution. Examples include reservation systems in which reservations and cancellations must be reported as they occur; word processing systems in which documents are developed in a dynamic write and rewrite manner; and computer games in which interaction with the machine is the central feature of the game.

To accommodate these needs, new operating systems were developed that allowed a program being executed to carry on a dialogue with the user through remote terminals—a feature known as **interactive processing** (Figure 51). (A terminal consisted of little more than an electronic typewriter by which the user could type input and read the computer's response that was printed on paper. Today terminals have evolved into more sophisticated devices called workstations and even into complete PCs that can function as stand-alone computers when desired.)

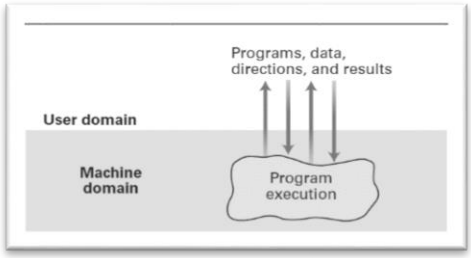


Figure 51: Interactive Processing

Copyright Virtual University of Pakistan

Module 46

46. Operating Systems: Basic Concepts (I)

Coordination with User

- ✓ Successfully interactive processing is to respond the users with sufficiently fast time.
- ✓ Printing record of all students VU versus word processing (typing characters)
- ✓ Execution of tasks under a deadline

Real-time Processing

- ✓ Computer performs the tasks in accordance with the deadlines in its external real-world environment.
- ✓ Example of Cruise Missile, Radar etc

Interactive system and Real-time Processing

- ✓ If a system is servicing only one user, real-time processing was relatively easier to implement, however, computers in 60s, 70s were expensive,
- ✓ Each machine had to serve more than one user at remote terminals.

Time Sharing

- ✓ Based on this problem, such OS were designed to service multiple users at the same time called time-sharing. Active, Ready, waiting states as illustrated in the Figure 52.

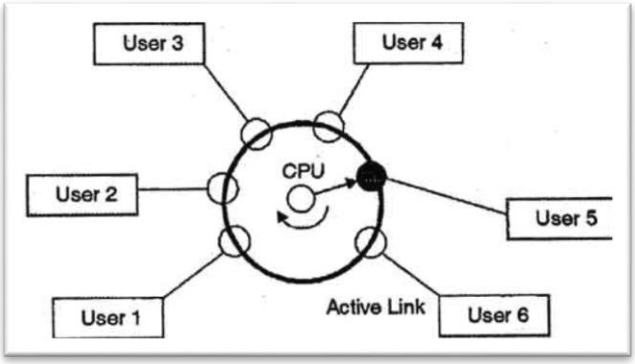


Figure 52: Time Sharing

Multi-programming

- ✓ One way of implementing time-sharing, small time intervals, and each job is executed for such a small interval.
- ✓ Early systems were able to service 30 users simultaneously.

Multitasking

- ✓ One user executing several tasks simultaneously.
- ✓ Today Multiprogramming is used in single user and multiple users.

Copyright Virtual University of Pakistan

Module 47

47. Operating Systems: Basic Concepts (II)

With the development of multiuser, time-sharing operating systems, a typical computer installation was configured as a large central computer connected to numerous workstations. From these workstations, users could communicate directly with the computer from outside the computer room rather than submit- ting requests to a computer operator. Commonly used programs were stored in the machine's mass storage devices, and operating systems were designed to execute these programs as requested from the workstations. In turn, the role of a computer operator as an intermediary between the users and the computer begins to fade.

Today, the existence of a computer operator has essentially disappeared, especially in the arena of personal computers where the computer user assumes all of the responsibilities of computer operation. Even most large computer installations run essentially unattended. Indeed, the job of computer operator has given way to that of a system administrator who manages the computer system—obtaining and overseeing the installation of new equipment and soft- ware, enforcing local regulations such as the issuing of new accounts and establishing mass storage space limits for the various users, and coordinating efforts to resolve problems that arise in the system—rather than operating the machines in a hands-on manner.

In short, operating systems have grown from simple programs that retrieved and executed programs one at a time into complex systems that coordinate time- sharing, maintain programs and data files in the machine's mass storage devices, and respond directly to requests from the computer's users.

But the evolution of operating systems continues. The development of multiprocessor machines has led to operating systems that provide time-sharing/ multitasking capabilities by assigning different tasks to different processors as well as by sharing the time of each single processor. These operating systems must wrestle with such problems as **load balancing** (dynamically allocating tasks to the various processors so that all processors are used efficiently) as well as **scaling** (breaking tasks into a number of subtasks compatible with the number of processors available).

Still another direction of research in operating systems focuses on devices that are dedicated to specific tasks such as medical devices, vehicle electronics, home appliances, cell phones, or other hand-held computers. The computer systems found in these devices are known as **embedded systems.** Embedded operating systems are often expected to conserve battery power, meet demanding real-time deadlines, or operate continuously with little or no human oversight. Successes in this endeavor are marked by systems such as VxWORKS, developed by Wind River Systems and used in the Mars Exploration Rovers named Spirit and Opportunity; Windows CE (also known as Pocket PC) developed by Microsoft; and Palm OS developed by PalmSource, Inc., especially for use in hand-held devices.

Copyright Virtual University of Pakistan

Module 48

48. Operating Systems: Software Classification

Let us begin by dividing a machine's software into two broad categories: **application software** and **system software** (Figure 53). Application software consists of the programs for performing tasks particular to the machine's utilization. A machine used to maintain the inventory for a manufacturing company will contain different application software from that found on a machine used by an electrical engineer. Examples of application software include spreadsheets, database systems, desktop publishing systems, accounting systems, program development software, and games.

In contrast to application software, system software performs those tasks that are common to computer systems in general. In a sense, the system software provides the infrastructure that the application software requires, in much the same manner as a nation's infrastructure (government, roads, utilities, financial institutions, etc.) provides the foundation on which its citizens rely for their individual lifestyles.

Within the class of system software are two categories: One is the operating system itself and the other consists of software units collectively known as **utility software.** The majority of an installation's utility software consists of programs for performing activities that are fundamental to computer installations but not included in the operating system. In a sense, utility software consists of software units that extend (or perhaps customize) the capabilities of the operating system. For example, the ability to format a magnetic disk or to copy a file from a magnetic disk to a CD is often not implemented within the operating system itself but instead is provided by means of a utility program. Other instances of utility software include software to compress and decompress data, software for playing multimedia presentations, and software for handling network communication.

Implementing certain activities as utility software allows system software to be customized to the needs of a particular installation more easily than if they were included in the operating system. Indeed, it is common to find companies or individuals who have modified, or added to, the utility software that was originally provided with their machine's operating system.

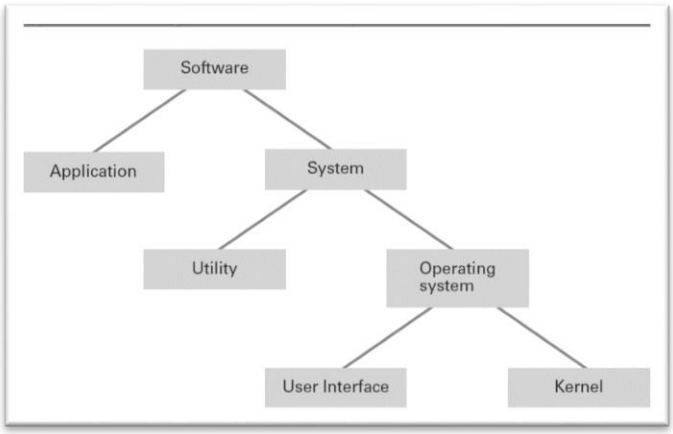


Figure 53: Software Classification

Copyright Virtual University of Pakistan

Module 49

49. Operating Systems: Components (I)

Let us focus now on components that are within the domain of an operating system. In order to perform the actions requested by the computer's users, an operating system must be able to communicate with those users. The portion of an operating system that handles this communication is often called the **user interface.** Older user interfaces, called **shells,** communicated with users through textual messages using a keyboard and monitor screen. More modern systems perform this task by means of a **graphical user interface (GUI**—pronounced "GOO–ee") in which objects to be manipulated, such as files and programs, are represented pictorially on the display as icons. These systems allow users to issue commands by using one of several common input devices. For example, a computer mouse, with one or more buttons, can be used to click or drag icons on the screen. In place of a mouse, special purpose pointing devices or styluses are often used by graphic artists or on several types of handheld devices. More recently, advances in fine-grained touch screens allow users to manipulate icons directly with their fingers. Whereas today's GUIs use two-dimensional image projection systems, three-dimensional interfaces that allow human users to communicate with computers by means of 3D projection systems, tactile sensory devices, and surround sound audio reproduction systems are subjects of current research.

Although an operating system's user interface plays an important role in establishing a machine's functionality, this framework merely acts as an intermediary between the computer's user and the real heart of the operating system (Figure 54). This distinction between the user interface and the internal parts of the operating system is emphasized by the fact that some operating systems allow a user to select among different interfaces to obtain the most comfortable interaction for that particular user. Users of the UNIX operating system, for example, can select among a variety of shells including the Bourne shell, the C shell, and the Korn shell, as well as a GUI called X11. The earliest versions of Microsoft Windows were a GUI application program that could be loaded from the MS-DOS operating system's command shell. The DOS cmd.exe shell can still be found as a utility program in the latest versions of Windows, although this interface is almost never required by casual users. Similarly, Apple's OS X retains a Terminal utility shell that hearkens back to that system's UNIX ancestors.

An important component within today's GUI shells is the **window manager,** which allocates blocks of space on the screen, called windows, and keeps track of which application is associated with each window. When an application wants to display something on the screen, it notifies the window manager, and the window manager places the desired image in the window assigned to the application. In turn, when a mouse button is clicked, it is the window manager that computes the mouse's location on the screen and notifies the appropriate application of the mouse action. Window managers are responsible for what is generally called the "style" of a GUI, and most managers offer a range of configurable choices. Linux users even have a range of choices for a window manager, with popular choices including KDE and Gnome.

Copyright Virtual University of Pakistan

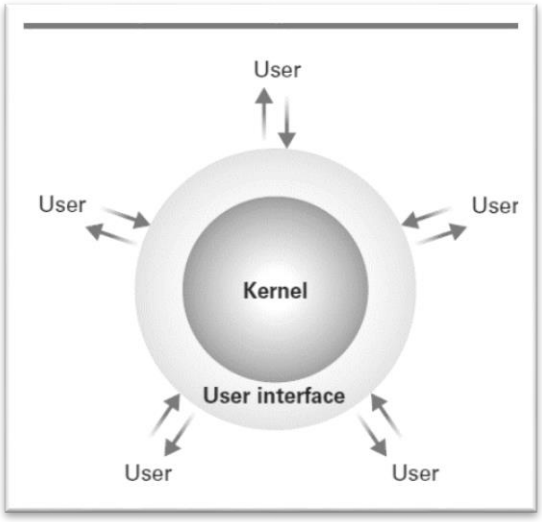


Figure 54: Components of OS

Copyright Virtual University of Pakistan

Module 50

50. Operating Systems: Components (II)

In contrast to an operating system's user interface, the internal part of an operating system is called the **kernel.** An operating system's kernel contains those software components that perform the very basic functions required by the computer installation. One such unit is the **file manager;** whose job is to coordinate the use of the machine's mass storage facilities. More precisely, the file manager maintains records of all the files stored in mass storage, including where each file is located, which users are allowed to access the various files, and which portions of mass storage are available for new files or extensions to existing files. These records are kept on the individual storage medium containing the related files so that each time the medium is placed online, the file manager can retrieve them and thus know what is stored on that particular medium.

For the convenience of the machine's users, most file managers allow files to be grouped into a bundle called a **directory** or **folder.** This approach allows a user to organize his or her files according to their purposes by placing related files in the same directory. Moreover, by allowing directories to contain other directories, called subdirectories, a hierarchical organization can be constructed. For example, a user may create a directory called MyRecords that contains sub-directories called FinancialRecords, MedicalRecords, and HouseHoldRecords. Within each of these subdirectories could be files that fall within that particular category. (Users of a Windows operating system can ask the file manager to display the current collection of folders by executing the utility program Windows Explorer.)

A chain of directories within directories is called a **directory path.** Paths

are often expressed by listing the directories along the path separated by slashes. For instance, animals/prehistoric/dinosaurs would represent the path start- ing at the directory named animals, passing through its subdirectory named prehistoric, and terminating in the sub-subdirectory dinosaurs. (For Windows users the slashes in such a path expression are reversed as in animals\ prehistoric\dinosaurs.)

Any access to a file by other software units is obtained at the discretion of the file manager. The procedure begins by requesting that the file manager grant access to the file through a procedure known as opening the file. If the file man- ager approves the requested access, it provides the information needed to find and to manipulate the file.

Another component of the kernel consists of a collection of **device drivers,** which are the software units that communicate with the controllers (or at times, directly with peripheral devices) to carry out operations on the peripheral devices attached to the machine. Each device driver is uniquely designed for its particular type of device (such as a printer, disk drive, or monitor) and translates generic requests into the more technical steps required by the device assigned to that driver. For example, a device driver for a printer contains the software for reading and decoding that particular printer's status word as well as all the other hand- shaking details. Thus, other software components do not have to deal with those technicalities in order to print a file. Instead, the other components can merely rely on the device driver software to print the file and let the device driver take care of the details. In this manner, the design of the other software units can be independent of the unique characteristics of particular devices. The result is a generic operating system that can be customized for particular peripheral devices by merely installing the appropriate device drivers.

Still another component of an operating system's kernel is the **memory**

manager, which is charged with the task of coordinating the machine's use of main memory. Such duties are minimal in an environment in which a computer is asked to perform only one task at a time. In these cases, the program for performing the current task is placed at a predetermined location in main memory, executed, and then replaced by the program for performing the next task. However, in mul- tiuser or multitasking environments in which the computer is asked to address many needs at the same time, the

Copyright Virtual University of Pakistan

duties of the memory manager are extensive. In these cases, many programs and blocks of data must reside in main memory concurrently. Thus, the memory manager must find and assign memory space for these needs and ensure that the actions of each program are restricted to the pro- gram's allotted space. Moreover, as the needs of different activities come and go, the memory manager must keep track of those memory areas no longer occupied. The task of the memory manager is complicated further when the total main memory space required exceeds the space actually available in the computer. In this case the memory manager may create the illusion of additional memory space by rotating programs and data back and forth between main memory and mass storage (a technique called **paging**). Suppose, for example, that a main memory of 8GB is required but the computer only has 4GB. To create the illusion of the larger memory space, the memory manager reserves 4GB of storage space on a magnetic disk. There it records the bit patterns that would be stored in main memory if main memory had an actual capacity of 8GB. This data is divided into uniform sized units called **pages,** which are typically a few KB in size. Then the memory manager shuffles these pages back and forth between main memory and mass storage so that the pages that are needed at any given time are actually present in the 4GB of main memory. The result is that the computer is able to function as though it actually had 8GB of main memory. This large "fictional" memory space created by paging is called **virtual memory.**

Two additional components within the kernel of an operating system are the **scheduler** and **dispatcher,** which we will study in the next section. For now, we merely note that in a multiprogramming system the scheduler determines which activities are to be considered for execution, and the dispatcher controls the allocation of time to these activities.

Copyright Virtual University of Pakistan

Module 51

51. Operating Systems: Process of Booting

We have seen that an operating system provides the software infrastructure required by other software units, but we have not considered how the operating system gets started. This is accomplished through a procedure known as **boot strapping** (often shortened to **booting**) that is performed by a computer each time it is turned on. It is this procedure that transfers the operating system from mass storage (where it is permanently stored) into main memory (which is essentially empty when the machine is first turned on). To understand the boot strap process and the reason it is necessary, we begin by considering the machine's CPU.

A CPU is designed so that its program counter starts with a particular predetermined address each time the CPU is turned on. It is at this location that the CPU expects to find the beginning of the program to be executed. Conceptually, then, all that is needed is to store the operating system at this location. However, for technical reasons, a computer's main memory is typically constructed from volatile technologies—meaning that the memory loses the data stored in it when the computer is turned off. Thus, the contents of main memory must be replenished each time the computer is restarted.

In short, we need a program (preferably the operating system) to be present in main memory when the computer is first turned on, but the computer's volatile memory is erased each time the machine is turned off. To resolve this dilemma, a small portion of a computer's main memory where the CPU expects to find its initial program is constructed from special nonvolatile memory cells. Such memory is known as **read-only memory (ROM)** because its contents can be read but not altered. As an analogy, you can think of storing bit patterns in ROM as blowing tiny fuses (some blown open—ones—and some blown closed—zeros), although the technology used is more advanced. More precisely, most ROM in today's PCs is constructed with flash memory technology (which means that it is not strictly ROM because it can be altered under special circumstances).

In a general-purpose computer, a program called the **boot loader** is permanently stored in the machine's ROM. This, then, is the program that is initially executed when the machine is turned on. The instructions in the boot loader direct the CPU to transfer the operating system from a predetermined location into the volatile area of main memory (Figure 55). Modern boot loaders can copy an operating system into main memory from a variety of locations. For example, in embedded systems, such as smartphones, the operating system is copied from special flash (nonvolatile) memory; in the case of small workstations at large companies or universities, the operating system may be copied from a distant machine over a network. Once the operating system has been placed in main memory, the boot loader directs the CPU to execute a jump instruction to that area of memory. At this point, the operating system takes over and begins controlling the machine's activities. The overall process of executing the boot loader and thus starting the operating system is called **booting** the computer.

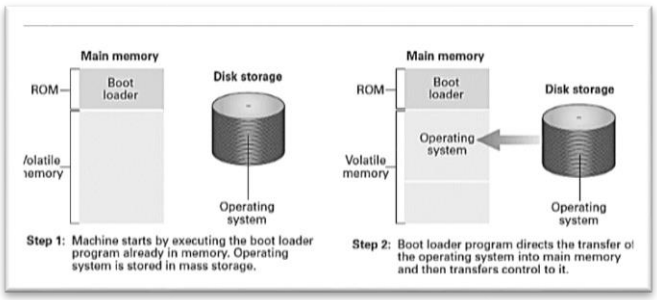


Figure 55: process of Booting

You may ask why desktop computers are not provided with enough ROM to hold the entire operating system so that booting from mass storage would not be necessary. While this is feasible for embedded

Copyright Virtual University of Pakistan

systems with small operating systems, devoting large blocks of main memory in general-purpose computers to nonvolatile storage is not efficient with today's technology. Moreover, computer operating systems undergo frequent updates in order to maintain security and keep abreast of new and improved device drivers for the latest hardware. While it is possible to update operating systems and boot loaders stored in ROM (often called a **firmware update**), the technological limits make mass storage the most common choice for more traditional computer systems

Copyright Virtual University of Pakistan

Module 52

52. Operating Systems: Process and its Administration

Coordinating Machine's activities

- ✓ How OS coordinates the execution of application software, utility software, or OS itself?

Program vs Process

- ✓ Fundamental concepts of modern operating systems is the distinction between a program and the activity of executing a program.
- ✓ Static vs dynamic activity.
- ✓ Analogy of music sheet.

Process

- ✓ The activity of executing a program under the control of the operating system is known as a process.
- ✓ Associated with a process is the current status of the activity, called the process state.

Process state

- ✓ The value of Program counter
- ✓ Values in other CPU registers
- ✓ Values in associated memory cell.
- ✓ Snapshot of the machine at particular time.

Copyright Virtual University of Pakistan

Module 53

53. Operating Systems: Process and its Administration

The tasks associated with coordinating the execution of processes are handled by the scheduler and dispatcher within the operating system's kernel. The scheduler maintains a record of the processes present in the computer system, introduces new processes to this pool, and removes completed processes from the pool. Thus when a user requests the execution of an application, it is the scheduler that adds the execution of that application to the pool of current processes.

To keep track of all the processes, the scheduler maintains a block of information in main memory called the **process table.** Each time the execution of a program is requested, the scheduler creates a new entry for that process in the process table. This entry contains such information as the memory area assigned to the process (obtained from the memory manager), the priority of the process, and whether the process is ready or waiting. A process is **ready** if it is in a state in which its progress can continue; it is **waiting** if its progress is currently delayed until some external event occurs, such as the completion of a mass storage operation, the pressing of a key at the keyboard, or the arrival of a message from another process.

The dispatcher is the component of the kernel that oversees the execution of the scheduled processes. In a time-sharing/multitasking system this task is accomplished by **multiprogramming**; that is, dividing time into short segments, each called a **time slice** (typically measured in milliseconds or microseconds), and then switching the CPU's attention among the processes as each is allowed to execute for one time slice (Figure 56). The procedure of changing from one process to another is called a **process switch** (or a **context switch**).

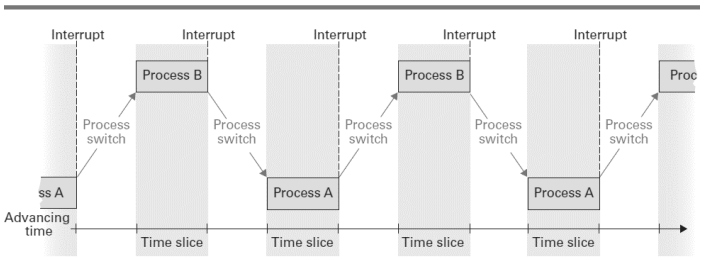


Figure 56: Multiprogramming between process A and process B

Each time the dispatcher awards a time slice to a process, it initiates a timer circuit that will indicate the end of the slice by generating a signal called an **interrupt.** The CPU reacts to this interrupt signal in much the same way that you react when interrupted from a task. You stop what you are doing, record where you are in the task (so that you will be able to return at a later time) and take care of the interrupting entity. When the CPU receives an interrupt signal, it completes its current machine cycle, saves its position in the current process, and begins executing a program, called an **interrupt handler,** which is stored at a predetermined location in main memory. This interrupt handler is a part of the dispatcher, and it describes how the dispatcher should respond to the interrupt signal.

Thus, the effect of the interrupt signal is to preempt the current process and transfer control back to the dispatcher. At this point, the dispatcher selects the process from the process table that has the highest priority among the ready processes (as determined by the scheduler), restarts the timer circuit, and allows the selected process to begin its time slice.

Paramount to the success of a multiprogramming system is the ability to stop, and later restart, a process. If you are interrupted while reading a book, your ability to continue reading at a later time depends on your ability to remember your location in the book as well as the information that you had accumulated to that point. In short, you must be able to re-create the environment that was present immediately prior to the interruption.

Copyright Virtual University of Pakistan

In the case of a process, the environment that must be re-created is the process's state, which as already mentioned, includes the value of the program counter as well as the contents of the registers and pertinent memory cells. CPUs designed for multiprogramming systems incorporate the task of saving this information as part of the CPU's reaction to the interrupt signal. These CPUs also tend to have machine-language instructions for reloading a previously saved state. Such features simplify the task of the dispatcher when performing a process switch and exemplify how the design of modern CPUs is influenced by the needs of today's operating systems.

Copyright Virtual University of Pakistan

Module 54

54. Operating Systems: Handling Competition between Processes

An important task of an operating system is the allocation of the machine's resources to the processes in the system. Here we are using the term *resource* in a broad sense, including the machine's peripheral devices as well as features within the machine itself. The file manager allocates access to files as well and allocates mass storage space for the construction of new files; the memory manager allocates memory space; the scheduler allocates space in the process table; and the dispatcher allocates time slices. As with many problems in computer systems, this allocation task may appear simple at first glance. Below the surface, however, lie several subtleties that can lead to malfunctions in a poorly designed system. Remember, a machine does not think for itself; it merely follows directions. Thus, to construct reliable operating systems, we must develop algorithms that cover every possible contingency, regardless of how minuscule it may appear.

Resources

- ✓ File manager allocates access to files
- ✓ Memory manager allocates the memory space.
- ✓ Scheduler allocates space in the process table
- ✓ Dispatcher allocates timeslots,

Resources Allocation

- ✓ Such resource allocation looks quite simpler, however, it's not so straight forward.
- ✓ Machine does not think about itself, it's You, who can/cannot design reliable OS

Resource Allocation Issues

- ✓ What if two processes demand the same resource at one time.
- ✓ What if 'Process A' is utilizing 'Resource 1' and waiting for 'Resource 2', while 'Process B' is using 'Resource 2' and waiting for 'Resource 1'. It has been shown in the Figure 57.

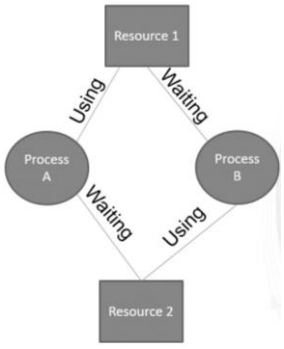


Figure 57: Resource Allocation Issues

Solutions to this problem will be discussed in next modules.

Copyright Virtual University of Pakistan

Module 55

55. Operating Systems: Semaphores

Let us consider a time-sharing/multitasking operating system controlling the activities of a computer with a single printer. If a process needs to print its results, it must request that the operating system give it access to the printer's device driver. At this point, the operating system must decide whether to grant this request, depending on whether the printer is already being used by another process. If it is not, the operating system should grant the request and allow the process to continue; otherwise, the operating system should deny the request and perhaps classify the process as a waiting process until the printer becomes avail- able. After all, if two processes were given simultaneous access to the computer's printer, the results would be worthless to both.

To control access to the printer, the operating system must keep track of whether the printer has been allocated. One approach to this task would be to use a flag, which in this context refers to a bit in memory whose states are often referred to as *set* and *clear*, rather than 1 and 0. A clear flag (value 0) indicates that the printer is available and a set flag (value 1) indicates that the printer is currently allocated. On the surface, this approach seems well-founded. The operating system merely checks the flag each time a request for printer access is made. If it is clear, the request is granted, and the operating system sets the flag. If the flag is set, the operating system makes the requesting process wait. Each time a process finishes with the printer, the operating system either allocates the printer to a waiting process or, if no process is waiting, merely clears the flag.

However, this simple flag system has a problem. The task of testing and possibly setting the flag may require several machine instructions. (The value of the flag must be retrieved from main memory, manipulated within the CPU, and finally stored back in memory.) It is therefore possible for a task to be interrupted after a clear flag has been detected but before the flag has been set. In particular, suppose the printer is currently available, and a process requests use of it. The flag is retrieved from main memory and found to be clear, indicating that the printer is available. However, at this point, the process is interrupted, and another process begins its time slice. It too requests the use of the printer. Again, the flag is retrieved from main memory and found still clear because the previous process was interrupted before the operating system had time to set the flag in main memory. Consequently, the operating system allows the second process to begin using the printer. Later, the original process resumes execution where it left off, which is immediately after the operating system found the flag to be clear. Thus the operating system continues by setting the flag in main memory and granting the original process access to the printer. Two processes are now using the same printer.

The solution to this problem is to insist that the task of testing and possibly setting the flag be completed without interruption. One approach is to use the interrupt disable and interrupt enable instructions provided in most machine languages. When executed, an interrupt disable instruction causes future interrupts to be blocked, whereas an interrupt enable instruction causes the CPU to resume responding to interrupt signals. Thus, if the operating system starts the flag-testing routine with a disable interrupt instruction and ends it with an enable interrupt instruction, no other activity can interrupt the routine once it starts.

Another approach is to use the **test-and-set** instruction that is available in many machine languages. This instruction directs the CPU to retrieve the value of a flag, note the value received, and then set the flag—all within a single machine instruction. The advantage here is that because the CPU always completes an instruction before recognizing an interrupt, the task of testing and setting the flag cannot be split when it is implemented as a single instruction.

A properly implemented flag, as just described, is called a **semaphore,** in reference to the railroad signals used to control access to sections of track. In fact, semaphores are used in software systems in much the same way as they are in railway systems. Corresponding to the section of track that can contain only one train at a time is a sequence of instructions that should be executed by only one process at a time. Such a sequence of instructions is called a **critical region.** The requirement that only one process at a time be

Copyright Virtual University of Pakistan

allowed to execute a critical region is known as **mutual exclusion.** In summary, a common way of obtaining mutual exclusion to a critical region is to guard the critical region with a semaphore. To enter the critical region, a process must find the semaphore clear and then set the semaphore before entering the critical region; then upon exiting the critical region, the process must clear the semaphore. If the semaphore is found in its set state, the process trying to enter the critical region must wait until the semaphore has been cleared.

Copyright Virtual University of Pakistan

Module 56

56. Operating Systems: Deadlock

Another problem that can arise during resource allocation is **deadlock**, the condition in which two or more processes are blocked from progressing because each is waiting for a resource that is allocated to another. For example, one process may have access to the computer's printer but be waiting for access to the computer's CD player, while another process has access to the CD player but is waiting for the printer. Another example occurs in systems in which processes are allowed to cre- ate new processes (an action called **forking** in the UNIX vernacular) to perform subtasks. If the scheduler has no space left in the process table and each process in the system must create an additional process before it can complete its task, then no process can continue. Such conditions, as in other settings (Figure 58), can severely degrade a system's performance Analysis of deadlock has revealed that it cannot occur unless all three of the following conditions are satisfied:

1. There is competition for non-sharable resources.
2. The resources are requested on a partial basis; that is, having received some resources, a process will return later to request more.
3. Once a resource has been allocated, it cannot be forcibly retrieved.

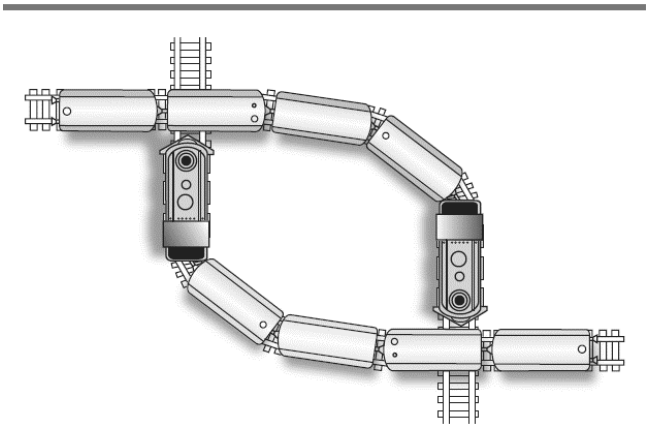


Figure 58: Deadlock Situation

The point of isolating these conditions is that the deadlock problem can be removed by attacking any one of the three. Techniques that attack the third condition fall into the category known as deadlock detection and correction schemes. In these cases, the occurrence of deadlock is considered so remote that no effort is made to avoid the problem. Instead, the approach is to detect it should it occur and then correct it by forcibly retrieving some of the allocated resources. Our example of a full process table might fall in this class. If deadlock should occur due to a full table, routines within the operating system (or perhaps a human administrator using his or her powers as "super user") can remove (the technical term is **killing**) some of the processes. This releases space in the process table, breaking the deadlock and allowing the remaining processes to continue their tasks.

Techniques that attack the first two conditions are known as deadlock avoidance schemes. One, for example, attacks the second condition by requiring each process to request all its resources at one time. Another scheme attacks the first condition, not by removing the competition directly but by converting non-sharable resources into sharable ones. For example, suppose the resource in question is a printer and a variety of processes require its use. Each time a process requests the printer, the operating system could grant the request. However, instead of connecting the process to the printer's device driver, the operating system would connect it to a device driver that stores the information to be printed in mass storage rather than sending it to the printer. Thus each process, thinking it has access to the printer, could execute in its normal way. Later, when the printer is available, the operating system could transfer the data from mass

Copyright Virtual University of Pakistan

storage to the printer. In this manner, the operating system would make the non-sharable resource appear sharable by creating the illusion of more than one printer. This technique of holding data for output at a later but more convenient time is called **spooling.**

Copyright Virtual University of Pakistan

Module 57

57. Operating Systems: Security Attacks from outside

An important task performed by operating systems is to protect the computer's resources from access by unauthorized personnel. In the case of computers used by multiple people, this is usually approached by means of establishing "accounts" for the various authorized users—an account being essentially a record within the operating system containing such entries as the user's name, password, and privileges to be granted to that user. The operating system can then use this information during each **login** procedure (a sequence of transactions in which the user establishes initial contact with a computer's operating system) to control access to the system.

Accounts are established by a person known as the **super user** or the **administrator.** This person gains highly privileged access to the operating system by identifying him- or herself as the administrator (usually by name and password) during the login procedure. Once this contact is established, the administrator can alter settings within the operating system, modify critical software packages, adjust the privileges granted to other users, and perform a variety of other maintenance activities that are denied normal users.

From this "lofty perch," the administrator is also able to monitor activity within the computer system in an effort to detect destructive behavior, whether malicious or accidental. To assist in this regard, numerous software utilities, called **auditing software,** have been developed that record and then analyze the activities taking place within the computer system. In particular, auditing software may expose a flood of attempts to login using incorrect passwords, indicating that an unauthorized user may be trying to gain access to the computer.

Auditing software may also identify activities within a user's account that do not conform to that user's past behavior, which may indicate that an unauthorized user has gained access to that account. (It is unlikely that a user who traditionally uses only word processing and spreadsheet software will suddenly begin to access highly technical software applications or try to execute utility packages that lie outside that user's privileges.)

Another culprit that auditing systems are designed to detect is the presence of **sniffing software**, which is software that, when left running on a computer, records activities and later reports them to a would-be intruder. An old, well- known example is a program that simulates the operating system's login procedure. Such a program can be used to trick authorized users into thinking they are communicating with the operating system, whereas they are actually supplying their names and passwords to an impostor.

With all the technical complexities associated with computer security, it is surprising to many that one of the major obstacles to the security of computer systems is the carelessness of the users themselves. They select passwords that are relatively easy to guess (such as names and dates), they share their passwords with friends, they fail to change their passwords on a timely basis, they subject offline mass storage devices to potential degradation by transferring them back and forth between machines, and they import unapproved software into the sys- tem that might subvert the system's security. For problems like these, most institutions with large computer installations adopt and enforce policies that catalog the requirements and responsibilities of the users.

not imp

Copyright Virtual University of Pakistan

Module 58

58. Operating Systems: Security Attacks from inside

Once an intruder (or perhaps an authorized user with malicious intent) gains access to a computer system, the next step is usually to explore, looking for information of interest or for places to insert destructive software. This is a straight- forward process if the prowler has gained access to the administrator's account, which is why the administrator's password is closely guarded. If, however, access is through a general user's account, it becomes necessary to trick the operating sys- tem into allowing the intruder to reach beyond the privileges granted to that user. For example, the intruder may try to trick the memory manager into allowing a process to access main memory cells outside its allotted area, or the prowler may try to trick the file manager into retrieving files whose access should be denied.

Today's CPUs are enhanced with features that are designed to foil such attempts. As an example, consider the need to restrict a process to the area of main memory assigned to it by the memory manager. Without such restrictions, a process could erase the operating system from main memory and take control of the computer itself. To counter such attempts, CPUs designed for multiprogramming systems typically contain special-purpose registers in which the operating system can store the upper and lower limits of a process's allotted memory area. Then, while performing the process, the CPU compares each memory reference to these registers to ensure that the reference is within the designated limits. If the reference is found to be outside the process's designated area, the CPU automatically transfers control back to the operating system (by performing an interrupt sequence) so that the operating system can take appropriate action.

Embedded in this illustration is a subtle but significant problem. Without further security features, a process could still gain access to memory cells outside of its designated area merely by changing the special-purpose registers that contain its memory limits. That is, a process that wanted access to additional memory could merely increase the value in the register containing the upper memory limit and then proceed to use the additional memory space without approval from the operating system.

To protect against such actions, CPUs for multiprogramming systems are designed to operate in one of two **privilege levels**; we will call one "privileged mode," the other we will call "non-privileged mode." When in privileged mode, the CPU is able to execute all the instructions in its machine language. How- ever, when in non-privileged mode, the list of acceptable instructions is limited. The instructions that are available only in privileged mode are called **privileged instructions.** (Typical examples of privileged instructions include instructions that change the contents of memory limit registers and instructions that change the current privilege mode of the CPU.) An attempt to execute a privileged instruction when the CPU is in non-privileged mode causes an interrupt. This interrupt converts the CPU to privileged mode and transfers control to an interrupt handler within the operating system.

When first turned on, the CPU is in privileged mode. Thus, when the operat- ing system starts at the end of the boot process, all instructions are executable. However, each time the operating system allows a process to start a time slice, it switches the CPU to nonprivileged mode by executing a "change privilege mode" instruction. In turn, the operating system will be notified if the process attempts to execute a privileged instruction, and thus the operating system will be in posi- tion to maintain the integrity of the computer system.

Privileged instructions and the control of privilege levels is the major tool available to operating systems for maintaining security. However, the use of these tools is a complex component of an operating system's design, and errors continue to be found in current systems. A single flaw in privilege level control can open the door to disaster from malicious programmers or from inadvertent programming errors. If a process is allowed to alter the timer that controls the system's multiprogramming system, that process can extend its time slice and dominate the machine. If a process is allowed to access peripheral devices directly, then it can read files without supervision by the system's file manager. If a process is allowed to access memory cells outside its allotted area, it can read and even alter data being used by other processes. Thus,

Copyright Virtual University of Pakistan

maintaining security continues to be an important task of an administrator as well as a goal in operating system design.

Copyright Virtual University of Pakistan

Module 59

59. Networking and the Internet: Network Classification

A computer network is often classified as being either a **personal area net- work (PAN),** a **local area network (LAN),** a **metropolitan area network (MAN),** or a **wide area network (WAN).** A PAN is normally used for short- range communications—typically less than a few meters—such as between a wireless headset and a smartphone or between a wireless mouse and its PC. In contrast, a LAN normally consists of a collection of computers in a single building or building complex. For example, the computers on a university campus or those in a manufacturing plant might be connected by a LAN. A MAN is a network of intermediate size, such as one spanning a local community. Finally, a WAN links machines over a greater distance—perhaps in neighboring cities or on opposite sides of the world.

Another means of classifying networks is based on whether the network's internal operation is based on designs that are in the public domain or on innovations owned and controlled by a particular entity such as an individual or a corporation. A network of the former type is called an **open** network; a network of the latter type is called a **closed,** or sometimes a **proprietary,** network. Open network designs are freely circulated and often grow in popularity to the point that they ultimately prevail over proprietary approaches whose applications are restricted by license fees and contract conditions.

The Internet (a popular worldwide network of networks that we will study in this chapter) is an open system. In particular, communication throughout the Internet is governed by an open collection of standards known as the TCP/IP proto- col suite. Anyone is free to use these standards without paying fees or signing license agreements. In contrast, a company such as Novell Inc. might develop proprietary systems for which it chooses to maintain ownership rights, allowing the company to draw income from selling or leasing these products.

Still another way of classifying networks is based on the topology of the network, which refers to the pattern in which the machines are connected. Two of the more popular topologies are the bus, in which the machines are all connected to a common communication line called a bus (Figure 59a), and the star, in which one machine serves as a central focal point to which all the others are connected (Figure 59b). The bus topology was popularized in the 1990s. When it was implemented under a set of standards known as Ethernet, and Ethernet networks remain one of the most popular networking systems in use today.

The star topology has roots as far back as the 1970s. It evolved from the paradigm of a large central computer serving many users. As the simple terminals employed by these users grew into small computers themselves, a star network emerged. Today, the star configuration is popular in wireless networks where communication is conducted by means of radio broadcast and the central machine, called the **access point (AP),** serves as a focal point around which all communication is coordinated.

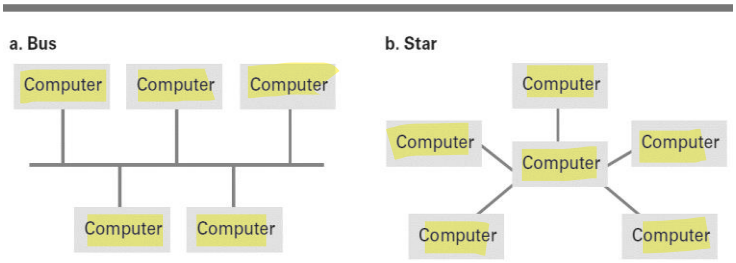


Figure 59: Two popular network topologies

Copyright Virtual University of Pakistan

Module 60

60. Networking and the Internet: Protocols

For a network to function reliably, it is important to establish rules by which activities are conducted. Such rules are called **protocols.** By developing and adopting protocol standards, vendors are able to build products for network applications that are compatible with products from other vendors. Thus, the development of protocol standards is an indispensable process in the development of networking technologies. As an introduction to the protocol concept, let us consider the problem of coordinating the transmission of messages among computers in a network. With- out rules governing this communication, all the computers might insist on trans- mitting messages at the same time or fail to assist other machines when that assistance is required.

In a bus network based on the Ethernet standards, the right to transmit messages is controlled by the protocol known as **Carrier Sense, Multiple Access with Collision Detection (CSMA/CD).** This protocol dictates that each message be broadcast to all the machines on the bus (Figure 60). Each machine monitors all the messages but keeps only those addressed to itself. To transmit a message, a machine waits until the bus is silent, and at this time it begins transmitting while continuing to monitor the bus. If another machine also begins transmitting, both machines detect the clash and pause for a brief, independently random period of time before trying to transmit again. The result is a system similar to that used by a small group of people in a conversation. If two people start to talk at once, they both stop. The difference is that people might go through a series such as, "I'm sorry, what were you going to say?", "No, no. You go first," whereas under the CSMA/CD protocol each machine merely tries again later.

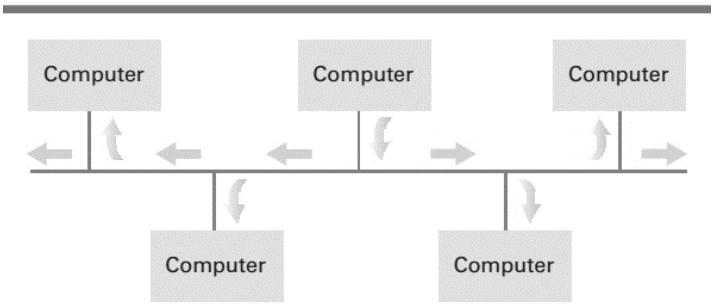


Figure 60: Communication over a bus network

Note that CSMA/CD is not compatible with wireless star networks in which all machines communicate through a central AP. This is because a machine may be unable to detect that its transmissions are colliding with those of another. For example, the machine may not hear the other because its own signal drowns out that of the other machine. Another cause might be that the signals from the different machines are blocked from each other by objects or distance even though they can all communicate with the central AP (a condition known as the **hidden terminal problem,** Figure 61). The result is that wireless networks adopt the policy of trying to avoid collisions rather than trying to detect them. Such policies are classified as **Carrier Sense, Multiple Access with Collision Avoidance (CSMA/CA),** many of which are standardized by IEEE.

Copyright Virtual University of Pakistan

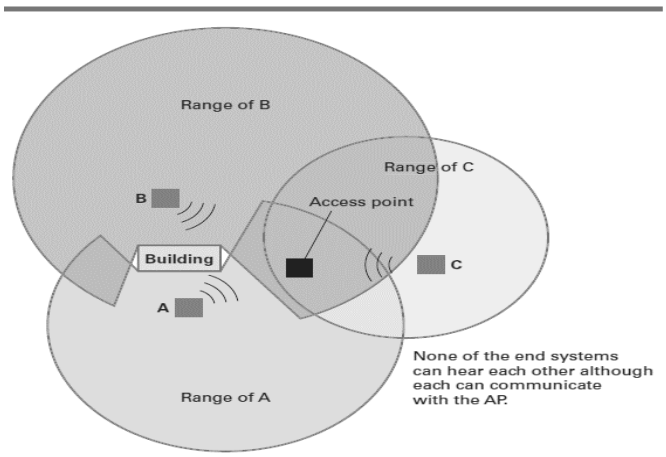


Figure 61: The hidden terminal problem

The most common approach to collision avoidance is based on giving advantage to machines that have already been waiting for an opportunity to transmit. The protocol used is similar to Ethernet's CSMA/CD. The basic difference is that when a machine first needs to transmit a message and finds the communication channel silent, it does not start transmitting immediately. Instead, it waits for a short period of time and then starts transmitting only if the channel has remained silent throughout that period. If a busy channel is experienced during this process, the machine waits for a randomly determined period before trying again. Once this period is exhausted, the machine is allowed to claim a silent channel without hesitation. This means that collisions between "newcomers" and those that have already been waiting are avoided because a "newcomer" is not allowed to claim a silent channel until any machine that has been waiting is given the opportunity to start.

This protocol, however, does not solve the hidden terminal problem. After all, any protocol based on distinguishing between a silent or busy channel requires that each individual station be able to hear all the others. To solve this problem, some WiFi networks require that each machine send a short "request" message to the AP and wait until the AP acknowledges that request before transmitting an entire message. If the AP is busy because it is dealing with a "hidden terminal," it will ignore the request, and the requesting machine will know to wait. Otherwise, the AP will acknowledge the request, and the machine will know that it is safe to transmit. Note that all the machines in the network will hear all acknowledgments sent from the AP and thus have a good idea of whether the AP is busy at any given time, even though they may not be able to hear the transmissions taking place.

Copyright Virtual University of Pakistan

Module 61

61. Networking and the Internet: Combining Networks

Sometimes it is necessary to connect existing networks to form an extended communication system. This can be done by connecting the networks to form a larger version of the same "type" of network. For example, in the case of bus networks based on the Ethernet protocols, it is often possible to connect the buses to form a single long bus. This is done by means of different devices known as repeaters, bridges, and switches, the distinctions of which are subtle yet informative. The simplest of these is the **repeater,** which is little more than a device that passes signals back and forth between the two original buses (usually with some form of amplification) without considering the meaning of the signals (Figure 62a).

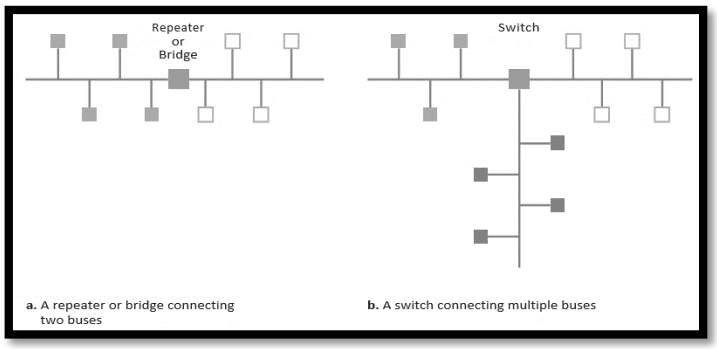


Figure 62: Building a large bus network from smaller ones

A **bridge** is similar to, but more complex than, a repeater. Like a repeater, it connects two buses, but it does not necessarily pass all messages across the connection. Instead, it looks at the destination address that accompanies each message and forwards a message across the connection only when that message is destined for a computer on the other side. Thus, two machines residing on the same side of a bridge can exchange messages without interfering with communication taking place on the other side. A bridge produces a more efficient system than that produced by a repeater.

A **switch** is essentially a bridge with multiple connections, allowing it to connect several buses rather than just two. Thus, a switch produces a network consist- ing of several buses extending from the switch as spokes on a wheel (Figure 62b). As in the case of a bridge, a switch considers the destination addresses of all messages and forwards only those messages destined for other spokes. Moreover, each message that is forwarded is relayed only into the appropriate spoke, thus minimizing the traffic in each spoke.

It is important to note that when networks are connected via repeaters, bridges, and switches, the result is a single large network. The entire system operates in the same manner (using the same protocols) as each of the original smaller networks.

Sometimes, however, the networks to be connected have incompatible characteristics. For instance, the characteristics of a WiFi network are not readily compatible with an Ethernet network. In these cases the networks must be connected in a manner that builds a network of networks, known as an **internet,** in which the original networks maintain their individuality and continue to function as autonomous networks. (Note that the generic term *internet* is distinct from *the Internet*. The Internet, written with an uppercase *I*, refers to a particular, world- wide internet that we will study in later sections of this chapter. There are many other examples of internets. Indeed, traditional telephone communication was handled by worldwide internet systems well before the Internet was popularized.) The connection between networks to form an internet is handled by devices known as **routers,** which are special purpose computers used for forwarding messages. Note that the task of a router is different from that of repeaters, bridges, and switches in that routers provide links between networks while allowing each network to maintain its unique internal characteristics. As an example, Figure 63 depicts two WiFi star networks and an Ethernet bus network connected by routers.

Copyright Virtual University of Pakistan

When a machine in one of the WiFi networks wants to send a message to a machine in the Ethernet network, it first sends the message to the AP in its network. From there, the AP sends the message to its associated router, and this router forwards the message to the router at the Ethernet. There the message is given to a machine on the bus, and that machine then forwards the message to its final destination in the Ethernet.

The reason that routers are so named is that their purpose is to forward messages in their proper directions. This forwarding process is based on an internet- wide addressing system in which all the devices in an internet (including the machines in the original networks and the routers) are assigned unique addresses. (Thus, each machine in one of the original networks has two addresses: its original "local" address within its own network and its internet address.) A machine wanting to send a message to a machine in a distant network attaches the internet address of the destination to the message and directs the message to its local router. From there it is forwarded in the proper direction. For this forwarding purpose, each router maintains a **forwarding table** that contains the router's knowledge about the direction in which messages should be sent depending on their destination addresses.

The "point" at which one network is linked to an internet is often called a **gateway** because it serves as a passageway between the network and the outside world. Gateways can be found in a variety of forms, and thus the term is used rather loosely. In many cases a network's gateway is merely the router through which it communicates with the rest of the internet. In other cases the term *gateway* may be used to refer to more than just a router. For example, in most residential WiFi networks that are connected to the Internet, the term *gateway* refers collectively to both the network's AP and the router connected to the AP because these two devices are normally packaged in a single unit.

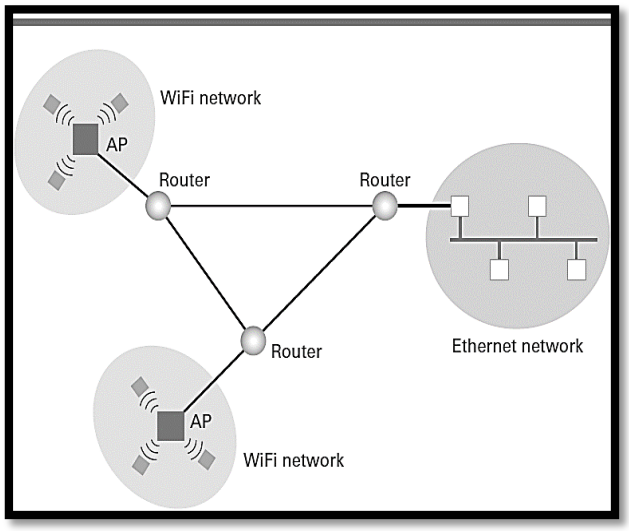


Figure 63: Routers connecting two WiFi networks and an Ethernet network to form an internet

Copyright Virtual University of Pakistan

Module 62

62. Networking and the Internet: Methods of Process Communication

The various activities (or processes) executing on the different computers within a network (or even executing on the same machine via time-sharing/multitasking) must often communicate with each other to coordinate their actions and to per- form their designated tasks. Such communication between processes is called **inter-process communication.** A popular convention used for inter-process communication is the **client/**

server model. This model defines the basic roles played by the processes as either a **client,** which makes requests of other processes, or a **server,** which satisfies the requests made by clients.

An early application of the client/server model appeared in networks connecting all the computers in a cluster of offices. In this situation, a single, high- quality printer was attached to the network where it was available to all the machines in the network. In this case the printer played the role of a server (often called a **print server**), and the other machines were programmed to play the role of clients that sent print requests to the print server.

Another early application of the client/server model was used to reduce the cost of magnetic disk storage while also removing the need for duplicate copies of records. Here one machine in a network was equipped with a high-capacity mass storage system (usually a magnetic disk) that contained all of an organization's records. Other machines on the network then requested access to the records as they needed them. Thus the machine that actually contained the records played the role of a server (called a **file server**), and the other machines played the role of clients that requested access to the files that were stored at the file server.

Today the client/server model is used extensively in network applications, as we will see later in this chapter. However, the client/server model is not the only means of inter-process communication. Another model is the **peer-to-peer** (often abbreviated **P2P**) model. Whereas the client/server model involves one process (the server) providing a service to numerous others (clients), the peer-to- peer model involves processes that provide service to and receive service from each other (Figure 64). Moreover, whereas a server must execute continuously so that it is prepared to serve its clients at any time, the peer-to-peer model usually involves processes that execute on a temporary basis. For example, applications of the peer-to-peer model include instant messaging in which people carry on a written conversation over the Internet as well as situations in which people play competitive interactive games.

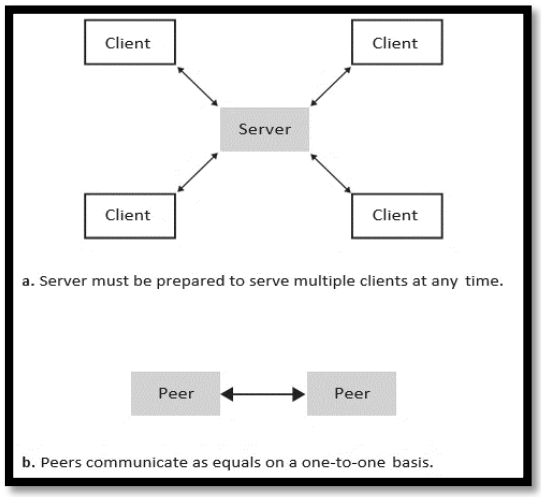


Figure 64: The client/server model compared to the peer-to-peer model

Copyright Virtual University of Pakistan

The peer-to-peer model is also a popular means of distributing files such as music recordings and motion pictures via the Internet. In this case, one peer may receive a file from another and then provide that file to other peers. The collection of peers participating in such a distribution is sometimes called a swarm. The swarm approach to file distribution is in contrast to earlier approaches that applied the client/server model by establishing a central distribution center (the server) from which clients downloaded files (or at least found sources for those files).

One reason that the P2P model is replacing the client/server model for file sharing is that it distributes the service task over many peers rather than concentrating it at one server. This lack of a centralized base of operation leads to a more efficient system. Unfortunately, another reason for the popularity of file distribution systems based on the P2P model is that, in cases of questionable legality, the lack of a central server makes legal efforts to enforce copyright laws more difficult. There are numerous cases, however, in which individuals have discovered that "difficult" does not mean "impossible" and have found themselves faced with significant liabilities due to copyright infringement violations. You might often read or hear the term *peer-to-peer network*, which is an exam-

ple of how misuse of terminology can evolve when technical terms are adopted by the nontechnical community. The term *peer-to-peer* refers to a system by which two processes communicate over a network (or internet). It is not a property of the network (or internet). A process might use the peer-to-peer model to communicate with another process and later use the client/server model to communicate with another process over the same network. Thus, it would be more accurate to speak of communicating by means of the peer-to-peer model rather than communicating over a peer-to-peer network.

Copyright Virtual University of Pakistan

Module 63

63. Networking and the Internet: Distributed Systems

With the success of networking technology, interaction between computers via networks has become common and multifaceted. Many modern software systems, such as global information retrieval systems, company-wide accounting and inventory systems, computer games, and even the software that controls a network's infrastructure itself are designed as **distributed systems,** meaning that they consist of software units that execute as processes on different computers. Early distributed systems were developed independently from scratch. But today, research is revealing a common infrastructure running throughout these systems, including such things as communication and security systems. In turn, efforts have been made to produce prefabricated systems that provide this basic infrastructure and therefore allow distributed applications to be constructed by merely developing the part of the system that is unique to the application.

Several types of distributed computing systems are now common. **Cluster computing** describes a distributed system in which many independent computers work closely together to provide computation or services comparable to a much larger machine. The cost of these individual machines, plus the high-speed network to connect them, can be less than a higher-priced supercomputer, but with higher reliability and lower maintenance costs. Such distributed systems are used to provide **high-availability**—because it is more likely that at least one member of the cluster will be able to answer a request, even if other cluster members break down or are unavailable—and **load-balancing**—because the workload can be shifted automatically from members of the cluster that have too much to do to those that may have too little. **Grid computing** refers to distributed systems that are more loosely coupled than clusters but that still work together to accomplish large tasks. Grid computing can involve specialized software to make it easier to distribute data and algorithms to the machines participating in a grid. Examples include University of Wisconsin's Condor system, or Berkeley's Open Infrastructure for Network Computing (BOINC). Both of these systems are often installed on computers that are used for other purposes, such as PCs at work or at home, that can then volunteer computing power to the grid when the machine is not otherwise being used. Enabled by the growing connectivity of the Internet, this type of voluntary, distributed grid computing has enabled mil- lions of home PCs to work on enormously complex mathematical and scientific problems. **Cloud computing,** whereby huge pools of shared computers on the network can be allocated for use by clients as needed, is the latest trend in distributed systems. Much as the spread of metropolitan electrical grids in the early twentieth century eliminated the need for individual factories and businesses to maintain their own generators, the Internet is making it possible for entities to entrust their data and computations to "the Cloud," which in this case refers to the enormous computing resources already available on the network. Services such as Amazon's Elastic Compute Cloud allow clients to rent virtual computers by the hour, without concern for where the computer hardware is actually located. Google Drive and Google Apps allow users to collaborate on information or build Web services without needing to know how many computers are working on the problem or where the relevant data are stored. Cloud computing services provide reasonable guarantees of reliability and scalability, but also raise concerns about privacy and security in a world where we may no longer know who owns and operates the computers that we use.

Copyright Virtual University of Pakistan

Module 64

64. Networking and the Internet: Internet Architecture

As we have already mentioned, the Internet is a collection of connected net- works. In general, these networks are constructed and maintained by organizations called **Internet Service Providers (ISPs).** It is also customary to use the term ISP in reference to the networks themselves. Thus, we will speak of connecting to an ISP, when what we really mean is connecting to the network provided by an ISP.

The system of networks operated by the ISPs can be classified in a hierarchy according to the role they play in the overall Internet structure (Figure 65). At the top of this hierarchy are relatively few **tier-1 ISPs** that consist of very high- speed, high-capacity, international WANs. These networks are thought of as the backbone of the Internet. They are typically operated by large companies that are in the communications business. An example would be a company that originated as a traditional telephone company and has expanded its scope into providing other communication services.

Connecting to the tier-1 ISPs are the **tier-2 ISPs** that tend to be more regional in scope and less potent in their capabilities. (The distinction between the tier-1 and tier-2 ISPs is often a matter of opinion.) Again, these networks tend to be operated by companies in the communications business.

Tier-1 and tier-2 ISPs are essentially networks of routers that collectively provide the Internet's communication infrastructure. As such, they can be thought of as the core of the Internet. Access to this core is usually provided by an intermediary called an **access** or **tier-3 ISP.** An access ISP is essentially an independent internet, sometimes called an **intranet,** operated by a single authority that is in the business of supplying Internet access to individual homes and businesses. Examples include cable and telephone companies that charge for their service as well as organizations such as universities or corporations that take it upon themselves to provide Internet access to individuals within their organizations.

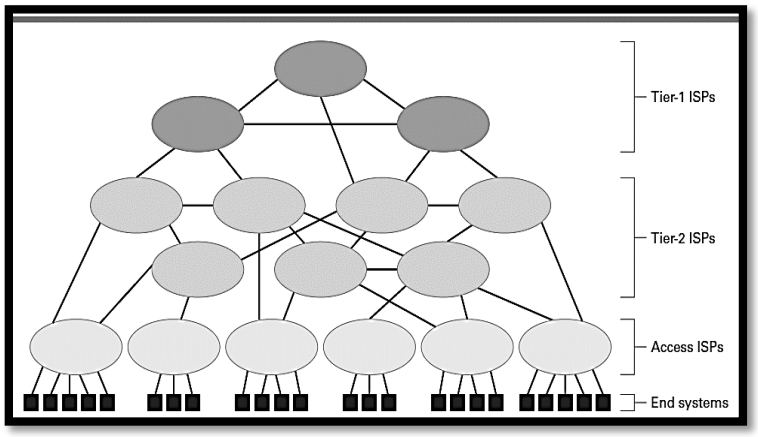


Figure 65: Internet composition

The devices that individual users connect to the access ISPs are known as **end systems** or **hosts.** These end systems may be laptops or PCs, but increasingly range over a multitude of other devices including telephones, video cameras, automobiles, and home appliances. After all, the Internet is essentially a communications system, and thus any device that would benefit from communicating with other devices is a potential end system.

The technology by which end systems connect to larger networks is also varied. Perhaps the fastest growing are wireless connections based on WiFi technology. The strategy is to connect the AP to an access ISP and thus provide Internet access through that ISP to end systems within the AP's broadcast range. The area within the AP or group of APs' range is often called a **hot spot,** particularly when the network access is publicly available or free. Hot spots can be found in individual residences, hotel and office buildings, small

Copyright Virtual University of Pakistan

businesses, parks, and in some cases span entire cities. A similar technology is used by the cellular telephone industry where hot spots are known as cells and the "routers" generating the cells are coordinated to provide continuous service as an end system moves from one cell to another.

Copyright Virtual University of Pakistan

Module 65

65. Networking and the Internet: Internet Addressing

Internet needs an internet-wide addressing system that assigns a unique identifying address to each computer in the sys- tem. In the Internet these addresses are known as **IP addresses.** Originally, each IP address was a pattern of 32 bits, but to provide a larger set of addresses, the process of converting to 128-bit addresses is currently underway. Blocks of consecutively numbered IP addresses are awarded to ISPs by the **Internet Corporation for Assigned Names and Numbers (ICANN),** which is a nonprofit corporation established to coordinate the Internet's operation. The ISPs are then allowed to allocate the addresses within their awarded blocks to machines within their region of authority. Thus, machines throughout the Internet are assigned unique IP addresses.

IP addresses are traditionally written in **dotted decimal notation** in which the bytes of the address are separated by periods and each byte is expressed as an integer represented in traditional base 10 notation. For example, using dotted decimal notation, the pattern 5.2 would represent the two-byte bit pattern 0000010100000010, which consists of the byte 00000101 (represented by 5) followed by the byte 00000010 (represented by 2), and the pattern 17.12.25 would represent the three-byte bit pattern consisting of the byte 00010001 (which is 17 written in binary notation), followed by the byte 00001100 (12 written in binary), followed by the byte 00011001 (25 written in binary). In summary, a 32-bit IP address might appear as 192.207.177.133 when expressed in dotted decimal notation.

Addresses in bit-pattern form (even when compressed using dotted decimal notation) are rarely conducive to human consumption. For this reason the Internet has an alternative addressing system in which machines are identified by mnemonic names. This addressing system is based on the concept of a **domain,** which can be thought of as a "region" of the Internet operated by a single authority such as a university, club, company, or government agency. (The word region is in quotations here because, as we will soon see, such a region may not correspond to a physical area of the Internet.) Each domain must be registered with ICANN—a process handled by companies, called **registrars,** that have been assigned this role by ICANN. As a part of this registration process, the domain is assigned a mnemonic **domain name,** which is unique among all the domain names throughout the Internet. Domain names are often descriptive of the organization registering the domain, which enhances their utility for humans.

As an example, the domain name of Marquette University is mu.edu. Note the suffix following the period. It is used to reflect the domain's classification, which in this case is "educational" as indicated by the edu suffix. These suffixes are called **top-level domains (TLDs).** Other TLDs include com for commercial institutions, gov for U.S. government institutions, org for nonprofit organizations, museum for museums, info for unrestricted use, and net, which was originally intended for ISPs but is now used on a much broader scale. In addition to these general TLDs, there are also two-letter TLDs for specific countries (called **country-code TLDs**) such as au for Australia and ca for Canada.

Once a domain's mnemonic name is registered, the organization that registered the name is free to extend the name to obtain mnemonic identifiers for individual items within the domain. For example, an individual host within Marquette University may be identified as eagle.mu.edu. Note that domain names are extended to the left and separated by a period. In some cases multiple extensions, called **subdomains,** are used as a means of organizing the names within a domain. These subdomains often represent different networks within the domain's jurisdiction. For example, if Yoyodyne Corporation was assigned the domain name yoyodyne.com, then an individual computer at Yoyodyne might have a name such as overthruster.propulsion.yoyodyne.com, meaning that the computer overthruster is in the subdomain propulsion within the domain yoyodyne within the TLD com. (We should emphasize that the dotted notation used in mnemonic addresses is not related to the dotted decimal notation used to represent addresses in bit pattern form.)

Although mnemonic addresses are convenient for humans, messages are always transferred over the Internet by means of IP addresses. Thus, if a human wants to send a message to a distant machine and

Copyright Virtual University of Pakistan

identifies the destination by means of a mnemonic address, the software being used must be able to convert that address into an IP address before transmitting the message. This conversion is performed with the aid of numerous servers, called **name servers,** that are essentially directories that provide address translation services to clients. Collectively, these name servers are used as an Internet-wide directory system known as the **domain name system (DNS).** The process of using DNS to perform a translation is called a **DNS lookup.**

Thus, for a machine to be accessible by means of a mnemonic domain name, that name must be represented in a name server within the DNS. In those cases, in which the entity establishing the domain has the resources, it can establish and maintain its own name server containing all the names within that domain. Indeed, this is the model on which the domain system was originally based. Each registered domain represented a physical region of the Internet that was operated by a local authority such as a company, university, or government agency. This authority was essentially an access ISP that provided Internet access to its members by means of its own intranet that was linked to the Internet. As part of this system, the organization maintained its own name server that provided translation services for all the names used within its domain.

This model is still common today. However, many individuals or small organizations want to establish a domain presence on the Internet without committing the resources necessary to support it. For example, it might be beneficial for a local chess club to have a presence on the Internet as KingsandQueens.org, but the club would likely not have the resources to establish its own network, maintain a link from this network to the Internet, and implement its own name server. In this case, the club can contract with an access ISP to create the appearance of a registered domain using the resources already established by the ISP. Typically, the club, perhaps with the assistance of the ISP, registers the name chosen by the club and contracts with the ISP to have that name included in the ISP's name server. This means that all DNS lookups regarding the new domain name will be directed to the ISP's name server, from which the proper translation will be obtained. In this way, many registered domains can reside within a single ISP, each often occupying only a small portion of a single computer.

Copyright Virtual University of Pakistan

Module 66

66. Networking and the Internet: Internet Applications

In the earlier days of the Internet, most applications were separate, simple programs that each followed a network protocol. A newsreader application contacted servers using the **Network News Transfer Protocol (NNTP),** an application for listing and copying files across the network implemented the **File Transfer Protocol (FTP),** or an application for accessing another computer from a great distance used the **Telnet** protocol, or later the **Secure Shell (SSH)** protocol. As webservers and browsers have become more sophisticated, more and more of these traditional network applications have come to be handled by webpages via the powerful **Hyper Text Transfer Protocol (HTTP).**

Electronic Mail A wide variety of systems now exist for exchanging messages between end users over the network; instant messaging (IM), browser-based online chatting, Twitter-based "tweets", and the Facebook "wall" are but a few. One of the oldest and most enduring uses of the Internet is the electronic mail system, or email for short.

VoIP As an example of a more recent Internet application, consider **VoIP (Voice over Internet Protocol)** in which the Internet infrastructure is used to provide voice communication similar to that of traditional telephone systems. In its simplest form, VoIP consists of two processes on different machines transferring audio data via the P2P model—a process that in itself presents no significant problems. However, tasks such as initiating and receiving calls, linking VoIP with traditional telephone systems, and providing services such as emergency 911 communication are issues that extend beyond traditional Internet applications. Moreover, governments that own their country's traditional telephone companies view VoIP as a threat and have either taxed it heavily or outlawed it completely.

Internet Multimedia Streaming An enormous portion of current Internet traffic is used for transporting audio and video across the Internet in real-time, known as streaming. Netflix streamed more than 4 billion hours of programming to end users in the first three months of 2013 alone. Combined with YouTube, these two services will consume more than half of the bandwidth of the Internet in 2014.

Copyright Virtual University of Pakistan

Module 67

67. Networking and the Internet: Internet Applications: Email

Messaging applications

- ✓ Instant Messaging
- ✓ Browser based chatting
- ✓ Twitter based tweets
- ✓ Facebook wall.
- ✓ One of the oldest is Electronic mail (Email)

Protocol

- ✓ Simple Mail Transfer Protocol (SMTP)

Scenario

- ✓ mafzal from cust.edu.pk wants to send email to hmaurer from iicm.tugraz.at, the flow has been shown in the Figure 66.

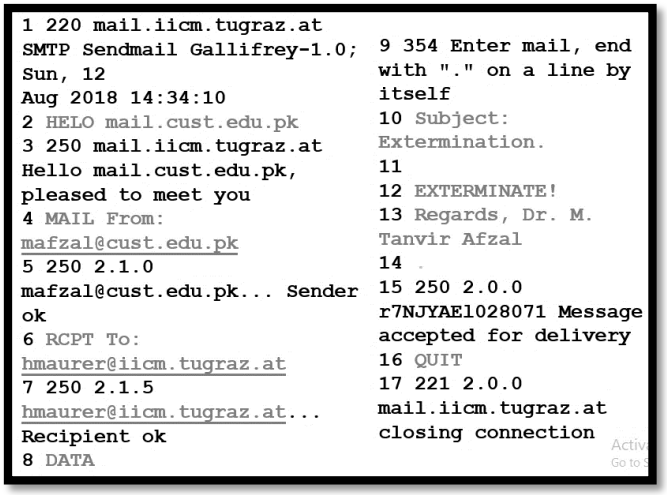


Figure 66: Email procedure

Other Protocols

- ✓ SMTP for text messages.
- ✓ MIME (Multipurpose Internet Mail Extensions) to convert non-ASCII to SMTP compatible form.
- ✓ Post Office Protocol Version 3 (POP3)
- ✓ Internet Mail Access Protocol (IMAP)
- ✓ User can download and maintain mail messages into folders etc, POP3 helps to store messages on local machine, IMAP on mail server machine.

Copyright Virtual University of Pakistan

Module 68

68. Networking and the Internet: VoIP

As an example of a more recent Internet application, consider **VoIP (Voice over Internet Protocol)** in which the Internet infrastructure is used to provide voice communication similar to that of traditional telephone systems. In its simplest form, VoIP consists of two processes on different machines transferring audio data via the P2P model—a process that in itself presents no significant problems. However, tasks such as initiating and receiving calls, linking VoIP with traditional telephone systems, and providing services such as emergency 911 communication are issues that extend beyond traditional Internet applications. Moreover, governments that own their country's traditional telephone companies view VoIP as a threat and have either taxed it heavily or outlawed it completely. Existing VoIP systems come in four different forms that are competing for popularity. VoIP **soft phones** consist of P2P software that allows two or more PCs to share a call with no more special hardware than a speaker and a microphone. An example of a VoIP soft phone system is Skype, which also provides its clients with links to the traditional telephone communication system. One drawback to Skype is that it is a proprietary system, and thus much of its operational structure is not publicly known. This means that Skype users must trust the integrity of the Skype software without third-party verification. For instance, to receive calls, a Skype user must leave his or her PC connected to the Internet and available to the Skype system, which means that some of the PC's resources may be used to support other Skype communications without the PC owner's awareness—a feature that has generated some resistance.

A second form of VoIP consists of **analog telephone adapters,** which are devices that allow a user to connect his or her traditional telephone to phone service provided by an access ISP. This choice is frequently bundled with traditional Internet service and/or digital television service.

The third type of VoIP comes in the form of embedded VoIP phones, which are devices that replace a traditional telephone with an equivalent handset connected directly to a TCP/IP network. Embedded VoIP phones are becoming increasingly common for large organizations, many of whom are replacing their traditional internal copper wire telephone systems with VoIP over Ethernet to reduce costs and enhance features.

Finally, the current generation of smartphones use wireless VoIP technology. That is, earlier generations of wireless phones only communicated with the telephone company's network using that company's protocols. Access to the Internet was obtained by gateways between the company's network and the Internet, at which point signals were converted to the TCP/IP system. However, the 4G phone network is an IP-based network throughout, which means a 4G telephone is essentially just another broadband-connected host computer on the global Internet.

Copyright Virtual University of Pakistan

Module 69

69. Networking and the Internet: Internet Multimedia Streaming

An enormous portion of current Internet traffic is used for transporting audio and video across the Internet in real-time, known as **streaming.** Netflix streamed more than 4 billion hours of programming to end users in the first three months of 2013 alone. Combined with YouTube, these two services will consume more than half of the bandwidth of the Internet in 2014.

On the surface, Internet streaming may not seem to require special consideration. For example, one might guess that an Internet radio station could merely establish a server that would send program messages to each of the clients who requested them. This technique is known as **N-unicast.** (More precisely, *unicast* refers to one sender sending messages to one receiver, whereas *N-unicast* refers to a single sender involved with multiple unicasts.) The N-unicast approach has been applied but has the drawback of placing a substantial burden on the station's server as well as on the server's immediate Internet neighbors. Indeed, N-unicast forces the server to send individual messages to each of its clients on a real-time basis, and all these messages must be forwarded by the server's neighbors.

Most alternatives to N-unicast represent attempts to alleviate this problem. One applies the P2P model in a manner reminiscent of file-sharing systems. That is, once a peer has received data, it begins to distribute that data to those peers that are still waiting, meaning that much of the distribution problem is transferred from the data's source to the peers.

Another alternative, called **multicast,** transfers the distribution problem to the Internet routers. Using multicast, a server transmits a message to multiple clients by means of a single address and relies on the routers in the Internet to recognize the significance of that address and to produce and forward copies of the message to the appropriate destinations. Note then that applications relying on multicast require that the functionality of the Internet routers be expanded beyond their original duties. Multicast support has been implemented in small networks, but has yet to expand to the global Internet.

More importantly, most applications in this category are now **on-demand streaming,** in which the end user expects to view or listen to media at an arbitrary time of his or her choosing. This is quite a different problem from the Inter- net radio station example, because each end user expects to be able to start, pause, or rewind content at his or her own pace. In this case, N-unicast and multicast technologies are of little help. Each on-demand stream is effectively unicast from a media server that stores the content to the end user that wishes to retrieve it.

In order for this type of streaming to scale to thousands or even millions of simultaneous users, each with his or her own personal stream, replication of the content to many distinct servers is essential. Large-scale streaming services make use of **content delivery networks (CDNs),** groups of servers distributed strategically around the Internet that specialize in streaming copies of content to nearby end users in their network "neighborhood." In many cases, CDN machines may reside in an access ISP network, allowing customers of that access ISP to stream copies of multimedia content at high speed from a nearby server that is much closer in the network than the streaming service's central server machines. A networking technology called **anycast,** which enables an end user to automatically connect to the closest server out of a defined group of servers, helps to make CDNs practical.

Internet streaming of high-definition, on-demand video has permeated far more than traditional PCs. A broad class of embedded devices such as televisions, DVD/Blu-ray players, smartphones, and game consoles connect directly to the TCP/IP network to select viewable content from a multitude of both free and subscription servers.

Copyright Virtual University of Pakistan

Module 70

70. Networking and the Internet: World Wide Web

The World Wide Web had its origins in the work of Tim Berners-Lee who realized the potential of combining internet technology with the concept of linked- documents, called **hypertext.** His first software for implementing the Web was released in December 1990. While this early prototype did not yet support multimedia data, it included the key components of what we now recognize as the World Wide Web: a hypertext document format for embedding **hyperlinks** to other documents; a protocol for transferring hypertext across the network, and a server process that supplied hypertext pages upon request. From this humble beginning, the Web quickly grew to support images, audio and video, and by the mid-1990s had become the dominant application powering the growth of the Internet.

Copyright Virtual University of Pakistan

Module 71

71. Networking and the Internet: Web implementations

Software packages that allow users to access hypertext on the Internet fall into one of two categories: **browsers** and **webservers.** A browser resides on the user's computer and is charged with the tasks of obtaining materials requested by the user and presenting these materials to the user in an organized manner. Common Internet browsers include Firefox, Safari, and Internet Explorer. The webserver resides on a computer containing hypertext documents to be accessed. Its task is to provide access to the documents under its control as requested by clients (browsers). Hypertext documents are normally transferred between browsers and webservers using a protocol known as the Hypertext Transfer Protocol (HTTP). In order to locate and retrieve documents on the Web, each document is given a unique address called a **Uniform Resource Locator (URL).** Each URL contains the information needed by a browser to contact the proper server and request the desired document. Thus, to view a webpage, a person first provides his or her browser with the URL of the desired document and then instructs the browser to retrieve and display the document.

A typical URL is presented in Figure 67. It consists of four segments: the protocol to use to communicate with the server controlling access to the document, the mnemonic address of the machine containing the server, the directory path needed for the server to find the directory containing the document, and the name of the document itself. In short, the URL in Figure 67 tells a browser to contact the webserver on the computer known as eagle.mu.edu using the protocol HTTP and to retrieve the document named Julius_Caesar.html found within the subdirectory Shakespeare within the directory called authors.

Sometimes a URL might not explicitly contain all the segments shown in Figure 67. For example, if the server does not need to follow a directory path to reach the document, no directory path will appear in the URL. Moreover, sometimes a URL will consist of only a protocol and the mnemonic address of a computer. In these cases, the webserver at that computer will return a predetermined document, typically called a home page, that usually describes the information available at that website. Such shortened URLs provide a simple means of contacting organizations. For example, the URL http://www.google.com will lead to the home page of Google, which contains hyperlinks to the services, products, and documents relating to the company.

To further simplify locating websites, many browsers assume that the HTTP protocol should be used if no protocol is identified. These browsers correctly retrieve the Google home page when given the "URL" consisting merely of www.google.com.

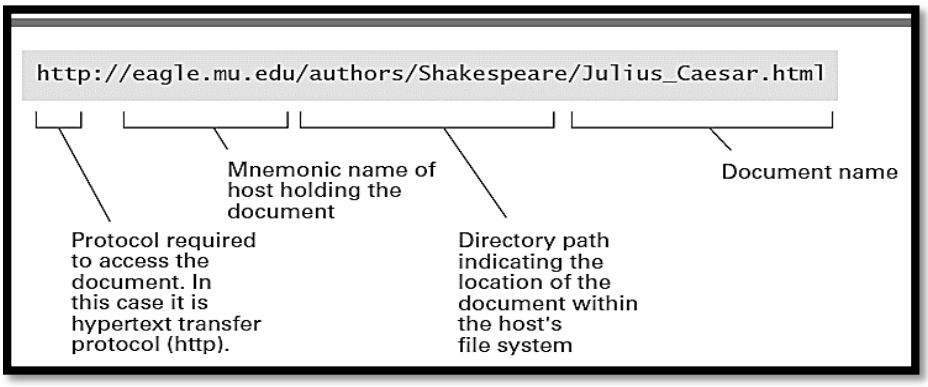


Figure 67: A typical URL

Copyright Virtual University of Pakistan

Module 72

72. Networking and the Internet: HTML

A traditional hypertext document is similar to a text file because its text is encoded character by character using a system such as ASCII or Unicode. The distinction is that a hypertext document also contains special symbols, called **tags,** that describe how the document should appear on a display screen, what multimedia resources (such as images) should accompany the document, and which items within the document are linked to other documents. This system of tags is known as **Hypertext Markup Language (HTML).**

Thus, it is in terms of HTML that an author of a webpage describes the information that a browser needs in order to present the page on the user's screen and to find any related documents referenced by the current page. The process is analogous to adding typesetting directions to a plain typed text (perhaps using a red pen) so that a typesetter will know how the material should appear in its final form. In the case of hypertext, the red markings are replaced by HTML tags, and a browser ultimately plays the role of the typesetter, reading the HTML tags to learn how the text is to be presented on the computer screen.

Copyright Virtual University of Pakistan

Module 73

73. Networking and the Internet: HTML

A traditional hypertext document is similar to a text file because its text is the HTML-encoded version (called the **source** version) of an extremely simple webpage is shown in Figure 68a. Note that the tags are delineated by the symbols < and >. The HTML source document consists of two sections—a head (surrounded by the <head> and </head> tags) and a body (surrounded by the <body> and </body> tags). The distinction between the head and body of a web- page is similar to that of the head and body of an interoffice memo. In both cases, the head contains preliminary information about the document (date, subject, etc. in the case of a memo). The body contains the meat of the document, which in the case of a webpage is the material to be presented on the computer screen when the page is displayed.

The head of the webpage displayed in Figure 68 contains only the title of the document (surrounded by "title" tags). This title is only for documentation purposes; it is not part of the page that is to be displayed on the computer screen. The material that is displayed on the screen is contained in the body of the document. The first entry in the body of the document in Figure 68a is a level-one heading (surrounded by the <h1> and </h1> tags) containing the text "My Web Page." Being a level-one heading means that the browser should display this text prominently on the screen. The next entry in the body is a paragraph of text (surrounded by the and tags) containing the text "Click here for another page." Figure 68b shows the page as it would be presented on a computer screen by a browser.

In its present form, the page in Figure 68 is not fully functional in the sense that nothing will happen when the viewer clicks on the word *here*,

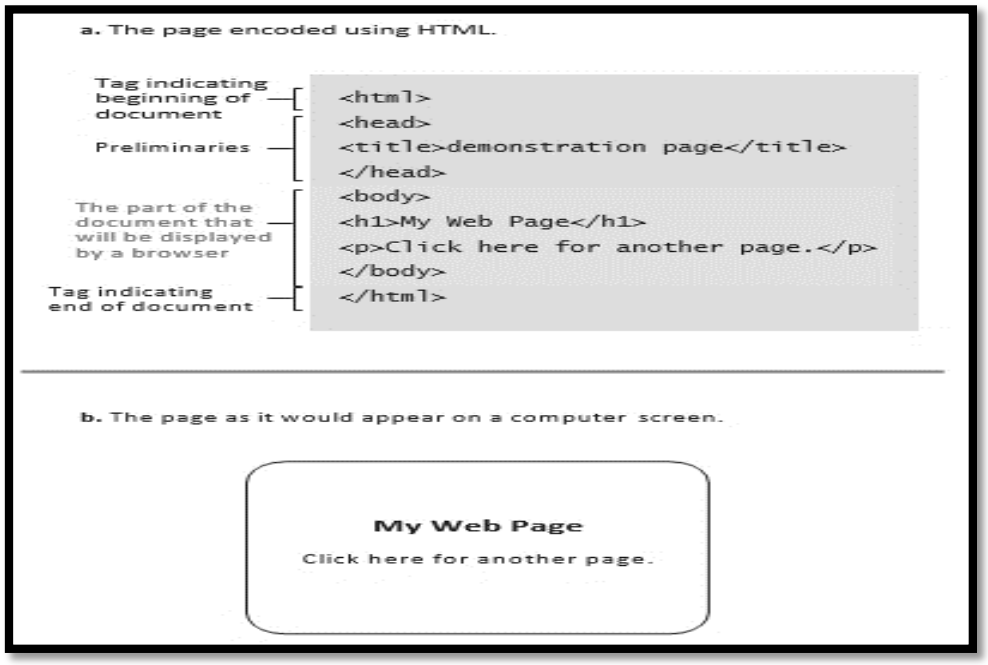


Figure 68: A simple webpage

Copyright Virtual University of Pakistan

even though the page implies that doing so will cause the browser to display another page. To cause the appropriate action, we must link the word *here* to another document.

Let us suppose that, when the word *here* is clicked, we want the browser to retrieve and display the page at the URL http://crafty.com/demo.html. To do so, we must first surround the word *here* in the source version of the page with the tags <a> and , which are called anchor tags. Inside the opening anchor tag we insert the parameter

href = http://crafty.com/demo.html

(as shown in Figure 69a) indicating that the hypertext reference (href) associated with the tag is the URL following the equal sign (http://crafty.com/demo.html). Having added the anchor tags, the webpage will now appear on a computer screen as shown in Figure 4.10b. Note that this is identical to Figure 68b except that the word *here* is highlighted by color indicating that it is a link to another webpage. Clicking on such highlighted terms will cause the browser to retrieve and display the associated webpage. Thus, it is by means of anchor tags that webpages are linked to each other.

Finally, we should indicate how an image could be included in our simple webpage. For this purpose, let us suppose that a JPEG encoding of the image we want to include is stored as the file named OurPic.jpg in the directory Images at Images.com and is available via the webserver at that location. Under these conditions, we can tell a browser to display the image at the top of the webpage by inserting the image tag <img src = "http://Images.com/Images/OurPic.jpg"> immediately after the <body> tag in the HTML source document. This tells the browser that the image named OurPic.jpg should be displayed at the beginning of the document. (The term src is short for "source," meaning that the information following the equal sign indicates the source of the image to be displayed.) When the browser finds this tag, it will send a message to the HTTP server at Images.com requesting the image called OurPic.jpg and then display the image appropriately.

If we moved the image tag to the end of the document just before the </body> tag, then the browser would display the image at the bottom of the webpage. There are, of course, more sophisticated techniques for positioning an image on a webpage, but these need not concern us now.

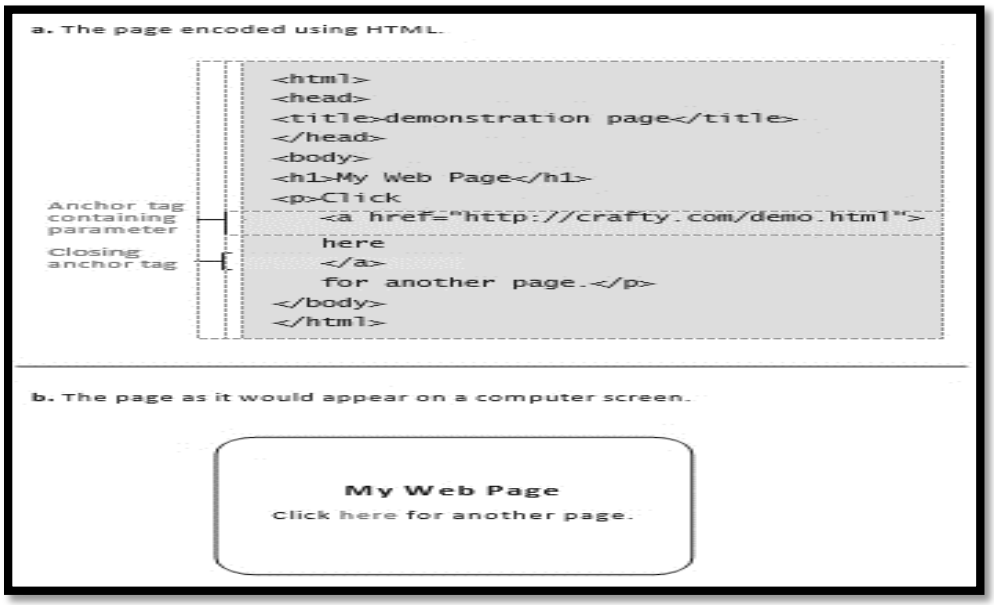


Figure 69: An enhanced simple webpage

Copyright Virtual University of Pakistan

Module 74

74. Networking and the Internet: More on HTML

If you are further interested to develop HTML pages, here is a detailed content available for you:

W3schools

https://www.w3schools.com/tags/default.asp

Here you can get information about all HTML tags and how they can be used. For example, if you click on the tag "<b>" available on the left panel on the above URL, you will be shown the screen like shown in the Figure 70.

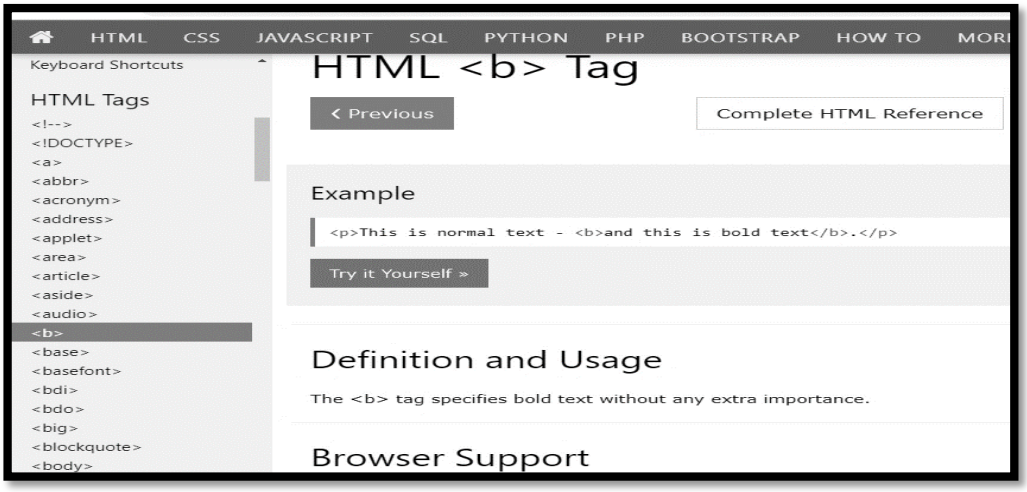


Figure 70: <b> tag on w3school

If you further click on the green button "Try it Yourself". You will see the details as shown in the Figure 71. Here you can notice, on the left side the code is demonstrated and on the right side, the output of the code is shown.

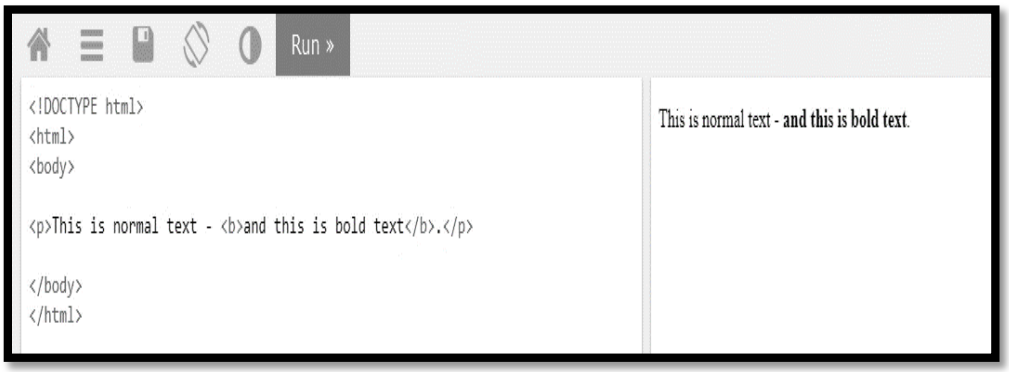


Figure 71: Code and output for <b> tag

Copyright Virtual University of Pakistan

Module 75

75. Networking and the Internet: XML

HTML is essentially a notational system by which a text document along with the document's appearance can be encoded as a simple text file. In a similar manner we can also encode non-textual material as text files—an example being sheet music. At first glance the pattern of staffs, measure bars, and notes in which music is traditionally represented does not conform to the character-by-character format dictated by text files. However, we can overcome this problem by developing an alternative notation system. More precisely, we could agree to represent the start of a staff by <staff clef = "treble">, the end of the staff by </staff>, a time signature with the form <time> 2/4 </time>, the beginning and ending of a measure by <measure> and </measure>, respectively, a note such as an eighth note on C as <notes> egth C </notes>, and so on. Then the text

<staff clef = "treble"> <key>C minor</key>

<time> 2/4 </time>

<measure> <rest> egth </rest> <notes> egth G, egth G, egth G </notes></measure>

<measure> <notes> hlf E </notes></measure>

</staff>

could be used to encode the music shown in Figure 72 using such notation, sheet music could be encoded, modified, stored, and transferred over the Internet as text files. Moreover, software could be written to present the contents of such files in the form of traditional sheet music or even to play the music on a synthesizer.
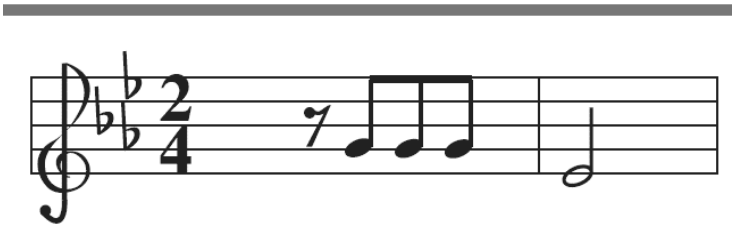


Figure 72: The first two bars of Beethoven's Fifth Symphony

Note that our sheet music encoding system encompasses the same style used by HTML. We chose to delineate the tags that identify components by the symbols < and >. We chose to indicate the beginning and end of structures (such as a staff, string of notes, or measure) by tags of the same name—the ending tag being designated by a slash (a <measure> was terminated with the tag </measure>). And we chose to indicate special attributes within tags by expressions such as clef = "treble". This same style could also be used to develop systems for representing other formats such as mathematical expressions and graphics.

The **eXtensible Markup Language (XML)** is a standardized style (similar to that of our music example) for designing notational systems for representing data as text files. (Actually, XML is a simplified derivative of an older set of standards called the Standard Generalized Markup Language, better known as SGML.) Following the XML standard, notational systems called **markup languages** have been developed for representing mathematics, multimedia presentations, and music. In fact, HTML is the markup language based on the XML standard that was developed for representing webpages. (Actually, the original version of HTML was developed before the XML standard was solidified, and therefore some features of HTML do not strictly conform to XML. That is why you might see references to XHTML, which is the version of

Copyright Virtual University of Pakistan

HTML that rigorously adheres to XML.) XML provides a good example of how standards are designed to have wide-ranging applications. Rather than designing individual, unrelated markup languages for encoding various types of documents, the approach represented by XML is to develop a standard for markup languages in general. With this standard, markup languages can be developed for various applications. Markup languages developed in this manner possess a uniformity that allows them to be combined to obtain markup languages for complex applications such as text documents that contain segments of sheet music and mathematical expressions.

Finally, we should note that XML allows the development of new markup languages that differ from HTML in that they emphasize semantics rather than appearance. For example, with HTML the ingredients in a recipe can be marked so that they appear as a list in which each ingredient is positioned on a separate line. But if we used semantic-oriented tags, ingredients in a recipe could be marked as ingredients (perhaps using the tags <ingredient> and </ingredient>) rather than merely items in a list. The difference is subtle but important. The semantic approach would allow **search engines** (websites that assist users in locating Web material pertaining to a subject of interest) to identify recipes that contain or do not contain certain ingredients, which would be a substantial improvement over the current state of the art in which only recipes that do or do not contain certain words can be isolated. More precisely, if semantic tags are used, a search engine can identify recipes for lasagna that do not contain spinach, whereas a similar search based merely on word content would skip over a recipe that started with the statement "This lasagna does not contain spinach." In turn, by using an Internet-wide standard for marking documents according to semantics rather than appearance, a World Wide *Semantic* Web, rather than the World Wide *Syntactic* Web we have today, would be created.

Copyright Virtual University of Pakistan

Module 76

76. Networking and the Internet: Client Side and Server Side

Consider now the steps that would be required for a browser to retrieve the simple webpage shown in Figure 69 and display it on the browser's computer screen. First, playing the role of a client, the browser would use the information in a URL (perhaps obtained from the person using the browser) to contact the webserver controlling access to the page and ask that a copy of the page be transferred to it. The server would respond by sending the text document displayed in Figure 69a to the browser. The browser would then interpret the HTML tags in the document to determine how the page should be displayed and present the document on its computer screen accordingly. The user of the browser would see an image like that depicted in Figure 69b. If the user then clicked the mouse over the word *here*, the browser would use the URL in the associated anchor tag to contact the appropriate server to obtain and display another webpage. In summary, the process consists of the browser merely fetching and displaying webpages as directed by the user.

But what if we wanted a webpage involving animation or one that allows a customer to fill out an order form and submit the order? These needs would require additional activity by either the browser or the webserver. Such activities are called **client-side** activities if they are performed by a client (such as a browser) or **server-side** activities if they are performed by a server (such as a webserver). As an example, suppose a travel agent wanted customers to be able to identify desired destinations and dates of travel, at which time the agent would present the customer with a customized webpage containing only the information pertinent to that customer's needs. In this case the travel agent's website would first provide a webpage that presents a customer with the available destinations. On the basis of this information, the customer would specify the destinations of interest and desired dates of travel (a client-side activity). This information would then be transferred back to the agent's server where it would be used to construct the appropriate customized webpage (a server-side activity), which would then be sent to the customer's browser.

Another example occurs when using the services of a search engine. In this case a user at the client specifies a topic of interest (a client-side activity), which is then transferred to the search engine where a customized webpage identifying documents of possible interest is constructed (a server-side activity) and sent back to the client. Another example occurs in the case of Web mail—an increasingly popular means by which computer users are able to access their email by means of Web browsers. In this case, the webserver is an intermediary between the client and the client's mail server. Essentially, the webserver builds webpages that contain information from the mail server (a server-side activity) and sends those pages to the client where the client's browser displays them (a client-side activity). Conversely, the browser allows the user to create messages (a client-side activity) and sends that information to the webserver, which then forwards the messages to the mail server (a server-side activity) for mailing.

There are numerous systems for performing client-side and server-side activities, each competing with the others for prominence. An early and still popular means of controlling client-side activities is to include programs written in the language JavaScript (developed by Netscape Communications, Inc.) within the HTML source document for the webpage. From there a browser can extract the programs and follow them as needed. Another approach (developed by Sun Microsystems) is to first transfer a webpage to a browser and then transfer additional program units called applets (written in the language Java) to the browser as requested within the HTML source document. Still another approach is the system Flash (developed by Macromedia) by which extensive multimedia client-side presentations can be implemented.

Copyright Virtual University of Pakistan

An early means of controlling server-side activities was to use a set of standards called CGI (Common Gateway Interface) by which clients could request the execution of programs stored at a server. A variation of this approach (developed by Sun Microsystems) is to allow clients to cause program units called servlets to be executed at the server side. A simplified version of the servlet approach is applicable when the requested server-side activity is the construction of a customized webpage, as in our travel agent example. In this case webpage templates called JavaServer Pages (JSP) are stored at the webserver and completed using information received from a client. A similar approach is used by Microsoft, where the templates from which customized webpages are constructed are called Active Server Pages (ASP). In contrast to these proprietary systems, PHP (originally standing for Personal Home Page but now considered to mean PHP Hypertext Preprocessor) is an open source system for implementing server-side functionality.

Finally, we would be remiss if we did not recognize the security and ethical problems that arise from allowing clients and servers to execute programs on the other's machine. The fact that webservers routinely transfer programs to clients where they are executed leads to ethical questions on the server side and security questions on the client side. If the client blindly executes any program sent to it by a webserver, it opens itself to malicious activities by the server. Likewise, the fact that clients can cause programs to be executed at the server leads to ethical questions on the client side and security questions on the server side. If the server blindly executes any program sent to it by a client, security breaches and potential damage at the server could result.

Copyright Virtual University of Pakistan

Module 77

77. Networking and the Internet: Layered Approach to Internet Software (I)

A principal task of networking software is to provide the infrastructure required for transferring messages from one machine to another. In the Internet, this message-passing activity is accomplished by means of a hierarchy of software units, which perform tasks analogous to those that would be performed if you were to send a gift in a package from the West Coast of the United States to a friend on the East Coast (Figure 73). You would first wrap the gift as a package and write the appropriate address on the outside of the package. Then, you would take the package to a shipping company such as the U.S. Postal Service. The shipping company might place the package along with others in a large container and deliver the container to an airline, whose services it has contracted. The airline would place the container in an aircraft and transfer it to the destination city, perhaps with intermediate stops along the way. At the final destination, the airline would remove the container from the aircraft and give it to the shipping company's office at the destination. In turn, the shipping company would take your package out of the container and deliver it to the addressee.

In short, the transportation of the gift would be carried out by a three-level hierarchy: (1) the user level (consisting of you and your friend), (2) the shipping company, and (3) the airline. Each level uses the next lower level as an abstract tool. (You are not concerned with the details of the shipping company, and the shipping company is not concerned with the internal operations of the airline.) Each level in the hierarchy has representatives at both the origin and the destination, with the representatives at the destination tending to do the reverse of their counterparts at the origin.

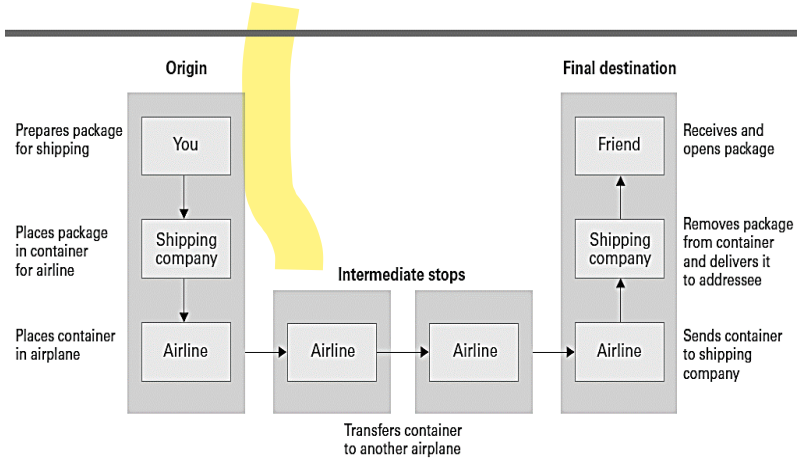


Figure 73: Package-shipping example

Such is the case with software for controlling communication over the Internet, except that the Internet software has four layers rather than three, each consisting of a collection of software routines rather than people and businesses. The four layers are known as the **application layer,** the **transport layer,** the **network layer,** and the **link layer** (Figure 74). A message typically originates in the application layer. From there it is passed down through the transport and network layers as it is prepared for transmission, and finally it is transmitted by the link layer. The message is received by the link layer at the destination and passed back up the hierarchy until it is delivered to the application layer at the message's destination.

Copyright Virtual University of Pakistan

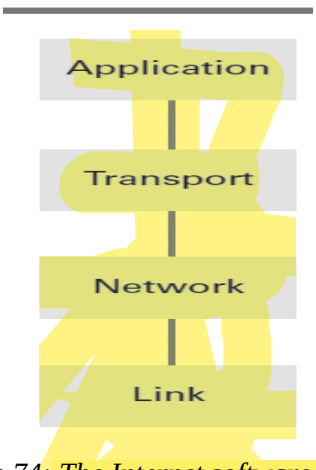


Figure 74: The Internet software layers

Copyright Virtual University of Pakistan

Module 78

78. Networking and the Internet: Layered Approach to Internet Software (II)

Let us investigate this process more thoroughly by tracing a message as it finds its way through the system (Figure 75). We begin our journey with the application layer.

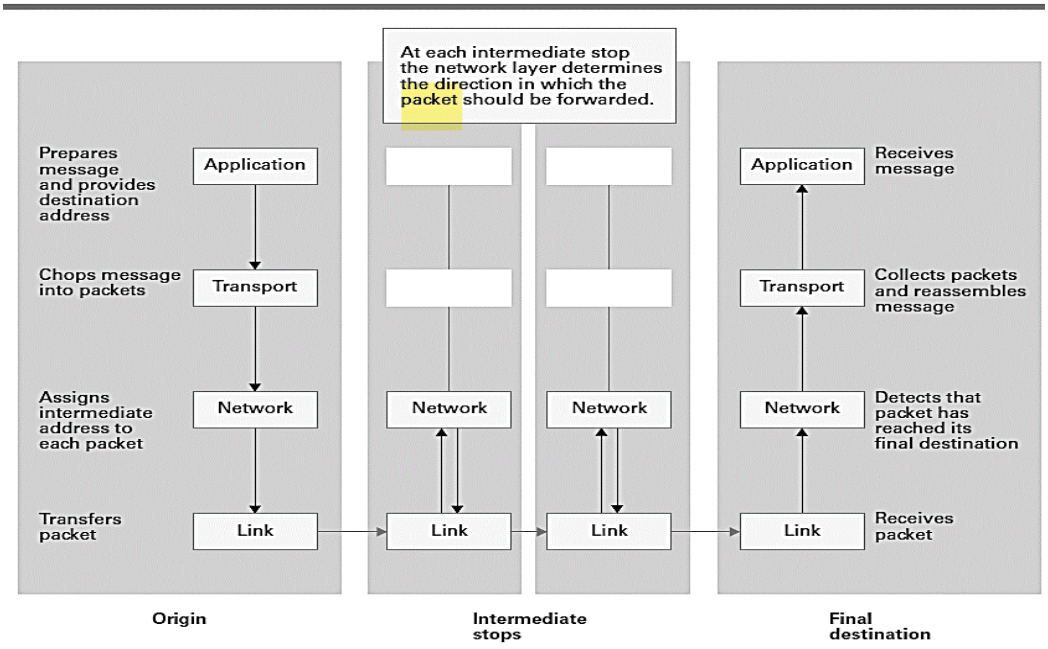


Figure 75: Following a message through the Internet

The application layer uses the transport layer to send and receive messages over the Internet in much the same way that you would use a shipping company to send and receive packages. Just as it is your responsibility to provide an address compatible with the specifications of the shipping company, it is the application layer's responsibility to provide an address that is compatible with the Internet infrastructure. To fulfill this need, the application layer may use the services of the name servers within the Internet to translate mnemonic addresses used by humans into Internet-compatible IP addresses. An important task of the transport layer is to accept messages from the application layer and to ensure that the messages are properly formatted for transmission over the Internet. Toward this latter goal, the transport layer divides long messages into small segments, which are transmitted over the Internet as individual units. This division is necessary because a single long message can obstruct the flow of other messages at the Internet routers where numerous messages cross paths. Indeed, small segments of messages can interweave at these points, whereas a long message forces other to wait while it passes (much like cars waiting for a long train to pass at a railroad crossing).

The transport layer adds sequence numbers to the small segments it produces so that the segments can be reassembled at the message's destination. Then it hands these segments, known as **packets,** to the network layer. From this point, the packets are treated as individual, unrelated messages until they reach the transport layer at their final destination. It is quite possible for the packets related to a common message to follow different paths through the Internet.

Copyright Virtual University of Pakistan

Module 79

79. Networking and the Internet: Layered Approach to Internet Software (III)

It is the network layer's job to decide in which direction a packet should be sent at each step along the packet's path through the Internet. In fact, the combination of the network layer and the link layer below it constitutes the software residing on the Internet routers. The network layer is in charge of maintaining the router's forwarding table and using that table to determine the direction in which to forward packets. The link layer at the router is in charge of receiving and transmitting the packets.

Thus, when the network layer at a packet's origin receives the packet from the transport layer, it uses its forwarding table to determine where the packet should be sent to get it started on its journey. Having determined the proper direction, the network layer hands the packet to the link layer for actual transmission. The link layer has the responsibility of transferring the packet. Thus, the link layer must deal with the communication details particular to the individual network in which the computer resides. For instance, if that network is an Ethernet, the link layer applies CSMA/CD. If the network is a WiFi network, the link layer applies CSMA/CA.

When a packet is transmitted, it is received by the link layer at the other end of the connection. There, the link layer hands the packet up to its network layer where the packet's final destination is compared to the network layer's forwarding table to determine the direction of the packet's next step. With this decision made, the network layer returns the packet to the link layer to be forwarded along its way. In this manner each packet hops from machine to machine on its way to its final destination.

Note that only the link and network layers are involved at the intermediate stops during this journey (see again Figure 4.14), and thus these are the only layers present on routers, as previously noted. Moreover, to minimize the delay at each of these intermediate "stops," the forwarding role of the network layer within a router is closely integrated with the link layer. In turn, the time required for a modern router to forward a packet is measured in millionths of a second.

At a packet's final destination, it is the network layer that recognizes that the packet's journey is complete. In that case, the network layer hands the packet to its transport layer rather than forwarding it. As the transport layer receives packets from the network layer, it extracts the underlying message segments and reconstructs the original message according to the sequence numbers that were provided by the transport layer at the message's origin. Once the message is assembled, the transport layer hands it to the appropriate unit within the application layer—thus completing the message transmission process.

Determining which unit within the application layer should receive an incoming message is an important task of the transport layer. This is handled by assigning unique **port numbers** (not related to the I/O ports discussed in Chapter 2) to the various units and requiring that the appropriate port number be appended to a message's address before starting the message on its journey. Then, once the message is received by the transport layer at the destination, the transport layer merely hands the message to the application layer software at the designated port number.

Users of the Internet rarely need to be concerned with port numbers because the common applications have universally accepted port numbers. For example, if a Web browser is asked to retrieve the document whose URL is:

http://www.zoo.org/animals/frog.html

The browser assumes that it should contact the HTTP server at www.zoo.org via port number 80. Likewise, when sending email, an SMTP client assumes that it should communicate with the SMTP mail server through port number 25.

Copyright Virtual University of Pakistan

In summary, communication over the Internet involves the interaction of four layers of software. The application layer deals with messages from the application's point of view. The transport layer converts these messages into segments that are compatible with the Internet and reassembles messages that are received before delivering them to the appropriate application. The network layer deals with directing the segments through the Internet. The link layer handles the actual transmission of segments from one machine to another. With all this activity, it is somewhat amazing that the response time of the Internet is measured in milliseconds, so that many transactions appear to take place instantaneously.

Copyright Virtual University of Pakistan

Module 80

80. Networking and the Internet: TCP /IP Protocol Suite

The demand for open networks has been generated a need for published standards by which manufacturers can supply equipment and software that function properly with products from other vendors. One standard that has resulted is the **Open System Interconnection (OSI)** reference model, produced by the International Organization for Standardization. This standard is based on a seventh-level hierarchy as opposed to the fourth-level hierarchy we have just described. It is an often-quoted model because it carries the authority of an international organization, but it has been shown to replace the fourth-level point of view, mainly because it was established after the fourth-level hierarchy had already become the de facto standard for the Internet.

The TCP/IP protocol suite is a collection of protocol standards used by the Internet to implement the four-level communication hierarchy. Actually, the **Transmission Control Protocol (TCP)** and the **Internet Protocol (IP)** are the names of only two of the protocols in this vast collection—so the fact that the entire collection is referred to as the TCP/IP protocol suite is rather misleading. More precisely, TCP defines a version of the transport layer. TCP(transmission control protocol)" and "UDP(user datagram protocol)" both are the version of transport layer. We say a version because the TCP/IP protocol suite provides for more than one way of implementing the transport layer; one of the other options is defined by the **User Datagram Protocol (UDP).** This diversity is analogous to the fact that when shipping a package, you have a choice of different shipping companies, each of which offers the same basic service but with its own unique characteristics. Thus, depending on the particular quality of service required, a unit within the application layer might choose to send data via a TCP or UDP version of the transport layer (Figure 76).

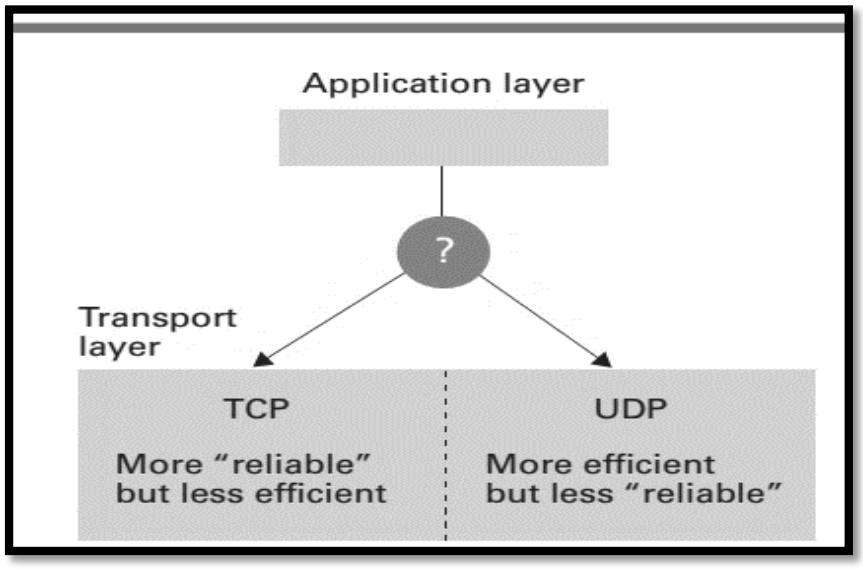


Figure 76: Choosing between TCP and UDP

There are several differences between TCP and UDP. One is that before sending a message as requested by the application layer, a transport layer based on TCP sends its own message to the transport layer at the destination telling it that a message is about to be sent. It then waits for this message to be acknowledged before starting to send the application layer's message. In this manner, a TCP transport layer is said to establish a connection before sending a message. A transport layer based on UDP does not establish such a connection prior to sending a message. It merely

Copyright Virtual University of Pakistan

sends the message to the address it was given and forgets about it. For all it knows, the destination computer might not even be operational. For this reason, UDP is called a connectionless protocol.

Another difference between TCP and UDP is that TCP transport layers at the origin and destination work together by means of acknowledgments and packet retransmissions to assure that all segments of a message are successfully transferred to the destination. For this reason, TCP is called a reliable protocol, whereas UDP, which does not offer such retransmission services, is said to be an unreliable protocol.

Another distinction between TCP and UDP is that TCP provides for both **flow control,** meaning that a TCP transport layer at a message's origin can reduce the rate at which it transmits segments to keep from overwhelming its counterpart at the destination, as well as **congestion control,** meaning that a TCP transport layer at a message's origin can adjust its transmission rate to alleviate congestion between it and the message's destination.

All this does not mean that UDP is a poor choice. After all, a transport layer based on UDP is more streamlined than a layer based on TCP, and thus if an application is prepared to handle the potential consequences of UDP, that option might be the better choice. For example, the efficiency of UDP makes it the protocol of choice for DNS lookups and VoIP. However, because email is less time sensitive, mail servers use TCP to transfer email.

IP is the Internet's standard for implementing the tasks assigned to the network layer. We have already observed that this task consists of **forwarding,** which involves relaying packets through the Internet, and **routing,** which involves updating the layer's forwarding table to reflect changing conditions. For instance, a router may malfunction, meaning that traffic should no longer be forwarded in its direction, or a section of the Internet may become congested, meaning that traffic should be routed around the blockage. Much of the IP standard associated with routing deals with the protocols used for communication among neighboring network layers as they interchange routing information.

An interesting feature associated with forwarding is that each time an IP network layer at a message's origin prepares a packet, it appends a value called a **hop count,** or time to live, to that packet. This value is a limit to the number of times the packet should be forwarded as it tries to find its way through the Internet. Each time an IP network layer forwards a packet, it decrements that packet's hop count by one. With this information, the network layer can protect the Internet from packets circling endlessly within the system. Although the Internet continues to grow on a daily basis, an initial hop count of 64 remains more than sufficient to allow a packet to find its way through the maze of routers within today's ISPs.

Copyright Virtual University of Pakistan

Module 81

81. Networking and the Internet: Security (Forms of Attacks)

There are numerous ways that a computer system and its contents can be attacked via network connections. Many of these incorporate the use of malicious software (collectively called **malware**). Such software might be transferred to, and executed on, the computer itself, or it might attack the computer from a distance. Examples of software that is transferred to, and executed on, the computer under attack include viruses, worms, Trojan horses, and spyware, whose names reflect the primary characteristics of the software.

A **virus** is software that infects a computer by inserting itself into programs that already reside in the machine. Then, when the "host" program is executed, the virus is also executed. When executed, many viruses do little more than try to transfer themselves to other programs within the computer. Some viruses, however, perform devastating actions such as degrading portions of the operating system, erasing large blocks of mass storage, or otherwise corrupting data and other programs.

A **worm** is an autonomous program that transfers itself through a network, taking up residence in computers and forwarding copies of itself to other computers. As in the case of a virus, a worm can be designed merely to replicate itself or to perform more extreme vandalism. A characteristic consequence of a worm is an explosion of the worm's replicated copies that degrades the performance of legitimate applications and can ultimately overload an entire network or internet.

A **Trojan horse** is a program that enters a computer system disguised as a desirable program, such as a game or useful utility package, that is willingly imported by the victim. Once in the computer, however, the Trojan horse performs additional activities that might have harmful effects. Sometimes these additional activities start immediately. In other instances, the Trojan horse might lie dormant until triggered by a specific event such as the occurrence of a preselected date. Trojan horses often arrive in the form of attachments to enticing email messages. When the attachment is opened (that is, when the recipient asks to view the attachment), the misdeeds of the Trojan horse are activated. Thus, email attachments from unknown sources should never be opened.

Another form of malicious software is **spyware** (sometimes called **sniffing** software), which is software that collects information about activities at the computer on which it resides and reports that information back to the instigator of the attack. Some companies use spyware as a means of building customer profiles, and in this context, it has questionable ethical merit. In other cases, spyware is used for blatantly malicious purposes such as recording the symbol sequences typed at the computer's keyboard in search of passwords or credit card numbers. As opposed to obtaining information secretly by sniffing via spyware, **phishing** is a technique of obtaining information explicitly by simply asking for it. The term *phishing* is a play on the word *fishing* because the process involved is to cast numerous "lines" in hopes that someone will "take the bait." Phishing is often carried out via email, and in this form, it is little more than an old telephone con. The perpetrator sends email messages posing as a financial institution, a government bureau, or perhaps a law enforcement agency. The email asks the potential victim for information that is supposedly needed for legitimate purposes. However, the information obtained is used by the perpetrator for hostile purposes.

In contrast to suffering from such internal infections as viruses and spyware, a computer in a network can also be attacked by software being executed on other computers in the system. An example is a **denial of service (DoS)** attack, which is the process of overloading a computer with messages. Denial of service attacks have been launched against large commercial web servers on

Copyright Virtual University of Pakistan

the Internet to disrupt the company's business and in some cases have brought the company's commercial activity to a halt.

A denial of service attack requires the generation of a large number of messages over a brief period of time. To accomplish this, an attacker usually plants software on numerous unsuspecting computers that will generate messages when a signal is given. Then, when the signal is given, all of these computers (sometimes called a **botnet**) swamp the target with messages. Inherent, then, in denial of service attacks is the availability of unsuspecting computers to use as accomplices. This is why all PC users are discouraged from leaving their computers connected to the Internet when not in use. It has been estimated that once a PC is connected to the Internet, at least one intruder will attempt to exploit its existence within 20 minutes. In turn, an unprotected PC represents a significant threat to the integrity of the Internet.

Another problem associated with an abundance of unwanted messages is the proliferation of unwanted junk email, called **spam.** However, unlike a denial of service attack, the volume of spam is rarely sufficient to overwhelm the computer system. Instead, the effect of spam is to overwhelm the person receiving the spam. This problem is compounded by the fact that, as we have already seen, spam is a widely adopted medium for phishing and instigating Trojan horses that might spread viruses and other detrimental software.

Copyright Virtual University of Pakistan

Module 82

82. Networking and the Internet: Protection and Cures

The old adage "an ounce of prevention is worth a pound of cure" is certainly true in the context of controlling vandalism over network connections. A primary prevention technique is to filter traffic passing through a point in the network, usually with a program called a **firewall.** For instance, a firewall might be installed at the gateway of an organization's intranet to filter messages passing in and out of the region. Such firewalls might be designed to block outgoing messages with certain destination addresses or to block incoming messages from origins that are known to be sources of trouble. This latter function is a tool for terminating a denial of service attack because it provides a means of blocking traffic from the attacking computers. Another common role of a firewall at a gateway is to block all incoming messages that have origin addresses within the region accessed through the gateway because such a message would indicate that an outsider is pretending to be a member of the inside region. Masquerading as a party other than one's self is known as **spoofing.**

Firewalls are also used to protect individual computers rather than entire networks or domains. For example, if a computer is not being used as a web server, a name server, or an email server, then a firewall should be installed at that computer to block all incoming traffic addressed to such applications. Indeed, one way an intruder might gain entry to a computer is by establishing contact through a "hole" left by a nonexistent server. In particular, one method for retrieving information gathered by spyware is to establish a clandestine server on the infected computer by which malicious clients can retrieve the spyware's findings. A properly installed firewall could block the messages from these malicious clients.

Some variations of firewalls are designed for specific purpose – an example being **spam filters,** which are firewalls designed to block unwanted email. Many spam filters use rather sophisticated techniques to distinguish between desirable email and spam. Some learn to make this distinction via a training process in which the user identifies items of spam until the filter acquires enough examples to make decisions on its own. These filters are examples of how a variety of subject areas (probability theory, artificial intelligence, etc.) can jointly contribute to developments in other fields.

Another preventative tool that has filtering connotations is the **proxy server.** A proxy server is a software unit that acts as an intermediary between a client and a server with the goal of shielding the client from adverse actions of the server. Without a proxy server, a client communicates directly with a server, meaning that the server has an opportunity to learn a certain amount about the client. Over time, as many clients within an organization's intranet deal with a distant server, that server can collect a multitude of information about the intranet's internal structure—information that can later be used for malicious activity. To counter this, an organization can establish a proxy server for a particular kind of service (FTP, HTTP, telnet, etc.). Then, each time a client within the intranet tries to contact a server of that type, the client is actually placed in contact with the proxy server. In turn, the proxy server, playing the role of a client, contacts the actual server. From then on the proxy server plays the role of an intermediary between the actual client and the actual server by relaying messages back and forth. The first advantage of this arrangement is that the actual server has no way of knowing that the proxy server is not the true client, and in fact, it is never aware of the actual client's existence. In turn, the actual server has no way of learning about the intranet's internal features. The second advantage is that the proxy server is in position to filter all the messages sent from the server to the client. For example, an FTP proxy server could check all incoming files for the presence of known viruses and block all infected files.

Another tool for preventing problems in a network environment is auditing software. Using network auditing software, a system administrator can detect a sudden increase in message traffic at various

Copyright Virtual University of Pakistan

locations within the administrator's realm, monitor the activities of the system's firewalls, and analyze the pattern of requests being made by the individual computers in order to detect irregularities. In effect, auditing software is an administrator's primary tool for identifying problems before they grow out of control.

Another means of defense against invasions via network connections is software called **antivirus software,** which is used to detect and remove the presence of known viruses and other infections. (Actually, antivirus software represents a broad class of software products, each designed to detect and remove a specific type of infection. For example, whereas many products specialize in virus control, others specialize in spyware protection.) It is important for users of these packages to understand that, just as in the case of biological systems, new computer infections are constantly coming on the scene that require updated vaccines. Thus, antivirus software must be routinely maintained by downloading updates from the software's vendor. Even this, however, does not guarantee the safety of a computer. After all, a new virus must first infect some computers before it is discovered, and a vaccine is produced. Thus, a wise computer user never opens email attachments from unfamiliar sources, does not download software without first confirming its reliability, does not respond to pop-up adds, and does not leave a PC connected to the Internet when such connection is not necessary.

Copyright Virtual University of Pakistan

Module 83

83. Networking and the Internet: Encryption

In some cases, the purpose of network vandalism is to disrupt the system (as in denial of service attacks), but in other cases the ultimate goal is to gain access to information. The traditional means of protecting information is to control its access through the use of passwords. However, passwords can be compromised and are of little value when data are transferred over networks and internet where messages are relayed by unknown entities. In these cases, encryption can be used so that even if the data fall into unscrupulous hands, the encoded information will remain confidential. Today, many traditional Internet applications have been altered to incorporate encryption techniques, producing what are called "secure versions" of the applications.

A prime example is the secure version of HTTP, known as **HTTPS,** which is used by most financial institutions to provide customers with secure Internet access to their accounts. The backbone of HTTPS is the protocol system known as **Secure Sockets Layer (SSL),** which was originally developed by Netscape to provide secure communication links between Web clients and servers. Most browsers indicate the use of SSL by displaying a tiny padlock icon on the computer screen. (Some use the presence or absence of the icon to indicate whether SSL is being used; others display the padlock in either the locked or unlocked position.)

One of the more fascinating topics in the field of encryption is **public-key encryption,** which involves techniques by which encryption systems are designed so that having knowledge about how messages are encrypted does not allow one to decrypt messages. This characteristic is somewhat counterintuitive. After all, intuition would suggest that if a person knows how messages are encrypted, then that person should be able to reverse the encryption process and thus decrypt messages. But public-key encryption systems defy this intuitive logic.

A public-key encryption system involves the use of two values called keys. One key, known as the **public key,** is used to encrypt messages; the other key, known as the **private key,** is required to decrypt messages. To use the system, the public key is first distributed to those who might need to send messages to a particular destination. The private key is held in confidence at this destination. Then, the originator of a message can encrypt the message using the public key and send the message to its destination with assurance that its contents are safe, even if it is handled by intermediaries who also know the public key. Indeed, the only party that can decrypt the message is the party at the message's destination who holds the private key. Thus if Bob creates a public-key encryption system and gives both Alice and Carol the public key, then both Alice and Carol can encrypt messages to Bob, but they cannot spy on the other's communication. Indeed, if Carol intercepts a message from Alice, she cannot decrypt it even though she knows how Alice encrypted it (Figure 77).

There are, of course, subtle problems lurking within public-key systems. One is to ensure that the public key being used is, in fact, the proper key for the destination party. For example, if you are communicating with your bank, you want to be sure that the public key you are using for encryption is the one for the bank and not an impostor. If an impostor presents itself as the bank (an example of spoofing) and gives you its public key, the messages you encrypt and send to the "bank" would be meaningful to the impostor and not your bank. Thus, the task of associating public keys with correct parties is significant.

One approach to resolving this problem is to establish trusted Internet sites, called **certificate authorities,** whose task is to maintain accurate lists of parties and their public keys. These authorities, acting as servers, then provide reliable public-key information to their clients in

Copyright Virtual University of Pakistan

packages known as **certificates.** A certificate is a package containing a party's name and that party's public key.

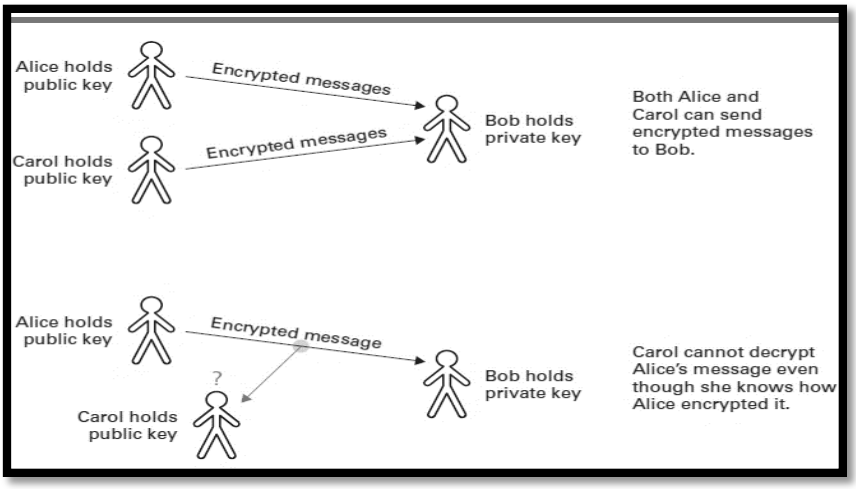


Figure 77: Public key encryption

Many commercial certificate authorities are now available on the Internet, although it is also common for organizations to maintain their own certificate authorities in order to maintain tighter control over the security of the organization's communication.

Finally, we should comment on the role public-key encryption systems play in solving problems of **authentication**—making sure that the author of a message is, in fact, the party it claims to be. The critical point here is that, in some public-key encryption systems, the roles of the encryption and decryption keys can be reversed. That is, text can be encrypted with the private key, and because only one party has access to that key, any text that is so encrypted must have originated from that party. In this manner, the holder of the private key can produce a bit pattern, called a **digital signature,** that only that party knows how to produce. By attaching that signature to a message, the sender can mark the message as being authentic. A digital signature can be as simple as the encrypted version of the message itself. All the sender must do is encrypt the message being transmitted using his or her private key (the key typically used for decrypting). When the message is received, the receiver uses the sender's public key to decrypt the signature. The message that is revealed is guaranteed to be authentic because only the holder of the private key could have produced the encrypted version.

Copyright Virtual University of Pakistan

Module 84

84. Networking and the Internet: Legal Approaches to Network Security

- ✓ Enhancing the security of computer networking systems is to apply legal remedies.
- ✓ Defining what is legal and what is illegal in different contexts.
- ✓ Illegal actions may lead to punishment.

Two Issues

- ✓ Making an action illegal does not stop the action.
- ✓ International law, Illegal in one country and legal in another country.

US – Computer Fraud and Abuse Act

- ✓ Passed in 1984 and has been revised many times.
- ✓ Covers:
  - Worms and Viruses
  - Theft of Information
  - Many more related things are covered anything of value.

US – Electronic Communication Privacy Act (ECPA)

- ✓ Employers/ISP rights to look into the communications of employees.
- ✓ Illegal for ISP to make available the private information
- ✓ Employers can access information especially when the equipment of employer is used?

Pakistan – CyberSecurity Law

- ✓ National Assembly Passes New Cybercrime Law. In 2016.
  - ✓ Punishment of 10 years for first offenders and 20 years for repeat offenders. Details can be found at: http://thehill.com/policy/cybersecurity/265285-judges-struggle-with-cyber-crime-punishment
  - ✓ Covers many security concerns e.g.,
    - Hacking
    - Phishing
    - Malware
    - Hacking tools
    - Identity Theft/fraud.
    - Electronic theft.

Pakistan – Applicable security Acts

Copyright Virtual University of Pakistan

- ✓ Prevention of Electronic Crimes Ordinance, 2007
- ✓ Electronic Transaction Ordinance, 2002, 2008
- ✓ Pakistan Telecommunication (Re-organization) Act, 1996
- ✓ Wireless telegraphy Act, 1933
- ✓ Telegraph Act, 1885.
- ✓ Federal Investigation Agency Act, 1974
- ✓ Payments and Electronic Fund Transfers Act, 2007

Copyright Virtual University of Pakistan

Module 85

85. Algorithm: an Informal Review

Algorithm

- ✓ Set of steps that define how a task is performed.
- ✓ Let's start discussing simple Algorithms

Making a Cup of Tea

1. Put water in kettle
2. Boil water
3. Put Tea in water
4. Add Milk
5. Stir
6. Add Sugar
7. Put tea in cup
8. Fetch cup

Convert KM to Meters

1. Take input of KM's
2. Multiply the input with 1000.
3. Display the result

Machine Cycle

As long as the halt instruction has not been executed continue to execute the following steps:

a. Fetch an instruction.

b. Decode the instruction.

c. Execute the instruction.

Algorithm Execution

Many researchers believe that every activity of the human mind, including imagination, creativity, and decision making, is actually the result of algorithm execution.

Copyright Virtual University of Pakistan

Module 86

86. Algorithm: Formal Definition of Algorithm

Formal Definition

An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process.

Order vs sequence

- ✓ Have a well-established structure in terms of the order of their execution
- ✓ Does not mean a sequence – like we do not have in parallel processing.
- ✓ Flip-flops producing their output individually and then together they have a meaning.

Executable

- ✓ Make a list of all the positive integers.
- ✓ Computer scientists use the term effective to capture the concept of being executable.

Unambiguous

The information in the state of the process must be sufficient to determine uniquely and completely the actions required by each step.

Make a pretty cartoon!

Terminating Process

The execution of an algorithm must lead to an end.

But!

- ✓ Monitoring the vital signs of a hospital patient
- ✓ Maintaining an aircraft's altitude in flight.

Copyright Virtual University of Pakistan

Module 87

87. Algorithm: Abstract Nature of Algorithms

It is important to emphasize the distinction between an algorithm and its representation—a distinction that is analogous to that between a story and a book. A story is abstract, or conceptual, in nature; a book is a physical representation of a story. If a book is translated into another language or republished in a different format, it is merely the representation of the story that changes—the story itself remains the same.

In the same manner, an algorithm is abstract and distinct from its representation. A single algorithm can be represented in many ways. As an example, the algorithm for converting temperature readings from Celsius to Fahrenheit is traditionally represented as the algebraic formula

F = (9/5)C + 32

But it could be represented by the instruction

Multiply the temperature reading in Celsius by 9⁄5

and then add 32 to the product

or even in the form of an electronic circuit. In each case the underlying algorithm is the same; only the representations differ.

The distinction between an algorithm and its representation presents a problem when we try to communicate algorithms. A common example involves the level of detail at which an algorithm must be described. Among meteorologists, the instruction "Convert the Celsius reading to its Fahrenheit equivalent" suffices, but a layperson, requiring a more detailed description, might argue that the instruction is ambiguous. The problem, however, is not with the underlying algorithm but that the algorithm is not represented in enough detail for the layperson. In the next section we will see how the concept of primitives can be used to eliminate such ambiguity problems in an algorithm's representation.

Finally, while on the subject of algorithms and their representations, we should clarify the distinction between two other related concepts—programs and processes. A *program* is a representation of an algorithm. (Here we are using the term *algorithm* in its less formal sense in that many programs are representations of nonterminating "algorithms.") In fact, within the computing community the term *program* usually refers to a formal representation of an algorithm designed for computer application. We defined a *process* already to be the activity of executing a program. Note, however, that to execute a program is to execute the algorithm represented by the program, so a process could equivalently be defined as the activity of executing an algorithm. We conclude that programs, algorithms, and processes are distinct, yet related, entities. A program is the representation of an algorithm, whereas a process is the activity of executing an algorithm.

Copyright Virtual University of Pakistan

Module 88

88. Algorithm: Representation (Primitives)

The representation of an algorithm requires some form of language. In the case of humans, this might be a traditional natural language (English, Spanish, Russian, Japanese) or perhaps the language of pictures, as demonstrated in Figure 78, which describes an algorithm for folding a bird from a square piece of paper. Often, however, such natural channels of communication lead to misunderstandings, sometimes because the terminology used has more than one meaning. (The sentence, "Visiting grandchildren can be nerve-racking," could mean either that the grandchildren cause problems when they come to visit or that going to see them is problematic.) Problems also arise over misunderstandings regarding the level of detail required. Few readers could successfully fold a bird from the directions given in Figure 78, yet a student of origami would probably have little difficulty. In short, communication problems arise when the language used for an algorithm's representation is not precisely defined or when information is not given in adequate detail.

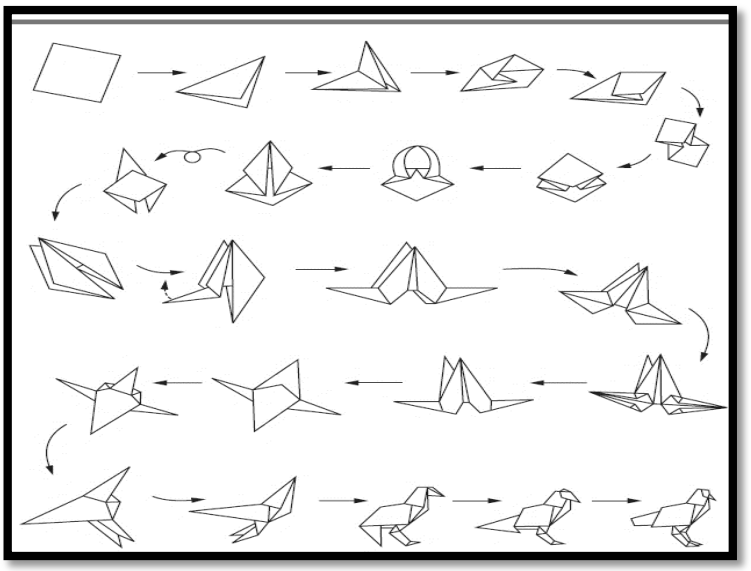


Figure 78: Folding a bird from a square piece of paper

Computer science approaches these problems by establishing a well-defined set of building blocks from which algorithm representations can be constructed. Such a building block is called a **primitive.** Assigning precise definitions to these primitives removes many problems of ambiguity and requiring algorithms to be described in terms of these primitives establishes a uniform level of detail. A collection of primitives along with a collection of rules stating how the primitives can be combined to represent more complex ideas constitutes a **programming language.** Each primitive has its own syntax and semantics. Syntax refers to the primitive's symbolic representation; semantics refers to the meaning of the primitive. The syntax of *air* consists of three symbols, whereas the semantics is a gaseous substance that surrounds the world. As an example, Figure 79 presents some of the primitives used in origami.

Copyright Virtual University of Pakistan

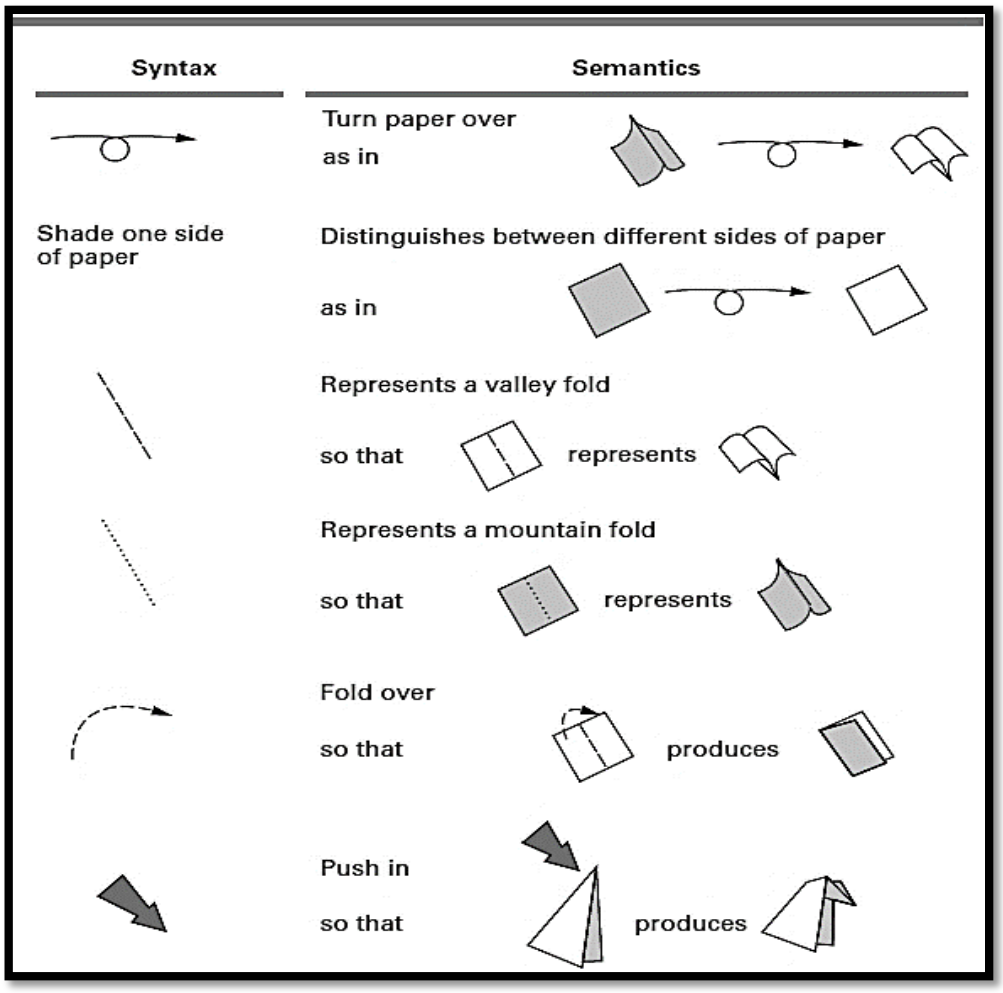


Figure 79: Origami primitives

To obtain a collection of primitives to use in representing algorithms for computer execution, we could turn to the individual instructions that the machine is designed to execute. If an algorithm is expressed at this level of detail, we will certainly have a program suitable for machine execution. However, expressing algorithms at this level is tedious, and so one normally uses a collection of "higher-level" primitives, each being an abstract tool constructed from the lower-level primitives provided in the machine's language. The result is a formal programming language in which algorithms can be expressed at a conceptually higher level than in machine language. We will discuss such programming languages in the next chapter

Copyright Virtual University of Pakistan

Module 89

89. Algorithm: Representation (Pseudocode)

For now, we forgo the introduction of a formal programming language in favor of a less formal, more intuitive notational system known as pseudocode. In general, a pseudocode is a notational system in which ideas can be expressed informally during the algorithm development process.

One way to obtain a pseudocode is simply to loosen the rules of a formal programming language, borrowing the syntax-semantic structures of the language, intermixed with less formal constructs. There are many such pseudocode variants, because there are many programming languages in existence. Two particularly popular choices are loose versions of the languages Algol and Pascal, largely because these were widely used in textbooks and academic papers for decades. More recently, pseudocode reminiscent of the Java and C languages has proliferated, again because most programmers will have at least a reading knowledge of these languages. Regardless of where a pseudocode borrows its syntax from, one essential property is required for it to serve its purpose in expressing algorithms: A pseudocode must have a consistent, concise notation for representing recurring semantic structures.

One semantic structure is the saving of a computed value. For example, if we have computed the sum of our checking and savings account balances, we may want to save the result so we can refer to it later. In such cases we will use the form

name = expression

where name is the name by which we will refer to the result and expression describes the computation whose result is to be saved. This pseudocode structure directly follows the equivalent Python assignment statement, which we introduced in Chapter 1 for storing a value into a Python variable. For example, the statement

RemainingFunds = CheckingBalance + SavingsBalance

is an assignment statement that assigns the sum of CheckingBalance and SavingsBalance to the name RemainingFunds? Thus, the term RemainingFunds can be used in future statements to refer to that sum.

Another recurring semantic structure is the selection of one of two possible activities depending on the truth or falseness of some condition. Examples include:

If the gross domestic product has increased, buy common stock otherwise sell common stock.

Buy common stock if the gross domestic product has increased and sell it otherwise.

Buy or sell common stock depending on whether the gross domestic product has increased or decreased, respectively.

Each of these statements could be rewritten to conform to the structure

if (condition):

activity

else:

activity

where we have used the keywords if and else to announce the different sub- structures within the main structure and have used colons and indentation to delineate the boundaries of these substructures. The condition and the else will always be followed immediately by a colon. The corresponding activity will be indented. If an activity consists of multiple steps, they will all be similarly indented. By adopting this

Copyright Virtual University of Pakistan

syntactic structure for our pseudocode, we acquire a uniform way in which to express this common semantic structure. Thus, whereas the statement

Depending on whether the year is a leap year, divide the total by 366 or 365, respectively might possess a more creative literary style, we will consistently opt for the straightforward condition:

if (year is leap year): daily total = total / 366

else:

daily total = total / 365

Copyright Virtual University of Pakistan

Module 90

90. Algorithm: Representation (Pseudocode) While-Structure

Another common semantic structure is the repeated execution of a statement or sequence of statements as long as some condition remains true. Informal examples include:

As long as there are tickets to sell, continue selling tickets.

And

While there are tickets to sell, keep selling tickets.

For such cases, we adopt the uniform pattern

while (condition):

Activity

For our pseudocode. In short, such a statement means to check the condition and, if it is true, perform the activity and return to check the condition again. If, however, the condition is found to be false, move on to the next instruction following the while structure. Thus, both of the preceding statements are reduced to:

while (tickets remain to be sold):

sell a ticket

Copyright Virtual University of Pakistan

Module 91

91. Algorithm: Representation (Pseudocode) Function-Structure

We want to use our pseudocode to describe activities that can be used as abstract tools in other applications. Computer science has a variety of terms for such program units, including subprogram, subroutine, procedure, method, and function, each with its own variation of meaning. We will follow Python convention, using the term **function** for our pseudocode and using the Python keyword def to announce the title by which the pseudocode unit will be known. More precisely, we will begin a pseudocode unit with a statement of the form

def name():

where name is the particular name of the unit. We will then follow this introductory statement with the statements that define the unit's action. For example, Figure 80 is a pseudocode representation of a function called Greetings that prints the message "Hello" three times.

```
def Greetings():
    Count = 3
    while (Count > 0):
        print('Hello')
        Count = Count - 1
```

Figure 80: The function Greetings in pseudocode

When the task performed by a function is required elsewhere in our pseudo code, we will merely request it by name. For example, if two functions were named **ProcessLoan** and **RejectApplication**, then we could request their services within an if-else structure by writing

if (. . .):

ProcessLoan()

else:

RejectApplication()

which would result in the execution of the function **ProcessLoan** if the tested condition were true or in the execution of **RejectApplication** if the condition were false.

Copyright Virtual University of Pakistan

Module 92

92. Algorithm: Discovery (The Art of Problem Solving)

The techniques of problem solving and the need to learn more about them are not unique to computer science but rather are topics pertinent to almost any field. The close association between the process of algorithm discovery and that of general problem solving has caused computer scientists to join with those of other disciplines in the search for better problem-solving techniques. Ultimately, one would like to reduce the process of problem solving to an algorithm in itself, but this has been shown to be impossible. Thus the ability to solve problems remains more of an artistic skill to be developed than a precise science to be learned.

As evidence of the elusive, artistic nature of problem solving, the following loosely defined problem-solving phases presented by the mathematician G. Polya in 1945 remain the basic principles on which many attempts to teach problem-solving skills are based today.

Phase 1. **Understand the problem.**

Phase 2. **Devise a plan for solving the problem.**

Phase 3. **Carry out the plan.**

Phase 4. **Evaluate the solution for accuracy and for its potential as a tool for solving other problems.**

Translated into the context of program development, these phases become

Phase 1. **Understand the problem.**

Phase 2. **Get an idea of how an algorithmic function might solve the problem.**

Phase 3. **Formulate the algorithm and represent it as a program.**

Phase 4. **Evaluate the program for accuracy and for its potential as a tool for solving other problems.**

As an example, consider the following problem:

Person A is charged with the task of determining the ages of person B's three children. B tells A that the product of the children's ages is 36. After considering this clue, A replies that another clue is required, so B tells A the sum of the children's ages. Again, A replies that another clue is needed, so B tells A that the oldest child plays the piano. After hearing this clue, A tells B the ages of the three children.

How old are the three children?

At first glance the last clue seems to be totally unrelated to the problem, yet it is apparently this clue that allows A to finally determine the ages of the children. How can this be? Let us proceed by formulating a plan of attack and following this plan, even though we still have many questions about the problem. Our plan will be to trace the steps described by the problem statement while keeping track of the information available to person A as the story progresses.

The first clue given A is that the product of the children's ages is 36. This means that the triple representing the three ages is one of those listed in Figure 81(a). The next clue is the sum of the desired triple. We are not told what this sum is but we are told that this information is not enough for A to isolate the correct triple; therefore the desired triple must be one whose sum appears at least twice in the table of Figure 81(b). But the only triples appearing in Figure 81(b) with identical sums are (1,6,6) and (2,2,9), both of which produce the sum 13. This is the information available to A at the time the last clue is given. It is at this point that we finally understand the significance of the last clue. It has nothing to do with playing the piano; rather

Copyright Virtual University of Pakistan

it is the fact that there is an oldest child. This rules out the triple (1,6,6) and thus allows us to conclude that the children's ages are 2, 2, and 9.

In this case, then, it is not until we attempt to implement our plan for solving the problem (Phase 3) that we gain a complete understanding of the problem (Phase 1). Had we insisted on completing Phase 1 before proceeding, we would probably never have found the children's ages. Such irregularities in the problem-solving process are fundamental to the difficulties in developing systematic approaches to problem solving.

a. Triples whose product is 36

(1,1,36)	(1,6,6)
(1,2,18)	(2,2,9)
(1,3,12)	(2,3,6)
(1,4,9)	(3,3,4)

b. Sums of triples from part (a)

1 + 1 + 36 = 38	1 + 6 + 6 = 13
1 + 2 + 18 = 21	2 + 2 + 9 = 13
1 + 3 + 12 = 16	2 + 3 + 6 = 11
1 + 4 + 9 = 14	3 + 3 + 4 = 10

Figure 81: Analyzing the possibilities

watch lms lecture for understanding only

Copyright Virtual University of Pakistan

Module 93

93. Algorithm: Getting your Foot in the Door

We have been discussing problem solving from a somewhat philosophical point of view while avoiding a direct confrontation with the question of how we should go about trying to solve a problem. There are, of course, numerous problem-solving approaches, each of which can be successful in certain settings. We will identify some of them shortly. For now, we note that there seems to be a common thread running through these techniques, which simply stated is "get your foot in the door." As an example, let us consider the following simple problem:

Before A, B, C, and D ran a race they made the following predictions:

A predicted that B would win.

B predicted that D would be last.

C predicted that A would be third

D predicted that A's prediction would be correct.

Only one of these predictions was true, and this was the prediction made by the winner. In what order did A, B, C, and D finish the race?

After reading the problem and analyzing the data, it should not take long to realize that since the predictions of A and D were equivalent and only one prediction was true, the predictions of both A and D must be false. Thus, neither A nor D were winners. At this point we have our foot in the door and obtaining the complete solution to our problem is merely a matter of extending our knowledge from here. If A's prediction was false, then B did not win either. The only remaining choice for the winner is C. Thus, C won the race, and C's prediction was true. Consequently, we know that A came in third. That means that the finishing order was either CBAD or CDAB. But the former is ruled out because B's prediction must be false. Therefore, the finishing order was CDAB.

only watch lms lecture for understanding

Copyright Virtual University of Pakistan

Module 94

94. Algorithm: Algorithm Discovery Strategies (I)

Of course, being told to get our foot in the door is not the same as being told how to do it. Obtaining this toehold, as well as realizing how to expand this initial thrust into a complete solution to the problem, requires creative input from the problem solver. There are, however, several general approaches that have been proposed by Polya and others for how one might go about getting a foot in the door. One is to try working the problem backward. For instance, if the problem is to find a way of producing a particular output from a given input, one might start with that output and attempt to back up to the given input. This approach is typical of people trying to discover the bird-folding algorithm in the previous section. They tend to unfold a completed bird in an attempt to see how it is constructed.

Another general problem-solving approach is to look for a related problem that is either easier to solve or has been solved before and then try to apply its solution to the current problem. This technique is of particular value in the context of program development. Generally, program development is not the process of solving a particular instance of a problem but rather of finding a general algorithm that can be used to solve all instances of the problem. More precisely, if we were faced with the task of developing a program for alphabetizing lists of names, our task would not be to sort a particular list but to find a general algorithm that could be used to sort any list of names. Thus, although the instructions

1. Interchange the names David and Alice.
2. Move the name Carol to the position between Alice and David.
3. Move the name Bob to the position between Alice and Carol.

correctly sort the list David, Alice, Carol, and Bob, they do not constitute the general-purpose algorithm we desire. What we need is an algorithm that can sort this list as well as other lists we might encounter. This is not to say that our solution for sorting a particular list is totally worthless in our search for a general-purpose algorithm. We might, for instance, get our foot in the door by considering such special cases in an attempt to find general principles that can in turn be used to develop the desired general-purpose algorithm. In this case, then, our solution is obtained by the technique of solving a collection of related problems.

only lms lecture

Copyright Virtual University of Pakistan

Module 95

95. Algorithm: Algorithm Discovery Strategies (II)

Another approach to getting a foot in the door is to apply **stepwise refinement,** which is essentially the technique of not trying to conquer an entire task (in all its detail) at once. Rather, stepwise refinement proposes that one first view the problem at hand in terms of several subproblems. The idea is that by breaking the original problem into subproblems, one is able to approach the overall solution in terms of steps, each of which is easier to solve than the entire original problem. In turn, stepwise refinement proposes that these steps be decomposed into smaller steps and these smaller steps be broken into still smaller ones until the entire problem has been reduced to a collection of easily solved subproblems.

In this light, stepwise refinement is a **top-down methodology** in that it progresses from the general to the specific. In contrast, a **bottom-up methodology** progresses from the specific to the general. Although contrasting in theory, the two approaches often complement each other in creative problem solving. The decomposition of a problem proposed by the top-down methodology of stepwise refinement is often guided by the problem solver's intuition, which might be working in a bottom-up mode.

Copyright Virtual University of Pakistan

Module 96

96. Algorithm: Iterative Structures (Sequential Search Algorithm)

Consider the problem of searching within a list for the occurrence of a particular target value. We want to develop an algorithm that determines whether that value is in the list. If the value is in the list, we consider the search a success; otherwise we consider it a failure. We assume that the list is sorted according to some rule for ordering its entries. For example, if the list is a list of names, we assume the names appear in alphabetical order, or if the list consists of numeric values, we assume its entries appear in order of increasing magnitude.

To get our foot in the door, we imagine how we might search a guest list of perhaps 20 entries for a particular name. In this setting we might scan the list from its beginning, comparing each entry with the target name. If we find the target name, the search terminates as a success. However, if we reach the end of the list without finding the target value, our search terminates as a failure. In fact, if we reach a name greater than (alphabetically) the target name without finding the target, our search terminates as a failure. (Remember, the list is arranged in alphabetical order, so reaching a name greater than the target name indicates that the target does not appear in the list.) In summary, our rough idea is to continue searching down the list as long as there are more names to be investigated and the target name is greater than the name currently being considered.

In our pseudocode, this process can be represented as

Select the first entry in the list as TestEntry. while (TargetValue > TestEntry and entries remain):

Select the next entry in the list as TestEntry

Upon terminating this while structure, one of two conditions will be true; Either the target value has been found or the target value is not in the list. In either case we can detect a successful search by comparing the test entry to the target value. If they are equal, the search has been successful. Thus, we add the statement

if (TargetValue == TestEntry):

Declare the search a success.

else:

Declare the search a failure.

to the end of our pseudocode routine.

Finally, we observe that the first statement in our routine, which selects the first entry in the list as the test entry, is based on the assumption that the list in question contains at least one entry. We might reason that this is a safe guess, but just to be sure, we can position our routine as the else option of the statement

if (List is empty):

Declare search a failure. else:

. . .

This produces the function shown in Figure 82. Note that this function can be used from within other functions by using statements such as

Search() the passenger list using Darrel Baker as the target value.

to find out if Darrel Baker is a passenger and

Copyright Virtual University of Pakistan

Search() the list of ingredients using nutmeg as the target value. to find out if nutmeg appears in the list of ingredients.

```
def Search(List, TargetValue):
  if (List is empty):
    Declare search a failure.
  else:
    Select the first entry in List to be TestEntry.
    while (TargetValue > TestEntry and
        there remain entries to be considered):
      Select the next entry in List as TestEntry.
    if (TargetValue == TestEntry):
      Declare search a success.
    else:
      Declare search a failure.
```

Figure 82: The sequential search algorithm in pseudocode

Copyright Virtual University of Pakistan

Module 97

97. Algorithm: Iterative Structures (Loop Control)

The repetitive use of an instruction or sequence of instructions is an important algorithmic concept. One method of implementing such repetition is the iterative structure known as the **loop,** in which a collection of instructions, called the **body** of the loop, is executed in a repetitive fashion under the direction of some control process. A typical example is found in the sequential search algorithm represented in Figure 82. Here we use a while statement to control the repetition of the single statement **Select the next entry in List as the TestEntry**. Indeed, the while statement.

while (condition):

Body

exemplifies the concept of a loop structure in that its execution traces the cyclic pattern

check the condition.

execute the body.

check the condition.

execute the body.

.

.

.

check the condition.

until the condition fails.

As a general rule, the use of a loop structure produces a higher degree of flexibility that would be obtained merely by explicitly writing the body several times. For example, to execute the statement

Add a drop of sulfuric acid, three times, we could write:

Add a drop of sulfuric acid. Add a drop of sulfuric acid. Add a drop of sulfuric acid.

But we cannot produce a similar sequence that is equivalent to the loop structure.

while (the pH level is greater than 4):

add a drop of sulfuric acid

because we do not know in advance how many drops of acid will be required.

Let us now take a closer look at the composition of loop control. You might be tempted to view this part of a loop structure as having minor importance. After all, it is typically the body of the loop that actually performs the task at hand (for example, adding drops of acid)—the control activities appear merely as the overhead involved because we chose to execute the body in a repetitive fashion. However, experience has shown that the control of a loop is the more error-prone part of the structure and therefore deserves our attention.

Copyright Virtual University of Pakistan

Module 98

98. Algorithm: Iterative Structures (Components of Repetitive Control)

The control of a loop consists of the three activities initialize, test, and modify (Figure 83), with the presence of each being required for successful loop control. The test activity has the obligation of causing the termination of the looping process by watching for a condition that indicates termination should take place. This condition is known as the **termination condition.** It is for the purpose of this test activity that we provide a condition within each while statement of our pseudocode. In the case of the while statement, however, the condition stated is the condition under which the body of the loop should be executed—the termination condition is the negation of the condition appearing in the while structure.

Thus, in the statement

while (the pH level is greater than 4): add a drop of sulfuric acid

the termination condition is "the pH level is *not* greater than 4," and in the while

statement of Figure 82, the termination condition could be stated as

(TargetValue <= TestEntry) or (there are no more entries to be considered)

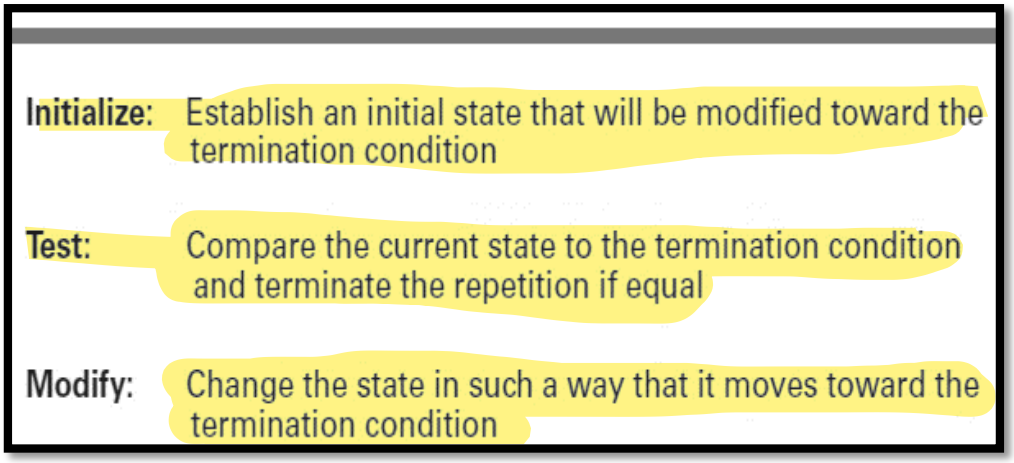


Figure 83: Components of repetitive control

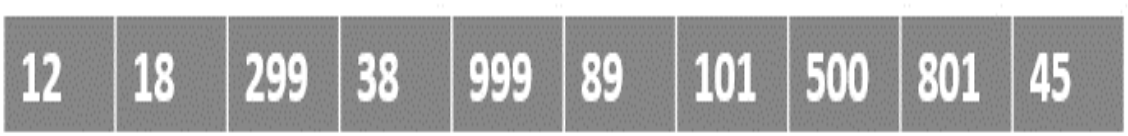
The other two activities in the loop control ensure that the termination condition will ultimately occur. The initialization step establishes a starting condition, and the modification step moves this condition toward the termination condition. For instance, in Figure 82, initialization takes place in the statement preceding the while statement, where the current test entry is established as the first list entry. The modification step in this case is actually accomplished within the loop body, where our position of interest (identified by the test entry) is moved towards the end of the list. Thus, having executed the initialization step, repeated application of the modification step results in the termination condition being reached. (Either we will reach a test entry that is greater than or equal to the target value or we ultimately reach the end of the list.)

Copyright Virtual University of Pakistan

Module 99

99. Algorithm: Iterative Structures: Loop Execution (Examples-1)

Let's write a pseudo-code for finding maximum number from a list. Let's assume the following numbers



def FindMax():

max = first number in the list

Current = second number in the list

While (elements in the list exist)

If (max<current)

max=current

Current = next value in the list

This function will keep on going until we reach at the last value (45) each time, the Max would be compared with the current value. As soon as the max remains small than the compared value, the value in comparison will be given to the max.

Copyright Virtual University of Pakistan

Module 100

100. Algorithm: Iterative Structures: Loop Execution (Examples-II)

Let's write a pseudo-code to find a factorial of a number n.

```
def FindFactorial():
f = 1
i=1;
While (i<=n)
f=f*i;
i=i+1;
```

Initially "f" and "i" both will have a value of one. The loop will keep executing until "i" remains smaller or equal to n. In each iteration of "i", the current value of "i" will be multiplied with the current value of f.

Let's execute it for n=5:

f=1

i=1

First iteration

f= f * I

f=1

i=2

Second iteration

f= f * I

f=2

i=3

Third iteration

f= f * I

f=6

i=4

Forth Iteration

f= f * I

f=24

Copyright Virtual University of Pakistan

i=5

Fifth Iteration

f= f * I

f=120

i=6

when “i” is 6, the loop will terminate as the condition is now false (i<=n)

NOTE THAT PROGRAM CAN COME IN SHORT AND LONG

Copyright Virtual University of Pakistan

Module 101

101. Algorithm: Iterative Structures (Pretest and Posttest loops)

Following the terminology of our pseudocode, we will usually refer to these structures as the while loop structure or the repeat loop structure. In a more generic context you might hear the while loop structure referred to as a **pretest loop** (since the test for termination is performed before the body is executed) and the repeat loop structure referred to as a **posttest loop** (since the test for termination is performed after the body is executed). This has been shown the Figure 84.

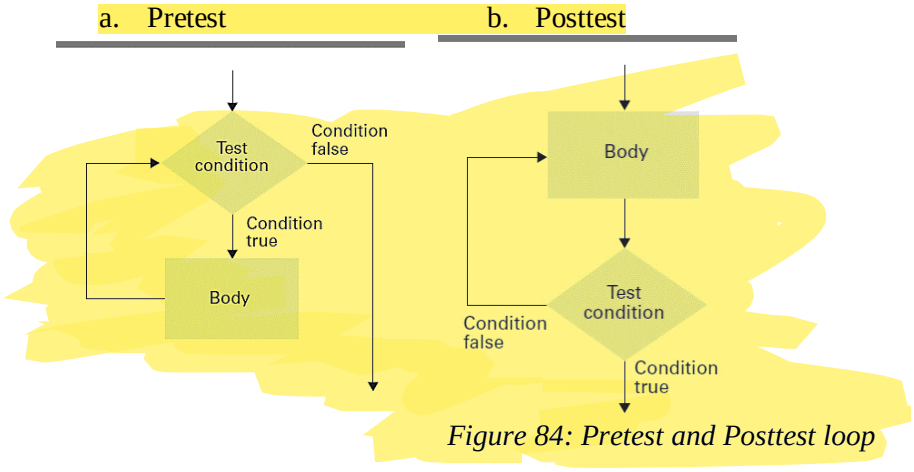


Figure 84: Pretest and Posttest loop

Copyright Virtual University of Pakistan

Module 102

102. Algorithm: Insertion Sort Algorithm

As an additional example of using iterative structures, let us consider the problem of sorting a list of names into alphabetical order. But before proceeding, we should identify the constraints under which we will work. Simply stated, our goal is to sort the list "within itself." In other words, we want to sort the list by shuffling its entries as opposed to moving the list to another location. Our situation is analogous to the problem of sorting a list whose entries are recorded on separate index cards spread out on a crowded desktop. We have cleared off enough space for the cards but are not allowed to push additional materials back to make more room. This restriction is typical in computer applications, not because the workspace within the machine is necessarily crowded like our desktop, but simply because we want to use the storage space available in an efficient manner.

Let us get a foot in the door by considering how we might sort the names on the desktop. Consider the list of names

Fred

Alex

Diana

Byron

Carol

One approach to sorting this list is to note that the sublist consisting of only the top name, Fred, is sorted but the sublist consisting of the top two names, Fred and Alex, is not. Thus we might pick up the card containing the name Alex, slide the name Fred down into the space where Alex was, and then place the name Alex in the hole at the top of the list. This example will continue in the next module.

Copyright Virtual University of Pakistan

Module 103

103. Algorithm: Insertion Sort Algorithm Example

One approach to sorting this list is to note that the sublist consisting of only the top name, Fred, is sorted but the sublist consisting of the top two names, Fred and Alex, is not. Thus we might pick up the card containing the name Alex, slide the name Fred down into the space where Alex was, and then place the name Alex in the hole at the top of the list as represented by the first row in Figure 85. At this point our list would be

Alex

Fred

Diana

Byron

Carol

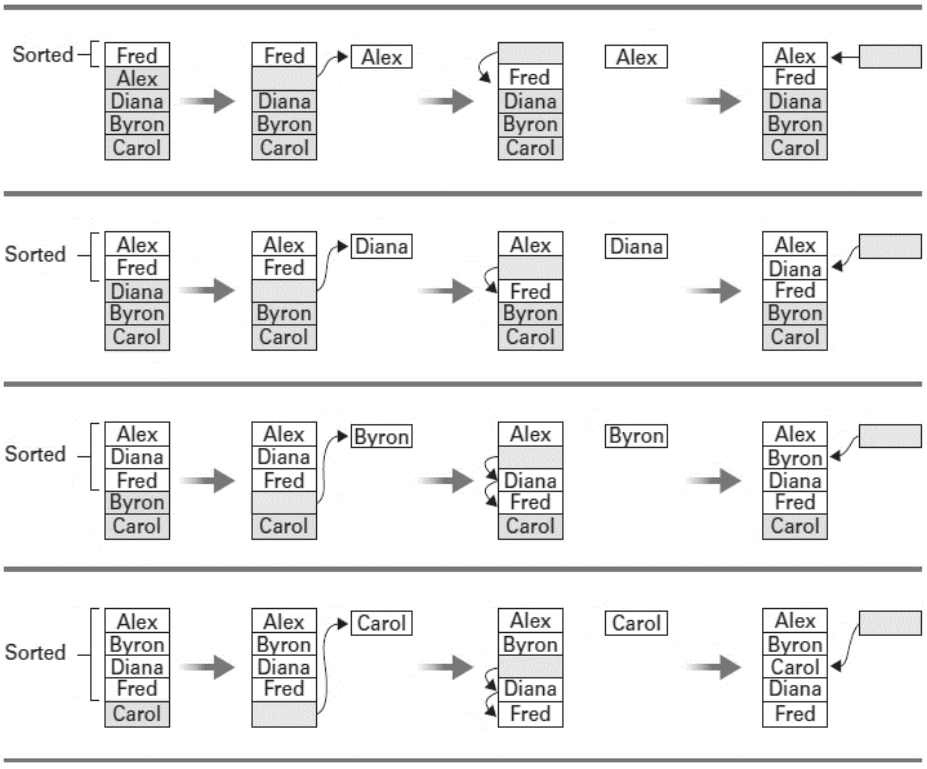


Figure 85: Sorting the list Fred, Alex, Diana, Byron, and Carol alphabetically

Now the top two names form a sorted sublist, but the top three do not. Thus, we might pick up the third name, Diana, slide the name Fred down into the hole where Diana was, and then insert Diana in the hole left by Fred, as summarized in the second row of Figure 5.10. The top three entries in the list would now be sorted. Continuing in this fashion, we could obtain a list in which the top four entries are sorted by picking up the fourth name, Byron, sliding the names Fred and Diana down, and then inserting Byron in the hole (see the third row of Figure 5.10). Finally, we can complete the sorting process by picking up Carol, sliding Fred and Diana down, and then inserting Carol in the remaining hole (see the fourth row of Figure 85).

Copyright Virtual University of Pakistan

Module 104

104. Algorithm: Recursive Structure (The Binary Search Algorithm)

Recursive structures provide an alternative to the loop paradigm for implementing the repetition of activities. Whereas a loop involves repeating a set of instructions in a manner in which the set is completed and then repeated, recursion involves repeating the set of instructions as a subtask of itself. As an analogy, consider the process of conducting telephone conversations with the call waiting feature. There, an incomplete telephone conversation is set aside while another incoming call is processed. The result is that two conversations take place. However, they are not performed one-after-the-other as in a loop structure, but instead one is performed within the other.

As a way of introducing recursion, let us again tackle the problem of searching to see whether a particular entry is in a sorted list, but this time we get our foot in the door by considering the procedure we follow when searching a dictionary. In this case we do not perform a sequential entry-by-entry or even a page-by-page procedure. Rather, we begin by opening the directory to a page in the area where we believe the target entry is located. If we are lucky, we will find the target value there; otherwise, we must continue searching. But at this point we will have narrowed our search considerably.

Of course, in the case of searching a dictionary, we have prior knowledge of where words are likely to be found. If we are looking for the word somnambulism, we would start by opening to the latter portion of the dictionary. In the case of generic lists, however, we do not have this advantage, so let us agree to always start our search with the "middle" entry in the list. Here we write the word middle in quotation marks because the list might have an even number of entries and thus no middle entry in the exact sense. In this case, let us agree that the "middle" entry refers to the first entry in the second half of the list. If the middle entry in the list is the target value, we can declare the search a success. Otherwise, we can at least restrict the search process to the first or last half of the list depending on whether the target value is less than or greater than the entry we have considered. (Remember that the list is sorted.)

To search the remaining portion of the list, we could apply the sequential search, but instead let us apply the same approach to this portion of the list that we used for the whole list. That is, we select the middle entry in the remaining portion of the list as the next entry to consider. As before, if that entry is the target value, we are finished. Otherwise we can restrict our search to an even smaller portion of the list.

This approach to the searching process is summarized in Figure 86, where we consider the task of searching the list on the left of the figure for the entry John. We first consider the middle entry Harry. Since our target belongs after this entry, the search continues by considering the lower half of the original list. The middle of this sublist is found to be Larry. Since our target should precede Larry, we turn our attention to the first half of the current sublist. When we interrogate the middle of that secondary sublist, we find our target John and declare the search a success. In short, our strategy is to successively divide the list in question into smaller segments until the target is found or the search is narrowed to an empty segment.

Copyright Virtual University of Pakistan

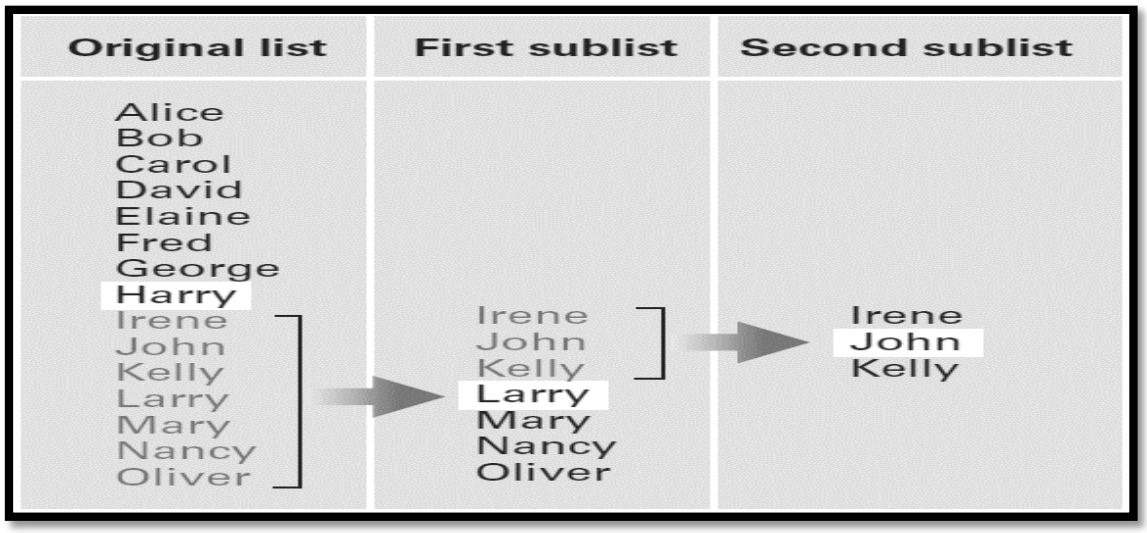


Figure 86: Applying our strategy to search a list for the entry John

We need to emphasize this last point. If the target value is not in the original list, our approach to searching the list will proceed by dividing the list into smaller segments until the segment under consideration is empty. At this point our algorithm should recognize that the search is a failure.

Figure 87 is a first draft of our thoughts using our pseudocode. It directs us to begin a search by testing to see if the list is empty. If so, we are told to report that the search is a failure. Otherwise, we are told to consider the middle entry in the list. If this entry is not the target value, we are told to search either the front half or the back half of the list. Both of these possibilities require a secondary search. It would be nice to perform these searches by calling on the services of an abstract tool. In particular, our approach is to apply a function named Search to carry out these secondary searches. To complete our program, therefore, we must provide such a function.

```
if (List is empty):
  Report that the search failed.
else:
  TestEntry = the "middle" entry in the List
  if (TargetValue == TestEntry):
    Report that the search succeeded.
  if (TargetValue < TestEntry):
    Search() the portion of List preceding TestEntry for TargetValue,
      and report the result of that search.
  if (TargetValue > TestEntry):
    Search() the portion of List following TestEntry for TargetValue,
      and report the result of that search.
```

Figure 87: A first draft of the binary search technique

But this function should perform the same task that is expressed by the pseudocode we have already written. It should first check to see if the list it is given is empty, and if it is not, it should proceed by considering the middle entry of that list. Thus we can supply the function we need merely by identifying the current routine as being the function named Search and inserting references to that function where the secondary searches are required. The result is shown in Figure 88.

Copyright Virtual University of Pakistan

```
def Search(List, TargetValue):
  if (List is empty):
    Report that the search failed.
  else:
    TestEntry = the "middle" entry in List
    if (TargetValue == TestEntry):
      Report that the search succeeded.
    if (TargetValue < TestEntry):
      Sublist = portion of List preceding
        TestEntry
      Search(Sublist, TargetValue)
    if (TargetValue > TestEntry):
      Sublist = portion of List following
        TestEntry
      Search(Sublist, TargetValue)
```

Figure 88: The binary search algorithm in pseudocode

ONLY LMS LECTURE FOR CONCEPT

Copyright Virtual University of Pakistan

Module 105

105. Algorithm: Recursive Control

The binary search algorithm is similar to the sequential search in that each algorithm requests the execution of a repetitive process. However, the implementation of this repetition is significantly different. Whereas the sequential search involves a circular form of repetition, the binary search executes each stage of the repetition as a subtask of the previous stage. This technique is known as **recursion.**

As we have seen, the illusion created by the execution of a recursive function is the existence of multiple copies of the function, each of which is called an activation of the function. These activations are created dynamically in a telescoping manner and ultimately disappear as the algorithm advances. Of those activations existing at any given time, only one is actively progressing. The others are effectively in limbo, each waiting for another activation to terminate before it can continue.

Being a repetitive process, recursive systems are just as dependent on proper control as are loop structures. Just as in loop control, recursive systems are dependent on testing for a termination condition and on a design that ensures this condition will be reached. In fact, proper recursive control involves the same three ingredients such as: initialization, modification, and test for termination – that are required in loop control.

In general, a recursive function is designed to test for the termination condition (often called the **base case** or **degenerative case**) before requesting further activations. If the termination condition is not met, the routine creates another activation of the function and assigns it the task of solving a revised problem that is closer to the termination condition than that assigned to the current activation. However, if the termination condition is met, a path is taken that causes the current activation to terminate without creating additional activations.

Let us see how the initialization and modification phases of repetitive control are implemented in our binary search function of Figure 88. In this case, the creation of additional activations is terminated once the target value is found or the task is reduced to that of searching an empty list. The process is initialized implicitly by being given an initial list and a target value. From this initial configuration the function modifies its assigned task to that of searching a smaller list. Since the original list is of finite length and each modification step reduces the length of the list in question, we are assured that the target value ultimately is found, or the task is reduced to that of searching the empty list. We can therefore conclude that the repetitive process is guaranteed to cease.

Copyright Virtual University of Pakistan

Module 106

106. Algorithm: Algorithm Efficiency

Even though today's machines are capable of executing millions or billions of instructions each second, efficiency remains a major concern in algorithm design. Often the choice between efficient and inefficient algorithms can make the difference between a practical solution to a problem and an impractical one.

Let us consider the problem of a university registrar faced with the task of retrieving and updating student records. Although the university has an actual enrollment of approximately 10,000 students during any one semester, its "current student" file contains the records of more than 30,000 students who are considered current in the sense that they have registered for at least one course in the past few years but have not completed a degree. For now, let us assume that these records are stored in the registrar's computer as a list ordered by student identification numbers. To find any student record, the registrar would therefore search this list for a particular identification number.

We have presented two algorithms for searching such a list: the sequential search and the binary search. Our question now is whether the choice between these two algorithms makes any difference in the case of the registrar. We consider the sequential search first.

Given a student identification number, the sequential search algorithm starts at the beginning of the list and compares the entries found to the identification number desired. Not knowing anything about the source of the target value, we cannot conclude how far into the list this search must go. We can say, though, that after many searches we expect the average depth of the searches to be halfway through the list; some will be shorter, but others will be longer. Thus, we estimate that over a period of time, the sequential search will investigate roughly 15,000 records per search. If retrieving and checking each record for its identification number requires 10 milliseconds (10 one-thousandths of a second), such a search would require an average of 150 seconds or 2.5 minutes—an unbearably long time for the registrar to wait for a student's record to appear on a computer screen. Even if the time required to retrieve and check each record were reduced to only 1 millisecond, the search would still require an average of 15 seconds, which is still a long time to wait.

In contrast, the binary search proceeds by comparing the target value to the middle entry in the list. If this is not the desired entry, then at least the remaining search is restricted to only half of the original list. Thus, after interrogating the middle entry in the list of 30,000 student records, the binary search has at most 15,000 records still to consider. After the second inquiry, at most 7,500 remain, and after the third retrieval, the list in question has dropped to no more than 3,750 entries. Continuing in this fashion, we see that the target record will be found after retrieving at most 15 entries from the list of 30,000 records. Thus, if each of these retrievals can be performed in 10 milliseconds, the process of searching for a particular record requires only 0.15 of a second—meaning that access to any particular student record will appear to be instantaneous from the registrar's point of view. We conclude that the choice between the sequential search algorithm and the binary search algorithm would have a significant impact in this application.

This example indicates the importance of the area of computer science known as algorithm analysis that encompasses the study of the resources, such as time or storage space, that algorithms require. A major application of such studies is the evaluation of the relative merits of alternative algorithms. Algorithm analysis often involves best-case, worst-case, and average-case scenarios. In our example, we performed an average-case analysis of the sequential search algorithm and a worst-case analysis of the binary search algorithm in order to estimate the time required to search through a list of 30,000 entries. In general such analysis is performed in a more generic context. That is, when considering algorithms for searching lists, we do not focus on a list of a particular length, but instead try to identify a formula that would indicate the algorithm's performance for lists of arbitrary lengths. It is not difficult to generalize our previous reasoning to lists of arbitrary lengths. In particular, when applied to a list with *n* entries, the sequential search

Copyright Virtual University of Pakistan

algorithm will interrogate an average of $n/2$ entries, whereas the binary search algorithm will interrogate at most $log_2 n$ entries in its worst-case scenario. ($log_2 n$ represents the base two logarithm of n. Unless otherwise stated, computer scientists usually mean base two when talking about logarithms.

Copyright Virtual University of Pakistan

Module 107

107. Algorithm: Software Verification

Recall that the fourth phase in Polya's analysis of problem solving (module 92) is to evaluate the solution for accuracy and for its potential as a tool for solving other problems. The significance of the first part of this phase is exemplified by the following example:

"A traveler with a gold chain of seven links must stay in an isolated hotel for seven nights. The rent each night consists of one link from the chain. What is the fewest number of links that must be cut so that the traveler can pay the hotel one link of the chain each morning without paying for lodging in advance?"

To solve this problem, we first realize that not every link in the chain must be cut. If we cut only the second link, we could free both the first and second links from the other five. Following this insight, we are led to the solution of cutting only the second, fourth, and sixth links in the chain, a process that releases each link while cutting only three (Figure 89). Furthermore, any fewer cuts leaves two links connected, so we might conclude that the correct answer to our problem is three.

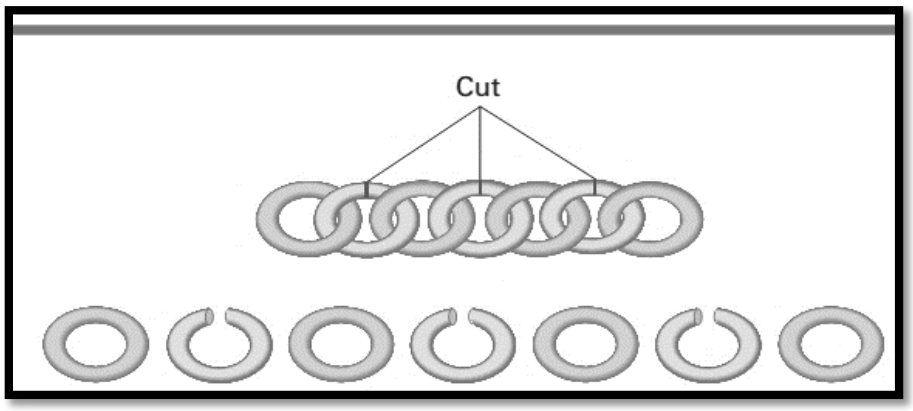


Figure 89: Separating the chain using only three cuts

Upon reconsidering the problem, however, we might make the observation that when only the third link in the chain is cut, we obtain three pieces of chain of lengths one, two, and four (Figure 90). With these pieces we can proceed as follows:

First morning: Give the hotel the single link.

Second morning: Retrieve the single link and give the hotel the two-link piece.

Third morning: Give the hotel the single link.

Fourth morning: Retrieve the three links held by the hotel and give the hotel the four-link piece.

Fifth morning: Give the hotel the single link.

Sixth morning: Retrieve the single link and give the hotel the double-link piece.

Seventh morning: Give the hotel the single link

Copyright Virtual University of Pakistan

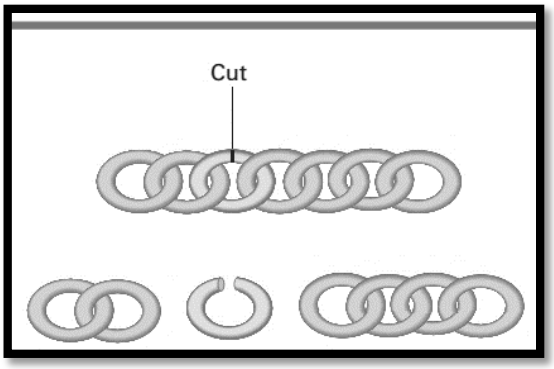


Figure 90: Solving the problem with only one cut

Consequently, our first answer, which we thought was correct, is incorrect. How, then, can we be sure that our new solution is correct? We might argue as follows: Since a single link must be given to the hotel on the first morning, at least one link of the chain must be cut, and since our new solution requires only one cut, it must be optimal.

Translated into the programming environment, this example emphasizes the distinction between a program that is believed to be correct and a program that is correct. The two are not necessarily the same. The data processing community is rich in horror stories involving software that although "known" to be correct, still failed at a critical moment because of some unforeseen situation. Verification of software is therefore an important undertaking, and the search for efficient verification techniques constitutes an active field of research in computer science.

Copyright Virtual University of Pakistan

Module 108

108. Algorithm: Software Verification Examples

Software Verification?

- ✓ Distinction between Program believed to be correct and the program that is correct.

Example #1 – Software Failure

- ✓ Fatal Therac-25 X-ray Radiation
- ✓ In 1986, a man in Texas received between 16,500-25,000 radiations in less than 10 sec, over an area of about 1 cm.
- ✓ He passed away 5 months later.
- ✓ The root cause of the incident was a SW failure

Example #2 – Welsch NHS IT Failure

- ✓ In 2018, doctors and staff were unable to access patients files in UK.
- ✓ This was a technical issue of software not the security issue.
- ✓ Hospitals and GP surgeries were affected.

Module 109

109. Programming Languages: Early Generations-I

As we learned in Figure 46, programs for modern computers consist of sequences of instructions that are encoded as numeric digits. Such an encoding system is known as a machine language. Unfortunately, writing programs in a machine language is a tedious task that often leads to errors that must be located and corrected (a process known as **debugging**) before the job is finished.

In the 1940s, researchers simplified the programming process by developing notational systems by which instructions could be represented in mnemonic rather than numeric form. For example, the instruction move the contents of register 5 to register 6 would be expressed as:

4056

using the machine language introduced in Figure 46, whereas in a mnemonic system it might appear as:

MOV R5, R6

As a more extensive example, the machine language routine

156C
166D
5056
306E
C000

which adds the contents of memory cells 6C and 6D and stores the result at location 6E (Figure 46) might be expressed as

LD R5,Price
LD R6, ShippingCharge
ADDI R0,R5 R6
ST R0,TotalCost
HLT

using mnemonics. (Here we have used LD, ADDI, ST, and HLT to represent *load*, *add*, *store*, and *halt*. Moreover, we have used the descriptive names Price, ShippingCharge, and TotalCost to refer to the memory

Copyright Virtual University of Pakistan

cells at locations 6C, 6D, and 6E, respectively. Such descriptive names are often called program variables or **identifiers.**) Note that the mnemonic form, although still lacking, does a better job of representing the meaning of the routine than does the numeric form.

Copyright Virtual University of Pakistan

Module 110

110. Programming Languages: Early Generations-II

In the continuation of the previous module.

Once such a mnemonic system was established, programs called **assemblers** were developed to convert mnemonic expressions into machine language instructions. Thus, rather than being forced to develop a program directly in machine language, a human could develop a program in mnemonic form and then have it converted into machine language by means of an assembler.

A mnemonic system for representing programs is collectively called an **assembly language.** At the time assembly languages were first developed, they represented a giant step forward in the search for better programming techniques. In fact, assembly languages were so revolutionary that they became known as second-generation languages, the first generation being the machine languages themselves.

Although assembly languages have many advantages over their machine language counterparts, they still fall short of providing the ultimate programming environment. After all, the primitives used in an assembly language are essentially the same as those found in the corresponding machine language. The difference is simply in the syntax used to represent them. Thus, a program written in an assembly language is inherently machine dependent – that is, the instructions within the program are expressed in terms of a particular machine's attributes. In turn, a program written in assembly language cannot be easily transported to another computer design because it must be rewritten to conform to the new computer's register configuration and instruction set.

Another disadvantage of an assembly language is that a programmer, although not required to code instructions in numeric form, is still forced to think in terms of the small, incremental steps of the machine's language. The situation is analogous to designing a house in terms of boards, nails, bricks, and so on. It is true that the actual construction of the house ultimately requires a description based on these elementary pieces, but the design process is easier if we think in terms of larger units such as rooms, windows, doors, and so on.

In short, the elementary primitives in which a product must ultimately be constructed are not necessarily the primitives that should be used during the product's design. The design process is better suited to the use of high-level primitives, each representing a concept associated with a major feature of the product. Once the design is complete, these primitives can be translated to lower-level concepts relating to the details of implementation.

Following this philosophy, computer scientists began developing programming languages that were more conducive to software development than were the low-level assembly languages. The result was the emergence of a third generation of programming languages that differed from previous generations in that their primitives were both higher level (in that they expressed instructions in larger increments) and **machine independent** (in that they did not rely on the characteristics of a particular machine). The best-known early examples are FORTRAN (FORmula TRANslator), which was developed for scientific and engi- neering applications, and COBOL (COmmon Business-Oriented Language), which was developed by the U.S. Navy for business applications.

In general, the approach to third-generation programming languages was to identify a collection of high-level primitives in which software could be developed. Each of these primitives was designed so that it could be implemented as a sequence of the low-level primitives available in machine languages. For example, the statement

assign TotalCost the value Price + ShippingCharge

Copyright Virtual University of Pakistan

expresses a high-level activity without reference to how a particular machine should perform the task, yet it can be implemented by the sequence of machine instructions discussed earlier. Thus, our pseudocode structure

identifier = expression

is a potential high-level primitive.

Once this collection of high-level primitives had been identified, a program, called a **translator,** was written that translated programs expressed in these high-level primitives into machine-language programs. Such a translator was similar to the second-generation assemblers, except that it often had to compile several machine instructions into short sequences to simulate the activity requested by a single high-level primitive. Thus, these translation programs were often called **compilers.**

An alternative to translators, called **interpreters,** emerged as another means of implementing third-generation languages. These programs were similar to translators except that they executed the instructions as they were translated instead of recording the translated version for future use. That is, rather than producing a machine-language copy of a program that would be executed later, an interpreter actually executed a program from its high-level form.

As a side issue, we should note that the task of promoting third-generation programming languages was not as easy as might be imagined. The thought of writing programs in a form similar to a natural language was so revolutionary that many in managerial positions fought the notion at first. Grace Hopper, who is recognized as the developer of the first compiler, often told the story of demonstrating a translator for a third-generation language in which German terms, rather than English, were used. The point was that the programming language was constructed around a small set of primitives that could be expressed in a variety of natural languages with only simple modifications to the translator. But she was surprised to find that many in the audience were shocked that, in the years surrounding World War II, she would be teaching a computer to "understand" German. Today we know that understanding a natural language involves much, much more than responding to a few rigorously defined primitives. Indeed, **natural languages** (such as English, German, and Latin) are distinguished from **formal languages** (such as programming languages) in that the latter are precisely defined by grammars, whereas the former evolved over time without formal grammatical analysis.

Copyright Virtual University of Pakistan

Module 111

111. Programming Languages: Machine Independence

With the development of third-generation languages, the goal of machine independence was largely achieved. Since the statements in a third-generation language did not refer to the attributes of any particular machine, they could be compiled as easily for one machine as for another. A program written in a third-generation language could theoretically be used on any machine simply by applying the appropriate compiler.

Reality, however, has not proven to be this simple. When a compiler is designed, particular characteristics of the underlying machine are sometimes reflected as conditions on the language being translated. For example, the different ways in which machines handle I/O operations have historically caused the "same" language to have different characteristics, or dialects, on different machines. Consequently, it is often necessary to make at least minor modifications to a program to move it from one machine to another.

Compounding this problem of portability is the lack of agreement in some cases as to what constitutes the correct definition of a particular language. To aid in this regard, the American National Standards Institute and the International Organization for Standardization have adopted and published standards for many of the popular languages. In other cases, informal standards have evolved because of the popularity of a certain dialect of a language and the desire of other compiler writers to produce compatible products. However, even in the case of highly standardized languages, compiler designers often provide features, sometimes called language extensions, that are not part of the standard version of the language. If a programmer takes advantage of these features, the program produced will not be compatible with environments using a compiler from a different vendor. In the overall history of programming languages, the fact that third-generation languages fell short of true machine independence is actually of little significance for two reasons. First, they were close enough to being machine independent that software could be transported from one machine to another with relative ease. Second, the goal of machine independence turned out to be only a seed for more demanding goals. Indeed, the realization that machines could respond to such high-level statements as

assign TotalCost the value Price + ShippingCharge

led computer scientists to dream of programming environments that would allow humans to communicate with machines in terms of abstract concepts rather than forcing them to translate these concepts into machine-compatible form.

Moreover, computer scientists wanted machines that could perform much of the algorithm discovery process rather than just algorithm execution. The result has been an ever-expanding spectrum of programming languages that challenges a clear-cut classification in terms of generations.

Copyright Virtual University of Pakistan

Module 112

112. Programming Languages: Imperative Paradigms

The generation approach to classifying programming languages is based on a linear scale (Figure 91) on which a language's position is determined by the degree to which the user of the language is freed from the world of computer gibberish and allowed to think in terms associated with the problem being solved. In reality, the development of programming languages has not progressed in this manner but has developed along different paths as alternative approaches to the programming process (called **programming paradigms**) have surfaced and been pursued. Consequently, the historical development of programming languages is better represented by a multiple-track diagram as shown in Figure 92, in which different paths resulting from different paradigms are shown to emerge and progress independently. In particular, the figure presents four paths representing the functional, object-oriented, imperative, and declarative paradigms, with various languages associated with each paradigm positioned in a manner that indicates their births relative to other languages. (It does not imply that one language necessarily evolved from a previous one.)

We should note that although the paradigms identified in Figure 92 are called *programming* paradigms, these alternatives have ramifications beyond the programming process. They represent fundamentally different approaches to building solutions to problems and therefore affect the entire software development process. In this sense, the term *programming paradigm* is a misnomer. A more realistic term would be *software development paradigm.*

The **imperative paradigm,** also known as the **procedural paradigm,** represents the traditional approach to the programming process. It is the paradigm on which Python and our pseudocode are based as well as the machine language. As the name suggests, the imperative paradigm defines the programming process to be the development of a sequence of commands that, when followed, manipulate data to produce the desired result. Thus, the imperative paradigm tells us to approach the programming process by finding an algorithm to solve the problem at hand and then expressing that algorithm as a sequence of commands.

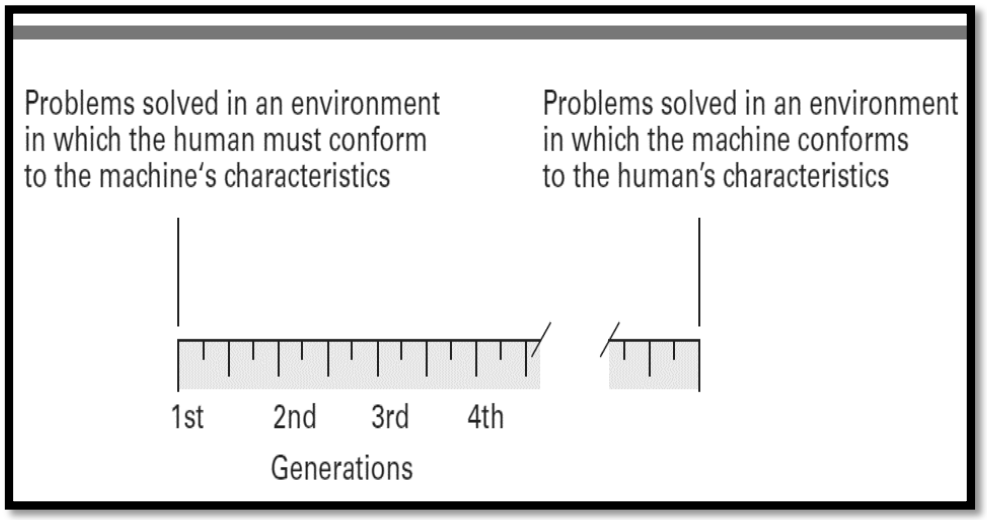


Figure 91: Generations of programming languages

Copyright Virtual University of Pakistan

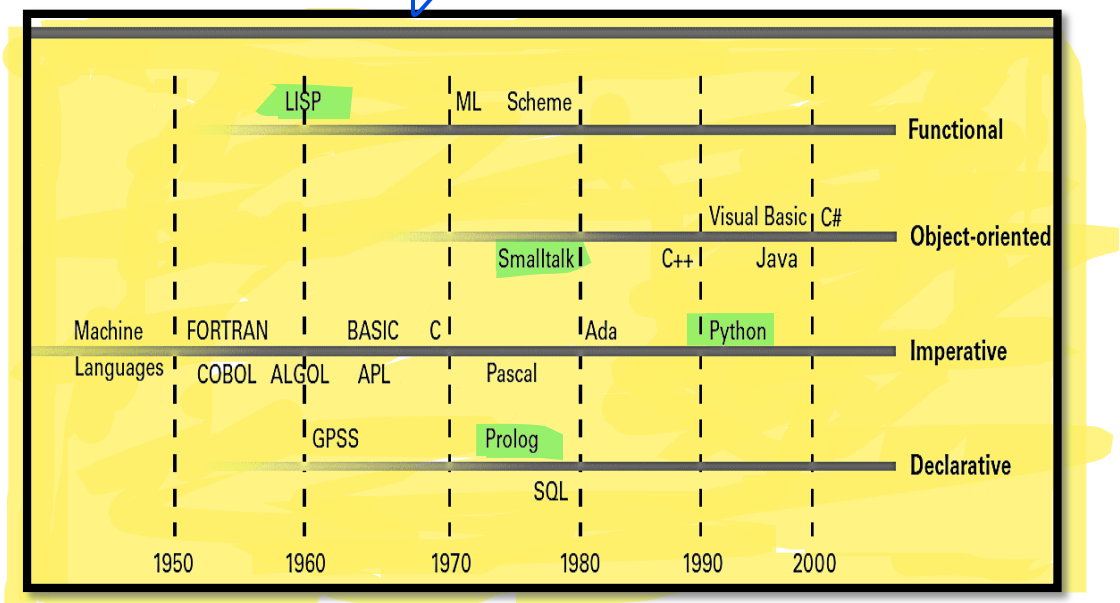


Figure 92: The evolution of programming paradigms

Copyright Virtual University of Pakistan

Module 113

113. Programming Languages: Declarative Paradigms

In contrast to the imperative paradigm is the **declarative paradigm**, which asks a programmer to describe the problem to be solved rather than an algorithm to be followed. More precisely, a declarative programming system applies a pre-established general-purpose problem-solving algorithm to solve problems presented to it. In such an environment the task of a programmer becomes that of developing a precise statement of the problem rather than of describing an algorithm for solving the problem.

A major obstacle in developing programming systems based on the declarative paradigm is the need for an underlying problem-solving algorithm. For this reason, early declarative programming languages tended to be special-purpose in nature, designed for use in particular applications. For example, the declarative approach has been used for many years to simulate a system (political, economic, environmental, and so on) in order to test hypotheses or to obtain predictions. In these settings, the underlying algorithm is essentially the process of simulating the passage of time by repeatedly re-computing values of parameters (gross domestic product, trade deficit, and so on) based on the previously computed values. Thus, implementing a declarative language for such simulations requires that one first implement an algorithm that performs this repetitive function. Then the only task required of a programmer using the system is to describe the situation to be simulated. In this manner, a weather forecaster does not need to develop an algorithm for forecasting the weather but merely describes the current weather status, allowing the underlying simulation algorithm to produce weather predictions for the near future.

A tremendous boost was given to the declarative paradigm with the discovery that the subject of formal logic within mathematics provides a simple problem- solving algorithm suitable for use in a general-purpose declarative programming system. The result has been increased attention to the declarative paradigm and the emergence of **logic programming.**

Copyright Virtual University of Pakistan

Module 114

114. Programming Languages: Functional Paradigm

Another programming paradigm is the **functional paradigm.** Under this paradigm a program is viewed as an entity that accepts inputs and produces outputs. Mathematicians refer to such entities as functions, which is the reason this approach is called the functional paradigm. Under this paradigm a program is constructed by connecting smaller predefined program units (predefined functions) so that each unit's outputs are used as another unit's inputs in such a way that the desired overall input-to-output relationship is obtained. In short, the programming process under the functional paradigm is that of building functions as nested complexes of simpler functions.

As an example, Figure 93 shows how a function for balancing your checkbook can be constructed from two simpler functions. One of these, called Find_sum, accepts values as its input and produces the sum of those values as its output. The other, called Find_diff, accepts two input values and computes their difference. The structure displayed in Figure 6.3 can be represented in the LISP programming language (a prominent functional programming language) by the expression

(Find_diff (Find_sum Old_balance Credits) (Find_sum Debits))

The nested structure of this expression (as indicated by parentheses) reflects the fact that the inputs to the function Find_diff are produced by two applications of Find_sum. The first application of Find_sum produces the result of adding all the Credits to the Old_balance. The second application of Find_sum computes the total of all Debits. Then, the function Find_diff uses these results to obtain the new checkbook balance.

To more fully understand the distinction between the functional and imperative paradigms, let us compare the functional program for balancing a checkbook to the following pseudocode program obtained by following the imperative paradigm:

Total_credits = sum of all Credits
Temp_balance = Old_balance + Total_credits
Total_debits = sum of all Debits
Balance = Temp_balance - Total_debits

NOTE THAT PROGRAM CAN COME IN SHORT AND LONG

Copyright Virtual University of Pakistan

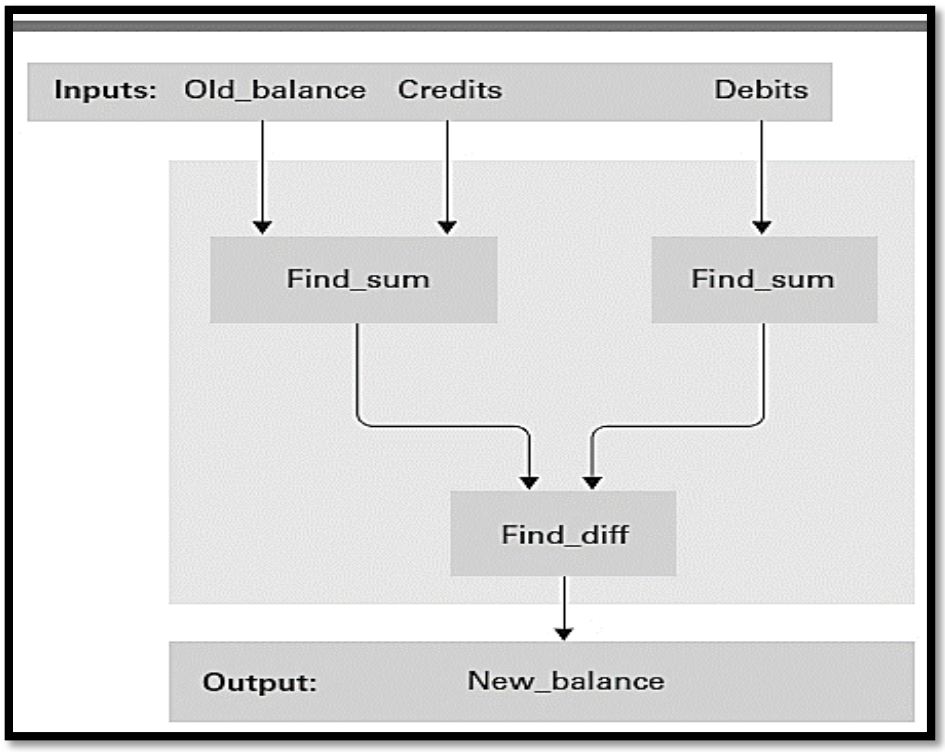


Figure 93: A function for checkbook balancing constructed from simpler functions

Note that this imperative program consists of multiple statements, each of which requests that a computation be performed and that the result be stored for later use. In contrast, the functional program consists of a single statement in which the result of each computation is immediately channeled into the next. In a sense, the imperative program is analogous to a collection of factories, each converting its raw materials into products that are stored in warehouses. From these warehouses, the products are later shipped to other factories as they are needed. But the functional program is analogous to a collection of factories that are coordinated so that each produces only those products that are ordered by other factories and then immediately ships those products to their destinations without intermediate storage. This efficiency is one of the benefits proclaimed by proponents of the functional paradigm.

Copyright Virtual University of Pakistan

Module 115

115. Programming Languages: Object Oriented Paradigm

Another programming paradigm (and the most prominent one in today's software development) is the **object-oriented paradigm,** which is associated with the programming process called **object-oriented programming (OOP).** Following this paradigm, a software system is viewed as a collection of units, called **objects,** each of which is capable of performing the actions that are immediately related to itself as well as requesting actions of other objects. Together, these objects interact to solve the problem at hand.

As an example of the object-oriented approach at work, consider the task of developing a graphical user interface. In an object-oriented environment, the icons that appear on the screen would be implemented as objects. Each of these objects would encompass a collection of functions (called **methods** in the object-oriented vernacular) describing how that object is to respond to the occurrence of various events, such as being selected by a click of the mouse button or being dragged across the screen by the mouse. Thus, the entire system would be constructed as a collection of objects, each of which knows how to respond to the events related to it.

To contrast the object-oriented paradigm with the imperative paradigm, consider a program involving a list of names. In the traditional imperative paradigm, this list would merely be a collection of data. Any program unit accessing the list would have to contain the algorithms for performing the required manipulations. In the object-oriented approach, however, the list would be constructed as an object that consisted of the list together with a collection of methods for manipulating the list. (This might include functions for inserting a new entry in the list, deleting an entry from the list, detecting if the list is empty, and sorting the list.) In turn, another program unit that needed to manipulate the list would not contain algorithms for performing the pertinent tasks. Instead, it would make use of the functions provided in the object. In a sense, rather than sorting the list as in the imperative paradigm, the program unit would ask the list to sort itself. Its significance in today's software development arena dictates that we include the concept of a class in this introduction. To this end, recall that an object can consist of data (such as a list of names) together with a collection of methods for per- forming activities (such as inserting new names in the list). These features must be described by statements in the written program. This description of the object's properties is called a **class.** Once a class has been constructed, it can be applied anytime an object with those characteristics is needed. Thus, several objects can be based on (that is, built from) the same class. Just like identical twins, these objects would be distinct entities but would have the same characteristics because they are constructed from the same template (the same class). (An object that is based on a particular class is said to be an **instance** of that class.)

Copyright Virtual University of Pakistan

Module 116

116. Programming Languages: Variable and Data Types

High-level programming languages allow locations in main memory to be referenced by descriptive names rather than by numeric addresses. Such a name is known as a **variable,** in recognition of the fact that by changing the value stored at the location, the value associated with the name changes as the program executes. Unlike Python, our example languages in this chapter require that variables be identified via a declarative statement prior to being used elsewhere in the program. These declarative statements also require that the programmer describe the type of data that will be stored at the memory location associated with the variable.

Such a type is known as a **data type** and encompasses both the manner in which the data item is encoded and the operations that can be performed on that data. For example, the type **integer** refers to numeric data consisting of whole numbers, probably stored using two's complement notation. Operations that can be performed on integer data include the traditional arithmetic operations and comparisons of relative size, such as determining whether one value is greater than another. The type **float** (sometimes called **real**) refers to numeric data that might contain values other than whole numbers, probably stored in floating-point notation. Operations performed on data of type float are similar to those performed on data of type integer. Recall, however, that the activity required for adding two items of type float differs from that for adding two items of type integer.

Suppose, then, that we wanted to use the variable WeightLimit in a program to refer to an area of main memory containing a numeric value encoded in two's complement notation. In the languages C, C++, Java, and C# we would declare our intention by inserting the statement

int WeightLimit;

toward the beginning of the program. This statement means "The name WeightLimit will be used later in the program to refer to a memory area containing a value stored in two's complement notation." Multiple variables of the same type can normally be declared in the same declaration statement. For example, the statement

int Height, Width;

would declare both Height and Width to be variables of type integer. Moreover, most languages allow a variable to be assigned an initial value when it is declared. Thus,

int WeightLimit = 100;

would not only declare WeightLimit to be a variable of type integer but also assign it the starting value 100. In contrast, dynamically typed languages like Python allow variables to be assigned without first declaring their type; such variables are checked for correct type later, when operations are performed upon them.

Other common data types include character and Boolean. The type character refers to data consisting of symbols, probably stored using ASCII or Unicode. Operations performed on such data include comparisons such as determining whether one symbol occurs before another in alphabetical order, testing to see whether one string of symbols appears inside another, and concatenating one string of symbols at the end of another to form one long string. The statement

char Letter, Digit;

could be used in the languages C, C++, C#, and Java to declare the variables

Letter and Digit to be of type character.

Copyright Virtual University of Pakistan

The type **Boolean** refers to data items that can take on only the values true or false. Operations on data of type Boolean include inquiries as to whether the current value is true or false. For example, if the variable LimitExceeded was declared to be of type Boolean, then a statement of the form

if (LimitExceeded) then (...) else (...)

would be reasonable.

The data types that are included as primitives in a programming language, such as int for integer and char for character, are called **primitive data types.** As we have learned, the data types such as: integer, float, character, and boolean are common primitives. Other data types that have not yet become widespread primitives include images, audio, video, and hypertext. However, types such as GIF, JPEG, and HTML might soon become as common as integer and float.

Copyright Virtual University of Pakistan

Module 117

117. Programming Languages: Data Structure

In addition to data type, variables in a program are often associated with **data structure,** which is the conceptual shape or arrangement of data. For example, text is normally viewed as a long string of characters, whereas sales records might be envisioned as a rectangular table of numeric values, where each row represents the sales made by a particular employee and each column represents the sales made on a particular day.

One common data structure is the **array,** which is a block of elements of the same type such as a one-dimensional list, a two-dimensional table with rows and columns, or tables with higher dimensions. To establish such an array in a program, many programming languages require that the declaration statement declaring the name of the array also specify the length of each dimension of the array. For example, Figure 94 displays the conceptual structure declared by the statement

int Scores[2][9];

in the language C, which means "The variable Scores will be used in the following program unit to refer to a two-dimensional array of integers having two rows and nine columns." The same statement in FORTRAN would be written as

INTEGER Scores (2, 9)

Once an array has been declared, it can be referenced elsewhere in the program by its name, or an individual element can be identified by means of integer values called **indices** that specify the row, column, and so on, desired. However, the range of these indices varies from language to language. For example, in C (and its derivatives C++, Java, and C#) indices start at 0, meaning that the entry in the second row and fourth column of the array called Scores (as declared above) would be referenced by Scores[1][3], and the entry in the first row and first column would be Scores[0][0]. In contrast, indices start at 1 in a FORTRAN program so the entry in the second row and fourth column would be referenced by Scores(2,4) (see again Figure 94).

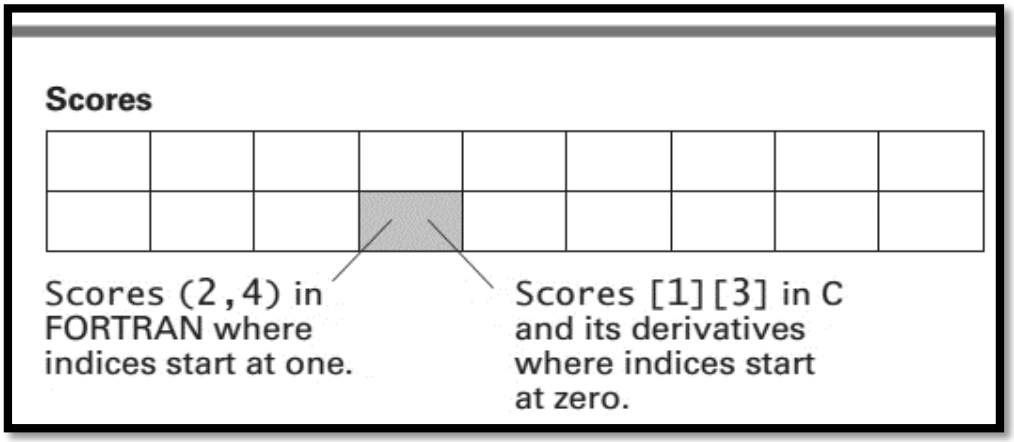


Figure 94: A two-dimensional array with two rows and nine columns

In contrast to an array in which all data items are the same type, an **aggregate type** (also called a **structure,** a **record,** or sometimes a **heterogeneous array**) is a block of data in which different elements can have different types. For instance, a block of data referring to an employee might consist of an entry called Name

Copyright Virtual University of Pakistan

of type character, an entry called Age of type integer, and an entry called SkillRating of type float. Such an aggregate type would be declared in C by the statement

```
struct
{char  Name[25];
int Age;
float SkillRating;
} Employee;
```

which says that the variable Employee is to refer to a structure (abbreviated struct) consisting of three components called Name (a string of 25 characters), Age, and SkillRating (Figure 95). Once such an aggregate has been declared, a programmer can use the structure name (Employee) to refer to the entire aggregate or can reference individual **fields** within the aggregate by means of the structure name followed by a period and the field name (such as Employee.Age).

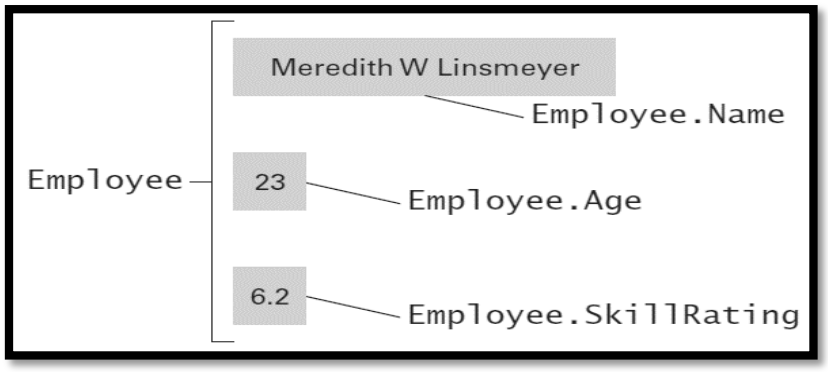


Figure 95: The conceptual layout of the structure Employee

NOTE THAT PROGRAM CAN COME IN SHORT AND LONG

Copyright Virtual University of Pakistan

Module 118

118. Programming Languages: Assignment Statement

Once the special terminology to be used in a program (such as the variables and constants) has been declared, a programmer can begin to describe the algorithms involved. This is done by means of imperative statements. The most basic imperative statement is the **assignment statement,** which requests that a value be assigned to a variable (or more precisely, stored in the memory area identified by the variable). Such a statement normally takes the syntactic form of a variable, followed by a symbol representing the assignment operation, and then by an expression indicating the value to be assigned. The semantics of such a statement is that the expression is to be evaluated and the result stored as the value of the variable. For example, the statement:

```
Z = X + Y;
```

in C, C++, C#, and Java requests that the sum of X and Y be assigned to the variable Z. The semicolon at the end of the line, which is used to separate statements in many imperative languages, is the only syntactic difference from an equivalent Python assignment statement. In some other languages (such as Ada) the equiva- lent statement would appear as:

```
Z := X + Y;
```


Copyright Virtual University of Pakistan

Module 119

119. Programming Languages: Control Structures (if-statement)

A **control statement** alters the execution sequence of the program. Of all the programming constructs.

if (condition)

StatementA

else

StatementB.

For example, if a student's gets more than or equal to 50 marks, the student passes the examination. To denote this using the if-statement, we will proceed as follow:

if (marks>=50)

You have passed the examination

Else

You have failed the examination

NOTE THAT PROGRAM CAN COME IN SHORT AND LONG

Copyright Virtual University of Pakistan

Module 120

120. Programming Languages: Control Structures (if-statement examples)

In this module, we will learn another example. Suppose a university wants to give a scholarship, if a student gets more than 3.0 CGPA in a given semester. We can write the program in the following way and you have also seen its implementation in the videos in the online compiler.

```
float f=3.5;
if (CGPA>=3.0)
cout<<"Give Scholarship";
else
cout<<"Sorry you do not qualify for the scholarship";
```

NOTE THAT PROGRAM CAN COME IN SHORT AND LONG AND MCQS

Copyright Virtual University of Pakistan

Module 121

121. Programming Languages: Control Structures (Loops)

There is another type of control structure known as loop. The loop control structure iterates a set of instructions based on the provided condition.

Consider the following example, we are interested to print the counting from 1 to 5. One way of doing this is as follows:

```
cout<<"1";
cout<<"2";
cout<<"3";
cout<<"4";
cout<<"5";
```

Doing this using loop is easier, you just need to give one statement that is printing counting and you need to tell that how many times, you want to do it. One can write the loop as follows:

```
int i=1;
while(i<=5)
{
cout<<i;
i=i+1;
}
```

This loop will execute 5 times as follows:

First the value of "i" would be 1. The condition would be checked and based on the condition as true (i<=5 i.e 1<=5), the loop will continue and will print the value of "i" which is 1. Then it will increment the value of "i". The new value of "i" would be 2. Then again condition would be checked, and the condition would be true as 2<=5. This will print "2" on the screen and will continue for "i" as 3, 4, and 5, and will print 3, 4, 5 on the screen as well. When the value of "i" would be 6, the loop will find the condition as false (as 6<=5). This would be the termination for the loop.

NOTE THAT PROGRAM CAN COME IN SHORT AND LONG

Copyright Virtual University of Pakistan

Module 122

122. Programming Languages: Programming Concurrent Activities

Suppose we were asked to design a program to produce animation for an action computer game involving multiple attacking enemy spaceships. One approach would be to design a single program that would control the entire animation screen. Such a program would be charged with drawing each of the spaceships, which (if the animation is to appear realistic) would mean that the program would have to keep up with the individual characteristics of numerous spacecraft. An alternate approach would be to design a program to control the animation of a single spaceship whose characteristics are determined by parameters assigned at the beginning of the program's execution. Then the animation could be constructed by creating multiple activations of this program, each with its own set of parameters. By executing these activations simultaneously, we could obtain the illusion of many individual spaceships streaking across the screen at the same time.

Such simultaneous execution of multiple activations is called **parallel processing** or **concurrent processing.** True parallel processing requires multiple CPU cores, one to execute each activation. When only one CPU is available, the illusion of parallel processing is obtained by allowing the activations to share the time of the single processor in a manner similar to that implemented by multiprogramming systems.

Many modern computer applications are more easily solved in the context of parallel processing than in the more traditional context involving a single sequence of instructions. In turn, newer programming languages provide syntax for expressing the semantic structures involved in parallel computations. The design of such a language requires the identification of these semantic structures and the development of a syntax for representing them.

Each programming language tends to approach the parallel processing paradigm from its own point of view, resulting in different terminology. For example, what we have informally referred to as an *activation* is called a *task* in the Ada vernacular and a *thread* in Java. That is, in an Ada program, simultaneous actions are performed by creating multiple *tasks*, whereas in Java one creates multiple *threads*. In either case, the result is that multiple activities are generated and executed in much the same way as processes under the control of a multitasking operating system. We will adopt the Java terminology and refer to such "processes" as threads.

Perhaps the most basic action that must be expressed in a program involving parallel processing is that of creating new threads. If we want multiple activations of the spaceship program to be executed at the same time, we need a syntax for saying so. Such spawning of new threads is similar to that of requesting the execution of a traditional function. The difference is that, in the traditional setting, the program unit that requests the activation of a function does not progress any further until the requested function terminates (recall Figure 6.8), whereas in the parallel context the requesting program unit continues execution while the requested function performs its task (Figure 96). Thus, to create multiple space- ships streaking across the screen, we would write a main program that simply generates multiple activations of the spaceship program, each provided with the parameters describing the distinguishing characteristics of that spaceship.

Copyright Virtual University of Pakistan

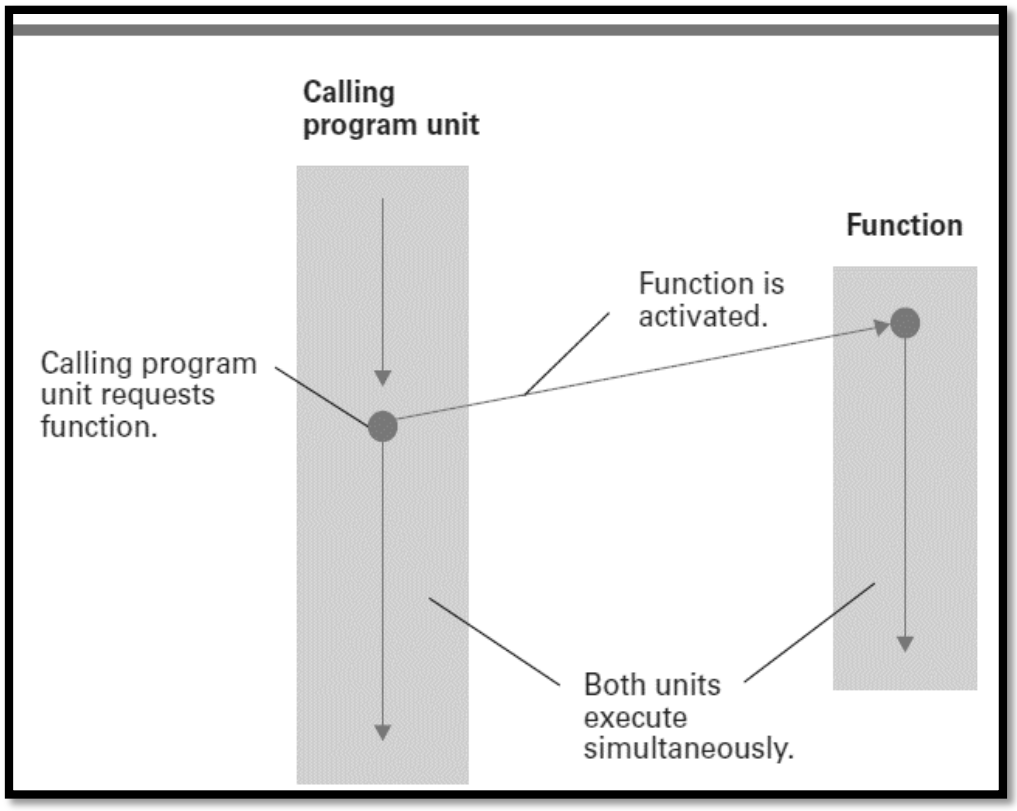


Figure 96: Spawning threads

A more complex issue associated with parallel processing involves handling communication between threads. For instance, in our spaceship example, the threads representing the different spaceships might need to communicate their locations among themselves in order to coordinate their activities. In other cases, one thread might need to wait until another reaches a certain point in its computation, or one thread might need to stop another one until the first has accomplished a particular task.

Such communication needs have long been a topic of study among computer scientists, and many newer programming languages reflect various approaches to thread interaction problems. As an example, let us consider the communication problems encountered when two threads manipulate the same data. If each of two threads that are executing concurrently need to add the value three to a common item of data, a method is needed to ensure that one thread is allowed to complete its transaction before the other is allowed to perform its task. Otherwise they could both start their individual computations with the same initial value, which would mean that the final result would be incremented by only three rather than six. Data that can be accessed by only one thread at a time is said to have mutually exclusive access.

One way to implement mutually exclusive access is to write the program units that describe the threads involved so that when a thread is using shared data, it blocks other threads from accessing that data until such access is safe.

Download More Highlighted Handouts From

VUAnswer.com

Copyright Virtual University of Pakistan

Module 123

123. Programming Languages: Arithmetic Operators Examples

C-language has the following arithmetic operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

+, -, and * are the same as used in the mathematics. However, the "/" has a difference. If one of the operands is decimal number, then it results in the same way as in mathematics, for example:

5.0/2.0 would result into 2.5.

However, when both operands are integers, then it would truncate the decimal point and

5/2 would result into 2.

The remaining "1" can be acquired by using the modulus operator (%).

5%2 would give 1.

MCQS PURPOSE

Module 124

124. Programming Languages: Relational Operators Examples

C-language has the following relational operators:

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
= =	Equal to
!=	Not Equal to

C++ Relational Operators are used to compare values of two variables. Here in example we used the operators in if statement.

Now if the result after comparison of two variables is True, then if statement returns value 1.

And if the result after comparison of two variables is False, then if statement returns value 0.

```
#include<iostream>
using namespace std;
void main()
{
  int a=10,b=20,c=10;

  if(a>b)
```


Copyright Virtual University of Pakistan

```
    cout<<"a is greater"<<endl;

  if(a<b)
    cout<<"a is smaller"<<endl;

  if(a<=c)
    cout<<"a is less than/equal to c"<<endl;

  if(a>=c)
    cout<<"a is less than/equal to c"<<endl;
  }
```

Output

```
a is smaller
a is less than/equal to c
a is greater than/equal to c
```

In C++ Relational operators, two operators that is = = (Is Equal to) and != (is Not Equal To), are used to check whether the two variables to be compared are equal or not.

Let us take one example which demonstrate these two operators.

```
#include<iostream>
using namespace std;
void main()
{
  int num1 = 30;
  int num2 = 40;
  int num3 = 40;

  if(num1!=num2)
    cout<<"num1 Is Not Equal To num2"<<endl;

  if(num2==num3)
    cout<<"num2 Is Equal To num3"<<endl;

  }
```

Output

```
num1 Is Not Equal To num2
num2 Is Equal To num3
```

NOTE THAT PROGRAM CAN COME IN SHORT AND LONG

Copyright Virtual University of Pakistan

Module 125

125. Programming Languages: Logical Operators Examples

1. C-language Logical Operators are used if we want to compare more than one condition.
2. Depending upon the requirement, proper logical operator is used.
3. Following table shows us the different C++ operators available

Operators	Name of the Operator	Type
&&	AND Operator	Binary
||	OR Operator	Binary
!	NOT Operator	Unary

According to names of the Logical Operators, the condition satisfied in following situation and expected outputs are given

Operator	Output
AND	Output is 1 only when conditions on both sides of Operator become True
OR	Output is 0 only when conditions on both sides of Operator become False
NOT	It gives inverted Output

Let us look at all logical operators with example-

```
#include<iostream>
using namespace std;

int main()
{
  int num1=30;
  int num2=40;

  if(num1>=40 || num2>=40)
    cout<<"OR If Block Gets Executed"<<endl;

  if(num1>=20 && num2>=20)
    cout<<"AND If Block Gets Executed"<<endl;

  if(!(num1>=40))
    cout<<"NOT If Block Gets Executed"<<endl;

  return 0;
}
```

Output:

```
OR If Block Gets Executed
```


Copyright Virtual University of Pakistan

```
AND If Block Gets Executed
NOT If Block Gets Executed
```

Explanation of the Program

Or statement gives output = 1 when any of the two condition is satisfied.

```
if(num1>40 || num2>=40)
```

Here in above program , num2=40. So, one of the two conditions is satisfied. So, statement is executed.

For AND operator, output is 1 only when both conditions are satisfied.

```
if(num1>=20 && num2>=20)
```

Thus, in above program, both the conditions are True so if block gets executed.

Truth Table

Operator	1st Condition	2nd Condition	Output
AND	True	True	True
	True	False	False
	False	True	False
	False	False	False
OR	True	True	True
	True	False	True
	False	True	True
	False	False	False
NOT	True	–	False
	False	–	True

NOTE THAT PROGRAM CAN COME IN SHORT AND LONG

Copyright Virtual University of Pakistan

Module 126

126. Software Engineering: Software Engineering Discipline

To appreciate the problems involved in software engineering, it is helpful to select a large complex device (an automobile, a multistory office building, or perhaps a cathedral) and imagine being asked to design it and then to supervise its construction. How can you estimate the cost in time, money, and other resources to complete the project? How can you divide the project into manageable pieces? How can you ensure that the pieces produced are compatible? How can those working on the various pieces communicate? How can you measure progress? How can you cope with the wide range of detail (the selection of the doorknobs, the design of the gargoyles, the availability of blue glass for the stained glass windows, the strength of the pillars, the design of the duct work for the heating system)? Questions of the same scope must be answered during the development of a large software system.

Because engineering is a well-established field, you might think that there is a wealth of previously developed engineering techniques that can be useful in answering such questions. This reasoning is partially true, but it overlooks fundamental differences between the properties of software and those of other fields of engineering. These distinctions have challenged software engineering projects, leading to cost overruns, late delivery of products, and dissatisfied customers. In turn, identifying these distinctions has proven to be the first step in advancing the software engineering discipline.

TAKE CONCEPT FORMCQS

One such distinction involves the ability to construct systems from generic prefabricated components. Traditional fields of engineering have long benefited from the ability to use "off-the-shelf" components as building blocks when constructing complex devices. The designer of a new automobile does not have to design a new engine or transmission but instead uses previously designed versions of these components. Software engineering, however, lags in this regard. In the past, previously designed software components were domain specific that is, their internal design was based on a specific application and thus their use as generic components was limited. The result is that complex software systems have historically been built from scratch. As we will see in this chapter, significant progress is being made in this regard, although more work remains to be done.

Another distinction between software engineering and other engineering disciplines is the lack of quantitative techniques, called **metrics,** for measuring the properties of software. For example, to project the cost of developing a software system, one would like to estimate the complexity of the proposed product, but methods for measuring the "complexity" of software are evasive. Similarly, evaluating the quality of a software product is challenging. In the case of mechanical devices, an important measure of quality is the mean time between failures, which is essentially a measurement of how well a device endures wear. Software, in contrast, does not wear out, so this method of measuring quality is not as applicable in software engineering.

The difficulties involved in measuring software properties in a quantitative manner is one of the reasons that software engineering has struggled to find a rigorous footing in the same sense as mechanical and electrical engineering. Whereas these latter subjects are founded on the established science of physics, software engineering continues to search for its roots.

Thus, research in software engineering is currently progressing on two levels: Some researchers, sometimes called practitioners, work toward developing techniques for immediate application, whereas others, called theoreticians, search for underlying principles and theories on which more stable techniques can someday be constructed. Being based on a subjective foundation, many methodologies developed and promoted by practitioners in the past have been replaced by other approaches that may themselves become obsolete with time. Meanwhile, progress by theoreticians continues to be slow.

Copyright Virtual University of Pakistan

The need for progress by both practitioners and theoreticians is enormous. Our society has become addicted to computer systems and their associated software. Our economy, healthcare, government, law enforcement, transportation, and defense depend on large software systems. Yet there continue to be major problems with the reliability of these systems. Software errors have caused such disasters and near disasters as the rising moon being interpreted as a nuclear attack, a one-day loss of $5 million by the Bank of New York, the loss of space probes, radiation overdoses that have killed and paralyzed, and the simultaneous disruption of telephone communications over large regions.

This is not to say that the situation is all bleak. Much progress is being made in overcoming such problems as the lack of prefabricated components and metrics. Moreover, the application of computer technology to the software development process, resulting in what is called **computer-aided software engineering (CASE),** is continuing to streamline and otherwise simplify the software development process. CASE has led to the development of a variety of computerized systems, known as **CASE tools,** which include project planning systems (to assist in cost estimation, project scheduling, and personnel allocation), project management systems (to assist in monitoring the progress of the development project), documentation tools (to assist in writing and organizing documentation), prototyping and simulation systems (to assist in the development of prototypes), interface design systems (to assist in the development of GUIs), and programming systems (to assist in writing and debugging programs). Some of these tools are little more than the word processors, spreadsheet software, and email communication systems that were originally developed for generic use and adopted by software engineers. Others are quite sophisticated packages designed primarily for the software engineering environment. Indeed, systems known as **integrated development environments (IDEs)** combine tools for developing software (editors, compilers, debugging tools, and so on) into a single, integrated package. Prime examples of such systems are those for developing applications for smartphones. These not only provide the programming tools necessary to write and debug the software but also provide simulators that, by means of graphical displays, allow a programmer to see how the software being developed would actually perform on a phone.

Download More Highlighted Handouts From

VUAnswer.com

Copyright Virtual University of Pakistan

Module 127

127. Software Engineering: Software Life cycle

The most fundamental concept in software engineering is the software life cycle. The software life cycle is shown in Figure 97. This figure represents the fact that once software is developed, it enters a cycle of being used and maintained—a cycle that continues for the rest of the software's life. Such a pattern is common for many manufactured products as well. The difference is that, in the case of other products, the maintenance phase tends to be a repair process, whereas in the case of software, the maintenance phase tends to consist of correcting or updating. Indeed, software moves into the maintenance phase because errors are discovered, changes in the software's application occur that require corresponding changes in the software, or changes made during a previous modification are found to induce problems elsewhere in the software.

Regardless of why software enters the maintenance phase, the process requires that a person (often not the original author) study the underlying program and its documentation until the program, or at least the pertinent part of the program, is understood. Otherwise, any modification could introduce more problems than it solves. Acquiring this understanding can be a difficult task, even when the software is well-designed and documented. In fact, it is often within this phase that a piece of software is discarded under the pretense (too often true) that it is easier to develop a new system from scratch than to modify the existing package successfully.

Experience has shown that a little effort during the development of software can make a tremendous difference when modifications are required.

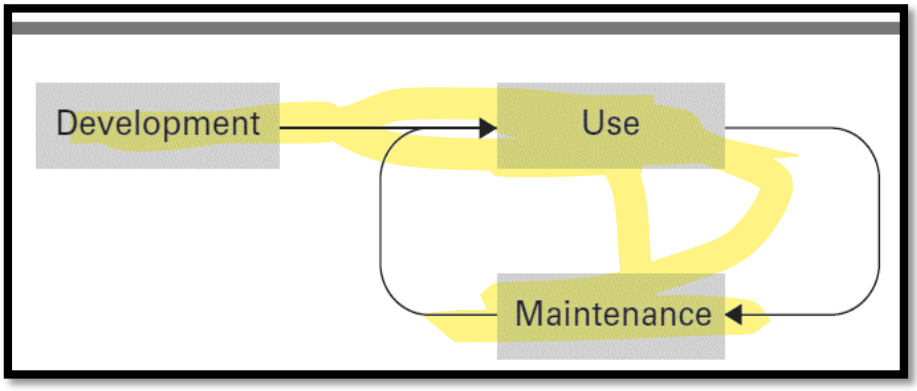


Figure 97: The software life cycle

Copyright Virtual University of Pakistan

Module 128

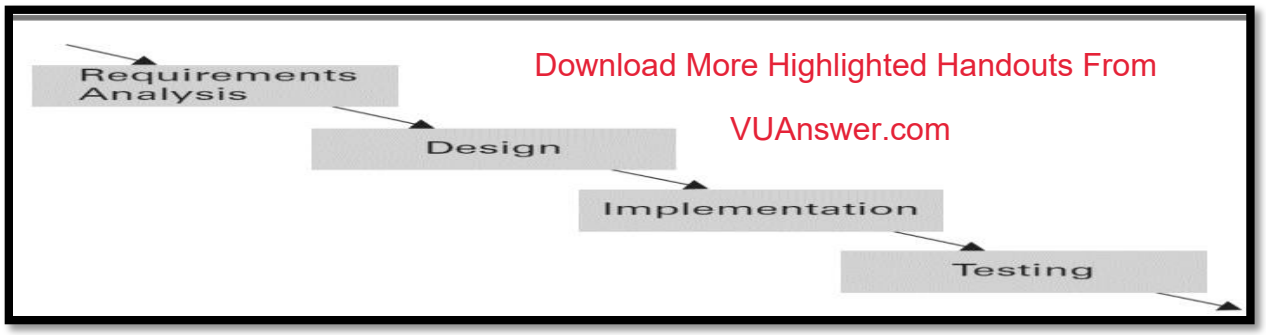
128. Software Engineering: Requirement Analysis Phase

The major steps in the traditional software development life cycle are requirements analysis, design, implementation, and testing (Figure 98).

Requirements Analysis: The software life cycle begins with requirements analysis— the goal of which is to specify what services the proposed system will provide, to identify any conditions (time constraints, security, and so on) on those services, and to define how the outside world will interact with the system.

Requirements analysis involves significant input from the **stakeholders** (future users as well as those with other ties, such as legal or financial interests) of the proposed system. In fact, in cases where the ultimate user is an entity, such as a company or government agency, that intends to hire a software developer for the actual execution of the software project, requirements analysis may start by a feasibility study conducted solely by the user. In other cases, the software developer may be in the business of producing **commercial off-the-shelf (COTS)** software for the mass market, perhaps to be sold in retail stores or downloaded via the Internet. In this setting the user is a less precisely defined entity, and requirements analysis may begin with a market study by the software developer. In any case, the requirements analysis process consists of compiling and analyzing the needs of the software user; negotiating with the project's stakeholders over trade-offs between wants, needs, costs, and feasibility; and finally developing a set of requirements that identify the features and services that the finished software system must have. These requirements are recorded in a document called a **software requirements specification.** In a sense, this document is a written agreement between all parties concerned, which is intended to guide the software's development and provide a means of resolving disputes that may arise later in the development process. The significance of the software requirements specification is demonstrated by the fact that professional organizations such as IEEE and large software clients such as the U.S. Department of Defense have adopted standards for its composition.

From the software developer's perspective, the software requirements specification should define a firm objective toward which the software's development can proceed. Too often, however, the document fails to provide this stability. Indeed, most practitioners in the software engineering field argue that poor communication and changing requirements are the major causes of cost overruns and late product delivery in the software engineering industry. Few customers would insist on major changes to a building's floor plan once the foundation has been constructed, but instances abound of organizations that have expanded, or otherwise altered, the desired capabilities of a software system well after the software's construction was underway. This may have been because a company decided that the system that was originally being developed for only a subsidiary should instead apply to the entire corporation or that advances in technology supplanted the capabilities available during the initial requirements analysis. In any case, software engineers have found that straightforward and frequent communication with the project's stakeholders is mandatory.



Download More Highlighted Handouts From
VUAnswer.com

Figure 98: The traditional development phase of the software life cycle

Copyright Virtual University of Pakistan

Module 129

129. Software Engineering: Design Phase

Whereas requirements analysis provides a description of the proposed software product, design involves creating a plan for the construction of the proposed system. In a sense, requirements analysis is about identifying the problem to be solved, while design is about developing a solution to the problem. From a lay- person's perspective, requirements analysis is often equated with deciding *what* a software system is to do, whereas design is equated with deciding *how* the system will do it. Although this description is enlightening, many software engineers argue that it is flawed because, in actuality, there is a lot of *how* considered during requirements analysis and a lot of *what* considered during design.

It is in the design stage that the internal structure of the software system is established. The result of the design phase is a detailed description of the software system's structure that can be converted into programs.

If the project were to construct an office building rather than a software system, the design stage would consist of developing detailed structural plans for a building that meets the specified requirements. For example, such plans would include a collection of blueprints describing the proposed building at various levels of detail. It is from these documents that the actual building would be constructed. Techniques for developing these plans have evolved over many years and include standardized notational systems and numerous modeling and diagramming methodologies.

Copyright Virtual University of Pakistan

Module 130

130. Software Engineering: Implementation Phase

Implementation involves the actual writing of programs, creation of data files, and development of databases. It is at the implementation stage that we see the distinction between the tasks of a **software analyst** (sometimes referred to as a system analyst) and a **programmer.** The former is a person involved with the entire development process, perhaps with an emphasis on the requirements analysis and design steps. The latter is a person involved primarily with the implementation step. In its narrowest interpretation, a programmer is charged with writing programs that implement the design produced by a software analyst. Having made this distinction, we should note again that there is no central authority controlling the use of terminology throughout the computing community. Many who carry the title of software analyst are essentially programmers, and many with the title programmer (or perhaps senior programmer) are actually software analysts in the full sense of the term. This blurring of terminology is founded in the fact that today the steps in the software development process are often intermingled.

Download More Highlighted Handouts From

VUAnswer.com

Copyright Virtual University of Pakistan

Module 131

131. Software Engineering: Testing Phase

In the traditional development phase of the past, testing was essentially equated with the process of debugging programs and confirming that the final software product was compatible with the software requirements specification. Today, however, this vision of testing is considered far too narrow. Programs are not the only artifacts that are tested during the software development process. Indeed, the result of each intermediate step in the entire development process should be "tested" for accuracy. Testing is now recognized as only one segment in the overall struggle for quality assurance, which is an objective that permeates the entire software life cycle. Thus, many software engineers argue that testing should no longer be viewed as a separate step in software development, but instead it, and its many manifestations, should be incorporated into the other steps, producing a three-step development process whose components might have names such as "requirements analysis *and confirmation,*" "design *and validation,*" and "implementation *and testing.*"

Unfortunately, even with modern quality assurance techniques, large software systems continue to contain errors, even after significant testing. Many of these errors may go undetected for the life of the system, but others may cause major malfunctions. The elimination of such errors is one of the goals of software engineering. The fact that they are still prevalent indicates that a lot of research remains to be done.

Copyright Virtual University of Pakistan

Module 132

132. Software Engineering: Software Engineering Methodologies (I)

Early approaches to software engineering insisted on performing requirements analysis, design, implementation, and testing in a strictly sequential manner. The belief was that too much was at risk during the development of a large software system to allow for variations. As a result, software engineers insisted that the entire requirements specification of the system be completed before beginning the design and, likewise, that the design be completed before beginning implementation. The result was a development process now referred to as the **water- fall model,** an analogy to the fact that the development process was allowed to flow in only one direction.

In recent years, software engineering techniques have changed to reflect the contradiction between the highly structured environment dictated by the water- fall model and the "free-wheeling," trial-and-error process that is often vital to creative problem solving. This is illustrated by the emergence of the **incremental model** for software development. Following this model, the desired software system is constructed in increments, the first being a simplified version of the final product with limited functionality. Once this version has been tested and perhaps evaluated by the future user, more features are added and tested in an incremental manner until the system is complete. For example, if the system being developed is a patient records system for a hospital, the first increment may incorporate only the ability to view patient records from a small sample of the entire record system. Once that version is operational, additional features, such as the ability to add and update records, would be added in a stepwise manner.

Another model that represents the shift away from strict adherence to the waterfall model is the **iterative model,** which is similar to, and in fact sometimes equated with, the incremental model, although the two are distinct. Whereas the incremental model carries the notion of *extending* each preliminary version of a product into a larger version, the iterative model encompasses the concept of *refining* each version. In reality, the incremental model involves an underlying iterative process, and the iterative model may incrementally add features.

A significant example of iterative techniques is the **rational unified process** (**RUP,** rhymes with "cup") that was created by the Rational Software Corporation, which is now a division of IBM. RUP is essentially a software development paradigm that redefines the steps in the development phase of the software life cycle and provides guidelines for performing those steps. These guidelines, along with CASE tools to support them, are marketed by IBM. Today, RUP is widely applied throughout the software industry. In fact, its popularity has led to the development of a nonproprietary version, called the **unified process,** that is available on a noncommercial basis.

Copyright Virtual University of Pakistan

Module 133

133. Software Engineering: Software Engineering Methodologies (II)

Incremental and iterative models sometimes make use of the trend in software development toward **prototyping** in which incomplete versions of the proposed system, called prototypes, are built and evaluated. In the case of the incremental model these prototypes evolve into the complete, final system, a process known as **evolutionary prototyping.** In a more iterative situation, the prototypes may be discarded in favor of a fresh implementation of the final design. This approach is known as **throwaway prototyping.** An example that normally falls within this throwaway category is **rapid prototyping** in which a simple example of the proposed system is quickly constructed in the early stages of development. Such a prototype may consist of only a few screen images that give an indication of how the system will interact with its users and what capabilities it will have. The goal is not to produce a working version of the product but to obtain a demonstration tool that can be used to clarify communication between the parties involved in the software development process. For example, rapid prototypes have proved advantageous in clarifying system requirements during requirements analysis or as aids during sales presentations to potential clients.

A less formal incarnation of incremental and iterative ideas that has been used for years by computer enthusiasts/hobbyists is known as **open-source development.** This is the means by which much of today's free software is produced. Perhaps the most prominent example is the Linux operating system whose open-source development was originally led by Linus Torvalds. The open-source development of a software package proceeds as follows: A single author writes an initial version of the software (usually to fulfill his or her own needs) and posts the source code and its documentation on the Internet. From there it can be downloaded and used by others without charge. Because these other users have the source code and documentation, they are able to modify or enhance the software to fit their own needs or to correct errors that they find. They report these changes to the original author, who incorporates them into the posted version of the software, making this extended version available for further modifications. In practice, it is possible for a software package to evolve through several extensions in a single week.

Perhaps the most pronounced shift from the waterfall model is represented by the collection of methodologies known as **agile methods,** each of which pro- poses early and quick implementation on an incremental basis, responsiveness to changing requirements, and a reduced emphasis on rigorous requirements analysis and design. One example of an agile method is **extreme programming (XP).** Following the XP model, software is developed by a team of less than a dozen individuals working in a communal workspace where they freely share ideas and assist each other in the development project. The software is developed incrementally by means of repeated daily cycles of informal requirements analysis, designing, implementing, and testing. Thus, new expanded versions of the software package appear on a regular basis, each of which can be evaluated by the project's stakeholders and used to point toward further increments. In summary, agile methods are characterized by flexibility, which is in stark contrast to the waterfall model that conjures the image of managers and programmers working in individual offices while rigidly performing well-defined portions of the overall software development task.

The contrasts depicted by comparing the waterfall model and XP reveal the breadth of methodologies that are being applied to the software development process in the hopes of finding better ways to construct reliable software in an efficient manner. Research in the field is an ongoing process. Progress is being made, but much work remains to be done.

Copyright Virtual University of Pakistan

Module 134

134. Software Engineering: Software Engineering Methodologies (II)

A key point is that to modify software one must understand the program or at least the pertinent parts of the program. Gaining such an understanding is often difficult enough in the case of small programs and would be close to impossible when dealing with large software systems if it were not for **modularity** that is, the division of software into manageable units, generically called **modules,** each of which deals with only a part of the software's overall responsibility.

Modules come in a variety of forms. We have already seen that in the context of the imperative paradigm, modules appear as functions. In contrast, the object-oriented paradigm uses objects as the basic modular constituents. These distinctions are important because they determine the underlying goal during the initial software design process. Is the goal to represent the overall task as individual, manageable processes or to identify the objects in the system and understand how they interact?

To illustrate, let us consider how the process of developing a simple modular program to simulate a tennis game might progress in the imperative and the object-oriented paradigms. In the imperative paradigm we begin by considering the actions that must take place. Because each volley begins with a player serving the ball, we might start by considering a function named Serve that (based on the player's characteristics and perhaps a bit of probability) would compute the initial speed and direction of the ball. Next, we would need to determine the path of the ball (Will it hit the net? Where will it bounce?). We might plan on placing these computations in another function named ComputePath. The next step might be to determine if the other player is able to return the ball, and if so, we must compute the ball's new speed and direction. We might plan on placing these computations in a function named Return.

Continuing in this fashion, we might arrive at the modular structure depicted by the **structure chart** shown in Figure 99, in which functions are represented by rectangles and function dependencies (implemented by function calls) are represented by arrows. In particular, the chart indicates that the entire game is overseen by a function named ControlGame, and to perform its task, ControlGame calls on the services of the functions Serve, Return, ComputePath, and UpdateScore.

Note that the structure chart does not indicate how each function is to perform its task. Rather, it merely identifies the functions and indicates the dependencies among the functions. In reality, the function ControlGame might perform its task by first calling the Serve function, then repeatedly calling on the functions ComputePath and Return until one reports a miss, and finally calling on the services of UpdateScore before repeating the whole process by again calling on Serve.

At this stage we have obtained only a very simplistic outline of the desired program, but our point has already been made. In accordance with the imperative paradigm, we have been designing the program by considering the activities that must be performed and are therefore obtaining a design in which the modules are functions.

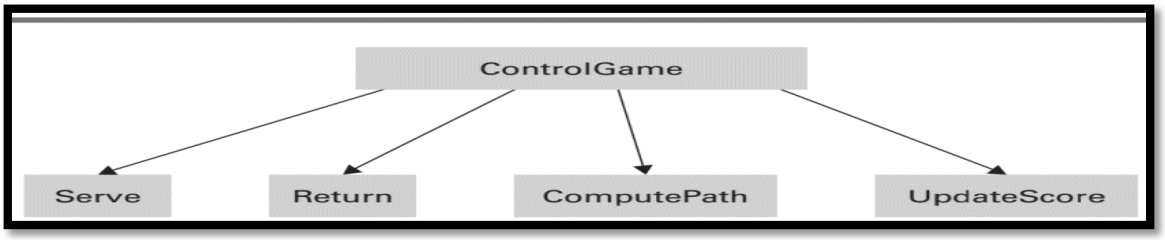


Figure 99: A simple structure chart

Copyright Virtual University of Pakistan

Module 135

135. Software Engineering: Coupling

We have introduced modularity as a way of producing manageable software. The idea is that any future modification will likely apply to only a few of the modules, allowing the person making the modification to concentrate on that portion of the system rather than struggling with the entire package. This, of course, depends on the assumption that changes in one module will not unknowingly affect other modules in the system. Consequently, a goal when designing a modular system should be to maximize independence among modules or, in other words, to minimize the linkage between modules (known as intermodule **coupling**). Indeed, one metric that has been used to measure the complexity of a software system (and thus obtain a means of estimating the expense of maintaining the software) is to measure its intermodule coupling.

Intermodule coupling occurs in several forms. One is **control coupling,** which occurs when a module passes control of execution to another, as in a function call. The structure chart in Figure 99 represents the control coupling that exists between functions. In particular, the arrow from the module ControlGame to Serve indicates that the former passes control to the latter. It is also control coupling that is represented in Figure 100, where the arrows trace the path of control as it is passed from object to object.

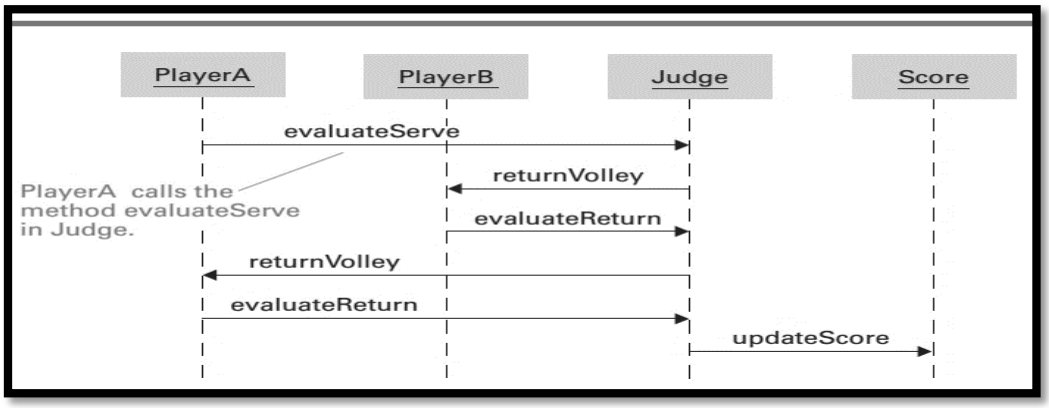


Figure 100: The interaction between objects resulting from PlayerA's serve.

Another form of intermodule coupling is **data coupling,** which refers to the sharing of data between modules. If two modules interact with the same item of data, then modifications made to one module may affect the other, and modifications to the format of the data itself could have repercussions in both modules.

Data coupling between functions can occur in two forms. One is by explicitly passing data from one function to another in the form of parameters. Such coupling is represented in a structure chart by an arrow between the functions that is labeled to indicate the data being passed. The direction of the arrow indicates the direction in which the item is transferred.

Similar data coupling occurs between objects in an object-oriented design. For example, when PlayerA asks the object Judge to evaluate its serve (see Figure 100), it must pass the trajectory information to Judge. On the other hand, one of the benefits of the object-oriented paradigm is that it inherently tends to reduce data coupling between objects to a minimum. This is because the methods within an object tend to include all those functions that manipulate the object's internal data. For example, the object PlayerA will contain information regarding that player's characteristics as well as all the methods that require that information. In turn, there is no need to pass that information to other objects and thus interobject data coupling is minimized.

Copyright Virtual University of Pakistan

In contrast to passing data explicitly as parameters, data can be shared among modules implicitly in the form of **global data,** which are data items that are automatically available to all modules throughout the system, as opposed to local data items that are accessible only within a particular module unless explicitly passed to another. Most high-level languages provide ways of implementing both global and local data, but the use of global data should be employed with caution. The problem is that a person trying to modify a module that is dependent on global data may find it difficult to identify how the module in question interacts with other modules. In short, the use of global data can degrade the module's usefulness as an abstract tool.

Copyright Virtual University of Pakistan

Module 136

136. Software Engineering: Cohesion

Just as important as minimizing the coupling between modules is maximizing the internal binding within each module. The term **cohesion** refers to this internal binding or, in other words, the degree of relatedness of a module's internal parts. To appreciate the importance of cohesion, we must look beyond the initial development of a system and consider the entire software life cycle. If it becomes necessary to make changes in a module, the existence of a variety of activities within it can confuse what would otherwise be a simple process. Thus, in addition to seeking low intermodule coupling, software designers strive for high intramodule cohesion. A weak form of cohesion is known as **logical cohesion.** This is the cohesion within a module induced by the fact that its internal elements perform activities logically similar in nature. For example, consider a module that performs all of a system's communication with the outside world. The "glue" that holds such a module together is that all the activities within the module deal with communication. However, the topics of the communication can vary greatly. Some may deal with obtaining data, whereas others deal with reporting results.

A stronger form of cohesion is known as **functional cohesion,** which means that all the parts of the module are focused on the performance of a single activity. In an imperative design, functional cohesion can often be increased by isolating subtasks in other modules and then using these modules as abstract tools. This is demonstrated in our tennis simulation example (see again Figure 99) where the module ControlGame uses the other modules as abstract tools so that it can concentrate on overseeing the game rather than being distracted by the details of serving, returning, and maintaining the score.

In object-oriented designs, entire objects are usually only logically cohesive because the methods within an object often perform loosely related activities—the only common bond being that they are activities performed by the same object. For example, in our tennis simulation example, each player object contains methods for serving as well as returning the ball, which are significantly different activities. Such an object would therefore be only a logically cohesive module. However, software designers should strive to make each individual method within an object functionally cohesive. That is, even though the object in its entirety is only logically cohesive, each method within an object should perform only one functionally cohesive task (Figure 101).

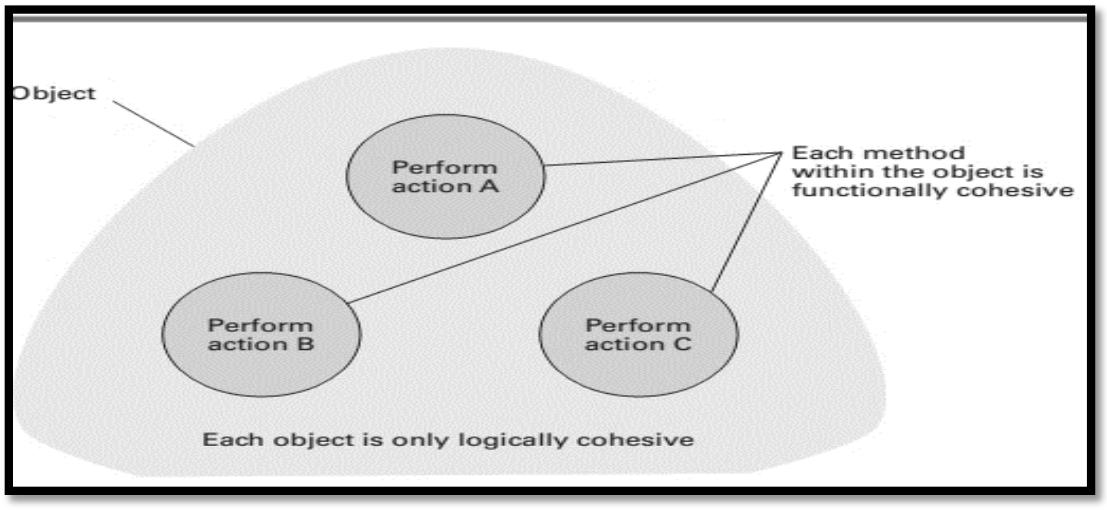


Figure 101: Logical and functional cohesion within an object

Copyright Virtual University of Pakistan

Module 137

137. Software Engineering: Information Hiding

One of the cornerstones of good modular design is captured in the concept of **information hiding,** which refers to the restriction of information to a specific portion of a software system. Here the term *information* should be interpreted in a broad sense, including any knowledge about the structure and contents of a program unit. As such, it includes data, the type of data structures used, encoding systems, the internal compositional structure of a module, the logical structure of a procedural unit, and any other factors regarding the internal properties of a module. The point of information hiding is to keep the actions of modules from having unnecessary dependencies or effects on other modules. Otherwise, the validity of a module may be compromised, perhaps by errors in the development of other modules or by misguided efforts during software maintenance. If, for example, a module does not restrict the use of its internal data from other modules, then that data may become corrupted by other modules. Or, if one module is designed to take advantage of another's internal structure, it could malfunction later if that internal structure is altered.

It is important to note that information hiding has two incarnations—one as a design goal, the other as an implementation goal. A module should be designed so that other modules do not need access to its internal information, and a module should be implemented in a manner that reinforces its boundaries. Examples of the former are maximizing cohesion and minimizing coupling. Examples of the latter involve the use of local variables, applying encapsulation, and using well- defined control structures.

Finally we should note that information hiding is central to the theme of abstraction and the use of abstract tools. Indeed, the concept of an abstract tool is that of a "black box" whose interior features can be ignored by its user, allowing the user to concentrate on the larger application at hand. In this sense then, information hiding corresponds to the concept of sealing the abstract tool in much the same way as a tamperproof enclosure can be used to safeguard complex and potentially dangerous electronic equipment. Both protect their users from the dangers inside as well as protect their interiors from intrusion from their users.

Copyright Virtual University of Pakistan

Module 138

138. Software Engineering: Components

We have already mentioned that one obstacle in the field of software engineering is the lack of prefabricated "off-the-shelf" building blocks from which large software systems can be constructed. The modular approach to software development promises hope in this regard. In particular, the object-oriented programming paradigm is proving especially useful because objects form complete, self-contained units that have clearly defined interfaces with their environments. Once an object, or more correctly a class, has been designed to fulfill a certain role, it can be used to fulfill that role in any program requiring that service. More- over, inheritance provides a means of refining prefabricated object definitions in those cases in which the definitions must be customized to conform to the needs of a specific application. It is not surprising, then, that the object-oriented programming languages C++, Java, and C# are accompanied by collections of prefabricated "templates" from which programmers can easily implement objects for performing certain roles. In particular, C++ is associated with the C++ Standard Template Library, the Java programming environment is accompanied by the Java Application Programmer Interface (API), and C# programmers have access to the .NET Framework Class Library.

The fact that objects and classes have the potential of providing prefabricated building blocks for software design does not mean that they are ideal. One problem is that they provide relatively small blocks from which to build. Thus, an object is actually a special case of the more general concept of a **component**, which is, by definition, a reusable unit of software. In practice, most components are based on the object-oriented paradigm and take the form of a collection of one or more objects that function as a self-contained unit.

Research in the development and use of components has led to the emerging field known as **component architecture** (also known as component-based software engineering) in which the traditional role of a programmer is replaced by a **component assembler** who constructs software systems from prefabricated components that, in many development environments, are displayed as icons in a graphical interface. Rather than be involved with the internal programming of the components, the methodology of a component assembler is to select pertinent components from collections of predefined components and then connect them, with minimal customization, to obtain the desired functionality. Indeed, a property of a well-designed component is that it can be extended to encompass features of a particular application without internal modifications.

An area where component architectures have found fertile ground is in smartphone systems. Due to the resource constraints of these devices, applications are actually a set of collaborating components, each of which provides some discrete function for the application. For example, each display screen within an application is usually a separate component. Behind the scenes, there may exist other service components to store and access information on a memory card, perform some continuous function (such as playing music), or access information over the Internet. Each of these components is individually started and stopped as needed to service the user efficiently; however, the application appears as a seamless series of displays and actions.

Aside from the motivation to limit the use of system resources, the component architecture of smartphones pays dividends in integration between applications. For example, Facebook (a well-known social networking system) when executed on a smartphone may use the components of the contacts application to add all Facebook friends as contacts. Furthermore, the telephony application (the one that handles the functions of the phone), may also access the contacts' components to lookup the caller of an incoming call. Thus, upon receiving a call from a Facebook friend, the friend's picture can be displayed on the phone's screen (along with his or her last Facebook post).

Copyright Virtual University of Pakistan

Module 139

139. Software Engineering: Design Patterns

An increasingly powerful tool for software engineers is the growing collection of design patterns. A **design pattern** is a pre-developed model for solving a recurring problem in software design. For example, the Adapter pattern provides a solution to a problem that often occurs when constructing software from prefabricated modules. In particular, a prefabricated module may have the functionality needed to solve the problem at hand but may not have an interface that is compatible with the current application. In such cases the Adapter pattern provides a standard approach to "wrapping" that module inside another module that translates between the original module's interface and the outside world, thus allowing the original, prefabricated module to be used in the application.

Another well-established design pattern is the Decorator pattern. It provides a means of designing a system that performs different combinations of the same activities depending on the situation at the time. Such systems can lead to an explosion of options that, without careful design, can result in enormously complex software. However, the Decorator pattern provides a standardized way of implementing such systems that leads to a manageable solution.

The identification of recurring problems as well as the creation and cataloging of design patterns for solving them is an ongoing considerations of good design principles such as minimizing coupling and maximizing cohesion play an important role in the development of design patterns.

The results of progress in design pattern development are reflected in the library of tools provided in today's software development packages such as the Java programming environments provided by Oracle and the .NET Framework provided by Microsoft. Indeed, many of the templates found in these tool kits are essentially design pattern skeletons that lead to ready-made, high-quality solutions to design problems.

In closing, we should mention that the emergence of design patterns in software engineering is an example of how diverse fields can contribute to each other. The origins of design patterns lie in the research of Christopher Alexander in traditional architecture. His goal was to identify features that contribute to high-quality architectural designs for buildings or building complexes and then to develop design patterns that incorporated those features. Today, many of his ideas have been incorporated into software design and his work continues to be an inspiration for many software engineers process in software engineering.

Copyright Virtual University of Pakistan

Module 140

140. Software Engineering: Design Patterns Examples

Factory design Pattern:

The Factory Design Pattern (Figure 102) is a commonly used design pattern where we need to create Loosely Coupled System. Basically, it comes under Creational Pattern and it is used to create instance and reuse it. Factory Pattern is based on real time factory concept. As we know, a factory is used to manufacture something as per the requirement and if new items are added in the manufacturing process, the factory starts manufacturing those items as well. Factory class provides abstraction between Client and Car when creating the instance of the Car [Honda, BMW etc] increasingly powerful tool for software engineers is the growing collection of design

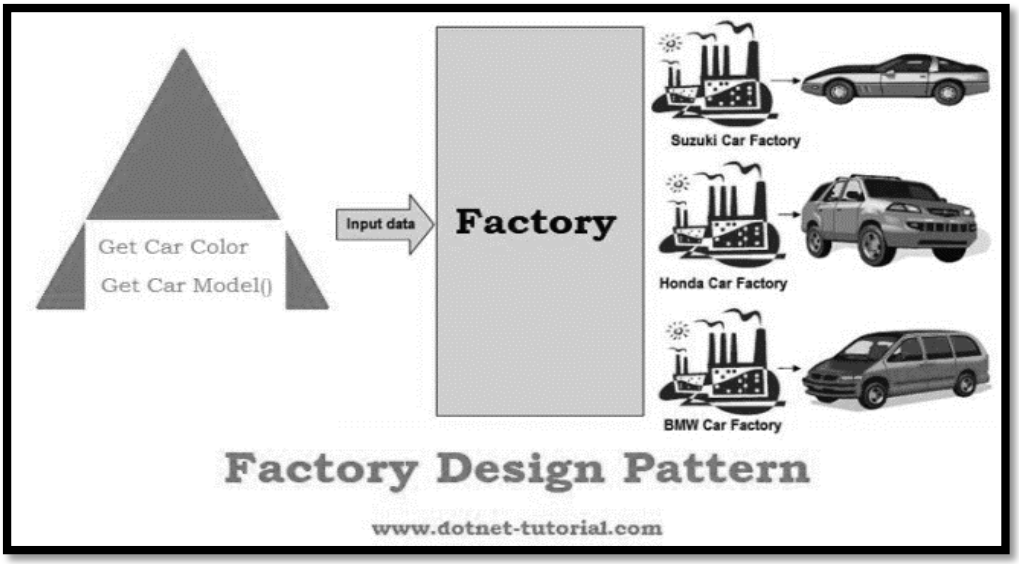


Figure 102: Factory Design Pattern

Shopping Cart Design Pattern:

- It has been shown in the Figure 103.
- Use when the user can possibly buy more than one product.
- Use when the user can possibly buy more than one instance of a product.
- Use when the user may want to return later to carry on shopping
- Use when the user may want to return at a later time to conduct payment
- Do not use when you only have one product to sell.
- Do not use when your site is arranged in a way, so that it does not make sense for the user to buy more than one product at a time (for instance for Application Service Providers (ASPs) allowing a user to upgrade his service).

Copyright Virtual University of Pakistan

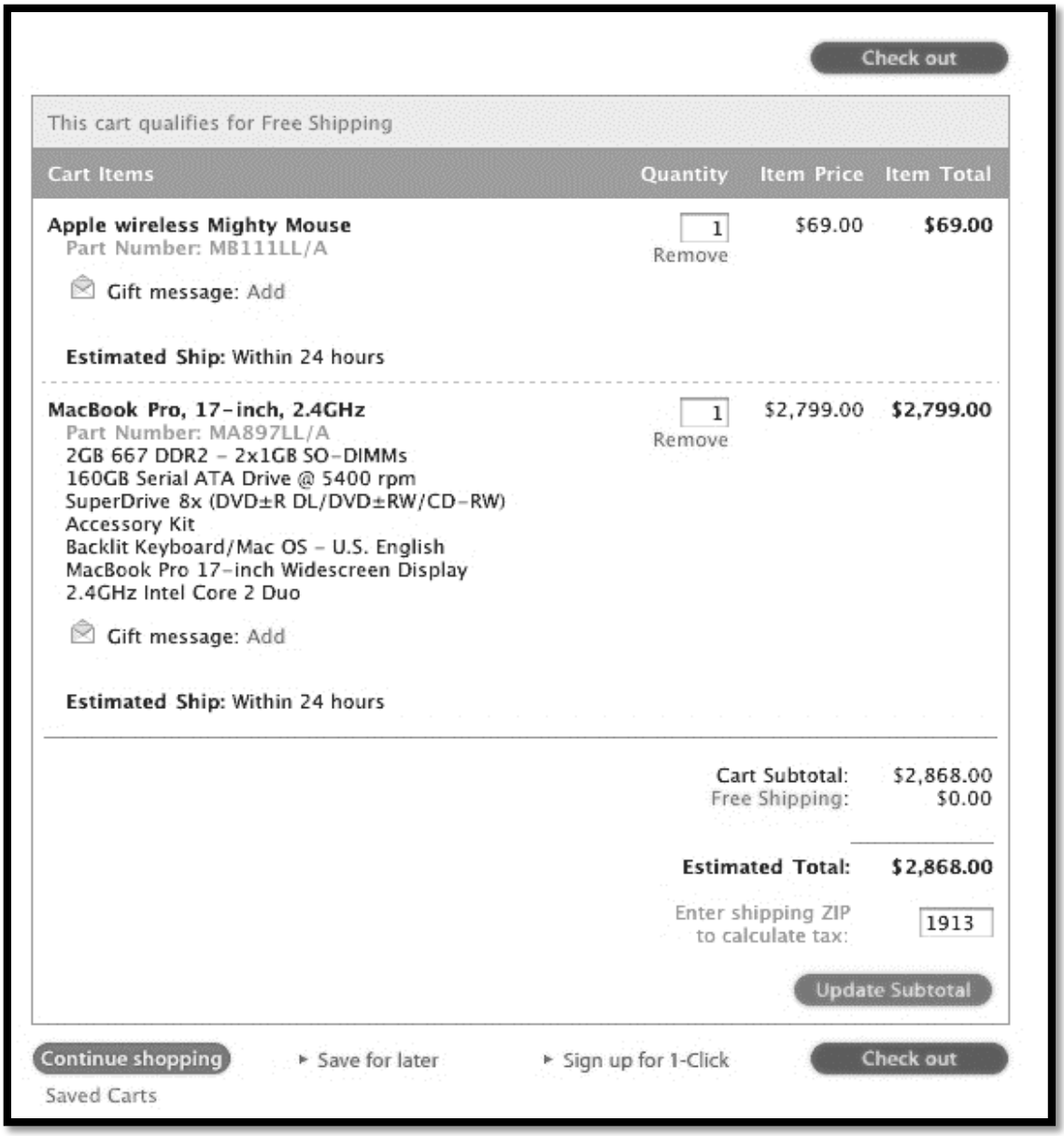


Figure 103: Shopping Cart Design Pattern

Copyright Virtual University of Pakistan

Module 141

141. Software Engineering: Scope of Quality Assurance

In the early years of computing, the problem of producing quality software focused on removing programming errors that occurred during implementation. Later in this section we will discuss the progress that has been made in this direction. However, today, the scope of software quality control extends far beyond the debugging process, with branches including the improvement of software engineering procedures, the development of training programs that in many cases lead to certification, and the establishment of standards on which sound software engineering can be based. In this regard, we have already noted the role of organizations such as ISO, IEEE, and ACM in improving professionalism and establishing standards for assessing quality control within software development companies. A specific example is the ISO 9000 series of standards, which address numerous industrial activities such as design, production, installation, and servicing. Another example is ISO/IEC 15504, which is a set of standards developed jointly by the ISO and the International Electrotechnical Commission (IEC).

Many major software contractors now require that the organizations they hire to develop software meet such standards. As a result, software development companies are establishing **software quality assurance (SQA)** groups, which are charged with overseeing and enforcing the quality control systems adopted by the organization. Thus, in the case of the traditional waterfall model, the SQA group would be charged with the task of approving the software requirements specification before the design stage began or approving the design and its related documents before implementation was initiated.

Several themes underlie today's quality control efforts. One is record keeping. It is paramount that each step in the development process be accurately documented for future reference. However, this goal conflicts with human nature. An issue is the temptation to make decisions or change decisions without updating the related documents. The result is the chance that records will be incorrect and hence their use at future stages will be misleading. Herein lies an important benefit of CASE tools. They make such tasks as redrawing diagrams and updating data dictionaries much easier than with manual methods. Consequently, updates are more likely to be made and the final documentation is more likely to be accurate. (This example is only one of many instances in which software engineering must cope with the faults of human nature. Others include the inevitable personality conflicts, jealousies, and ego clashes that arise when people work together.)

Another quality-oriented theme is the use of reviews in which various parties involved in a software development project meet to consider a specific topic. Reviews occur throughout the software development process, taking the form of requirements reviews, design reviews, and implementation reviews. They may appear as a prototype demonstration in the early stages of requirements analysis, as a structured walkthrough among members of the software design team, or as coordination among programmers who are implementing related portions of the design. Such reviews, on a recurring basis, provide communication channels through which misunderstandings can be avoided and errors can be corrected before they grow into disasters. The significance of reviews is exemplified by the fact that they are specifically addressed in the IEEE Standard for Software Reviews, known as IEEE 1028. IEEE FULL FORM

Some reviews are pivotal in nature. An example is the review between representatives of a project's stakeholders and the software development team at which the final software requirements specification is approved. Indeed, this approval marks the end of the formal requirements analysis phase and is the basis on which the remaining development will progress. However, all reviews are significant, and for the sake of quality control, they should be documented as part of the ongoing record maintenance process.

Copyright Virtual University of Pakistan

Module 142

142. Software Engineering: Software Testing

Whereas software quality assurance is now recognized as a subject that permeates the entire development process, testing and verification of the programs themselves continues to be a topic of research. In module 107, we discussed techniques for verifying the correctness of algorithms in a mathematically rigorous manner but concluded that most software today is "verified" by means of testing. Unfortunately, such testing is inexact at best. We cannot guarantee that a piece of software is correct via testing unless we run enough tests to exhaust all possible scenarios. But, even in simple programs, there may be billions of different paths that could potentially be traversed. Thus, testing all possible paths within a complex program is an impossible task.

On the other hand, software engineers have developed testing methodologies that improve the odds of revealing errors in software with a limited number of tests. One of these is based on the observation that errors in software tend to be clumped. That is, experience has shown that a small number of modules within a large software system tend to be more problematic than the rest. Thus, by identifying these modules and testing them more thoroughly, more of the system's errors can be discovered than if all modules were tested in a uniform, less-thorough manner. This is an instance of the proposition known as the **Pareto principle,** in reference to the economist and sociologist Vilfredo Pareto (1848−1923) who observed that a small part of Italy's population controlled most of Italy's wealth. In the field of software engineering, the Pareto principle states that results can often be increased most rapidly by applying efforts in a concentrated area.

Another software testing methodology, called **basis path testing,** is to develop a set of test data that insures that each instruction in the software is executed at least once. Techniques using an area of mathematics known as graph theory have been developed for identifying such sets of test data. Thus, although it may be impossible to ensure that every path through a software system is tested, it is possible to ensure that every statement within the system is executed at least once during the testing process.

Techniques based on the Pareto principle and basis path testing rely on knowledge of the internal composition of the software being tested. They therefore fall within the category called **glass-box testing,** meaning that the software tester is aware of the interior structure of the software and uses this knowledge when designing the test. In contrast is the category called **black-box testing,** which refers to tests that do not rely on knowledge of the software's interior composition. In short, black-box testing is performed from the user's point of view. In black-box testing, one is not concerned with how the software goes about its task but merely with whether the software performs correctly in terms of accuracy and timeliness.

An example of black-box testing is the technique, called **boundary value analysis,** that consists of identifying ranges of data, called **equivalence classes,** over which the software should perform in a similar manner and then testing the software on data close to the edge of those ranges. For example, if the software is supposed to accept input values within a specified range, then the software would be tested at the lowest and highest values in that range, or if the software is supposed to coordinate multiple activities, then the software would be tested on the largest possible collection of activities. The underlying theory is that by identifying equivalence classes, the number of test cases can be minimized because correct operation for a few examples within an equivalence class tends to validate the software for the entire class. Moreover, the best chance of identifying an error within a class is to use data at the class edges.

Another methodology that falls within the black-box category is **beta testing** in which a preliminary version of the software is given to a segment of the intended audience with the goal of learning how the software performs in real-life situations before the final version of the product is solidified and released to the market. (Similar testing performed at the developer's site is called **alpha testing.**) The advantages of beta testing extend far beyond the traditional discovery of errors. General customer feedback (both positive and negative) is obtained that may assist in refining market strategies. Moreover, early distribution of beta

Copyright Virtual University of Pakistan

software assists other software developers in designing compatible products. For example, in the case of a new operating system for the PC market, the distribution of a beta version encourages the development of compatible utility software so that the final operating system ultimately appears on store shelves surrounded by companion products. Moreover, the existence of beta testing can generate a feeling of anticipation within the marketplace—an atmosphere that increases publicity and sales.

Copyright Virtual University of Pakistan

Module 143

143. Software Engineering: Documentation

A software system is of little use unless people can learn to use and maintain it. Hence, documentation is an important part of a final software package, and its development is, therefore, an important topic in software engineering.

Software documentation serves three purposes, leading to three categories of documentation: user documentation, system documentation, and technical documentation. The purpose of **user documentation** is to explain the features of the software and describe how to use them. It is intended to be read by the user of the software and is therefore expressed in the terminology of the application. Today, user documentation is recognized as an important marketing tool. Good user documentation combined with a well-designed user interface makes a software package accessible and thus increases its sales. Recognizing this, many soft- ware developers hire technical writers to produce this part of their product, or they provide preliminary versions of their products to independent authors so that how-to books are available in bookstores when the software is released to the public. User documentation traditionally takes the form of a physical book or booklet, but in many cases the same information is included as part of the software itself. This allows a user to refer to the documentation while using the software. In this case the information may be broken into small units, sometimes called help packages, that may appear on the display screen automatically if the user dallies too long between commands.

The purpose of **system documentation** is to describe the software's internal composition so that the software can be maintained later in its life cycle. A major component of system documentation is the source version of all the programs in the system. It is important that these programs be presented in a readable format, which is why software engineers support the use of well-designed, high-level programming languages, the use of comment statements for annotating a program, and a modular design that allows each module to be presented as a coherent unit. In fact, most companies that produce software products have adopted conventions for their employees to follow when writing programs. These include indentation conventions for organizing a program on the written page; naming conventions that establish a distinction between names of different program constructs such as variables, constants, objects, and classes; and documentation conventions to ensure that all programs are sufficiently documented. Such conventions establish uniformity throughout a company's software, which ultimately simplifies the software maintenance process.

Another component of system documentation is a record of the design documents including the software requirements specification and records showing how these specifications were obtained during design. This information is useful during software maintenance because it indicates why the software was implemented as it was—information that reduces the chance that changes made during maintenance will disrupt the integrity of the system.

The purpose of **technical documentation** is to describe how a software system should be installed and serviced (such as adjusting operating parameters, installing updates, and reporting problems back to the software's developer). Technical documentation of software is analogous to the documentation provided to mechanics in the automobile industry. This documentation does not discuss how the car was designed and constructed (analogous to system documentation), nor does it explain how to drive the car and operate its heating/cooling system (analogous to user documentation). Instead, it describes how to service the car's components, for example, how to replace the transmission or how to track down an intermittent electrical problem.

The distinction between technical documentation and user documentation is blurred in the PC arena because the user is often the person who also installs and services the software. However, in multiuser environments, the distinction is sharper. Therefore, technical documentation is intended for the system administrator who

Copyright Virtual University of Pakistan

is responsible for servicing all the software under his or her jurisdiction, allowing the users to access the software packages as abstract tools.

Copyright Virtual University of Pakistan

Module 144

144. Software Engineering: Human Machine Interface

One of the tasks during requirements analysis is to define how the proposed software system will interact with its environment. In this section we consider topics associated with this interaction when it involves communicating with humans—a subject with profound significances. After all, humans should be allowed to use a software system as an abstract tool. This tool should be easy to apply and designed to minimize (ideally eliminate) communication errors between the system and its human users. This means that the system's interface should be designed for the convenience of humans rather than merely the expediency of the software system.

The importance of good interface design is further emphasized by the fact that a system's interface is likely to make a stronger impression on a user than any other system characteristic. After all, a human tends to view a system in terms of its usability, not in terms of how cleverly it performs its internal tasks. From a human's perspective, the choice between two competing systems is likely to be based on the systems' interfaces. Thus, the design of a system's interface can ultimately be the determining factor in the success or failure of a software engineering project.

For these reasons, the human-machine interface has become an important concern in the requirements stage of software development projects and is a growing subfield of software engineering. In fact, some would argue that the study of human-machine interfaces is an entire field in its own right.

A beneficiary of research in this field is the smartphone interface. In order to attain the goal of a convenient pocket-sized device, elements of the traditional human-machine interface (full-sized keyboard, mouse, scroll bars, menus) are being replaced with new approaches; such as gestures performed on a touch screen, voice commands, and virtual keyboards with advanced auto-completion of words and phrases. While these represent significant progress, most smartphone users would argue that there is plenty of room for further innovation.

Research in human-machine interface design draws heavily from the areas of engineering called **ergonomics,** which deals with designing systems that harmonize with the physical abilities of humans, and **cognetics,** which deals with designing systems that harmonize with the mental abilities of humans. Of the two, ergonomics is the better understood, largely because humans have been interacting physically with machines for centuries. Examples are found in ancient tools, weaponry, and transportation systems. Much of this history is self-evident; however, at times the application of ergonomics has been counterintuitive. An often-cited example is the design of the typewriter keyboard (now reincarnated as the computer keyboard) in which the keys were intentionally arranged to reduce a typist's speed so that the mechanical system of levers used in the early machines would not jam. Mental interaction with machines, in contrast, is a relatively new phenomenon, and thus, it is cognetics that offers the higher potential for fruitful research and enlightening insights. Often the findings are interesting in their subtlety. For example, humans form habits—a trait that, on the surface, is good because it can increase efficiency. But habits can also lead to errors, even when the design of an interface intentionally addresses the problem. Consider the process of a human asking a typical operating system to delete a file. To avoid unintentional deletions, most interfaces respond to such a request by asking the user to confirm the request—perhaps via a message such as, "Do you really want to delete this file?" At first glance, this confirmation requirement would seem to resolve any problem of unintentional deletions. However, after using the system for an extended period, a human develops the habit of automatically answering the question with "yes." Thus, the task of deleting a file ceases to be a two-step process consisting of a delete command followed by a thoughtful response to a question. Instead, it becomes a one-step "delete-yes" process, meaning that by the time the human realizes that an incorrect delete request has been submitted, the request has already been confirmed and the deletion has occurred.

Copyright Virtual University of Pakistan

The formation of habits may also cause problems when a human is required to use several application software packages. The interfaces of such packages may be similar yet different. Similar user actions may result in different system responses or similar system responses may require different user actions. In these cases habits developed in one application may lead to errors in the other applications.

Another human characteristic that concerns researchers in human-machine interface design is the narrowness of a human's attention, which tends to become more focused as the level of concentration increases. As a human becomes more engrossed in the task at hand, breaking that focus becomes more difficult. In 1972, a commercial aircraft crashed because the pilots became so absorbed with a landing gear problem (actually, with the process of changing the landing gear indicator light bulb) that they allowed the plane to fly into the ground, even though warnings were sounding in the cockpit.

Less critical examples appear routinely in PC interfaces. For example, a "Caps Lock" light is provided on most keyboards to indicate that the keyboard is in "Caps Lock" mode (i.e., the "Caps Lock" key has been pressed). However, if the key is accidentally pressed, a human rarely notices the status of the light until strange characters begin to appear on the display screen. Even then, the user often puzzles over the predicament for a while until realizing the cause of the problem. In a sense, this is not surprising—the light on the keyboard is not in the user's field of view. However, users often fail to notice indicators placed directly in their line of sight. For example, users can become so engaged in a task that they fail to observe changes in the appearance of the cursor on the display screen, even though their task involves watching the cursor.

Still another human characteristic that must be anticipated during interface design is the mind's limited capacity to deal with multiple facts simultaneously. In an article in *Psychological Review* in 1956, George A. Miller reported research indicating that the human mind is capable of dealing with only about seven details at once. Thus, it is important that an interface be designed to present all the relevant information when a decision is required rather than to rely on the human user's memory. In particular, it would be poor design to require that a human remember precise details from previous screen images. Moreover, if an interface requires extensive navigation among screen images, a human can get lost in the maze. Thus, the content and arrangement of screen images becomes an important design issue. Although applications of ergonomics and cognetics give the field of human- machine interface design a unique flavor, the field also encompasses many of the more traditional topics of software engineering. In particular, the search for metrics is just as important in the field of interface design as it is in the more traditional areas of software engineering. Interface characteristics that have been subjected to measurement include the time required to learn an interface, the time required to perform tasks via the interface, the rate of user-interface errors, the degree to which a user retains proficiency with the interface after periods of nonuse, and even such subjective traits as the degree to which users like the interface.

The **GOMS** (rhymes with "Toms") model, originally introduced in 1954, is representative of the search for metrics in the field of human-machine interface design. The model's underlying methodology is to analyze tasks in terms of user goals (such as delete a word from a text), operators (such as click the mouse but- ton), methods (such as double-click the mouse button and press the delete key), and selection rules (such as choose between two methods of accomplishing the same goal). This, in fact, is the origin of the acronym GOMS—goals, operators, methods, and selection rules. In short, GOMS is a methodology that allows the actions of a human using an interface to be analyzed as sequences of elementary steps (press a key, move the mouse, make a decision). The performance of each elementary step is assigned a precise time period, and thus, by adding the times assigned to the steps in a task, GOMS provides a means of comparing different proposed interfaces in terms of the time each would require when performing similar tasks.

Understanding the technical details of systems such as GOMS is not the purpose of our current study. The point in our case is that GOMS is founded on features of human behavior (moving hands, making decisions, and so on). In fact, the development of GOMS was originally considered a topic in psychology. Thus,

Copyright Virtual University of Pakistan

GOMS reemphasizes the role that human characteristics play in the field of human-machine interface design, even in the topics that are carryovers from traditional software engineering.

The design of human-machine interfaces promises to be an active field of research in the foreseeable future. Many issues dealing with today's GUIs are yet unresolved, and a multitude of additional problems lurk in the use of three-dimensional interfaces that are now on the horizon. Indeed, because these interfaces promise to combine audio and tactile communication with three- dimensional vision, the scope of potential problems is enormous.

Copyright Virtual University of Pakistan

Module 145

145. Software Engineering: Software Ownership and Liability

Most would agree that a company or individual should be allowed to recoup, and profit from, the investment needed to develop quality software. Otherwise, it is unlikely that many would be willing to undertake the task of producing the software our society desires. In short, software developers need a level of ownership over the software they produce.

Legal efforts to provide such ownership fall under the category of **intellectual property** law, much of which is based on the well-established principles of copyright and patent law. Indeed, the purpose of a copyright or patent is to allow the developer of a product to release that product (or portions thereof) to intended parties while protecting his or her ownership rights. As such, the developer of a product (whether an individual or a corporation) will assert his or her ownership by including a copyright statement in all produced works; including requirement specifications, design documents, source code, test plans, and in some visible place within the final product. A copyright notice clearly identifies ownership, the personnel authorized to use the work, and other restrictions. Furthermore, the rights of the developer are formally expressed in legal terms in a **software license.**

A software license is a legal agreement between the owner and user of a software product that grants the user certain permissions to use the product without transferring ownership rights to the intellectual property. These agreements spell out, to a fine level of detail, the rights and obligations of both parties. Thus, it is important to carefully read and understand the terms of the software license before installing and using a software product.

While copyrights and software license agreements provide legal avenues to inhibit outright copying and unauthorized use of software, they are generally insufficient to prevent another party from independently developing a product with a nearly identical function. It is sad that over the years there have been many occasions where the developer of a truly revolutionary software product was unable to capitalize fully on his or her invention (two notable examples are spreadsheets and web browsers). In most of these cases, another company was successful in developing a competitive product that secured a dominant share of the market. A legal path to prevent this intrusion by a competitor is found in patent law.

Patent laws were established to allow an inventor to benefit commercially from an invention. To obtain a patent, the inventor must disclose the details of the invention and demonstrate that it is new, useful, and not obvious to others with similar backgrounds (a requirement that can be quite challenging for software). If a patent is granted, the inventor is given the right to prevent others from making, using, selling, or importing the invention for a limited period of time, which is typically 20 years from the date the patent application was filed.

One drawback to the use of patents is that the process to obtain a patent is expensive and time-consuming, often involving several years. During this time a software product could become obsolete, and until the patent is granted the applicant has only questionable authority to exclude others from appropriating the product.

The importance of recognizing copyrights, software licenses, and patents is paramount in the software engineering process. When developing a software product, software engineers often choose to incorporate software from other products, whether it be an entire product, subset of components, or even portions of source code downloaded over the Internet. However, failure to honor intellectual property rights during this process may lead to huge liabilities and consequences. For example, in 2004, a little-known company, NPT Inc., successfully won a lawsuit against Research in Motion (RIM—the makers of the BlackBerry smartphones) for patent infringement of a few key technologies embedded in RIM's email systems. The

Copyright Virtual University of Pakistan

judgment included an injunction to suspend email services to all BlackBerry users in the United States! RIM eventually reached an agreement to pay NPT a total of $612.5 million, thereby averting a shutdown.

Finally, we should address the issue of liability. To protect themselves against liability, software developers often include disclaimers in the software licenses that state the limitations of their liability. Such statements as "In no event will Company X be liable for any damages arising out of the use of this software" are common. Courts, however, rarely recognize a disclaimer if the plaintiff can show negligence on the part of the defendant. Thus, liability cases tend to focus on whether the defendant used a level of care compatible with the product being produced. A level of care that might be deemed acceptable in the case of developing a word processing system may be considered negligent when developing software to control a nuclear reactor. Consequently, one of the best defenses against software liability claims is to apply sound software engineering principles during the software's development, to use a level of care compatible with the software's application, and to produce and maintain records that validate these endeavors.

Copyright Virtual University of Pakistan

Module 146

146. Data Abstraction: Arrays and Aggregates

Recall that an **array** is a "rectangular" block of data whose entries are of the same type. The simplest form of array is the one-dimensional array, a single row of elements with each position identified by an index. A one-dimensional array with 26 elements could be used to store the number of times each alphabet letter occurs in a page of text, for example. A two-dimensional array consists of multiple rows and columns in which positions are identified by pairs of indices—the first index identifies the row associated with the position; the second index identifies the column. An example would be a rectangular array of numbers representing the monthly sales made by members of a sales force—the entries across each row representing the monthly sales made by a particular member and the entries down each column representing the sales by each member for a particular month. Thus, the entry in the third row and first column would represent the sales made by the third salesperson in January.

In contrast to an array, recall that an **aggregate type** is a block of data items that might be of different types and sizes. The items within the block are usually called **fields.** An example of an aggregate type would be the block of data relating to a single employee, the fields of which might be the employee's name (an array of type character), age (of type integer), and skill rating (of type float). Fields in an aggregate type are usually accessed by field name, rather than by a numerical index number.

Module 147

147. Data Abstraction: List, Stacks and Queues

Another basic data structure is a **list,** which is a collection whose entries are arranged sequentially (Figure 104a). The beginning of a list is called the **head** of the list. The other end of a list is called the **tail.**

Almost any collection of data can be envisioned as a list. For example, text can be envisioned as a list of symbols, a two-dimensional array can be envisioned as a list of rows, and music recorded on a CD can be envisioned as a list of sounds. More traditional examples include guest lists, shopping lists, class enrollment lists, and inventory lists. Activities associated with a list vary depending on the situation. In some cases we may need to remove entries from a list, add new entries to a list, "process" the entries in a list one at a time, change the arrangement of the entries in a list, or perhaps search to see if a particular item is in a list. We will investigate such operations later in this chapter.

By restricting the manner in which the entries of a list are accessed, we obtain two special types of lists known as stacks and queues. A **stack** is a list in which entries are inserted and removed only at the head. An example is a stack of books where physical restrictions dictate that all additions and deletions occur at the top (Figure 104b). Following colloquial terminology, the head of a stack is called the **top** of the stack. The tail of a stack is called its **bottom** or **base.** Inserting a new entry at the top of a stack is called **pushing** an entry. Removing an entry from the top of a stack is called **popping** an entry. Note that the last entry placed on a stack will always be the first entry removed—an observation that leads to a stack being known as a **last-in, first-out,** or **LIFO** (pronounced "LIE-foe") structure.

This LIFO characteristic means that a stack is ideal for storing items that must be retrieved in the reverse order from which they were stored, and thus a stack is often used as the underpinning of backtracking activities. (The term **backtracking** refers to the process of backing out of a system in the opposite order from which the system was entered. A classic example is the process of retracing one's steps in order to find one's way out of a forest.) For instance, consider the underlying structure required to support a recursive process. As each new activation is started, the previous activation must be set aside. Moreover, as each activation is completed, the last activation that was set aside must be retrieved. Thus, if the

Copyright Virtual University of Pakistan

activations are pushed on a stack as they are set aside, then the proper activation will be on the top of the stack each time an activation needs to be retrieved.

A **queue** is a list in which the entries are removed only at the head and new entries are inserted only at the tail. An example is a line, or queue, of people waiting to buy tickets at a theater (Figure 104c)—the person at the head of the queue is served while new arrivals step to the rear (or tail) of the queue. A queue is a **first-in, first-out,** or **FIFO** (pronounced "FIE-foe") structure, meaning that the entries are removed from a queue in the order in which they were stored.

Queues are often used as the underlying structure of a buffer, which is a storage area for the temporary placement of data being transferred from one location to another. As the items of data arrive at the buffer, they are placed at the tail of the queue. Then, when it comes time to forward items to their final destination, they are forwarded in the order in which they appear at the head of the queue. Thus, items are forwarded in the same order in which they arrived.

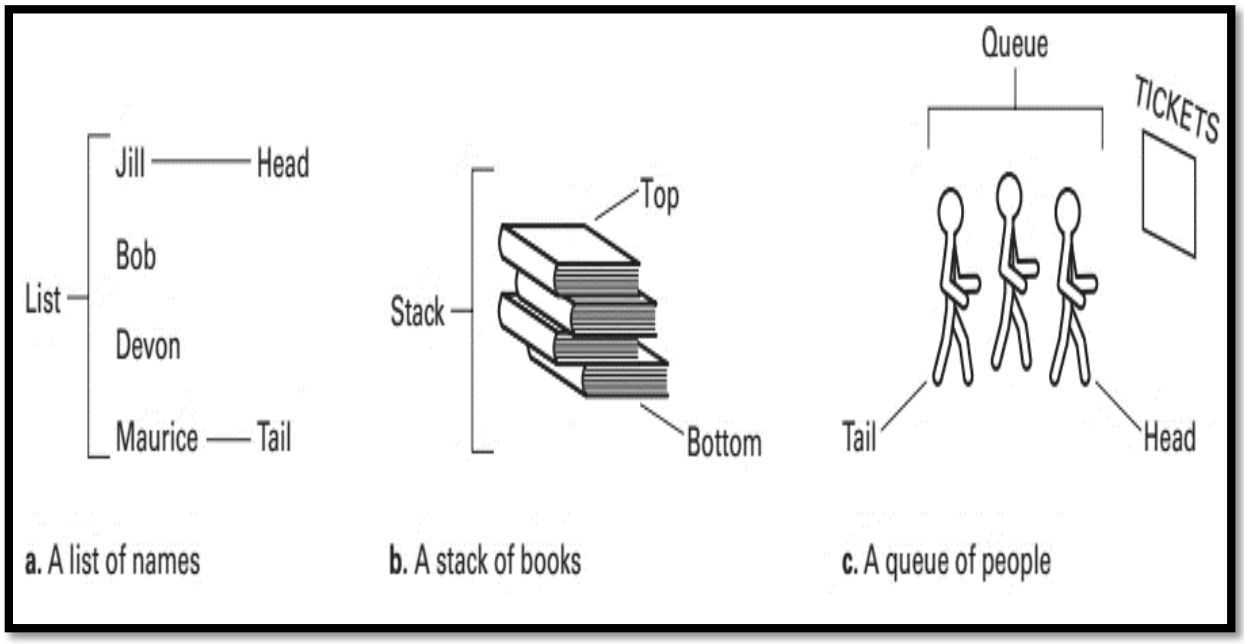


Figure 104: Lists, stacks, and queues

Copyright Virtual University of Pakistan

Module 148

148. Data Abstraction: Trees

A **tree** is a collection whose entries have a hierarchical organization similar to that of an organization chart of a typical company (Figure 105). The president is represented at the top, with lines branching down to the vice presidents, who are followed by regional managers, and so on. To this intuitive definition of a tree structure we impose one additional constraint, which (in terms of an organization chart) is that no individual in the company reports to two different superiors. That is, different branches of the organization do not merge at a lower level. (We have already seen examples of trees in Chapter 6 where they appeared in the form of parse trees.)

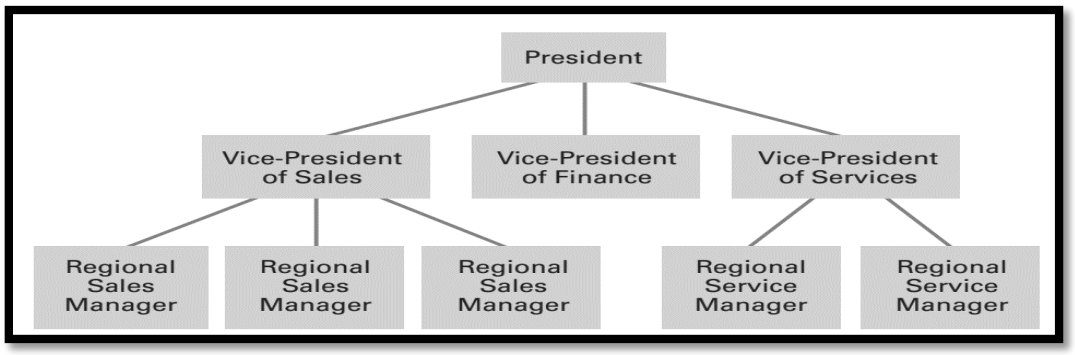


Figure 105: An example of an organization chart

Each position in a tree is called a **node** (Figure 106). The node at the top is called the **root node** (if we turned the drawing upside down, this node would represent the base or root of the tree). The nodes at the other extreme are called **terminal nodes** (or sometimes **leaf nodes**). We often refer to the number of nodes in the longest path from the root to a leaf as the **depth** of the tree. In other words, the depth of a tree is the number of horizontal layers within it.

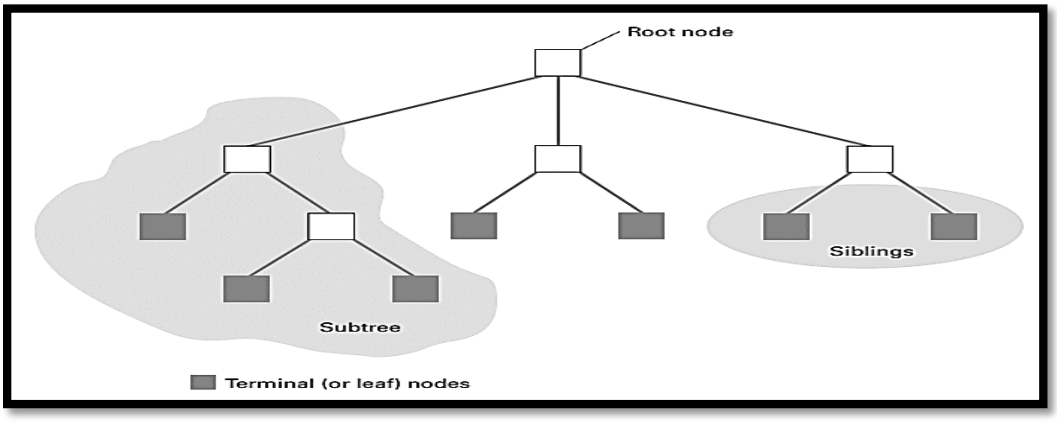


Figure 106: Tree terminology

At times we refer to tree structures as though each node gives birth to those nodes immediately below it. In this sense, we often speak of a node's ancestors or descendants. We refer to its immediate descendants as its **children** and its immediate ancestor as its **parent.** Moreover, we speak of nodes with the same parent as being **siblings.** A tree in which each parent has no more than two children is called a **binary tree.**

Copyright Virtual University of Pakistan

If we select any node in a tree, we find that that node together with the nodes below it also has the structure of a tree. We call these smaller structures **subtrees.** Thus, each child node is the root of a subtree below the child's parent. Each such subtree is called a **branch** from the parent. In a binary tree, we often speak of a node's left branch or right branch in reference to the way the tree is displayed.

Copyright Virtual University of Pakistan

Module 149

149. Data Abstraction: Pointers

Recall that the various cells in a machine's main memory are identified by numeric addresses. Being numeric values, these addresses themselves can be encoded and stored in memory cells. A **pointer** is a storage area that contains such an encoded address. In the case of data structures, pointers are used to record the location where data items are stored. For example, if we must repeatedly move an item of data from one location to another, we might designate a fixed location to serve as a pointer. Then, each time we move the item, we can update the pointer to reflect the new address of the data. Later, when we need to access the item of data, we can find it by means of the pointer. Indeed, the pointer will always "point" to the data.

We have already encountered the concept of a pointer in our study of CPUs in the module 36. There we found that a register called a program counter is used to hold the address of the next instruction to be executed. Thus, the program counter plays the role of a pointer. In fact, another name for a program counter is **instruction pointer.**

As an example of the application of pointers, suppose we have a list of novels stored in a computer's memory alphabetically by title. Although convenient in many applications, this arrangement makes it difficult to find all the novels by a particular author—they are scattered throughout the list. To solve this problem, we can reserve an additional memory cell within each block of cells representing a novel and use this cell as a pointer to another block representing a book by the same author. In this manner the novels with common authorship can be linked in a loop (Figure 8.4). Once we find one novel by a given author, we can find all the others by following the pointers from one book to another.

Many modern programming languages include pointers as a primitive data type. That is, they allow the declaration, allocation, and manipulation of pointers in ways reminiscent of integers and character strings. Using such a language, a programmer can design elaborate networks of data within a machine's memory where pointers are used to link related items to each other.

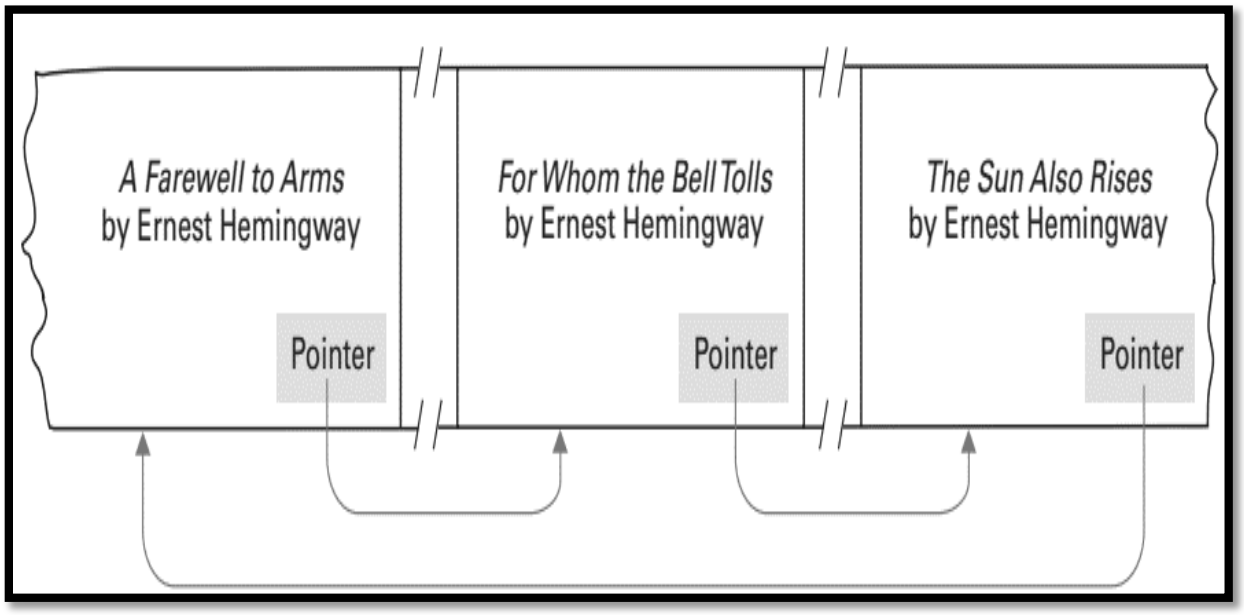


Figure 107: Novels arranged by title but linked according to authorship

Copyright Virtual University of Pakistan

Module 150

150. Database Systems: The Significance of Database Systems

The term **database** refers to a collection of data that is multidimensional in the sense that internal links between its entries make the information accessible from a variety of perspectives. This is in contrast to a traditional file system, sometimes called a **flat file,** which is a one-dimensional storage system, meaning that it presents its information from a single point of view. Whereas a flat file containing information about composers and their compositions might provide a list of compositions arranged by composer, a database might present all the works by a single composer, all the composers who wrote a particular type of music, and perhaps the composers who wrote variations of another composer's work.

Historically, as computing machinery found broader uses in information management, each application tended to be implemented as a separate system with its own collection of data. Payroll was processed using the payroll file, the personnel department maintained its own employee records, and inventory was managed via an inventory file. This meant that much of the information required by an organization was duplicated throughout the company, while many different but related items were stored in separate systems. In this setting, database systems emerged as a means of integrating the information stored and maintained by a particular organization (Figure 108). With such a system, the same sales data could be used to produce restocking orders; create reports on market trends, direct advertisements, and product announcements to customers who are most likely to respond favorably to such information; and generate bonus checks for members of the sales force.

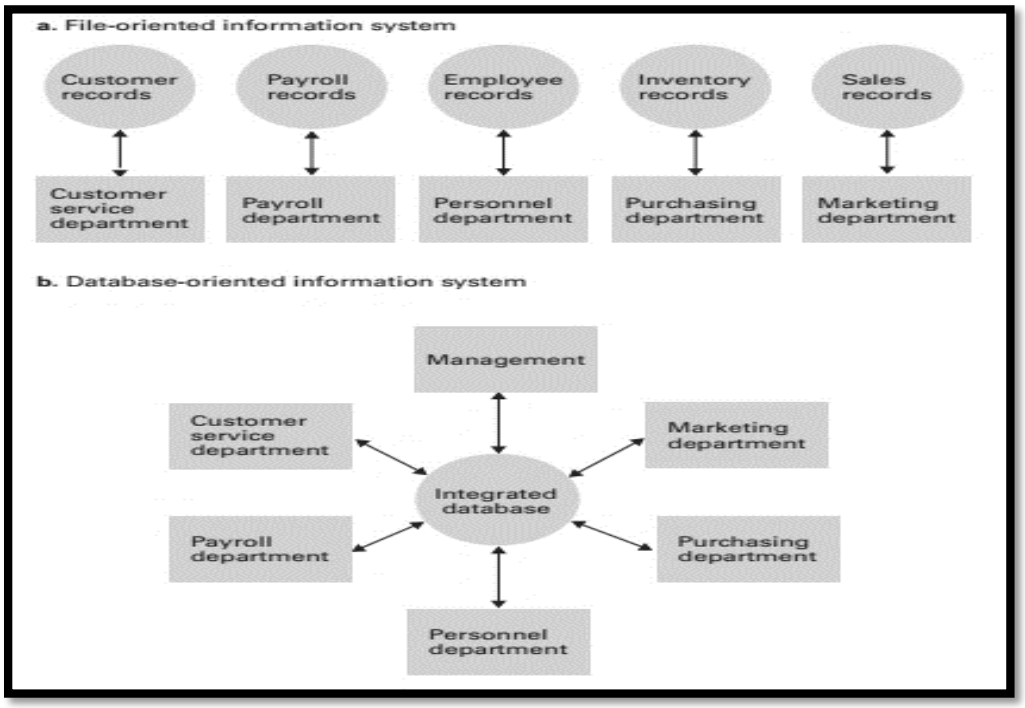


Figure 108: A file versus a database organization

Such integrated pools of information provided a valuable resource with which management decisions could be made, assuming the information could be accessed in a meaningful way. In turn, database research focused on developing techniques by which the information in a database could be brought to the decision-

Copyright Virtual University of Pakistan

making process. Much progress has been made in this regard. Today, database technology, combined with data mining techniques, is an important management tool, allowing the management of an organization to extract pertinent information from enormous amounts of data covering all aspects of the organization and its environment.

Moreover, database systems have become the underlying technology that supports many of the more popular sites on the World Wide Web. The underlying theme of sites such as Google, eBay, and Amazon is to provide an interface between clients and databases. To respond to a client's request, the server interrogates a database, organizes the results in the form of a Web page, and sends that page to the client. Such Web interfaces have popularized a new role for database technology in which a database is no longer a means of storing a company's records but, instead, is the company's product. Indeed, by combining database technology with Web interfaces, the Internet has become a major world-wide information source.

Copyright Virtual University of Pakistan

Module 151

151. Database Systems: Role of Schema

Among the disadvantages of the proliferation of database technology is the potential of sensitive data being accessed by unauthorized personnel. Someone placing an order at a company's website should not have access to the company's financial data; similarly, an employee in a company's benefits department may need access to the company's employee records but should not have access to the corporation's inventory or sales records. Thus, the ability to control access to the information in the database is as important as the ability to share it.

To provide different users access to different information within a database, database systems often rely on schemas and subschemas. A **schema** is a description of the entire database structure that is used by the database software to maintain the database. A **subschema** is a description of only that portion of the database pertinent to a particular user's needs. For example, a schema for a university database would indicate that each student record contains such items as the current address and phone number of that student in addition to the student's academic record. Moreover, it would indicate that each student record is linked to the record of the student's faculty adviser. In turn, the record for each faculty member would contain the person's address, employment history, and so on. Based on this schema, a linkage system would be maintained that ultimately connected the information about a student to the employment history of a faculty member.

To keep the university's registrar from using this linkage to obtain privileged information about the faculty, the registrar's access to the database must be restricted to a subschema whose description of the faculty records does not include employment history. Under this subschema, the registrar could find out which faculty member is a particular student's adviser but could not obtain access to additional information about that faculty member. In contrast, the subschema for the payroll department would provide the employment history of each faculty member but would not include the linkage between students and advisers. Thus the payroll department could modify a faculty member's salary but could not obtain the names of the students advised by that person.

Copyright Virtual University of Pakistan

Module 152

152. Database Systems: Database Management Systems

A typical database application involves multiple software layers, which we will group into two major layers—an application layer and a database management layer (Figure 109). The application software handles the communication with the user of the database and may be quite complex, as exemplified by applications in which users access a database by means of a website. In that case the entire application layer consists of clients throughout the Internet and a server that uses the database to fill the requests from the clients.

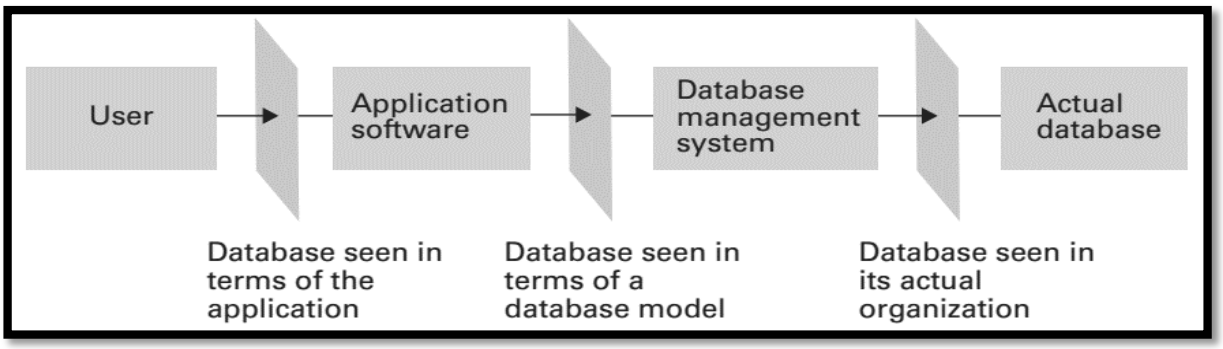


Figure 109: The conceptual layers of a database implementation

Note that the application software does not directly manipulate the data- base. The actual manipulation of the database is accomplished by the **database management system (DBMS).** Once the application software has determined what action the user is requesting, it uses the DBMS as an abstract tool to obtain the results. If the request is to add or delete data, it is the DBMS that actually alters the database. If the request is to retrieve information, it is the DBMS that performs the required searches.

This dichotomy between the application software and the DBMS has sev- eral benefits. One is that it allows for the construction and use of abstract tools, which we have repeatedly found to be a major simplifying concept in software design. If the details of how the database is actually stored are isolated within the DBMS, the design of the application software can be greatly simplified. For instance, with a well-designed DBMS, the application software does not have to be concerned with whether the database is stored on a single machine or scattered among many machines within a network as a **distributed database.** Instead, the DBMS would deal with these issues, allowing the application soft- ware to access the database without concern for where the data is actually stored.

A second advantage of separating the application software from the DBMS is that such an organization provides a means for controlling access to the database. By dictating that the DBMS performs all access to the database, the DBMS can enforce the restrictions imposed by the various subschemas. In particular, the DBMS can use the entire database schema for its internal needs but can require that the application software employed by each user remain within the bounds described by that user's subschema.

Still another reason for separating the user interface and actual data manipulation into two different software layers is to achieve **data independence**—the ability to change the organization of the database itself without changing the application software. For example, the personnel department might need to add an additional field to each employee's record to indicate whether the corresponding employee chose to participate in the company's new health insurance program. If the application software dealt directly with the database, such a change in the data's format could require modifications to all application programs dealing with the database. As a result, the change instigated by the personnel department might cause

Copyright Virtual University of Pakistan

changes to the payroll program as well as to the program for printing mailing labels for the company's newsletter.

The separation between application software and a DBMS removes the need for such reprogramming. To implement a change in the database required by a single user, one needs to change only the overall schema and the subschemas of those users involved in the change. The subschemas of all the other users remain the same, so their application software, which is based on the unaltered subschemas, does not need to be modified.

Copyright Virtual University of Pakistan

Module 153

153. Database Systems: Relational Database model

We have repeatedly seen how abstraction can be used to hide internal complexities. Database management systems provide yet another example. They hide the complexities of a database's internal structure, allowing the user of the database to imagine that the information stored in the database is arranged in a more useful format. In particular, a DBMS contains routines that translate commands stated in terms of a conceptual view of the database into the actions required by the actual data storage system. This conceptual view of the database is called a **database model.**

Relational database model portrays data as being stored in rectangular tables, called **relations,** which are similar to the format in which information is displayed by spreadsheet programs. For example, the relational model allows information regarding the employees of a firm to be represented by a relation such as that shown in Figure 110.

A row in a relation is called a **tuple** (some say "TOO-pul," others say "TUH-pul"). In the relation of Figure 110, tuples consist of the information about a particular employee. Columns in a relation are referred to as **attributes** because each entry in a column describes some characteristic, or attribute, of the entity represented by the corresponding tuple.

Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555
•	•	•	•
•	•	•	•
•	•	•	•

Figure 110: A relation containing employee information

Copyright Virtual University of Pakistan

Module 154

154. Database Systems: Issues of Relational Designs

A pivotal step in designing a relational database is to design the relations making up the database. Although this might appear to be a simple task, many subtleties are waiting to trap the unwary designer.

Suppose that in addition to the information contained in the relation of Figure 110, we want to include information about the jobs held by the employees. We might want to include a job history associated with each employee that consists of such attributes as job title (secretary, office manager, floor supervisor), a job identification code (unique to each job), the skill code associated with each job, the department in which the job exists, and the period during which the employee held the job in terms of a starting date and termination date. (We use an asterisk as the termination date if the job represents the employee's current position.)

One approach to this problem is to extend the relation in Figure 110 to include these attributes as additional columns in the table, as shown in Figure 111. However, close examination of the result reveals several problems. One is a lack of efficiency due to redundancy. The relation no longer contains one tuple for each employee but rather one tuple for each assignment of an employee to a job. If an employee has advanced in the company through a sequence of several jobs, several tuples in the new relation must contain the same information about the employee (name, address, identification number, and Social Security number). For example, the personal information about Baker and Smith is repeated because they have held more than one job. Moreover, when a particular position has been held by numerous employees, the department associated with that job along with the appropriate skill code must be repeated in each tuple representing an assignment of the job. For example, the description of the floor manager job is duplicated because more than one employee has held this position.

Empl Id	Name	Address	SSN	Job Id	Job Title	Skill Code	Dept	Start Date	Term Date
25X15	Joe E. Baker	33 Nowhere St.	111223333	F5	Floor manager	FM3	Sales	9-1-2009	9-30-2010
25X15	Joe E. Baker	33 Nowhere St.	111223333	D7	Dept. head	K2	Sales	10-1-2010	*
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999	F5	Floor manager	FM3	Sales	10-1-2009	*
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555	S25X	Secretary	T5	Personnel	3-1-1999	4-30-2010
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555	S26Z	Secretary	T6	Accounting	5-1-2010	*
•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•

Figure 111: A relation containing redundancy

Another, perhaps more serious, problem with our extended relation surfaces when we consider deleting information from the database. Suppose for example that Joe E. Baker is the only employee to hold the job identified as D7. If he were to leave the company and be deleted from the database represented in Figure 111, we would lose the information about job D7. After all, the only tuple containing the fact that job D7 requires a skill level of K2 is the tuple relating to Joe Baker.

You might argue that the ability to erase only a portion of a tuple could solve the problem, but this would in turn introduce other complications. For instance, should the information relating to job F5 also be retained in a partial tuple, or does this information reside elsewhere in the relation? Moreover, the temptation to use partial tuples is a strong indication that the design of the database can be improved.

Copyright Virtual University of Pakistan

The cause of all these problems is that we have combined more than one concept in a single relation. As proposed, the extended relation in Figure 111 contains information dealing directly with employees (name, identification number, address, Social Security number), information about the jobs available in the company (job identification, job title, department, skill code), and information regarding the relationship between employees and jobs (start date, termination date). On the basis of this observation, we can solve our problems by redesigning the system using three relations—one for each of the preceding categories. We can keep the original relation in Figure 110 (which we now call the EMPLOYEE relation) and insert the additional information in the form of the two new relations called JOB and ASSIGNMENT, which produces the database in Figure 112

Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555

JOB relation

Job Id	Job Title	Skill Code	Dept
S25X	Secretary	T5	Personnel
S26Z	Secretary	T6	Accounting
F5	Floor manager	FM3	Sales
•	•	•	•
•	•	•	•
•	•	•	•

ASSIGNMENT relation

Empl Id	Job Id	Start Date	Term Date
23Y34	S25X	3-1-1999	4-30-2010
34Y70	F5	10-1-2009	*
23Y34	S26Z	5-1-2010	*
•	•	•	•
•	•	•	•
•	•	•	•

Figure 112: An employee database consisting of three relations

Copyright Virtual University of Pakistan

Module 155

155. Database Systems: Relational operators

Now that you have a basic understanding of how data can be organized in terms of the relational model, it is time to see how information can be extracted from a database consisting of relations. We begin with a look at some operations that we might want to perform on relations.

At times we need to select certain tuples from a relation. To retrieve the information about an employee, we must select the tuple with the appropriate identification attribute value from the EMPLOYEE relation, or to obtain a list of the job titles in a certain department, we must select the tuples from the JOB relation having that department as their department attribute. The result of such a process is another relation consisting of the tuples selected from the parent relation. The outcome of selecting information about a particular employee results in a relation containing only one tuple from the EMPLOYEE relation. The outcome of selecting the tuples associated with a certain department results in a relation that probably contains several tuples from the JOB relation.

Copyright Virtual University of Pakistan

Module 156

156. Database Systems: Select Operation

One operation we might want to perform on a relation is to select tuples possessing certain characteristics and to place these selected tuples in a new relation. To express this operation, we adopt the syntax

NEW <- SELECT from EMPLOYEE where EmplId = '34Y70'

The semantics of this statement is to create a new relation called NEW containing those tuples (there should be only one in this case) from the relation EMPLOYEE whose EmplId attribute equals 34Y70 (Figure 113).

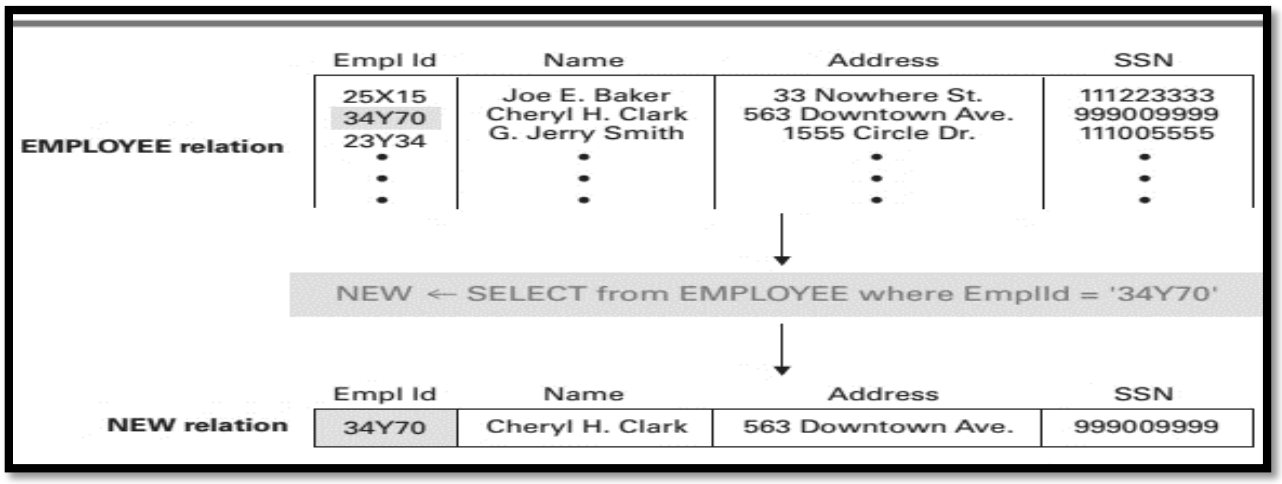


Figure 113: The SELECT operation

Copyright Virtual University of Pakistan

Module 157

157. Database Systems: project Operation

In contrast to the SELECT operation, which extracts rows from a relation, the PROJECT operation extracts columns. Suppose, for example, that in searching for the job titles in a certain department, we had already selected the tuples from the JOB relation that pertained to the target department and placed these tuples in a new relation called NEW1. The list we are seeking is the JobTitle column within this new relation. The PROJECT operation allows us to extract this column (or columns if required) and place the result in a new relation. We express such an operation as

NEW2 <- PROJECT JobTitle from NEW1

The result is the creation of another new relation (called NEW2) that contains the single column of values from the JobTitle column of relation NEW1.

As another example of the PROJECT operation, the statement

MAIL <- PROJECT Name, Address from EMPLOYEE

can be used to obtain a listing of the names and addresses of all employees. This list is in the newly created (two-column) relation called MAIL (Figure 114).

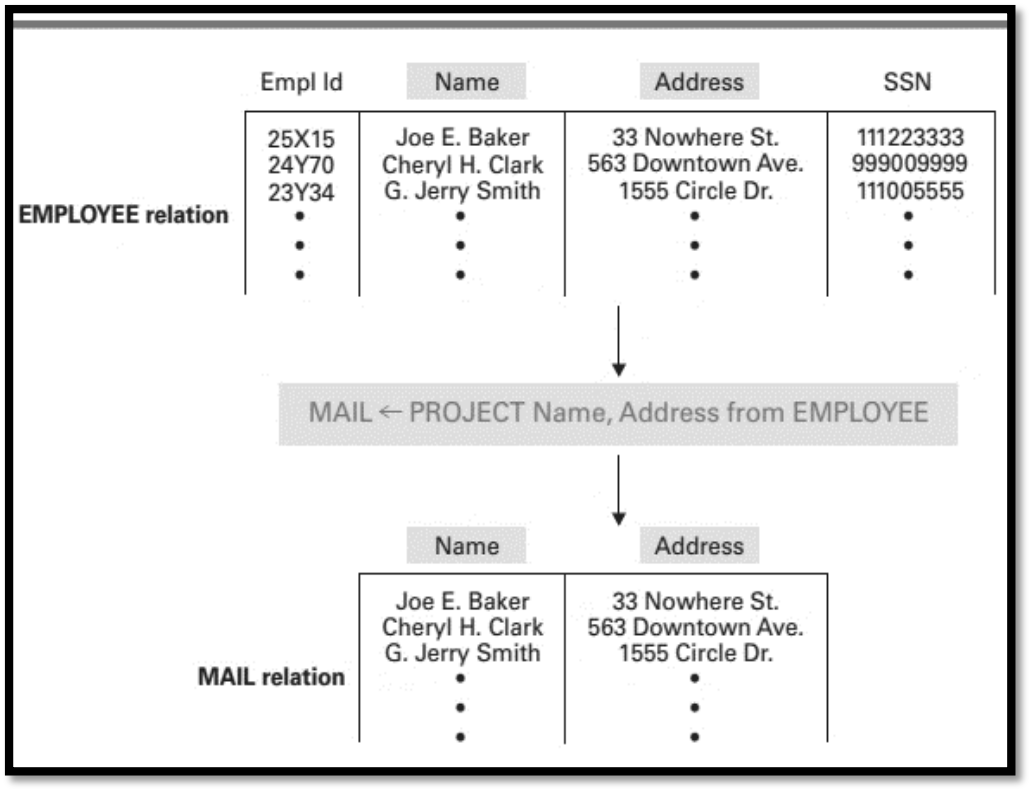


Figure 114: The PROJECT operation

Copyright Virtual University of Pakistan

Module 158

158. Database Systems: Join Operation

Another operation used in conjunction with relational databases is the JOIN operation. It is used to combine different relations into one relation. The JOIN of two relations produces a new relation whose attributes consist of the attributes from the original relations (Figure 115). The names of these attributes are the same as those in the original relations except that each is prefixed by the relation of its origin. (If relation A containing attributes V and W is JOINed with relation B containing attributes X, Y, and Z, then the result has five attributes named A.V, A.W, B.X, B.Y, and B.Z.) This naming convention ensures that the attributes in the new relation have unique names, even though the original relations might have attribute names in common. The tuples (rows) of the new relation are produced by concatenating tuples from the two original relations (see again Figure 115). Which tuples are actually joined to form tuples in the new relation is determined by the condition under which the JOIN is constructed? One such condition is that designated attributes have the same value. This, in fact, is the case represented in Figure 115, where we demonstrate the result of executing the statement

C <- JOIN A and B where A.W = B.X

In this example, a tuple from relation A should be concatenated with a tuple from relation B in exactly those cases where the attributes W and X in the two tuples are equal. Thus the concatenation of the tuple (r, 2) from relation A with the tuple (2, m, q) from relation B appears in the result because the value of attribute W in the first equals the value of attribute X in the second. On the other hand, the result of concatenating the tuple (r, 2) from relation A with the tuple (5, g, p) from relation B does not appear in the final relation because these tuples do not share common values in attributes W and X.

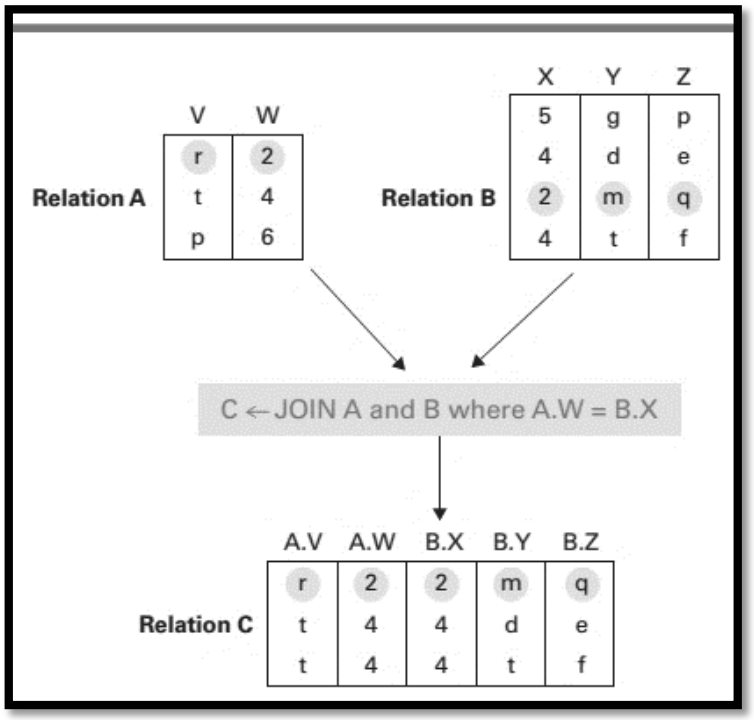


Figure 115: The Join Operation

As another example, Figure 116 represents the result of executing the statement

Copyright Virtual University of Pakistan

C <- JOIN A and B where A.W < B.X

Note that the tuples in the result are exactly those in which attribute W in relation

A is less than attribute X in relation B.

Let us now see how the JOIN operation can be used with the database of Figure 112 to obtain a listing of all employee identification numbers along with the department in which each employee works. Our first observation is that the information required is distributed over more than one relation, and thus the process of retrieving the information must entail more than SELECTions and PROJECTions. In fact, the tool we need is the statement

NEW1 <-JOIN ASSIGNMENT and JOB

where ASSIGNMENT.JobId = JOB.JobId

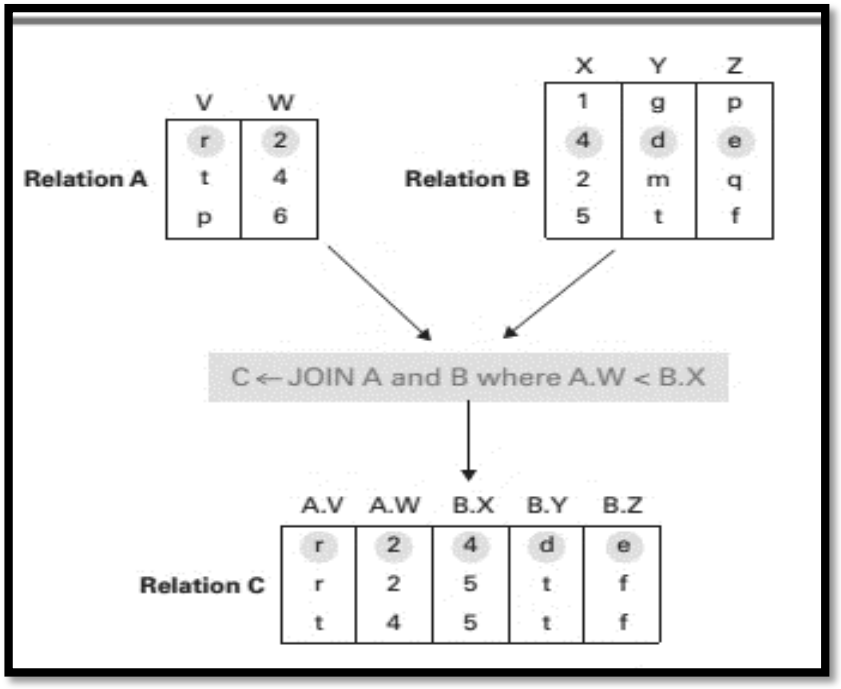


Figure 116: Another example of the JOIN operation

that produces the relation NEW1, as shown in Figure 117. From this relation, our problem can be solved by first SELECTing those tuples in which ASSIGNMENT. TermDate equals '*' (which indicates "still employed") and then PROJECTing the attributes ASSIGNMENT.EmplId and JOB.Dept. In short, the information we need can be obtained from the database in Figure 112 by executing the sequence

NEW1 <-JOIN ASSIGNMENT and JOB

where ASSIGNMENT.JobId = JOB.JobId

NEW2 <- SELECT from NEW1 where ASSIGNMENT.TermDate = '*'

LIST <- PROJECT ASSIGNMENT.EmplId, JOB.Dept from NEW2

Copyright Virtual University of Pakistan

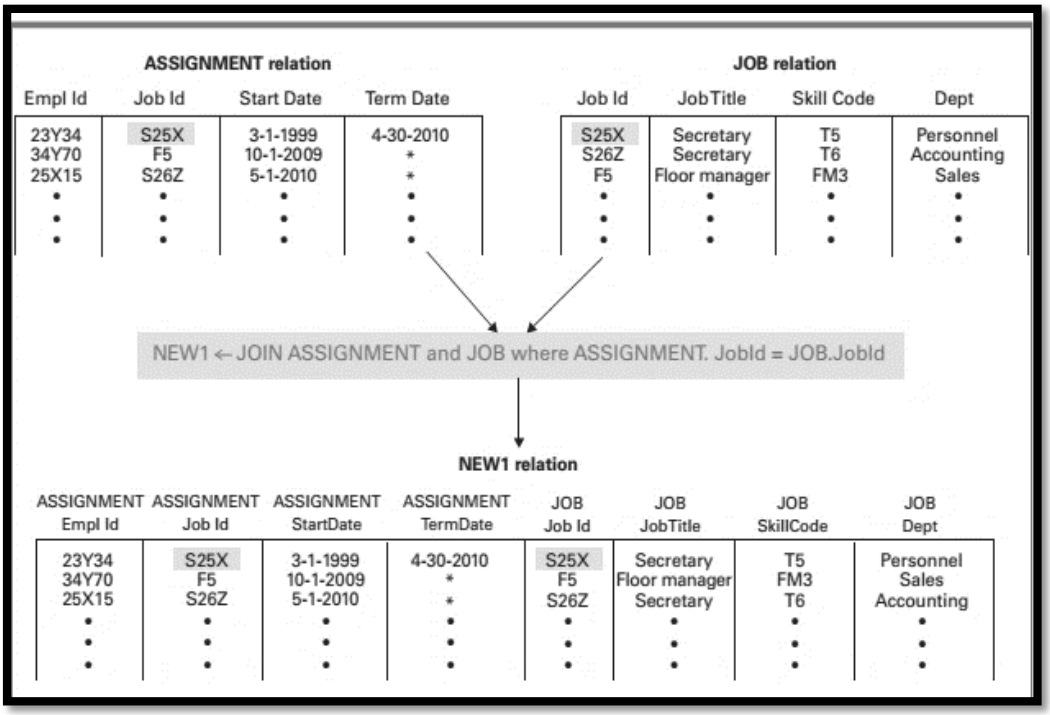


Figure 117: An application of the JOIN operation

Copyright Virtual University of Pakistan

Module 159

159. Database Systems: Object Oriented Databases

Another database model is based on the object-oriented paradigm. This approach leads to an object-oriented database consisting of objects that are linked to each other to reflect their relationships. For example, an object-oriented implementation of the employee database from the previous section could consist of three classes (types of objects): Employee, Job, and Assignment. An object from the Employee class could contain such entries as EmplId, Name, Address, and SSNum; an object from the class Job could contain such entries as JobId, JobTitle, SkillCode, and Dept; and each object from the class Assignment could contain entries such as StartDate and TermDate.

A conceptual representation of such a database is shown in Figure 118 where the links between the various objects are represented by lines connecting the related objects. If we focus on an object of type Employee, we find it linked to a collection of objects of type Assignment representing the various assignments that that particular employee has held. In turn, each of these objects of type Assignment is linked to an object of type Job representing the job associated with that assignment. Thus, all the assignments of an employee can be found by following the links from the object representing that employee. Similarly, all the employees who have held a particular job can be found by following the links from the object representing that job.

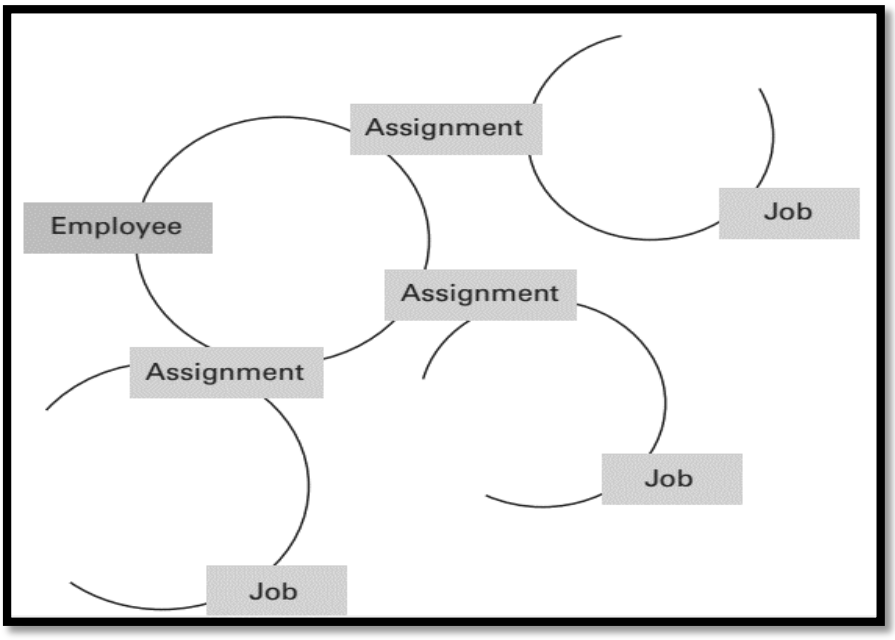


Figure 118: The associations between objects in an object-oriented database

The links between objects in an object-oriented database are normally maintained by the DBMS, so the details of how these links are implemented are not a concern of the programmer writing application software. Instead, when a new object is added to the database, the application software merely specifies the other objects to which it should be linked. The DBMS then creates any linkage system that might be required to record these associations. In particular, a DBMS might link the objects representing the assignments of a given employee in a manner similar to a linked list.

Another task of an object-oriented DBMS is to provide permanent storage for the objects entrusted to it—a requirement that might seem obvious but is inherently distinct from the manner in which objects are normally treated. Normally, when an object-oriented program is executed, the objects created during the program's execution are discarded when the program terminates. In this sense the objects are considered

Copyright Virtual University of Pakistan

transient. But objects that are created and added to a database must be saved after the program that created them terminates. Such objects are said to be **persistent.** Thus, creating persistent objects is a significant departure from the norm.

Proponents of object-oriented databases offer numerous arguments to show why the object-oriented approach to database design is better than the relational approach. One is that the object-oriented approach allows the entire software sys- tem (application software, DBMS, and the database itself) to be designed in the same paradigm. This is in contrast to the historically common practice of using an imperative programming language to develop application software for interrogating a relational database. Inherent in such a task is the clash between imperative and relational paradigms. This distinction is subtle at our level of study, but the difference has been the source of many software errors over the years. Even at our level, we can appreciate that an object-oriented database combined with an objected-oriented application program produces a homogeneous image of objects communicating with each other throughout the system. On the other hand, a relational database combined with an imperative application program conjures an image of two inherently different organizations trying to find a common interface. To appreciate another advantage that object-oriented databases have over their relational counterparts, consider the problem of storing employee names in a relational database. If an entire name is stored as a single attribute in a relation, then inquiries regarding only surnames are awkward. However, if the name is stored as individual attributes, such as first name, middle name, and surname, then the number of attributes becomes problematic because not all names con- form to a specific structure—even when the population is restricted to a single culture. In an object-oriented database, these issues can be hidden within the object that holds the employee's name. An employee's name can be stored as an intelligent object that is capable of reporting the related employee's name in a variety of formats. Thus, from outside these objects, it would be just as easy to deal with only surnames as with entire names, maiden names, or nicknames. The details involved with each perspective would be encapsulated within the objects. This ability to encapsulate the technicalities of different data formats is advantageous in other cases as well. In a relational database, the attributes in a relation are part of the overall design of the database, and thus the types associated with these attributes permeate the entire DBMS. (Variables for temporary storage must be declared to be the appropriate type, and functions for manipulating data of the various types must be designed.) Thus, extending a relational database to include attributes of new types (audio and video) can be problematic. In particular, a variety of functions throughout the database design might need to be expanded to incorporate these new data types. In an object-oriented design, however, the same functions used to retrieve an object representing an employee's name can be used to retrieve an object representing a motion picture because the distinctions in type can be hidden within the objects involved. Thus, the object-oriented approach appears to be more compatible with the construction of multimedia databases—a feature that is already proving to be a great advantage.

Still another advantage the object-oriented paradigm offers to database design is the potential for storing intelligent objects rather than merely data. That is, an object can contain methods describing how it should respond to messages regarding its contents and relationships. For example, each object of the class Employee in Figure 118 could contain methods for reporting and updating the information in the object as well as a method for reporting that employee's job history and perhaps a method for changing that employee's job assignment. Likewise, each object from the Job class could have a method for reporting the specifics of the job and perhaps a method for reporting those employees who have held that particular job. Thus, to retrieve an employee's job history, we would not need to construct an elaborate function. Instead, we could merely ask the appropriate employee object to report its job history. This ability to construct databases whose components respond intelligently to inquiries offers an exciting array of possibilities beyond those of more traditional relational databases.

Copyright Virtual University of Pakistan

Module 160

160. Database Systems: Maintaining DB Integrity

Inexpensive database management systems for personal use are relatively simple systems. They tend to have a single objective—to shield the user from the technical details of the database implementation. The databases maintained by these systems are relatively small and generally contain information whose loss or corruption would be inconvenient rather than disastrous. When a problem does arise, the user can usually correct the erroneous items directly or reload the database from a backup copy and manually make the modifications required to bring that copy up to date. This process might be inconvenient, but the cost of avoiding the inconvenience tends to be greater than the inconvenience itself. In any case, the inconvenience is restricted to only a few people, and any financial loss is generally limited.

In the case of large, multiuser, commercial database systems, however, the stakes are much higher. The cost of incorrect or lost data can be enormous and can have devastating consequences. In these environments, a major role of the DBMS is to maintain the database's integrity by guarding against problems such as operations that for some reason are only partially completed or different operations that might interact inadvertently to cause inaccurate information in the database.

Copyright Virtual University of Pakistan

Module 161

161. Database Systems: The Commit/Rollback Protocol

A single transaction, such as the transfer of funds from one bank account to another, the cancellation of an airline reservation, or the registration of a student in a university course, might involve multiple steps at the database level. For example, a transfer of funds between bank accounts requires that the balance in one account be decremented and the balance in the other be incremented. Between such steps the information in the database might be inconsistent. Indeed, funds are missing during the brief period after the first account has been decremented but before the other has been incremented. Likewise, when reassigning a passenger's seat on a flight, there might be an instant when the passenger has no seat or an instant when the passenger list appears to be one passenger greater than it actually is.

In the case of large databases that are subject to heavy transaction loads, it is highly likely that a random snapshot will find the database in the middle of some transaction. A request for the execution of a transaction or an equipment malfunction will therefore likely occur at a time when the database is in an inconsistent state.

Let us first consider the problem of a malfunction. The goal of the DBMS is to ensure that such a problem will not freeze the database in an inconsistent state. This is often accomplished by maintaining a log containing a record of each transaction's activities in a nonvolatile storage system, such as a magnetic disk. Before a transaction is allowed to alter the database, the alteration to be performed is first recorded in the log. Thus the log contains a permanent record of each transaction's actions.

The point at which all the steps in a transaction have been recorded in the log is called the **commit point.** It is at this point that the DBMS has the information it needs to reconstruct the transaction on its own if that should become necessary. At this point the DBMS becomes committed to the transaction in the sense that it accepts the responsibility of guaranteeing that the transaction's activities will be reflected in the database. In the case of an equipment malfunction, the DBMS can use the information in its log to reconstruct the transactions that have been completed (committed) since the last backup was made.

If problems should arise before a transaction has reached its commit point, the DBMS might find itself with a partially executed transaction that cannot be completed. In this case the log can be used to **roll back** (undo) the activities actually performed by the transaction. In the case of a malfunction, for instance, the DBMS could recover by rolling back those transactions that were incomplete (non-committed) at the time of the malfunction.

Rollbacks of transactions are not restricted, however, to the process of recovering from equipment malfunctions. They are often a part of a DBMS's normal operation. For example, a transaction might be terminated before it has completed all its steps because of an attempt to access privileged information, or it might be involved in a deadlock in which competing transactions find themselves waiting for data being used by each other. In these cases, the DBMS can use the log to roll back a transaction and thus avoid an erroneous database due to incomplete transactions.

To emphasize the delicate nature of DBMS design, we should note that there are subtle problems lurking within the rollback process. The rolling back of one transaction might affect database entries that have been used by other transactions. For example, the transaction being rolled back might have updated an account balance, and another transaction might have already based its activities on this updated value. This might mean that these additional transactions must also be rolled back, which might adversely affect still other transactions. The result is the problem known as **cascading rollback.**

Copyright Virtual University of Pakistan

Module 162

162. Database Systems: Locking

We now consider the problem of a transaction being executed while the database is in a state of flux from another transaction, a situation that can lead to inadvertent interaction between the transactions and produce erroneous results. For instance, the problem known as the **incorrect summary problem** can arise if one transaction is in the middle of transferring funds from one account to another when another transaction tries to compute the total deposits in the bank. This could result in a total that is either too large or too small depending on the order in which the transfer steps are performed. Another possibility is known as the **lost update problem,** which is exemplified by two transactions, each of which makes a deduction from the same account. If one transaction reads the account's current balance at the point when the other has just read the balance but has not yet calculated the new balance, then both transactions will base their deductions on the same initial balance. In turn, the effect of one of the deductions will not be reflected in the database.

To solve such problems, a DBMS could force transactions to execute in their entirety on a one-at-a-time basis by holding each new transaction in a queue until those preceding it have completed. But a transaction often spends a lot of time waiting for mass storage operations to be performed. By inter- weaving the execution of transactions, the time during which one transaction is waiting can be used by another transaction to process data it has already retrieved. Most large database management systems therefore contain a scheduler to coordinate time-sharing among transactions in much the same way that a multiprogramming operating system coordinates interweaving of processes.

To guard against such anomalies as the incorrect summary problem and the lost update problem, these schedulers incorporate a **locking protocol** in which the items within a database that are currently being used by some transaction are marked as such. These marks are called locks; marked items are said to be locked. Two types of locks are common—**shared locks** and **exclusive locks.** They correspond to the two types of access to data that a transaction might require—shared access and exclusive access. If a transaction is not going to alter a data item, then it requires shared access, meaning that other transactions are also allowed to view the data. However, if the transaction is going to alter the item, it must have exclusive access, meaning that it must be the only transaction with access to that data.

In a locking protocol, each time a transaction requests access to a data item, it must also tell the DBMS the type of access it requires. If a transaction requests shared access to an item that is either unlocked or locked with a shared lock, that access is granted, and the item is marked with a shared lock. If, however, the requested item is already marked with an exclusive lock, the additional access is denied. If a transaction requests exclusive access to an item, that request is granted only if the item has no lock associated with it. In this manner, a transaction that is going to alter data protects that data from other transactions by obtaining exclusive access, whereas several transactions can share access to an item if none of them are going to change it. Of course, once a transaction is finished with an item, it notifies the DBMS, and the associated lock is removed.

Various algorithms are used to handle the case in which a transaction's access request is rejected. One algorithm is that the transaction is merely forced to wait until the requested item becomes available. This approach, however, can lead to deadlock, since two transactions that require exclusive access to the same two data items could block each other's progress if each obtains exclusive access to one of the items and then insists on waiting for the other. To avoid such dead- locks, some database management systems give priority to older transactions. That is, if an older transaction requires access to an item that is locked by a younger transaction, the younger transaction is forced to release all of its data items, and its activities are rolled back (based on the log). Then, the older transaction is given access to the item it required, and the younger transaction is forced to start again. If a younger transaction is repeatedly preempted, it will grow older in the process and ultimately become one of the older transactions with high priority. This protocol,

Copyright Virtual University of Pakistan

known as the **wound-wait protocol** (old transactions wound young transactions, young transactions wait for old ones), ensures that every transaction will ultimately be allowed to complete its task.

Copyright Virtual University of Pakistan

Module 163

163. Database Systems: Sequential Files

A **sequential file** is a file that is accessed in a serial manner from its beginning to its end as though the information in the file were arranged in one long row. Examples include audio files, video files, files containing programs, and files containing textual documents. In fact, most of the files created by a typical personal computer user are sequential files. For instance, when a spreadsheet is saved, its information is encoded and stored as a sequential file from which the spreadsheet application software can reconstruct the spreadsheet.

Text files, which are sequential files in which each logical record is a single symbol encoded using ASCII or Unicode, often serve as a basic tool for constructing more elaborate sequential files such as an employee records file. One only needs to establish a uniform format for representing the information about each employee as a string of text, encode the information according to that format, and then record the resulting employee records one after another as one single string of text. For example, one could construct a simple employee file by agreeing to enter each employee record as a string of 31 characters, consisting of a field of 25 characters contain- ing the employee's name (filled with enough blanks to complete the 25-character field), followed by a field of 6 characters representing the employee's identification number. The final file would be a long string of encoded characters in which each 31-character block represents the information about a single employee (Figure 119). Information would be retrieved from the file in terms of logical records consisting of 31-character blocks. Within each of these blocks, individual fields would be identi- fied according to the uniform format with which the blocks were constructed.

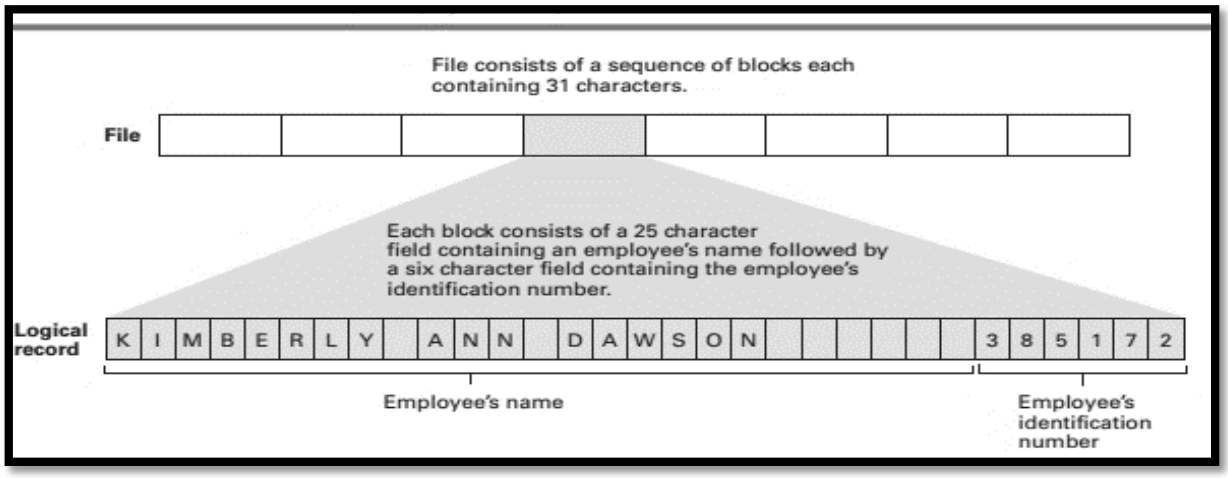


Figure 119: The structure of a simple employee file implemented as a text file

The data in a sequential file must be recorded in mass storage in such a way that the sequential nature of the file is preserved. If the mass storage system is itself sequential (as in the case of a magnetic tape or CD), this is a straightforward undertaking. We need merely record the file on the storage medium according to the sequential properties of the medium. Then processing the file is the task of merely reading and processing the file's contents in the order in which they are found. This is exactly the process followed in playing audio CDs, where the music is stored as a sequential file sector by sector along one continuous spiraling track.

In the case of magnetic disk storage, however, the file would be scattered over different sectors that could be retrieved in a variety of orders. To preserve the proper order, most operating systems (more precisely, the file manager) maintain a list of the sectors on which the file is stored. This list is recorded as part of the

Copyright Virtual University of Pakistan

disk's directory system on the same disk as the file. By means of this list, the operating system can retrieve the sectors in the proper sequence as though the file were stored sequentially, even though the file is actually distributed over various portions of the disk.

Inherent in processing a sequential file is the need to detect when the end of the file is reached. Generically, we refer to the end of a sequential file as the **end-of-file (EOF).** There are a variety of ways of identifying the EOF. One is to place a special record, called a **sentinel,** at the end of the file. Another is to use the information in the operating system's directory system to identify a file's EOF. That is, since the operating system knows which sectors contain the file, it also knows where the file terminates. A classic example involving sequential files is payroll processing in a small company. Here we imagine a sequential file consist- ing of a series of logical records, each of which contains the information about an employee's pay (name, employee identification number, pay scale, and so on) from which checks must be printed on a routine basis. As each employee record is retrieved, that employee's pay is calculated, and the appropriate check produced.

Copyright Virtual University of Pakistan

Module 164

164. Database Systems: Indexed Files

Sequential files are ideal for storing data that will be processed in the order in which the file's entries are stored. However, such files are inefficient when records within the file must be retrieved in an unpredictable order. In such situations what is needed is a way to identify the location of the desired logical record quickly. A popular solution is to use an index for the file in much the same way that an index in a book is used to locate topics within the book. Such a file system is called an indexed file.

An index for a file contains a list of the keys stored in the file along with entries indicating where the record containing each key is stored. Thus to find a particular record, one finds the identifying key in the index and then retrieves the block of information stored at the location associated with that key.

A file's index is normally stored as a separate file on the same mass storage device as the indexed file. The index is usually transferred to main memory before file processing begins so that it is easily accessible when access to records in the file is required (Figure 120).

A classic example of an indexed file occurs in the context of maintaining employee records. Here an index can be used to avoid lengthy searches when you are retrieving an individual record. In particular, if the file of employee records is indexed by employee identification numbers, then an employee's record can be retrieved quickly when the employee's identification number is known. Another example is found on audio CDs where an index is used to allow relatively quick access to individual recordings.

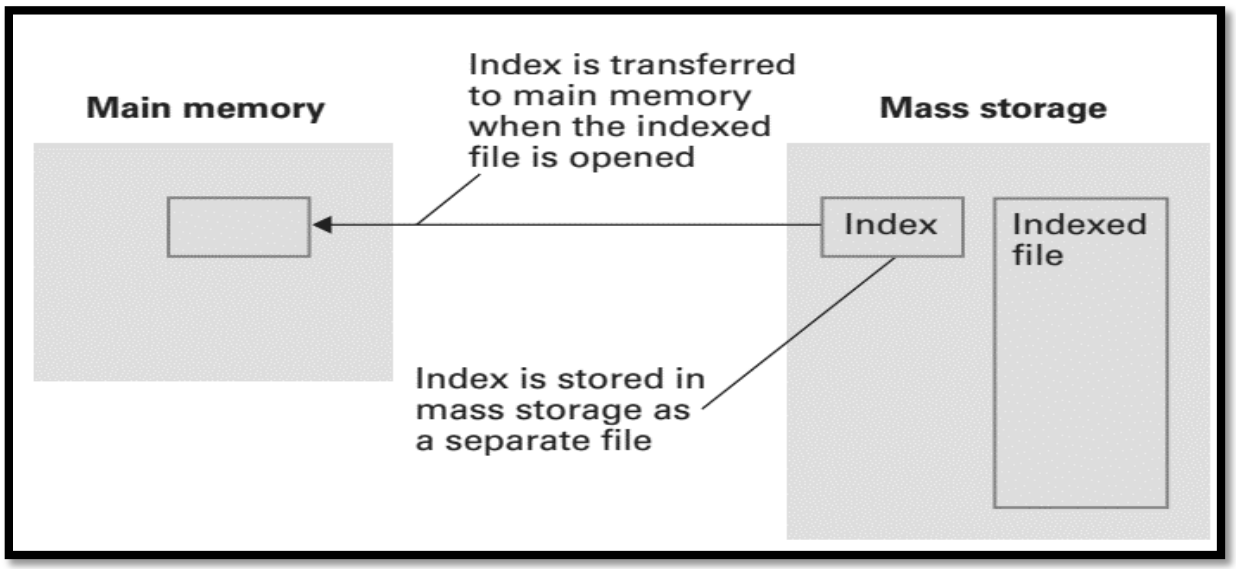


Figure 120: Opening an indexed file

Over the years numerous variations of the basic index concept have been used. One variation constructs an index in a hierarchical manner so that the index takes on a layered or tree structure. A prominent example is the hierarchical directory system used by most operating systems for organizing file storage. In such a case, the directories, or folders, play the role of indexes, each containing links to its sub-indexes. From this perspective, the entire file system is merely one large indexed file.

Copyright Virtual University of Pakistan

Module 165

165. Database Systems: Hash Files

Although indexing provides relatively quick access to entries within a data storage structure, it does so at the expense of index maintenance. **Hashing** is a technique that provides similar access without such overhead. As in the case of an indexed system, hashing allows a record to be located by means of a key value. But, rather than looking up the key in an index, hashing identifies the location of the record directly from the key.

A hash system can be summarized as follows: The data storage space is divided into several sections, called **buckets,** each of which is capable of holding several records. The records are dispersed among the buckets according to an algorithm that converts key values into bucket numbers. (This conversion from key values to bucket numbers is called a **hash function.**) Each record is stored in the bucket identified by this process. Therefore, a record that has been placed in the storage structure can be retrieved by first applying the hash function to the record's identifying key to determine the appropriate bucket, then retrieving the contents of that bucket, and finally searching through the data retrieved for the desired record.

Hashing is not only used as a means of retrieving data from mass storage but also as a means of retrieving items from large blocks of data stored in main memory. When hashing is applied to a storage structure in mass storage, the result is called a **hash file.** When applied to a storage structure within main memory, the result is usually called a **hash table.**

Copyright Virtual University of Pakistan

Module 166

166. Database Systems: Hash File Example

Let us apply the hashing technique to the classic employee file in which each record contains information about a single employee in a company. First, we establish several available areas of mass storage that will play the role of buckets. The number of buckets and the size of each bucket are design decisions that we will consider later. For now, let us assume that we have created 41 buckets, which we refer to as bucket number 0, bucket number 1, through bucket number 40. (The reason we selected 41 buckets rather than an even 40 will be explained shortly.)

Let us assume that an employee's identification number will be used as the key for identifying the employee's record. Our next task, then, is to develop a hash function for converting these keys into bucket numbers. Although the employee identification "numbers" might have the form 25X3Z or J2X35 and are therefore not numeric, they are stored as bit patterns, and we can interpret the bit patterns as numbers. Using this numeric interpretation, we can divide any key by the number of buckets available and record the remainder, which in our case will be an integer in the range from 0 to 40. Thus we can use the remainder of this division process to identify one of the 41 buckets (Figure 121).

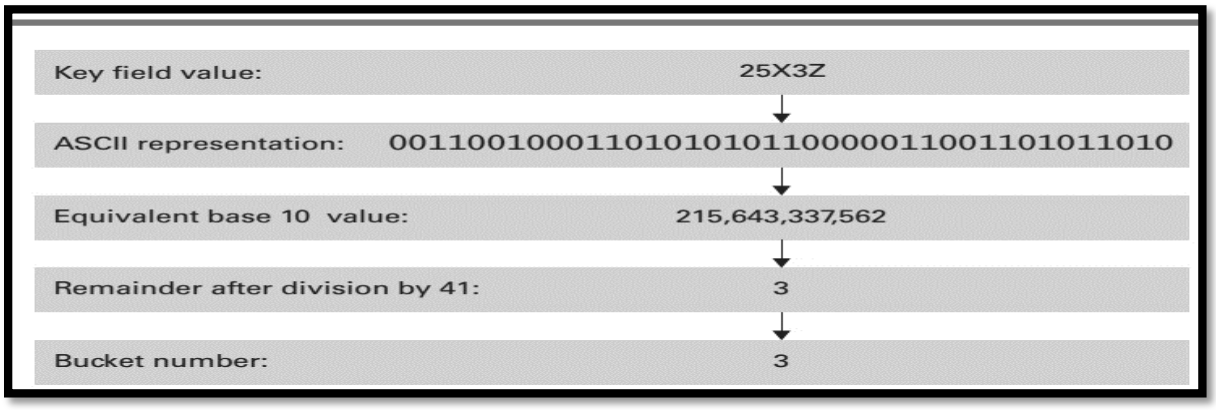


Figure 121: Hashing the key field value 25X3Z to one of 41 buckets

Using this as our hash function, we proceed to construct the file by considering each record individually, applying our divide-by-41 hash function to its key to obtain a bucket number, and then storing the record in that bucket (Figure 122). Later, if we need to retrieve a record, we need merely apply our hash function to the record's key to identify the appropriate bucket and then search that bucket for the record in question.

At this point let us reconsider our decision to divide the storage area into 41 buckets. First, note that to obtain an efficient hash system, the records being stored should be distributed evenly among the buckets. If a disproportionate num- ber of keys happen to hash to the same bucket (a phenomenon called **clustering**), then a disproportionate number of records will be stored in a single bucket. In turn, retrieving a record from that bucket could require a time-consuming search, causing the loss of any benefits gained by hashing.

Now observe that if we had chosen to divide the storage area into 40 buckets rather than 41, our hash function would have involved dividing the keys by the value 40 rather than 41. But, if a dividend and a divisor both have a common factor, that factor will be present in the remainder as well. In particular, if the keys to the entries stored in our hash file happened to be multiples of 5 (which is also a divisor of 40), then the factor of 5 would appear in the remainders when divided by 40, and the entries would cluster in those buckets associated with the remainders 0, 5, 10, 15, 20, 25, 30, and 35. Similar situations would occur in the case of keys that are multiples of 2, 4, 8, 10, and 20, because they are all also fac- tors of 40.

Copyright Virtual University of Pakistan

Consequently, we choose to divide the storage area into 41 buckets because the choice of 41, being a prime number, eliminated the possibility of common factors and therefore reduced the chance of clustering.

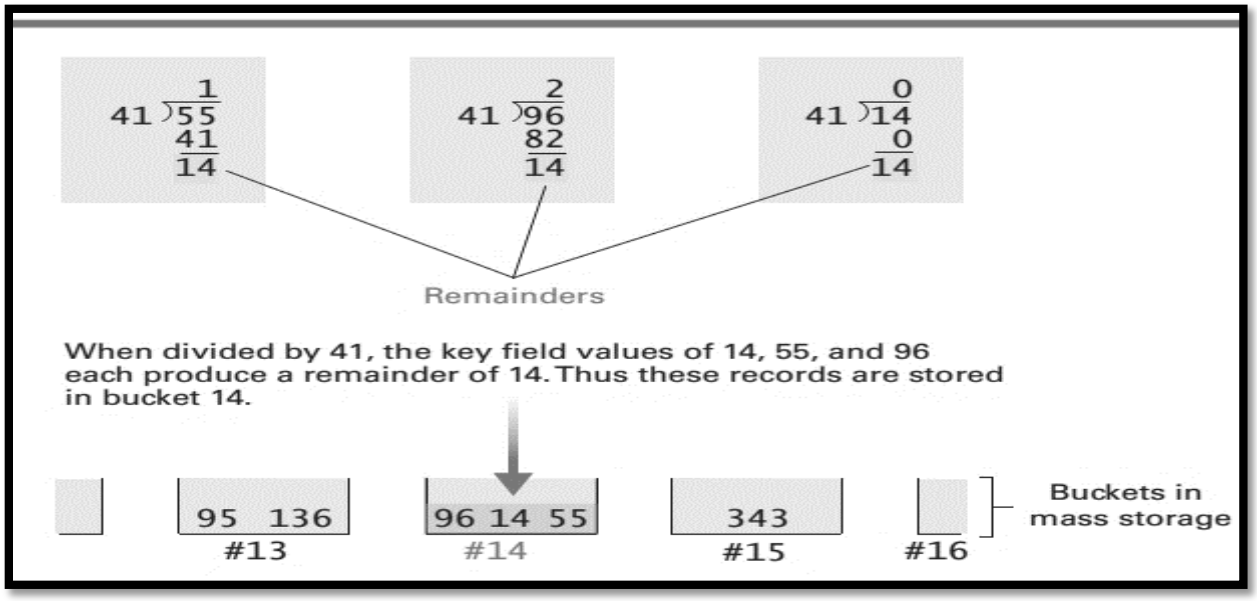


Figure 122: The rudiments of a hashing system

Copyright Virtual University of Pakistan

Module 167

167. Database Systems: Data Mining

A rapidly expanding subject that is closely associated with database technology is data mining, which consists of techniques for discovering patterns in collections of data. Data mining has become an important tool in numerous areas including marketing, inventory management, quality control, loan risk management, fraud detection, and investment analysis. Data mining techniques even have applications in what might seem unlikely settings as exemplified by their use in identifying the functions of particular genes encoded in DNA molecules and characterizing properties of organisms.

Data mining activities differ from traditional database interrogation in that data mining seeks to identify previously unknown patterns as opposed to traditional database inquiries that merely ask for the retrieval of stored facts. More- over, data mining is practiced on static data collections, called **data warehouses,** rather than "online" operational databases that are subject to frequent updates. These warehouses are often "snapshots" of databases or collections of databases. They are used in lieu of the actual operational databases because finding patterns in a static system is easier than in a dynamic one.

We should also note that the subject of data mining is not restricted to the domain of computing but has tentacles that extend far into statistics. In fact, many would argue that since data mining had its origins in attempts to perform statistical analysis on large, diverse data collections, it is an application of statistics rather than a field of computer science.

Two common forms of data mining are **class description** and **class discrimination.** Class description deals with identifying properties that characterize a given group of data items, whereas class discrimination deals with identifying properties that divide two groups. For example, class description techniques would be used to identify characteristics of people who buy small economical vehicles, whereas class discrimination techniques would be used to find proper- ties that distinguish customers who shop for used cars from those who shop for new ones.

Copyright Virtual University of Pakistan

Module 168

168. Database Systems: Data Mining Examples and Implications

Another form of data mining is **cluster analysis,** which seeks to discover classes. Note that this differs from class description, which seeks to discover properties of members within classes that are already identified. More precisely, cluster analysis tries to find properties of data items that lead to the discovery of groupings. For example, in analyzing information about people's ages who have viewed a particular motion picture, cluster analysis might find that the customer base breaks down into two age groups—a 4-to-0 age group and a 25-to-40 age group. (Perhaps the motion picture attracted children and their parents?)

Still another form of data mining is **association analysis,** which involves looking for links between data groups. It is association analysis that might reveal that customers who buy potato chips also buy beer and soda or that people who shop during the traditional weekday work hours also draw retirement benefits.

Outlier analysis is another form of data mining. It tries to identify data entries that do not comply with the norm. Outlier analysis can be used to identify errors in data collections, to identify credit card theft by detecting sudden deviations from a customer's normal purchase patterns, and perhaps to identify potential terrorists by recognizing unusual behavior.

Finally, the form of data mining called **sequential pattern analysis** tries to identify patterns of behavior over time. For example, sequential pattern analysis might reveal trends in economic systems such as equity markets or in environ- mental systems such as climate conditions.

As indicated by our last example, results from data mining can be used to predict future behavior. If an entity possesses the properties that characterize a class, then the entity will probably behave like members of that class. However, many data mining projects are aimed at merely gaining a better understanding of the data, as witnessed by the use of data mining in unraveling the mysteries of DNA. In any case, the scope of data mining applications is potentially enormous, and thus data mining promises to be an active area of research for years to come. Note that database technology and data mining are close cousins, and thus research in one will have repercussions in the other. Database techniques are used extensively to give data warehouses the capability of presenting data in the form of **data cubes** (data viewed from multiple perspectives—the term *cube* is used to conjecture the image of multiple dimensions) that make data mining possible. In turn, as researchers in data mining improve techniques for implementing data cubes, these results will pay dividends in the field of database design.

In closing, we should recognize that successful data mining encompasses much more than the identification of patterns within a collection of data. Intelligent judgment must be applied to determine whether those patterns are significant or merely coincidences. The fact that a particular convenience store has sold a high number of winning lottery tickets should probably not be considered significant to someone planning to buy a lottery ticket, but the discovery that customers who buy snack food also tend to buy frozen dinners might constitute meaningful information to a grocery store manager. Likewise, data mining encompasses a vast number of ethical issues involving the rights of individuals represented in the data warehouse, the accuracy and use of the conclusions drawn, and even the appropriateness of data mining in the first place.

Copyright Virtual University of Pakistan

Module 169

169. Database Systems: Social Impact of Database Technology

With the development of database technology, information that was once buried in arcane records has become accessible. In many cases, automated library systems place a patron's reading habits within easy reach, retailers maintain records of their customers' purchases, and Internet search engines keep records of their clients' requests. In turn this information is potentially available to marketing firms, law enforcement agencies, political parties, employers, and private individuals.

This is representative of the potential problems that permeate the entire spectrum of database applications. Technology has made it easy to collect enormous amounts of data and to merge or compare different data collections to obtain relationships that would otherwise remain buried in the heap. The ramifications, both positive and negative, are enormous. These ramifications are not merely subjects for academic debate—they are realities.

In some cases, the data collection process is readily apparent; in others it is subtle. Examples of the first case occur when one is explicitly asked to provide information. This may be done in a voluntary manner, as in surveys or contest registration forms, or it may be done in an involuntary manner, such as when imposed by government regulations. Sometimes whether it is voluntary depends on one's point of view. Is providing personal information when applying for a loan voluntary or involuntary? The distinction depends on whether receiving the loan is a convenience or a necessity. To use a credit card at some retailers now requires that you allow your signature to be recorded in a digitized format. Again, providing the information is either voluntary or involuntary depending on your situation.

More subtle cases of data collection avoid direct communication with the subject. Examples include a credit company that records the purchasing practices of the holders of its credit cards, websites that record the identities of those who visit the site, and social activists who record the license plate numbers on the cars parked in a targeted institution's parking lot. In these cases, the subject of the data collection might not be aware that information is being collected and less likely to be aware of the existence of the databases being constructed.

Sometimes the underlying data-collection activities are self-evident if one merely stops to think. For example, a grocery store might offer discounts to its regular customers who register in advance with the store. The registration process might involve the issuance of identification cards that must be presented at the time of purchase to obtain the discount. The result is that the store is able to compile a record of the customers' purchases—a record whose value far exceeds the value of the discounts awarded.

Of course, the force driving this boom in data collection is the value of the data, which is amplified by advances in database technology that allow data to be linked in ways that reveal information that would otherwise remain obscure. For example, the purchasing patterns of credit card holders can be classified and cross-listed to obtain customer profiles of immense marketing value. Subscription forms for body-building magazines can be mailed to those who have recently purchased exercise equipment, whereas subscription forms for dog obedience magazines can be targeted toward those who have recently purchased dog food. Alternative ways of combining information are sometimes very imaginative. Welfare records have been compared to criminal records to find and apprehend parole violators, and in 1984 the Selective Service in the United States used old birthday registration lists from a popular ice cream retailer to identify citizens who had failed to register for the military draft.

There are several approaches to protecting society from abusive use of data- bases. One is to apply legal remedies. Unfortunately, passing a law against an action does not stop the action from occurring but merely makes the action illegal. A prime example in the United States is the Privacy Act of 1974 whose purpose was to protect citizens from abusive use of government databases. One provision of this act required

Copyright Virtual University of Pakistan

government agencies to publish notice of their databases in the Federal Register to allow citizens to access and correct their personal information. However, government agencies were slow to comply with this provision. This does not necessarily imply malicious intent. In many cases the problem was one of bureaucracy. But the fact that a bureaucracy might be constructing personnel databases that it is unable to identify is not reassuring.

Another, and perhaps more powerful, approach to controlling database abuse is public opinion. Databases will not be abused if the penalties outweigh the benefits; and the penalty businesses fear the most is adverse public opinion—this goes right to the bottom line. In the early 1990s it was public opinion that ultimately stopped major credit bureaus from selling mailing lists for marketing purposes. More recently, Google discontinued its Google Buzz social networking tool in 2011 after its automatic sharing of contact information from the popular Gmail tool was greeted with harsh public criticism. Even government agencies have bowed to public opinion. In 1997 the Social Security Administration in the United States modified its plan to make Social Security records available via the Internet when public opinion questioned the security of the information. In some of these cases results were obtained in days—a stark contrast to the extended time periods associated with legal processes.

Of course, in many cases database applications are beneficial to both the holder and the subject of the data, but in all cases, there is a loss of privacy that should not be taken lightly. Such privacy issues are serious when the information is accurate, but they become gigantic when the information is erroneous. Imagine the feeling of hopelessness if you realized that your credit rating was adversely affected by erroneous information. Imagine how your problems would be amplified in an environment in which this misinformation was readily shared with other institutions. Privacy problems are, and will be, a major side effect of advancing technology in general and database techniques in particular. The solutions to these problems will require an educated, alert, and active citizenry.

VIEW LMS LECTURE FOR CONCEPT

Copyright Virtual University of Pakistan

Module 170

170. Artificial Intelligence: Introduction and Vision

Artificial intelligence is the field of computer science that seeks to build autonomous machines - machines that can carry out complex tasks without human intervention. AI is helping to make Robots as shown in the Figure 123, and self-driving cars as shown in the Figure 124.



Figure 123: Robot



Figure 124: Self Driving Cars

Robots can help patients, for example one typical robot shown in the Figure 125 can alleviate a person even 40 times a day.

Copyright Virtual University of Pakistan



Figure 125: Medical robot helping the patients

Pepper Robot works as receptionist in Belgian hospital shown in the Figure 126.

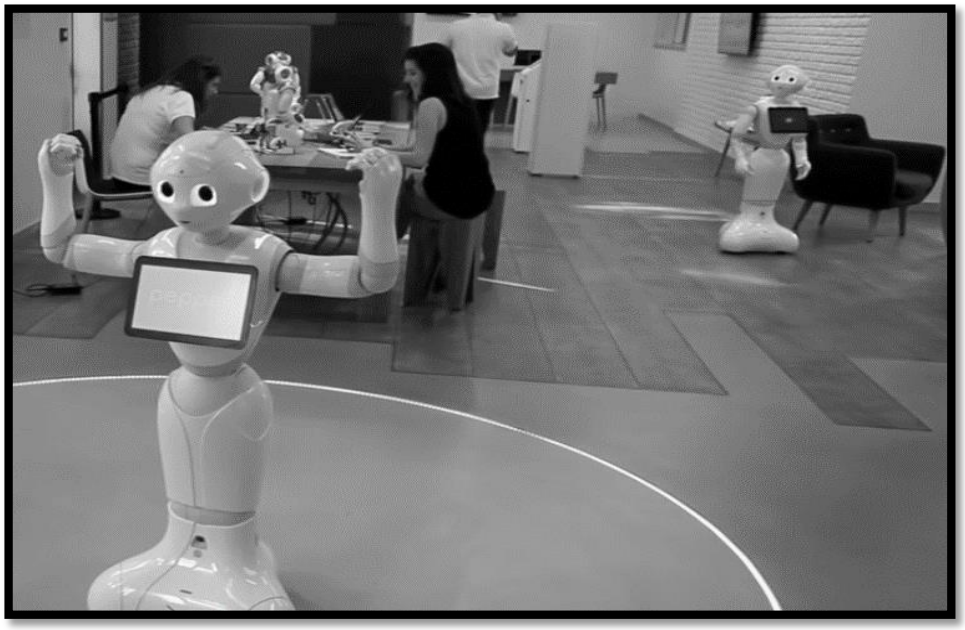


Figure 126: Receptionist Robot

Another robot is shown in the Figure 127. This robot is able to take blood from the veins of the humans and it is more accurate than humans. It performs image processing to locate the veins and further confirmed by the ultrasound.

Copyright Virtual University of Pakistan

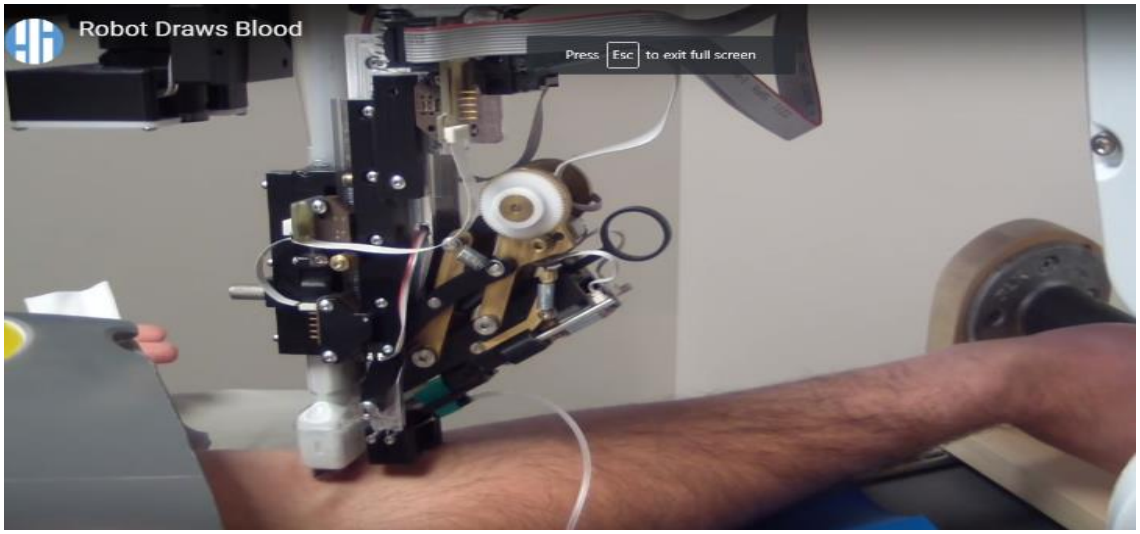


Figure 127: Robot draws blood

Copyright Virtual University of Pakistan

Module 171

171. Artificial Intelligence: Intelligent Agents

An **agent** is a "device" that responds to stimuli from its environment. It is natural to envision an agent as an individual machine such as a robot, although an agent may take other forms such as an autonomous airplane, a character in an interactive video game, or a process communicating with other processes over the Internet (perhaps as a client, a server, or a peer). Most agents have sensors by which they receive data from their environments and actuators by which they can affect their environments. Examples of sensors include microphones, cameras, range sensors, and air or soil sampling devices. Examples of actuators include wheels, legs, wings, grippers, and speech synthesizers.

Much of the research in artificial intelligence can be characterized in the context of building agents that behave intelligently, meaning that the actions of the agent's actuators must be rational responses to the data received though its sensors. In turn, we can classify this research by considering different levels of these responses.

The simplest response is a reflex action, which is merely a predetermined response to the input data. Higher levels of response are required to obtain more "intelligent" behavior. For example, we might empower an agent with knowledge of its environment and require that the agent adjust its actions accordingly. The process of throwing a baseball is largely a reflex action but determining how and where to throw the ball requires knowledge of the current environment. (There is one out with runners on first and third.) How such real-world knowledge can be stored, updated, accessed, and ultimately applied in the decision-making process continues to be a challenging problem in artificial intelligence.

Another level of response is required if we want the agent to seek a goal such as winning a game of chess or maneuvering through a crowded passageway. Such goal-directed behavior requires that the agent's response, or sequence of responses, be the result of deliberately forming a plan of action or selecting the best action among the current options.

In some cases, an agent's responses improve over time as the agent learns. This could take the form of developing **procedural knowledge** (learning "how") or storing **declarative knowledge** (learning "what"). Learning procedural knowledge usually involves a trial-and-error process by which an agent learns appropriate actions by being punished for poor actions and rewarded for good ones. Following this approach, agents have been developed that, over time, improve their abilities in competitive games such as checkers and chess. Learning declarative knowledge usually takes the form of expanding or altering the "facts" in an agent's store of knowledge. For example, a baseball player must repeatedly adjust his or her database of knowledge (there is still just one out, but now runners are on first and second) from which rational responses to future events are determined.

To produce rational responses to stimuli, an agent must "understand" the stimuli received by its sensors. That is, an agent must be able to extract information from the data produced by its sensors, or in other words, an agent must be able to perceive. In some cases, this is a straightforward process. Signals obtained from a gyroscope are easily encoded in forms compatible with calculations for determining responses. But in other cases, extracting information from input data is difficult. Examples include understanding speech and images. Likewise, agents must be able to formulate their responses in terms compatible with their actuators. This might be a straightforward process, or it might require an agent to formulate responses as complete spoken sentences—meaning that the agent must generate speech. In turn, such topics as image processing and analysis, natural language understanding, and speech generation are important areas of research. The agent attributes that we have identified here represent past as well as cur- rent areas of research. Of course, they are not totally independent of each other. We would like to develop agents that possess all of them, producing agents that understand the data received from their environments and develop new response patterns through a learning process whose goal is to maximize the agent's abili- ties.

Copyright Virtual University of Pakistan

However, by isolating various types of rational behavior and pursuing them independently, researchers gain a toehold that can later be combined with progress in other areas to produce more intelligent agents.

We close this subsection by introducing an agent that will provide a context for our discussion in Sections 11.2 and 11.3. The agent is designed to solve the eight-puzzle, which consists of eight square tiles labeled 1 through 8 mounted in a frame capable of holding a total of nine such tiles in three rows and three columns (Figure 128). Among the tiles in the frame is a vacancy into which any of the adjacent tiles can be pushed, allowing the tiles in the frame to be scrambled. The problem posed is to move the tiles in a scrambled puzzle back to their initial positions (Figure 128).

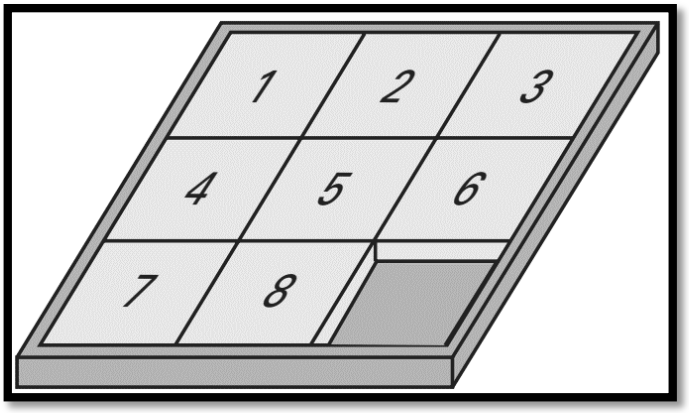


Figure 128: The eight-puzzle in its solved configuration

Our agent takes the form of a box equipped with a gripper, a video camera, and a finger with a rubber end so that it does not slip when pushing something (Figure 129). When the agent is first turned on, its gripper begins to open and close as if asking for the puzzle. When we place a scrambled eight-puzzle in the gripper, the gripper closes on the puzzle. After a short time, the machine's finger lowers and begins pushing the tiles around in the frame until they are back in their original positions. At this point the machine releases the puzzle and turns itself off.

This puzzle-solving machine exhibits two of the agent attributes that we have identified. First, it must be able to perceive in the sense that it must extract the current puzzle state from the image it receives from its camera.

Copyright Virtual University of Pakistan

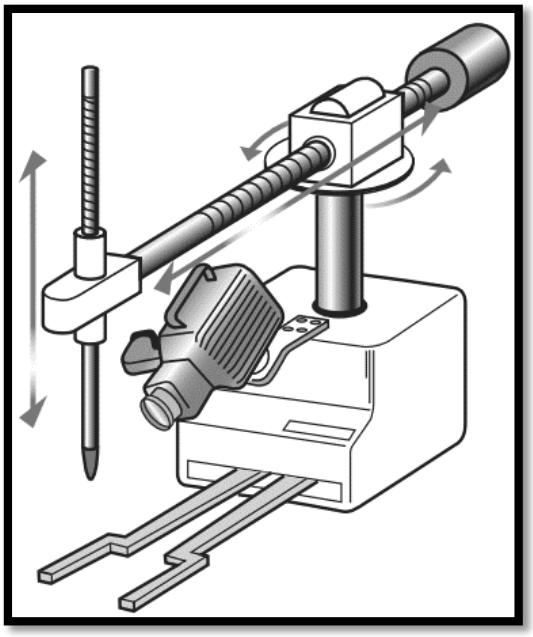


Figure 129: Our puzzle-solving machine

Copyright Virtual University of Pakistan

Module 172

172. Artificial Intelligence: Research Methodologies

To appreciate the field of artificial intelligence, it is helpful to understand that it is being pursued along two paths. One is the engineering track in which researchers are trying to develop systems that exhibit intelligent behavior. The other is a theoretical track in which researchers are trying to develop a computational understanding of animal—especially human—intelligence. This dichotomy is clarified by considering the manner in which the two tracks are pursued. The engineering approach leads to a performance-oriented methodology because the underlying goal is to produce a product that meets certain performance goals. The theoretical approach leads to a simulation-oriented methodology because the underlying goal is to expand our understanding of intelligence and thus the emphasis is on the underlying process rather than the exterior performance.

As an example, consider the fields of natural language processing and linguistics. These fields are closely related and benefit from research in each other, yet the underlying goals are different. Linguists are interested in learning how humans process language and thus tend toward more theoretical pursuits. Researchers in the field of natural language processing are interested in developing machines that can manipulate natural language and therefore lean in the engineering direction. Thus, linguists operate in simulation-oriented mode- building systems whose goals are to test theories. In contrast, researchers in natural language processing operate in performance-oriented mode-building systems to perform tasks. Systems produced in this latter mode (such as document translators and systems by which machines respond to verbal commands) rely heavily on knowledge gained by linguists but often apply "shortcuts" that happen to work in the restricted environment of the particular system.

As an elementary example, consider the task of developing a shell for an operating system that receives instructions from the outside world through verbal English commands. In this case, the shell (an agent) does not need to worry about the entire English language. More precisely, the shell does not need to distinguish between the various meanings of the word *copy*. (Is it a noun or a verb? Should it carry the connotation of plagiarism?) Instead, the shell needs merely to distinguish the word *copy* from other commands such as *rename* and *delete*. Thus, the shell could perform its task just by matching its inputs to predetermined audio patterns. The performance of such a system may be satisfactory to an engineer, but the way it is obtained would not be aesthetically pleasing to a theoretician.

Copyright Virtual University of Pakistan

Module 173

173. Artificial Intelligence: The Turing Test

In the past the **Turing test** (proposed by Alan Turing in 1950) has served as a benchmark in measuring progress in the field of artificial intelligence. Today the significance of the Turing test has faded although it remains an important part of the artificial intelligence folklore. Turing's proposal was to allow a human, whom we call the interrogator, to communicate with a test subject by means of a typewriter system without being told whether the test subject was a human or a machine. In this environment, a machine would be declared to behave intelligently if the interrogator was not able to distinguish it from a human. Turing predicted that by the year 2000 machines would have a 30 percent chance of passing a five-minute Turing test—a conjecture that turned out to be surprisingly accurate.

One reason that the Turing test is no longer considered to be a meaningful measure of intelligence is that an eerie appearance of intelligence can be produced with relative ease. A well-known example arose as a result of the pro- gram DOCTOR (a version of the more general system called ELIZA) developed by Joseph Weizenbaum in the mid-1960s. This interactive program was designed to project the image of a Rogerian analyst conducting a psychological interview; the computer played the role of the analyst while the user played the patient. Internally, all that DOCTOR did was restructure the statements made by the patient according to some well-defined rules and direct them back to the patient. For example, in response to the statement "I am tired today," DOCTOR might have replied with "Why do you think, you are tired today?" If DOCTOR was unable to recognize the sentence structure, it merely responded with something like "Go on" or "That's very interesting."

Weizenbaum's purpose in developing DOCTOR dealt with the study of nat- ural language communication. The subject of psychotherapy merely provided an environment in which the program could "communicate." To Weizenbaum's dismay, however, several psychologists proposed using the program for actual psychotherapy. (The Rogerian thesis is that the patient, not the analyst, should lead the discussion during the therapeutic session, and thus, they argued, a computer could possibly conduct a discussion as well as a therapist could.) Moreover, DOCTOR projected the image of comprehension so strongly that many who "communicated" with it became subservient to the machine's question-and- answer dialogue. In a sense, DOCTOR passed the Turing test. The result was that ethical, as well as technical, issues were raised, and Weizenbaum became an advocate for maintaining human dignity in a world of advancing technology.

More recent examples of Turing test "successes" include Internet viruses that carry on "intelligent" dialogs with a human victim in order to trick the human into dropping his or her malware guard. Moreover, phenomena similar to Turing tests occur in the context of computer games such as chess-playing programs. Although these programs select moves merely by applying brute-force techniques, humans competing against the computer often experience the sensation that the machine possesses creativity and even a personality. Similar sensations occur in robotics where machines have been built with physical attributes that project intelligent characteristics. Examples include toy robot dogs that project adorable personalities merely by tilting their heads or lifting their ears in response to a sound.

Copyright Virtual University of Pakistan

Module 174

174. Artificial Intelligence: Understanding Images

Let us consider the problems posed by the puzzle-solving machine introduced in the module number 171. The opening and closing of the gripper on the machine present no serious obstacle, and the ability to detect the presence of the puzzle in the gripper during this process is straightforward because our application requires very little precision. Even the problem of focusing the camera on the puzzle can be handled simply by designing the gripper to position the puzzle at a particular predetermined position for viewing. Consequently, the first intelligent behavior required by the machine is the extraction of information through a visual medium.

It is important to realize that the problem faced by our machine when looking at the puzzle is not that of merely producing and storing an image. Technology has been able to do this for years as in the case of traditional photography and television systems. Instead, the problem is to understand the image in order to extract the cur- rent status of the puzzle (and perhaps later to monitor the movement of the tiles). In the case of our puzzle-solving machine, the possible interpretations of the puzzle image are relatively limited. We can assume that what appears is always an image containing the digits 1 through 8 in a well-organized pattern. The problem is merely to extract the arrangement of these digits. For this, we imagine that the picture of the puzzle has been encoded in terms of bits in the computer's memory, with each bit representing the brightness level of a particular pixel. Assuming a uniform size of the image (the machine holds the puzzle at a predetermined location in front of the camera), our machine can detect which tile is in which position by comparing the different sections of the picture to prerecorded templates consisting of the bit patterns produced by the individual digits used in the puzzle. As matches are found, the condition of the puzzle is revealed.

This technique of recognizing images is one method used in optical character readers. It has the drawback, however, of requiring a certain degree of uniformity for the style, size, and orientation of the symbols being read. In particular, the bit pattern produced by a physically large character does not match the template for a smaller version of the same symbol, even though the shapes are the same. Moreover, you can imagine how the problems increase in difficulty when trying to process handwritten material.

Another approach to the problem of character recognition is based on matching the geometric characteristics rather than the exact appearance of the symbols. In such cases the digit 1 might be characterized as a single vertical line, 2 might be an opened curved line joined with a horizontal straight line across the bottom, and so on. This method of recognizing symbols involves two steps: The first is to extract the features from the image being processed, and the second is to compare the features to those of known symbols. As with the template-matching approach, this technique for recognizing characters is not foolproof. For instance, minor errors in the image can produce a set of entirely different geometric features, as in the case of distinguishing between an O and a C or, in the case of the eight-puzzle, a 3 and an 8. We are fortunate in our puzzle application because we do not need to under- stand images of general three-dimensional scenes. Consider, for example, the advantage we have by being assured that the shapes to be recognized (the digits 1 through 8) are isolated in different parts of the picture rather than appearing as overlapping images, as is common in more general settings. In a general photograph, for instance, one is faced not only with the problem of recognizing an object from different angles but also with the fact that some portions of the object might be hidden from view.

The task of understanding general images is usually approached as a two-step process: (1) **image processing,** which refers to identifying characteristics of the image, and (2) **image analysis,** which refers to the process of understanding what these characteristics mean. We have already observed this dichotomy in the context of recognizing symbols by means of their geometric features. In that situation, we found image processing represented by the process of identifying the geometric features found in the image and image analysis represented by the process of identifying the meaning of those features.

Copyright Virtual University of Pakistan

Image processing entails numerous topics. One is edge enhancement, which is the process of applying mathematical techniques to clarify the boundaries between regions in an image. In a sense, edge enhancement is an attempt to convert a photograph into a line drawing. Another activity in image analysis is known as region finding. This is the process of identifying those areas in an image that have common properties such as brightness, color, or texture. Such a region probably represents a section of the image that belongs to a single object. (It is the ability to recognize regions that allows computers to add color to old-fashioned black-and- white motion pictures.) Still another activity within the scope of image processing is smoothing, which is the process of removing flaws in the image. Smoothing keeps errors in the image from confusing the other image-processing steps, but too much smoothing can cause the loss of important information as well.

Smoothing, edge enhancement, and region finding are all steps toward identifying the various components in an image. Image analysis is the process of determining what these components represent and ultimately what the image means. Here one faces such problems as recognizing partially obstructed objects from different perspectives. One approach to image analysis is to start with an assumption about what the image might be and then try to associate the components in the image with the objects whose presence is conjectured. This appears to be an approach applied by humans. For instance, we sometimes find it hard to recognize an unexpected object in a setting in which our vision is blurred, but once we have a clue to what the object might be, we can easily identify it.

The problems associated with general image analysis are enormous, and much research in the area remains to be done. Indeed, image analysis is one of the fields that demonstrates how tasks that are performed quickly and apparently easily by the human mind continue to challenge the capabilities of machines.

Module 175

175. Artificial Intelligence: Language Processing

Another perception problem that has proven challenging is that of understanding language. The success obtained in translating formal high-level programming languages into machine language led early researchers to believe that the ability to program computers to understand natural language was only a few years away. Indeed, the ability to translate programs gives the illusion that the machine actually understands the language being translated.

What these researchers failed to understand was the depth to which formal programming languages differ from natural languages such as English, German, and Latin. Programming languages are constructed from well-designed primitives so that each statement has only one grammatical structure and only one meaning. In contrast, a statement in a natural language can have multiple meanings depending on its context or even the manner in which it is communicated. Thus, to understand natural language, humans rely heavily on additional knowledge.

For example, the sentences

Norman Rockwell painted people.

and

Cinderella had a ball.

have multiple meanings that cannot be distinguished by parsing or translating each word independently. Instead, to understand these sentences requires the ability to comprehend the context in which the statement is made. In other instances, the true meaning of a sentence is not the same as its literal translation. For example,

Do you know what time it is?

Copyright Virtual University of Pakistan

Often means "Please tell me what time it is," or if the speaker has been waiting for a long time, it might mean "You are very late."

To unravel the meaning of a statement in a natural language therefore requires several levels of analysis. The first of these is **syntactic analysis,** whose major component is parsing. It is here that the subject of the sentence

Mary gave John a birthday card. is recognized as *Mary*, while the subject of

John got a birthday card from Mary.

is found to be *John.*

Another level of analysis is called **semantic analysis.** In contrast to the parsing process, which merely identifies the grammatical role of each word, semantic analysis is charged with the task of identifying the semantic role of each word in the statement. Semantic analysis seeks to identify such things as the action described, the agent of that action (which might or might not be the subject of the sentence), and the object of the action. It is through semantic analysis that the sentences "Mary gave John a birthday card" and "John got a birthday card from Mary" would be recognized as saying the same thing.

A third level of analysis is **contextual analysis.** It is at this level that the context of the sentence is brought into the understanding process. For example, it is easy to identify the grammatical role of each word in the sentence:

The bat fell to the ground.

We can even perform semantic analysis by identifying the action involved as *falling,* the agent as *bat,* and so on. But it is not until we consider the context of the statement that the meaning of the statement becomes clear. In particular, it has a different meaning in the context of a baseball game than it does in the context of cave exploration. Moreover, it is at the contextual level that the true meaning of the question "Do you know what time it is?" would finally be revealed.

We should note that the various levels of analysis—syntactic, semantic, and contextual—are not necessarily independent. The subject of the sentence

Stampeding cattle can be dangerous.

is the noun *cattle* (modified by the adjective *stampeding*) if we envision the cattle stampeding on their own. But the subject is the gerund *stampeding* (with object *cattle*) in the context of a troublemaker whose entertainment consists of starting stampedes. Thus the sentence has more than one grammatical structure—which one is correct depends on the context.

Another area of research in natural language processing concerns an entire document rather than individual sentences. Here the problems of concern fall into two categories: **information retrieval** and **information extraction.** Information retrieval refers to the task of identifying documents that relate to the topic at hand. An example is the problem faced by users of the World Wide Web as they try to find the sites that relate to a particular topic. The current state of the art is to search sites for key words, but this often produces an avalanche of false leads and can overlook an important site because it deals with "automobiles" instead of "cars." What is needed is a search mechanism that understands the contents of the sites being considered. The difficulty of obtaining such under- standing is the reason many are turning to techniques such as XML to produce a semantic Web as was introduced in the module 75.

Copyright Virtual University of Pakistan

Module 176

176. CS Impact: CS impact on Society

Computer Science has impacted lives of humans largely. In this module we will learn the positive and negative impacts of CS on our society.

Positive impact of CS:

- ✓ It has improved the communication between people.
- ✓ Increased access to educational information via the Internet
- ✓ Increased productivity of people as now their manual work has reduced significantly.
- ✓ Exponential business growth
- ✓ Faster response times (for example, email, instant messaging and chat)

Negative Impact:

- ✓ People are working in sitting position and thus have more chance to develop obesity.
- ✓ Exposure to inappropriate material via the Internet
- ✓ Increased crime and access to private information
- ✓ Hacking bank account and transfer money to your account
- ✓ People are now over-dependent on technology. They even try to catch the people online even if the person is sitting near them geographically. This has decreased the social interactions as well.

Copyright Virtual University of Pakistan

Module 177

177. CS Impact: CS Impact on Health

Computer Science has both positive and negative impacts on health.

Positive impact

- ✓ Easy access to online health centers
- ✓ We can take online appointment
- ✓ We can receive reports via email at home
- ✓ Hospitals can be managed more efficiently
- ✓ Online health tips/issues are available.
- ✓ Health news can be viewed online.

Negative impact

- ✓ Computer wrists – Typing too much. This creates issues in the hands.
- ✓ Eyesight – eye is not meant to look 2D for long amount of time. This increases the risk of eye diseases.
- ✓ Hearing loss is more common in those listening to high volume music on computer.
- ✓ Social Isolation
- ✓ Communication Issues
- ✓ Posture could lead bone issues
- ✓ Radiation Damage when the device is running. Its better to put off all such digital devices especially mobile phones or if it's really necessary to use mobile phone as alarm, then turn on the "airplane mode" in your phones.

Copyright Virtual University of Pakistan

Module 178

178. CS Impact: CS Impact on Environment

Computer Science has both positive and negative impacts on the environment as well.

Positive impact

- ✓ Paperless – Online reading and publishing.
- ✓ Americans use about 680 pounds of paper per person, per year
- ✓ This amounts to about 85 million tons of paper or 2 billion trees.
- ✓ The average American household throws away 13,000 pieces of paper (around 1 billion trees in total in USA) each year. Most in the form of packaging and junk mail.
- ✓ Due to the usage of computer, the above waste of trees is minimizing in the world.

Negative impact

- ✓ In offices, people print unnecessarily.
- ✓ Computers are made from heavy metals (Lead, mercury, Beryllium, Cadmium, PCV) when thrown-away cause water contamination and air pollution.
- ✓ E-waste is sent to developing countries where people extract materials from these electronics such as gold, silver, and copper.
- ✓ They extract them by burning the substances and that process releases hazardous smoke into the air.
- ✓ The people present around the e-waste gain diseases such as skin cancer, brain damage, lung cancer, kidney disease
- ✓ Usage of energy has also increased with the use of computer.

Copyright Virtual University of Pakistan

Module 179

179. CS Impact: Ethical Issues

Ethical issues are related to moral principles or the branch of knowledge dealing with these. Questions of ethics and legality are essential in many industries. Doctors, teachers, government officials and businesspeople all have legal and ethical oversight to control how their professions function. Information technology, by contrast, has no overarching standardization in place. However, as information technology becomes increasingly influential, the ethical and legal considerations become similarly relevant. Here are the five most pressing ethical and legal issues confronting the industry today.

Privacy

Most people have their personal data spread throughout the digital world. Even things thought to be secure, such as email or private accounts, can be accessed by unintended sources. Most employers actively check their employees' computer habits. Privacy has evolving legal implications, but there are also ethical considerations. Do people know how their accounts are monitored? To what extent is such monitoring occurring? Privacy concerns can easily become a slippery slope, slowly eroding an individual's right to privacy completely.

Digital Ownership

Digital mediums have allowed information to flow more freely than before. This exchange of ideas comes with a legal and ethical backlash. How can ownership be established in the digital realm? Things can be easily copied and pasted online, which makes intellectual property hard to control. Legal notions such as copyright have struggled to keep up with the digital era. Companies in the music and entertainment industries have pushed for greater legal protections for intellectual properties while other activists have sought to provide greater freedoms for the exchange of ideas in the digital realm.

Data Gathering

On some level, everyone knows that their online lives are monitored. The United States has even passed legislation allowing the government to actively monitor private citizens in the name of national security. These measures have revived a debate about what information can be gathered and why. This debate applies on a smaller scale as well because companies need to consider what information to collect from their employees. This issue invokes a question of consent. Do people know what information is being monitored? Do they have a right to know how their data is being used?

Copyright Virtual University of Pakistan

Module 180

180. CS Impact: Software Licenses and Information Privacy

Software licenses typically provide end users with the right to one or more copies of the software without violating copyrights. The license also defines the responsibilities of the parties entering into the license agreement and may impose restrictions on how the software can be used. Software licensing terms and conditions usually include fair use of the software, the limitations of liability, warranties and disclaimers and protections if the software or its use infringes on the intellectual property rights of others.

Software licenses typically are either proprietary, free or open source, the distinguishing feature being the terms under which users may redistribute or copy the software for future development or use.

Information Privacy

Data privacy, also called information privacy deals with the ability an organization or individual has to determine what data in a computer system can be shared with third parties.

Children's Online Privacy Protection Act

(COPPA) - gives parents control over what information websites can collect from their kids.

Health Insurance Portability and Accountability Act

(HIPPA) - ensures patient confidentiality for all healthcare-related data.

Electronic Communications Privacy Act

ECPA extends government restrictions on wire taps to include transmissions of electronic data.

Video Privacy Protection Act

Prevents wrongful disclosure of an individual's personally identifiable information stemming from their rental or purchase of audiovisual material.

Gramm-Leach-Bliley Act

Mandates how financial institutions must deal with the private information of individuals.

Copyright Virtual University of Pakistan

Module 181

181. CS Impact: Intellectual Property

Intellectual property (IP) is ideas, information and knowledge. In the University context IP can be viewed as the results and outcomes of research – 'intellectual' because it is creative output and 'property' because it is viewed as a tradable commodity. Intellectual property rights (IPR) are specific legal rights which protect the owners of IP. IPR can be subdivided into the major categories below.

Patent

A patent is a legal monopoly lasting 20 years granted in exchange for describing an invention and paying fees to the Patent Office. A patent position is destroyed by public disclosure of the idea before a patent application is filed (except for a short grace period in the US). Think patent before you publish.

Copyright

Copyright applies to literary and dramatic works, artistic and musical works, audio and video recordings, broadcasts and cable transmissions. Copyright is also the usual way of protecting software, although some software may be patented if it is a functional part of an invention. Copyright arises automatically – it does not need to be applied for – and lasts 70 years after the death of the author.

Database right

Database rights apply to databases which are not protected by copyright (an EU right only – maximum duration is 15 years).

Design right

Design right applies to aspects of the shape or configuration of an article. Unregistered design right (which covers computer chips, for example) can protect internal or external features. In the case of registered designs, the features must appeal to and be judged by the eye. (Registered design rights are protected for a maximum of 25 years.)

Trademark

A trademark is a mark (logo) or other distinctive sign applied to or associated with products or services, which does not describe the products or services. (Once registered the trademark IPR is unlimited.)

Confidential information

Confidential information is knowledge which only you possess and which you have only revealed under a non-disclosure/confidentiality agreement. The comprehensive table has been shown to highlight the details of all the above intellectual properties.

IPR	Covers	Need to apply?	Maximum duration
Patent	Inventions	Yes	20 years
Copyright	Literary, musical, artistic works, and software	No	70 years after death of author

Copyright Virtual University of Pakistan

Registered design	Image – look and feel	Yes	25 years
Registered trademark	Name, logo	Yes	Unlimited
Confidential information	Unpublished secret information	No	Unlimited
Database right	Databases	No	15 years

Successful management of IPR provides the means by which individuals and institutions are able to protect their creative output from imitators.

An understanding of IP and IPRs is an increasingly important aspect of University and business life. Now, more than ever, IP is recognized as a tradable commodity.

Copyright Virtual University of Pakistan

Module 182

182. CS Impact: Security

Computer security, also known as **cyber-security** or **IT security**, is the protection of information systems from theft or damage to the hardware, the software, and to the information on them, as well as from disruption or misdirection of the services they provide. It includes controlling physical access to the hardware, as well as protecting against harm that may come via network access, data and code injection, and due to malpractice by operators, whether intentional, accidental, or due to them being tricked into deviating from secure procedures.

What are the concerns of computer security?

Computer Security is concerned with four main areas:

1. Confidentiality:- Only authorized users can access the data resources and information.
2. Integrity:- Only authorized users should be able to modify the data when needed.
3. Availability:- Data should be available to users when needed.
4. Authentication:- are you really communicating with whom you think you are communicating with

Why is computer security important?

Prevention of data theft such as bank account numbers, credit card information, passwords, work related documents or sheets, etc. is essential in today's communications since many of our day to day actions depend on the security of the data paths.
Data present in a computer can also be misused by unauthorized intrusions. An intruder can modify and change the program source codes and can also use your pictures or email accounts to create derogatory content such as pornographic images, fake misleading and offensive social accounts.

Malicious intents can also be a factor in computer security. Intruders often use your computers for attacking other computers or websites or networks for creating havoc. Vengeful hackers might crash someone's computer system to create data loss. DDOS attacks can be made to prevent access to websites by crashing the server.
Above factors imply that your data should remain safe and confidential. Therefore, it is necessary to protect your computer and hence the need for Computer Security arises.

What is firewall?

A firewall is a security-conscious piece of hardware or soft-ware that sits between the Internet and your network with a single-minded task: preventing them from getting to us. The firewall acts as a security guard between the Internet and your local area network (LAN). All network traffic into and out of the LAN must pass through the firewall, which prevents unauthorized access to the network.

Copyright Virtual University of Pakistan

Module 183

183. CS Impact: Privacy

Individual's right to own the data generated by his or her life and activities, and to restrict the outward flow of that data. Privacy uses the theory of natural rights, and generally responds to new information and communication technologies. In North America, Samuel D. Warren and Louis D. Brandeis wrote that privacy is the "right to be let alone".

From where data comes from:

- ✓ Social Sites (Facebook, twitter)
- ✓ Telephone companies (mobile networks)
- ✓ Smart city cameras on road
- ✓ Emails
- ✓ Personal Software
- ✓ Passwords, logins etc

How to handle on internet

- ✓ Emails can be encrypted
- ✓ Anonymizing proxies
- ✓ Anonymizing networks

Privacy Information

Privacy information of famous websites is given below:

- ✓ Google:

https://policies.google.com/privacy

- ✓ msn:

https://www.msn.com/en-us/autos/partner/privacypolicy

- ✓ WhatsApp:

https://www.whatsapp.com/legal/#privacy-policy

EPIC

- ✓ Electronic Privacy Information Center
- ✓ Independent non-profit research center in Wasgingon D.C. Its mission is to focus public attention on emerging Privacy and related human rights issues.
- ✓ EPIC has pursued several successful consumer privacy complaints with the US Federal Trade Commission, concerning Snapchat (faulty privacy Technology), WhatsAp (privacy policy after acquisition by Facebook (changes in user privacy settings), Google (roll-out of GoogleBuzz), Microsoft (Hailstorm log-in),issues.

PI

- ✓ Privacy International

Copyright Virtual University of Pakistan

- ✓ Charity organization of London, UK

Priority Areas:

- ✓ Challenging Data Exploitation
- ✓ Building the Global Privacy Movement
- ✓ Contesting Surveillance

Download More Highlighted Handouts From

VUAnswer.com

Copyright Virtual University of Pakistan

Module 184

184. CS Impact: Social Issues of IT

The growth in the availability of affordable computing technology has caused a number of major shifts in the way that society operates. The majority of these have been for the better, with home computers and the internet providing unlimited access to all of the information ever created and discovered by humanity.

There are, however, some less positive social issues generated as a direct result of technological advances. In the interests of balance, it is important to analyze these and assess the severity of their impact so that steps can be taken to better understand and combat the negative effects.

1. Communication Breakdown

Socializing within a family unit has always been important, as it strengthens the bonds between us and ensures cohesion within the group. But with more and more households owning several computers and numerous portable devices granting access to information and entertainment, some argue that this is leading to a lack of family communication.

2. Defamation of Character

The only means of getting in touch with major corporations or famous people in the public eye prior to the advent of digital communication was via a stiffly written letter. This was, of course, accessible only to the intended recipient and thus a very private way for the disgruntled to vent their spleen. But first message boards and now social media services like Facebook and Twitter are being used to defame people and businesses in an intrinsically public manner.

3. Identity Theft

Fraud is another spurious activity that has been able to evolve in the wake of easily accessible computers and the internet. Perhaps most problematic and prevalent of the various fraudulent activities is identity theft, in which personal details of innocent people are harvested by a third party so that they can be used for malicious purposes. This includes carrying out illicit online transactions and other damaging activities that can have serious ramifications.

4. Cyber Bullying

As with the defamation of public figures, the internet and computers have also made it easier for spiteful people to attack people they know personally as well as perfect strangers via the anonymous platforms that are available to them.

5. Gaming Addiction

Whilst computers and the internet have made it easier for gambling addicts to get their fix, a new type of addiction has also arisen, in the form of addiction to videogames. This is something that can impact people of all ages and leads inevitably to a number of problems, from the social to the financial.

6. Privacy

Whilst high profile cases of online identity theft and fraud should have caused people to become more careful about how they use their personal information, issues of privacy and a lack of appreciation for the

Copyright Virtual University of Pakistan

risks are still widespread. This extends beyond simply giving away private data via chat rooms, message boards and e-commerce sites and extends into the compromising world of social media.

7. Health & Fitness

We are living increasingly sedentary lifestyles, because computers are removing the need for us to physically carry out many tasks, as well as keeping us rooted to one spot throughout our working days and during our leisure time.

8. Education

The educational properties of computers are well known and universally lauded but having all the information in existence on tap has its own issues. In particular, the practice of plagiarism has become a major problem, as students can simply copy and paste whole chunks of text from online sources without attributing the work to anyone else. This has become the bane of educational institutions, which tend to come down hard on detected plagiarists in order to discourage similar activities from others.

9. Terrorism & Crime

Computers have been a positive force in allowing for the creation of global movements and righteous activism in a number of forms. However, the other side of the coin is that terrorists and organized criminals also exploit the web for their own nefarious purposes.

10. Access to forbidden literature

Computer has made available the access to the forbidden literature. This has led us to many social problems in the society.

Copyright Virtual University of Pakistan

Module 185

185. CS Impact: Content Filtering, Email-Spams and Laws

Content filtering, in the most general sense, involves using a program to prevent access to certain items, which may be harmful if opened or accessed. The most common items to filter are executable, emails or websites.

How it works

Content filtering works by matching strings of characters. When the strings match, the content is not allowed through. Content filters are often part of Internet firewalls. In such a usage content filtering is serving a security purpose. Content filtering is also used to implement company policies related to information system usage. For example, it's common to filter websites like Facebook, youtube etc which are considered unrelated to work.

Spam emails

As email spam continues to become a major issue, governments around the world have put specific regulations in place to protect their citizens from spams.

Important Laws

- ✓ CAN-SPAM Act in US
- ✓ Canada's Anti-Spam Legislation (CASL)
- ✓ Anti-spam law in Europe
- ✓ Spam Act of 2003 in Australia
- ✓ Africa, Asia and South America have looser spam-law requirements

CAN-SPAM Act

1. Don't use false or misleading header information
2. Don't use deceptive subject lines
3. Identify the message as an ad
4. Tell recipients where you're located
5. Honor opt-out requests promptly
6. Monitor what others are doing on your behalf

Major requirements

- ✓ Ask for permission before adding emails
- ✓ User could identify you as Sender
- ✓ Do not give email address to others
- ✓ Be honest

Copyright Virtual University of Pakistan

Module 186

186. CS Impact: Children Protection and Electronic Theft

There are lots of reasons why people start using parental controls. Perhaps to stop late night phone checking. Perhaps to help stop arguments when it's time put the iPad down and get on with homework. Maybe to reduce the total amount of time being spent on screens in general. There are many more cases. Ultimately though all of these reasons amount to a desire for one's children to become more responsible and disciplined in their use of digital technology.

We hope and aim to encourage an attitude and approach in our kids that will render parental controls unnecessary. But how can we make sure that our use of parental controls doesn't wind up as abstract punishment or authoritarian control but is a genuine tool for building responsibility and right priorities in our kids? One method that's got quite a bit of attention on social media is the idea of a parent/child cell phone contract.

The following risks are involved in such scenarios:

- ✓ Content Risks.
- ✓ Contact Risks
- ✓ Online Marketing
- ✓ Overspending
- ✓ Electronic Theft
- ✓ Information Privacy
- ✓ Information Security

Content Risks

- ✓ Illegal Content
- ✓ Harmful Content
- ✓ Harmful Advice

Contact Risks

- ✓ Online Harassment
- ✓ Cyberbullying
- ✓ Illegal interactions
- ✓ Problematic content sharing

Online Marketing

- ✓ Inappropriate or unsuitable products
- ✓ Illegal and Age-restricted products

Electronic Theft

- ✓ Online Fraud
- ✓ Online scam
- ✓ Identify theft

Copyright Virtual University of Pakistan

Information Privacy

- ✓ Personal data collection from children

Information Security

- ✓ Malicious code
- ✓ Commercial spyware

How to protect

- ✓ Web Filtering
- ✓ Parental control software
- ✓ Enable safety features
- ✓ Guide kids

Copyright Virtual University of Pakistan

Module 187

187. Word Processing: MS Word

One of the most important application software are the word processors. These software help individuals and companies to prepare digital documents in a proper format. One of the famous word processing software is MS Word, which is included in the Microsoft Office.

Figure 130 is the screen short for windows XP to open MS Word 2007.

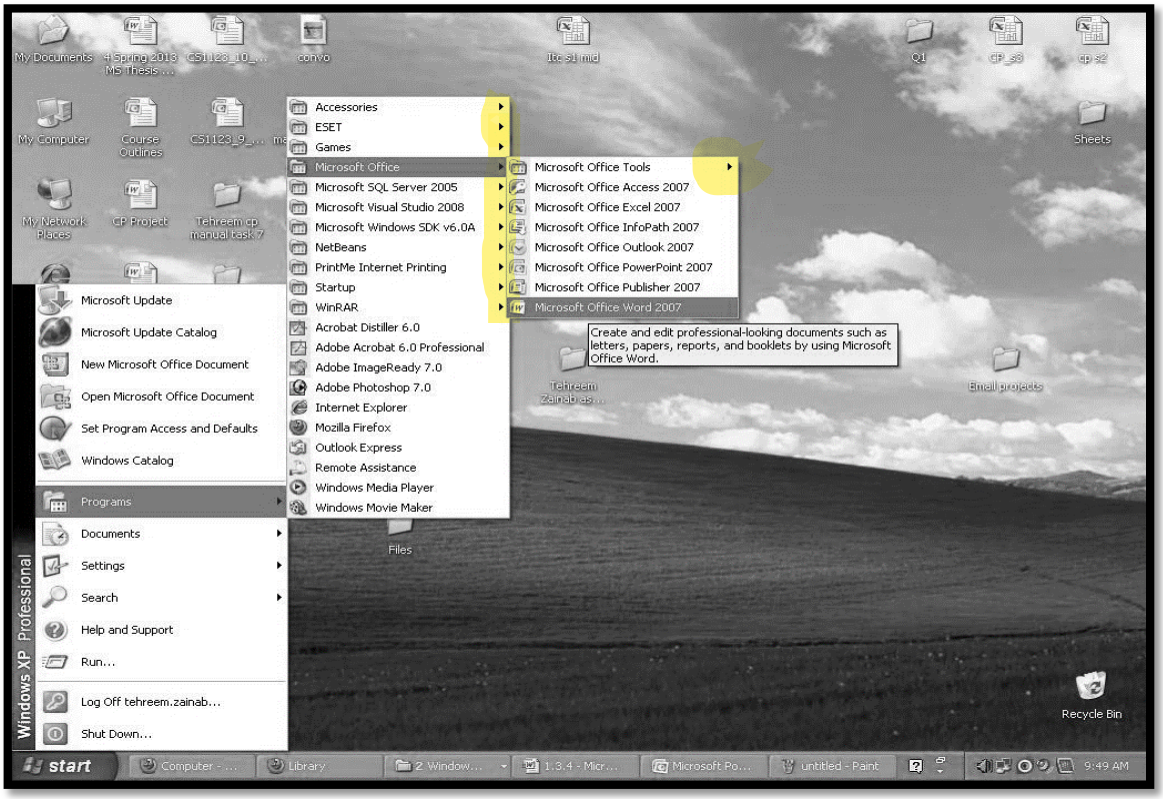


Figure 130: Start Menu to launch MS Word

The Office Button

Figure 130 shows the MS word home ribbon.



Figure 131: MS Word Home ribbon

Copyright Virtual University of Pakistan


The colorful icon in the top left-hand corner of the **Word** window (Figure 131) is the **Office button.** When you point to the Office button with the mouse pointer it will turn orange**.** When you click the left mouse button while pointing to the Office button, a menu will open that contains some of the most commonly used commands which used to be found in the **File** drop-down menu in earlier versions of **Word**. **Recent documents** are listed in the **Recent Documents** pane on the right side of the menu as shown in Figure 132.

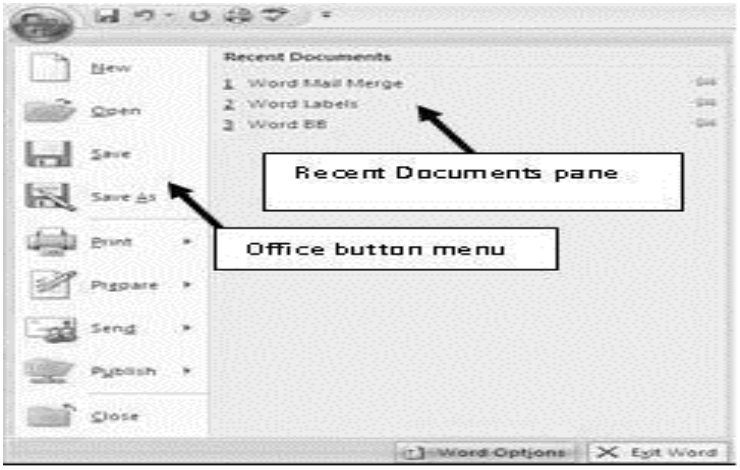


Figure 132: MS Word home ribbon after click

The ribbon is divided into Tabs, Groups, and Commands.

Each tab opens a different ribbon with groups of commands inside it. Microsoft has tried to make the placement of the commands within both groups and tabs as intuitive as possible to make them easy to find. By using the Ribbon system, the user has many less mouse clicks to execute the desired command.

The items on the ribbons are grouped together by common task. The main ribbon is the Home Tab/ribbon. The home tab/ribbon in Excel and PowerPoint are similar. Below is the image of MS Word Home Ribbon.

Below is the explanation of different components in the home ribbon of MS Word 2007.


In the lower right-hand corner of some groups is a small down pointing arrow. These are referred to as **Dialog Expanders** or **Launchers**. Clicking on the arrow will open a **Dialog window** or a **Command pane**. The dialog box may provide additional options that may not be contained on the active tab. Most of the dialog boxes that appear should be familiar; many are taken directly from earlier versions of the program.

The Ribbon has three types of Commands:

- Simple Buttons
- Drop-down menus Buttons
- Launchers (or Dialog Expanders) Buttons

Simple Buttons work as **Toggles** or **on/off switches**. When the command is off it will be surrounded by blue and look like all other commands as shown in the Figure 133. When a command button is on, it will be surrounded by orange. A command will only work when it is on. **Bold** is an example of a button as shown in Figure 133.

Copyright Virtual University of Pakistan



Figure 133: Font Panel

If you aren't sure what a command is or what a command does, place the mouse pointer over the command and hold it there. A text box will open up. The text box contains the command name, the keyboard shortcuts for the command, along with an explanation of the command as shown in the Figure 134.

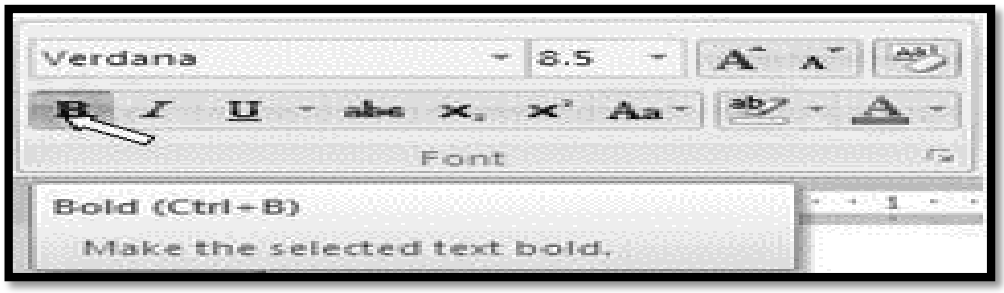


Figure 134: Bold Clicked in Font Panel

Drop-down menu buttons will have a down arrow to the right of the command. Just like buttons, the drop-down menu command will be surrounded by orange and the drop-down list will open up. **Change case** control is an example of a drop-down menu as shown in the Figure 135.

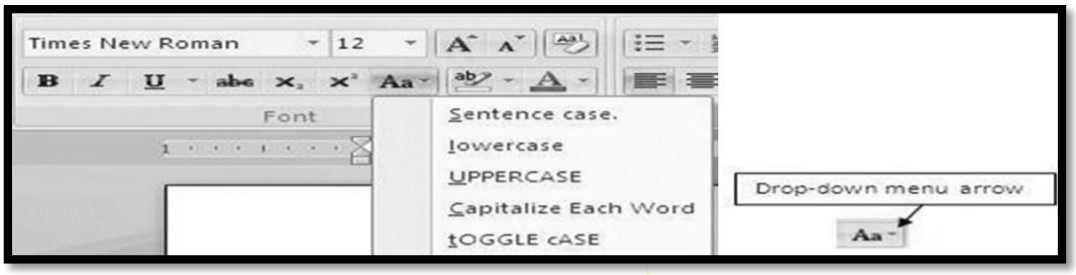


Figure 135: Font Panel Changing Case

Some commands are a combination of a **simple button** and a **drop-down menu button**. There are two buttons for these commands. The left button is a simple button and the right buttons is drop-down menu arrow button. If one clicks on the simple button the default value will be used, if one click on the drop-down menu arrow there will be a number of options available from which to choose. The default value is the last used value for the command. **Underline** is an example of a combination command as shown in the Figure 136.

Copyright Virtual University of Pakistan

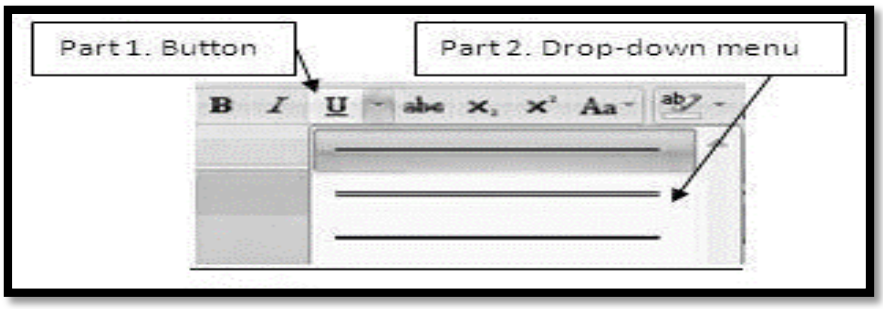


Figure 136: Underline Panel

Launcher (or Dialog Expanders) buttons

Launcher (or Dialog Expanders) buttons as shown in figure 8are usually displayed in the bottom right corner of some groups. Clicking the launcher button will open up a series of options either in a Task pane (such as the Clipboard) or a Dialog window such as the Font group.



Figure 137: Panel Launch Buttons

Mini Tool Bar

The Mini Toolbar appears automatically whenever you select text and contains common text formatting commands as shown in the Figure 138. To Use the Mini Toolbar: Select the text you want to format and click the desired command on the Mini Toolbar. Click anywhere outside the Mini Toolbar to close it.

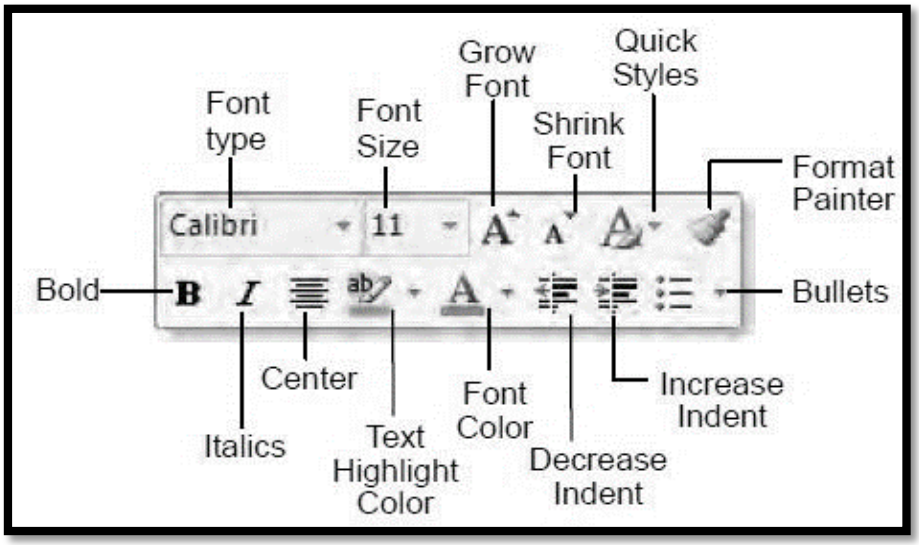


Figure 138: Mini Toolbar

Copyright Virtual University of Pakistan

Module 188

188. Word Processing: MS Word (Quick Access bar)

The Quick Access Toolbar (Figure 139) provides easy access to the commands you use most frequently. The Save, Undo, Redo/Repeat, and Quick Print buttons appear on the Quick Access Toolbar by default, but you can add and remove commands to meet your needs. You can quickly reverse most commands you execute by using Undo. If you then change your mind again, and want to reapply a command, you can use Redo as shown in the Figure 139.

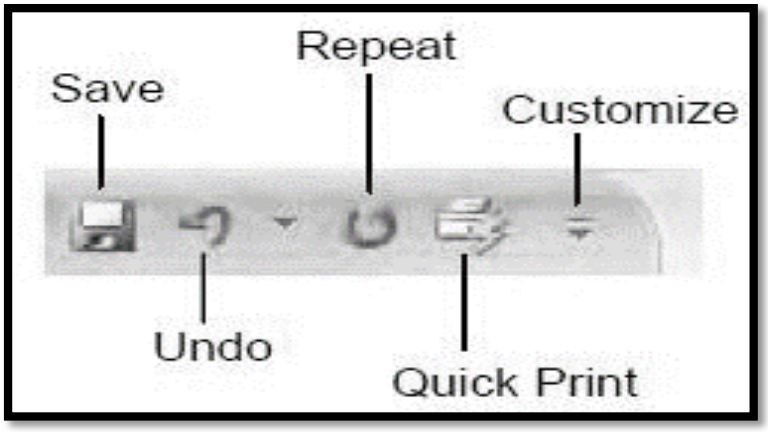


Figure 139: Quick Access toolbar

Exercise

1. Type word “Undo example”.
2. Click the Undo button on the Quick Access menu. The typing disappears.
3. Click the Redo button on the Quick Access menu. The typing reappears.
4. Select "Undo example."
5. Press Ctrl+b to bold. Word bolds the text.
6. Press Ctrl+i. Word italicizes the text.
7. Press Ctrl+u Word underlines the text.
8. Click the down arrow next to the Undo icon. You will see the actions you performed listed. To undo the underline, click Underline; to undo the underline and italic, click Underline Italic; to undo the underline, italic, and bold click Bold etc.
9. To redo, click the Redo icon several times.

Alternate Method -- Undo & Redo by Using Keys

1. Type word “Undo example”.
2. Press Ctrl+z. The typing disappears.
3. Press Ctrl+y. The typing reappears.
4. Select "Undo example."
5. Press Ctrl+u to underline.
6. Press Ctrl+z. The underline is removed.
7. Press Ctrl+y. The underline reappears.

Copyright Virtual University of Pakistan

Module 189

189. Word Processing: MS Word (Home Ribbon)

The home ribbon is one of the most important ribbon available in the MS Word. Figure 140 shows the home ribbon. It contains different important groups such as: Clipboard, Font, Paragraph, Styles and Editing. We will be learning each module of home ribbon in the next modules in details.

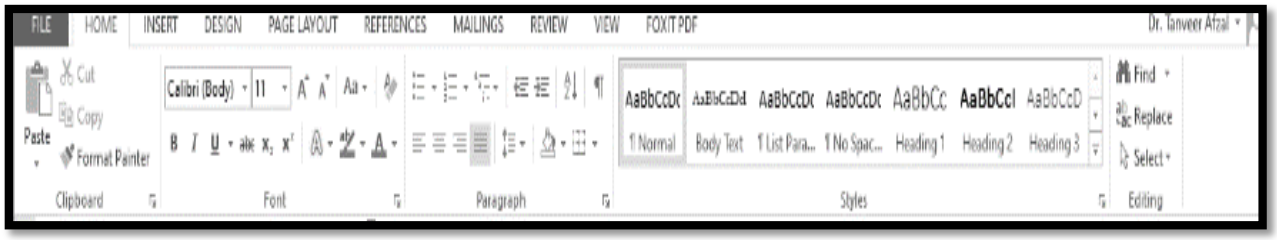


Figure 140: Home ribbon

Copyright Virtual University of Pakistan

Module 190

190. Word Processing: MS Word (Clipboard Group)

The **Home** ribbon is made up of the most used commands in Word. The first group on the **Home** ribbon is the **Clipboard. Copy**, **Cut**, **Paste** and **Format Paste** are the commands within the Clipboard group as shown in the Figure 141. The Clipboard commands are on the Home ribbon of **Word** and all other Office 2007 applications that use the ribbon. The dialog expander arrow of the Clipboard group will open up the **Clipboard pane**, showing all items that can be pasted.



Figure 141: Clipboard Group

Cut and Paste

You can use Word's Cut feature to remove text from a document. You can use the Paste feature to place the information you cut anywhere in the same or another document. In other words, you can move information from one place in a document to another place in the same or different document by using the Cut and Paste features. The Office Clipboard is a storage area. When you cut, Word stores the data you cut on the Clipboard. You can paste the information that is stored on the Clipboard as often as you like.

Cut with the Ribbon

1. Type the following:
   I want to move. I am content where I am.
2. Select "I want to move."
3. Choose the Home tab.
4. Click the Cut button 

in the Clipboard group as shown in the Figure 142. Word cuts the text you selected and places it on the Clipboard. Your text should now read:
"I am content where I am."

Copyright Virtual University of Pakistan

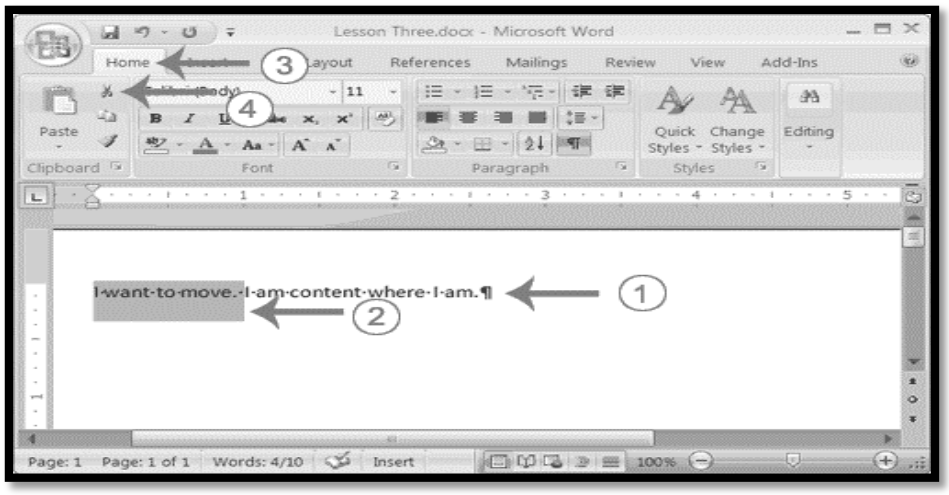


Figure 142: Cut option in clipboard group

Paste with the Ribbon

1. Place the cursor after the period in the sentence
   "I am content where I am."
2. Press the spacebar to leave a space.
3. Choose the Home tab.
4. Click the Paste button 

in the Clipboard group as shown in the Figure 143. Word pastes the text on the Clipboard. Your text should now read:

"I am content where I am. I want to move."

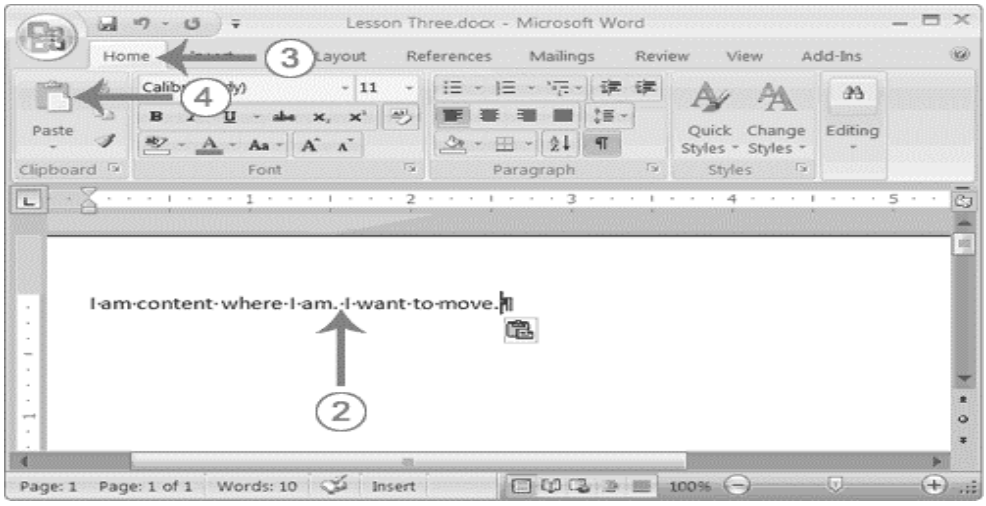


Figure 143: Pasting in Document

Alternate Method—Cut with a Context Menu

1. Type the following:
   I want to move. I am content where I am.
2. Select "I want to move."

Copyright Virtual University of Pakistan

3. Right-click. The Mini toolbar and a context menu appear.
4. Click Cut on the menu as shown in the Figure 144. Your text should now read: "I am content where I am."

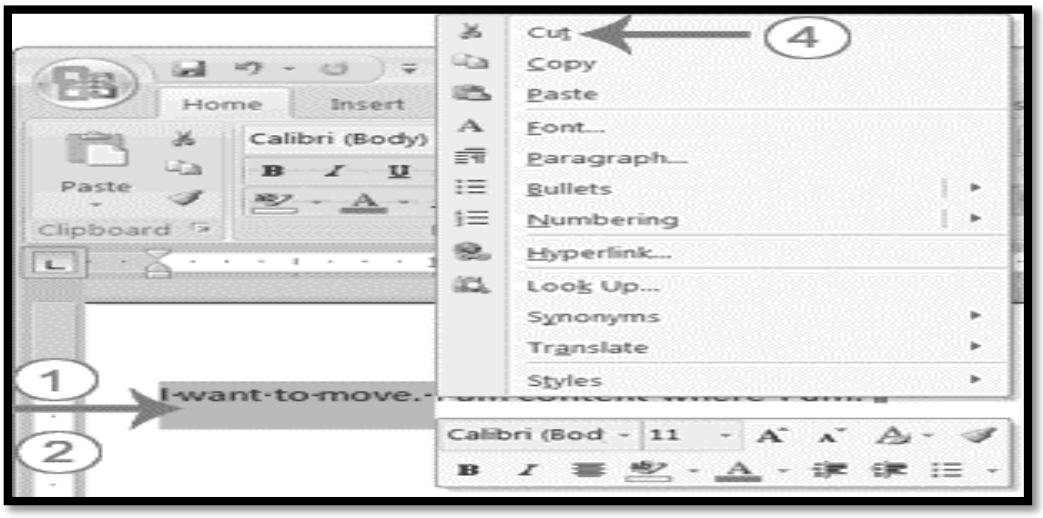


Figure 144: Pasting in Document (Alternate Method)

Alternate Method—Paste with a Context Menu

1. Place the cursor after the period in the sentence "I am content where I am."
2. Press the spacebar to leave a space.
3. Right-click. A Mini toolbar and a context menu appear as shown in the Figure 145.
4. Click Paste. Your text should now read:

"I am content where I am. I want to move."

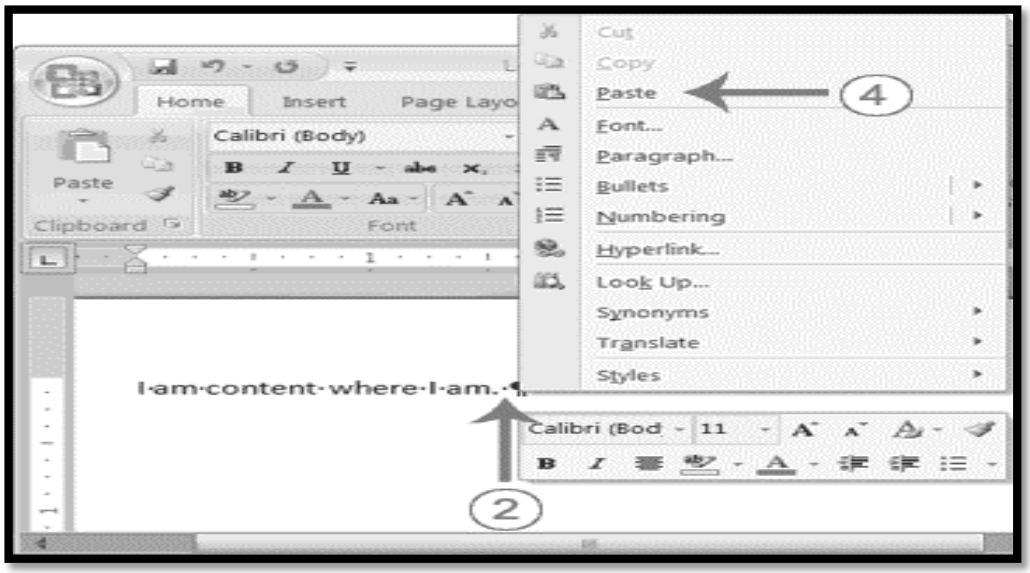


Figure 145: Pasting in Document (Alternate Method)

Copyright Virtual University of Pakistan

Alternate Method—Cut with Keys

1. Type the following:

I want to move. I am content where I am.

2. Select "I want to move."
3. Press Ctrl+x.

Your text should now read: " I am content where I am."

Alternate Method—Paste with Keys

1. Place the cursor after the period in the sentence: "I am content where I am."
2. Press the spacebar to leave a space.
3. Press Ctrl+v.
4. Your text should now read:
   "I am content where I am. I want to move."

Copyright Virtual University of Pakistan

Module 191

191. Word Processing: MS Word (Font Group)

The second group of commands on the Home ribbon is the Font group. The font group commands are format enhancing tool that includes font typefaces, font size, font effects (bold, italics, underline, etc.), colors and more. Remember that you can preview how the new font will look by highlighting the text, and hovering over the new font typeface as shown in the Figure 146. The expander arrow in the Font group will open up the Font dialog window.

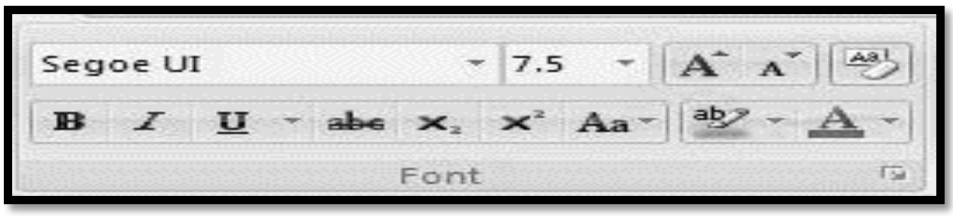


Figure 146: Font group

7.1.1 Basic Text Editing (Font Group Task)

Type your name in document window at insertion point as shown below in the Figure 147.



Figure 147: Typing your name

Change the font of your name to Arial. (Home Tab -> Font Group -> Font Arrow) as shown in the Figure 148.



Figure 148: Changing font

Copyright Virtual University of Pakistan

By keep selecting the name, make it bold and underlined. (Home Tab ->Font Group -> Bold & Underline Buttons) as shown in the Figure 149.

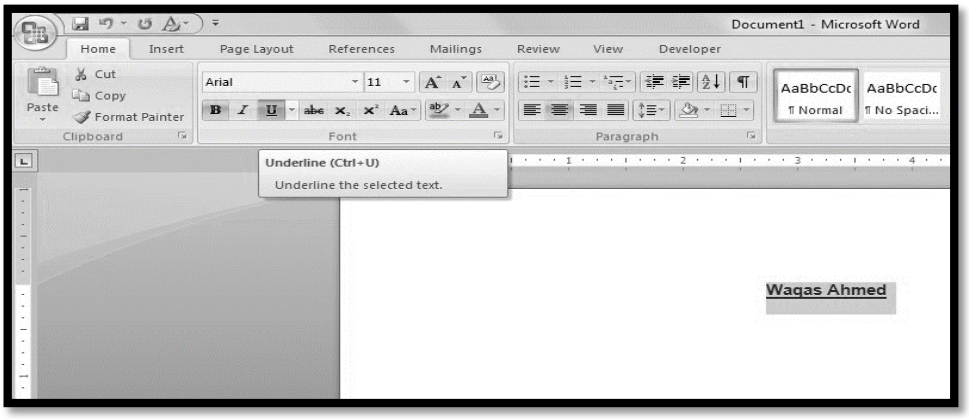


Figure 149: Applying bold and underline

Change the font size of your name to 24. (Home Tab -> Font Group -> Font Size Arrow) as shown in the Figure 150.

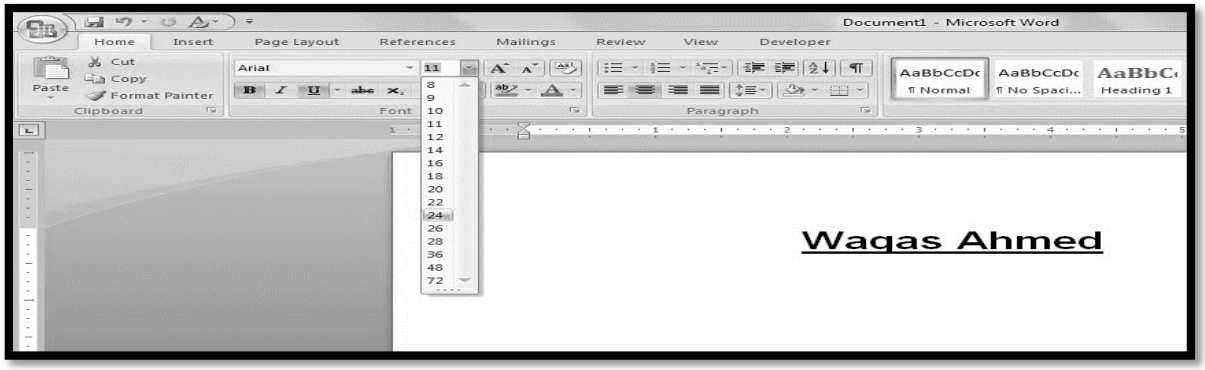


Figure 150: Changing Font Size

Copyright Virtual University of Pakistan

Module 192

192. Word Processing: MS Word (Paragraph Group-I)

The third group on the Home ribbon is the **Paragraph** group. The paragraph group allows you to change **Paragraph Alignment** (left, right, centered, or justified), adjust **Line Spacing** within a paragraph, and adjust **spacing before and after paragraphs,** along with working with paragraph **Indentation** as shown in figure 151**.** This is also the area where you can add **bullet lists**, number **lists**, or **outlines form** to a documents. The Dialog expander arrow will open the **Paragraph dialog** window.

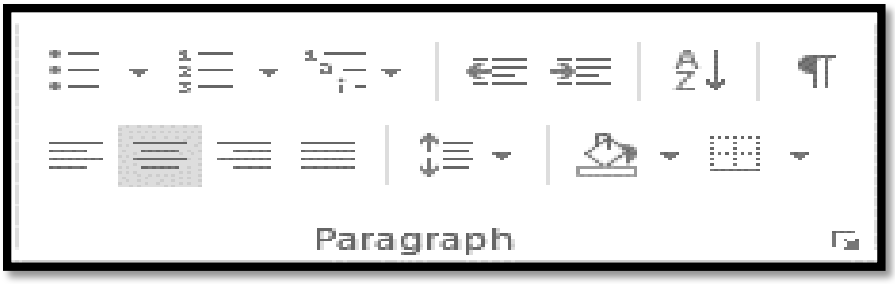


Figure 151: Paragraph Group

Bullets and Numbers

If you have lists of data, you may want to bullet or number them. When using Microsoft Word, bulleting and numbering are easy. The first part of this lesson teaches you to bullet and number.

In Microsoft Word, you can easily create bulleted or numbered lists of items. Several bulleting and numbering styles are available, as shown in the examples. You can select the one you wish to use as shown in the Figure 152 and Figure 53.

1.	Apple	1)	Apple	i.	Apple
2.	Orange	2)	Orange	ii.	Orange
3.	Grape	3)	Grape	iii.	Grape
4.	Mango	4)	Mango	iv.	Mango
5.	Cherry	5)	Cherry	v.	Cherry
A.	Apple	a)	Apple	a.	Apple
B.	Orange	b)	Orange	b.	Orange
C.	Grape	c)	Grape	c.	Grape
D.	Mango	d)	Mango	d.	Mango
E.	Cherry	e)	Cherry	e.	Cherry

Figure 152: Numbers varriations in paragraph group

Copyright Virtual University of Pakistan

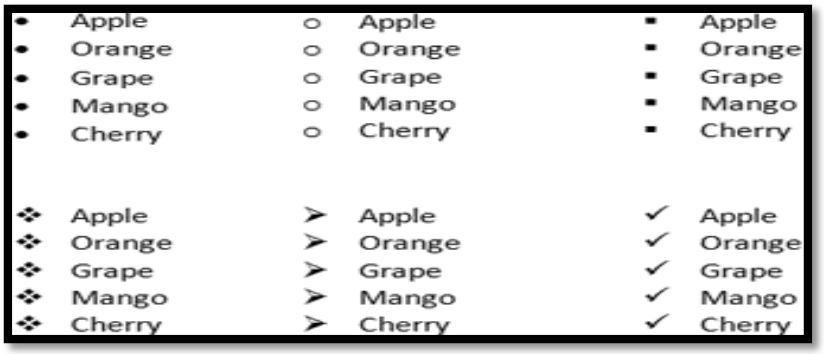


Figure 153: Bullets variations in paragraph group

Copyright Virtual University of Pakistan

Module 193

193. Word Processing: MS Word (Paragraph Group-II)

You can create your own bullet library. Perform the following steps:

1. Type the following list as shown:

Apple
Orange
Grape
Mango
Cherry

2. Select the words you just typed.

3. Choose the Home tab.

4. In the Paragraph group, click the down arrow next to the Bullets button . The Bullet Library appears.

5. Click to select the type of bullet you want to use as shown in the Figure 154. Word adds bullets to your list.

Note: As you move your cursor over the various bullet styles, Word displays the bullet style onscreen.

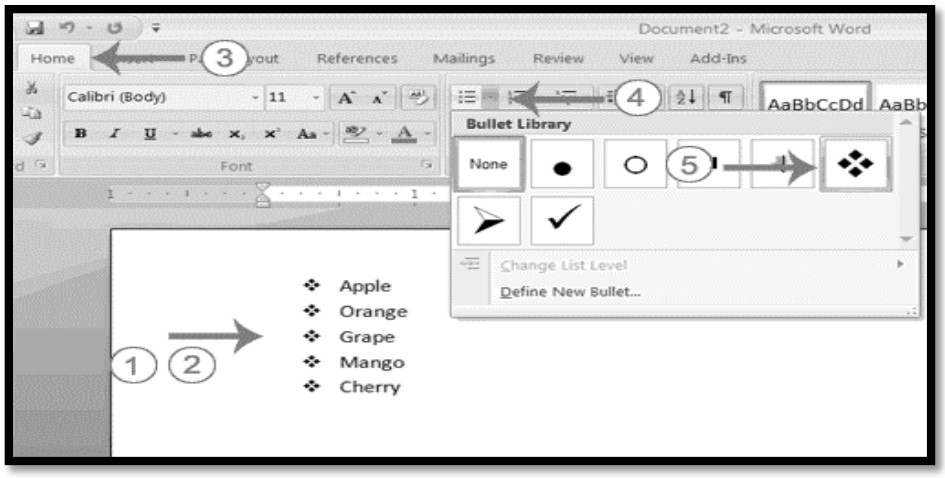


Figure 154: Bullet selection

To remove the bulleting:

1. Select the list again.
2. Choose the Home tab.
3. In the Paragraph group, click the down arrow next to the Bullets icon. The Bullet dialog box appears.
4. Click None. Word removes the bullets from your list.

Copyright Virtual University of Pakistan

Numbers Task

1. Type the following list as shown:

Apple Orange Grape Mango Cherry

2. Select the words you just typed.
3. Choose the Home tab.
4. In the Paragraph group, click the down arrow next to the Numbering button  . The Numbering Library appears as shown in the Figure 155.

Click to select the type of numbering you want to use. Word numbers your list. **Note:** As you move your cursor over the various number styles, Word displays the number style onscreen.

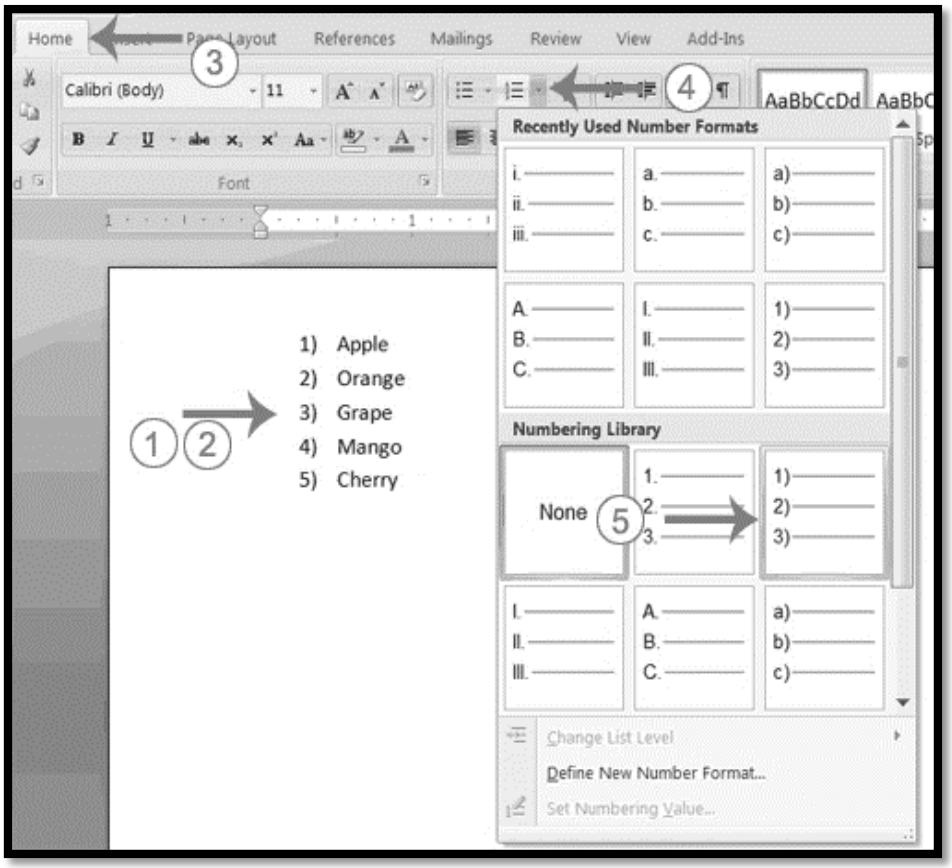


Figure 155: Adding Numbered Lists

To remove the numbering:

1. Select the list again.
2. Choose the Home tab.
3. In the Paragraph group, click the down arrow next to the Numbering icon. The Number dialog box appears.
4. Click None. Word removes the numbering from your list.

Copyright Virtual University of Pakistan

Module 194

194. Word Processing: MS Word (Style Group)

The fourth group on the Home ribbon is the **Style** group as shown in the Figure 156. Styles are a collection of formatting options that you can apply to text. When you use styles to format your document, you can quickly and easily apply a set of formatting choices consistently throughout your document.



Figure 156: Style Group

A **style** is a set of **formatting characteristics**, such as font name, size, color, paragraph alignment and spacing. Some styles even include borders and shading. For example, instead of taking three separate steps to format your heading as 16-point, bold, Cambria, you can achieve the same result in one step by applying the built-in Heading 1 style. You do not need to remember the characteristics of the Heading 1 style. For each heading in your document, you just click in the heading (you don't even need to select all the text), and then click Heading 1 in the gallery of styles.

Copyright Virtual University of Pakistan

Module 195

195. Word Processing: MS Word (Editing Group)

The Fifth and final group (Figure 157) on the Home ribbon is the **Editing** Group. The commands in the Editing group are **Find**, **Replace**, and **Select**. When you create a document in Microsoft Word, you may decide to change a certain word or phrase that is repeated throughout the document. Let the computer do the hard work with the **Find** and **Replace** feature in Word and you can be sure you didn't miss any.



Figure 157: Editing group

Under the **Find** command, there is the **Go To** command. The Go To command can be used to immediately go to a page, a section or any marked location within your document.

The **Select** command can be used for different purpose.

- To select all of the text in the document, click **Select All**.
- To select shapes that are hidden, stacked, or behind text, click **Select Objects**, and then draw a box over the shapes.
- To select text with similar formatting, click **Select Text with Similar Formatting**.

Copyright Virtual University of Pakistan

Module 196

196. Word Processing: MS Word (Insert Functionalities)

Insert Tab can be used if you want to add to graphics or link your document with another document. The initial part of the insert tab in word ribbon has been shown in the Figure 158.

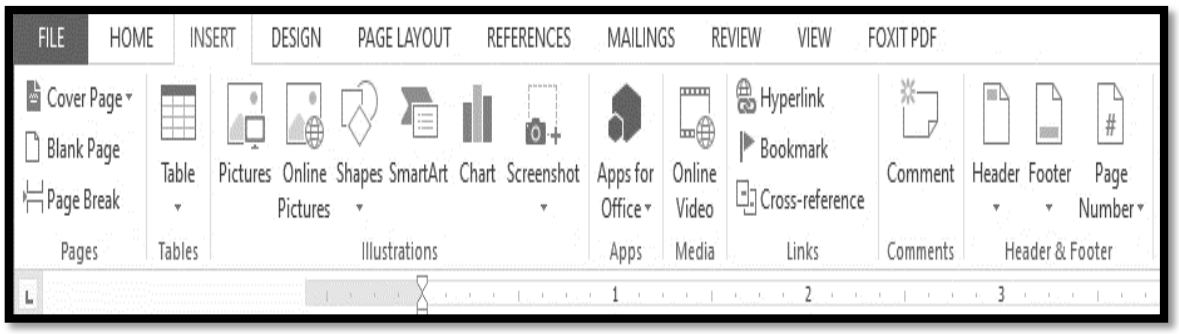


Figure 158: Inert tab

In the insert tab, the following groups are available which will be covered in the next modules:

- ✓ Pages Group
- ✓ Tables Group
- ✓ Illustrations Group
- ✓ Media
- ✓ Links
- ✓ Comments
- ✓ Header & Footer
- ✓ Text
- ✓ Symbols

Copyright Virtual University of Pakistan

Module 197

197. Word Processing: MS Word (page Group)

Pages group is in the far left of the insert tab. You can insert from the three categories for pages which are; Cover Page, Blank Page, and Page Break as shown in the Figure 159. These features are useful if you are creating a professional or long document.



Figure 159: Pages group in the insert tab

Cover Page

Cover Page is the title page of the document. Word 2007 provides a number of preformed cover pages to give your document a professional look.

Cover page Exercise: Creating a Cover Page

1. Click the **Insert** tab to make it the active tab as shown in the Figure 158.
2. To open the **Built-in Cover Page Gallery**, click the **down arrow** to the right of **Cover Page** in the **Pages** group on the **Insert** tab as shown in the Figure 160.

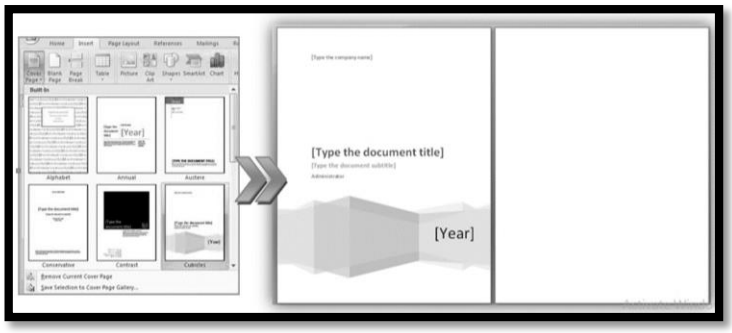


Figure 160: Inserting Cover Page

3. Scroll down and **click** the desired cover for the document from the **Cover Page Gallery**. Use the scroll bar or scroll arrow to see all the cover page choices.
4. A Cover page will be added to your document.

Remove the Cover Page

1. Click the **Insert** tab to make it the active tab.
2. To open the **Built-in Cover Page Gallery**, click the **down arrow** to the right of **Cover Page** in the Pages group on the **Insert** tab.

Copyright Virtual University of Pakistan

3. Scroll down and select **Remove Current Cover Page** as shown in the Figure 161.

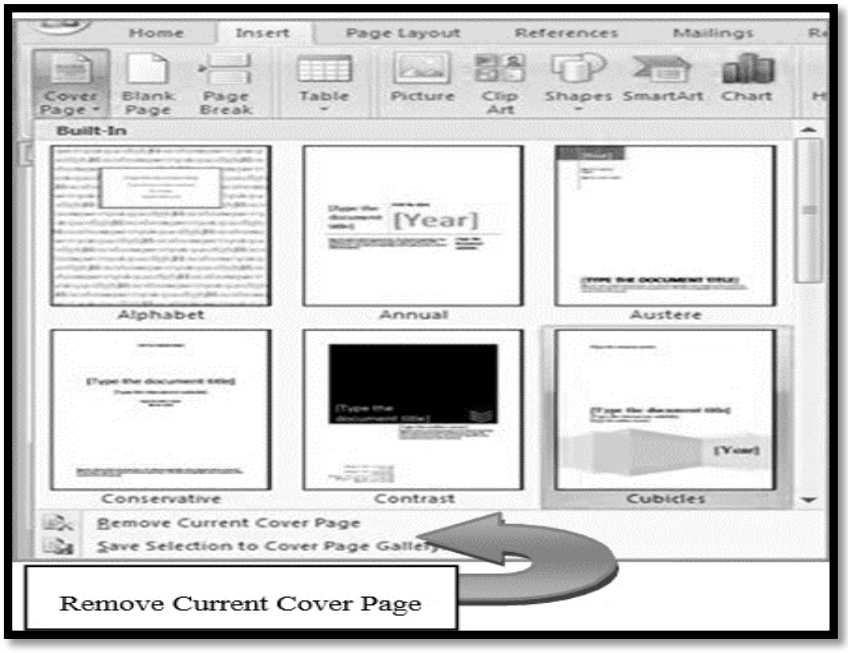


Figure 161: Removing cover page

Blank Page

Generally, Word inserts a blank page automatically while you are typing. Blank Page in Insert tab can help you add a new page anywhere in a document when required.

Try inserting a blank page to your document.

1. Click where you want to insert a new page i.e. before or after some text.
2. Click on Blank Page in Insert Tab (Figure 158).

Page Break

Page Break button will force a page break anywhere you select in your document.

1. click where you want to insert Page Break
2. Click page break (Figure 158).

Copyright Virtual University of Pakistan

Module 198

198. Word Processing: MS Word (Table Group)

Word provides tables option for placing data in a more formal way.

1. Click on table in insert tab. (Figure 162).



Figure 162: Tables group

2. To add Rows and Columns, select top to bottom for rows & left to right for columns (Figure .163)
3. Click to apply when happy with the selection.
4. Click on the box to enter text.
5. Observe the new tab opened "Table Tools". You can use this tool to apply further formatting to your created table

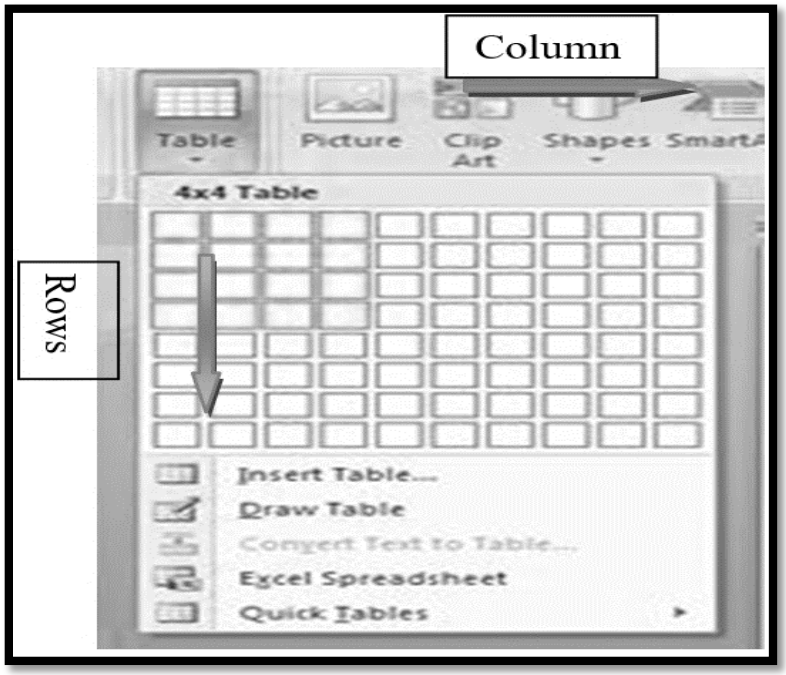


Figure 163: Selecting Rows and Columns

Copyright Virtual University of Pakistan

Module 199

199. Word Processing: MS Word (Illustration Group)

Using Illustrations (Figure 164), you can add pictures of all types and styles to your document. Followed is detail description on how to add illustrations.

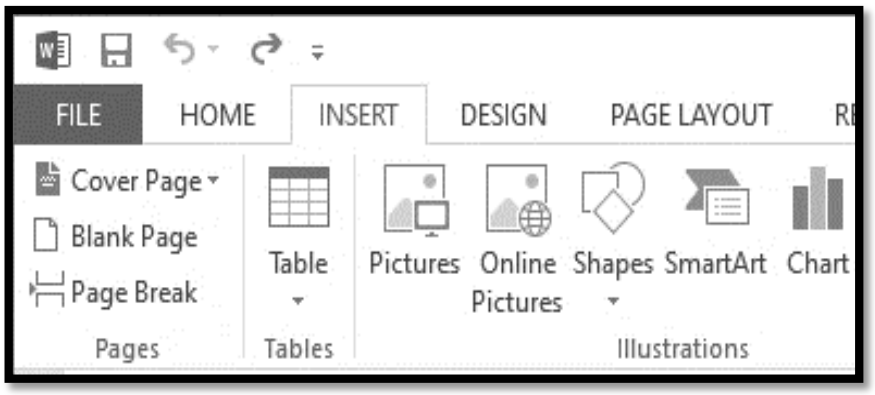


Figure 164: Illustrations

Picture

The first selection in Illustrations is Insert a picture from a file. When you click on this a window will open for you to browse to a photograph or other picture you have saved on your computer.

Clip Art

To illustrate a specific concept, you can Insert Clip Art in your document, including media file (sound and video clips) and drawing.

Shapes

Word 2007 provides a wide range of shapes (Figure 165) to select and insert in your document as per your requirement. It also provides Canvas to place all the Shapes at one place without being disturbed when further changes are made as shown in the figure 166. Each shape belongs to one of the following subcategory i.e. Lines, Basic Shapes, Block Arrows, Flow Charts, Callouts, Stars and Banners.

1. Click on Shapes in Insert Tab and draw a new Canvas Figure 165.
2. Insert shapes given in Figure 166.
3. Arrange inserted shape to form Snow Man as done in Figure 167.

Copyright Virtual University of Pakistan

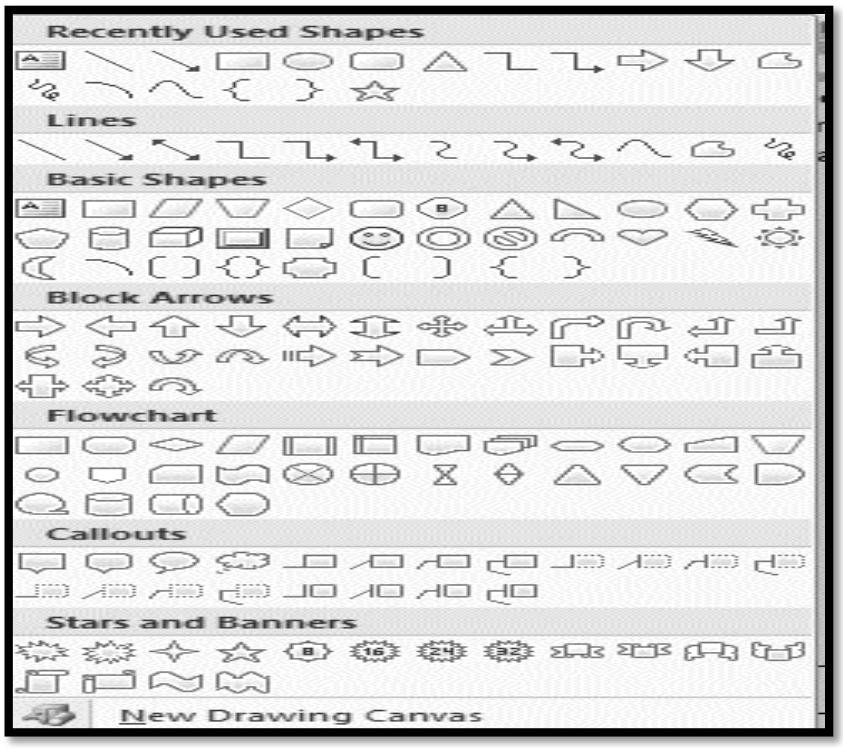


Figure 165: Shapes types in Word

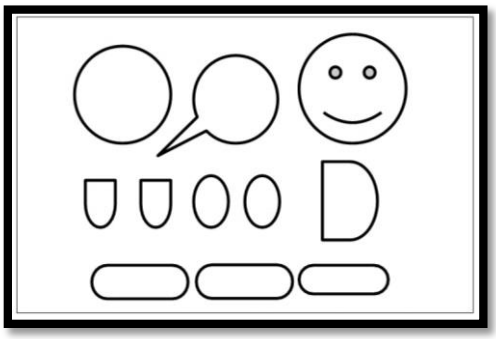


Figure 166: Shapes required to make snowman

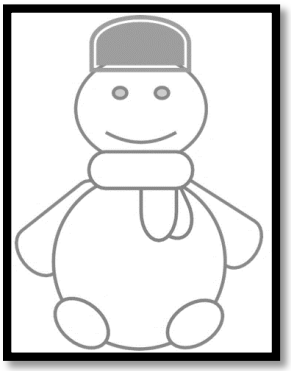


Figure 167: Snowman

Copyright Virtual University of Pakistan

Smart Art

A new feature introduced in Word 2007 is Smart Art. Using Smart Art, you can communicate your ideas visually. Whether it's a process, hierarchy or life cycle all can be illustrated using Smart Art as shown in the Figure 168.

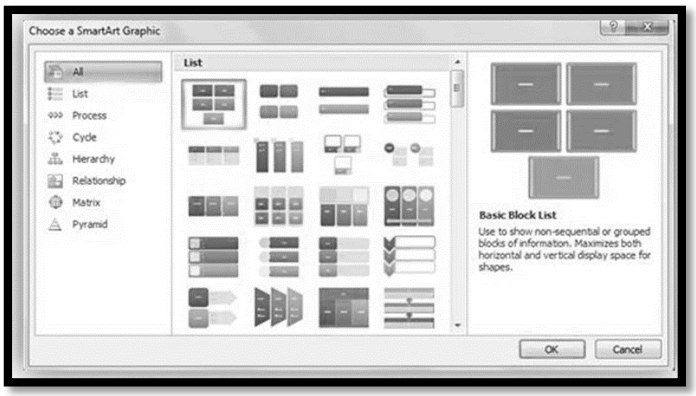


Figure 168: Smart Art

Exercise: How to add Cycle Smart Art

1. Click on Smart Art Button in Illustration group of Insert Tab
2. Click Cycle in the left pane of opening window (Figure 169)
3. Select Basic Cycle Figure 169.
4. Click On [TEXT] to add text. "Baby", "Child", "Teen" and "Adult". Notice the extra Circle left (Figure 170).

Click on the remaining circle and press Delete (Del) Key on your keyboard to remove it (Figure 171).

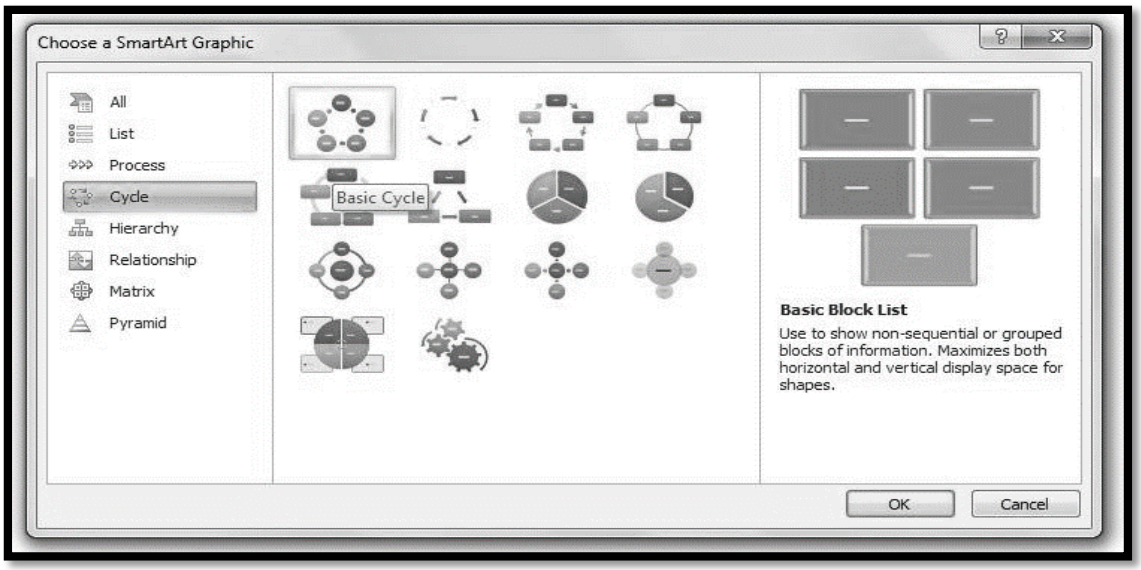


Figure 169: Basic cycle

Copyright Virtual University of Pakistan

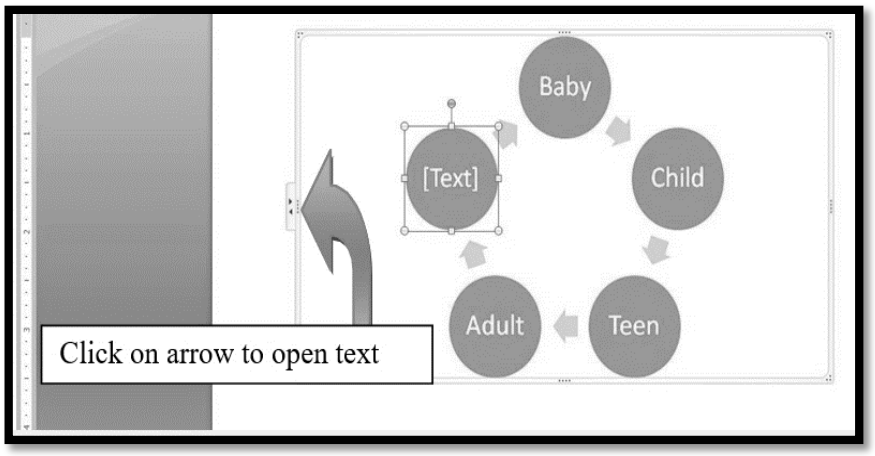


Figure 170: Human Life Cycle

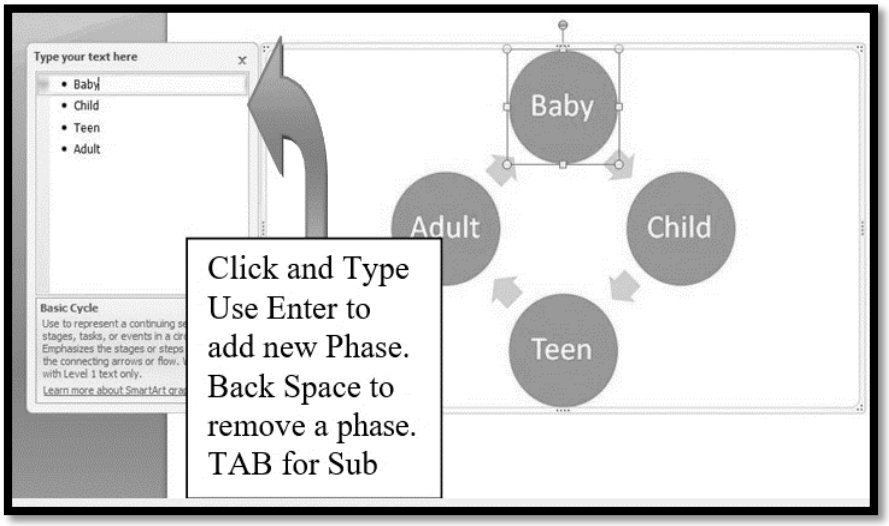


Figure 171: Adding and Removing Objects

Copyright Virtual University of Pakistan

Module 200

200. Word Processing: MS Word (Media and Links Groups)

Links toolbox (Figure 172) in Insert tab helps link other documents in your current document and also helps you keep track of reading by providing Bookmark facility.

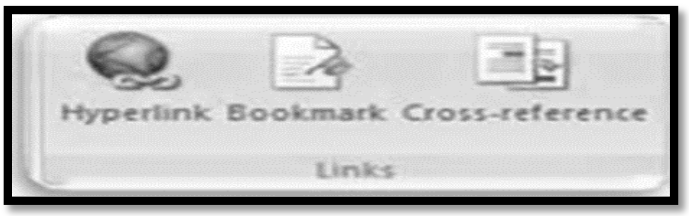


Figure 172: Links group

Hyperlink

Hyperlink is used to add link of other documents or web pages in your Document. Here is how you can add Hyperlink. Click on hyperlink in insert ribbon tab, a window will open as in the Figure 173.

1. Enter word or phrase to display hyperlink
2. Click to add text displayed when mouse is over the hyperlink text
3. Select Bookmark
4. Select respective option to locate file or web links that you want to add in document
5. Address bar
6. You can link to various files i.e. existing file in your system, link to place within document (ideal to link table of contents with headings).

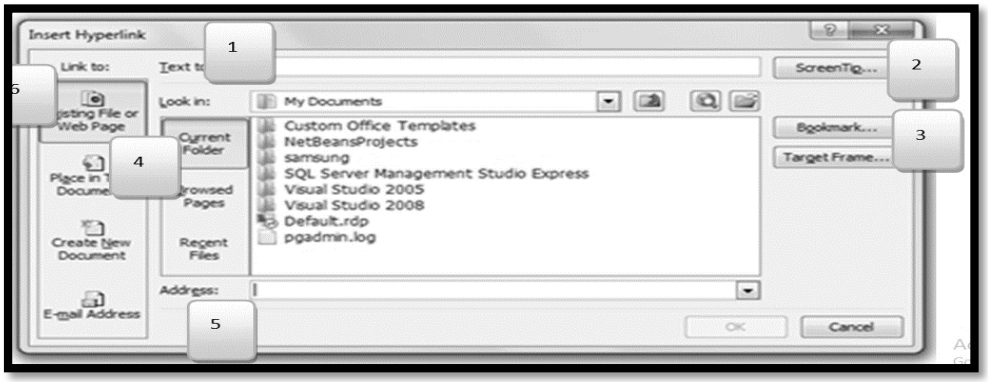


Figure 173: Adding Hyperlink

Bookmarks

Bookmarks are a way to keep track of reading. They are similar to bookmarks you keep while reading a favorite novel or story book like, turn the corner of the page or place pen or pencil.

Bookmarks Exercise: How to Use Bookmark

1. Click where you want to place bookmark.
2. From insert ribbon tab select bookmark (Figure 174)
3. Enter bookmark name say" Mark1" (without space) and click add.

Copyright Virtual University of Pakistan

4. To continue from where you placed bookmark repeat step 2 and select saved "Mark1" bookmark.

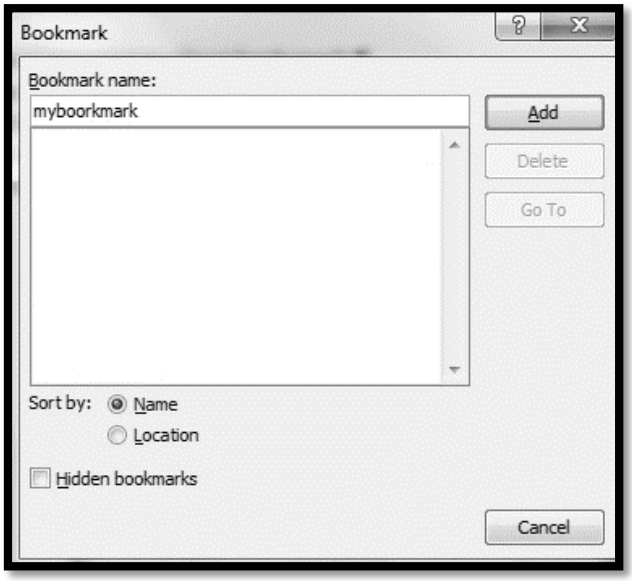


Figure 174: Adding Bookmark

Copyright Virtual University of Pakistan

Module 201

201. Word Processing: MS Word (Comments and Header & footer)

Header and Footers are areas where you can add Text or Graphics. Any formatting applied in these areas will be shown throughout the document. A header is at the top margin; a footer is printed in the bottom margin For Example observer the top of this page and all the pages in the document you will see same header and footer format.



Figure 175: Header and Footer

Exercise: How to Use Header & Footers

1. Click on insert ribbon tab select header.
2. Page header will open in edit mode. (Figure 176)
3. Enter text "Virtual University of Pakistan, as shown in the Figure 176.
4. Close header& footer (Figure 177).

Similarly, you can add footer. Please note you can even add images to the header, footer or both.

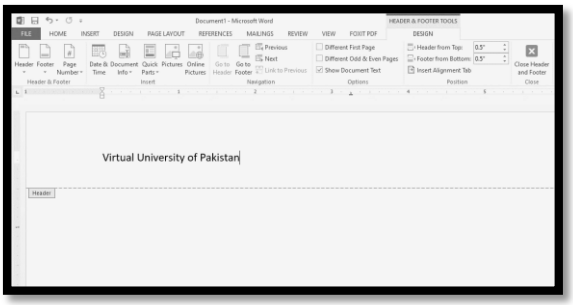


Figure 176: Header in edit mode

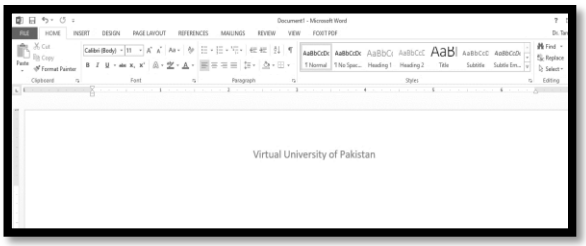


Figure 177: After Header Insertion

Copyright Virtual University of Pakistan

Module 202

202. Word Processing: MS Word (Text Group (Part-1))

Text Box in Insert Ribbon tab can be used to add graphical formatting. This has been shown in the Figure 178.



Figure 178: Text group

Exercise: How to Use Text box (Figure 179)

1. Click the Text Box button, located in the Text group on the Insert tab. From the menu that appears, choose Draw Text Box.
2. Click where you want one corner of the text box to appear and drag to where you want the opposite corner.
3. Release the mouse button and then type your text.

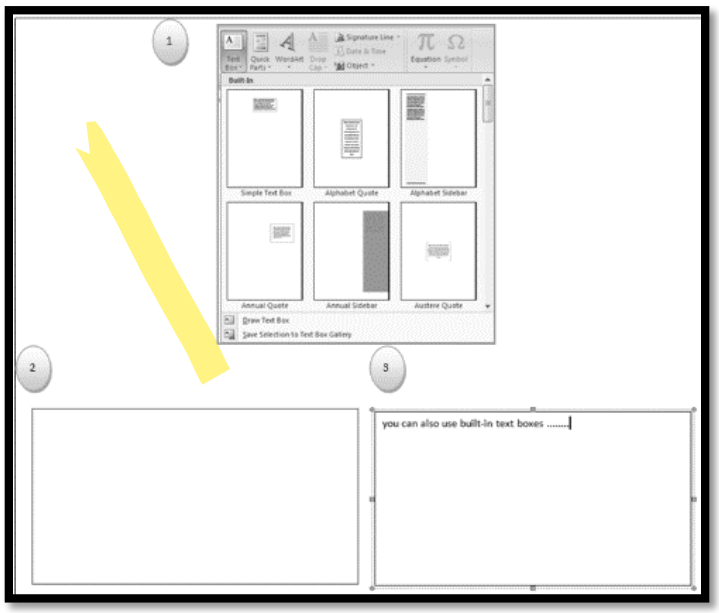


Figure 179: Adding Text Box

Quick parts

Quick Parts are a new feature in Word 2007 that allows you to save text or graphics in what Microsoft calls "Building Blocks" and then call it up quickly when you need it. This is especially handy for inserting information such as a Company Name, Logo, Company Address or any other information that you use often.

Exercise: How to Create a Quick Part

1. Select the text you want to reuse. To include paragraph formatting, make sure you select the ending paragraph mark.
2. Click the Insert tab on the Ribbon.
3. Click the Quick Parts button and click Save Selection to Quick Part Gallery.
4. Enter in the desired properties for the Quick Part.
5. To insert a Quick Part that you have saved, click the Quick Parts button and choose the Quick Part you want to use.

Word Art

WordArt can be used to produce special text effects in a Microsoft Word document. For example, you can create curved, slanted, or three-dimensional text by inserting a WordArt object.

Copyright Virtual University of Pakistan

Exercise: How to add Word Art

1. On the Insert tab and then Click WordArt.
2. In the WordArt Gallery dialog box, double-click the style that you want (Figure 180).
3. In the Edit WordArt Text dialog box, type your text and select font and size.
4. Click Bold or Italic to make all the text bold or italic. (Note you cannot apply bold or italic to only a part of the text.).

5. In the Edit WordArt Text dialog box, click OK. To insert into the document.

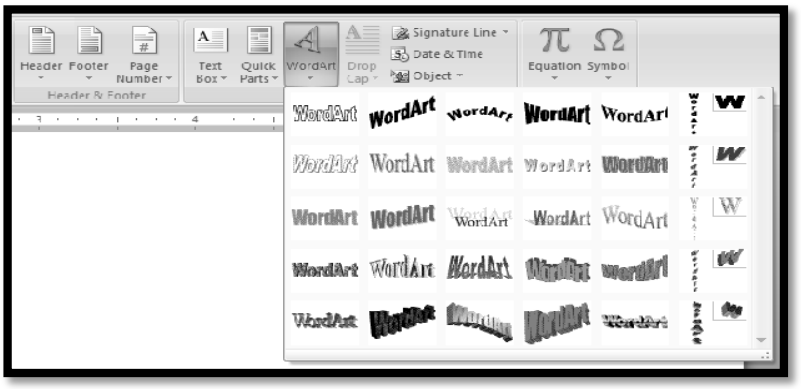


Figure 180: Word Art Gallery

Copyright Virtual University of Pakistan

Module 203

203. Word Processing: MS Word (Text Group (Part-II) and Symbols Group)

Drop Cap

A drop cap is a specially formatted letter that appears at the beginning of a paragraph just like you see in newspapers or in books. Microsoft word offers drop caps in two styles. Most commonly used drop cap begins the paragraph with a large letter that spills down into the text. Thus, the drop cap shifts the first few lines of the paragraph.

Exercise: How to Use Drop Cap

1. Type the paragraph as you normally would.
2. Open the Insert tab on the Ribbon and click the Drop Cap button.(Figure 181)
3. Choose the drop cap style you want to use.
4. Adjust the drop cap, if you want to.

Symbols

Another feature found is Microsoft Word is Symbols. You can add pre-made mathematical Equations and various Symbols where required in your document. However, if you cannot find what you want, you can also create your own equations.

Exercise: Using Symbols

1. Locate the Equation button in the Symbols group of the Insert tab on the Ribbon.
2. Click the button (not its menu triangle), and two things happen. First, an equation control is inserted into your document at the insertion pointer's location. Second, the Equation Tools Design tab appears on the Ribbon.

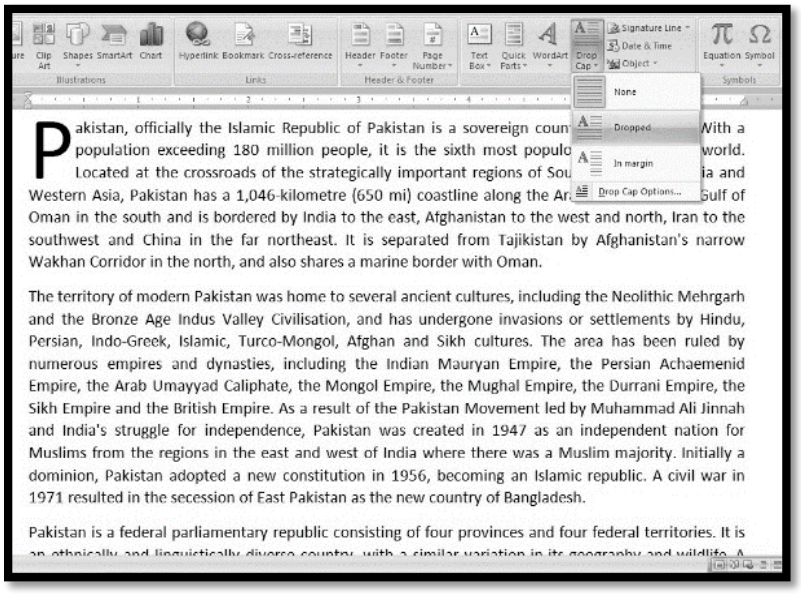


Figure 181: Drop cap 'P'

Copyright Virtual University of Pakistan

Module 204

204. Word Processing: MS Word (Design Ribbon)

Design ribbon (as shown in the Figure 182) contains the following groups:



Figure 182: Design Ribbon

Themes - Drop-Down. The drop-down contains the commands: Built-in, More Themes on Microsoft Office Online, Browse for Themes and Save Current Theme. The built-in themes are: Office, Apex, Aspect, Civic, Concourse, Equity, Flow, Foundry, Median, Metro, Module, Opulent, Oriel, Origin, Paper, Solstice, Technic, Trek, Urban and Verve. Tooltip indicates the current theme. The default theme is "Office". Tooltip indicates the current theme. Quick Access to more themes saved down on Microsoft Office Online. Default theme is "Office". Changes the overall design of the entire document including colors, fonts and effects. This drop-down has an additional command not found in Excel or PowerPoint which is "Reset to Theme from Template".
Style Set - Change the look of your document by choosing a new style set. Style sets change the font and paragraph properties of your entire document.

Colors - Drop-Down. Displays a list of all the available colors and lets you change the color component of the active theme.
Fonts - Drop-Down. Displays a list of all the available fonts and lets you change the font component of the active theme.
Paragraph Spacing - Drop-Down. Quickly change the line and paragraph spacing in your document. The drop-down contains the commands: No Paragraph Space, Compact, Tight, Open, Relaxed, Double, Custom Paragraph Spacing.

Effects - Drop-Down. Displays a list of all the available effects and lets you change the effect component of the active theme. The drop-down contains the commands: Office, Apex, Aspect, Civic, Concourse, Equity, Flow, Foundry, Median, Metro, Module, Opulent, Oriel, Origin, Paper, Solstice, Technic, Trek, Urban and Verve.
Set As Default - Use the current look for all new documents.

Page Background – This tab contains watermark, page color and page borders as shown in the Figure 183.

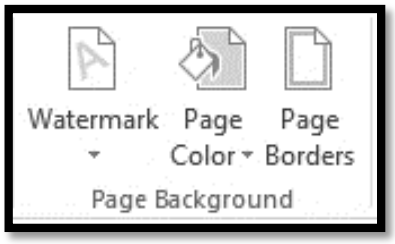


Figure 183: Page background group

Page Color – Drop-Down. Let's you change the background color of the page. Displays the full theme color palette.

Copyright Virtual University of Pakistan

Page Borders – Displays the "Border and Shading" dialog box.

Watermarks

Watermarks in the documents can be attractive, but they aren't just about looks: A watermark can be a way of letting the reader know that the document is confidential.

Exercise: How to add Watermark

1. Click the Watermark button on the Page Layout tab.
2. Click one of the watermarks to insert it or select Custom Watermark from the bottom of the gallery.
3. To select text for your custom watermark, select the Text Watermark option in the Printed Watermark dialog box. Then use the controls to select the language and the text to be used. Click OK.

You can also select a picture to be used for the watermark by selecting the Picture Watermark option and then using the Select Picture button to find the image you want to use. For most images, you want to leave the Washout option selected, which makes the text on top of the image easier to read.

Copyright Virtual University of Pakistan

Module 205

205. Word Processing: MS Word (Page Setup Group in Page Layout Ribbon)

Using page setup group (Figure 184), you can easily place margins, display text in two or three columns, Add section break or even set page setup.

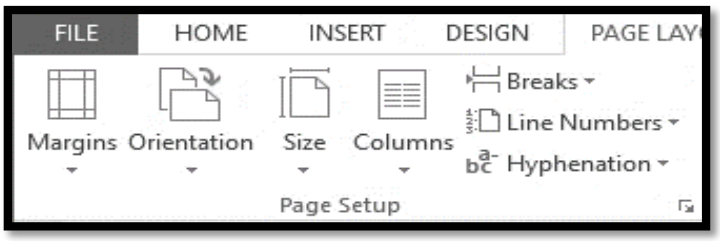


Figure 184: Page setup group

Displaying Text in two Columns

As you might have observed in newspapers text is displayed in two or three columns. Microsoft Word provides this feature in Page Layout Ribbon tab. By default the text is displayed in one column format. Here's how you can display in two columns.

Exercise: How to add Columns

1. Select text that you want to display in column form. (Figure 185)

In Page Layout Tab click on Columns and select "Two". See Figure 186.

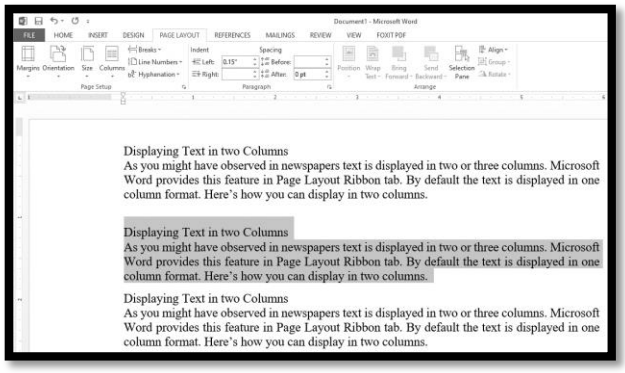


Figure 185: Text selections for 2-columns display

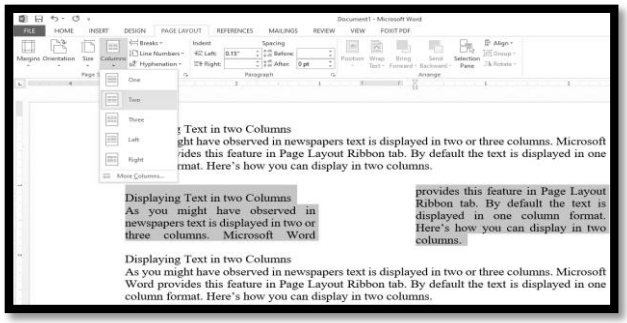


Figure 186: Text in two columns

Copyright Virtual University of Pakistan

Module 206

206. Word Processing: MS Word (Page Setup Group in Page Layout Ribbon)

You can quickly display the "Paragraph" dialog box, Indents and Spacing tab, by clicking on the launcher in the bottom right corner of this group. These are options taken from the (Format Paragraph)(Indents and Spacing tab) for quick access.

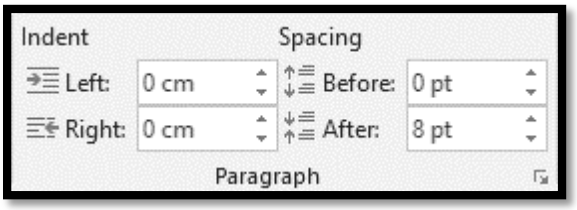


Figure 187: Paragraph group in page layout

Indent Left - TextBox. This automatically updates to indicate how much indentation has been applied to the paragraph of the current selection. This can be used to change the left indentation for the current selection.

Indent Right - TextBox. This automatically updates to indicate how much indentation has been applied to the paragraph of the current selection. This can be used to change the right indentation for the current selection.

Spacing Before - TextBox. This automatically updates to indicate how much spacing is defined before the paragraph of the current selection. This can be used to change the spacing for the current selection.
Spacing After - TextBox. The automatically updates to indicate how much spacing is defined after the paragraph of the current selection. This can be used to change the spacing for the current selection.

Copyright Virtual University of Pakistan

Module 207

207. Word Processing: MS Word (Arrange Group in Page Layout Ribbon)

This group also appears on the Drawing Tools - Format contextual tab as shown in the Figure 188.

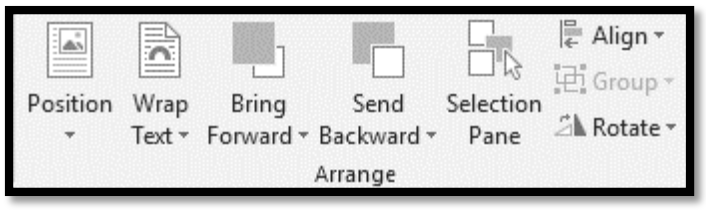


Figure 188: Arrange Group

Position - Drop-Down. Displays a list of picture positioning options. The drop-down contains the commands: In Line with Text and Text Wrapping. You can select More Layout Options to display the "Advanced Layout" dialog box.

Wrap Text – Drop-Down. The drop-down contains the commands: In Line with Text, Square, Tight, Through, Top and Bottom, Behind Text, In Front of Text, Edit Wrap Points and More Layout Options.

Bring Forward - Button with Drop-Down. The button brings the selected object forward one level. The drop-down provides a command to bring the selected object in front of all the other objects. **Send Backward** - (Send to Back in 2007). Button with Drop-Down. The button brings the selected object back one level. The drop-down provides a command to send the selected object to the back of all the other objects.

Selection Pane - (Added in 2010). Displays the Selection Pane task pane.

Align - Drop-Down. The drop-down contains the commands: Align Left, Align Center, Align Right, Align Top, Align Middle, Align Bottom, Distribute Horizontally, Distribute Vertically, Align to Page, Align to Margin, Align Selected Objects, View Gridlines and Grid Settings. The Grid Settings displays the "Drawing Grid" dialog box.

Group - Drop-Down. The drop-down contains the commands: Group, Regroup and Ungroup.

Rotate - Drop-Down. The drop-down contains the commands: Rotate Right 90, Rotate Left 90, Flip Vertical, Flip Horizontal and More Rotation Options.

Copyright Virtual University of Pakistan

Module 208

208. Word Processing: MS Word (References Ribbon)

This tab gives you access to all the commands for creating references within your documents.

Table of Contents - Drop-Down. Provide an overview of your document by adding a table of contents. The drop-down contains the commands: Built-in, Insert Table of Contents and Save Selection to Table of Contents Gallery as shown in the Figure189.



Figure 189: Table of Content

Add Text - Drop-Down. The drop-down contains the commands: Do Not Show in Table of Contents, Level 1, Level 2 and Level 3.
Update Table - Updates the table of contents so that all the entries refer to the correct page numbers.

Footnotes - You can quickly display the "Footnote and Endnote" dialog box by clicking on the dialog box launcher in the bottom right corner of this group as shown in the Figure 190.

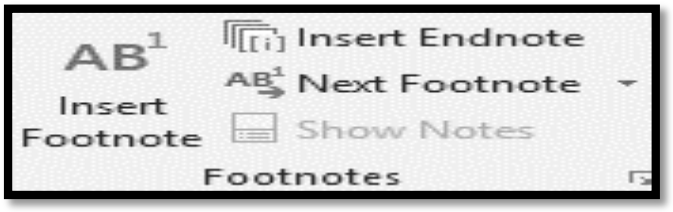


Figure 190: Footnotes Group

Insert Footnote - (Alt + Ctrl + F). Inserts a footnote at the current position. Footnotes are automatically renumbered as you move text around the document.

Insert Endnote - (Alt + Ctrl + D). Inserts an endnote at the end of the document. End notes are always placed at the end of a document.
Next Footnote - Button with Drop-Down. The button moves to the next footnote. The drop-down provides the commands Next Footnote, Previous Footnote, Next Endnote and Previous Endnote.
Show Notes - Shows where footnotes and endnotes are located.

Copyright Virtual University of Pakistan

Module 209

209. Word Processing: MS Word (Proofing Group in Review Ribbon)

The proofing group is shown in the Figure 191.

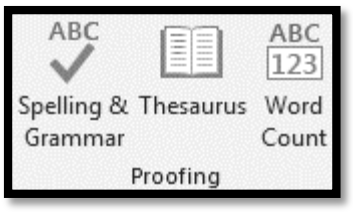


Figure 191: Proofing group

Spelling & Grammar - Displays the "Spelling and Grammar" dialog box. This allows you to check the spelling and grammar in the active document.

Thesaurus - Toggles the display of the Research task pane defaulting the research service to the thesaurus. Same as Excel.

Word Count - This displays the "Word Count" dialog box displaying the document statistics. This replaces the Word Count toolbar. This dialog box can also be displayed by clicking on the word count indicator on the status bar.

Copyright Virtual University of Pakistan

Module 210

210. Word Processing: MS Word (Language Group in Review Ribbon)

The language group has been shown in the Figure 192.



Figure 192: Language group

Translate - Drop-Down. Displays the Research task pane. This drop-down contains the commands: Translate Document, Translate Selected Text, Mini Translator and Choose Translation Language.
Language - Drop-Down. This drop-down contains the commands: Set Proofing Language and Language Preferences. Set Proofing Language display the "Language" dialog box. Language Preferences displays the "Options" dialog box, Language Tab.

Copyright Virtual University of Pakistan

Module 211

211. Word Processing: MS Word (Comments Group in Review Ribbon)

The comments group has been shown in the Figure 193.



Figure 193: Comments group

New Comment - (Shift + F2). Inserts a comment at the active cell. This command does not change to Edit Comment when a comment is selected like it does in Excel.

Delete - Button with Drop-Down. Deletes the selected comment. The button deletes the comment in the active selection. The drop-down contains the commands: Delete All Comments Shown and Delete all comments in Document. This is disabled when the document does not contain any comments.

Previous - Goes to the previous comment in the active document. This is disabled when the document does not contain any comments.

Next - Goes to the next comment in the active document. This is disabled when the document does not contain any comments.

Copyright Virtual University of Pakistan

Module 212

212. Word Processing: MS Word (Tracking and Changes Groups in Review Ribbon)

The Tracking group has been shown in the Figure 194.

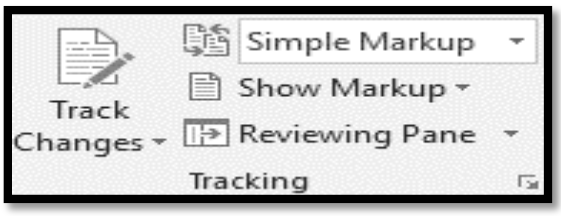


Figure 194: Tracking group

Track Changes - (Ctrl + Shift + E). Button with Drop-Down. The button is the Track Changes command from the Reviewing. It contains the commands Track Changes, Change Tracking Options and Change Username. Change Tracking Options displays the Track Changes Options dialog box. Change Username displays the Word Options dialog box (Popular tab).

Show Markup - Drop-Down. The drop-down contains the commands: Comments, Ink, Insertions and Deletions, Formatting, Markup Area Highlight.

Reviewing Pane - Button with Drop-Down. The button toggles the display of the Reviewing pane. The drop-down contains the commands: Reviewing Pane Vertical and Reviewing Pane Horizontally. There is now also summary information at the top of the pane.

The changes group is shown in the Figure 195.

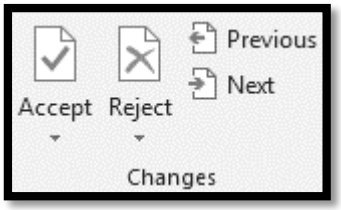


Figure 195: Changes group

Accept - Button with Drop-Down. The button is the Accept and Move to Next command which accepts the current change and moves to the next proposed change. The drop-down contains the commands: Accept and Move to Next, Accept Change, accept all Changes Shown and Accept all Changes in Document. This drop-down is disabled when the document is protected.

Reject - Button with Drop-Down. The button is the Reject and Move to Next command which rejects the current change and moves to the next proposed change. The drop-down contains the commands: Reject and Move to Next, Reject Change, reject all Changes Shown and Reject all Changes in Document. This drop-down is disabled when the document is protected.

Previous - Moves to the previous revision in the active document. This is a bit confusing as it does exactly the same as the Previous in the comments group.

Next - Moves to the next revision in the active document.

Copyright Virtual University of Pakistan

Module 213

213. Word Processing: MS Word (Compare and Protect Groups in Review Ribbon)

The Compare group is shown in the Figure 196.



Figure 196: Compare group

Compare - Drop-Down. The drop-down contains the commands: Compare and Combine. The Compare command lets you compare two versions of the same document and displays the "Compare Documents" dialog box. The Combine command lets you combine revisions from multiple authors and displays the "Combine Documents" dialog box. This drop-down is disabled when the document is protected.

The protect group is shown in the Figure 197.

Block Authors - Drop-Down. Prevent others from making changes to the selected text.

Restrict Editing - Toggles the display of the Restrict Editing task pane.
Protect Document - The drop-down contains the commands: Unrestricted Access, Restricted Access, Manage Credentials.



Figure 197: Protect group

Copyright Virtual University of Pakistan

Module 214

214. Word Processing: MS Word (View Ribbon)

The View ribbon contains Views group, Show Group, Zoom Group, and Windows group. The Views group contains all the commands relating to the different ways you can view your documents as shown in the Figure 198.



Figure 198: Views group

Read Mode – It maximizes the Word window on the screen and removes all toolbars etc to allow easy reading.

Print Layout - Displays the document as it would appear if printed and is the default view.

Web Layout - Displays the document as it would appear as a web page.

Outline - Displays the document as an outline is displays the Outlining contextual tab.

Draft - Displays the document in draft mode allowing for quick editing. When using this view certain aspects of the document are not visible, for example any headers or footers.

The show group is shown in the Figure 199.

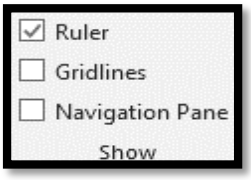


Figure 199: Show group

Ruler - Toggles the display of the ruler(s).

Gridlines - Toggles the display of gridlines.

Navigation Pane - Toggles the display of the Navigation task pane. This is a combination of the Find, Document Map and Thumbnails.

The Zoom group is shown in the Figure 200.

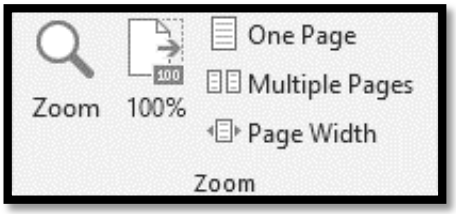


Figure 200: Zoom group

Copyright Virtual University of Pakistan

Zoom - Displays the "Zoom" dialog box. This can also be accessed from the status bar by clicking on the view percentage.

100% - Adjusts the zoom to 100% of its normal size. Lets you quickly return to 100%.

One Page - Adjust the zoom so an entire page fits in the application window.

Two Pages - Adjust the zoom so two entire pages fit in the application window.

Page Width - Adjust the zoom so the width of the page is the same as the width of the application window.

The Windows group has been shown in the Figure 201. Every document you open in Word can be thought of as a window. It is possible to open multiple windows of the same document.

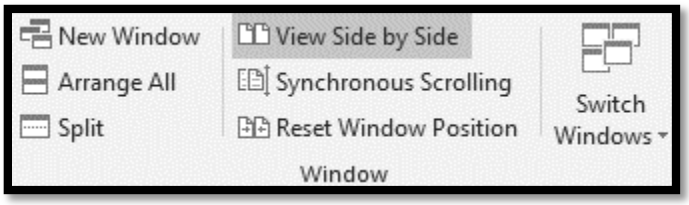


Figure 201: Windows group

New Window - Lets you create a new window of the active document. **Arrange All** - Tile all the open windows side by side on the screen. This will also maximize the application / document to a full screen.

Split - Splits the current window into two parts.

View Side by Side - Displays two documents side by side so they can be easily compared. If you have more than two documents open the "Compare Side by Side" dialog box is displayed so you can choose which document to display next to the active document, allowing you to scroll multiple windows at the same time.

Synchronous Scrolling - Toggles the synchronize scrolling of the two documents that are displayed side by side. This is only enabled when you are viewing two documents side by side.

Reset Window Position - Resets the windows positions so they take up the same amount of space on the screen when two documents are displayed side by side. This is only enabled when you are viewing two documents side by side.

Switch Windows - Drop-Down. Lets you switch between all the currently active documents. This displays all the window/documents that are currently open in the particular session.

ONLY MCQS

Copyright Virtual University of Pakistan

Module 215

215. Presentations: MS-PowerPoint (Introduction)

Microsoft offers another powerful tool known as MS-PowerPoint. This is used to develop presentations. Microsoft PowerPoint is an easy program to use and a powerful tool for giving a presentation. Whether your presentation needs a visual kick, tools for collaboration, easy access or the ability to share information beyond the initial meeting, PowerPoint is a good option. It can even help reduce speaking anxiety by drawing eyes away from the speaker and towards a screen. Just do not expect this technology to substitute for sound and dynamic speaking skills. All ribbons of MS Powerpoint are shown in the Figure 202.

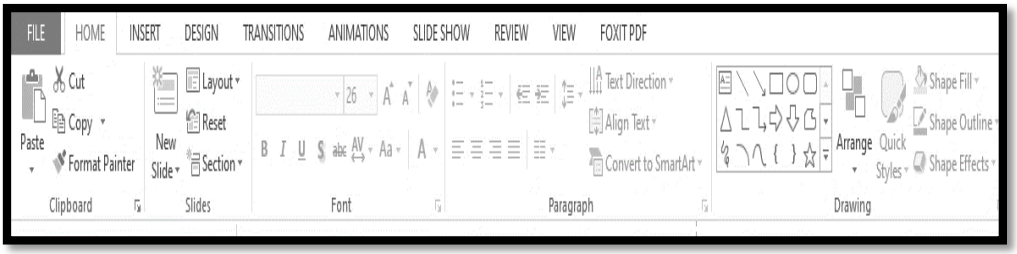


Figure 202: MS: Powerpoint

Some of the functions in different groups in Powerpoint are exactly same as per the functions in MS: Word. We will only cover those groups and functions which are new in the MS: Powerpoint as compared to the MS: Word. These groups and their functionalities will be discussed in the next modules.

Copyright Virtual University of Pakistan

Module 216

216. Presentations: MS-PowerPoint (Slides Group on Home Ribbon)

The Slides group is shown in the Figure 203.



Figure 203: Slides group in home ribbon

New Slide - Button with Drop-Down. The button inserts a new blank Title and Content Slide. The drop-down contains the slides: Title Slide, Title and Content, Duplicate Selected Slides, Slides from Outline and Reuse Slides.

Layout - Drop-Down. The drop-down contains the commands: Title Slide and Title and Content.

Reset - Reset the position, size and formatting of the slide placeholders to their default settings.

Delete - Removes the slides currently selected.

Module 217

217. Presentations: MS-PowerPoint (Design Ribbon)

The design ribbon is the ribbon that helps you to design your slides as per your requirement. The first group is called the: "Themes" as shown in the Figure 204. This helps to choose the required theme of your presentation. You can try different themes to see the difference.

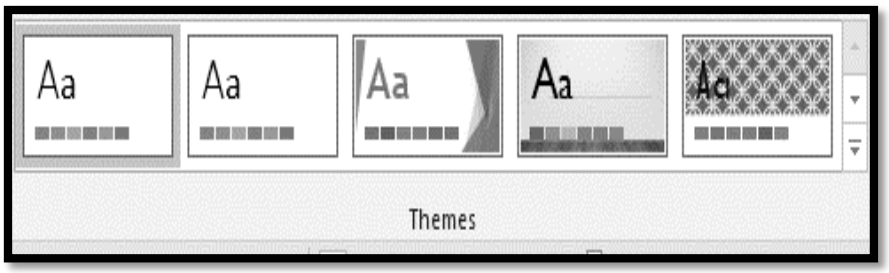


Figure 204: Themes group

Furhermore, you can select slide size in the next group and format background according to your requirements.

Copyright Virtual University of Pakistan

Module 218

218. Presentations: MS-PowerPoint (Transition Ribbon)

If you've ever seen a PowerPoint presentation that had special effects between each slide, you've seen **slide transitions**. A transition can be as simple as fading to the next slide or as flashy as an eye-catching effect. PowerPoint makes it easy to apply transitions to some or all of your slides, giving your presentation a polished, professional look. The transition ribbon has "Transition to this slide" group as shown in the Figure 205.

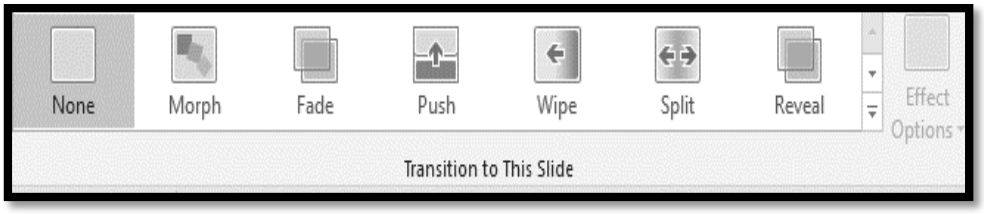


Figure 205: Transition group

Once you are done with one typical transition style, the next thing is to configure other options available in the Figure 206.

Sound – You can add any sound file over here which you need to play in this transition
Duration – you can set the duration of the transition.
Apply to All - Set the transition between all the slides in the presentation to be the same as this slide.
On Mouse Click - Lets you wait until there is a mouse click before moving on to the next slide.

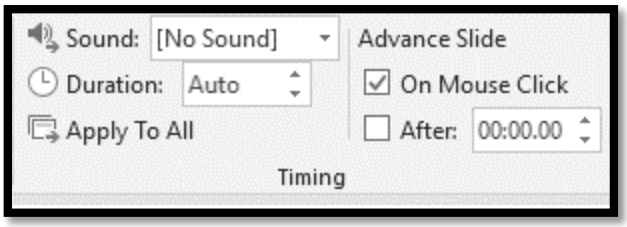


Figure 206: Transition timing group

Copyright Virtual University of Pakistan

Module 219

219. Presentations: MS-PowerPoint (Animation Ribbon)

Transition discussed in the last module were the animations added in between slides. Whereas, If you want to animate text, an image, shape, graph or chart within the slide, this is the type of animation you select.

There are variety of animations available as shown in the Figure 207.

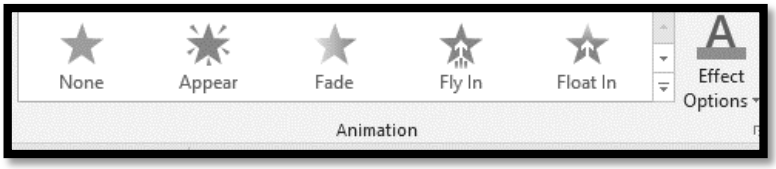


Figure 207: Animations group

Copyright Virtual University of Pakistan

Module 220

220. Spreadsheets: MS Excel (Introduction)

What is a Spreadsheet?

Suppose you would like to maintain personal finance register also known as check register that will keep track of the expenditures that you have been making during the current semester. Another scenario can be the budget management of a small house. You may be interested in question such as how much money has been spent on electricity bills and telephone bills. How much saving has been made? You may imagine a book that keeps record of all transaction/ expenditures and income. Such a book can be called a manual or paper spreadsheet. A sample is shown in the Figure 208. In this case you may have to save data that can be numerical or alphanumeric (involving letters or numbers).

Spreadsheet is a tool that is used to organize data, such as a check register. Spreadsheets have been used for many, many years in business to keep track of expenses and other calculations.

Electronic Spreadsheet: Microsoft Excel is an example of spreadsheet application program that can be used for storing, organizing and manipulating data. The key benefit to using a spreadsheet program is that you can make changes easily, including correcting spelling or values, adding, deleting and formatting. It consists of a grid made from columns and rows similar to what you have seen in your Mathematics notebooks in school days. This grid environment makes number manipulation very easy.

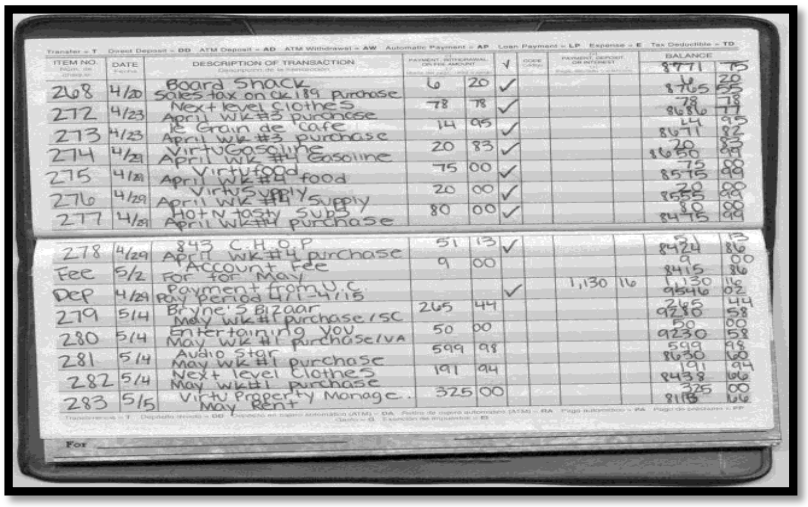


Figure 208: Spread Sheet Example

Please refer to Figure 209 to observe the layout of MS Excel. On the top you will find a ribbon similar to MS Word and PowerPoint that we will be discussing shortly. The different thing you will notice is the rectangular boxes on the screen divided with the help of rows and columns

Rows are identified by Numbers as shown in the leftmost columns whereas Columns are identified by Alphabets as shown in the topmost row.

If you want to select a particular cell, you will have to identify its address. The selected cell in the Figure 209. For. e.g. is A1, A being the first column and 1 being the first row

Copyright Virtual University of Pakistan

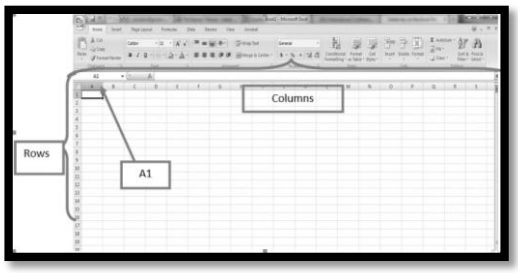


Figure 209: Excel Layout

Cell Formatting

The cell formatting is show in the Figure 2010. You can adjust the row height by clicking "Auto fit row height". This will adjust the height of all the cells in a row according to the font size of the data you have inserted.

Charts

Charts allow you to present information contained in the worksheet in a graphic format. Excel offers many types of charts including: Column, Line, Pie, Bar, Area, Scatter and more. To view the charts available click the Insert Tab on the Ribbon.

Creating a Chart:

- ✓ Select the cells that contain the data you want to use in the chart. In our example we can select the shipping cost column to be used as chart data.

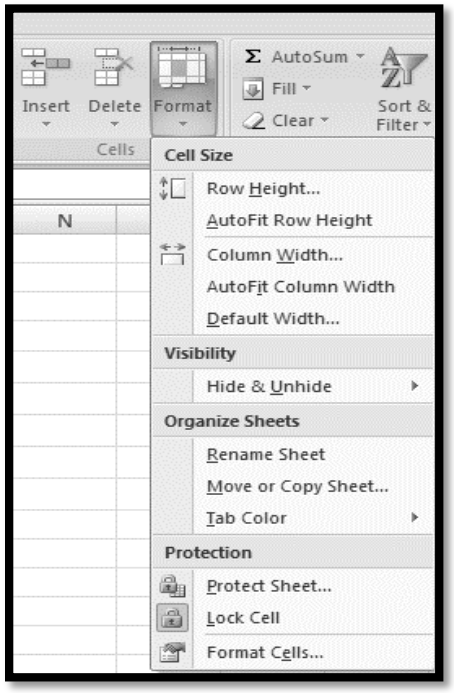


Figure 210: Cell formatting

Copyright Virtual University of Pakistan

- ✓ Click the Insert tab on the Ribbon
- ✓ Click the type of Chart you want to create. There are different chart types as show in the Figure 211.

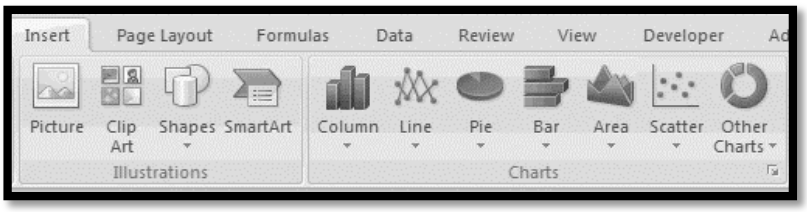


Figure 211: Charts group

- ✓ Click the Column chart button and then select the 2d chart
- ✓ A bar chart will appear showing the shipping cost as chart as shown in Figure 212.
- ✓ You can observe the costs at the Y-axis and as there are four items so you can see 4 bars at X-axis
- ✓ This chart has four components and you can right clock on them separately to see the advance options available.



Figure 212: Chart example in Excel

Module 221

221. Spreadsheets: MS Excel (Functions)

A formula is a set of mathematical instructions that can be used in Excel to perform calculations. Formals are started in the formula box starting with an = sign. A sample formula is shown in Figure 213.

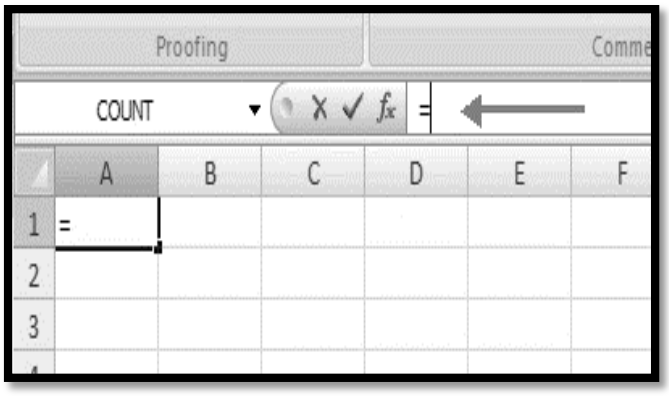


Figure 213: Formula Example

Copyright Virtual University of Pakistan

The formula ribbon is shown in the Figure 214. This ribbon gives you different options for applying formulas. The first section represents Function library. Function library contains different formulas from various categories. Figure 214 gives you an overview of the Formula ribbon.

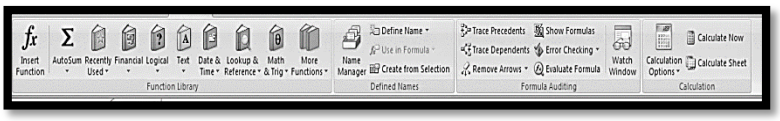


Figure 214: Formula ribbon

Creating Basic Formula

To create a basic formula in Excel we need the data on which to apply the formula. Data is represented in Figure 215.

Product	Price	Shipping Cost
Toothpaste	25	10
Shampoo	120	20
Soap	30	10
MouthWash	50	15

Figure 215: Data for Sum Formula

To create a basic formula, you need to do the following:

Select the cell for the formula

Type "=" (the equal sign) and the formula

The formula is of sum which would sum the values.

":" is used to specify the range of the cell on which you want to apply the formula

Click Enter

Creating a basic formula is shown in the Figure 216.

Product	Price	Shipping Cost	Total Price
Toothpaste	25	10	=sum(F5:G5)
Shampoo	120	20	
Soap	30	10	
MouthWash	50	15	

Figure 216: Sum Formula

Copyright Virtual University of Pakistan

When you click enter you would get the result as shown in figure 217 the label 1 represents the formula applied to get the desired result and label 2 represents the result after applying the formula.

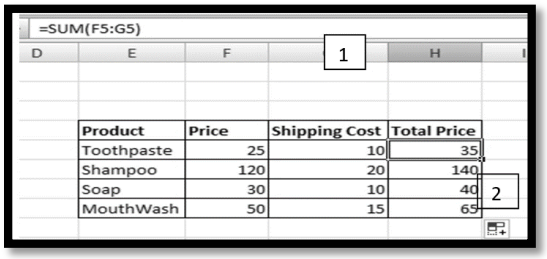


Figure 217: Result of Sum Formula

Calculate with Functions

A function is a built in formula in Excel. A function has a name and arguments (the mathematical function) in parentheses.

- ✓ Sum: Adds all cells in the argument
- ✓ Average: Calculates the average of the cells in the argument
- ✓ Min: Finds the minimum value
- ✓ Max: Finds the maximum value
- ✓ Count: Finds the number of cells that contain a numerical value within a range of the argument

To calculate a function:

- ✓ Click the cell where you want the function applied
- ✓ Click the Insert Function button
- ✓ Choose the function
- ✓ Click OK

The process of calculating through a function is shown in figure 218.

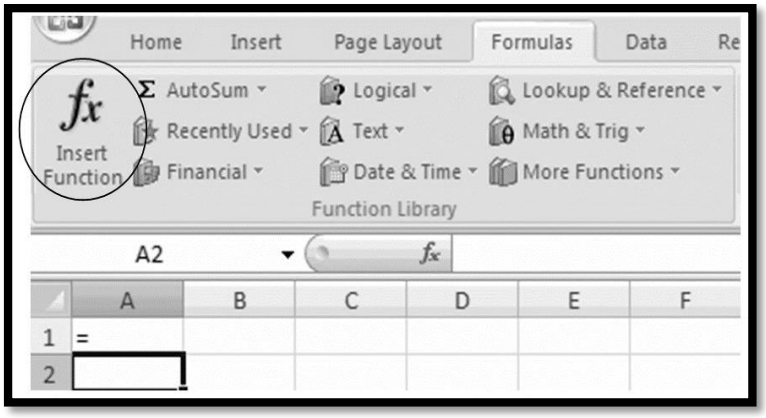


Figure 218: Function Computation

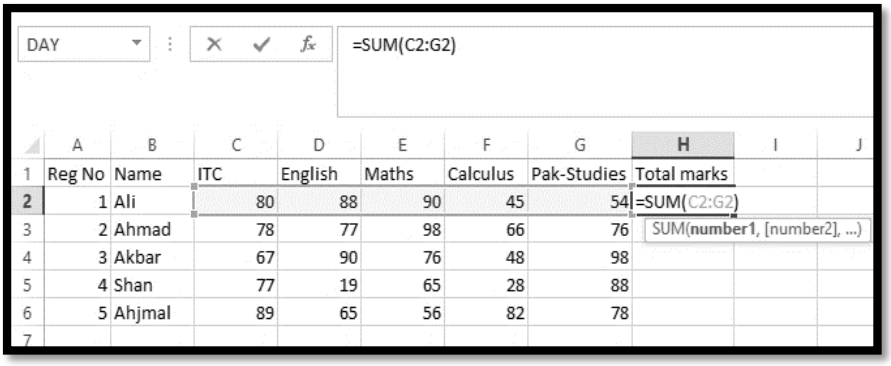
Copyright Virtual University of Pakistan

Module 222

222. Spreadsheets: MS Excel (Application Scenarios-I)

After learning the formulas in details in the last module. Lets apply those in the real scenario.

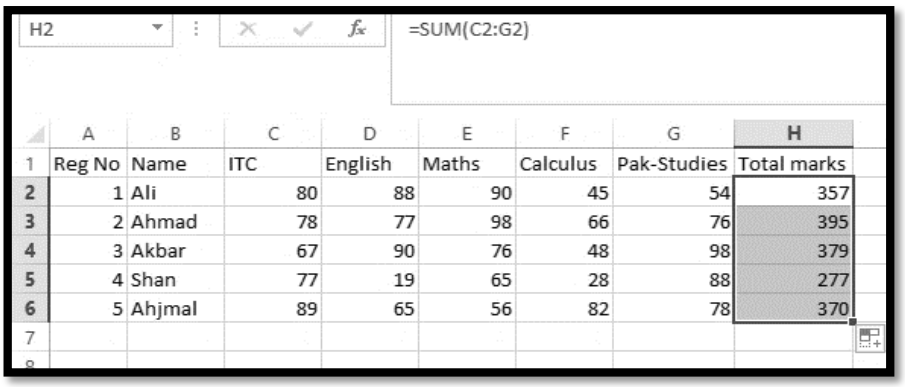
Consider the Figure 219 which contains the data of five students and their marks in five subjects. We want to add the total marks of each student. The formula has been written for your reference.



	A	B	C	D	E	F	G	H	I	J
1	Reg No	Name	ITC	English	Maths	Calculus	Pak-Studies	Total marks		
2	1	Ali	80	88	90	45	54	=SUM(C2:G2)		
3	2	Ahmad	78	77	98	66	76			
4	3	Akbar	67	90	76	48	98			
5	4	Shan	77	19	65	28	88			
6	5	Ahjmal	89	65	56	82	78			
7										

Figure 219: Sum of marks

When this formula is applied and copied in the following cells, the result can be seen in the Figure 220.



	A	B	C	D	E	F	G	H
1	Reg No	Name	ITC	English	Maths	Calculus	Pak-Studies	Total marks
2	1	Ali	80	88	90	45	54	357
3	2	Ahmad	78	77	98	66	76	395
4	3	Akbar	67	90	76	48	98	379
5	4	Shan	77	19	65	28	88	277
6	5	Ahjmal	89	65	56	82	78	370
7								

Figure 220: Result of sum formula

Copyright Virtual University of Pakistan

Module 223

223. Spreadsheets: MS Excel (Application Scenarios-II)

Let's apply the learned formulas in another application scenario. Remember, in the module 220, we mentioned a scenario that if you want to manage your personal expenses on a register, its very difficult to perform different queries. In this module, we will consider this scenario and will perform number of queries on this example.

The Figure 221 contains the personal expenses details of the year 2020 month wise in different heads.

	A	B	C	D	E	F	G	H
1	Perosnal Expenses for the Year 2020							
2	Months	Electricity Bill	Sui-gas Bill	Grocery	Tuition Fees	Broadband Bill	Car maintenance	Total Expenses
3	January	5000	528	15821	50000	5412	15241	
4	February	7000	635	17852	50000	5412	12345	
5	March	8457	425	18241	50000	5412	14235	
6	April	5697	525	14250	50000	5412	15152	
7	May	8574	852	19245	56000	5412	14162	
8	June	9585	741	13258	56000	5412	13252	
9	July	7568	247	17412	56000	5412	18654	
10	August	10258	259	28935	56000	5412	17420	
11	September	7532	635	25878	56000	5412	14582	
12	October	3218	526	24125	60000	5412	16254	
13	November	7521	325	10741	60000	5412	18521	
14	December	2358	741	12365	60000	5412	13254	
15								
16	Total Expense							
17								

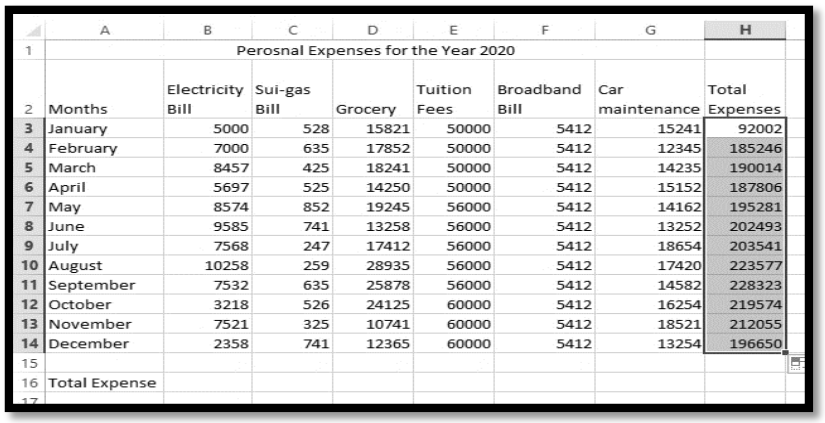
Figure 221: Personal expense sheet

Now our first query is to see the total expense for the January month. The applied formula can be seen in the Figure 222 and its result can be seen in the Figure 223.

	A	B	C	D	E	F	G	H
1	Perosnal Expenses for the Year 2020							
2	Months	Electricity Bill	Sui-gas Bill	Grocery	Tuition Fees	Broadband Bill	Car maintenance	Total Expenses
3	January	5000	528	15821	50000	5412	15241	=sum(b2:g3)
4	February	7000	635	17852	50000	5412	12345	
5	March	8457	425	18241	50000	5412	14235	
6	April	5697	525	14250	50000	5412	15152	
7	May	8574	852	19245	56000	5412	14162	
8	June	9585	741	13258	56000	5412	13252	
9	July	7568	247	17412	56000	5412	18654	
10	August	10258	259	28935	56000	5412	17420	
11	September	7532	635	25878	56000	5412	14582	
12	October	3218	526	24125	60000	5412	16254	
13	November	7521	325	10741	60000	5412	18521	
14	December	2358	741	12365	60000	5412	13254	
15								
16	Total Expense							
17								

Figure 222: Finding total expense for the January month

Copyright Virtual University of Pakistan

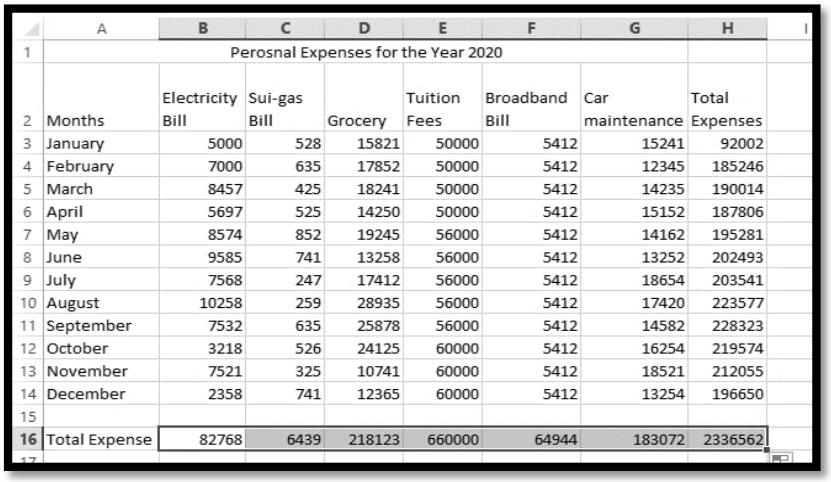


	A	B	C	D	E	F	G	H
1	Perosnal Expenses for the Year 2020							
2	Months	Electricity Bill	Sui-gas Bill	Grocery	Tuition Fees	Broadband Bill	Car maintenance	Total Expenses
3	January	5000	528	15821	50000	5412	15241	92002
4	February	7000	635	17852	50000	5412	12345	185246
5	March	8457	425	18241	50000	5412	14235	190014
6	April	5697	525	14250	50000	5412	15152	187806
7	May	8574	852	19245	56000	5412	14162	195281
8	June	9585	741	13258	56000	5412	13252	202493
9	July	7568	247	17412	56000	5412	18654	203541
10	August	10258	259	28935	56000	5412	17420	223577
11	September	7532	635	25878	56000	5412	14582	228323
12	October	3218	526	24125	60000	5412	16254	219574
13	November	7521	325	10741	60000	5412	18521	212055
14	December	2358	741	12365	60000	5412	13254	196650
15								
16	Total Expense							

Figure 223: Result of query from Figure 222

Now if we are interested to see what the total expense of the Electricity Bill is, we can apply the following formula and can copy it in the next cell to see the result as shown in the Figure 224.

=sum(b3:b14)



	A	B	C	D	E	F	G	H
1	Perosnal Expenses for the Year 2020							
2	Months	Electricity Bill	Sui-gas Bill	Grocery	Tuition Fees	Broadband Bill	Car maintenance	Total Expenses
3	January	5000	528	15821	50000	5412	15241	92002
4	February	7000	635	17852	50000	5412	12345	185246
5	March	8457	425	18241	50000	5412	14235	190014
6	April	5697	525	14250	50000	5412	15152	187806
7	May	8574	852	19245	56000	5412	14162	195281
8	June	9585	741	13258	56000	5412	13252	202493
9	July	7568	247	17412	56000	5412	18654	203541
10	August	10258	259	28935	56000	5412	17420	223577
11	September	7532	635	25878	56000	5412	14582	228323
12	October	3218	526	24125	60000	5412	16254	219574
13	November	7521	325	10741	60000	5412	18521	212055
14	December	2358	741	12365	60000	5412	13254	196650
15								
16	Total Expense	82768	6439	218123	660000	64944	183072	2336562

Figure 224: Total bill Expense-head wise

Similarly, you can try the following formulas as well:

=average(b3:b14)

=max(b3:b14)

=min(b3:b14)

The total expense for the year 2020 can be seen after applying the following formula:

=SUM(B16:H16)

Copyright Virtual University of Pakistan

Module 224

224. Spreadsheets: MS Excel (Sorting and Filter)

Sorting and Filtering allow you to manipulate data in a worksheet based on given set of criteria. You can sort the data in ascending and descending order. There are two types of sorts in Microsoft Excel i.e. Basic Sorts and Custom Sorts.

Basic Sort

To execute a basic descending or ascending sort based on one column perform the following steps:

- Highlight the cells that will be sorted
- Click the Sort & Filter button on the Home tab

Click the Sort Ascending (A-Z) button or Sort Descending (Z-A) button.

These steps are shown in the Figure 225.



Figure 225: Sort and Filter

Custom Sort

To sort on the basis of more than one column:

- ✓ Click the Sort & Filter button on the Home tab
- ✓ Choose which column you want to sort by first
- ✓ Click Add Level
- ✓ Choose the next column you want to sort
- ✓ Click OK

This process is represented in the Figure 226.

Copyright Virtual University of Pakistan

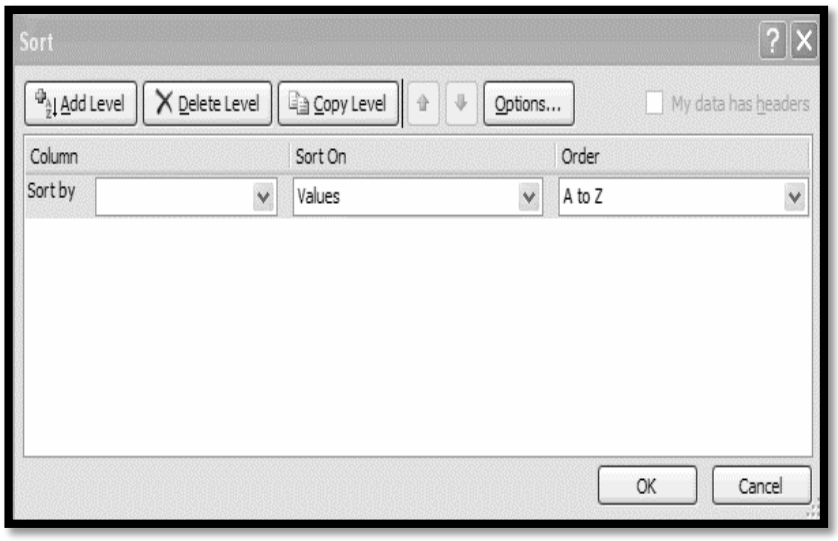


Figure 226: Custom Sort

Copyright Virtual University of Pakistan

Module 225

225. Database: MS Access (Introduction)

Database is collections of files which are also referred as tables. Information is organized in these tables in the form of fields and records for easy access and management purpose.

SQL stands for structured query language. In order to access and retrieve data from the database we use SQL. This is the most commonly used syntax for querying a relational database.

It is important to understand the difference between Database and DBMS. Let's understand this difference with an analogy of office Cleric. Office cleric organize and stores different files in the filing cabinet whenever he needs to access information from the file or add new information in the file he goes to the filling cabinet and retrieve the file which he has already stored in the filling cabinet. After updating the information in the file he stores the file back to the filling cabinet. In the above example Database is the filling cabinet where the information is stored in the form of files. And the analogy of DBMS is the office cleric, whose job is to organize and retrieve files.

You should be quite familiar with the ribbon as presented in the MS Word and MS Excel. The similar ribbon is available in MS Access. The same ribbon is also present in Access with few modifications. There is addition of three new tabs i.e. Create, External data and Database Tools as shown in the Figure 227.

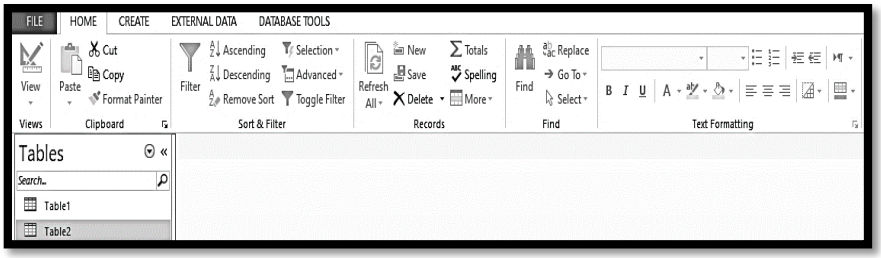


Figure 227: MS Access ribbon

To create a new database

- ✓ Click on Blank database option as shown in Figure 228
- ✓ Type Database name and browse to location where you save the new database.

Click Create button to create database. (Figure 229).

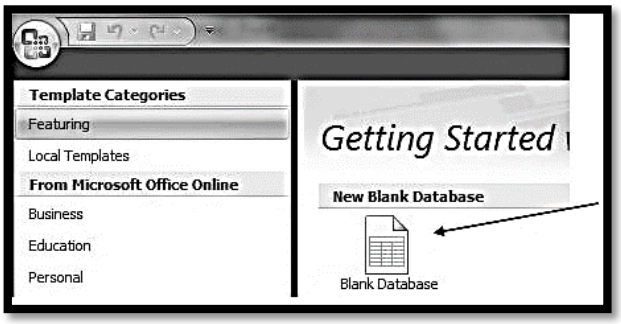


Figure 228: Create Blank Database

Copyright Virtual University of Pakistan

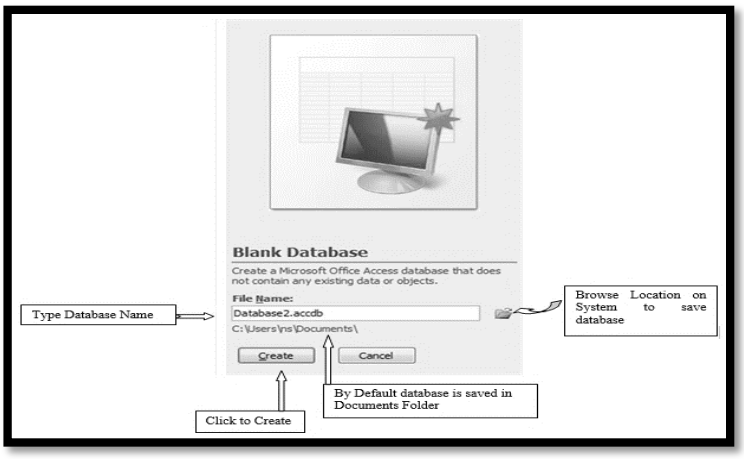


Figure 229: Naming and saving database

Navigation Panel

When your database is created new window screen appear and by default table is added. On the left side of the window is navigation panel (Figure 230) here you will see all your saved tables, forms, queries and reports. On the Right of the navigation pane is working or view panel. You can view table data, query output, generate reports and form here.

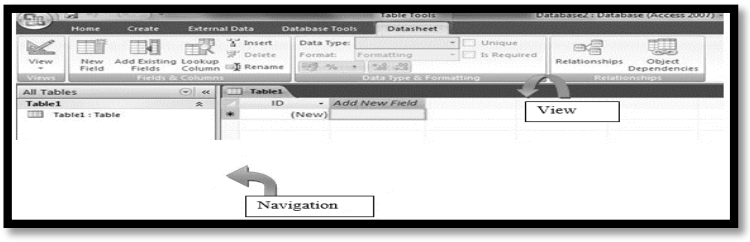


Figure 230: Blank Database

External data

External data tab is used to import, export, collect and share data between different databases. You can also export and import data from excel document as shown in the Figure 231.

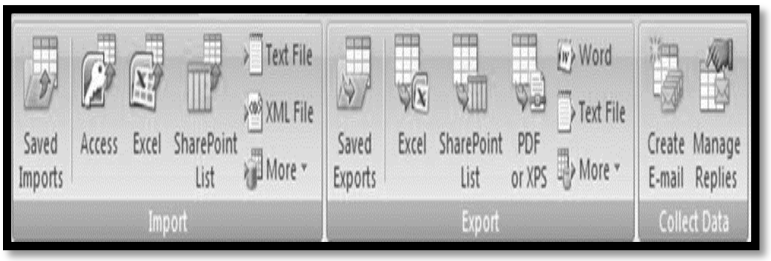


Figure 231: External data Ribbon

Copyright Virtual University of Pakistan

Module 226

226. Database: MS Access (Creating and Managing Tables)

Create Tab in the Ribbon helps you to provide quick and easy way to create tables, forms, report and queries as shown in the Figure 232.

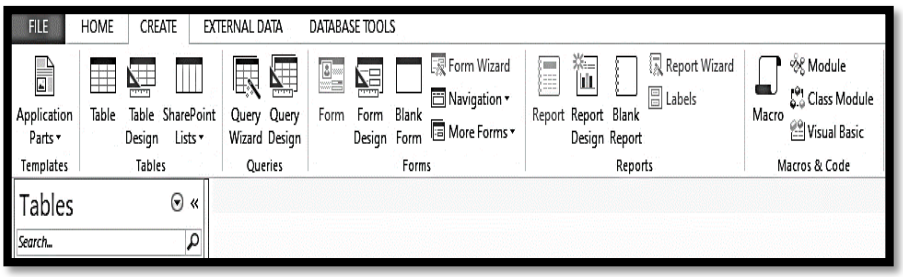


Figure 232: Create tab

Tables Group

Tables in Access are used to store data. It is a set of columns and rows. Each column in table is referred to as a field. Each value in the field signifies a single type of data. Numbers of Record are number of rows in a Table hence each row of a table is referred to as a single record.

Exercise: Create Courses Table

1. Click on Create tab (Figure 232).
2. Click Table in Tables group. A blank table with default Name "Table 1" is created.
3. Click on Design view. A popup dialogue box will appear with a text box with "Table 1".
4. Rename the table name to "courses" and click OK.
5. In Design view click on second row in field name column.
6. Add a new field for course title as "ctitle". Set its data type to "Text".
7. Add another field for section as "csection". Set its data type to number.
8. Add a new field name for course Instructor as "cinstructor". Set its data type to "Text".
9. To return to datasheet view click on view button and select.
10. A Popup dialogue box will appear to save the changes, click yes to continue.
11. In datasheet view your newly added fields are visible.
12. Insert 10 courses data.
13. Close table.

Copyright Virtual University of Pakistan

Module 227

227. Database: MS Access (Creating Forms)

Forms option provides you the ability to enter new record and navigate through existing records easily. You can arrange fields in different layouts add Icon and also apply designs as can be seen in the Figure 232.

Creating Form for Student Table

1. Select student table. Double click to open.
2. Click Create tab.
3. From the Forms group Select Form (Figure 232).
4. A Form will be created with Table Fields as Labels followed by Text Boxes (Figure 233)
5. Use Crtl+ S key to Save or right click on form tab "student" and select save.
6. In the popup dialogue box enter form name "studentForm" and click OK.
7. Your newly created form will show in Navigation Pane.(Figure 234)

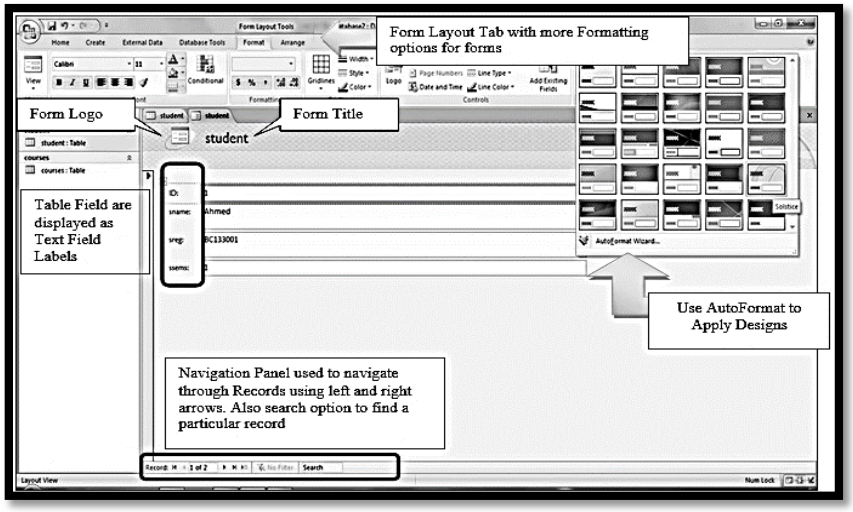


Figure 233: Student Table Form



Figure 2334: Saved Form "studentForm"

Copyright Virtual University of Pakistan

Module 228

228. Database: MS Access (Creating Reports)

Reports are organized and summarized form of your data so you can print or view it onscreen. Reports are often use when you want to analyze your data or present it to others. It is available as an independent group of Figure 232.

Creating Report for Student Table

1. Select student table. Double click to open.
2. Click Create tab.
3. From the Reports group select report (Figure 235).
4. A Report will be generated as shown in Figure 236.
5. Use Crtl+ S key to Save or right click on form tab "student" and select save.
6. In the popup dialogue box enter form name "srudentReport" and click OK.
7. In the Navigation Pane you can see your newly created form by the name you have saved.(Figure 236)
8. Rename Report Title to "studentReport".

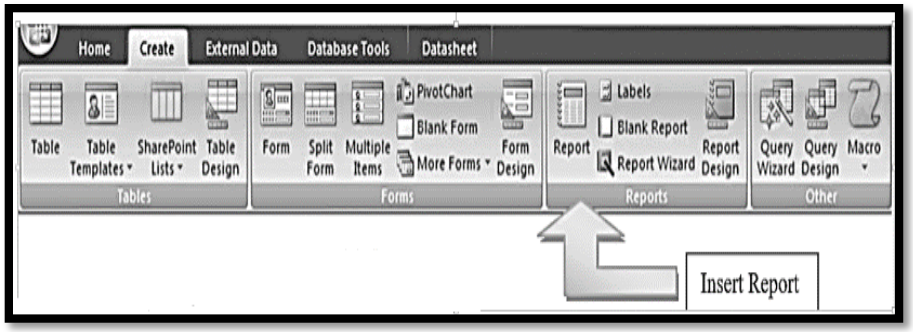


Figure 234: Insert Report

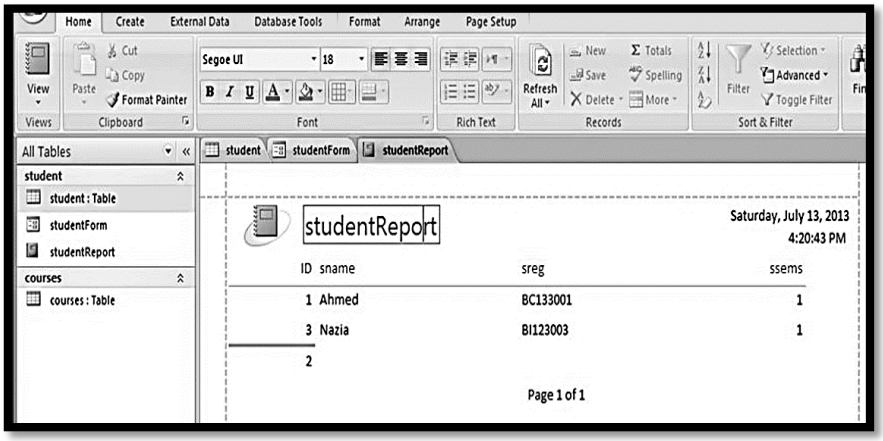


Figure 235: Student Table Report

Copyright Virtual University of Pakistan

Module 229

229. Database: MS Access (Query Wizard)

Queries are to retrieve or fetch specific data from your database. The query wizard group has been shown in the Figure 236.

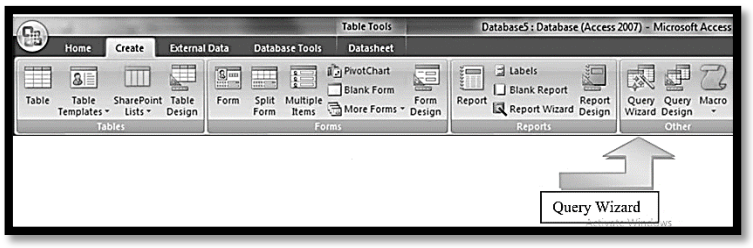


Figure 236: Query Group: Query Wizard

Query Types

There are four basic types of queries used to manipulate the databases. These are briefly described in the Table 1 along with generic syntax.

Table 1: Basic Queries

Query	Description	Generic SQL Syntax
Select	Used to retrieve selected data one or more field can be selected by using comma as seperator or use * to select all table fields, followed by table name from where the data is to be retrieved. You can also add WHERE clause to set criteria for selection.	SELECT field1,field2…,fieldN FROM table_name1,table_name2; OR SELECT * FROM table_name, table_name2; OR SELECT *FROM table_name, table_name2 WHERE some criteria;
Insert	Used to add new records.	INSERT INTO table_name (field1, field2,...fieldN) VALUES
Update	Used to update existing records in database. Given some criteria in WHERE clause.	UPDATE table_name SET field1=newvalue1, field2=newvalue2
Delete	Used to remove data from database.	DELETE FROM table_name WHERE some criteria;

Query Student Table using Wizard

1. Click Create tab. From the Query group select Query Wizard (Figure 236).
2. From popup window select simple query and click OK.

Copyright Virtual University of Pakistan

3. Another window will appear. Select table from which you want to query results in our case it is the student table.
4. Select sname and ssems from the displayed fields as shown in Figure 237
5. When done click next and next again.
6. Before you click finish entering the name you want to give your query. For example, "StudentTBQuery1". Click finish.
7. Your query will be represented in a table form with two selected fields student name and semester.
8. Save query as you saved your report and form (Figure 238).

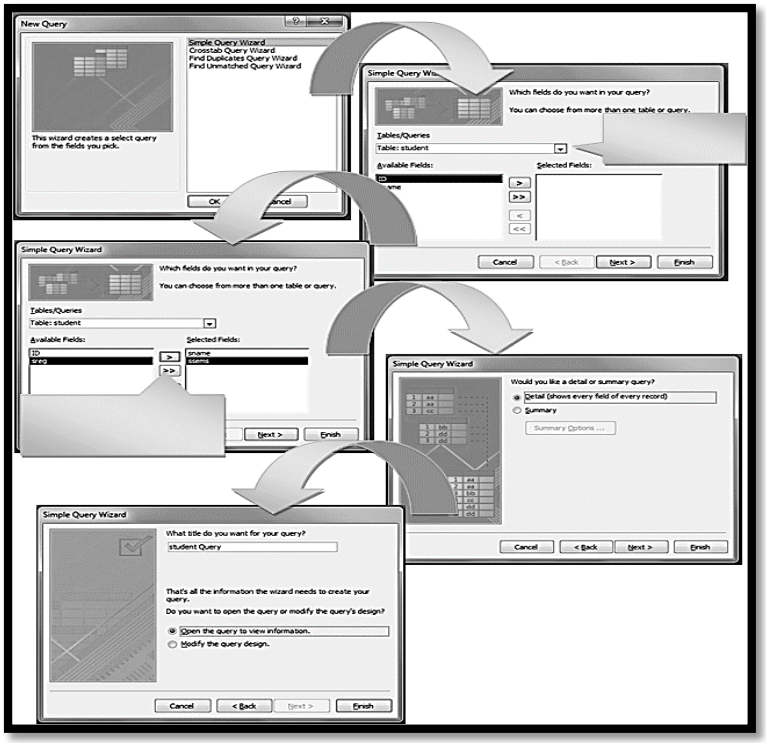


Figure 237: Creating Query using wizard

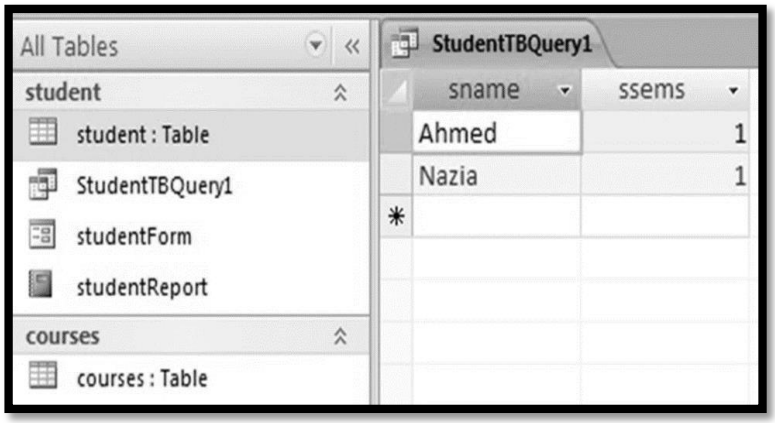


Figure 238: Query Result after saving

Copyright Virtual University of Pakistan

SQL stands for Structured Query Language. This is the most commonly used syntax for querying a relational database. Microsoft Access Provides SQL View feature to view the SQL Syntax of the Query generated. To view the SQL Syntax of your Query right click on query tab and Select SQL View as shown in the Figure 239.

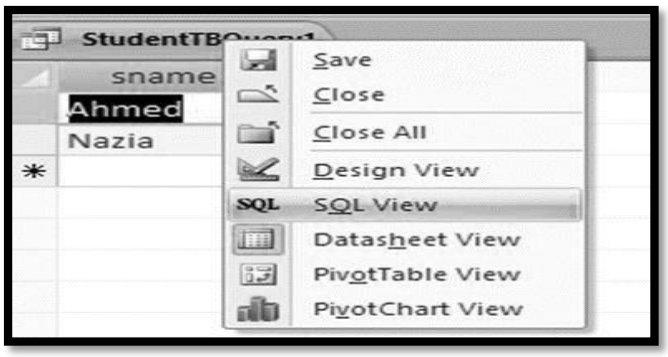


Figure 239: SQL View

Query Design View

Query design view (see Figure 240) can be used to design or create a query with addition of further parameters such as Where Clause, Sort or Order By clause and other useful features.

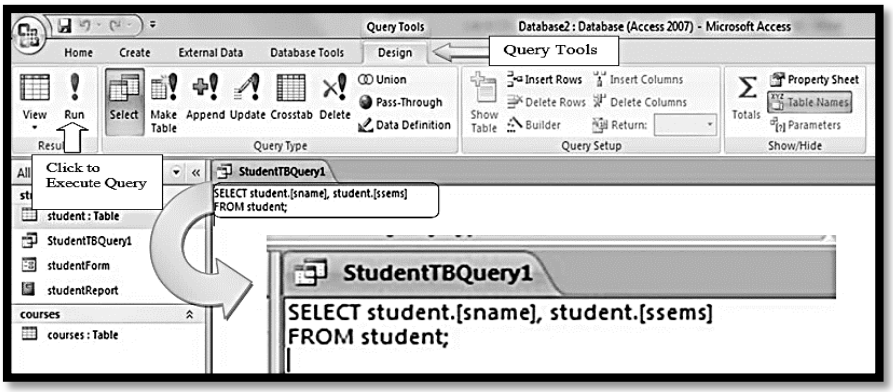


Figure 240: SQL View Query Syntax

Exercise: Query Student Table

1. Click Create tab. From the Query group select Query Design (Figure 241).
2. From popup window select table name.
3. You can add multiple tables to perform query as well for simplicity we will use one table, click Add and then close.
4. Table with is all fields will be shown in a small box.
5. From the options shown at the bottom. Click on text box and from drop down menu select field name “sname” it’s table ”student” in first column.(Figure 242)
6. Select field name “ssems” it’s table ”student” in second column.

Let’s say we want to view records of only those students who are in 3rd semester. Here our criteria is “3rd semester”

Copyright Virtual University of Pakistan

7. In the criteria box of the field ssems type 3.
8. Repeat step 5 to select remaining fields of the table.
9. When happy with the selection click Run on the top left corner of the Query Tools (Figure 240).



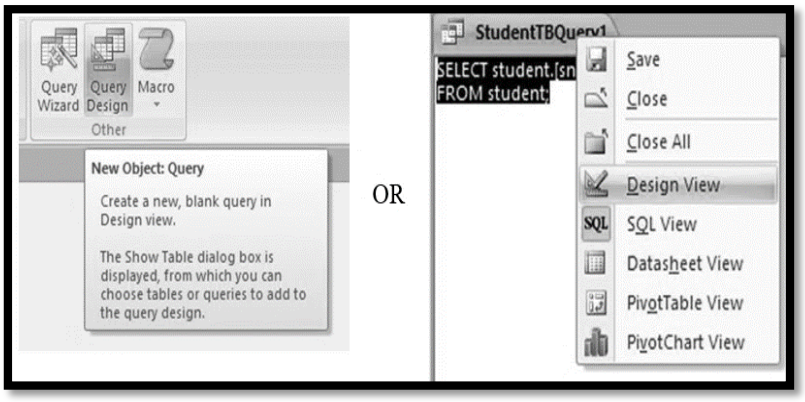


Figure 241: Query Design View

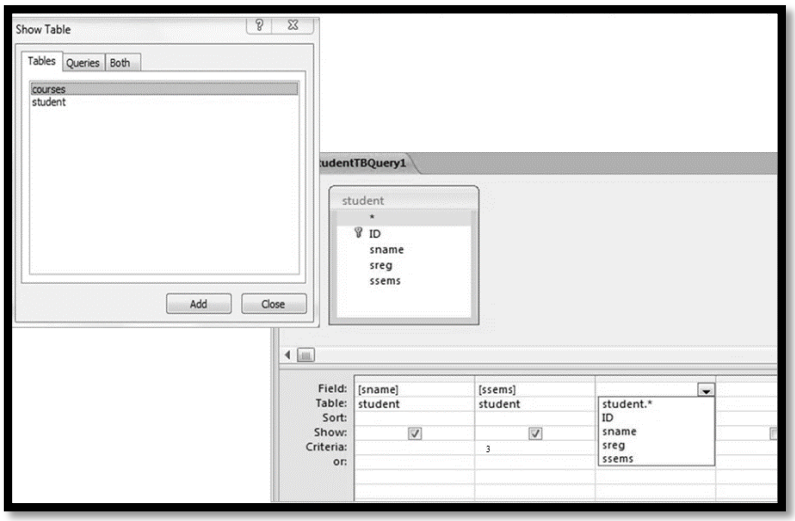


Figure 242: Create Query Using Design View

Copyright Virtual University of Pakistan

Module 230

230. Web page Development (Notepad Editor)

In this module, you will learn what is web page, how it can be developed using notepad editor.

What is a Webpage?

A web page is nothing more than a file, a HTML file to be exact. It's called HTML because web page documents have the file extension **.html** or **.htm.** A web browser displays a web page on a monitor or mobile device. We can use WebPages to display information to a large audience. WebPages provides an opportunity to market your business at a larger scale and let the world know your existence.

HyperText Markup Language is a markup language that web browsers use to interpret and compose text, images and other material into visual or audible web pages. Default characteristics for every item of HTML markup are defined in the browser. The purpose of a web browser is to read HTML documents and compose them into visible or audible web pages. Whenever you open a HTML document in a web browser, the web browser does not display the HTML tags, but uses the tags to interpret the content of the page.

HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. It can embed scripts written in languages such as JavaScript which affect the behavior of HTML web pages.

1.1 HTML Elements:

An HTML element has a starting and ending tag. Content is placed between start and end tag. Elements can have various attributes.

1. The text between tags <html> and </html> describes the start and end of web page.
2. The text between tags <body> and </body> represents the content of the page that is visible to the end user.
3. The text between tags <h1> and </h1> is presented as a first heading.
4. The text between tags and is presented of the page as one paragraph. Each tag has a stating tag and an ending tag. Some exceptions are there which will be discussed later.

An element can have multiple elements in it before end tag.

BLOCK elements always start after a new line and the content is displayed in the next line in browser.

Examples: <h1>, , <ul>, <table>(will be used in coming sections)

Inline elements are usually displayed on the browser without starting a new line. Examples: <b>, <td>, <a>, <img>

HTML Attributes

Elements have attributes that provides additional information about that element. Attributes are certain name that comes in between starting and ending tags.

Example:

<img src="VU logo.jpg" width="100" height="150">

Copyright Virtual University of Pakistan

img is element. src, width and height are attributes for image tag. src defines the location of the element. Width and height represent sizes of the image.

Using Notepad to develop HTML page

1. Open any text editor for example notepad
2. Write the code shown in the Figure 243.



Figure 243: HTML code in Notepad

3. Now save the document as html illustrated in the Figure 244.

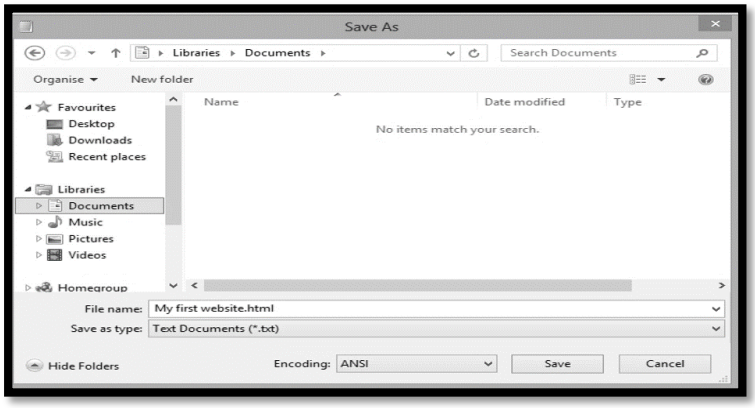


Figure 244: Saving as html page

Double click on that html file and open with any of installed browsers. The icon will appear as shown in the Figure 245.



Figure 245: First web page output

Copyright Virtual University of Pakistan

Module 231

231. Web page Development (Introduction to Dreamweaver)

Although the HTML markup of a web page can be written with any text editor such as Notepad as we have written in the last module. However, when we are making large webpages with many different tags, in that case, the code becomes complex. To cope with such situation, specialized HTML editors are used.

HTML editor is a software application for creating web pages, specialized HTML editors can offer convenience and added functionality. For example, many HTML editors work not only with HTML, but also with related technologies such as CSS (Cascading Style Sheets), XML(Extensible Markup Language) and JavaScript. Anyone who wants to make professional web pages, he has to use these technologies. Almost all pages that you see now a days on the Internet consist of (fully or partially) components that are made from CSS,XML and JavaScript.

In this module we will be introducing a toll known as Adobe Dreamweaver that makes it easier for web page developers to utilize the technologies like CSS and JavaScript. We will not be making complex web pages in this lab however; we will be going through a basic introduction of Adobe Dreamweaver. Although there are several other HTML editing tools available, we have chosen this tool because it is among the most commonly used tool for professional web development.

Adobe Dreamweaver is a web design and development application that provides a visual WYSIWYG editor (What You See Is What You Get) and a code editor with standard features such as syntax highlighting, code completion, and code collapsing as well as more sophisticated features such as real-time syntax checking and code introspection for generating code hints to assist the user in writing code.

Dreamweaver is a powerful application for developing websites and mobile applications. You can save time, work more efficiently, and create compelling designs with the complete set of tools. It is leading software for developing websites and mobile applications. Using its powerful HTML coding environment and standard-setting visual interface, you can create engaging and dynamic websites for a wide range of devices, whatever your level of design or development experience.

The common misconception with Dreamweaver is that Dreamweaver is designed to or is even capable of completely removing the text editors like Notepad for HTML and CSS coding from web design. This is like saying that a machine-made football can completely replace a man-made football A machine made football can make good quantity of footballs in a commercial production however, it cannot achieve the durability and perfection of a man made football. As you may know, man-made footballs from Sialkot are still being used in international tournament s even though we have the choice of machine-made footballs.

Dreamweaver, like the machine that makes football, is designed to make your life easier. You may never learn HTML or CSS, but without knowing them, you are limited to Dreamweaver's way of doing things. This is not altogether a bad thing: it is simply a slightly narrow perspective on a large field. Dreamweaver is not designed to completely remove the utility of HTML and CSS: it is meant to assist you with your HTML and CSS and to make some of the more mundane aspects of Web Publishing less terrible.

Dreamweaver interface has been shown in the Figure 246.

Copyright Virtual University of Pakistan

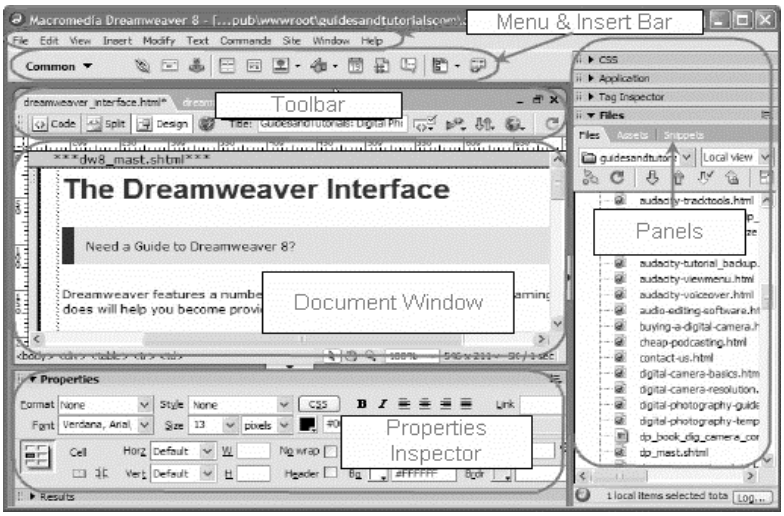


Figure 246: Dreamweaver interface

The Document toolbar: Contains buttons that provide options for different views of the Document window (such as Design view and Code view), various viewing options, and some common operations such as previewing in a browser.

The Menu and Insert bar: Across the top of the application window contains a workspace switcher, menus and other application controls. To display the Standard toolbar, select View

> Toolbars > Standard. The toolbar contains buttons for common operations from the File and Edit menus very similar to MS Word: New, Open, Browse in Bridge, Save, Save All, Print Code, Cut, Copy, Paste, Undo, and Redo.

The Document window: Displays the current document as you create and edit it. To switch to a document, click its tab. The Document Window is the main workspace where you insert and arrange the elements on your web page. Most of your work inserting text and images on your web page is done in the Document Window.

You can select any of the views. Description of the important views is given below.

Design view: A design environment for visual page layout, visual editing, and rapid application development. In this view, Dreamweaver displays a fully editable, visual representation of the document, similar to what you would see when viewing the page in a browser.

Code view: A hand-coding environment for writing and editing HTML, JavaScript,

Split Code view: A split version of Code view that lets you scroll to work on different sections of the document at the same time.

Code and Design view: Lets you see both Code view and Design view for the same document in a single window.

Live view: Similar to Design view, Live view displays a more realistic representation of what your document will look like in a browser, and lets you interact with the document exactly as you would in a browser. Live view is not editable. However, you can edit in Code view and refresh Live view to see your changes.

Copyright Virtual University of Pakistan

Live Code view: Only available when viewing a document in Live view. Live Code view displays the actual code that a browser uses to execute the page, and can dynamically change as you interact with the page in live view. Live Code view is not editable.

Properties Inspector as shown in the Figure 247 also called the Tag Inspector. The Property Inspector is used to display and edit attributes or properties of any element that is selected in the Document Window. Click the Property Inspector's Hide button to hide the panel temporarily.

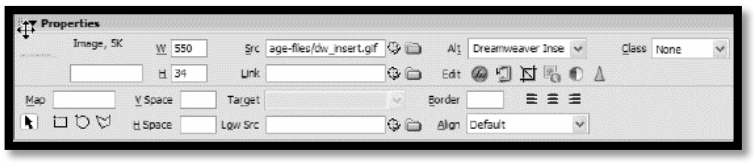


Figure 247: Properties panel

Copyright Virtual University of Pakistan

Module 232

232. Web page Development (Inserting Tables using Dreamweaver)

First save the document that you got after clicking new HTML document in the welcome screen as shown in the Figure 248.

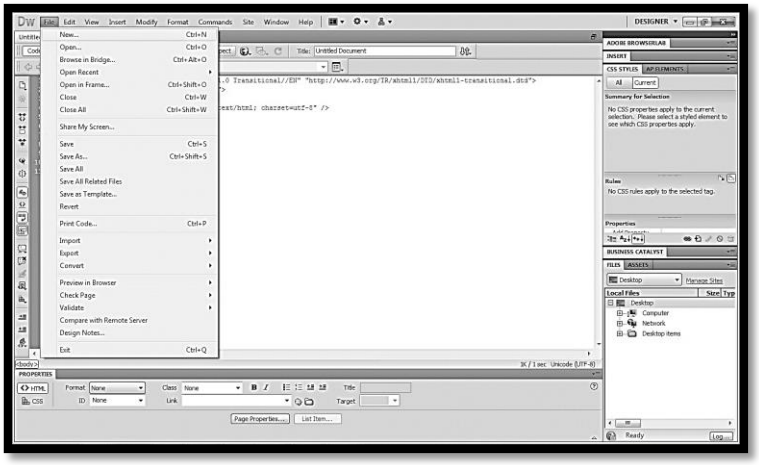


Figure 248: Add new page

Now add HTML page as shown in the Figure 249.

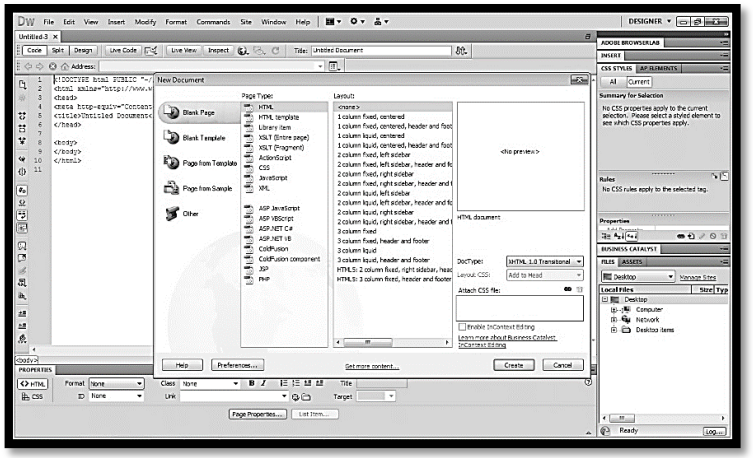


Figure 249: Adding HTML page

Copyright Virtual University of Pakistan

Now give a name to the page as shown in the Figure 250.

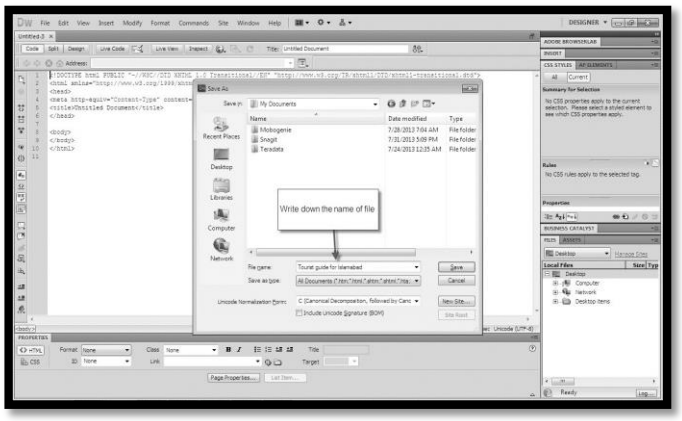


Figure 250: Naming the page

To insert table, just go to the Insert tab and add the table as shown in the Figure 251.



Figure 251: Insert Table

You can then specify number of rows and columns as shown in the Figure 252.

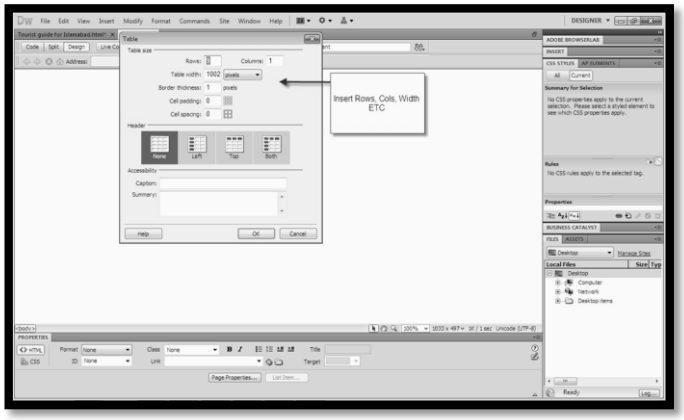


Figure 252: Rows and columns

You can then adjust the size of the rows as shown in the Figure 253.

Copyright Virtual University of Pakistan

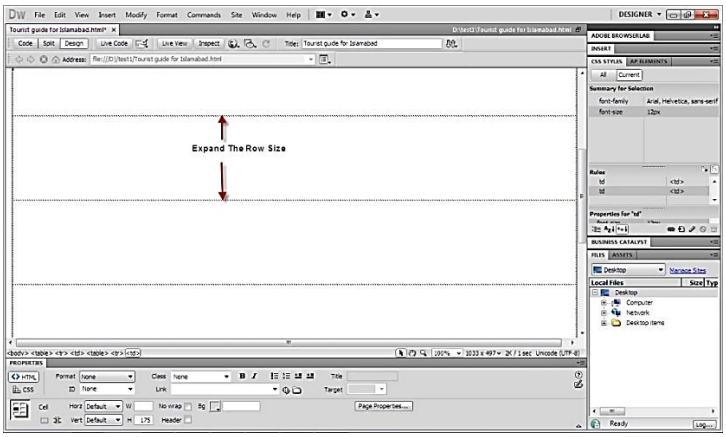


Figure 253: Adjusting row sizes

Copyright Virtual University of Pakistan

Module 233

233. Web page Development (Inserting Lists)

Bulleted lists might be familiar to you if you have worked with word processing or desktop publishing applications. Lists are a helpful way to present information to a reader without the formal constraints of a paragraph. They are especially important on the Web. Studies indicate that people typically skim web pages instead of reading them from beginning to end. Creating lists will make it easier for your visitors to get the most from your website without sifting through several paragraphs of text.

On the events.html page, click and drag to highlight the text which you want to put in the list.

Make sure you have the HTML button selected in the Property Inspector at the bottom of your page, and click the Unordered List button as shown in the Figure 254. The highlighted text becomes indented, and a bullet point is placed at the beginning of each line.

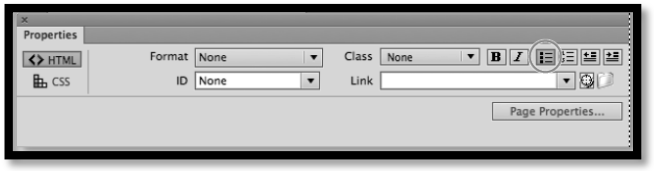


Figure 254: Adding unordered list

Use the Unordered List button in the Property Inspector to create a bulleted list.

Click the Ordered List button to the right of the Unordered List button. The bullets change to sequential numbers.

Choose Format > List > Properties to open the List Properties dialog box. Choose Bulleted List from the List type drop-down menu to return to your first style of list. The Numbered List and Bulleted List options in the List type drop-down menu also allow you to switch between ordered and unordered lists.

From the Style drop-down menu (see Figure 255), choose Square. This changes the default circular bullets to square bullets. Click OK to exit the List Properties dialog box.

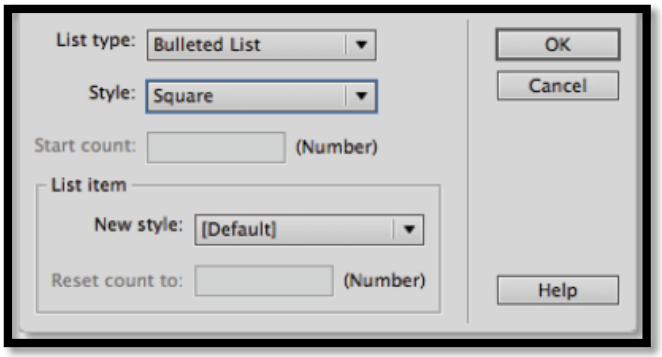


Figure 255: List of properties

Copyright Virtual University of Pakistan

Module 234

234. Web page Development (Inserting Images)

Adding image is very easy using Dreamweaver.

Go to Insert tab and press Image button as shown in the Figure 256.

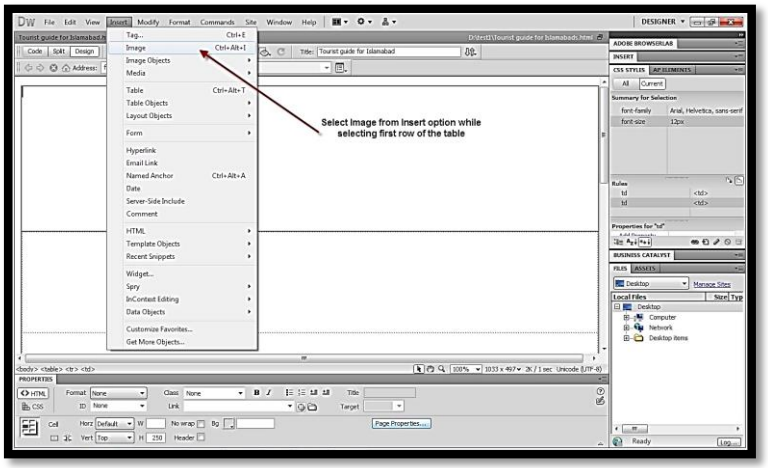


Figure 256: Inserting Image

Then you will be given the option to select the impage from your drive as shown in the Figure 257. You can select the required image and it will be added to your webpage wherever you had placed the cursor.

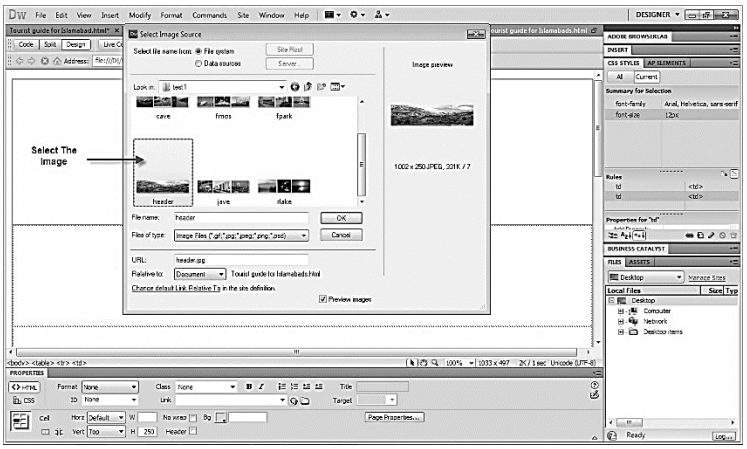


Figure 257: Browse Image

Copyright Virtual University of Pakistan