Lecture 15

Let's consider the implementation of the C++ classes for the non-terminals. We start with Expr.

```
bool Expr::isPresent()
{
      Term* op1 = new Term(s);
      if(!op1->isPresent())
            return false;
      tree = op1->AST();
      Eprime* op2 = new Eprime(s, tree);
      if(op2->isPresent())
          tree = op2->AST();
      return true;
}

bool Eprime::isPresent()
{
 int op=s->nextToken();
 if(op==PLUS || op==MINUS){
  s->advance();
  Term* op2=new Term(s);
  if(!op2->isPresent())
     syntaxError(s);
  TreeNode* t2=op2->AST();
  tree = new TreeNode(op,exprSofar,t2);
  Eprime* op3 = new Eprime(s, tree);
  if(op3->isPresent())
     tree = op3->AST();
     return true;
  }
 else return false;
}

bool Term::isPresent()
{
 Factor* op1 = new Factor(s);
 if(!op1->isPresent())
     return false;
 tree = op1->AST();
 Tprime* op2 = new Tprime(s, tree);
 if(op2->isPresent())
     tree = op2->AST();
 return true;
}
```

```
bool Tprime::isPresent()
{
 int  op=s->nextToken();
 if(op == MUL || op == DIV){
  s->advance();
  Factor* op2=new Factor(s);
  if(!op2->isPresent())
     syntaxError(s);
  TreeNode* t2=op2->AST();
  tree = new TreeNode(op,exprSofar,t2);
  Tprime* op3 = new Tprime(s, tree);
  if(op3->isPresent())
     tree = op3->AST();
     return true;
  }
 else return false;
}


bool Factor::isPresent()
{
 int op=s->nextToken();
 if(op == ID || op == NUM)
 {
  tree = new TreeNode(op,s->tokenValue());
  s->advance();
  return true;
 }
 if( op == LPAREN ){
  s->advance();
  Expr* opr = new Expr(s);
  if(!opr->isPresent() )
   syntaxError(s);
  if(s->nextToken() != RPAREN)
   syntaxError(s);
  s->advance();
  tree = opr->AST();
  return true;
 }
 return false;
}
```