Object-oriented design

▶ Goal: construct a representation for program.
  ▶ Identify and encapsulate information associated with different aspects of a program through C++ classes.
  ▶ Identify generalization and specialization relationship among various classes and represent them.
    Use inheritance to represent them.

▶ Approach:
  ▶ Construct an intermediate representation of program by constructing a parse tree.
    What is a parse tree?
  ▶ Represent nodes of a tree by specific C++ classes.
    How to recognize nodes? Nodes of tree are classified according to certain syntactic category.
  ▶ Associate attributes with parse tree nodes.
  ▶ Perform computation by traversing the tree.
    Also, during building of the tree

▶ Advantage:
  ▶ Modular design of compiler
  ▶ Encapsulation of information
  ▶ Extensibility of program
  ▶ Re-usability
How to construct Parse Tree?

- Two components:
  1. Identify what parse tree nodes are
  2. Identify relationship between nodes.

- Identification and representation of parse tree nodes:
  1. Identification: simplest approach is to represent each syntactic category as a parse tree node. Although you may want to optimize this to combine intermediate nonterminals.
  2. Representation mechanism: Use C++ class to represent each node. Attributes associated with each node can be defined as a member of the class. (Note: inheritance can help us here.) Methods: represent certain dynamic behavior of the node. can be used to access certain information; perform any semantic analysis; Do some computation such as code generation.

- A parse tree is a set of parse tree nodes. Can be implemented using STL vector mechanism.

```cpp
class ParseTreeNode {
    vector<ParseTreeNode *> children;
};
```

- Result of this analysis will lead you to construct a set of parse tree nodes, disjoint at this juncture.
Example

Grammar:

Statement ::= IfThenStatement | IfThenElseStatement
     | ForStatement  | Block
     | EmptyStatement | ExpressionStatement
     | ContinueStatement | ReturnStatement

EmptyStatement ::= ’ ; ’
ExpressionStatement ::= StatementExpression ’ ; ’
StatementExpression ::= Assignment
     | MethodInvocation
     | ClassInstanceCreationExpression

IfThenStatement ::= if ‘ ( ’ Expression ’ ) ’ Statement
IfThenElseStatement ::= if ‘ ( ’ Expression ’ ) ’ Statement
     else Statement

ForStatement ::= for ‘ ( ’ [ForInit] ’ ; ’ [Expression] ’ ; ’
     [ForUpdate] ) ’ Statement

ForInit ::= StatementExpressionList
     | LocalVariableDeclaration

ForUpdate ::= StatementExpressionList

StatementExpressionList ::= StatementExpression
     ( ’ , ’ StatementExpression)*

ContinueStatement ::= continue ’ ; ’
ReturnStatement ::= return [Expression] ’ ; ’

Create a class for each nonterminal

    class StatementClass ... { ... }
    class EmptyStatementClass ... { ... }
    class ExpressionStatement ... { ... }
    class StatementExpression ... { ... }
    class IfThenStatementClass ... { ... }
    class IfThenElseStatementClass ... { ... }
    class ForStatementClass ... { ... }
    class StatementExpressionListClass ... { ... }
    class ContinueStatementClass ... { ... }
    class ReturnStatementClasss ... { ... }
What should each class contain?

• For each NT, and its rule, look at its RHS and construct the class:
  
  $S ::= A \ c \ B \ d$

  Class for $S$ will contain:

  ```
  class SClass: public ParseTreeNode {
    ParseTreeNode *getA();
    void setA(ParseTreeNode *A);
    ParseTreeNode *getc();
    void setc(ParseTreeNode *c);
    ParseTreeNode *getB();
    void setB(ParseTreeNode *B);
    ParseTreeNode *getd();
    void setd(ParseTreeNode *d);
    ...
  }
  ```

  Note that ParseTreeNode contains a vector. So get and set methods can access the corresponding elements from vector.

• Example for ForStatement:

  ```
  class ForStatementClass: public ParseTreeNode {
    ParseTreeNode *GetInitExpression() {
      return children[0];
    }
    ParseTreeNode *GetLoopCondition(); {
      return children[1];
    }
    ParseTreeNode *GetLoopUpdateExpression() {
      return children[2];
    }
    ParseTreeNode *GetBody(); {
      return children[3];
    }
    ...
  }
  ```
Step 2: construct class-subclass relationships among parse tree node type.

Hierarchy is an important aspect of context free grammar.

```plaintext
Statement ::= IfThenStatement
| IfThenElseStatement
| ForStatement
| Block
| EmptyStatement
| ExpressionStatement
| ContinueStatement
| ReturnStatement
```

Captures information that `Statement` denotes general statements, whereas if-then-else, case, etc. are more specific kind.

Inheritance precisely captures this relationships:

```plaintext
class StatementClass: public ...
 { ... }
class IfThenClass: public StatementClass
 { ... }
class WhileStatementClass: public StatementClass
 { ... }
```

- `class StatementClass` will declare all common attributes and virtual methods.
- `IfThenElseClass` will extend its behavior by adding things that are specific for if-then-else statements.

Once you have designed the hierarchy, you can then now start to push information as well as computation up in the hierarchy.
Example

Grammar:

Expression ::= Expression ‘*’ Expression
   | Expression ‘/’ Expression
   | Expression ‘+’ Expression
   | Expression ‘-’ Expression
   | Expression ‘&&’ Expression
   | Expression ‘||’ Expression
   | Expression ‘==’ Expression
   | Expression ‘!=' Expression
   | Expression ‘<’ Expression
   | Expression ‘>’ Expression
   | Expression ‘<=’ Expression
   | Expression ‘>=’ Expression
   | Assignment | ‘-’ Expression
   | ‘!’ Expression | ‘!’ Expression
   | PrimitiveExpression

Hierarchy:
How can parse tree be constructed from yacc?

- After every rule, add an action that will create tree and add nodes.
- A blind approach to creating parse tree:

```plaintext
Statement ::= IfThenStatement
             | IfThenElseStatement
             | ForStatement
             | Block
             | EmptyStatement
             | ExpressionStatement
             | ContinueStatement
             | ReturnStatement
             { $$ = $1}
EmptyStatement ::= ';;'
             { $$ = new EmptyStatementClass(); }
ExpressionStatement ::= StatementExpression ';;'
             { $$ = new ExpressionStatementClass($1); }
StatementExpression ::= Assignment
                       | AutoExpression
                       | MethodInvocation
                       | ClassInstanceCreationExpression
             { $$ = $1; }
IfThenStatement ::= if '( Expression ')' Statement
             { $$ = new IfThenStatementClass($3, $5); }
IfThenElseStatement ::= if '( Expression ')' Statement else Statement
             { $$ = new IfThenStatementClass($3, $5, $7); }
WhileStatement ::= while '( Expression ')' Statement
             { $$ = new WhileStatementClass($3, $5); }
```

- Can perform many optimizations in terms of creating parse tree node.