Programming Assignment 1

Abstract Syntax Tree and Symbol Table

Objectives

This assignment is to create a parser which builds an abstract syntax tree (AST) and symbol tables from reading a miniC++ program. In the process of parsing, you need to build an AST and symbol table. At the end of parsing, you must print out an AST and symbol tables in two different text files. When you print out AST, try to print out C++ like format – it does not have to be the exact C++ but resembles the structure of source code and reflects the AST.

How to grade your submission?

1) Comparing a meaning of parsing result and input source code
2) Correctness of symbol tables

Due & files included in your submission

Please submit your source code packaged together through the KLAS. All files must be handed in one **tar.gz** file and it also need to include a README file which describes how to build and run your submission briefly. The tar.gz file also needs to include a Makefile, and a test code. Makefile have to basically contain “make” and “make clean”. Your grade will be mainly based on your program outputs which will be built and tested according to your instructions.

Questions & T.A. office hours

If you are not clear about the assignment, you can ask through email.

TAs

Park, Jeonghwan – j.park@skku.edu

miniC++ Grammar

```
// (a)+ : one or more repetition of a

// (a)* : zero or more repetition of a

// (a)? : a optionally exists

// unop : -

// addiop : +, -

// multop : *, /

// relaop :  <, >, <=, >=

// eqltop : ==, !=

// id : [A-Za-z_][A-Za-z0-9_]*

// intnum : [0-9]+

// floatnum : [0-9]+.[0-9]+


Program := (ClassList)? (ClassMethodList)? MainFunc

ClassList := (Class)+

Class := class id { (private : Member)? (public : Member)? }
Member := (VarDeclList)? (MethodDeclList)? (MethodDefList)?

VarDeclList := (VarDecl)+
MethodDeclList := (FuncDecl)+
MethodDefList := (FuncDef)+

VarDecl := Type Ident (= (int | float))? ;
FuncDecl := Type id ( (ParamList)? ) ;
FuncDef := Type id ( (ParamList)? ) CompoundStmt

ClassMethodList := (ClassMethodDef)+
ClassMethodDef := Type id :: id ( (ParamList)? ) CompoundStmt

MainFunc := int main ( ) CompoundStmt

ParamList := Param (, Param)*
Param := Type Ident

Ident := id | id [ intnum ]
Type := int | float | id
```

```
CompoundStmt := { (VarDeclList)? (StmtList)? }

StmtList := (Stmt)+

Stmt := ExprStmt
      | AssignStmt
      | RetStmt
      | WhileStmt
      | DoStmt
      | ForStmt
      | IfStmt
      | CompoundStmt
      | ;

ExprStmt := Expr ;

AssignStmt := RefVarExpr = Expr ;

RetStmt := return (Expr)? ;

WhileStmt := while ( Expr ) Stmt

DoStmt := do Stmt while ( Expr ) ;

ForStmt := for ( Expr ; Expr ; Expr ) Stmt

IfStmt := if ( Expr ) Stmt (else Stmt)?

Expr := OperExpr
      | RefExpr
      | intnum
      | floatnum

OperExpr := unop Expr
          | Expr addiop Expr
          | Expr multop Expr
          | Expr relaop Expr
          | Expr eqltop Expr
          | ( Expr )

RefExpr := RefVarExpr | RefCallExpr

RefVarExpr := (RefExpr .)? IdentExpr

RefCallExpr := (RefExpr .)? CallExpr

IdentExpr := id [ Expr ]
           | id

CallExpr := id ( (ArgList)? )

ArgList := Expr (, Expr)*
```

Operator Precedence and Associativity

Precedence	Operator	Description	Associativity
1	( )	Function call	Left-to-right
2	-	Unary minus	Right-to-left
3	*, /	Multiplication and division	Left-to-right
4	+, -	Addition and subtraction	Left-to-right
5	<, >, <=, >=	Relational operators	Left-to-right
6	==, !=	Equality operators	Left-to-right
7	=	Assignment	Right-to-left

Symbol Table (refer to the last slide of the lecture note)

Symbol table entry := <symbol name, symbol info>

Symbol info := <type, location, pointer to its associated declaration>

Location := level of nested blocks and order of blocks