Alexandre Charron

26606954

Compiler Design: Assignment #3

Modified grammar:

All rules colored red are semantic actions that are used to add symbols to tables.

All rules colored blue are used to check if the symbols are correct and may do some code generation.

All rules colored green are used for code generation

prog -> #newGlobal classDeclList progBody

classDeclList -> classDecl classDeclList

| EPSILON

classDecl -> class #pushId %pushId id #newClass %enter { classBody } ; #exit %exit

classBody -> #pushType type #pushId %pushId id varOrFunc | EPSILON

varOrFunc -> #newFunc %enter ( fParams ) funcBody ; #exit %exit funcDefList

| %pop arraySizeList #newVar ; classBody

funcDefList -> funcDef funcDefList

| EPSILON

progBody -> &enter %pushId program #newProg %enter funcBody ; #exit %exit &exit funcDefList

funcHead -> #pushType type #pushId %pushId id #newFunc %enter ( fParams )

funcDef -> funcHead funcBody ; #exit %exit

funcBody -> { funcBlock }

funcBlock -> for ( #pushType type #pushId id #newVar assignOp expr ; relExpr ; assignStat ) statBlock ; statementList

| if ( expr ) then statBlock else statBlock ; statementList

| get ( variable ) ; statementList

| put ( expr ) ; statementList

| return ( expr ) ; statementList

| #pushType float #pushId id arraySizeList #newVar ; funcBlock

| #pushType int #pushId id arraySizeList #newVar ; funcBlock

| #pushType %pushId id varOrStatement

| EPSILON

varOrStatement -> #pushId id arraySizeList #newVar ; funcBlock

| #pop indiceList %checkId variable1 assignOp expr %ass ; statementList

statementList -> statement statementList

| EPSILON

arraySizeList -> arraySize arraySizeList

| EPSILON

statement -> for ( #pushType type #pushId id #newVar assignOp expr %ass ; relExpr ; assignStat ) statBlock ;

| if ( expr ) then statBlock else statBlock ;

| get ( variable ) ;

| put ( expr ) ;

| return ( expr ) ;

| assignStat ;

assignStat -> variable assignOp expr %ass

statBlock -> { statementList } | statement

| EPSILON

expr -> arithExpr expr1

expr1 -> relOp arithExpr %rel | EPSILON

relExpr -> arithExpr relOp arithExpr %rel

arithExpr -> term arithExpr1

arithExpr1 -> addOp term %add arithExpr1

| EPSILON

sign -> + | -

term -> factor term1

term1 -> multOp factor %mul term1

| EPSILON

factor -> ( arithExpr ) | %factorF fnum | %pushId id factor1

| %factorI inum | not factor | sign factor

factor1 -> ( %currentScope aParams %exitScope ) %checkFunc | indiceList %checkId factor2

factor2 -> . %pushId id factor1 | EPSILON %exitScope %currentScope

variable -> %pushId id indiceList %checkId variable1

variable1 -> . %pushId id indiceList %checkId variable1

| EPSILON %exitScope %currentScope

indiceList -> indice indiceList | EPSILON

indice -> [ arithExpr %addIndex ]

arraySize -> [ #addDimension inum ]

type -> float | id | int

fParams -> #pushType type #pushId id arraySizeList #newVar #changeToParam fParamsTailList

| EPSILON

fParamsTailList -> fParamsTail fParamsTailList

| EPSILON

aParams -> expr %addParam aParamsTailList

| EPSILON

aParamsTailList -> aParamsTail aParamsTailList

| EPSILON

fParamsTail -> , #pushType type #pushId id arraySizeList #newVar #changeToParam

aParamsTail -> , expr %addParam

assignOp -> =

relOp -> < | <= | <> | == | > | >=

addOp -> + | - | or

multOp -> \* | / | and

Semantic checking:

-It will check if a variable, function or class has been redeclared.

-it will check if a variable or class has been declared before being defined.

-it will check if a function has the right amount and type and array size of parameters.

-it will check if a variable is being used with more array indexes than it has been defined with.

-it will check if both sides of an operator have the same type and array size.

-it will give a warning if opposite sides of an operator are an int and a float.

-it will check if array indices are integers, even if the index is the return value of a function.

-Entire arrays arent allowed in operations, but can be passed as an argument to a function.

DataStructures:

To create the symbol tables, I added a few classes to the program:

-SymbolTable: holds an ArrayList of Symbols and methods to manipulate SymbolTables, aswell as a pointer to it’s own symbol in another SymbolTable.

-Symbol: holds the symbol name, symbol kind(class, function, variable or paramater), the types and a link to another SymbolTable.

-Type: ArrayList that holds all types and dimensions associated to symbol. If variable or parameter the arraylist is of size 1 holding the type of the variable. If the type belongs to a function symbol, the first type in the arrayList is the return type, the rest are the function parameters.

The SemanticActions class has a method called action() that is called whenever the syntactic analyzer comes across a rule that starts with #. SemanticActions holds the global SymbolTable aswell as the current SymbolTable used to add new Symbols in the appropriate scope. There is also a scope stack of SymbolTables that is used to check if member functions or variables are properly declared, however it doesn’t work properly.

Error messages are printed in the error-filename.txt file and the SymbolTables are printed in symbolTables-filename.txt

Tools:

For this part of the assignment I just wrote down all the rules and iteratively added Semantic actions to the rules to accomplish new tasks. I didn’t use any additional software except for microsoft word and eclipse. I used the eclipse debugger a lot to fix and understand what the program is doing.