COMP 442 – Assignment 3

Semantic Analysis

Gordon Bailey

9541098

I certify that this submission is my original work and meets the Faculty's Expectations of Originality

Design Description

The symbol table is implemted as a singleton class which manages the current state of the symbol table creation. The singleton class SemanticContext stores the current symbol table, as well as information which is propogated throughout the parse tree during parsing.

Semantic actions are implemented as additional values in the same enumeration as terminal and nonterminal symbols, with the addition of an association with an instance of a class which implements the interface SemanticAction. This is a simple interface which essntially amounts to a function pointer. The use of this interface allows minimal modifications to the actual parser to add the processing of semantic actions.

The only modification to the parser is that it now checks the type of the current symbol, if the symbol is a semantic action, then the action is executed with the previously matched token as its argument. If the symbol is a terminal or nonterminal, then the parsing process continues as usual.

The symbol table itself is a very straightforward structure. It is simply a map of names to symbol table entries, and a pointer to the parent symbol table. Each symbol table entry contains the name of the entry, the kind of entry: either **variable**, **parameter**, **class** or **function**, a pointer to the symbol table associated to the entry (for functions or classes), and the type of the entry. The type is either a plain type: **int**, **float** or a **class type**, an array type, which is a plain type plus a set of dimensions, or a function type, which is a return type, and a list of parameter types.

Augmented Grammar

The grammar has been augmented with the following semantic actions.

Semantic Action	Meaning
#sem_CreateClassScope#	Creates an entry in the current symbol table for a new class, creates a new symbol table for that class's scope, and sets the current symbol table to the newly created symbol table.
#sem_StartFunction#	Initializes the data structures to begin storing the information for a new function definition. Uses the currently propogated id and type as the name and return type for the function.
#sem_AddFunctionParameter#	Adds a parameter to the current partially created function. This action should occur 0 or more times between a StartFunction action and a CreateFunction action. Uses the currently propogated id and type as the parameter type and parameter name.
#sem_CreateFunction#	Completes the process of creating a symbol table entry for a function. Uses the stored function name and parameter list, and creates a new symbol table entry for the function in the current symbol table, creates a new symbol table for the function's scope, and sets the current symbol table to the newly created symbol table.
#sem_StoreType#	Saves the type represented by the previously encountered token. This type information is effectively propogated through the parse tree until it is needed.
#sem_StoreId#	Saves the id represented by the previously encountered token. This ID is effectively propogated through the parse tree until it is needed.
#sem_StoreDimension#	Saves the dimension represented by the previously encountered token. This value is pushed onto the end of the current list of stored dimensions, and is effectively propogated through the parse tree until used.
#sem_CreateVariable#	Creates a new variable in the current symbol table using the currently propogated type, id, and dimensions.
#sem_EndScope#	Closes the current symbol table, setting the current symbol table to the parent of the closed table.

The completed augmented grammar is presented on the following pages.

<prog>	::=	<classDecl> <prog>
	|	<progBody>
<classDecl>	::=	class id #sem_CreateClassScope# { <classBody>
<classBody>	::=	<type> #sem_StoreType# id #sem_StoreId# <classBodyPrime>
	|	} ; #sem_EndScope#
<classBodyPrime>	::=	<classBodyVar>
	|	<classBodyFunc>
<classBodyVar>	::=	<varDeclArray> #sem_CreateVariable# <classBody>
<classBodyFunc>	::=	(#sem_StartFunction# <fParams>) #sem_CreateFunction# <funcBody> ; <classBodyFuncPrime>
<classBodyFuncPrime>	::=	<type> #sem_StoreType# id #sem_StoreId# <classBodyFunc>
	|	} ; #sem_EndScope#
<progBody>	::=	program #sem_StoreId# #sem_StoreType# #sem_StartFunction# #sem_CreateFunction# <funcBody> ; <funcDefsPrime>
<funcDefs>	::=	(#sem_StartFunction# <fParams>) #sem_CreateFunction# <funcBody> ; <funcDefsPrime>
	|	<funcDefsPrime>
<funcDefsPrime>	::=	<type> #sem_StoreType# id #sem_StoreId# <funcDefs>
	|	ε
<funcBody>	::=	{ <funcBodyVar>
<funcBodyVar>	::=	int #sem_StoreType# id #sem_StoreId# <varDeclArray> #sem_CreateVariable# <funcBodyVar>
	|	float #sem_StoreType# id #sem_StoreId# <varDeclArray> #sem_CreateVariable# <funcBodyVar>
	|	id <funcBodyVarPrime>
	|	<controlStat> <funcBodyStmt>
	|	} #sem_EndScope#
<funcBodyVarPrime>	::=	#sem_StoreType# id #sem_StoreId# <varDeclArray> #sem_CreateVariable# <funcBodyVar>
	|	<indices> <variablePrime> <assignStat> <funcBodyStmt>
<funcBodyStmt>	::=	<variable> <assignStat> <funcBodyStmt>
	|	<controlStat> <funcBodyStmt>
	|	} #sem_EndScope#
<varDeclArray>	::=	<arraySize> <varDeclArray>
	|	;
<controlStat>	::=	if (<expr>) then <statBlock> else <statBlock> ;
	|	for (<type> id <assignOp> <expr> ; <relExpr> ; <variable> <assignExpr>) <statBlock> ;
	|	get (<variable>) ;
	|	put (<expr>) ;
	|	return (<expr>) ;

```
<assignStat>              ::= <assignExpr> ;
<assignExpr>              ::= <assignOp> <expr>
<statBlock>               ::= { <statBlockStmts>
                          |   <variable> <assignStat>
                          |   <controlStat>
                          |   ε
<statBlockStmts>          ::= <variable> <assignStat> <statBlockStmts>
                          |   <controlStat> <statBlockStmts>
                          |   }
<expr>                    ::= <term> <exprPrime>
<exprPrime>               ::= <relOp> <expr>
                          |   <addOp> <expr>
                          |   ε
<relExpr>                 ::= <arithExpr> <relOp> <arithExpr>
<arithExpr>               ::= <term> <arithExprPrime>
<arithExprPrime>          ::= <addOp> <arithExpr>
                          |   ε
<sign>                    ::= +
                          |   –
<term>                    ::= <factor> <termPrime>
<termPrime>               ::= <multOp> <factor> <termPrime>
                          |   ε
<factor>                  ::= <factorIdNest>
                          |   int_literal
                          |   float_literal
                          |   ( <arithExpr> )
                          |   not <factor>
                          |   <sign> <factor>
<factorIdNest>            ::= id <factorIdNestPrime>
<factorIdNestPrime>       ::= <indices> <factorIdNestPrimePrime>
                          |   ( <aParams> )
<factorIdNestPrimePrime>  ::= . <factorIdNest>
                          |   ε
<variable>                ::= id <indices> <variablePrime>
<variablePrime>           ::= . <variable>
                          |   ε
<indice>                  ::= [ <arithExpr> ]
<indices>                 ::= <indice> <indices>
                          |   ε
<arraySize>               ::= [ int_literal #sem_StoreDimension# ]
```

<type> ::= int
| float
| id

<fParams> ::= <type> #sem_StoreType# id #sem_StoreId# <fParamsArraySz>
| ε

<fParamsArraySz> ::= <arraySize> <fParamsArraySz>
| #sem_AddFunctionParameter# <fParamsTailStar>

<fParamsTailStar> ::= <fParamsTail> <fParamsTailStar>
| ε

<fParamsTail> ::= , <type> #sem_StoreType# id #sem_StoreId# <arraySizeStar> #sem_AddFunctionParameter#

<arraySizeStar> ::= <arraySize> <arraySizeStar>
| ε

<aParams> ::= <expr> <aParamsTailStar>
| ε

<aParamsTailStar> ::= <aParamsTail> <aParamsTailStar>
| ε

<aParamsTail> ::= , <expr>

<assignOp> ::= =

<relOp> ::= ==
| <>
| <
| >
| <=
| >=

<addOp> ::= +
| -
| or

<multOp> ::= *
| /
| and