COMP 442 – Assignment 1

Syntactic Analyzer

Gordon Bailey

9541098

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

Original Grammar

The original grammar with all star notations replaced with list-generating productions is as follows.

```
<prog>          ::= <classDecl> <prog> | <progBody>
<classDecl>     ::= class id { <classBodyVar>
<classBodyVar>  ::= <varDecl> <classBodyVar> | <classBodyFunc>
<classBodyFunc> ::= <funcDef> <classBodyFunc> | } ;
<progBody>      ::= program <funcBody> ; <funcDefs>
<funcDefs>      ::= <funcDef> <funcDefs> | ε
<funcHead>      ::= <type> id ( <fParams> )
<funcDef>       ::= <funcHead><funcBody>;
<funcBody>      ::= { <funcBodyVar>
<funcBodyVar>   ::= <varDecl> <funcBodyVar> | <funcBodyStmt>
<funcBodyStmt>  ::= <statement> <funcBodyStmt> | }
<varDecl>       ::= <type> id <varDeclArray>
<varDeclArray>  ::= <arraySize> <varDeclArray> | ;
<statement>     ::= <assignStat> ;
                  | if ( <expr> ) then <statBlock> else <statBlock> ;
                  | for(<type>id<assignOp><expr>;<relExpr>;<assignStat>)<statBlock>;
                  | get(<variable>);
                  | put(<expr>);
                  | return(<expr>);
<assignStat>    ::= <variable><assignOp><expr>
<statBlock>     ::= { <statBlockStmts> | <statement> | ε
<statBlockStmts>::= <statement> <statBlockStmts> | }
<expr>          ::= <arithExpr> | <relExpr>
<relExpr>       ::= <arithExpr><relOp><arithExpr>
<arithExpr>     ::= <arithExpr><addOp><term> | <term>
<sign>          ::= + | –
<term>          ::= <term><multOp><factor> | <factor>
<factor>        ::= <variable>
                  | <factorIdNest>
                  | int | float
                  | (<arithExpr>)
                  | not <factor>
                  | sign <factor>
<factorIdNest>  ::= <idNest> <factorIdNest> | id ( <aParams> )
<variable>      ::= <idNest> <variable> | id <variableIndice>
<variableIndice>::= <indice> <variableIndice> | ε
<idNest>        ::= id <idNestIndices>
<idNestIndices> ::= <indice> <idNestIndices> | .
<indice>        ::= [ <arithExpr> ]
<arraySize>     ::= [ int ]
<type>          ::= int | float | id
```

<fParams> ::= <type> id <fParamsArraySz> | ε
<fParamsArraySz> ::= <arraySize> <fParamsArraySize> | <fParamsTailStar>
<fParamsTailStar> ::= <fParamsTail> <fParamsTailStar> | ε
<fParamsTail> ::= , <type> id <arraySizeStar>
<arraySizeStar> ::= <arraySize> <arraySizeStar> | ε
<aParams> ::= <expr> <aParamsTailStar> | ε
<aParamsTailStar> ::= <aParamsTail> <aParamsTailStar> | ε
<aParamsTail> ::= , <expr>
<assignOp> ::= =
<relOp> ::= == | <> | < | > | <= | >=
<addOp> ::= + | - | or
<multOp> ::= * | / | and

Modified Grammar

An algorithm was written to analyze the previously presented grammar and to identify any left recursions or ambiguities. Based on this testing, the following left-recursions exist.

<arithExpr>	::=	<arithExpr><addOp><term> | <term>
<term>	::=	<term><multOp><factor> | <factor>

They are replaced as follows.

<arithExpr>	::=	<term><arithExprPrime>
<arithExprPrime>	::=	<addOp><term><arithExprPrime> | ε
<term>	::=	<factor><termPrime>
<termPrime>	::=	<multOp><factor><termPrime> | ε

Based on this testing, the following ambiguities exist.

LHS	terminals
<factorIdNest>	tok_id
<variable>	tok_id
<expr>	tok_not, tok_float_literal, tok_id, tok_plus, tok_minus, tok_int_literal, tok_open_paren
<classBodyVar>	tok_id, tok_int, tok_float
<funcBodyVar>	tok_id
<factor>	tok_id

These ambiguities were removed after some careful consideration and re-organization of the previously presented grammar. The final grammar with all ambiguities removed is presented on the following page. This grammar has been tested using the same algorithm to ensure that no left-recursions or ambiguities exist.

The final revised grammar is as follows.

```
<prog>            ::= <classDecl> <prog>
                    | <progBody>
<classDecl>       ::= class id { <classBody>
<classBody>       ::= <type> id <classBodyPrime>
                    | } ;
<classBodyPrime>  ::= <classBodyVar>
                    | <classBodyFunc>
<classBodyVar>    ::= <varDeclArray> <classBody>
                    | <classBody>
<classBodyFunc>   ::= ( <fParams> ) <funcBody> ; <classBodyFuncPrime>
<classBodyFuncPrime> ::= <type> id <classBodyFunc>
                    | } ;
<progBody>        ::= program <funcBody> ; <funcDefsPrime>
<funcDefs>        ::= ( <fParams> ) <funcBody> ; <funcDefsPrime>
                    | <funcDefsPrime>
<funcDefsPrime>   ::= <type> id <funcDefs>
                    | ε
<funcBody>        ::= { <funcBodyVar>
<funcBodyVar>     ::= int id <varDeclArray> <funcBodyVar>
                    | float id <varDeclArray> <funcBodyVar>
                    | id <funcBodyVarPrime>
                    | <controlStat> <funcBodyStmt>
                    | }
<funcBodyVarPrime> ::= id <varDeclArray> <funcBodyVar>
                    | <indices> <variablePrime> <assignStat> <funcBodyStmt>
<funcBodyStmt>    ::= <variable> <assignStat> <funcBodyStmt>
                    | <controlStat> <funcBodyStmt>
                    | }
<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]
<type>	::=	int
	|	float
	|	id
<fParams>	::=	<type> id <fParamsArraySz>
	|	ε
<fParamsArraySz>	::=	<arraySize> <fParamsArraySz>
	|	<fParamsTailStar>
<fParamsTailStar>	::=	<fParamsTail> <fParamsTailStar>
	|	ε
<fParamsTail>	::=	, <type> id <arraySizeStar>
<arraySizeStar>	::=	<arraySize> <arraySizeStar>
	|	ε
<aParams>	::=	<expr> <aParamsTailStar>
	|	ε
<aParamsTailStar>	::=	<aParamsTail> <aParamsTailStar>
	|	ε
<aParamsTail>	::=	, <expr>
<assignOp>	::=	=

```
<relOp>   ::=   ==
          |     <>
          |     <
          |     >
          |     <=
          |     >=
<addOp>   ::=   +
          |     –
          |     or
<multOp>  ::=   *
          |     /
          |     and
```

First and Follows Sets

An algorithm was written to analyze the previously presented grammar and obtain the First and Follows sets. The First set for each non terminal is presented below. (Not the First set for each terminal is simply the terminal itself).

controlStat	[tok_for, tok_put, tok_if, tok_return, tok_get]
funcDefs	[EPSILON, tok_float, tok_int, tok_open_paren, tok_id]
expr	[tok_plus, tok_float_literal, tok_minus, tok_int_literal, tok_open_paren, tok_not, tok_id]
variable	[tok_id]
multOp	[tok_and, tok_star, tok_slash]
type	[tok_float, tok_int, tok_id]
addOp	[tok_or, tok_plus, tok_minus]
classBodyVar	[tok_float, tok_close_brace, tok_int, tok_id, tok_semicolon, tok_open_square]
classBody	[tok_close_brace, tok_float, tok_int, tok_id]
funcBodyVar	[tok_for, tok_close_brace, tok_float, tok_int, tok_put, tok_if, tok_return, tok_id, tok_get]
classDecl	[tok_class]
relExpr	[tok_float_literal, tok_plus, tok_int_literal, tok_minus, tok_open_paren, tok_not, tok_id]
arithExprPrime	[EPSILON, tok_or, tok_plus, tok_minus]
arraySize	[tok_open_square]
term	[tok_float_literal, tok_plus, tok_int_literal, tok_minus, tok_open_paren, tok_not, tok_id]
factorIdNestPrime	[EPSILON, tok_open_paren, tok_open_square]
progBody	[tok_program]
funcBodyVarPrime	[EPSILON, tok_id, tok_open_square]
fParamsTailStar	[EPSILON, tok_comma]
variablePrime	[EPSILON, tok_dot]
classBodyPrime	[tok_close_brace, tok_float, tok_int, tok_open_paren, tok_semicolon, tok_id, tok_open_square]
aParamsTail	[tok_comma]
statBlockStmts	[tok_for, tok_close_brace, tok_put, tok_if, tok_return, tok_id, tok_get]
funcBody	[tok_open_brace]

prog	[tok_program, tok_class]
factorIdNestPrimePrime	[EPSILON, tok_dot]
fParamsTail	[tok_comma]
fParamsArraySz	[EPSILON, tok_comma, tok_open_square]
classBodyFunc	[tok_open_paren]
arraySizeStar	[EPSILON, tok_open_square]
exprPrime	[tok_equals, tok_less_than, EPSILON, tok_or, tok_less_than_equals, tok_plus, tok_minus, tok_diamond, tok_greater_than, tok_greater_than_equals]
fParams	[EPSILON, tok_float, tok_int, tok_id]
funcDefsPrime	[EPSILON, tok_float, tok_int, tok_id]
assignExpr	[tok_assignment]
indice	[tok_open_square]
funcBodyStmt	[tok_for, tok_close_brace, tok_put, tok_if, tok_return, tok_id, tok_get]
aParamsTailStar	[EPSILON, tok_comma]
statBlock	[tok_for, EPSILON, tok_open_brace, tok_put, tok_if, tok_return, tok_id, tok_get]
arithExpr	[tok_plus, tok_float_literal, tok_minus, tok_int_literal, tok_open_paren, tok_not, tok_id]
sign	[tok_plus, tok_minus]
classBodyFuncPrime	[tok_close_brace, tok_float, tok_int, tok_id]
assignStat	[tok_assignment]
varDeclArray	[tok_semicolon, tok_open_square]
assignOp	[tok_assignment]
termPrime	[EPSILON, tok_and, tok_star, tok_slash]
relOp	[tok_less_than, tok_equals, tok_less_than_equals, tok_diamond, tok_greater_than, tok_greater_than_equals]
factor	[tok_plus, tok_float_literal, tok_minus, tok_int_literal, tok_open_paren, tok_not, tok_id]
indices	[EPSILON, tok_open_square]
factorIdNest	[tok_id]
aParams	[EPSILON, tok_float_literal, tok_plus, tok_int_literal, tok_minus, tok_open_paren, tok_not, tok_id]

An algorithm was also used to determine the follows set for each non-terminal, and the sets are presented below.

aParamsTail	[tok_comma, tok_close_paren]
classBodyFuncPrime	[tok_program, tok_class]
assignOp	[tok_open_paren, tok_int_literal, tok_not, tok_float_literal, tok_plus, tok_id, tok_minus]
funcBodyVar	[tok_semicolon]
arithExpr	[tok_diamond, tok_greater_than_equals, tok_greater_than, tok_equals, tok_less_than, tok_close_paren, tok_semicolon, tok_less_than_equals, tok_close_square]
fParamsTailStar	[tok_close_paren]
factor	[tok_diamond, tok_greater_than_equals, tok_equals, tok_less_than, tok_or, tok_minus, tok_comma, tok_and, tok_star, tok_greater_than, tok_close_paren, tok_semicolon, tok_less_than_equals, tok_close_square, tok_plus, tok_slash]
fParamsArraySz	[tok_close_paren]
addOp	[tok_open_paren, tok_int_literal, tok_not, tok_float_literal, tok_plus, tok_id, tok_minus]
fParamsTail	[tok_comma, tok_close_paren]
arraySizeStar	[tok_comma, tok_close_paren]
classBody	[tok_program, tok_class]
funcBodyStmt	[tok_semicolon]
controlStat	[tok_else, tok_return, tok_for, tok_if, tok_semicolon, tok_id, tok_put, tok_close_brace, tok_get]
termPrime	[tok_diamond, tok_comma, tok_greater_than_equals, tok_equals, tok_greater_than, tok_less_than, tok_close_paren, tok_semicolon, tok_less_than_equals, tok_close_square, tok_plus, tok_or, tok_minus]
assignExpr	[tok_close_paren, tok_semicolon]
exprPrime	[tok_comma, tok_close_paren, tok_semicolon]
classBodyPrime	[tok_program, tok_class]
factorIdNestPrimePrime	[tok_diamond, tok_greater_than_equals, tok_equals, tok_less_than, tok_or, tok_minus, tok_comma, tok_and, tok_star, tok_greater_than, tok_close_paren, tok_semicolon, tok_less_than_equals, tok_close_square, tok_plus, tok_slash]

arithExprPrime	[tok_diamond, tok_greater_than_equals, tok_greater_than, tok_equals, tok_less_than, tok_close_paren, tok_semicolon, tok_less_than_equals, tok_close_square]
arraySize	[tok_comma, tok_open_square, tok_close_paren, tok_semicolon]
funcBody	[tok_semicolon]
classDecl	[tok_program, tok_class]
relExpr	[tok_semicolon]
variable	[tok_close_paren, tok_assignment]
relOp	[tok_open_paren, tok_int_literal, tok_not, tok_float_literal, tok_plus, tok_id, tok_minus]
multOp	[tok_open_paren, tok_int_literal, tok_not, tok_float_literal, tok_plus, tok_id, tok_minus]
factorIdNestPrime	[tok_diamond, tok_greater_than_equals, tok_equals, tok_less_than, tok_or, tok_minus, tok_comma, tok_and, tok_star, tok_greater_than, tok_close_paren, tok_semicolon, tok_less_than_equals, tok_close_square, tok_plus, tok_slash]
aParamsTailStar	[tok_close_paren]
variablePrime	[tok_close_paren, tok_assignment]
funcBodyVarPrime	[tok_semicolon]
factorIdNest	[tok_diamond, tok_greater_than_equals, tok_equals, tok_less_than, tok_or, tok_minus, tok_comma, tok_and, tok_star, tok_greater_than, tok_close_paren, tok_semicolon, tok_less_than_equals, tok_close_square, tok_plus, tok_slash]
prog	[END_MARKER]
funcDefs	[END_MARKER]
sign	[tok_open_paren, tok_int_literal, tok_not, tok_float_literal, tok_plus, tok_id, tok_minus]
classBodyVar	[tok_program, tok_class]
indices	[tok_diamond, tok_greater_than_equals, tok_equals, tok_less_than, tok_assignment, tok_or, tok_minus, tok_comma, tok_and, tok_star, tok_greater_than, tok_close_paren, tok_semicolon, tok_less_than_equals, tok_close_square, tok_plus, tok_dot, tok_slash]
funcDefsPrime	[END_MARKER]

varDeclArray	[tok_int, tok_return, tok_float, tok_for, tok_if, tok_id, tok_put, tok_close_brace, tok_get]
assignStat	[tok_else, tok_return, tok_for, tok_if, tok_semicolon, tok_id, tok_put, tok_close_brace, tok_get]
aParams	[tok_close_paren]
progBody	[END_MARKER]
classBodyFunc	[tok_program, tok_class]
type	[tok_id]
statBlock	[tok_else, tok_semicolon]
fParams	[tok_close_paren]
statBlockStmts	[tok_else, tok_semicolon]
term	[tok_diamond, tok_comma, tok_greater_than_equals, tok_equals, tok_greater_than, tok_less_than, tok_close_paren, tok_semicolon, tok_less_than_equals, tok_close_square, tok_plus, tok_or, tok_minus]
expr	[tok_comma, tok_close_paren, tok_semicolon]
indice	[tok_diamond, tok_greater_than_equals, tok_equals, tok_open_square, tok_less_than, tok_assignment, tok_or, tok_minus, tok_comma, tok_and, tok_star, tok_greater_than, tok_close_paren, tok_semicolon, tok_less_than_equals, tok_close_square, tok_plus, tok_dot, tok_slash]

Implementation

The implementation is somewhat of a hybrid of the table-based method. Instead of building a parse table, the grammar itself is stored in a data structure inside the parser. Upon startup, the parser calculates the first and follows sets for each nonterminal. A potential optimization would be to pre-generate these sets once and store them statically, but so far performance has not appeared to be an issue.

The parse function itself is implemented as a recursive function which operates on the input stream of tokens to build a tree structure which is returned. At each step, this recursive function looks at the current symbol, and analyzes all of the productions for that symbol, choosing the correct one based on the first and follows sets of the first symbol of that production. This approach sacrifices some performance versus a table based or hand-written method, but is far more maintanable than either. The code could also easily be adapted to automatically generate a parser table and be converted to a table-based method if performance becomes an issue.

Error Recovery

The error recovery method implemented is a panic mode method which skips input tokens until the next token is a member of the First set of the current symbol. At the beginning of each recursive call to the parse function, the error recovery function is called, and skips tokens until a valid one is present. Each skipped token is reported as an error, and the line number, unexpected token, and set of possible expected tokens is reported.