Alexandru Bara 40132235 **Comp 442 Assignment 2**

Section 1: Transformed Grammar.

From the original grammar, I was able to run Dr. Paquets tool in order to remove the left recursion. Once that was done I was still left with some ambiguities. Here are the ones that were left:

arithExpr -> arithExpr addOp term
arithExpr -> term

arraySize -> '[' 'intNum' ']'
arraySize -> '[' ']'

expr -> arithExpr
expr -> relExpr

factor -> variable
factor -> functionCall
factor -> 'intLit'
factor -> 'floatLit'
factor -> '(' arithExpr ')'
factor -> 'not' factor
factor -> sign factor

funcHead -> 'function' opt-funcHead1 'id' '(' fParams ')' 'arrow' returnType
funcHead -> 'function' 'id' 'sr' 'constructor' '(' fParams ')'

idnest -> 'id' rept-idnest1 '.'
idnest -> 'id' '(' aParams ')' '.'

localVarDecl -> 'localVar' 'id' ':' type rept-localVarDecl4 ';'
localVarDecl -> 'localVar' 'id' ':' type '(' aParams ')' ';'

opt-funcHead1 -> 'id' 'sr'
opt-funcHead1 -> EPSILON

rept-functionCall0 -> idnest rept-functionCall0
rept-functionCall0 -> EPSILON

rept-variable0 -> idnest rept-variable0
rept-variable0 -> EPSILON

statement -> assignStat ';'
statement -> 'if' '(' relExpr ')' 'then' statBlock 'else' statBlock ';'
statement -> 'while' '(' relExpr ')' statBlock ';'
statement -> 'read' '(' variable ')' ';'
statement -> 'write' '(' expr ')' ';'
statement -> 'return' '(' expr ')' ';'
statement -> functionCall ';'

From here, some of the ambiguities were pretty easy to fix like the arraySize in which all I had to do was factor out the '['. Some of the ambiguities were a bit harder to fix and I had to change the full structure of the grammar, for example the statement and factor. Once that was done, I was left with this following grammar. I put the grammar in Ucalgary format since I needed it to create the parsing table and the first/follow sets.

START -> REPTSTART0 .

APARAMS -> EXPR REPTAPARAMS1 .
APARAMS -> .

APARAMSTAIL -> comma EXPR .

ADDOP -> plus .
ADDOP -> minus .
ADDOP -> or .

ARITHEXPR -> TERM RIGHTREPTARITHEXPR .

ARRAYSIZE -> lsqbr ARRAYSIZE2 .
ARRAYSIZE2 -> intlit rsqbr .
ARRAYSIZE2 -> rsqbr .

ASSIGNOP -> equal .

CLASSDECL -> class id OPTCLASSDECL2 lcurbr REPTCLASSDECL4 rcurbr semi .

CLASSDECL2 -> CLASSDECL .
CLASSDECL2 -> FUNCDEF .

EXPR -> ARITHEXPR EXPR2 .

EXPR2 -> RELOP ARITHEXPR .
EXPR2 -> .

```
FPARAMS -> id colon TYPE REPTFPARAMS3 REPTFPARAMS4   .
FPARAMS ->  .

FPARAMSTAIL -> comma id colon TYPE REPTFPARAMSTAIL4   .

FACTOR -> FUNCTIONCALLORVARIABLE   .
FACTOR -> intlit   .
FACTOR -> floatlit   .
FACTOR -> lpar ARITHEXPR rpar   .
FACTOR -> not FACTOR   .
FACTOR -> SIGN FACTOR   .

FUNCBODY -> lcurbr REPTFUNCBODY1 rcurbr   .

FUNCDEF -> FUNCHEAD FUNCBODY  .

FUNCHEAD -> function id FUNCHEAD3  .

FUNCHEAD2 -> id lpar FPARAMS rpar arrow RETURNTYPE   .
FUNCHEAD2 -> constructor lpar FPARAMS rpar   .

FUNCHEAD3 -> sr FUNCHEAD2  .
FUNCHEAD3 -> lpar FPARAMS rpar arrow RETURNTYPE   .

ASSIGNSTAT -> VARIABLE ASSIGNOP EXPR   .

FUNCTIONCALL -> id FUNCALL3  .
FUNCALL3 -> lpar APARAMS rpar  .
FUNCALL3 -> FUNCALL2  .
FUNCALL2 -> dot id FUNCALL4  .
FUNCALL4 -> INDICE FUNCALL2  .
FUNCALL4 -> lpar APARAMS rpar FUNCALL5  .
FUNCALL5 -> FUNCALL2  .
FUNCALL5 ->  .

VARIABLE -> id VARIABLE3  .
VARIABLE3 -> INDICE  .
VARIABLE3 -> VARIABLE2  .
VARIABLE3 ->  .
VARIABLE2 -> dot id VARIABLE4  .
VARIABLE4 -> lpar APARAMS rpar VARIABLE2  .
VARIABLE4 -> INDICE VARIABLE5  .
VARIABLE5 -> VARIABLE2  .
VARIABLE5 ->  .
```

```
FUNCTIONCALLORVARIABLE -> id FUNCTIONCALLORVARIABLE1  .
FUNCTIONCALLORVARIABLE1 -> INDICELOOP  FUNCTIONCALLORVARIABLE2  .
FUNCTIONCALLORVARIABLE1 -> lpar APARAMS rpar FUNCTIONCALLORVARIABLE2  .
FUNCTIONCALLORVARIABLE2 -> dot id FUNCTIONCALLORVARIABLE3  .
FUNCTIONCALLORVARIABLE2 ->  .
FUNCTIONCALLORVARIABLE3 -> INDICELOOP FUNCTIONCALLORVARIABLE2  .
FUNCTIONCALLORVARIABLE3 -> lpar APARAMS rpar FUNCTIONCALLORVARIABLE2  .

IDNEST1 -> dot id IDNEST2  .
IDNEST2 -> lsqbr ARITHEXPR rsqbr IDNEST2  .
IDNEST2 -> lpar APARAMS rpar  .
IDNEST2 ->  .

INDICE -> lsqbr ARITHEXPR rsqbr   .

LOCALVARDECL -> localvar id colon TYPE LOCALVARDECL2  .
LOCALVARDECL2 -> REPTLOCALVARDECL4 semi   .
LOCALVARDECL2 -> lpar APARAMS rpar semi   .

LOCALVAR2 -> LOCALVARDECL   .
LOCALVAR2 -> STATEMENT   .

MEMBERDECL -> MEMBERFUNCDECL   .
MEMBERDECL -> MEMBERVARDECL   .

MEMBERFUNCDECL -> function id colon lpar FPARAMS rpar arrow RETURNTYPE semi   .
MEMBERFUNCDECL -> constructor colon lpar FPARAMS rpar semi   .

MEMBERVARDECL -> attribute id colon TYPE REPTMEMBERVARDECL4 semi   .

MULTOP -> mult   .
MULTOP -> div   .
MULTOP -> and   .

OPTCLASSDECL2 -> isa id REPTOPTCLASSDECL22   .
OPTCLASSDECL2 ->  .

RELEXPR -> ARITHEXPR RELOP ARITHEXPR   .

RELOP -> eq   .
RELOP -> neq   .
RELOP -> lt   .
RELOP -> gt   .
RELOP -> leq   .
RELOP -> geq   .

REPTSTART0 -> CLASSDECL2 REPTSTART0   .
REPTSTART0 ->  .

REPTAPARAMS1 -> APARAMSTAIL REPTAPARAMS1   .
REPTAPARAMS1 ->  .

REPTCLASSDECL4 -> VISIBILITY MEMBERDECL REPTCLASSDECL4   .
REPTCLASSDECL4 ->  .

REPTFPARAMS3 -> ARRAYSIZE REPTFPARAMS3   .
REPTFPARAMS3 ->  .

REPTFPARAMS4 -> FPARAMSTAIL REPTFPARAMS4   .
REPTFPARAMS4 ->  .

REPTFPARAMSTAIL4 -> ARRAYSIZE REPTFPARAMSTAIL4   .
REPTFPARAMSTAIL4 ->  .

REPTFUNCBODY1 -> LOCALVAR2 REPTFUNCBODY1   .
REPTFUNCBODY1 ->  .

REPTLOCALVARDECL4 -> ARRAYSIZE REPTLOCALVARDECL4   .
REPTLOCALVARDECL4 ->  .

REPTMEMBERVARDECL4 -> ARRAYSIZE REPTMEMBERVARDECL4   .
REPTMEMBERVARDECL4 ->  .

REPTOPTCLASSDECL22 -> comma id REPTOPTCLASSDECL22   .
REPTOPTCLASSDECL22 ->  .

REPTSTATBLOCK1 -> STATEMENT REPTSTATBLOCK1   .
REPTSTATBLOCK1 ->  .

RETURNTYPE -> TYPE   .
RETURNTYPE -> void   .

RIGHTREPTARITHEXPR ->  .
RIGHTREPTARITHEXPR -> ADDOP TERM RIGHTREPTARITHEXPR   .

RIGHTRECTERM ->  .
RIGHTRECTERM -> MULTOP FACTOR RIGHTRECTERM   .
```

```
SIGN -> plus   .
SIGN -> minus   .

STATBLOCK -> lcurbr REPTSTATBLOCK1 rcurbr   .
STATBLOCK -> STATEMENT   .
STATBLOCK ->  .

STATEMENT -> STATEMENT2 semi   .
STATEMENT -> if lpar RELEXPR rpar then STATBLOCK else STATBLOCK semi   .
STATEMENT -> while lpar RELEXPR rpar STATBLOCK semi   .
STATEMENT -> read lpar VARIABLE rpar semi   .
STATEMENT -> write lpar EXPR rpar semi   .
STATEMENT -> return lpar EXPR rpar semi  .


STATEMENT2 -> id STATEMENT3  .

STATEMENT3 -> lpar APARAMS rpar AFTERFUNCTIONCALL  .
STATEMENT3 -> INDICELOOP AFTERVARIABLE  .

AFTERFUNCTIONCALL -> dot id MIDDLESTATE  .
AFTERVARIABLE -> dot id MIDDLESTATE  .

MIDDLESTATE -> INDICELOOP AFTERVARIABLE  .
MIDDLESTATE -> lpar APARAMS rpar AFTERFUNCTIONCALL  .

AFTERVARIABLE -> ENDASSIGN  .
AFTERFUNCTIONCALL ->  .

INDICELOOP -> INDICE INDICELOOP  .
INDICELOOP ->  .
ENDASSIGN -> ASSIGNOP EXPR  .

TERM -> FACTOR RIGHTRECTERM   .

TYPE -> integer   .
TYPE -> float   .
TYPE -> id   .

VISIBILITY -> public   .
VISIBILITY -> private   .
VISIBILITY ->  .
```

Section 2: First and follow sets

nonterminal	first set	follow set	nullable	endable
START	class function	∅	yes	yes
ARRAYSIZE2	intlit rsqbr	lsqbr semi rpar comma	no	no
CLASSDECL	class	class function	no	yes
EXPR2	eq neq lt gt leq geq	semi comma rpar	yes	no
FUNCDEF	function	class function	no	yes
FUNCBODY	lcurbr	class function	no	yes
FUNCHEAD	function	lcurbr	no	no
FUNCHEAD3	sr lpar	lcurbr	no	no
FUNCHEAD2	id constructor	lcurbr	no	no
ASSIGNSTAT	id	∅	no	no
FUNCTIONCALL	id	∅	no	no
FUNCALL3	lpar dot	∅	no	no
FUNCALL4	lpar lsqbr	∅	no	no
FUNCALL5	dot	∅	yes	no
FUNCALL2	dot	∅	no	no
VARIABLE3	lsqbr dot	equal rpar	yes	no
VARIABLE4	lpar lsqbr	equal rpar	no	no
VARIABLE5	dot	equal rpar	yes	no
VARIABLE2	dot	equal rpar	no	no
FUNCTIONCALLORVARIABLE	id	semi mult div and rsqbr eq neq lt gt leq geq plus minus or comma rpar	no	no
FUNCTIONCALLORVARIABLE1	lpar dot lsqbr	semi mult div and rsqbr eq neq lt gt leq geq plus minus or comma rpar	yes	no

FUNCTIONCALLORVARIABLE3	lpar dot lsqbr	semi mult div and rsqbr eq neq lt gt leq geq plus minus or comma rpar	yes	no
FUNCTIONCALLORVARIABLE2	dot	semi mult div and rsqbr eq neq lt gt leq geq plus minus or comma rpar	yes	no
IDNEST1	dot	∅	no	no
IDNEST2	lsqbr lpar	∅	yes	no
LOCALVARDECL2	semi lpar lsqbr	localvar if while read write return id rcurbr	no	no
LOCALVARDECL	localvar	localvar if while read write return id rcurbr	no	no
MEMBERFUNCDECL	function constructor	public private function constructor attribute rcurbr	no	no
FPARAMS	id	rpar	yes	no
MEMBERVARDECL	attribute	public private function constructor attribute rcurbr	no	no
OPTCLASSDECL2	isa	lcurbr	yes	no
ARITHEXPR	intlit floatlit lpar not id plus minus	semi rsqbr eq neq lt gt leq geq comma rpar	no	no
RELOP	eq neq lt gt leq geq	intlit floatlit lpar not id plus minus	no	no
CLASSDECL2	class function	class function	no	yes
REPTSTART0	class function	∅	yes	yes
APARAMSTAIL	comma	comma rpar	no	no
REPTAPARAMS1	comma	rpar	yes	no

MEMBERDECL	function constructor attribute	public private function constructor attribute rcurbr	no	no
REPTCLASSDECL4	public private function constructor attribute	rcurbr	yes	no
REPTFPARAMS3	lsqbr	rpar comma	yes	no
FPARAMSTAIL	comma	comma rpar	no	no
REPTFPARAMS4	comma	rpar	yes	no
REPTFPARAMSTAIL4	lsqbr	comma rpar	yes	no
LOCALVAR2	localvar if while read write return id	localvar if while read write return id rcurbr	no	no
REPTFUNCBODY1	localvar if while read write return id	rcurbr	yes	no
REPTLOCALVARDECL4	lsqbr	semi	yes	no
ARRAYSIZE	lsqbr	lsqbr semi rpar comma	no	no
REPTMEMBERVARDECL4	lsqbr	semi	yes	no
REPTOPTCLASSDECL22	comma	lcurbr	yes	no
RETURNTYPE	void integer float id	semi lcurbr	no	no
ADDOP	plus minus or	intlit floatlit lpar not id plus minus	no	no
RIGHTREPTARITHEXPR	plus minus or	semi rsqbr eq neq lt gt leq geq comma rpar	yes	no
MULTOP	mult div and	intlit floatlit lpar not id plus minus	no	no

SIGN	plus minus	intlit floatlit lpar not id plus minus	no	no
REPTSTATBLOCK1	if while read write return id	rcurbr	yes	no
STATEMENT	if while read write return id	else semi localvar if while read write return id rcurbr	no	no
RELEXPR	intlit floatlit lpar not id plus minus	rpar	no	no
STATBLOCK	lcurbr if while read write return id	else semi	yes	no
VARIABLE	id	equal rpar	no	no
STATEMENT2	id	semi	no	no
STATEMENT3	lpar dot lsqbr equal	semi	no	no
MIDDLESTATE	lpar dot lsqbr equal	semi	no	no
AFTERVARIABLE	dot equal	semi	no	no
APARAMS	intlit floatlit lpar not id plus minus	rpar	yes	no
AFTERFUNCTIONCALL	dot	semi	yes	no
INDICE	lsqbr	semi mult div and dot lsqbr equal rsqbr eq neq lt gt leq geq plus minus or comma rpar	no	no
INDICELOOP	lsqbr	semi mult div and dot equal rsqbr eq neq lt gt leq geq plus minus or comma rpar	yes	no
ENDASSIGN	equal	semi	no	no

ASSIGNOP	equal	intlit floatlit lpar not id plus minus	no	no
EXPR	intlit floatlit lpar not id plus minus	semi comma rpar	no	no
TERM	intlit floatlit lpar not id plus minus	semi rsqbr eq neq lt gt leq geq plus minus or comma rpar	no	no
FACTOR	intlit floatlit lpar not id plus minus	semi mult div and rsqbr eq neq lt gt leq geq plus minus or comma rpar	no	no
RIGHTRECTERM	mult div and	semi rsqbr eq neq lt gt leq geq plus minus or comma rpar	yes	no
TYPE	integer float id	rpar lcurbr comma lpar lsqbr semi	no	no
VISIBILITY	public private	function constructor attribute	yes	no

Section 3: Design

I have a class called Parser and it takes in a path to a file. It will create the first and follow set. It will also create the hashmap used to store the table output for each terminal and nonterminal. I also store the nullable and endable non terminals. Once all my data structures have been made and populated with the information, the user can call the parse method. It will run the basic algorithm that is also seen in the table top down driven approach seen in class. If the top of the stack is a terminal and it is the same as the next token from the tokenizer, it will pop the stack and move on. If it is not the same, it will throw an error. If it is not a terminal, it will then check the output from the table based on the current token and the top of the stack. If valid, the stack will get popped and we will reverse the output from the table and insert it in the stack. Otherwise we will also throw an error.

Instead of throwing an error when invalid code, I have upgraded the code to skip the error and store it in a file. The skip error has 2 parts to it. It can pop the stack if the next token is in the follow set of our current non-terminal on top of the stack. Otherwise it will scan tokens until we get one with which we can resume the parsing. I also have some methods that help me increase readability for my code.

Section 4: Use of Tools

I used AtoCC in order to validate if my grammar is in LL1 format.
I used Ucalgary smlweb in order to create the parsing table and to get the first and follow sets. I tried to use the Ucalgary tool in order to fix my language, but I find that doing it by hand worked better.
I used google sheets to take the table from Ucalgary and make it in a csv file.
I used Joey Paquet grammar tool to remove some ambiguities and left recursion.
Finally, I used the built in tools from Java like arraylist, hashmap and stacks in order to store my code.