Parser Design

Tatum Alenko

February 19, 2019

Table of Contents

Contents

Section 1. Transformed Grammar Into LL(1)

Steps to correcting first and follow set conflicts

Once the transformed grammar without EBNF optional/repeating constructs were removed along with left-recursion using Dr. Paquet's `grammartool.jar`, the resulting grammar output was entered in the UCalgary Grammar Tool. The result of the analysis of the grammar is shown in the following code block:

```
FUNCHEAD has a first set conflict.
FUNCHEAD_1 is nullable with clashing first and follow sets.
MEMBERDECL has a first set conflict.
EXPR has a first set conflict.
FACTOR has a first set conflict.
FUNCTIONCALL_1 is nullable with clashing first and follow sets.
IDNEST has a first set conflict.
FUNCDECL has a first set conflict.
STATEMENT has a first set conflict.
VARIABLE_1 is nullable with clashing first and follow sets.
ARRAYSIZE has a first set conflict.
```

`FUNCHEAD` and `FUNCHEAD_1`

Left factored the `FUNCHEAD_1 id lpar FPARAMS rpar colon ...` portion and changed the `FUNCHEAD_1` to take into that they both can start by `id` with optional `sr id` before the `lpar` terminal.

Before:

```
FUNCHEAD -> FUNCHEAD_1 id lpar FPARAMS rpar colon TYPE .
FUNCHEAD -> FUNCHEAD_1 id lpar FPARAMS rpar colon void .

FUNCHEAD_1 -> id sr .
FUNCHEAD_1 -> .
```

After:

```
FUNCHEAD -> id FUNCHEAD_1 lpar FPARAMS rpar colon TYPEORVOID .

FUNCHEAD_1 -> sr id .
FUNCHEAD_1 -> .

TYPEORVOID -> TYPE .
TYPEORVOID -> void .
```

`FUNCDECL`

Left factored the `id lpar FARAMS rpar colon` portion and replaced the rule to account for `TYPEORVOID`:

Before:

```
FUNCDECL -> id lpar FPARAMS rpar colon TYPE semi .
FUNCDECL -> id lpar FPARAMS rpar colon void semi .
```

After:

```
FUNCDECL -> id lpar FPARAMS rpar colon TYPEORVOID semi .
```

ARRAYSIZE

Factored the possibly nullable second `intnum` terminal.

Before:

```
ARRAYSIZE -> lsqbr intnum rsqbr .
ARRAYSIZE -> lsqbr rsqbr .
```

After:

```
ARRAYSIZE -> lsqbr MAYBEINTNUM rsqbr .

MAYBEINTNUM -> intnum .
MAYBEINTNUM -> .
```

IDNEST

Left factored the `id` terminal.

Before:

```
IDNEST -> id IDNEST_1 dot .
IDNEST -> id lpar APARAMS rpar dot .

IDNEST_1 -> INDEX IDNEST_1 .
IDNEST_1 -> .
```

After:

```
IDNEST -> id IDNEST_1 .

IDNEST_1 -> IDNEST_2 dot .
IDNEST_1 -> lpar APARAMS rpar dot .

IDNEST_2 -> INDEX IDNEST_2 .
IDNEST_2 -> .
```

EXPR

Left factored the common `ARITHEXPR` in both `RELEXPR` and `ARITHEXPR`

Before:

```
EXPR -> ARITHEXPR .
EXPR -> RELEXPR .

ARITHEXPR -> TERM ARITHEXPR_1 .

ARITHEXPR_1 -> ADDOP TERM ARITHEXPR_1 .
ARITHEXPR_1 -> .

RELEXPR -> ARITHEXPR RELOP ARITHEXPR .
```

After:

```
EXPR -> ARITHEXPR EXPR_1 .

EXPR_1 -> RELOP ARITHEXPR .
EXPR_1 -> .

ARITHEXPR -> TERM ARITHEXPR_1 .
```

```
ARITHEXPR_1 -> ADDOP TERM ARITHEXPR_1 .
ARITHEXPR_1 -> .

RELEXPR -> ARITHEXPR RELOP ARITHEXPR .
```

MEMBERDECL

The first step was to factor the FUNCDECL to take care of TYPE or void. This was easily done by eliminating the two productions into one by using a seperate production TYPEORVOID. Afterwards, the id shared by both FUNCDECL and possibly VARDECL was slightly trickier because only when the TYPE of VARDECL uses the id terminal this would clash. To eliminate the clash, the id needed to be left factored out of both FUNCDECL and VARDECL. This was done with the help of INTEGERORFLOAT to take into account the case when id isn't the first terminal of VARDECL. This left FUNCDECL unsued now, so it was removed and the new production was renamed to FUNCDECLIDLESS to signifie that it used to be the old definition of FUNCDECL but is missing the first id token that was prepended.

After:

```
MEMBERDECL -> FUNCDECL .
MEMBERDECL -> VARDECL .

FUNCDECL -> id lpar FPARAMS rpar colon TYPE semi .
FUNCDECL -> id lpar FPARAMS rpar colon void semi .

VARDECL -> TYPE id VARDECL_1 semi .

VARDECL_1 -> ARRAYSIZE VARDECL_1 .
VARDECL_1 -> .
```

Before:

```
MEMBERDECL -> id MEMBERDECL_1 .
MEMBERDECL -> INTEGERORFLOAT VARDECL_2 .

MEMBERDECL_1 -> VARDECL_1 .
MEMBERDECL_1 -> FUNCDECLIDLESS .

FUNCDECLIDLESS -> lpar FPARAMS rpar colon TYPEORVOID semi .

VARDECL -> INTEGERORFLOAT VARDECL_1 .
VARDECL -> id VARDECL_1 .

VARDECL_1 -> id VARDECL_2 semi .

VARDECL_2 -> ARRAYSIZE VARDECL_2 .
VARDECL_2 -> .
```

FUNCTIONCALL_1

The removal of the first and follow set clash for this one was tricky. At first, the clash was with respect to the id terminal. This was fixed by left factoring it. However, another first and follow set clash remained afterwards with respect to the lpar terminal. This was corrected by factoring the IDNEST components that were causing the issue by further decomposing it into IDNEST_2 and IDNEST_3. This way, the conflicting lpar would not occur from a possibly null terminal.

Before:

```
FUNCTIONCALL -> FUNCTIONCALL_1 id lpar APARAMS rpar .

FUNCTIONCALL_1 -> IDNEST FUNCTIONCALL_1 .
FUNCTIONCALL_1 -> .

IDNEST -> id IDNEST_1 dot .
```

```
IDNEST -> id lpar APARAMS rpar dot .

IDNEST_1 -> INDEX IDNEST_1 .
IDNEST_1 -> .
```

After:

```
FUNCTIONCALL -> id FUNCTIONCALL_1 .

FUNCTIONCALL_1 -> IDNEST_1 FUNCTIONCALL_2 .

FUNCTIONCALL_2 -> dot id IDNEST_3 .
FUNCTIONCALL_2 -> .

IDNEST -> id IDNEST_1 dot .

IDNEST_1 -> IDNEST_2 .
IDNEST_1 -> IDNEST_3 .

IDNEST_2 -> INDEX IDNEST_2 .
IDNEST_2 -> .

IDNEST_3 -> lpar APARAMS rpar .
```

`VARIABLE_1`

This first and follow set conflict required to factor the `id` terminal in `VARIABLE` that occured inside `IDNEST` (which was replaced by it's `IDNEST_1 id` form).

Before:

```
VARIABLE -> VARIABLE_1 id VARIABLE_2 .

VARIABLE_1 -> IDNEST VARIABLE_1 .
VARIABLE_1 -> .

VARIABLE_2 -> INDEX VARIABLE_2 .
VARIABLE_2 -> .
```

After:

```
VARIABLE -> id VARIABLE_1 VARIABLE_2 .

VARIABLE_1 -> IDNEST_1 dot VARIABLE_3 .

VARIABLE_2 -> INDEX VARIABLE_2 .
VARIABLE_2 -> .

VARIABLE_3 -> IDNEST VARIABLE_3 .
VARIABLE_3 -> .
```

`FACTOR`

The first conflict for this was easy to fix with a simple left factorization of the `id` terminal shared between `VARIABLE` and `FUNCTIONCALL`. However, the new resulting production of this factorization introduced yet another first set conflict with respect to the `lsqbr` terminal. After fixing this one, yet another came along with regards to `id`, and another one after with regards to `IDNEST_3`, all of which were fixed using left factorization.

Before:

```
FACTOR -> VARIABLE .
FACTOR -> FUNCTIONCALL .
FACTOR -> intnum .
```

```
FACTOR -> floatnum .
FACTOR -> lpar ARITHEXPR rpar .
FACTOR -> not FACTOR .
FACTOR -> SIGN FACTOR .

VARIABLE -> id VARIABLE_1 VARIABLE_2 .

VARIABLE_1 -> IDNEST_1 dot VARIABLE_3 .

VARIABLE_2 -> INDEX VARIABLE_2 .
VARIABLE_2 -> .

VARIABLE_3 -> IDNEST VARIABLE_3 .
VARIABLE_3 -> .

FUNCTIONCALL -> id FUNCTIONCALL_1 .

FUNCTIONCALL_1 -> IDNEST_1 FUNCTIONCALL_2 .

FUNCTIONCALL_2 -> dot id IDNEST_3 .
FUNCTIONCALL_2 -> .
```

After:

```
FACTOR -> id FACTOR_1 .
FACTOR -> intnum .
FACTOR -> floatnum .
FACTOR -> lpar ARITHEXPR rpar .
FACTOR -> not FACTOR .
FACTOR -> SIGN FACTOR .

FACTOR_1 -> IDNEST_1 FACTOR_2 .

FACTOR_2 -> dot FACTOR_3 .
FACTOR_2 -> .

FACTOR_3 -> id FACTOR_4 .
FACTOR_3 -> .

FACTOR_4 -> IDNEST_2 dot VARIABLE_3 .
FACTOR_4 -> IDNEST_3 FACTOR_5 .

FACTOR_5 -> dot VARIABLE_3 .
FACTOR_5 -> .

VARIABLE -> id VARIABLE_1 VARIABLE_2 .

VARIABLE_1 -> IDNEST_1 VARIABLE_4 .

VARIABLE_2 -> INDEX VARIABLE_2 .
VARIABLE_2 -> .

VARIABLE_3 -> IDNEST VARIABLE_3 .
VARIABLE_3 -> .

VARIABLE_4 -> dot VARIABLE_3 .

FUNCTIONCALL -> id FUNCTIONCALL_1 .

FUNCTIONCALL_1 -> IDNEST_1 FUNCTIONCALL_2 .

FUNCTIONCALL_2 -> dot id IDNEST_3 .
FUNCTIONCALL_2 -> .
```

`STATEMENT`

The initial conflict with this rule was `id` (corrected after left factoring and creating new production `STATEMENT_1`), which was factored out that involved the `ASSIGNSTAT` and `FUNCTIONCALL` variables. However, multiple new first set

conflicts arose, such as: `IDNEST_1`, `dot`, and `id`, and `IDNEST_3` as made evident by the new productions `STATEMENT_2`, `STATEMENT_3`, `STATEMENT_4`, and `STATEMENT_5`, respectively. Because of the need for decomposing the rule so much, it ended up that the `FUNCTIONCALL`, `FUNCTIONCALL_1`, and `FUNCTIONCALL_2` productions were not needed anymore.

Before:

```
STATEMENT -> ASSIGNSTAT 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 .
STATEMENT -> FUNCTIONCALL semi .

ASSIGNSTAT -> VARIABLE ASSIGNOP EXPR .
```

After:

```
STATEMENT -> id STATEMENT_1 .
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 .

STATEMENT_1 -> IDNEST_1 STATEMENT_2 .

STATEMENT_2 -> dot STATEMENT_3 .
STATEMENT_2 -> semi .

STATEMENT_3 -> id STATEMENT_4 .
STATEMENT_3 -> VARIABLE_2 ASSIGNSTAT_1 semi .

STATEMENT_4 -> IDNEST_2 dot VARIABLE_3 VARIABLE_2 ASSIGNSTAT_1 semi .
STATEMENT_4 -> IDNEST_3 STATEMENT_5 .

STATEMENT_5 -> dot VARIABLE_3 VARIABLE_2 ASSIGNSTAT_1 semi .
STATEMENT_5 -> semi .

ASSIGNSTAT -> VARIABLE ASSIGNSTAT_1 .

ASSIGNSTAT_1 -> ASSIGNOP EXPR .
```

Transformed grammar after removal of EBNF optional/repeating notation and left-recursion

```
START -> PROG .

PROG -> PROG_1 PROG_2 main FUNCBODY .

PROG_1 -> CLASSDECL PROG_1 .
PROG_1 -> .

PROG_2 -> FUNCDEF PROG_2 .
PROG_2 -> .

CLASSDECL -> class id CLASSDECL_1 lcurbr CLASSDECL_2 rcurbr semi .

CLASSDECL_1 -> inherits id CLASSDECL_3 .
CLASSDECL_1 -> .

CLASSDECL_2 -> VISIBILITY MEMBERDECL CLASSDECL_2 .
CLASSDECL_2 -> .

CLASSDECL_3 -> comma id CLASSDECL_3 .
CLASSDECL_3 -> .
```

```
FUNCDEF -> FUNCHEAD FUNCBODY semi .

FUNCHEAD -> FUNCHEAD_1 id lpar FPARAMS rpar colon TYPE .
FUNCHEAD -> FUNCHEAD_1 id lpar FPARAMS rpar colon void .

FUNCHEAD_1 -> id sr .
FUNCHEAD_1 -> .

MEMBERDECL -> FUNCDECL .
MEMBERDECL -> VARDECL .

FUNCDECL -> id lpar FPARAMS rpar colon TYPE semi .
FUNCDECL -> id lpar FPARAMS rpar colon void semi .

VARDECL -> TYPE id VARDECL_1 semi .

VARDECL_1 -> ARRAYSIZE VARDECL_1 .
VARDECL_1 -> .

EXPR -> ARITHEXPR .
EXPR -> RELEXPR .

ARITHEXPR -> TERM ARITHEXPR_1 .

ARITHEXPR_1 -> ADDOP TERM ARITHEXPR_1 .
ARITHEXPR_1 -> .

RELEXPR -> ARITHEXPR RELOP ARITHEXPR .

FACTOR -> VARIABLE .
FACTOR -> FUNCTIONCALL .
FACTOR -> intnum .
FACTOR -> floatnum .
FACTOR -> lpar ARITHEXPR rpar .
FACTOR -> not FACTOR .
FACTOR -> SIGN FACTOR .

FUNCBODY -> FUNCBODY_1 do FUNCBODY_2 end .

FUNCBODY_1 -> local FUNCBODY_3 .
FUNCBODY_1 -> .

FUNCBODY_2 -> STATEMENT FUNCBODY_2 .
FUNCBODY_2 -> .

FPARAMS -> TYPE id FPARAMS_1 FPARAMS_2 .
FPARAMS -> .

FPARAMSTAIL -> comma TYPE id FPARAMSTAIL_1 .

FPARAMSTAIL_1 -> ARRAYSIZE FPARAMSTAIL_1 .
FPARAMSTAIL_1 -> .

FPARAMS_1 -> ARRAYSIZE FPARAMS_1 .
FPARAMS_1 -> .

FPARAMS_2 -> FPARAMSTAIL FPARAMS_2 .
FPARAMS_2 -> .

FUNCBODY_3 -> VARDECL FUNCBODY_3 .
FUNCBODY_3 -> .

FUNCTIONCALL -> FUNCTIONCALL_1 id lpar APARAMS rpar .

FUNCTIONCALL_1 -> IDNEST FUNCTIONCALL_1 .
FUNCTIONCALL_1 -> .

APARAMS -> EXPR APARAMS_1 .
APARAMS -> .
```

```
APARAMS_1 -> APARAMSTAIL APARAMS_1 .
APARAMS_1 -> .

APARAMSTAIL -> comma EXPR .

IDNEST -> id IDNEST_1 dot .
IDNEST -> id lpar APARAMS rpar dot .

IDNEST_1 -> INDEX IDNEST_1 .
IDNEST_1 -> .

INDEX -> lsqbr ARITHEXPR rsqbr .

STATBLOCK -> do STATBLOCK_1 end .
STATBLOCK -> STATEMENT .
STATBLOCK -> .

STATBLOCK_1 -> STATEMENT STATBLOCK_1 .
STATBLOCK_1 -> .

STATEMENT -> ASSIGNSTAT 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 .
STATEMENT -> FUNCTIONCALL semi .

ASSIGNSTAT -> VARIABLE ASSIGNOP EXPR .

TERM -> FACTOR TERM_1 .

TERM_1 -> MULTOP FACTOR TERM_1 .
TERM_1 -> .

VARIABLE -> VARIABLE_1 id VARIABLE_2 .

VARIABLE_1 -> IDNEST VARIABLE_1 .
VARIABLE_1 -> .

VARIABLE_2 -> INDEX VARIABLE_2 .
VARIABLE_2 -> .

ASSIGNOP -> equal .

VISIBILITY -> public .
VISIBILITY -> private .

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

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

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

ARRAYSIZE -> lsqbr intnum rsqbr .
ARRAYSIZE -> lsqbr rsqbr .

SIGN -> plus .
SIGN -> minus .

MULTOP -> mult .
MULTOP -> div .
```

```
MULTOP -> and .
```

Ambiguity-free grammar

```
START -> PROG .

PROG -> PROG_1 PROG_2 main FUNCBODY .

PROG_1 -> CLASSDECL PROG_1 .
PROG_1 -> .

PROG_2 -> FUNCDEF PROG_2 .
PROG_2 -> .

CLASSDECL -> class id CLASSDECL_1 lcurbr CLASSDECL_2 rcurbr semi .

CLASSDECL_1 -> inherits id CLASSDECL_3 .
CLASSDECL_1 -> .

CLASSDECL_2 -> VISIBILITY MEMBERDECL CLASSDECL_2 .
CLASSDECL_2 -> .

CLASSDECL_3 -> comma id CLASSDECL_3 .
CLASSDECL_3 -> .

FUNCDEF -> FUNCHEAD FUNCBODY semi .

FUNCHEAD -> id FUNCHEAD_1 lpar FPARAMS rpar colon TYPEORVOID .

FUNCHEAD_1 -> sr id .
FUNCHEAD_1 -> .

TYPEORVOID -> TYPE .
TYPEORVOID -> void .

MEMBERDECL -> id MEMBERDECL_1 .
MEMBERDECL -> INTEGERORFLOAT VARDECL_2 .

MEMBERDECL_1 -> VARDECL_1 .
MEMBERDECL_1 -> FUNCDECLIDLESS .

VARDECL -> INTEGERORFLOAT VARDECL_1 .
VARDECL -> id VARDECL_1 .

VARDECL_1 -> id VARDECL_2 semi .

VARDECL_2 -> ARRAYSIZE VARDECL_2 .
VARDECL_2 -> .

FUNCDECLIDLESS -> lpar FPARAMS rpar colon TYPEORVOID semi .

EXPR -> ARITHEXPR EXPR_1 .

EXPR_1 -> RELOP ARITHEXPR .
EXPR_1 -> .

ARITHEXPR -> TERM ARITHEXPR_1 .

ARITHEXPR_1 -> ADDOP TERM ARITHEXPR_1 .
ARITHEXPR_1 -> .

RELEXPR -> ARITHEXPR RELOP ARITHEXPR .

FACTOR -> id FACTOR_1 .
FACTOR -> intnum .
FACTOR -> floatnum .
FACTOR -> lpar ARITHEXPR rpar .
FACTOR -> not FACTOR .
FACTOR -> SIGN FACTOR .
```

```
FACTOR_1 -> IDNEST_1 FACTOR_2 .

FACTOR_2 -> dot FACTOR_3 .
FACTOR_2 -> .

FACTOR_3 -> id FACTOR_4 .
FACTOR_3 -> .

FACTOR_4 -> IDNEST_2 dot VARIABLE_3 .
FACTOR_4 -> IDNEST_3 FACTOR_5 .

FACTOR_5 -> dot VARIABLE_3 .
FACTOR_5 -> .

FUNCBODY -> FUNCBODY_1 do FUNCBODY_2 end .

FUNCBODY_1 -> local FUNCBODY_3 .
FUNCBODY_1 -> .

FUNCBODY_2 -> STATEMENT FUNCBODY_2 .
FUNCBODY_2 -> .

FPARAMS -> TYPE id FPARAMS_1 FPARAMS_2 .
FPARAMS -> .

FPARAMSTAIL -> comma TYPE id FPARAMSTAIL_1 .

FPARAMSTAIL_1 -> ARRAYSIZE FPARAMSTAIL_1 .
FPARAMSTAIL_1 -> .

FPARAMS_1 -> ARRAYSIZE FPARAMS_1 .
FPARAMS_1 -> .

FPARAMS_2 -> FPARAMSTAIL FPARAMS_2 .
FPARAMS_2 -> .

FUNCBODY_3 -> VARDECL FUNCBODY_3 .
FUNCBODY_3 -> .

APARAMS -> EXPR APARAMS_1 .
APARAMS -> .

APARAMS_1 -> APARAMSTAIL APARAMS_1 .
APARAMS_1 -> .

APARAMSTAIL -> comma EXPR .

INDEX -> lsqbr ARITHEXPR rsqbr .

STATBLOCK -> do STATBLOCK_1 end .
STATBLOCK -> STATEMENT .
STATBLOCK -> .

STATBLOCK_1 -> STATEMENT STATBLOCK_1 .
STATBLOCK_1 -> .

STATEMENT -> id STATEMENT_1 .
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 .

STATEMENT_1 -> IDNEST_1 STATEMENT_2 .

STATEMENT_2 -> dot STATEMENT_3 .
STATEMENT_2 -> semi .

STATEMENT_3 -> id STATEMENT_4 .
STATEMENT_3 -> VARIABLE_2 ASSIGNSTAT semi .
```

```
STATEMENT_4 -> IDNEST_2 dot VARIABLE_3 VARIABLE_2 ASSIGNSTAT semi .
STATEMENT_4 -> IDNEST_3 STATEMENT_5 .

STATEMENT_5 -> dot VARIABLE_3 VARIABLE_2 ASSIGNSTAT semi .
STATEMENT_5 -> semi .

ASSIGNSTAT -> ASSIGNOP EXPR .

TERM -> FACTOR TERM_1 .

TERM_1 -> MULTOP FACTOR TERM_1 .
TERM_1 -> .

VARIABLE -> id VARIABLE_1 VARIABLE_2 .

VARIABLE_1 -> IDNEST_1 VARIABLE_4 .

VARIABLE_2 -> INDEX VARIABLE_2 .
VARIABLE_2 -> .

VARIABLE_3 -> IDNEST VARIABLE_3 .
VARIABLE_3 -> .

VARIABLE_4 -> dot VARIABLE_3 .

IDNEST -> id IDNEST_1 dot .

IDNEST_1 -> IDNEST_2 .
IDNEST_1 -> IDNEST_3 .

IDNEST_2 -> INDEX IDNEST_2 .
IDNEST_2 -> .

IDNEST_3 -> lpar APARAMS rpar .

ASSIGNOP -> equal .

VISIBILITY -> public .
VISIBILITY -> private .

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

INTEGERORFLOAT -> integer .
INTEGERORFLOAT -> float .

TYPE -> INTEGERORFLOAT .
TYPE -> id .

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

ARRAYSIZE -> lsqbr MAYBEINTNUM rsqbr .

MAYBEINTNUM -> intnum .
MAYBEINTNUM -> .

SIGN -> plus .
SIGN -> minus .

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

Section 2. First and Follow Sets

The first sets generated are shown in the following code block:

```
Start: { id class main }
Prog: { id class main }
Prog1: { class \epsilon }
Prog2: { id \epsilon }
ClassDecl: { class }
ClassDecl1: { inherits \epsilon }
ClassDecl2: { public private \epsilon }
ClassDecl3: { , \epsilon }
Visibility: { public private }
MemberDecl: { id integer float }
MemberDecl1: { id ( }
FuncHead: { id }
FuncHead1: { :: \epsilon }
FuncDef: { id }
FuncDeclIdless: { ( }
FuncBody: { do local }
FuncBody1: { local \epsilon }
FuncBody2: { id if while read write return \epsilon }
FuncBody3: { id integer float \epsilon }
VarDecl: { id integer float }
VarDecl1: { id }
VarDecl2: { [ \epsilon }
Stat: { id if while read write return }
Stat1: { = ; . ( [ }
Stat2: { ; . }
Stat3: { id = [ }
Stat4: { . ( [ }
Stat5: { ; . }
Stat6: { = ; . }
AssignStat: { = }
StatBlock: { id if while do read write return \epsilon }
StatBlock1: { id if while read write return \epsilon }
Expr: { intnum floatnum id + - ( not }
Expr1: { < > == <= >= <> \epsilon }
RelExpr: { intnum floatnum id + - ( not }
ArithExpr: { intnum floatnum id + - ( not }
ArithExpr1: { + - or \epsilon }
Sign: { + - }
Term: { intnum floatnum id + - ( not }
Term1: { * / and \epsilon }
Factor: { intnum floatnum id + - ( not }
Factor1: { . ( [ \epsilon }
Factor2: { . \epsilon }
Factor3: { id \epsilon }
Factor4: { . ( [ }
Factor5: { . \epsilon }
Var: { id }
Var1: { . ( [ }
Var2: { [ \epsilon }
Var3: { id \epsilon }
Var4: { . }
IdNest: { id }
IdNest1: { ( [ \epsilon }
IdNest2: { [ \epsilon }
IdNest3: { ( }
Index: { [ }
ArraySize: { [ }
Type: { id integer float }
TypeOrVoid: { id integer float void }
FParams: { id integer float \epsilon }
FParams1: { [ \epsilon }
FParams2: { , \epsilon }
AParams: { intnum floatnum id + - ( not \epsilon }
AParams1: { , \epsilon }
FParamsTail: { , }
FParamsTail1: { [ \epsilon }
AParamsTail: { , }
```

```
AssignOp: { = }
RelOp: { < > == <= >= <> }
AddOp: { + - or }
MultOp: { * / and }
IntegerOrFloat: { integer float }
MaybeIntNum: { intnum \epsilon }
```

The follow sets generated are shown in the following code block:

```
Start: { $ }
Prog: { $ }
Prog1: { id main }
Prog2: { main }
ClassDecl: { id class main }
ClassDecl1: { { }
ClassDecl2: { } }
ClassDecl3: { { }
Visibility: { id integer float }
MemberDecl: { } public private }
MemberDecl1: { } public private }
FuncHead: { do local }
FuncHead1: { ( }
FuncDef: { id main }
FuncDeclIdless: { } public private }
FuncBody: { ; $ }
FuncBody1: { do }
FuncBody2: { end }
FuncBody3: { do }
VarDecl: { id integer float do }
VarDecl1: { id } integer float do public private }
VarDecl2: { ; } public private }
Stat: { id ; if else while end read write return }
Stat1: { id ; if else while end read write return }
Stat2: { id ; if else while end read write return }
Stat3: { id ; if else while end read write return }
Stat4: { id ; if else while end read write return }
Stat5: { id ; if else while end read write return }
Stat6: { id ; if else while end read write return }
AssignStat: { ; }
StatBlock: { ; else }
StatBlock1: { end }
Expr: { ; , ) }
Expr1: { ; , ) }
RelExpr: { ) }
ArithExpr: { < > ; , ) ] == <= >= <> }
ArithExpr1: { < > ; , ) ] == <= >= <> }
Sign: { intnum floatnum id + - ( not }
Term: { + - < > ; , ) ] == <= >= <> or }
Term1: { + - < > ; , ) ] == <= >= <> or }
Factor: { + - * < > / ; , ) ] == <= >= <> or and }
Factor1: { + - * < > / ; , ) ] == <= >= <> or and }
Factor2: { + - * < > / ; , ) ] == <= >= <> or and }
Factor3: { + - * < > / ; , ) ] == <= >= <> or and }
Factor4: { + - * < > / ; , ) ] == <= >= <> or and }
Factor5: { + - * < > / ; , ) ] == <= >= <> or and }
Var: { ) }
Var1: { ) [ }
Var2: { = ) }
Var3: { = + - * < > / ; , ) [ ] == <= >= <> or and }
Var4: { ) [ }
IdNest: { id = + - * < > / ; , ) [ ] == <= >= <> or and }
IdNest1: { = + - * < > / ; , . ) ] == <= >= <> or and }
IdNest2: { = + - * < > / ; , . ) ] == <= >= <> or and }
IdNest3: { = + - * < > / ; , . ) ] == <= >= <> or and }
Index: { = + - * < > / ; , . ) [ ] == <= >= <> or and }
ArraySize: { ; , ) [ } public private }
Type: { id ; do local }
TypeOrVoid: { ; do local }
FParams: { ) }
FParams1: { , ) }
```

```
FParams2: { ) }
AParams: { ) }
AParams1: { ) }
FParamsTail: { , ) }
FParamsTail1: { , ) }
AParamsTail: { , ) }
AssignOp: { intnum floatnum id + - ( not }
RelOp: { intnum floatnum id + - ( not }
AddOp: { intnum floatnum id + - ( not }
MultOp: { intnum floatnum id + - ( not }
IntegerOrFloat: { id ; [ } do public private local }
MaybeIntNum: { ] }
```

Section 3. Design

The overall strategy to designing the parser involved a few steps:

- Implement a first and follow set generator based on a text file containing a DSL based grammar syntax
- Using the generated first and follow sets, implement a paser table generator
- Using the generated table parser, implement a table-driven parser capable of validating a given input file containing source code of the language defined by the grammar and generate a derivation table and list of syntax errors that result
- Add semantic action identifiers to the transformed grammar to direct the parser to perform abstract syntax tree construction steps

The error recovery strategy used was panic mode. Once an empty table cell is found, the implementation attempts to skip the tokens at fault until it finds one that resyncs with the current derivation and continues on parsing the next expressions. This was chosen for its simplicity, even though it does not always identify the true source of the problem in all cases.

Unfortunately, due to a lack time, the AST generation was not complete.

Section 4. Use of Tools

The two main tools used for this assignment were Dr. Paquet's `grammartool.jar` and UCalgary's Grammar Tool. The `grammartool.jar` was used to quickly remove EBNF repeating/optional notations and left-recursion in the original grammar provided and output the resulting grammar in the notation compatible with the UCalgary Grammar Tool. By having a EBNF notation free grammar without left-recursion, the next step was to use the UCalgary Grammar Tool to iteratively attempt to solve all the ambiguities (as described in Section 1).

Other tools that were used were GraphViz when generating the AST diagram.