COMP6421 Assignment 2

Current Version: 0.2
Date: Feb. 28, 2012
Author: Yuan Tao
Instructor: Dr. Joey Paquet

Concordia University

Table of Contents

1. Grammar modifications

The requirements for assignment 2 please see:
http://newton.cs.concordia.ca/~paquet/wiki/index.php/Image:Assignment2.COMP442.paquet.2012.4.pdf

1.1. Replace all XXX* notations with XXXList

```
prog -> classDeclList progBody
classDeclList -> classDecl classDeclList | EPSILON
classDecl   ->   class id { varDeclList funcDefList } ;
varDeclList -> varDecl varDeclList   | EPSILON
funcDefList -> funcDef funcDefList   | EPSILON
progBody   ->   program funcBody ; funcDefList
funcHead   ->   type id ( fParams )
funcDef   ->   funcHead funcBody ;
funcBody   ->   { varDeclList statementList }
statementList -> statement statementList   | EPSILON
varDecl   ->   type id arraySizeList ;
arraySizeList -> arraySize arraySizeList   | EPSILON
statement   ->   variable assignOp expr ;
  |   if ( expr ) then statBlock else statBlock ;
  |   while ( expr ) do statBlock ;
  |   read ( variable ) ;
  |   write ( expr ) ;
  |   return ( expr ) ;
statBlock   ->   { statementList } | statement | EPSILON
expr   -> arithExpr | arithExpr relOp arithExpr
arithExpr -> arithExpr addOp term | term
sign   ->   + | -
term -> factor multOp factor | factor
factor   -> variable
  |   idnestList id ( aParams )
  |   num
  |   ( expr )
  |   not factor
  | sign factor
  variable   ->   idnestList id indiceList
```

```
  indiceList -> indice indiceList  | EPSILON
  idnestList -> idnest idnestList  | EPSILON
  idnest  ->  id indiceList .
indice  ->  [ arithExpr ]
arraySize  ->  [ int ]
type  ->  integer | real | id
fParams  ->  type id arraySizeList fParamsTailList | EPSILON
fParamsTailList -> fParamsTail fParamsTailList  | EPSILON
aParams  ->  expr aParamsTailList | EPSILON
aParamsTailList -> aParamsTail aParamsTailList  | EPSILON
fParamsTail  ->  , type id arraySizeList
aParamsTail  -> , expr
```

Note that, in order to use AtoCC generating First & Follow Set, the angle brackets are removed.

1.2. Left recursions

Here are the left recursive grammars:

```
arithExpr -> arithExpr addOp term | term
```

It is changed to:

```
arithExpr -> term arithExprP
arithExprP -> EPSILON | addOp term arithExprP
```

1.3. Ambiguities

Here are the ambiguous grammars:

1.3.1. expr

```
expr  -> arithExpr | arithExpr relOp arithExpr
```

It is changed to:

```
expr  -> arithExpr exprP
exprP  -> EPSILON | relOp arithExpr
```

1.3.2. term

```
term -> factor multOp factor | factor
```

It is changed to:

```
term -> factor termP
termP -> EPSILON | multOp factor termP
```

1.3.3. factor

```
factor   -> variable
   |   idnestList id ( aParams )
variable   ->   idnestList id indiceList
```

It is changed to:

```
factor   -> idnestList id factorP
factorP -> indiceList | ( aParams )
```

1.3.4. idnestList & factor & factorP

```
factor   -> idnestList id factorP
idnestList -> idnest idnestList   | EPSILON
idnest   ->   id indiceList .
```

'id' is one of the FIRST set as well as one of the FOLLOW set of idnestList, we have to eliminate this type of ambiguity as well. So above grammar is changed to:

```
factor   -> id factorPP
factorPP -> factorP | indiceList . id factorPP
idnestList -> idnest idnestList   | EPSILON
```

The new grammar factorPP has ambiguity with the factorP we have changed above.

```
factorP -> indiceList | ( aParams )
```

So again, factorP and factorPP are changed to:

```
factorPP -> indiceList factorP | ( aParams )
factorP -> EPSILON | . id factorPP
```

1.3.5. idnestList & variable

variable -> idnestList id indiceList
idnestList -> idnest idnestList | EPSILON
idnest -> id indiceList .

The same as idneestList & factor, above grammar is changed to:
variable -> id indiceList variableP
variableP -> EPSILON | . id indiceList variableP
~~idnestList -> idnest idnestList | EPSILON~~
~~idnest -> id indiceList .~~

Note that idnestList and idnest are removed from the grammar, because it is no longer used.

1.3.6. varDeclList & classDecl

classDecl -> class id { varDeclList funcDefList }
varDeclList -> varDecl varDeclList | EPSILON
funcDefList -> funcDef funcDefList | EPSILON
varDecl -> type id arraySizeList ;
funcDef -> funcHead funcBody ;
funcHead -> type id (fParams)

From the grammar, the first token of varDeclList could be 'type', the first token of FOLLOW(varDeclList) could also be 'type' which is actually the first token of funcDefList.
So we have to modify the above grammar to eliminate the ambiguity.
Here is the result after modification:
classDecl -> class id { varFuncDeclList } ;
varFuncDeclList -> EPSILON | type id varFuncDeclListP
varFuncDeclListP -> arraySizeList ; varFuncDeclList | (fParams) funcBody ; funcDefList
varDeclList -> varDecl varDeclList | EPSILON
funcDefList -> funcDef funcDefList | EPSILON
varDecl -> type id arraySizeList ;
funcDef -> funcHead funcBody ;
funcHead -> type id (fParams)

1.3.7. varDeclList & funcBody

funcBody -> { varDeclList statementList }
varDeclList -> varDecl varDeclList | EPSILON
varDecl -> type id arraySizeList ;
type -> integer | real | id
statementList -> statement statementList | EPSILON
statement -> variable assignOp expr ;
| if (<expr>) then <statBlock> else <statBlock> ;
| while (<expr>) do <statBlock> ;
| read (<variable>) ;
| write (<expr>) ;
| return (<expr>) ;
variable -> id indiceList variableP

Same as classDecl, we can see that FIRST(statementList) and FOLLOW(varDeclList) have same 'id'.
Above grammar can be changed to:

funcBody -> { varStateList }
varStateList -> EPSILON | id varStateListP
| integer id arraySizeList ; varStateList
| real id arraySizeList ; varStateList
| if (expr) then statBlock else statBlock ; statementList
| while (expr) do statBlock ; statementList
| read (variable) ; statementList
| write (expr) ; statementList
| return (expr) ; statementList
varStateListP -> id arraySizeList ; varStateList | indiceList variableP assignOp expr ; statementList
~~varDeclList -> varDecl varDeclList | EPSILON~~
~~varDecl -> type id arraySizeList ;~~
type -> integer | real | id
statementList -> statement statementList | EPSILON
statement -> variable assignOp expr ;
| if (<expr>) then <statBlock> else <statBlock> ;
| while (<expr>) do <statBlock> ;
| read (<variable>) ;
| write (<expr>) ;
| return (<expr>) ;
variable -> id indiceList variable

1.4. Grammar after modification

prog -> classDeclList progBody
classDeclList -> classDecl classDeclList | EPSILON

classDecl -> class id { varFuncDeclList } ;
varFuncDeclList -> EPSILON | type id varFuncDeclListP
varFuncDeclListP -> arraySizeList ; varFuncDeclList | (fParams) funcBody ; funcDefList

funcDefList -> funcDef funcDefList | EPSILON
funcDef -> funcHead funcBody ;
funcHead -> type id (fParams)

funcBody -> { varStateList }
varStateList -> EPSILON | id varStateListP
| integer id arraySizeList ; varStateList
| real id arraySizeList ; varStateList
| if (expr) then statBlock else statBlock ; statementList
| while (expr) do statBlock ; statementList
| read (variable) ; statementList
| write (expr) ; statementList
| return (expr) ; statementList
varStateListP -> id arraySizeList ; varStateList | indiceList variableP assignOp expr ; statementList

progBody -> program funcBody ; funcDefList
statementList -> statement statementList | EPSILON
arraySizeList -> arraySize arraySizeList | EPSILON
statement -> variable assignOp expr ;
| if (expr) then statBlock else statBlock ;
| while (expr) do statBlock ;
| read (variable) ;
| write (expr) ;
| return (expr) ;
statBlock -> { statementList } | statement | EPSILON

expr -> arithExpr exprP
exprP -> EPSILON | relOp arithExpr

arithExpr -> term arithExprP
arithExprP -> EPSILON | addOp term arithExprP

sign -> + | -

term -> factor termP
termP -> EPSILON | multOp factor termP

factor -> id factorPP
| num
| (expr)
| not factor
| sign factor
factorPP -> indiceList factorP | (aParams)
factorP -> EPSILON | . id factorPP

variable -> id indiceList variableP
variableP -> EPSILON | . id indiceList variableP

indiceList -> indice indiceList | EPSILON
indice -> [arithExpr]
arraySize -> [int]
type -> integer | real | id
fParams -> type id arraySizeList fParamsTailList | EPSILON
fParamsTailList -> fParamsTail fParamsTailList | EPSILON
aParams -> expr aParamsTailList | EPSILON
aParamsTailList -> aParamsTail aParamsTailList | EPSILON
fParamsTail -> , type id arraySizeList
aParamsTail -> , expr

2. FIRST & FOLLOW sets

The FIRST and FOLLOW sets are generated by kfg Edit (v1.39), here is a copy of the results:

	FIRST	FOLLOW
prog	class, program	
classDeclList	EPSILON, class	program
classDecl	class	
varFuncDeclList	EPSILON, id, integer, real	}
varFuncDeclListP	[, ;, (	
funcDefList	EPSILON, id, integer, real	}, $
funcDef	id, integer, real	
funcHead	id, integer, real	
funcBody	{	
varStateList	EPSILON, id, integer, real, if, while read, return, write	}
varStateListP	Id, [, ., assignOp	
progBody	program	
statementList	EPSILON, if, while, read, return, write, id	}
arraySizeList	EPSILON, [	;,), \,
statement	if, while, read, return, write, id	
statBlock	EPSILON, if, while, read, return, write, id, {	else, ;
expr	(, id, num, not, +, -	
exprP	EPSILON, relOp	;,), \,
arithExpr	(, id, num, not, +, -	
arithExprP	EPSILON, addOp	;,), \,, relOp,]
sign	+, -	
term	(, id, num, not, +, -	
termP	EPSILON, multOp	;,), \,, relOp,], addOp
factor	+, -, not, num, id, (	
factorPP	[, EPSILON, ., (	;,), \,, relOp,], addOp, multOp
factorP	EPSILON, .	;,), \,, relOp,], addOp, multOp
variable	id	
variableP	EPSILON, .	assignOp,)
indiceList	EPSILON, [	., assignOp, ;,), \,, relOp,],

		addOp, multOp
indice	[	
arraySize	[	
type	id, integer, real	
fParams	EPSILON, id, integer, real	)
fParamsTailList	EPSILON, \,	)
aParams	EPSILON, (, id, num, not, +, -	)
aParamsTailList	EPSILON, \,	)
fParamsTail	\,	
aParamsTail	\,	

3. Lexical Analyzer Modifications

Here are the modifications of lexical conventions required by assignment 2:

- ‘Or’, ‘And’ and ‘Not’ are changed to ‘or’, ‘and’ and ‘not’.
- 0 is a num, not an int.
- ‘program’ is added as a new keyword.
- ‘/*’, ‘*/’ and ‘//’ are not comments any more.
- Four types of operations are defined: assignOp, relOp, AddOp, multOp.

4. Error Recovery

This program is developed based on the pseudo-code of the lecture slides in the recursive descent way.

4.1. Always matching terminal token

When calling the match function, if it does not find a matching terminal token, a new token is added automatically assuming it found the matching token. So the program will be able to continue running. Meanwhile, an error message will be printed showing that a new token is inserted.

4.2. Inserts ‘)’ or ‘;’ at the end of line

When calling the skipsErrors function which will be called at the beginning of the functions for each non-terminal symbol, if the next token is not a token in the FIRST set or the FOLLOW set (if applicable), an error message will be printed showing that the grammar expects a correct token. Then, the program will skip all the invalid tokens except the newline token or the keyword of ‘class’ or ‘program’, until it finds a valid token as the grammar required.

In this case, when the skipsErros function finds the newline token:

- Automatically inserts ‘)’ if it is in the First or Follow sets.
- Automatically inserts ‘;’ if it is in the First or Follow sets.

4.3. Skips to ‘class’ or ‘program’

In above case, when the skipsErros function finds the keyword of ‘class’ or ‘program’:

- Discards previous grammars, skips directly to the keyword of ‘class’ or ‘program’.

Let us see a simple example showing above error recovery techniques:
Source code:

```
class Foo {
    Type func(
    {}
    int func() {
        integer i[2][4]
        real def[23 3];
        def[23 3];
```

```
    };
};

program {
    real a;

    while (1) do {
    };
};
```

Note that line 7 “def[23 3]” is invalid grammar, the analyzer will skip to ‘program’ directly.

Here are the error messages:

Syntax error at line: 3, col: 5. Token: {. Expected tokens: id integer real)

Syntax error at line: 3, col: 5. Token:). Inserts a new token:)

Syntax error at line: 4, col: 5. Token: int. Expected the token: ;

Syntax error at line: 4, col: 5. Token: int. Inserts a new token: ;

Syntax error at line: 6, col: 9. Token: real. Expected tokens: [;) ,

Syntax error at line: 6, col: 9. Token: ;. Inserts a new token: ;

Syntax error at line: 6, col: 21. Token: 3. Expected the token:]

Syntax error at line: 6, col: 21. Token: 3. Inserts a new token:]

Syntax error at line: 6, col: 21. Token: 3. Expected tokens: [;) ,

Syntax error at line: 6, col: 21. Token: 3. Skips a token: 3

Syntax error at line: 6, col: 22. Token:]. Skips a token:]

Syntax error at line: 7, col: 16. Token: 3. Expected tokens: multOp ;) , relOp] addOp

Syntax error at line: 7, col: 16. Token: 3. Skips a token: 3

Syntax error at line: 7, col: 21. Token: ;. Expected tokens: assignOp)

Syntax error at line: 7, col: 21. Token: ;. Skips a token: ;

Syntax error at line: 8, col: 5. Token:). Inserts a new token:)

Syntax error at line: 8, col: 5. Token:). Expected the token: assignOp

Syntax error at line: 8, col: 5. Token:). Inserts a new token: assignOp

Syntax error at line: 8, col: 5. Token:). Expected tokens: (id num not + -

Syntax error at line: 8, col: 5. Token: }. Skips a token: }

Syntax error at line: 8, col: 6. Token: ;. Skips a token: ;

Syntax error at line: 9, col: 1. Token: }. Skips a token: }

Syntax error at line: 9, col: 2. Token: ;. Skips a token: ;

Syntax error at line: 11, col: 1. Token:program. Finding the keyword 'program' while skiping invalid tokens. Discards previous grammars.

Here are the derivation procedures:

Line: 2, Col: 5, Token: Type, Grammar: type -> id

Line: 2, Col: 14, Token: (, Grammar: fParams -> EPSILON

```
Line:   3, Col:   5, Token:   {,   Grammar: varStateList      -> EPSILON
Line:   3, Col:   6, Token:   },   Grammar: funcBody          -> { varStateList }
Line:   4, Col:   5, Token: int,   Grammar: type              -> id
Line:   4, Col:  13, Token:   (,   Grammar: fParams           -> EPSILON
Line:   4, Col:  14, Token:   ),   Grammar: funcHead          -> type id ( fParams )
Line:   5, Col:  20, Token:   ],   Grammar: arraySize         -> [ int ]
Line:   5, Col:  23, Token:   ],   Grammar: arraySize         -> [ int ]
Line:   5, Col:  23, Token:   ],   Grammar: arraySizeList     -> EPSILON
Line:   5, Col:  23, Token:   ],   Grammar: arraySizeList     -> arraySize arraySizeList
Line:   5, Col:  23, Token:   ],   Grammar: arraySizeList     -> arraySize arraySizeList
Line:   6, Col:  18, Token:  23,   Grammar: arraySize         -> [ int ]
Line:   6, Col:  22, Token:   ],   Grammar: arraySizeList     -> EPSILON
Line:   6, Col:  22, Token:   ],   Grammar: arraySizeList     -> arraySize arraySizeList
Line:   7, Col:  13, Token:  23,   Grammar: factor            -> num
Line:   7, Col:  16, Token:   3,   Grammar: termP             -> EPSILON
Line:   7, Col:  16, Token:   3,   Grammar: term              -> factor termP
Line:   7, Col:  16, Token:   3,   Grammar: arithExprP        -> EPSILON
Line:   7, Col:  16, Token:   3,   Grammar: arithExpr         -> term arithExprP
Line:   7, Col:  17, Token:   ],   Grammar: indice            -> [ arithExpr ]
Line:   7, Col:  17, Token:   ],   Grammar: indiceList        -> EPSILON
Line:   7, Col:  17, Token:   ],   Grammar: indiceList        -> indice indiceList
Line:   7, Col:  21, Token:   ;,   Grammar: variableP         -> EPSILON
Line:  12, Col:  10, Token:   a,   Grammar: arraySizeList     -> EPSILON
Line:  13, Col:  12, Token:   1,   Grammar: factor            -> num
Line:  13, Col:  12, Token:   1,   Grammar: termP             -> EPSILON
Line:  13, Col:  12, Token:   1,   Grammar: term              -> factor termP
Line:  13, Col:  12, Token:   1,   Grammar: arithExprP        -> EPSILON
Line:  13, Col:  12, Token:   1,   Grammar: arithExpr         -> term arithExprP
Line:  13, Col:  12, Token:   1,   Grammar: exprP             -> EPSILON
Line:  13, Col:  12, Token:   1,   Grammar: expr              -> arithExpr exprP
Line:  13, Col:  18, Token:   {,   Grammar: statementList     -> EPSILON
Line:  14, Col:   5, Token:   },   Grammar: statBlock         -> { statementList }
Line:  14, Col:   9, Token:   ;,   Grammar: statementList     -> EPSILON
Line:  14, Col:   9, Token:   ;,   Grammar: varStateList      -> while ( expr ) do statBlock ;
statementList
Line:  14, Col:   9, Token:   ;,   Grammar: varStateList   -> real id arraySizeList ; varStateList
Line:  15, Col:   1, Token:   },   Grammar: funcBody          -> { varStateList }
Line:  15, Col:   2, Token:   ;,   Grammar: funcDefList       -> EPSILON
Line:  15, Col:   2, Token:   ;,   Grammar: progBody          -> program funcBody ; funcDefList
```

5. How to run

1. Install JRE (v5.0 or higher)
2. Extract COMP6421Ass2.zip to $DIR
3. cd $DIR\COMP6421Ass2
4. $ java -jar COMP6421Ass2.jar

The results of derivations and error messages are stored under $DIR\output.

Appendix A: Revision History

Version	Date	Author	Remark
V0.1	Feb. 24, 2012	Yuan Tao	Draft
V0.2	Feb.28, 2012	Yuan Tao	Updated varStateList First set