Concordia University
Department of Computer Science and Software Engineering

Compiler Design (COMP 442/6421)
Winter 2019

Assignment 2, Syntactic Analyzer

Deadline:	Monday February 25th, 2019
Evaluation:	10% of final grade
Late submission:	penalty of 50% for each late working day

This assignment is about the design and implement a syntactic analyzer for the language specified by the grammar specified below.

Work to submit

Document

- **Transformed grammar into LL(1)** : Remove all the EBNF notations and replace them by right-recursive list-generating productions. Analyze the syntactical definition. List in your documentation all the ambiguities and left recursions, or any error you may have found in the grammar. Modify the productions so that the left recursions and ambiguities are removed without modifying the language. Include in your documentation the set of productions that can be parsed using the top-down predictive parsing method, i.e. an LL(1) grammar.
- **FIRST and FOLLOW sets** : Derive the FIRST and FOLLOW sets for each non-terminal in your transformed grammar.
- **Design**: Give a brief overview of the overall structure of your solution, as well as a brief description of the role of each component of your implementation.
- **Use of tools**: Identify all the tools/libraries/techniques that you have used in your analysis or implementation and justify why you have used these particular ones as opposed to others.

Implementation

- **Parser** : Implement a predictive parser (recursive descent or table-driven) for your modified set of grammar rules.
- The result of the parser should be the creation of a tree data structure representing the parse tree as identified by the parsing process. This tree will become the intermediate representation used by the two following assignments.
- **Derivation output** : Your parser should write to a file the derivation that proves that the source program can be derived from the starting symbol (exactly as in the right side of the window of the “derivation” tab of the kfgEdit tool).
- **Error reporting** : The parser should properly identify all the errors in the input program and print a meaningful message to the user for each error encountered. The error messages should be informative on the nature of the errors, as well as the location of the errors in the input file.
- **Error recovery** : The parser should implement an error recovery method that permits to report all errors present in the source code.
- **Test cases** : Write a set of source files that enable to test the parser for all syntactical structures involved in the language. Include cases testing for a variety of different errors to demonstrate the accuracy of your error reporting and recovery.
- Note that the grammar analysis/transformation process can be greatly simplified by the use of tools such as the kfgEdit AtoCC tool. See the lab resources for more information.

Grammar

The syntactical definition is using the following conventions:

- Terminals (lexical elements, or tokens) are represented in single quotes `'likeThis'`.
- Non-terminals are represented in italics *`likeThis`*.
- The empty phrase is represented by `EPSILON`.
- EBNF-style repetition notation is represented using curly brackets `{like this}`. It represents zero or more occurrence of the sentential form enclosed in the brackets.
- EBNF-style optionality notation is represented using square brackets `[like this]`. It represents zero or one occurrence of the sentential form enclosed in the brackets.
- The non-terminal `<prog>` is the starting symbol of the grammar.

Except from the EBNF constructs, the grammar is expressed using the syntax used by the kfgEdit AtoCC toolkit application.

```
prog         -> {classDecl} {funcDef} 'main' funcBody ';'
classDecl    -> 'class' 'id' [':' 'id' {',' 'id'}] '{' {varDecl} {funcDecl} '}' ';'
funcDecl     -> type 'id' '(' fParams ')' ';'
funcHead     -> type ['id' 'sr'] 'id' '(' fParams ')'
funcDef      -> funcHead funcBody ';'
funcBody     -> '{' {varDecl} {statement} '}'
varDecl      -> type 'id' {arraySize} ';'
statement    -> assignStat ';'
              | 'if'     '(' expr ')' 'then' statBlock 'else' statBlock ';'
              | 'for'    '(' type 'id' assignOp expr ';' relExpr ';' assignStat ')' statBlock ';'
              | 'read'   '(' variable ')' ';'
              | 'write'  '(' expr ')' ';'
              | 'return' '(' expr ')' ';'
assignStat   -> variable assignOp expr
statBlock    -> '{' {statement} '}' | statement | EPSILON
expr         -> arithExpr | relExpr
relExpr      -> arithExpr relOp arithExpr
arithExpr    -> arithExpr addOp term | term
sign         -> '+' | '-'
term         -> term multOp factor | factor
factor       -> variable
              | functionCall
              | 'intNum' | 'floatNum'
              | '(' arithExpr ')'
              | 'not' factor
              | sign factor
variable     -> {idnest} 'id' {indice}
functionCall -> {idnest} 'id' '(' aParams ')'
idnest       -> 'id' {indice} '.'
              | 'id' '(' aParams ')' '.'
indice       -> '[' arithExpr ']'
arraySize    -> '[' 'intNum' ']'
type         -> 'integer' | 'float' | 'id'
fParams      -> type 'id' {arraySize} {fParamsTail} | EPSILON
aParams      -> expr {aParamsTail} | EPSILON
fParamsTail  -> ',' type 'id' {arraySize}
aParamsTail  -> ',' expr
assignOp     -> '='
relOp        -> 'eq' | 'neq' | 'lt' | 'gt' | 'leq' | 'geq'
addOp        -> '+' | '-' | 'or'
multOp       -> '*' | '/' | 'and'
```

Tokens

Keywords	`main | class |`
	`if | then | else | for | read | write | return |`
	`integer | float`

Opreators	`eq (==) | neq (<>) | lt (<) | gt (>) | leq (<=) | geq (>=)`
	`+ | - | * | /`
	`not (!) | and (&&) | or (||)`
	`=`
	`sr (::)`

Punctuation	`: | , | . | ; | [ | ] | { | } | ( | )`
id	follows specification for program identifiers found in assignment#1
floatNum	follows specification for float literals found in assignment#1
intNum	follows specification for integer literals found in assignment#1

Assignment submission requirements and procedure

You have to submit your assignment before midnight on the due date using the ENCS Electronic Assignment Submission system under the category "*programming assignment 2*". The file submitted must be a **.zip** file containing:

- all your code
- a set of input files to be used for testing purpose, as well as a printout of the resulting output of the program for each input file (derivation and error reporting, as described above)
- a simple document containing the information requested above

The marking will be done in a short presentation to the marker. A schedule will be provided to you by email in the days before the due date. You will be given a short time for the presentation, so make sure that you are ready to effectively demonstrate all the elements mentioned in the "Work to submit" section above.

Evaluation criteria and grading scheme

Analysis:		
List of left recursions and ambiguities in the original grammar.	ind 2.1	1 pt
Description of method used to apply changes to the original grammar.	ind 2.2	2 pts
Correctly transformed grammar and FIRST/FOLLOW sets.	ind 2.2	2 pts
Design/implementation:		
Description/rationale of the overall structure of the solution and the roles of the individual components used in the applied solution.	ind 4.3	2 pts
Correct implementation of a top-down predictive parser following the grammar given in this handout.	ind 4.4	10 pts
Output of clear error messages (error description and location).	ind 4.4	2 pts
Output of a derivation in a separate stream or file.	ind 4.4	2 pts
Implementation of an error recovery mechanism.	ind 4.4	2 pts
Creation of a tree data structure as intermediate representation of the program	ind 4.4	7 pts
Completeness of test cases.	ind 4.4	15 pts
Use of tools:		
Description/justification of tools/libraries/techniques used in the analysis/implementation.	ind 5.2	2 pts
Successful/correct use of tools/libraries/techniques used in the analysis/implementation.	ind 5.1	3 pts
Total		**50 pts**

Example program

```
class InheritedUtility {
  integer member1;};

class Utility : InheritedUtility {
  integer var1[4][5][7][8][9][1][0];
  float var2;
  integer findMax(integer array[100]);
  integer findMin(integer array[100]);};

integer Utility::findMax(integer array[100]){
  integer maxValue;
  integer idx;
  maxValue = array[100];
  for( integer idx = 99; idx > 0; idx = idx  - 1 ){
    if(array[idx] > maxValue) then {
      maxValue = array[idx];}
    else{};
  };
  return (maxValue);};

integer Utility::findMin(integer array[100]){
  integer minValue;
  integer idx;
  minValue = array[100];
  for( integer idx = 1; idx <= 99; idx = ( idx ) + 1) {
    if(array[idx] < maxValue + 1 / 8 || idx) then {
      maxValue = array[idx];}
    else{};
  };
  return (minValue);};

float randomize(){
  float value;
  value = 100 * (2 + 3.0 / 7.0006);
  value = 1.05 + ((2.04 * 2.47) - 3.0) + 7.0006 > 1 && ! - 1;
  return (value);};

main {
  integer sample[100];
  integer idx;
  integer maxValue;
  integer minValue;
  Utility utility;
  Utility arrayUtility[2][3][6][7];
  for(integer t = 0; t<=100 ; t = t + 1) {
    read(sample[t]);
    sample[t] = (sample[t] * randomize());
  };
  maxValue = utility.findMax(sample);
  minValue = utility.findMin(sample);
  utility.var1[4][1][0][0][0][0][0] = 10;
  arrayUtility[utility.var1[1][2][3][4][5][6][idx+maxValue]][1][1][1].var2 = 2.5;
  write(maxValue);
  write(minValue);};
```