CMPSC 470 **Project 2** Due: Thursday, October 26, 2017

Syntax Checker (100 points)

Syntax checker is a program that check syntax error in programming languages. It read a source program, and check if there is any syntax error, and output error messages describing any syntax error found.

In this assignment, you should build the simplified syntax checker implemented using java. Following describes the task of your program:

- Your program should read an input foo source program using **`jflex`** for tokenizing.
- Your program should check if the input program follows the ***foo*** grammar. Your program should parse the input program using the recursive descent parser or the recursive predictive parser, discussed in class. The ***foo*** grammar will be given at the end of this document, in the form of CFG.
- If there is **no** syntax error, then your program should print "**`Success`**" on console;
  If there is **any** syntax error(s), then your program should print "**`Error: Syntax error`**" on console.

For example, let you have the `syntax-checker` java program, and the following sample input foo programs:

`test02_assign_1.foo`

```
main()
{
    a = 1;
    print a;
}
```

`test02_assign_err1.foo`

```
main()
{
    a = 1        // there is a syntax error
    print a+;    // there is a syntax error
}
```

Running `syntax-checker` program with the above foo programs should print the following outputs on console:

```
> java syntax-checker test02_assign_1.foo
Success
>
```

```
> java syntax-checker test02_assign_err1.foo
Error: Syntax error
>
```

Note that this `syntax-checker` does not check semantic errors, such as variable type, return type, or declaration of variable.

Test cases and points

Totally 45 test cases (foo programs) will be available at https://turing.cs.hbg.psu.edu/cmpsc470/proj2-testcases.zip in course website, and `/home/cmpsc470/f17/proj2-testcases.zip` in sunlab. Each foo program whose filename has "err" should have syntax error; each foo program without "err" should not have syntax error.

You will get 2 points if your `syntax-checker` program correctly identify if the input foo program has any syntax error. Totally, $2 * 45 = 90$ points will be determined using the 45 test foo programs. The grader will add 5 additional complex foo test program to check if your `syntax-checker` program works correct.

You **may lose some points** if your program does not terminate after printing "`Success`" or "`Error: Syntax error`" on console.

What to submit:

- Submit one zip file containing following files via **canvas by 11:59:59 PM, Thursday, October 26, 2017**.
- **Readme file** describing how to compile your syntax checker program and how to run it. Grader will recompile your jflex file(s).
- Your own **jflex** and **java** source files that implements the syntax checker program.

The *foo* grammar:

The following describes the ***foo*** grammar, is left-factored and does not have left-recursion.

```
1.  program       -> main_decl
2.  type_spec     -> "int" | "float"
3.  main_decl     -> "main" "(" ")" compound_stmt
4.  stmt_list     -> stmt stmt_list | epsilon
5.  stmt          -> expr_stmt | print_stmt | compound_stmt | if_stmt | while_stmt | return_stmt
6.  print_stmt    -> "print" expr ";"
7.  expr_stmt     -> ID "=" expr ";" | ";"
8.  while_stmt    -> "while" "(" bexpr ")" stmt
9.  compound_stmt-> "{" local_decls stmt_list "}"
10. local_decls   -> local_decl local_decls | epsilon
11. local_decl    -> type_spec ID ";"
12. if_stmt       -> "if" " (" bexpr ")" stmt "else" stmt
13. return_stmt   -> "return" ";" | "return" expr ";"
14. expr          -> term expr'
15. expr'         -> "+" term expr' | "-" term expr' | epsilon
16. term          -> factor term'
17. term'         -> "*" factor term' | "/" factor term' | "%" factor term' | epsilon
18. factor        -> "(" expr ")" | ID | NUM | REAL
19. bexpr         -> "true" | "false" | expr "==" expr | expr "!=" expr | expr "<" expr
                   | expr ">" expr | expr "<=" expr | expr ">=" expr
```

In the above, italicized words represent non-terminals (such as *program* and *type_spec*), *program* is the starting symbol, words or symbols enclosed by quotation marks are keywords and symbols (such as "int" and "{"), respectively, and

- *epsilon* represents the empty string,
- **NUM** represents positive integers,
- **REAL** represents positive real numbers, which is not integer. It can be `1E2`, `2.0`, `2.`, `1E+3`, `1E-4`, etc., but not `10`,
- **ID** represents C language identifiers, which start with lowercase letter or capital letter or '_'.

Comments (a string that starts with "`//`" and ends at the end of line), new line (`\n`), and whitespaces(`[ \t\r]+`) must be ignored in lexical analyzer implemented using **`jflex`**.

In the foo grammar,

- The productions 1-3 define the main function;
- The productions 4-13 define the statements, such that if-statement, while-statement, assignment-statement; The print-statement, and compound-statement (that can declare variables and other statements in between curly brackets);
- The productions 14-18 define algebraic expression;
- The production 19 defines Boolean expression.

The above grammar may be extended in next project that uses **`BYacc/J`**.