CMPSC 470 **Project 3** Due: Wednesday, November 28, 2018

Semantic Checker (200 points)

Semantic checker is a program that check semantic error in programming languages. It read a source program, and check if there is any error regarding type and scope of variables, and output error messages describing any semantic error found. In this assignment, you should build a class that maintain symbol tables to check scope of variable, and build the semantic checker using the class.

1. Chained Symbol Tables (50 points)

One way of checking the scope of variables is using the chained symbol tables. In this assignment, to build semantic checker, you should use the **`Env`** class that maintains the chained symbol tables. It has, at least, three operations as described in the class:

- **`Env(Env p)`** : a constructor;
- **`Put(String s, Object sym)`** : adding a new symbol entry into the current symbol table;
- **`Get(String s)`** : returning a symbol entry whose key is s, or **`null`** if the symbol entry does not exists.

The **`Env`** prototype class and the **`TestEnv`** class, containing 10 test cases, are provided in the startup program. You will get 5 points per each test case in **`TestEnv`** class.

2. Semantic Checker (150 points)

After completing the **`Env`** class, you should build the semantic checker implemented using java. Following describes the task of your program:

- Your program should use **`jflex`** to determine tokens from an input *mini-C* source program.
- Your program should use bottom-up parser, generated by **`BYacc/J`**. The updated *mini-C* grammar will be given at the end of this document, in the form of CFG.
- You should use the **`Env`** class to check scopes of variables, their types, and types of declared functions.
- If there is **no** semantic error, then your program should print "**`Success: no error found.`**" on console;
  If there is **any** semantic error(s), then your program should print detailed error message on console, such as "**`Error at line 11 : "int" value is tried to assign to "float" variable c.`**
  `Error: there exists error(s).`".

For example, let you have the `semantic-checker` java program, and the following sample input *mini-C* programs:

success

```
int main()
{
    int x;
    x = x + 1;
    return 0;
}
```

failure

```
int main()
{
    int x;
    x = x + 1.0;
    return 0;
}
```

Running `semantic-checker` program with the above *mini-C* programs should print the following outputs on console:

```
> java semantic-checker success03.minc
Success: no error found.
>
```

```
> java semantic-checker error03b.minc
Error at line 4 : operation of "int" * " float " is not allowed.
Error: there exists error(s).
>
```

Note that this `semantic-checker` determines only the first semantic error, such as variable type mismatch or function type mismatch or the use of non-declared variables/functions, and then prints its appropriate error message with the line number of the error location in the input program.

Start-up program, test cases, and points

The zip file containing the start-up program, the **`TestEnv`** test program, and test *mini-C* programs will be available at in course website: https://turing.cs.hbg.psu.edu/cmpsc470/proj3-startup.zip. The zip file contains the following contents:

- `src/Parser.grammar`: The grammer file shows the list of production and association rules. In the file, the rules describing "`to support record`" are about record keywords and only for the extra credits.
- `src/Lexer.flex`: The lexer file describes basid lexical analyzing rules. You must update the lexer to send token attributes to Parser.
- `src/Parser.y`: The yacc file to generate `Parser` class.
- `src/ParserImpl.java`: Empty super class of `Parser` class
- `src/Program.java`: Main program
- `src/Run.bat`: It shows the basic command to compile and run.
- `src/Env.java`: Class for symbol table
- `src/TestEnv.java`: Symbol table test program
- `sample/minc/`: The directory contains 44 test mini-C programs: 11 success cases and 33 failure cases.
- `sample/output/`: The directory contains 44 solution output messages
- `sample/extra/`: The directory contains 10 min-c programs and their corresponding outputs for extra points

Regarding the **`TestEnv`** test program, you will get 3 points from each 10 test cases: totally 30 points.

Regarding 44 test *mini-C* programs, each test *mini-C* program whose filename contains "**succ**" does not have semantic error, and whose file name contains "**fail**" has at least one semantic error, and whose file name ends with "**.output**" shows its error message. You will get 1-3 points per each test *mini-C* program from your output message, by the following rules:

- Regarding each "**succ**" file, you will get 1 point if your `semantic-checker` program correctly identifyies it as "**`Success: no error found.`**".
- Regarding each "**fail**" file, you will get 1-3 points depending on the error message printed your program:
  - 1 point if your program correctly identifies it as "**`Error: there exists error(s).`**";
  - 1 point if your program correctly identify the line having the error, such as "**`Error at line 4 :`**"; and
  - 1 point if your program correctly identify the error message, such as "**`operation of "int" * " float " is not allowed.`**"

The total 170 points will be determined from the 44 *mini-C* test programs (33 failure and 11 success cases = 110 points) and additional *mini-C* programs provided by the grader after the deadline.

The startup program contains 10 extra testcases that uses "`record`" syntax: 2 success + 8 failure cases = 26 points. With including few more testcases by grader, you may get totally 40 extra credits if you correctly identify their messages.

You **may lose some points** if your program does not terminate after printing "`Success`" or "`Error`" on console. You **will lose huge points** if you use improper function/variable names or comments, such as insulting words, or if you hardcode the outputs into your program.

What to submit:

- Submit one zip file containing following files via **canvas by 11:59:59 PM, Wednesday, November 28, 2018**.
- **Readme file** describing how to compile your semantic checker program using **`jflex`** and **`BYacc/J`** and how to run your program. Grader will recompile both your **`jflex`** `(*.flex)` and **`yacc`** `(*.y)` file(s).
- Your own **jflex** , **`yacc`**, and **java** source files that implements the semantic checker program.