Assignment 1

Due 11:59pm Oct. 2nd (Friday)

This assignment is done by a group of 2 to 3 students (each group submits only 1 copy of the assignment).

Goal :

1. Learn how to write and use makefile: http://www.delorie.com/djgpp/doc/ug/larger/makefiles.html
2. Acquaint yourself with flex and bison.

Specifications

You will extend calc.l and calc.y to parse and type check programs whose syntax is defined below.

Prog → main() {Stmts}
Stmts → **ε** | Stmt; Stmts
Stmt → int Id | float Id | Id = E | **printvar** Id
E → Integer | Float | Id | E + E | E * E
Integer → digit+
Float -> Integer . Integer

Stmts is a sequence of statements. **Id** is an identifier, which is a sequence of one or more lower-case letters or digits. **Id** should start with a lower-case letters. For example, x, x1, xy are identifiers, but 1x and A are not. Each variable is either a positive integer (int) or a positive floating point (float).

Expression E is an integer, a floating point, an identifier, or an infix arithmetic expression with operators "+" and "*" only. These two operators are left associative (e.g., 1 + 2 + 3 is equivalent to (1 + 2) + 3). "*" has higher precedence than "+".

Id = E assigns the value of an expression E to the variable Id. **printvar Id** outputs the value of Id.

If there is any **syntax error**, you are expected to interpret the program until the statement where you find the error. Also, **your error message must contain the line number where the error was found.**

Tokens may be separated by **any number of** white spaces, tabs, or new lines.

Type checking rules are given below:

Stmt -> int Id |	{Id.type = 0}
float Id |	{Id.type = 1}
Id = E	{if (Id.type \= E.type) then type error}
E → Integer |	{E.type = 0}
Float |	{E.type = 1}
Id |	{E.type = Id.type}
E1 + E2|	{if (E1.type==E2.type) then E.type = E1.type; else type error}
E1 * E2 |	{if (E1.type==E2.type) then E.type = E1.type; else type error}

If one of the aboverules is violated, your program should terminate and print "<line number>: type error". In addition, if a variable is used but is not declared, then your program should print "<line number>: <variable name> is used but is not declared".

Compile your program:

```
flex –l calc.l
bison -dv calc.y
gcc -o calc calc.tab.c lex.yy.c –lfl
```

Execution (example):.

./calc < input

Where **input** is the name of the input file

Example Programs (Note that the test cases used in grading may be different from the examples)

Program 1:

```
main() {int x; x = 3;}
```

Output:

Program 2:

```
main() {int x; x = 3; printvar x;}
```

Output:

```
3
```

Program 3:

```
main() {int x; x = 3;  x = x + 4; printvar x;}
```

Output:

```
7
```

Program 4:

```
main() {x=3;}
```

Output:

```
Line 1: x is used but is not declared
```

Program 5:

```
main() {int x; x = 1-2; }
```

Output:

```
Parsing error: line 1
```

Program 6:

```
main() {
  int 1x;
}
```

Output:

```
Parsing error: line 2
```

Program 7:

```
main() {
        int x;
        x = 3;
printvar    x;
}
```

Output:

```
3
```

Program 8:

```
main() {
        float x; x = 1.2; printvar x;
}
```

Output:

1.2

Program 9:

```
int x;
```

Output:

```
Parsing error: line 1
```

Program 10:

```
main() {int x; int y; x = 3; y = x; printvar y;};}
```

Output:

```
3
```

Program 11:

```
main() {int x; float y; x = 1;  y = x;}
```

Output: Line 1: type error

Program 12:

```
main() {int x; x = 3 + 1.5; printvar x;}
```

Output: Line 1: type error

Program 13:

```
main() {printvar 1 + 2;}
```

Output: Parsing error: Line 1

Submission guideline

- Please hand in your **source code** and a **Makefile** electronically (**please do not submit .o or executable code**). You must make sure that your code compiles and runs correctly on bingsuns.binghamton.edu. The Makefile **must** give the executable code the name **calc**
- Write a **README** file (**text file, do not submit a .doc file**) which contains
  - The name, the section number, and the email address of group members
  - Whether your code was tested on bingsuns.
  - How to execute your program.
  - Briefly describe your algorithm or anything special about your submission that the TA should take note of.
- Place all your files under one directory with a unique name (such as p1-[userid] for assignment 1, e.g. p1-pyang).
- Tar the contents of this directory using the following command.
  tar –cvf [directory_name].tar [directory_name]
  E.g. tar -cvf p1-pyang.tar p1-pyang/
- Use the Blackboard to upload the tared file you created above.

Grading guideline

- Readme, correct executable names: 4’
- Correct makefile: 8’
- lexical analysis, parsing, output: 64’
- Type checking: 24’

Academic Honesty:

All students should follow Student Academic Honesty Code (http://watson.binghamton.edu/acadhonorcode.html). All forms of cheating will be treated with utmost seriousness. You may discuss the problems with other students, however, you must write your OWN codes and solutions. Discussing solutions to the problem is NOT acceptable. Copying an assignment from another student or allowing another student to copy your work may lead to an automatic **F** for this course. If you borrow small parts of code/text from Internet, you must acknowledge this in your submission. Also, you must clearly understand and be able to explain the material. Copying entire material or large parts of such material from the Internet will be considered academic dishonesty. **Moss will be used to detect plagiarism in programming assignments.** You need ensure that your code and documentation are protected and not accessible to other students. Use **chmod 700** command to change the permissions of your working directories before you start working on the assignments. If you have any questions about whether an act of collaboration may be treated as academic dishonesty, please consult the instructor before you collaborate.