Assignment 1

Due 11:59pm Feb. 16 (Friday)

This assignment is done by a group of 2 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 programs whose syntax is defined below.

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

Stmts is a sequence of statements. **float Id** is variable declaration. A new variable gets **0.0** as its initial value.

Id is an identifier, which starts with a lower-case letters followed by a sequence of 0 or more (lower case or capital) letters or digits. For example, x, x1, xy, xY are identifiers, but 1x and Ax are not.

Expression E is a floating point (Float), an identifier, or an infix arithmetic expression with operators "-" (subtraction) and "*" (multiplication) only. These two operators are left associative (e.g., 1 - 2 - 3 is equivalent to (1 - 2) - 3). * has higher precedence than -. Your program should support **both positive and negative** floating points (e.g. 0.4, 5.0, (-6.4)).

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

If an input does not match any token, output "lexical analysis error: <input>", where <input> is the input.

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.**

Block is a sequence of statements that are grouped together using "{" and "}". This is similar to what goes on in C/C++ and Java. The block construct is used to operate on local variables that may have same names as variables used in enclosing blocks. Blocks may be nested to arbitrary depth (i.e., any statement within a block can itself be a block). Static scoping mechanism (the scoping mechanism used in C) should be used.

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

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

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

Program 1:

```
main(){float x; print x;}
```

Output: 0.0

Program 2:

```
main(){float x; x = 3.0;}
```

Output:

Program 3:

```
main(){float x; x = 3.0; print x;}
```

Output:

3.0

Program 4:

```
main(){float x; x = 4.0; x = x-3.0; print x;}
```

Output:

1.0

Program 5:

```
main(){float x; x = 2.0+1.0;}
```

Output:

Lexical analysis error: +

Program 6:

```
  float x;
```

Output:

Parsing error: line 1

Program 7:

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

Output:

Lexical analysis error: 1x

Program 8:

```
main(){
        float x;
}
```

Output:

Program 9:

```
main() {
        float x;
                x = 3.0;
print       x;
}
```

Output:

3.0

Program 10:

```
main(){
float x;
x = 1.0;
print x;
{
        float x;
        x = 3.0;
        print x;

};
print x;
}
```

Output:

1.0
3.0
1.0

Program 11:

```
main(){
        float x;
        float y;
        float z;
        x = 1.0;
        {   float x;
            x = 5.0;
            y = x-3.0;
            print y;
            {   float y;
                y = 6.0-x;
                print y;
            }; };
        z= y-x;
        print z;
}
```

Output:

2.0
1.0
1.0

Program 12:

```
main(){float x;  x = 3.0 - (-1.0); print x;}
```

output: 4.0

Program 13:

```
main() { }
```

Output:

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, please do not submit a .doc file**) which contains
  - The name and the email address of group members
  - Whether your code was tested on bingsuns.
  - How to execute your program.
  - (optional) 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 mycourses to upload the tared file you created above.

Grading guideline

- Readme (must be a text file), correct executable names: 4'
- Correct makefile (all files are compiled when typing make): 8'
- Correctness of the program: 84'

Academic Honesty:

All students should follow Student Academic Honesty Code (http://watson.binghamton.edu/acadhonor-code.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.