Assignment 1

Due 11:59pm Sept. 19 (Sunday)

This assignment is done individually or by a group of 2 students.
Each group please submits only ONE 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 studied in the class to parse programs that print the value of postfix expressions. The syntax of the program is defined below.

Stmts → **ε** | Stmt Stmts
Stmt → **print** E;
E → Float | Int | E E + | E E –
Int → digit+
Float → digit+**.**digit+

Stmts is an empty statement (i.e., **ε**) or a sequence of statements. Each statement ends with ";". **ε** is represented as white space in Bison. Each statement is of the form **print E, which** outputs the value of an expression E followed by a new line. **Expression E** can be a positive integer, a positive floating point, a postfix arithmetic expression with operators "+" (addition) and "-" (subtraction). A floating-point number is a decimal point (e.g., 2.16, 2.4598, 2.292273). **Your parser can use "%f" to print both integers and floating points (i.e., the output of your parser will be floating point(s) if the input is valid).**

A postfix expression is an expression in which the operator is placed after the operand. For example, postfix expression 1 2 + represents (1+2), postfix expression 1 2 3 + - represents 1- (2+3), and postfix expression 1 2 + 3 4 - + represents (1+2) + (3-4). More information about postfix expressions is given at: http://www.btechsmartclass.com/data_structures/postfix-evaluation.html

https://www.youtube.com/watch?v=84BsI5VJPq4

If an input character does not match any token, then your program prints **"lexical analysis error: <input>"**, where <input> is the input. If there is a **syntax error**, you are expected to interpret the program until the statement that has the syntax error. Also, your error message must contain the **line number** where the error was found. Variable yylineno in flex/bison records the line number. By adding **%option yylineno** in the flex code and **extern int yylineno;** in the bison code, you can use the variable yylineno in your bison file. You can use google to find more information about yylineno.

You can assume that there is at most one syntax error in each testcase used in grading.

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):.

Put your testcase in a file, e.g. input, then you can use the following command to execute your program to run your program.

```
./calc < input
```

Also, you can save each testcase in a separate file and test your program.

A video that shows how to test your program is posted on brightspace.

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

Program 1 (print an integer):

```
print 2;
```

Output:

```
2.000000 (or 2, 2.0, 2.00, 2.000, etc. – same for other examples)
```

Program 2 (print a floating point):

```
print 2.121;
```

Output:

```
2.121000
```

Program 3 (integer addition):

```
print 1 2 +;
```

Output:

```
3.000000
```

Program 4 (floating point addition):

```
print 1.2 2.3 +;
```

Output:

```
3.500000
```

Program 5 (subtraction):

```
print 1 3 -;
```

Output:

```
-2.000000
```

Program 6 (1 – (3+2)):

```
print 1 3 2 + -;
```

Output:

```
-4.000000
```

Program 7 (1+(3+2)):

```
print 1 3 2 + +;
```

Output:

```
6.000000
```

Program 8 ((1+3) - 2):

```
print 1 3 + 2 -;
```

Output:

```
2.000000
```

Program 9 (1+((3+4)-2)):

```
print 1 3 4 + 2 - +;
```

Output:

```
6.000000
```

Program 10 (1-(3+4)+(2-1)+4):

```
print 1 3 4 + - 2 1 - + 4 +;
```

Output:

```
-1.000000
```

Program 11 (tokens can be separated with any number of \n, \t, and space):

```
    print
    1    2
+;
```

Output:

```
3.000000
```

Program 12 (tokens can be separated with any number of \n, \t, and space):

```
print
            1;
```

Output:

```
1.000000
```

Program 13 (expressions should end with “;”):

```
 print 1
```

Output:

```
Parsing error: line 2
```

Program 14 (the expression should not support * and /):

```
print 1 2 *;
```

Output:

```
Lexical analysis error: *
Parsing error: line 1
```

Program 15 (print is missing):

```
1 2 +;
```

Output:

```
Parsing error: line 1
```

Program 16 (postfix expression syntax error):

```
print 1 2 + -;
```

Output:

```
Parsing error: line 1
```

Program 17 (multiple expressions):

```
print 1 2 +;
print 2.1 3.1 -;
print 2 1 3 - 4 + -;
```

Output:

```
3.000000
-1.000000
0.000000
```

Program 18 (syntax error in multiple expressions, the program stops after parsing error):

```
print 1 2 +;
print 2.1 3.1 - +;
print 2 1 3 - 4 + -;
```

Output:

```
3.000000
Parsing error: Line 2
```

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 remote.cs.
  - 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 brightspace.binghamton.edu to upload the tared file you created above. The ID and password should be the same as your university email account.

Grading guideline

- Readme (must be a text file), correct executable names: 4'
- Correct makefile ((all files should be compiled after typing make):: 8'
- Correctness: 88'

Academic Honesty:

All students should follow Student Academic Honesty Code (https://www.binghamton.edu/watson/about/academic-honesty.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.