Stacks - Reverse Polish Notation Calculator

Date assigned: Wednesday, September 14, 2011
Final Date due: Monday, September 26, 2011
Points: 30

Reverse Polish Notation, or Post-Fix Notation, is a method of writing mathematical expressions. In this method, the operators follow the operands. This is different from the more common In-Fix Notation where operators are placed between their operands.

Post-Fix Notation is nice because complicated expressions can be represented without parentheses and the notation lends itself very well to evaluation using a Stack data structure.

An example Post-Fix expression is shown below.

5 4 + 1 2 9 * - /

This is equivalent to: (5 + 4) / (1 - (2*9))

To evaluate this Post-Fix expression using a stack the following operations will occur:
5 4 + 1 2 9 * - /

Read 5, Push 5
Read 4, Push 4
Read +, Pop two values off the stack, apply +, and Push the result (9)
Read 1, Push 1
Read 2, Push 2
Read 9, Push 9
Read *, Pop two values off the stack, apply *, and Push the result (18)
Read -, Pop two values off the stack, apply -, and Push the result (-17)
Read /, Pop two values off the stack, apply /, and Push the result (-0.529412)
Read '\n', Pop one value off the stack for the solution. (-0.529412)

Your task is to build a calculator that will read RPN expressions from a file (one per line) and evaluate each expression, printing the final result to the screen.

Note that we will only implement the **binary** operations: +, -, /, *.

The following is an example of a possible data file:

```
1 2 +
1 2 -
1 2 1 - -
1 2 1 + -
2 2 *
2 3 2 * *
```

As output print the result of each expression followed by a newline. Use **%g** to output each floating point number. When the EOF is encountered, print 'PROCESSING COMPLETE'.

```
3
-1
0
-2
4
12
PROCESSING COMPLETE
```

Note1: Make sure you thoroughly test your program.

Note2: The following are assumptions for this assignment:

1) The expressions will be valid but may contain divide by zero errors. (see notes below).

2) Only single digit, positive integers will appear in the data file but results will be real numbers

3) Spaces and tabs may appear anywhere in the file and any combination of spaces and/or tabs may appear in succession.

4) Each line in the file will contain an expression (exception: see note about the Bonus)

As always,

1. You are to break up your program into appropriate .h/.c files and on the day the assignment is due, turn in a hard copy of your Makefile and each .h/.c combination (fully documented). You might not have a .h file for every .c file.
2. Your code is to be written in C using Eclipse 3.7. Programs written in other environments will not be graded. Name your project **cs300_stack_PUNetID**.

Submit a file **cs300_stack_PUNetID.tar.gz** into the CS300 Drop Box by 9:15am on the day in which the assignment is due.

3. Test your modules one function at a time. This will lessen your level of frustration greatly.
4. You are to use V6.0 of the coding standards.

Goals for this assignment:

1. Break your program up into well thought out modules.

2. Use well thought out functions in solving this problem. Don't break code out later into a function.

3. Code and test your program one function at a time.

4. Implement the stack data structure efficiently in solving this problem

5. Write efficient/clean code

6. Use the debugger to effectively develop a correct solution

Part I (Due Monday, September 19, 2011 by 11:59pm)

For the first part of this assignment, you must completely code and test your **stack module**. A stack must be used correctly in your calculator solution. We will discuss this in detail in class. The following stack header found in file **stk.h** will be placed on Zeus in the **CS300Public** folder. You are to use this exact header file without modification for this assignment, so copy the file from the CS300Public folder and use the exact file in your Eclipse project.

You are to write a driver called stackdriver.c that **thoroughly tests** each function of your stack module. Be sure to test each error and boundary condition. Make sure your driver outputs results to the display screen so that I can see the results of your testing.

By 5pm on the date listed in Part I above, you are to submit a tarball called **cs300_stack_PUNetID.tar.gz** that contains your Eclipse project which solves the Part I description.

You must turn in a color hard copy to the instructor no later than **Tuesday, September 20, 2011, 9am.** The hard copy can be slid under my office door.

Part II (Due Monday, September 26, 2011 by 9:15am)

You are to complete the remainder of the assignment. To do so,

1) Create a folder called testfiles in your project folder. Inside of this folder, create a data file called **testdata.txt** that contains the example below. Create a file in testfiles named **`testdata.out.expected`** that contains the expected output of running testdata.txt through your calculator.

2) Write any remaining modules to solve the above problem. Minimally, you should have at least three modules: (a) a stack module, (b) a calculator module, and (c) a driver module. Your Makefile is to produce a driver called calculator. This Makefile still needs to produce stackdriver.

testdata.txt

```
1 2 +
1 2 -
1 2 1 - -
1 2 1 + -
2 2 *
2 3 2 * *
5 4 + 1 2 9 * - /
4 2 2 - /
2 0 /
```

The ONLY way in which an expression can be invalid is to contain a divide by zero error.

Divide by zero: 4 2 2 - / or 2 0 /

When a divide by zero error is encountered, display each operand in the error message as shown below and read and throw away all the characters until the next '\n'.

```
Divide By Zero: 4 / 0

Divide By Zero: 2 / 0
```

The expected output for testdata.txt is:

Expected Output

```
3
-1
0
-2
4
12
-0.529412
Divide By Zero: 4 / 0
Divide By Zero: 2 / 0
PROCESSING COMPLETE
```

Bonus: +1.5 pts (5%)

Allow the file testdata.txt to contain comments similar to those in shell script files. That is, a **#** starts a comment and the comment goes until the next '\n'. A file may contain a line that consists only of a comment or a line with an expression followed by a comment.

```
/*****************************************************************************
 File name:  stk.h
 Author:     Computer Science, Pacific University
 Date:       9/14/11
 Class:      CS300
 Assignment: RPN Calculator
 Purpose:    To define the header file for the stack module.
*****************************************************************************/

#ifndef STK_H_
#define STK_H_

#define MAX_STACK 1024

#define TRUE 1
#define FALSE 0

#define EMPTY_STACK -1
#define NO_ERROR 0
#define ERROR_STACK_EMPTY 1
#define ERROR_STACK_FULL 2
#define ERROR_NO_STACK_CREATE 3


// User-defined datatypes for easier reading

typedef short int BOOLEAN;
typedef short int ERRORCODE;
typedef double DATATYPE;

typedef struct Stack
{
  int top;
  DATATYPE data[MAX_STACK];
} Stack;

BOOLEAN stkCreate (Stack *);
BOOLEAN stkTerminate (Stack *);
BOOLEAN stkIsFull (Stack);
BOOLEAN stkIsEmpty (Stack);
BOOLEAN stkPush (Stack *, DATATYPE);
BOOLEAN stkPop (Stack *, DATATYPE *);
BOOLEAN stkPeek (Stack, DATATYPE *);

#endif
```

Here is what my Makefile looks like. From this Makefile you can see that I have created three modules to solve this problem. You need at least three modules in your solution. If you find a need for more, feel free to create more. This Makefile will also be placed in CS300Public on zeus.

Note that this Makefile expects you to create a file `testfiles/testdata.out.expected` which contains the expected output from running testdata.txt through your calculator.

```
CC=gcc
CFLAGS=-g -Wall
TARGETS=calculator stackDriver

.PHONY: all clean dist testMe

all: ${TARGETS}

stackDriver: bin/stackDriver.o bin/stk.o
	${CC} ${CFLAGS} -o stackDriver bin/stackDriver.o bin/stk.o

bin/stackDriver.o: src/stackDriver.c include/stk.h
	${CC} ${CFLAGS} -o bin/stackDriver.o -c src/stackDriver.c

calculator: bin/calcDriver.o bin/stk.o bin/calcEvaluator.o
	${CC} ${CFLAGS} -o calculator bin/calcDriver.o bin/stk.o \
	                        bin/calcEvaluator.o

bin/calcDriver.o: src/calcDriver.c include/stk.h include/calcEvaluator.h
	${CC} ${CFLAGS} -o bin/calcDriver.o -c src/calcDriver.c

bin/calcEvaluator.o: src/calcEvaluator.c include/stk.h
	${CC} ${CFLAGS} -o bin/calcEvaluator.o -c src/calcEvaluator.c

bin/stk.o: src/stk.c include/stk.h
	${CC} ${CFLAGS} -o bin/stk.o -c src/stk.c

clean:
	rm ${TARGETS} bin/*.o

dist:  clean
	tar czf ../cs300_stack_PUNetID.tar.gz ../cs300_stack_PUNetID


testMe: calculator
	./calculator testfiles/testdata.txt > testfiles/testdata.out.actual
	diff testfiles/testdata.out.actual testfiles/testdata.out.expected
```