CS-212: LAB 6 – Stacks

Learning Objective:

- Using pointers to create a stack.

Lab Objective:

For this lab, you will have an input file that is filled with math problems (Infix form). Each line will be one match problem. Your lab will use a stack to convert each math problem to post fix form.

There is an extra credit for this lab that is worth 20%. The extra credit will evaluate the post expressions.

Lab 6 Setup:

1. Open a new terminal window to SSH into your account on the [bingweb.binghamton.com] server. When logged in, make sure you are in your home directory by typing the following command if you are not already there. The "~" is just a shortcut symbol that means home directory:
   cd ~

2. Change your current directory to the "CS212" directory by typing:
   cd CS212

3. Confirm that you are in the "CS212" directory by typing:
   pwd

4. Create a directory for Lab6.
   mkdir Lab6

5. Move into the new directory
   cd Lab6

6. Create a new c file for you to write your lab.
   > <Last Name><FirstInitial>_Lab6.c
   > <Last Name><FirstInitial>_Lab6_Functions.
   > <Last Name><FirstInitial>_Lab6.h

WRITING THE LAB:

First Structure:

In your header file (using typedef) create a structure called recordType

In this structure you will declare a character called symbol.

NOTE: Because all numeric symbols are less than 10, a character symbol will suffice for this lab.

Second Structure:

Define a second structure called nodeType

Your structure will contain only two items.

An instance of recordType and a node type pointer called pNext

Third Structure:

Define a third structure called stackType.

This structure will hold a nodeType pointer named pTop.

You will be creating a stack with dynamic memory. (Make sure you look at the notes on Brightspace and review rules of the stack)

Write the following Functions:

FUNCTION 0: Copy in your PrintHeader, CenterString, and PrintDivider functinos

- Don't forget to define your macros and prototypes in your header file

FUNCTION 1: void Push(stackType* pTop, nodeType * newElement)

- This function will add a new element to the stack

FUNCTION 2: *void or data type* Pop(stackType * pTop)

- This function will return the top element of the stack.
- You may process the data at the top of the stack and then pop it or
  You may pop it (returning a copy of the data) and then process it. This part is up to you.

FUNCTION 3: int ReturnPriority(char symbol)

- This function will return 0 for the character '('
- This function will return 1 for the '+' or the '-' character
- This function will return 2 for the '*' or the '/' character

FUNCTION 4: OpenDataFiles(you figure out the parameters)

- This function will open your input and your files

FUNCTION 5: CloseDataFiles (you figure out the parameters)

- This function will close your input and output files.

FUNCTION 6: Your main function

HINT: The main function will become long in length, documenting your code as you go along will help avoid errors.

Open your data files and initialize your stack pointer.

The following code should only execute if your input file pointer is not equal to NULL:

You're going to set up nested loops:

First loop: you have data in your input file:
HINT: use fgets to read in an entire line from the input file

Second loop: For every symbol in the string, you just read in (inside your input file)
HINT: Use a for loop from 0…strlen(your line of data)
Do The following:

Optional if-statement: If the character is not a space, an end line character, and does not equal 13 (the extra character for the enter key in Windows) do the following:

(This if-statement can make debugging easier):

- If the current symbol is a number (check the ascii range or use isdigit), then print the number to your output file

- If the current symbol is a '(', push the symbol onto your stack

HINT: This requires you to create and initialize a node

- If the current symbol is a ')' then do the following:

  Loop until you find the '('
  Pop a symbol off your stack, if the symbol is not a '(', print it to your output file.
  HINT: Popping the stack will be your primer and changer

- If your current symbol is an operator (Multiply, Divide, Add, or Subtract) then do the following:

  Loop while ReturnPriority(stack top) >= ReturnPriority(current symbol)

  Pop the top of the stack and print it to your output file

  When the while-loop is done, push current symbol onto the stack.

Restart loop 2 for the next character.

When you are done with loop 2:

Pop and print each symbol to your output file until the stack is empty.

When the stack is empty (and the line is done), print an enter key to your output file.

Restart loop 1 (if there are no other lines in your input file, the loop will exit, and you are done.

SAMPLE OUTPUT:

From your input file: (2 * 3 + 4 * (5 – 6))

Your output line should be: 2 3 * 4 5 6 - * +

NOTE: Make sure your input file has Unix enter key symbols. If you have an extra blank line in your output this is probably why.

EXTRA CREDIT - 20%

If your lab does not run with proper output, no extra credit may be earned.

For the extra credit you will use the post fix results file from the above and evaluate the equations from the input file above.

Before you begin:

- Add a double called number to your recordType.
- Also, because we will use the output file as an input file, you will want to change the opening of that file from “w” to “w+”.

If your output file from above is correct (and it needs to be) you may assume perfect input.

The original equation (the original input file) and the answer will be printed to a new output file called “*ExtraCreditOutput.txt*”

Coding the extra credit:

After your while-loop has completed, rewind both files so that they can be used as input files.

Example function calls to rewind your files:

```
// Move the file pointer back to the beginning of the file
rewind(pInput);
rewind(pOutput);
```

Read in a line from the original input file and print that line to your new output file.

Below that string, print "ANSWER = " to your new output file.

Using a loop to read one line from your original output file.
HINT: Use the fgets function

Using a line from the original output file, evaluate each character as follows:

```
If the symbol is a number, push the number onto the stack.
If the symbol is an operation (add, subtract, multiply, divide):
  Pop the stack twice (two doubles)
  Perform the operation dictated by the symbol (Write a function for this)
  Push the result onto the stack.
```

When you are done processing the characters from the string, you should have only one double on the stack (the result). Pop the result and print the answer to your new output file.

NOTE: Your output must be correct to receive any points for the extra credit

Sample Extra Credit Output:

```
( 2 * 3 + 4 * ( 5 - 6 ) )
ANSWER = 2.00
```

RUNNING YOUR PROGRAM

Your program must run in gcc on the unix operating system with 0 errors and 0 warnings.

To run your program:

```
gcc *.c -Wall
```

This will create an output file named a.out

Valgrind must run with 0 errors and 0 warnings.

To run Valgrind:

```
valgrind --leak-check=full ./a.out
```

FILES TO SUBMIT:

<Last Name><FirstInitial>_Lab6.c
<Last Name><FirstInitial>_Lab6_Functions.
<Last Name><FirstInitial>_Lab6.h

Lab6_Output.txt

ExtraCreditOutput.txt ← *IF YOU DID THE EXTRA CREDIT*

ADDITIONAL PARTIAL CREDIT GRADING RUBRIC:

NOTE: The full rubric is on Brightspace under Misc

None.

Note to Grader: Any partial credit outside of the rubric must be approved by the teacher.