Lab 3 – Stack Module – Paren Revisited

Learning Goals

1) Develop your ability to program in C and use a Unix shell.
2) Develop your understanding of the stack data structure.
3) Develop your understanding of header files.
4) Develop your understanding of "blackbox" testing.

Your Task

Previously, you implemented a stack for storing characters. You then built upon it to write a parentheses checking program.

Arguably, the stack implementation should have been in a separate module from the parentheses checker, allowing it to be used by different files. In this lab, we will do exactly that.

1)

Create **stack.c**, implementing the declarations in **stack.h**. (in /home/public/labs/lab3/)

1) Make sure to include stack.h,
2) Define a max STACK_SIZE as 100.

You can reuse a lot of your work from Lab 2b, but there are also three new methods:

- peek() returns the element at the top of the stack, but does not pop it off
- isStackEmpty() returns whether the stack is empty (1) or not (0)
- emptyStack() empties the stack entirely.

Notice that there should be NO main method.

2)

Next, test your stack implementation using **stack_test.c** (in /home/public/labs/lab3/)

stack_test.c performs what is termed as a **black box** test. It does not see anything inside your stack.c implementation. However, stack_test knows what stack.c is supposed to do according to our interface defined in stack.h. stack_test rigorously tests many possible calls to stack.c, including popping an empty stack and overfilling a full stack. Good developers make black box testing a habit.

Debug until your stack.c passes all 10 tests.

If your stack is not passing some tests, debug by using printStack() to see the contents of your stack. You can call printStack() just at the point of a failed test in stack_test.c.

See the Sample Run below for an example of how to compile and run.

3)

Finally, make a copy of your parenCheck.c

Edit it as follows:

1) Add an include **stack.h** to the top of parenCheck.c
2) Remove all traces of the original stack from the program (The only thing left after the include statements should be your original main method!)

Compile and test your parenCheck on the same input files from the previous lab. Confirm that it works as well as your original version.

Finally, submit stack.c and parenCheck.c to the submission directory.

Sample Run, passing all tests.

```
$ gcc -o test stack.c stack_test.c

$ ./test
Passed 1
Passed 2
Passed 3
Passed 4
Passed 5
Passed 6
Passed 7
Passed 8
Passed 9
Passed 10
Passed 11
Passed 12
Passed 13
Passed 14
Passed 15
Passed 16
Passed 17
Passed 18

$ gcc -o parenCheck stack.c parenCheck.c

$ ./parenCheck < INPUT_FILE.txt
```

Grading

This assignment will be scored out of 100 points:

20 pts – stack.c implements stack.h correctly.

30 pts – All tests in stack_test.c pass.

20 pts – parenCheck.c now works only when compiled with stack.c – all traces of old stack code are gone.

30 pts – Valid test cases all correctly execute. Messages for invalid cases accurately reports the line, position, and expected characters.