Fun with Stacks
EE 312
Due: Thursday 2/28/19 at 10:00 PM

100 points

Stack implementation:

We will be defining the Stack ADT. In two seperate projects, you will be implementing the Stack ADT using an array of structures in one and using a linked list in the other. You will then use a stack to solve the problems shown below.

Flood Fill

Given an input file of the following format (a fake picture):

```
yyywwbbbbbbbggggg
yyybbbbbbbgggbbbb
ybybybybybwwwwwyy
ybbbbggwwwwwwbbbg
bgggggwwbbbbbbbbb
yyyyyyyyybbbyyyyy
ggggyyyygggggyyyy
```

that uses characters to represent colors in a picture, you will need to write a function that will "flood fill" an area with another "color." For example. If I were to flood fill the pixel at row 0 col 6 (a"b") with a "P", I would get this:

```
yyywwPPPPPPPggggg
yyyPPPPPPPgggbbbb
yPyPyPyPyPwwwwwyy
yPPPPggwwwwwwbbbg
Pgggggwwbbbbbbbbb
yyyyyyyyybbbyyyyy
ggggyyyygggggyyyy
```

Every pixel that has the same color and is connected to the area of the flood fill is changed to the new color.

Write a program that reads in a file (provided at the linux prompt) that is at most 25 rows and 25 columns and repeatedly prompts the user for a row and column number, and a "color". The program will fill that area with the new color, show the new picture and prompt the user again. The program will end when the user enters -1 for the row or column. You will use the "stack312_ll.h" file as the definition for the stack use you create to solve this problem.

Example Run:

```
linux prompt> ./flood_fill fake_picture.txt
```

```
yyywwbbbbbbbggggg
yyybbbbbbbgggbbbb
ybybybybybwwwwwyy
ybbbbggwwwwwwbbbg
bggggwwbbbbbbbbb
yyyyyyyyybbbyyyyy
ggggyyyygggggyyyy
```

Enter a row: 0
Enter a column: 6
Enter a color: P

```
yyywwPPPPPPPggggg
yyyPPPPPPPgggbbbb
yPyPyPyPyPwwwwwyy
yPPPPggwwwwwwbbbg
PggggwwbbbbbbbbbB
yyyyyyyyybbbyyyyy
ggggyyyygggggyyyy
```

Enter a row: 1
Enter a column: 1
Enter a color: G

```
GGGwwPPPPPPPggggg
GGGPPPPPPPgggbbbb
GPGPyPyPyPwwwwwyy
GPPPPggwwwwwwbbbg
Pggggwwbbbbbbbbb
yyyyyyyyybbbyyyyy
ggggyyyygggggyyyy
```

Enter a row: -1
Enter a column: 1
Enter a color: G

Equation Checker

Given an equation such as: ([1+3]-42/(4*4)) your task is to determine if the parenthesis, square braces, and angle braces match. If they do, the output of the program will be "valid expression". If the equation has a problem, your output will reflect the error. For example: ((a+b+) would result in "missing)" . You will use the "stack312_arr.h" file as the definition for the stack use you create to solve this problem.

You can ignore the actual equation and just focus on the delimiters.
Note: You may assume the equation is at most 80 characters long.

Input to program - Put the test expressions in a file (for example, "exp.dat") one expression per line and then get the file name from the command line.

if the file exp.dat contained:

```
(<a+b>-6
[(hey)-9]

linux prompt > ./check exp.dat

(<a+b>-6 === missing )

[(hey)-9] === valid expression
```

NOTES:

- You must do these programs by yourself.
- The programs must be done using a Linux environment. Note: Your code must compile and run on kamek.ece.utexas.edu.
- The programs must be modular, with significant work done by functions. Each function should perform a single, well-defined task. When possible, create re-usable functions. Do not write trivial functions such as a function to read a single value from an input file.
- You must place appropriate functions in a library. You will be given a starting Stack ADT header file. Don't change the function definitions in the .h file.
- You will be turning in two zipped projects.
- We should be able to read a "readme.txt" file (for instructions on how to make and run the program), unzip the file, and type "make" to compile and link the project.

Turn in: Two sets of files that include: readme.txt (gives instructions for unzipping and running code), driver.c, stack312_??.c, stack312_??.h, and a makefile (you must write your own makefile). The "??" will either be the "ll" or the "arr" version of the stack implementation.

Upload: Turn in two zipped files named prog03list_xxxxx.zip and prog03array_xxxxx.zip, where xxxxxx is your UT EID to Canvas.

Be sure to follow the style standards for the course.

rlp 9/20/18