CS 2123
Summer 2014

Homework 2

Assigned: 6/9/2014
Due: 6/16/2014 11:59 pm

1. Stacks in C (50 pts)
Implement a stack to store and remove integers that will be read from a file. The data file contains a series of integers, one per line. There are two kinds of data in the file. Any integer > -99999 is an int that will be pushed onto your stack. Whenever you read -99999, this is a signal to pop your stack.

The format of the data file is:
<int to push>
<int to push>…
-99999
<int to push>
-99999
etc.

Note: the data file WILL NOT cause an underflow error.

To create the initial stack, use malloc to allocate enough space to store 10 ints. Keep track of how many elements are on your stack. When you stack reaches capacity, you need to allocate more space to your stack (another 10 ints). You can use realloc, or something else like malloc/copy/swap. You do not need to shrink your stack's capacity.

You are required to implement the following stack functions: push, pop, empty, and full.
push takes an int parameter (the value to push onto the stack) and should probably return success or failure. failure to push might happen if growing the stack fails.
pop returns the int that was removed from the stack. pop could an error condition if the stack is empty when pop happens (again, the data file will not cause this).
empty returns TRUE if the stack has no elements, otherwise FALSE.
full returns TRUE if the stack does not have any room for more elements, otherwise FALSE.

Your program must print the assignment 2 and your name. Additionally, each time you read -99999 (i.e., each time you receive a signal to pop the stack) you should print the # of elements in the stack after popping and the integer that is popped off the stack. You should also print a message every time your stack grows.

For example, my program might print the following (NOTE: the data file results are fake):

```
Assignment 2 Problem 1 by El Professor
# elements after popping: 5   integer popped: 22
# elements after popping: 3   integer popped: 7
Stack capacity has grown from 10 elements to 20 elements
# elements after popping: 12  integer popped: 14
```

2. Recursion in C (50 pts)
Exercise 3.2.2 in the textbook. You need to write 2 functions, a recursive and an iterative solution for gcd. Your main should call each one and then print the arguments used and the result of each gcd call. Use the following values for x and y for testing:

x	**y**
3	3
9	21
12	18
36	27
1	12
105	91

Clearly, the results from both the recursive and iterative functions should be the same otherwise something is wrong with one or both of your functions.

The output should look something like:
```
Assignment 2 Problem 2 by El Professor
Recursive GCD: x = 3, y = 3, result = <result goes here>
Iterative GCD: x = 3, y = 3, result = <result goes here>
Recursive GCD: x = 9, y = 21, result = <result goes here>
Iterative GCD: x = 9, y = 21, result = <result goes here>
```

Deliverables:
Answers to homework problems should be submitted as two C source code files, one file for each problem. Archive and submit the files in UTSA's Blackboard system under Assignment 2.