Homework 5 - Bubble-Pop

In this assignment, you will do the following:

1. Gain a working knowledge of Abstract Data Types and how they can be implemented in C.

2. Experience *implementing* an ADT from a given specification.

3. Using your result from (3) to build a relatively playable version of a fun puzzle game. (Actually, a client program playing the game is given to you…).

Due: Wed, April 29 at 12:59PM.

"Bubble Pop"

You've probably played puzzle games a lot like this…

The game starts with a grid of colored balloons. The player selects a balloon to pop; if there is at least one other balloon of the same color "connected" to the selected balloon, the selected balloon does indeed pop along with all other balloons of the same color that are "connected" to it. We will call such a collection a "cluster".

The balloons are filled with helium, so if a balloon pops, all balloons *below* it rise up (there is a "ceiling" at the top of the grid).

You can play a version of the game here:

http://poppit.pogo.com/hd/PoppitHD.html

The static screen shots below show a couple of moves.

Scoring: for our version, the player is rewarded for popping larger clusters: if a move results in a cluster of size `n` being popped, the player gets `n(n-1)` added to their score.

The game ends when no clusters remain (the end board may or may not have any remaining balloons).

<table>
<tr>
<td>select green pair X
  </td>
<td>select 3 green XX
  </td>
</tr>
<tr>
<td>  </td>
<td></td>
</tr>
</table>

Your implementation of the game will be accomplished mainly through completing the implementation of an ADT for which I have given you the specifications (in the form of a header file).

Tasks

1. Complete and test the `BBoard` ADT. This means:
    a. Study the function specifications in `bboard.h`
    b. Study them again.
    c. Decide how you want to represent a board.
    d. Start working on your implementation of the ADT in a file called **`bboard.c`**
    e. Design `struct bboard`. Make sure you understand where it belongs and what code has access to the data elements -- use the `stack` ADT from Part 1 as a model.
    f. Start working on the functions. Adopt the following approach:
        i. Write a function
        ii. Test and debug that function
        iii. Move on to next function

Notes

You have been given a client program `bpop.c` which actually plays the game by creating a board (either from a file or at random). It does some primitive animation. You are welcome to modify it to your liking.

You should, of course probably test the individual functions via small tester programs first rather than by just trying to run bpop and hoping for the best.

You are also given a makefile.

Even if you don't get the entire bpop game to work, you can still receive partial credit for correctly implemented individual functions.

Prioritize: for example, the undo operation will be worth no more than 15% of your final score, so you might save it to the end -- you can still get a working game without supporting that operation.