CISC 3110 **Ari Mermelstein**

Homework #3: Due October 10

Overview

In the previous assignment you designed a program that played a game. In this assignment, you will add some features to this program. Some common pitfalls are detailed below. The other changes are:

1 To allow up to 6 players
2 If two players land on the same place, the one that was there gets put back to start.
3 The winner must reach the "Home" space on the board, and if it overshoots, it moves back that many spaces.

Details follow.

Save your work! We will still be developing more complicated versions of this game.

Summary of Common Pitfalls

- Never duplicate code. If you are copying/pasting you should be creating a function. For example, a turn for player1 is exactly the same as a turn for player2, except for the actual player variable (which should be passed as a parameter).
- A function should be several statements; it should perform a logical task; and it should not be more than a page long.
- Do not hardcode literals into your program!
- Instead of a series of unnecessary "if statements" use the obstacle stored in the board array.
- Do not use "break" statements or "return" statements to fall out of loops (each function should have one return statement at the end)
- There should be a logical flow of function calls, such as: main calls play which calls takeaturn, etc.

Modifications

1 Allow up to 6 players
When the game begins, ask the user: "How many players will be playing (at most 6)?" Most of your functions will need to be modified to allow several

players instead of only 2. Note that if you programmed correctly, you simply have to put in for loops instead of alternating.

2 As far as determining the player that goes first, you still choose the one with the shortest first name. The rest you can put in any order that you choose, but of course every round should be consistent.

3 You should use the same Player structure that you used for homework #2. However, in the main program, rather than having only 2 players, you will **dynamically allocate** an array of Player structures based on the user's input.

4 Note: the switch places function should now switch a player with the player that is the farthest along. To this end, you can write a function that searches an array for the max value.

5 In order to force the player on a space back to start, it will be necessary to know that a player is located at that spot. In order to accomplish this, you may do a search on the players array. Or, you can construct an array parallel to the board array that stores players' positions.

6 In order for a player to win, it is not enough to pass home, but it must land exactly on home. If the player picks a card that would take it past home, you should subtract the "extra" moves from its position, in effect moving it backwards.

Advice:
Work incrementally! Only do one thing at a time. TEST to make sure your modification works. Then, continue with next step. For example: at first you should ignore the order that the players are playing, and make sure your program works for 2-6 players, taking turns in the given order. Only add the ordering after the program has been tested to work with more than two players.

Note: TY3 students should hand this homework in in class on October 10.
ERQ6 students should email me the homework code to mermelstein.homework@gmail.com with the subject ERQ6 HW#3 <Your name>.