SYSC 2006 Foundations of Imperative Programming

Objective:

Remember: Blue words are vocabulary you must know on tests

Write a program to review all concepts that are considered prerequisite: calling library **functions**, arrays, console output and random number generation.

Introduction

We will write a console-version of the Snakes & Ladders™ game. Background information on this game is at www.bigmoneyarcade.com. Your board must match their board, so I know that you played the game online as part of your "*background research*" (That's what you'll call it when you're doing your 4th year project).

Design Solution

Remember: Blue words are vocabulary you must know on tests… These reminders will no longer be given

The first step in writing a program is figuring out how to represent the problem domain (in this case, the board for Snakes & Ladders and the player). At this level of programming, this means figuring out the key data structures to be used by your program.

The board has 100 squares. The key data structure will be an array of 100 integers.

Why integers? What does the integer represent? (Honestly, if you want to do well in this course, cover the answer in the line below and try to imagine what that integer will represent).

The integer value in this array of 100 will equal the CHANGE in a player's position if a player lands on this square:

- If the START of a ladder is on this square, the integer will be a positive number (because the player gets to move "forward" to the END of the ladder)
- If the START of a snake is on this square, the integer will be a negative number (because the player has to move "backward" to the END of the snake).
- If neither, the integer will be zero (because the player stays here)

Looking at the picture on www.bigmoneyarcade.com:

- The square with the number 4 has a ladder that ends at square 26, so the difference will be +26
- The square with the number 23 has a snake that ends at square 6, so the difference will be -17

For this assignment, we will only program the behavior of one player. Consequently, there is only one more required data variable required: an integer that stores the current square on which the player is located.

Program Requirements

Your program will play a game of Snakes & Ladders with one COMPUTER player. A computer player means that the program will play by itself, with no input required by the user. Yes, this is boring, but it will keep the assignment small.

The requirements listed now represent the minimum required, and what you will be marked on (also known as a shall-list). If you have a question about something else:

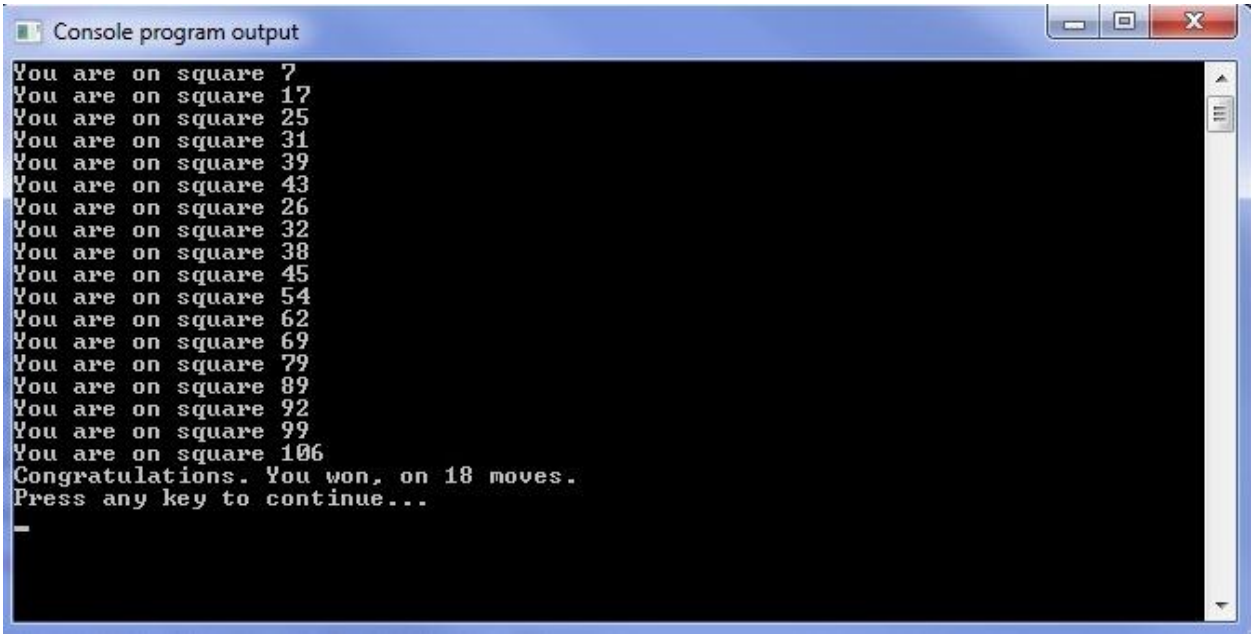
Rule 1: If it does not contradict any stated requirement, you are free to add or extend your program

Rule 2: If in doubt, put a comment at the top of your program, warning the marking TA of the difference of your program.

1. The program will be stored in a file called **assign1.c**
2. The board must match the one given on www.bigmoneyarcade.com
3. Each turn shall be randomly generated by the simulated rolling of two die. You must randomly generate two numbers between 1 and 6 for each turn.
    - The function for random generation is rand(). See www.cplusplus.com
    - For true variability, rand() must be initialized by calling srand() ONCE before any/all calls to rand() are made.

      ```
      #include <stdlib.>
      #include <time.h>
          srand( time(0) );
      ```
4. The player shall start on the first square, and the program shall continue until the player reaches the last square, or beyond: You do not have to land exactly on the last square.
5. After each turn, the program shall print where the player is located.
6. Upon completion, the program shall print the number of turns taken to reach the end.
7. The submitted program shall contain no extra printf() statements. If you use printf() to debug, the extraneous outputs must be removed before submission.

A sample program output is shown.



Detailed Instructions

If you are comfortable with the assignment, go for it. If you still do not know what to do, read on. The steps given will lead you through an incremental development of the program. You need to learn this skill of breaking down a big problem into small increments in order to be capable of doing later assignments in this course. As in the labs, if you follow these steps (instead of rushing through and ignoring the advice) you will actually save time.

1. Write a basically-empty program that declares an array and **initializes all the elements to zero**. For now, ignore the fact there are snakes and ladders. Compile-and-run. Yes, it does nothing yet, but still, compile-and-run!
    - Be a perfectionist: If you are a smart kind, you may think that arrays and variables in general are automatically initialized to zero. Don't take a chance. Explicitly set all elements to zero.
2. Modify the program to now move a single player from start to finish.
    - Declare a variable to represent the player's position and initialize it to the first square.
    - Declare a second variable to represent the roll of the dice.
    - Write a loop which keeps "rolling the dice" and using that number to advance the player's position. Again, ignore snakes and ladders for now. The player should advance continuously in steps of 2 to 12 from start until finish. Each time through the loop, you should print the player's current position. Do not worry about the final message yet.
    - Compile-and-run. Run until this part works properly.

3. Let's fine-tune what we have so far:
    - If you haven't already done so, initialize the random number generator.
    - Add in a counter for the number of turns taken and print out the message upon completion.
    - Compile-and-run.
    - Check all your print statements and make sure they match the sample given above. The exact number of turns will differ each time.
4. Finally, we will add the snakes and ladders.
    - Before we initialized ALL elements to zero. After this default initialization, now add in the exceptions. One-by-one, for the START of each ladder, set that array element to the appropriate (you need to calculate this) positive number; repeat for the snakes, with negative numbers.
    - For each turn, after the player has moved according to the roll, if necessary, "move" the player "up" a ladder or "down" a snake.
    - Compile-and-run.

Question: How will you know that your program is now working, including snakes and ladders?

Submission

A single file **assign1.c** shall be submitted through the SUBMIT program that is available for download from a link on the Course Resources page, before the deadline.