CS 162 Assignment #1 Go Hooping

Design Exercise & Document due: Sunday, 1/14/2024 Sunday, 1/21/2024 11:59 p.m. (Canvas)
Assignment due: Sunday, 1/21/2024 Sunday, 1/28/2024, 11:59 p.m. (TEACH)
Demo due: Friday 2/2/2024 2/9/2024 (without late-demo penalties)

Goals:
- Practice good software engineering design principles:
  - Design your solution before writing the program
  - Develop test cases before writing the program
- Review conditionals, loops, functions, and array
- Use functions to modularize code to increase readability and reduce repeated code

Problem Statement:
You are tasked to implement a C++ program that simulates a basketball shooting game for a local high school. The game supports 2 players, with each player attempting to score as many points as possible by making shots from 5 fixed shooting positions.

At each shooting position, there is a rack of 5 basketballs. Out of the 5 balls, the first 4 are regular balls, worth 1 point each. The 5th ball, often nicknamed the "money ball", is worth 2 points. The "money ball" can only be shot after the 4 regular balls are shot.

Of those 5 positions, the player can choose a position to have a rack consisting only of "money balls", which are all worth 2 points. This position is called "money-ball rack".

The goal of this game is to score as many points as possible. Player 1 needs to finish shooting from all 5 positions before Player 2 shoots. The player with the highest score at the end of the game is the winner. If the 2 players have the same score, then it is a tie game.

After the game is completed, the players will have the option to play again or quit. The program should run until being asked to quit.

Here is a video demonstration. Note: ignore the 2 Starry balls and the shot clock restriction.
Implementation Requirements:
1. Before each player shooting, prompt them to choose the “money-ball rack” position. The program should re-prompt until a valid input is given.
2. Before each player shooting, prompt them for their “shooting ability”. The shoot ability is in a range from 1 to 99, inclusive. The program should re-prompt until a valid input is given.
3. Both players’ shooting accuracy is determined by their shooting ability. For instance, if Player 1 has a shooting ability of 60, it means they have a 60% chance of successfully making each shot, whether it’s a regular ball or a money ball. This chance is determined randomly for each individual shot, not for all shots combined. Even if a player's shooting ability is less than 100, there is still a possibility that they could make every shot in the game. (hint: use rand() function)
4. **For each player, you must use an array to keep track of the shooting result** for displaying later and calculating the final score.
   (*Note: it is okay to use multiple arrays to store the shooting result for one player.)
5. The shooting result of each player should be displayed using the following notation:
   - X for missed shots, worth 0 points
   - O for made shots, worth 1 point
   - M for made shots, worth 2 points (the "money ball")
6. For each player, the program needs to display the outcomes of each rack, scores for each rack, and the total score.
7. After both players have completed their shooting, declare the winner (or a tie game).
8. Prompt the players to enter whether they want to play again or quit the game, until a valid input is given. If they choose to play again, restart the program from the very beginning.
9. Your program must handle the same data type of errors and be able to recover from them. That is, you may assume that the user will enter data of the correct type, but you must verify that they entered a valid value and recover otherwise. (hint: using loops)

Example Output: (User inputs are highlighted)
Note: You don’t have to follow the example output exactly, but you need to display all required fields and they must be readable.

Welcome to the basketball shooting contest!

Player 1:
Where do you want to put your money-ball rack? Enter 1-5: 4
Enter your shooting ability, from 1 to 99: 60

Rack 1: X X O O M | 4 pts
Rack 2: O O O X X | 3 pts
Rack 3: O X O O M | 5 pts
Rack 4: M M X M M | 8 pts
Rack 5: O X O X X | 2 pts

Total: 22 pts

Player 2:
Where do you want to put your money-ball rack? Enter 1-5: 0
That’s not a valid input.
Where do you want to put your money-ball rack? Enter 1-5: 5
Enter your shooting ability, from 1 to 99: 65

Rack 1: X X X X X | 0 pts
Rack 2: O X X X X | 1 pts
Rack 3: O O O O M | 6 pts
Rack 4: O X O X X | 2 pts
Rack 5: M M M M M | 10 pts

Total: 19 pts

Player 1 is the winner!!

Do you want to play again? (1-yes, 0-no): 3

Sorry, that’s not a valid input.
Do you want to play again? (1-yes, 0-no): 0

(15 pts) Extra Credit: Multi-players using Dynamic Arrays

Instead of supporting only 2 players, modify the program so it can support N players.
Additional requirements:
  • The value of N is determined by a user input during runtime.
  • Use dynamic arrays allocated on the heap to keep track of the total scores of players.
  • You must not have memory leaks. Make sure you use valgrind!

Programming Style/Comments

In your implementation, make sure that you include a program header. Also ensure that you use proper indentation/spacing and include comments! Below is an example header to include. Make sure you review the style guidelines for this class, and begin trying to follow them, i.e. don’t align everything on the left or put everything on one line!

```cpp
//******************************************************************************
** Program: hooping.cpp
** Author: Your Name
** Date: 1/10/2024
** Description:
** Input:
** Output:
******************************************************************************
```

Design – Due Sunday 1/14/2024 1/21/2024, 11:59pm on Canvas
Refer to the Canvas Design Guide page for instructions and expectations.

Step 1: Take the Design Exercise 1 on Canvas.

Step 2: Create a design document (Example Design Doc.pdf)
Understanding the Problem/Problem Analysis:
  • What are the user inputs, program outputs, etc.?
  • What assumptions are you making?
  • What are all the tasks and subtasks in this problem?

Program Design:
  • What does the overall big picture of each function look like? (Use Flowchart or pseudocode)
    o What variables do you need to create, when do you read input from the user?
What are the decisions that need to be made in this program?
What tasks are repeated?
How would you modularize the program, how many functions are you going to create, and what are they? What are the detailed steps in each of those functions?
- What kind of bad input are you going to handle?

Based on your answers above, list the specific steps or provide a flowchart of what is needed to create. Be very explicit!!!

**Program Testing:**
Create a test plan with the test cases (bad, good, and edge cases). What do you hope to be the expected results?
- What are the good, bad, and edge cases for ALL input in the program? Make sure to provide enough of each and for all different inputs you get from the user.

Electronically submit your Design Doc (.pdf file!!!) by the design due date, on Canvas.

---

**Program Code – Due Sunday, 1/21/2024 1/28/2024, 11:59pm on TEACH**

**Additional Implementation Requirements:**
- Your user interface must provide clear instructions for the user and information about the data being presented
- Use of array is required.
- Your program must catch all required errors and recover from them.
- You are not allowed to use libraries or functions that are not introduced in class.
  - Libraries that are allowed: `<cmath>`, `rand()` and `srand()` from `<cstdlib>`, `time()` from `<ctime>`, `<string>`, `<iostream>`
- Your program should be properly decomposed into tasks and subtasks using functions. To help you with this, use the following:
  - Make each function do one thing and one thing only.
  - No more than 15 lines inside the curly braces of any function, including main().
    - Whitespace, variable declarations, print statements (cout), vertical spacing, comments, and function headers do not count towards those 15 lines.
  - Functions over 15 lines need justification in comments.
  - Do not put multiple statements into one line.
- No global variables allowed (those declared outside of many or any other function, global constants are allowed).
- No goto function allowed.
- No vectors are allowed.
- Your program should not have any runtime error, e.g. segmentation fault.
- Make sure you follow the style guidelines, have a program header and function headers with appropriate comments, and be consistent.

---

**Compile and submit your assignment**
Electronically submit your C++ program (.cpp file, not your executable!!!) by due date, on TEACH.

Remember to sign up with a TA to demo your assignment. The deadline for demoing this assignment without penalties is 2/2/2024 2/9/2024.