Lahore University of Management and Sciences

Lab-3 Manual

for

Introduction to Programming

(CS200)

Dr. Mian Muhammad Awais

LAB GUIDELINES

- Make sure you submit the lab before 11:50 AM. Any late submission will not be graded afterwards. In case of internet connectivity or electricity issues make sure to email your assigned TA before 2:00 PM. **You MUST cc your breakout room TA in that email as well**. If you do not **cc** your breakout room TA, your case will not be entertained.
- For every lab, there will be a folder created on LMS. You must submit your work in the respective folder during the lab time, you and only you are responsible for your submissions.
- You will be allowed to discuss the questions in the first half of the lab session for a few questions. After that, there will be a portion of lab where you cannot converse and must work for yourself. No discussion is allowed in later time period.

- You should do your work with utmost clarity and precision. Do not waste your time trying to do something you do not understand. Ask Lab instructors for help, that is what they are there for.
- Any legitimate cheating case can and will be reported to Disciplinary Committee without any leniency. Plagiarism Software make our task easier.
- Please follow the lab etiquettes and follow code of conduct in the session.
- Do not start Personal chat during your zoom meeting and raise your hands before asking questions.
- **Make separate .cpp files for each question. Naming convention for .cpp files is: YourRollNumber_TaskX.cpp. Before submission, copy all the .cpp files in a folder named LabX_AssignedTAName_YourRollNumber.zip. Submit the .zip file only! (no .rar file submissions). X should be replaced by appropriate number.** Failure to follow the naming convention may lead to deduction of marks.

OBJECTIVES

- Structs
- Pointers
- Setters/Getters
- Access Modifiers

LAB EXERCISES

Question # 1 [Marks: 20] Est. Time: 30 mins

This question is designed to warm you up to the concepts of structs, access modifiers and pointers. You are required to make a struct named "**practiceStruct**" which should have a private integer variable and 4 member functions:

- Setter (should take a pointer to integer as its parameter)
- Getter (should return a pointer to integer)
- Function named "**print**" (should print the value of private member using the defined getter function)
- Function named "**incrementPrint**" function (should first increment the private integer variable by 1 and then print it using the defined getter function)

For int main(), implement functionality in the following order:

- Prompt user to enter any integer number
- Set the number entered as the value of the private integer variable of the struct
- Make a call to the "print" function
- Make a call to the "incrementPrint" function
- Store the updated value of the private integer variable in some variable (declared in main) and print the value of this variable

SAMPLE OUTPUT # 1

Enter number: 5

Your number is: 5

Your incremented number is: 6

Number returned to main is: 6

SAMPLE OUTPUT # 2

Enter number: 20

Your number is: 20

Your incremented number is: 21

Number returned to main is: 21

Marks' distribution:

Proper Skeleton of the structure = 5

Properly working functions of the struct = 10

Int main() and output formatting = 5

Question # 2 [Marks: 20] Est. Time: 40 mins

Make a struct named "**swap**" which has one member function named "**oddSwap**". This function takes a pointer to integer array as its parameter and return a pointer to an integer array. The function's body will be performing the following tasks:

1. The function should swap the values of odd indices of the input array such that value at index 1 is replaced by value at index 3, value at index 5 gets replaced by value at index 7 and so on
2. If the number of odd indices is odd (swap pairs of two cannot be formed for all of them for example the last index of array is 13 & you have already swapped 1 with 3, 5 with 7, 9 with 11) in that case the function should make the value of index 13 = - the value of index 1 after being swapped with value at index 3. For this purpose, you store the value of Index 3 in a Temp variable.

NOTE: All the swappings should be made in the array passed to the function without the help of any new/second array.

In int main(), make 2 integer arrays: one of size 8 and other of size 10. Take input from user in both the arrays. Make a call to oddSwap and store the returned array in a new variable. Print both the input arrays as well as the returned/updated arrays

SAMPLE OUTPUT

```
intputArray_1:

78,91,2,202,6,19, -55,48

outputArray_1:

78,202,2,91,6,48, -55,19

inputArray_2

78,91,2,202,6,19, -55,48,32,10

outputArray_2

78,202,2,91,6,48, -55,19,32, -202
```

Marks' Distribution:

Return type of Function should be pointer = 5

Odd swap Function with original array unchanged = 10

Output Format = 5

Question # 3 [Marks: 60] Est. Time: 40 mins

Write a struct named “**player**”. The struct contain two attributes: name (string) and id (integer) of a Player. You must implement the following functions for the Struct:

- **Default Constructor** (Assign 0 to id and name)
- **Overloaded Constructor** (Take input a string and int and initialize object)
- “**rollADie**” (The function rolls a die, that is, it randomly generates a number between 1 and 6, and returns it. Include <time.h> library, you have to initialize srand(time(0)); in the main as well to generate a new value each time and use rand() % 6 + 1, to generate number between 1 and 6)
- Make **Getter/Setters**.

Using the struct above. You must make another Struct named “**team**’**.** In this struct, you will have following attributes:

- **An array of 3 Players.**
- **Number of Players (int).**
- **Name of team (string).**

You must implement following functions for this struct as well:

- **Default Constructor** (Assign 0 to num of players and name)
- **Overloaded Constructor** (Take as input a string for team name, an int =3 for number of players. And in this function, you will take input from user the name and ids of every player and save them in the array)
- “**printTeam**” (This function simply prints the name, number of players, and information of all players)
- “**chooseAndPlay**” (This function chooses the random player from the team and print its information. Then, this player rolls the die (remember the function we created in the Player struct), and that die value is return from this function)
- Add any **getters/setters** needed as per use.

In int main(), create two teams. Print the team using the function defined above. Then, it starts match between two teams. The match is played as follow:

- The number of rounds is infinite until user says stop. Implement an appropriate condition.
- In each round a random player is picked from both teams using the “**chooseRandom**” function. The player who rolls the 6 first wins the round, and the respective team’s score is increased by one. When the user has had enough and asks us to stop; displays the winning Team (use the print function again).

Marks' distribution:

For each perfectly functional struct = 10 (20 in total)

Perfectly working choose and play functionality = 30

Main = 10

Best of Luck! ☺