Programming Assignment - 2

CS200 - Introduction to Programming

Dr. Mian Muhammad Awais

Instructions (Important):

- **Start the assignment early** so that you can finish within the deadline. We have the following late submission policy: you are given a pool of total 5 grace days that you can utilize for all assignments with a 0% reduction in your marks. Remember that there will be **no extensions** after you have utilized all the grace days, so **use them wisely** across all assignments. **Any submission past the deadline time will count as having utilized a grace day, so do not wait for the last minute to submit to avoid late submission due to connectivity issues or any other issues.**
- Due Date: **6th November 2020, 11:55 pm.** Total Marks: **215 + 20 + 25 = 260**
- You have to submit a **.zip** file named **<yourRollNumber>_PA2.zip** containing ***THREE (or FOUR - if you do q4)*** **.cpp files** named: **<yourRollNumber>_question<1/2/3/4>_PA2.cpp**
    - Do not submit **ANY** other files except your **.cpp files**.
        - .exe files, .cbp files, other project files or any other files should **not** be present in your submission.
- We expect your programs to work on all valid inputs, if it fails to work on any valid input, you will be awarded half marks for that part. The examples given under questions are **not** exhaustive, be sure to check all corner cases.
- The marks for every part of all questions have been written in square brackets.
- You will be marked out of [**20]** for:
    - writing clean code: **[5].**
    - Correct and clean input/output formats, intuitive flow of code and error handling: **[15]**.
- In case you have any queries, please feel free to post them on Piazza and ask the TAs in office hours.
- Do not copy code from the internet. Do not share or look at any other student's code. If in doubt, talk to sir or any TA.

Question 1: Operator Overloading [75]

In this question, you will have to implement fractions as a class and perform operations on them using operator overloading.

Create a **class Fractions** with private member variables **[2]**

- int numerator
- int denominator

Implement the following:

- Default constructor (choose the fraction appropriately) **[3]**
- Parameterized constructor **[5]** **Fractions(int n, int d)**

Note: ALL RESULTS OF COMPUTATIONS SHOULD HAVE THE FRACTION IN REDUCED FORM. (if the result is 4/6, it should be displayed as ⅔ where needed);

Other member functions to include using operator overloading are. The operator to overload is mentioned in parentheses:

- Addition of two fractions returning the sum of the fractions ***(+)*** **[10]**
- Subtraction of two fractions returning the difference of the fractions ***(-)*** **[10]**

Note: For the above-mentioned functions, you may find it helpful to make a function for calculating the LCM, but it is not mandatory

- Check if two fractions are equal and return whether they are equal or not. ***(==)***

Note: The implementation should cater for equivalent fractions and return them as true **[5]**

- Increment the fraction ***(+)*** (fraction + 1) **[5]**

Note: You will have two overloaded ***(+)*** functions that will have different definitions and so will be used differently. Hint: it will be defined how Sir did it in class.

- Decrement the fraction ***(--)*** (fraction - 1) **[5]**
- Multiply two fractions and return the result ***(*)*** **[5]**
- Divide two fractions and return the result ***(/)*** **[5]**

Note: Keep in mind that division of fractions is really just multiplication with a fraction's reciprocal

- Check whether the current fraction is greater than the other ***(>)*** **[10]**
- All input of fractions should be using the ***(>>)*** operator **[5]**
- All display of fractions should be with the ***(<<)*** operator **[5]**

In main, prompt the user to enter 2 fractions. The result of each should be displayed along with the input.

e.g. for addition, print: The sum of ‘frac1’ and ‘frac2’ = ‘sum_frac’

1. Check if both fractions entered are equal
2. Add the fractions and store the result
3. Increment fraction 1 and display result
4. Decrement fraction 2 and display result
5. Subtract the updated fraction 2 from updated fraction 1 and store the result
6. Multiply fraction 1 with the sum of fraction 1 and fraction 2
7. Divide fraction 1 by fraction 2
8. Check if fraction 1 > fraction 2

Sample Output:

e.g. if frac1 = 3/9 and frac2 = 1/3, the results of the above instructions would correspond like this

1. compare returns true
2. sum = 2/3
3. frac1 = 4/3
4. frac2 = -2/3
5. difference = (4/3 – (-2/3)) = 2
6. product = 2/9
7. division = 1
8. frac1 > frac2 returns false

Question 2: Multi-dimensional Arrays [55]

Initialize a 2-D *char* array. Both size (rows and columns) and input of characters into array will be done by the user.

Part A - Initialization [15]:

You are required to save a number of rows, and an array to save a number of columns of sizes equal to the number of rows.

The user will enter a number of columns for each row at run time. That could be of variable size.

For example:

D	**D**	**N**	**M**	**A**	**V**	**V**	**V**
J	**H**	**E**	**E**	**E**	**B**	**P**	
K	**L**	**D**	**D**	**V**	**A**		
C	**R**	**F**	**P**	**Y**	**R**	**X**	
O	**H**	**H**	**H**	**G**			

Part B - Remove duplicates [10]

Now you have to compress this 2-D array. By removing characters that occur more than once **consecutively** (in rows), remove them and place a '&' in their place.

For example:

D	**&**	**N**	**M**	**A**	**V**	**&**	**&**
J	**H**	**E**	**&**	**&**	**B**	**P**	
K	**L**	**D**	**&**	**V**	**A**		
C	**R**	**F**	**P**	**Y**	**R**	**X**	
O	**H**	**&**	**&**	**G**			

Part C - Reinitialization [20]

Need to reinitialize arrays such that after compressing each array there is no additional extra space allocated. For instance:

D	**N**	**M**	**A**	**V**		
J	**H**	**E**	**B**	**P**		
K	**L**	**D**	**V**	**A**		
C	**R**	**F**	**P**	**Y**	**R**	**X**
O	**H**	**G**				

Hint to resize array: create temp array, delete current array, point to temp array

Part D - Deallocation and Testing[10]

Properly Deallocate the 2D dynamic array and implement a main function to test

Question 3: Dynamic Memory Allocation/Deallocation + Operator Overloading [85]

You have to design a program for a newly admitted Chief Police Officer (CPO) of a police station. This program will help the CPO to gather data on the prisoners in the prison. You have to store the following variables for each prisoner in a class **Prisoner**:

1. Name
2. Age
3. Blood Group
4. Crime
5. Date of Entry (DD-MM-YYYY format)
6. Duration of Sentence
7. Tentative Release Date
8. Prison-cell number
    a. Assume every prisoner has a **unique** prison-cell number.
    b. Prison-cell numbers start with 0.
    c. As every prisoner has a unique prison cell number, you should use this variable for indexing different prisoners for addition/removal.

Only the CPO should be able to access prisoner data and must be able to keep track of the behaviour of the prisoners depending on their recent behaviour in prison. For this, you need another member variable for the Prisoner class named ‘behaviour’. The CPO can increase or decrease the sentence period of any prisoner depending on behaviour. The types of behaviour are:

9. Behaviour
    a. Very Bad: +5 years, 3 months, and 20 days
    b. Bad: +2.5 years, 9 months, and 15 days
    c. Average: No change
    d. Good: -6 months, and 20 days
    e. Very Good: -1 Year and 3 months

Note: (+) denotes increase in sentence period, (-) denotes a decrease in sentence period. In cases where the dates suggest that the prisoner has served his/her full sentence, s(he) should be automatically removed from the prison data. The default value for behaviour should be **Average**.

Make another class, **Prison**, which would store all the data of the prisoners in a **dynamic array** of type Prisoner.

The CPO should also be able to:

1. Initialize the prisoner data of the number of prisoners present in the prison on his first day as CPO. **[10 - Proper Dynamic Allocation]**
2. Add a prisoner using the overloaded **(++)** operator. Keep in mind that a new prisoner can be added only to a prison cell that is **not** occupied by any other prisoner. **[10]**
3. Remove a prisoner manually using the **(--)** operator. **[10]**

   Hint for 2. And 3.: You can implement this functionality by making a new temporary dynamic array with a new size, delete the current array, and point to the temporary array.

4. Remove a prisoner automatically if the release date has crossed (using behaviour functionality) **[15]**
5. Edit the information of a prisoner (in the case where the CPO tries to edit the cell number of a particular prisoner to a cell number which is already occupied, the CPO should be given the option to swap cell numbers or enter another cell number that is not occupied) **[5]**
6. Make deep copies of the data using the overloaded copy constructor and the overloaded assignment operator **(=) [10]**
7. Sort the details of all prisoners in ascending order by their **release date** using the **insertion sort** algorithm. **[10]**
   a. Here's a short video on insertion sort to get you started: https://www.youtube.com/watch?v=JU767SDMDvA&ab_channel=MichaelSambol
8. View all prisoners' details using the overloaded **(<<)** operator in a clean table form.**[5]**
9. Get a specific prisoners' details using the overloaded **[ ]** operator for viewing, editing, etc. For example if the CPO has an instance *prison* of class Prison, then s(he) could do: *prison[0]* to get the details of the prisoner in prison-cell number 0. **[5]**
10. Delete all prisoners' data. **[5 - Proper Dynamic Deallocation]**

Any extra classes/structs, member functions, global functions, arrays you make and the flow of the program is up to you (**Note**: should be easy to follow and functionality should be correct).

Note: You should print the details of all prisoners to the CPO **everytime** s(he) adds data, removes data, edits data, copies data, edits a deep copy of data to help ensure that the data is processed correctly.

Best of Luck!

BONUS QUESTION

Question 4: Vectors [25]

This next task is very short and simple and will introduce you to the concept of vectors (very useful in this course and later courses if you learn them). Vectors are easier to use than arrays and are commonly used so it is important to learn them in this course. To get you acquainted with vectors, the following video is attached. Please go through this and you will find almost everything you need for this task in this video.

C++ Tutorial 18 - Vectors and Vector Functions

vector - C++ Reference

Instructions:

Note: DO NOT USE INDEX NOTATION TO ACCESS THE VECTOR ELEMENTS FOR ANY PURPOSE OTHER THAN PRINTING THE VECTOR i.e. myVec[i]

1. Create a vector of strings. **[2]**
2. Prompt the user to keep entering a name and adding it to the end of the vector until the user enters the string “stop”. **[3]**
3. Using the built-in function for vectors, determine the number of names entered and print this value. **[3]**
4. Using the total number of names, print the names entered. **[2]**
5. Prompt the user to enter the index value to replace at and a replacement string. Replace the old name with this new name at the specified location and display all the names again. **[5]**
6. ‘Pop’ the last value using the built-in function (which you can lookup). **[2]**
7. Prompt the user to enter an index to delete the value at, and then delete the value at the specified index. **[3]**
8. Print the new size of the vector. **[2]**
9. Clear the contents of the vector and then check whether the vector is empty and display the result. (you cannot use the size of the vector to determine this **[3]**