RMIT UNIVERSITY

Advanced Programming Techniques
COSC1076 | Semester 1 2022
24 hour Programming Exercise

Assessment Type	24 hour Programming Exercise
Start Date	9.00am Wednesday July 8th 2022
Due Date	9.00am Thursday July 9th 2022
Weight	25% of the final course mark
Submission	Online via Canvas.
Learning Outcomes	This assignment contributes to CLOs: 1, 2, 3, 4

General Information

- You must not share your questions or answers. These are individual questions. It is **academic misconduct** to share your questions or work with any individual.
- Please follow the specific instructions for each question that describe where to place your answers in your submission.
- Submission instructions are located in the Appendix on the last page.
- Any code you write must comply with the Course Style Guide.
- You may only use C++14 language features and C++ STL elements that have been taught in the course. The reason for this is to limit the scope of what you are expected to answer to the contents of the course and therefore be fair to all students.

Question 1: Recursion & Debugging (10 marks)

Instructions:

- Place all of your code files for this question in sub-folder named `question1`.
- The sub-folder `question1` in "starter code" provides a starting point. The starter code contains errors and you should fix those while developing your solution.
- You **should not** add more files or use additional libraries that are not already included in the starter code.

In this question, you will work on a code aimed at reading a set of Red7 cards from the standard input, store them in a Binary Search Tree (BST) and output some information about the data stored. The starter code is given in "question1" folder and you will have to *complete and rectify* the given starter code to achieve desired output. The starter code is a slightly modified version of the BST developed during the lectorial (workshops). The starter code for `question1` consists of the following files:

1. `main.cpp`: Contains code that manipulate Card/BST objects. ***Do not modify this file.***
2. `BST_Node.h`: Represent a node of the BST. ***Do not modify this file.***
3. `Makefile`: ***Do not modify this file.***
4. `tests`: Folder Containing sample tests and the expected outputs. ***Do not modify the files in this folder.***
5. `BST.h`: Represent the BST. *You may modify this file.*
6. `Card.h`: Header file for Card class. *You may modify this file.*
7. `Card.cpp`: Implementation of Card class. *You may modify this file.*
8. `Utils.h`: Support functions. *You may modify this file.*

You should *carefully go through* the starter code and modify it **(only the files that are marked "you may modify")** so that it archives the following functionality:

1. Should read a given number of cards from the standard input.
2. Add each card to the BST. *The BST should not store a card if that card is already in it.*
3. After reading all the cards, the program should print the Depth of the BST. The Depth of a BST is the maximum number of node one should traverses to get to a leaf node (leaf node is a node where both left and right nodes are null). The implementation of height function should use recursive programming. *Hint: height of BST at root node is: 1 + maximum(height of left sub-tree , height of right sub-tree).*
4. Then, all the cards in the BST should be printed out to standard output in order. All the functions that determine the order (operators $<, ==$ for cards) is given in the starter code and are correctly implemented.

REQUIREMENTS: Your answer should fulfil the following requirements.

- Should not modify the files marked as "Do not modify".
- Should achieve the functionality given above and pass tests including the two given in "tests" folder. The outputs should match the ".expout" files' output style.
- The depth function **should** be implemented using **recursive programming**. The other functions already implemented in BST, should remain recursive.
- Should make **minimal** modifications to achieve the required functionality.

The marks are distributed as follows:

- Code does not compile or the files marked "do not modify" has been modified *(0 marks).*
- Code compiles without errors or warnings *(+3 marks).*
- Code show desired behaviour and passes tests *(+3 marks).*
- Depth function implemented recursively *(+2 marks).*
- Code style: minimal (required) modification of starter code. i.e., no unnecessary modifications. *(+2 marks).*

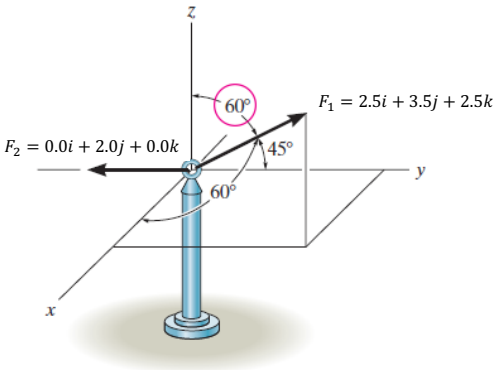
Continued . . .

Question 2: Operator Overloading (7 marks)

Instructions:

- Place all of your code files for this question in a sub-folder called `question2`.
- The sub-folder `question2` in "starter code" provides a starting point.
- You should not add more files or use libraries that are not already included in the starter code.

A force in 3D space can be represented by three components (F_x, F_y, F_z) which stands for the force component along the x-axis, y-axis and z-axis respectively. This form of a force is called Cartesian Vector Form and a force vector is usually written as: $F = F_x i + F_y j + F_z k$.



In this question, you will implement and use a data structure to hold a force expressed in Cartesian Vector Form. Here, Newton (N) is the unit for force.

The starter code for `question2` consists of the following files:

- `main.cpp`: Contains code that manipulate Force objects. ***Do not modify this file.***
- `Force.h`/`Force.cpp`: The files for the Force class.
- `tests`: Folder containing "black-box" tests (input and expected outputs for two senarios). ***Do not modify the files in this folder.***
- `Makefile`

You are expected complete `Force.h` and `Force.cpp` so that the program compiles and show the expected behaviour.

Hint 1: If there are two forces, $F_a = F_{xa}i + F_{ya}j + F_{za}k$ and $F_b = F_{xb}i + F_{yb}j + F_{zb}k$, acting on a single point, then the resulting force is calculated as follows:

$F_{res} = (F_{xa} + F_{xb})i + (F_{ya} + F_{yb})j + (F_{za} + F_{zb})k$

Hint 2: $F_a > F_b$ if:

$(F_{xa})^2 + (F_{ya})^2 + (F_{za})^2 > (F_{xb})^2 + (F_{yb})^2 + (F_{zb})^2$

Programming by Contract Paradigm

In the *implementation* of your classes, you must use *Programming by Contract* paradigm. That is, The programmer should specify a "contract" (where applicable) that must be complied with when using a specific functionality of the class.

The marks are distributed as follows:

- Code does not compile or the files marked "do not modify" has been modified (0 marks).
- Implementation: Program compiles *(+3 marks)*
- Implementation: Correct behaviour - test cases, good code style, follow structured programming paradigm. *(+3 marks)*
- Conformance to Programming by Contract Paradigm *(+1 marks)*

Question 3: Inheritance, Type casting & Move semantics (8 marks)

Instructions:

- Place all of your code files for this question in a sub-folder called `question3`.
- The sub-folder `question3` in "starter code" provides a starting point.
- You should not add more files or use libraries that are not already included in the starter code.

In this question, you will work on a code aimed at reading a set of Red7 cards from the standard input, store them in ADT called deck and output some information about the data stored. The starter code is given in "question3" folder and you will have to *complete and rectify* the given starter code to achieve desired output. The starter code is a slightly modified version of the Red7 program developed during the lectorial (workshops).

The starter code for `question3` consists of the following files:

1. `main.cpp`: Contains code that manipulate Deck/card objects. ***Do not modify this file.***
2. `Makefile`: ***Do not modify this file.***
3. `Deck.h`: An abstract class that describes the interface for the deck. ***Do not modify this file.***
4. `Card.h`/`Card.cpp`: Modified version of the card class used in workshops.
5. `DeckVector.h`/`DeckVector.cpp`: Deck implementation using vectors. Derived from Deck.
6. `DeckDeQue.h`/`DeckDeQue.cpp`: Deck implementation using deque. Derived from Deck.
7. `Utils.h`: Support functions.

You should *carefully go through* the starter code and modify it (only the files that are **not** marked "**Do not modify this file**") so that it archives the following functionality:

1. Should read a number from terminal that indicate what ADT implementation to be used for storing cards. If the input is 0, then "DeckVector" object should be used, else if 1, a "DeckDeQue" object should be used.
2. Store each card read from the standard input in the deck object.
3. Print the size of the deck and the cards in it.
4. Finally the program should print some details about the underlying implementation of the deck. If deck is an DeckVector object, then print "Deck implemented using Vector". Else if deck is an DeckDeQue object, then print "Deck implemented using Deque. This is to be achieved by completing the function in `Utils.h`.

REQUIREMENTS: Your answer should fulfil the following requirements.

- Should not modify the files `main.cpp` and `Makefile`.
- Should achieve the functionality given above and pass tests including the two given in "tests" folder.
- The outputs should match the ".expout" files' output style.
- should make minimal modifications to achieve the required functionality.

The marks are distributed as follows:

- Code does not compile or the files marked "do not modify" has been modified *(0 marks)*.
- Code compiles without errors or warnings *(+3 marks)*.
- Code show desired behaviour and passes tests *(+3 marks)*.
- Good Code style *(+2 marks)*.

Continued . . .

Appendix A: Submission Instructions

Combined all of the files for the questions into a single ZIP file called `s123456.zip` (change filename to match your student number). Upload this file to the Canvas assessment module for the `24 Hour Programming exercise - Alternate`.

Finally you **must** also include *this* PDF files of questions in your ZIP.

Appendix B: Late Submission Policy

Ensure that you give yourself sufficient time to prepare and submit your work. The submission is a hard deadline. That is, there will be no leeway on late submissions. The late submission policy is:

- 20% penalty if submitted up to 1 hour late
- 50% penalty if submitted up to 2 hours late
- Grade of 0 if submitted 2 or more hours late