CS214: Data Structures 2022

Assignment (1) V1.0

[260 points]

Deadline & Submission

1. At least one team member should submit the compressed group solution as zip file containing the programs to blackboard course under tab–> Assignments (name your assignment file **"A1_ID1_ID2.zip"**).
2. The deadline for submitting the electronic solution of this assignment is **Wednesday 20th April 2022 at 10:00 pm.**

About this assignment

1. This assignment will be solved in teams of 4 except those who got exceptions.
2. The weight of the assignment is 260 points
3. All team members should understand all assignment problems.
4. All code must be standard ANSI C++.
5. Assume any missing details and search for information yourself.
6. Any cheating in any part of the assignment is the responsibility of the whole team and wholeteam will be punished.

Problems

Problem 1 *[25 points]*

Different variations of types **int** and **float** exist in C++ and other languages. They are usually limited by minimum and maximum values. Sometimes it is desired to have versions of these types with unlimited bounds. Java solves this problem by providing **BigInteger** and **BigDecimal** classes. In this problem it is required to develop a new C++ type (class) that can hold unlimited decimal integer values and performs arithmetic operations on them. You will develop in C++ a class, **BigInt** that supports writing statements with extremely long integer values like these:

```
BigInt num1("12345678901234567890123456789012345678901234567890");
BigInt num2("11345678901134567890113456789011345678901134567890");
BigInt num3 = num2 + num1;
cout << "num1 = " << num1 << endl;
```

```
cout << "num2 = " << num2 << endl;
//2369135780236913578023691357 80
cout << "num2 + num1 = " << num3 << endl;
```

Your task is:

(1) Design the class **BigInt** that has the following public interface (set of operations available to use by developers using the class): ***[18 points, 3 points for each function]***

```
BigInt (string decStr); // Initialize from string and rejects bad input
BigInt (int decInt); // Initialize from integer
BigInt operator+ (BigInt anotherDec);
BigInt operator= (BigInt anotherDec);
Int size();
```

You will also need to overwrite the << operator as follows:

```
friend ostream& operator << (ostream& out, BigInt b)
```

Using data encapsulation, you are free to store the digits of the big decimal integer in whatever container you like. You might store them in an array, a vector, a string or whatever. These are details that are not important to the user of your class. You will need to build + and – operations that work on the representation you chose.

(2) Implement the class **BigInt** and write five test cases (including –ve numbers) to test it. Implement a program that runs the test cases and verifies the result. ***[7 points, 2 for each test case and 2 for the class]***

(3) Name a folder "**A1_P1_ID1_ID2**" and put your files inside it (even if it's only one file)

Problem 2 *[30 points]*

You will develop an application for performing calculations on fractions.

(1) First, develop a class Fraction that represents a fraction by one integer divided by another, e.g.,1/3 or 3/7. ***[24 points]***

a. This class defines adding, subtracting, multiplying, dividing and comparing (<, >, ==, <= and >=) fractions by overloading the standard operators for these operations. ***[18 points]***

b. It should also contain a function for reducing fractions. For example 2/6 is reduced after calling the function to 1/3, etc***. [2 points]***

c. You also need to overload I/O operators to be able to input and output fractions naturally using >> and << operators. ***[4 points]***

(2) Separate class specifications from implementation by creating Fraction.h for specs and Fraction.cpp for implementation. ***[2 points]***

(3) Second, develop a class FractionCalculator that utilizes the class Fraction and allows the user to input a fraction and perform calculations by adding, subtracting, etc. another fraction and then keeping the result as a fraction for further calculations. ***[4 points]***

(4) Name a folder "**A1_P2_ID1_ID2**" and put your files inside it (even if it's only one file)

Problem 3 *[30 points]*

You will develop an application for matrix calculations.

(1) It is required to design and implement a generic class **Matrix**, in the form of a class template that accepts a type parameter. This way, when the class Matrix is instantiated, we decide if it should accept float, int or double, etc. ***[2 points]***

(2) **Matrix** class holds a matrix of any size and allocates the required memory as needed. ***[2 points]*** dynamic

(3) **Matrix** class should have a destructor that frees used memory at the end of lifetime of each Matrix objects. ***[2 points]***

(4) **Matrix** class specifications should be in a separate header ".h" file. ***[2 points]***

(5) It should have a pointer-to-pointer attribute that points to the matrix content. It should havesuitable constructors and methods for allocating the required memory space based on the dimensions decided by the user. ***[2 points]***

(6) Overload standard operators and I/O operators to enable **Matrix** class with addition, subtraction and multiplication and suitable input and output capabilities. Add a method for matrix transpose. ***[12 points,2 points for each function]***

(7) Then develop a **MatrixCalculator** class which offers the user a menu of operations to perform on int matrices as follows. Each of these options should be able to accept matrices of varying dimensions, which the user inputs. For multiplication, the calculator should check that two matrices are of dimensions n x m and m x p. ***[8 points, 2 for each choice]***

Welcome to (Your Name) Matrix Calculator

1- Perform Matrix Addition

2- Perform Matrix Subtraction

3- Perform Matrix Multiplication

4- Matrix Transpose

(8) Name a folder "**A1_P3_ID1_ID2**" and put your files inside it (even if it's only one file)

Problem 4 *[5 points]*

The given function **ListPermutations** below prints all the permutations of a given string. It is required to modify this function so that it only prints unique strings. The current function will do exhaustive recursion to calculate all permutations. So, if the given string has duplicate characters, the output will have duplicate words. For example, if it is given the string "Makka", it will print "Mkkaa" four times. Try this function and see how it works.

It is required to change the code so that it only prints unique combinations formed from the characters of the string. So for the above mentioned example, it should print "Mkkaa" once.

Hint. To solve this, all what you need is storing each new word you form. Before printing a new word or storing it, check if it is not already stored. You will need an array, a vector or a set to store the words formed so far.

Name a folder "**A1_P4_ID1_ID2**" and put your files inside it (even if it's only one file)

```
void RecPermute(string soFar, string rest)
{
        if (rest == "") // No more characters
        cout << soFar << endl; // Print the word
        else // Still more chars

        // For each remaining char
        for (int i = 0; i < rest.length(); i++) {
        string next = soFar + rest[i]; // Glue next char
        string remaining = rest.substr(0, i)+ rest.substr(i+1);
        RecPermute(next, remaining);

        }
}
// "wrapper" function


void ListPermutations(string s) {
        RecPermute("", s);
}
```

Problem 5 *[20 points]*

You will develop an application for performing operations on strings.

(1) First, develop a class **StudentName** that represents your full name and has only a variable **name** of type **string**. ***[2 points]***

(2) Your class should contain a **constructor** that takes a string from user. The input string should contain at least 2 spaces. If the user violates this rule, you should copy the last name many times to make the names variable a name with 2 spaces. ***[4 points]***

e.g., "ahmed Mohamed sayed" ➔ "ahmed Mohamed sayed"
"sara ahmed" ➔ "sara ahmed ahmed"
"Khaled" ➔ " khled Khaled Khaled"
"aya ali ahmed sayed" ➔ " aya ali ahmed sayed"

(3) Add a function **print** that prints the detailed parts of the name each in one line. ***[4 points]***

e.g., "aya ali ahmed sayed"

detailed parts of the name are:
1) aya
2) ali
3) ahmed
4) sayed

(4) Add function **replace(int i,int j)** that replaces the name at position I with the name at position j and return true if operation is valid and false if one of the two indices is out of range. ***[5 points]***
e.g., "ahmed hassan ali"
replace(1,2) ➔ true ➔ "Hassan ahmed ali"
replace(3,1) ➔ true ➔ "ali Hassan ahmed"
replace(2,4) ➔ false

(5) Write a **main** function and 5 test cases with different names. For each test case you should call the **replace** function and the **print** function to check the effect of the replacement if valid. ***[5 points]***

(6) Name a folder "**A1_P5_ID1_ID2**" and put your files inside it (even if it's only one file)

Problem 6 *[20 points]*

Write a program that handles an address book program, to process the following functions

(1) First, write a class called **PhoneDirectory.** Add a first name, last name and phone number entry as a private.
(2) The main program uses a simple text-based interface to give the user access to the directory. In a while loop, the program presents the user with a menu of options:

1. Add an entry to the directory
2. Look up a phone number
3. Look up by first name
4. Delete an entry from the directory
5. List All entries in phone directory
6. Exit form this program

(3) Delete an entry by first name. [4 points]
(4) List the directory in alphabetical order by first name. [4 points]
(5) Use at least 3 sorting techniques for points 3 & 4, to perform time analysis on these different algorithms. [4 points]
(6) Lookup an entry by first name. [4 points]
(7) Lookup an entry by phone number. [4 points]
(8) Name a folder "**A1_P6_ID1_ID2**" and put your files inside it (even if it's only one file)

Problem 7 *[20 points]*

If you have a list of songs in a music library and want to sort this list alphabetically. However, you want songs with the title "Untitled" to always appear at the top. Write a function called BiasedSort that accepts a vector by reference and sorts the songs according to the explained rules above.

Name a folder "**A1_P7_ID1_ID2**" and put your files inside it (even if it's only one file)

Problem 8 *[20 points]*

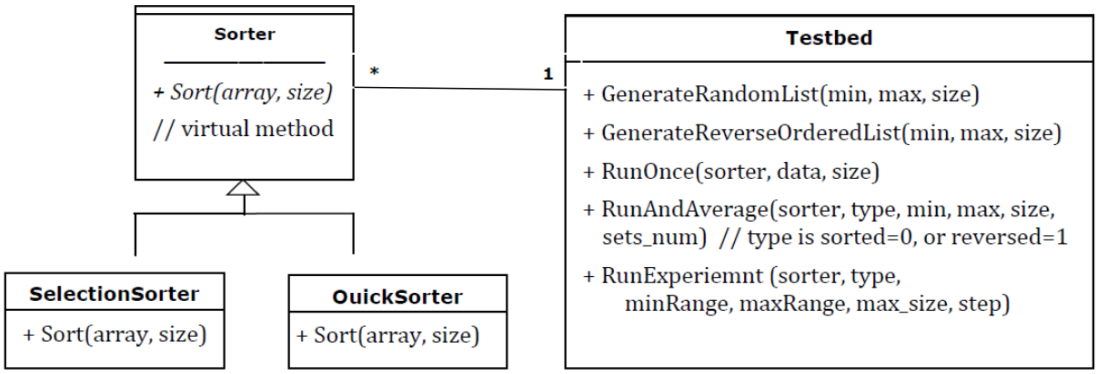
Insertion sort uses linear search to find the right place for the next item to insert. Would it be faster to find the place using binary search (reduce number of comparisons)? We still must shift 1 item at a time from the largest till the right place.
Use binary search on the already sorted items to find the place where the new element should go and then shift the exact number of items that need to be shifted and placing the new item in its place. The algorithm works the same, except that instead comparing and shifting item by item, it will compare quickly using binary search but it will still shift one by one till the right place (without comparison).

1) Plot the performance of the algorithm **against the original** insertion sort.

2) Name a folder "**A1_P8_ID1_ID2**" and put your files inside it (even if it's only one file)

Problem 9 *[40 points]*

In this problem, we will develop classes to use for testing two sorting algorithms (Selection and Quick sort). The class will have methods to support experimenting and analyzing sorting algorithms performance. Below is a high-level UML diagram for your classes. Add any missing details. **Testbed** class has the following functions that you should complete:



(1) **GenerateRandomList(min, max, size)** Generate a given number of random integer data from a certain range. For example, one can generate a vector/array of 10000 integer numbers that fall in the range from 1 to 100000, e.g., [5554, 32300, 98000, 342, …]
(2) **GenerateReverseOrderedList(min, max, size)** Generate a given number of reverse ordered integer data from a certain range. You can first generate random data and then sort them reversed
(3) **RunOnce(sorter, data, size)** Run a given sorting algorithm on a given set of data and calculate the time taken to sort the data
(4) **RunAndAverage(sorter, type, min, max, size, sets_num)** Run a given sorting algorithm on several sets of data of the same length and same attributes (from the same range and equally sorted; e.g., random or reversed) and calculate the average time
(5) **RunExperient (sorter, type, min, max, min_val, max_val, sets_num, step)** Develop an experiment to run a given sorting algorithm and calculate its performance on sets of different sizes (e.g., data of size 10000, 20000, etc.) as follows:
    I. All sets are generated with values between min and max
    II. First, generate **sets_num** sets with size **min_val**. Use **RunAndAverage** () and record average time to process the sets
    III. Next, repeat step ii but with sets whose size increases by step till reaching **max_val**. Each time record average time to process the sets
    IV. For example I should be able to design an experiment to run Quick sort algorithm on randomly sorted integers data taken from the range (1 to 1,100,000) and with input value (data size) from 0 to 100000, with step 5000. This means we will run the algorithms on data sets of 5000, 10000, 15000, …, 100000 randomly sorted integers. Note that with each step you will generate **sets_num** different sets and take the average of their runs
    V. The output of the experiment goes to screen as a table with two columns; first column indicates set size, and second column indicates average time

Write a **main()** demo to show that the function works correctly and to measure the performance of Quick sort and Selection sort in cases of random data and reverse ordered data using **Testbed** class. **Draw plots of your results**.

Name a folder "**A1_P9_ID1_ID2**" and put your files inside it (even if it's only one file)

Notes:

In problems 8 and 9 requires you to plot your results which means you provide chart in excel document with a table of the numbers you obtained from the experiment.
The plot will have running time (in sec or msec) which is represented on y-axis and size of the data on x-axis. Array size rages from 100 to 1000000 or more.

Problem 10 *[50 points]*

In this problem, you should develop a linked list class similar to that provided in the C++ STL. The public interface of your class should provide basic insertion and deletion functions. In addition, it should provide an iterator class **as an inner class** in order to access the data stored in the list. For example, if we have a list of three elements, and want to access the second element, we should declare an iterator and initialize it to the position of the first element, and move to the second position as shown in the code below:

```
list<int> myList;
myList.push_back(1);
myList.push_back(2);
myList.push_back(3);
list<int>::iterator it = myList.begin();
it++;
cout<< *it;
```

notice the usage of the scope operator in the declaration of the iterator, this is because the iterator class is defined as an inner class inside the list class:

```
template<class type>
class myList {
public:
class iterator {
// your code for the iterator class here
};
// your code for the list class here
};
```

Your list class should be a template class. [3 points]
The list class should have the following public interface:

- `list()` – default constructor. [3 points]
- `list(type value, int initial_size)` – constructs a list having 'initial_size' elements whose values are 'value'. [3 points]
- `~list()` – a destructor to clear the list and leave no memory leaks. [3 points]
- `int size()` – returns the current number of elements in the list. [3 points]
- `void insert(type value, iterator position)` – adds an element at position specified by the iterator. For example, if the passed iterator currently points to the second element, this element will be shifted on position, and the new value should be added at the second position. [3 points]
- `iterator erase(iterator position)` – erases the element specified by the iterator and return an iterator to the next element, throws exception if position points **after**

the last element. [4 points]

- `list<type>& operator = (list<type> another_list)` – overloads the assignment operator to **deep copy** a list into another list and return the current list by reference. [4 points]
- `iterator begin()` – returns an iterator pointing to the first element. [3 points]
- `iterator end()` – returns an iterator pointing **after the last element**. [3 points]

You should develop an iterator class the following public interface:

- `void operator ++ ()` – overloads the operator ++, it should advance the iterator one position towards the end of the list, throws exception if it is currently pointing **after the last element**. [3 points]
- `void operator -- ()` – overloads the operator --, it should move the iterator one position toward the beginning of the list, throws exception if it is currently pointing to the first element of the list. [3 points]
- `type& operator * ()` – overloads the dereference operator to return the value contained in the current node by refence to allow its modification. [3 points]
- `bool operator == (const iterator &)` – overloads the equality comparison operator, should return true if the passed operator points to the same node. [3 points]

All node pointers in the list class of the iterator class should be private and inaccessible from outside of the class. [3 points]
No memory leaks or dangling pointers. [3 points]
It is highly recommended to implement it as a double linked list, however, it is up to you.
As mentioned above, our .end() function should return an iterator pointing to a position **after the last element** as the STL .end() function does. This can be done easily by having a dummy node after the actual tail, this dummy node contains no data, but mark the end of the list. So, physically, our list is never empty, which will ease the implementation of insertion and remove operations, as now we don't have to handle an "empty list" case. However, this dummy node should be disregarded when we return the size.
Write a main function to test all the above

Name a folder "**A1_P10_ID1_ID2**" and put your files inside it (even if it's only one file)

Rules

- **Cheating will be punished by giving -2 * assignment mark.**
- **Cheating is submitting code or report taken from any source that you did not fully write yourself (from the net, from a book, from a colleague, etc.)**
- **Giving your code to others is also considered cheating. Both the giver and the taker will get -10.**
- **People are encouraged to help others fix their code but cannot give them their own code.**
- **Do not say we solved it together and we understand it. You can write the algorithm on paper together but each group should implement it alone.**
- **If you do not follow the delivery style (time and files names), your assignment will be rejected**