BCS/CS214: Data Structures

2019 Assignment (1) V1.0

[120 points]

Deadline & Submission

1. At least one team member should submit the compressed group solution as zip file containing the programs under acadox –> tasks (name your assignment file **"A1_ID1_ID2_ID3.zip"**).
2. The deadline for submitting the electronic solution of this assignment is **Thursday 7/3/2019 at 10:00 pm.**

About this assignment

1. This assignment will be solved in teams of 3 except those who got exceptions.
2. The weight of the assignment is 120 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 whole team 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, **BigDecimalInt** that supports writing statements with extremely long integer values like these:

```
BigDecimalInt num1("123456789012345678901234567890");

BigDecimalInt num2("113456789011345678901134567890");

BigDecimalInt num3 = num2 + num1;
```

```
cout << "num1 = " << num1 << endl;
cout << "num2 = " << num2 << endl;
//236913578023691357802369135780
cout << "num2 + num1 = " << num3 << endl;
```

__Your task is:__

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

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

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

friend ostream& operator << (ostream& out, BigDecimalInt 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 **BigDecimalInt** 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_ID3**" 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_ID3**" 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]***

(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 have suitable 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_ID3**" and put your files inside it (even if it's only one file)

Problem 4 *[10 points]*

(1) Design a recursive function to calculate **a** to the power **n** **(a^n)** by following the recurrence equation: **power(a,n) = a* power(a,n-1)** ***[4 points]***

(2) Design a recursive function to calculate **a** to the power **n** **(a^n)** by following the recurrence equation: **power(a,n) = power(a,n/2) * power(a,n/2)**. Optimize your function if n is odd. ***[4 points]***

(3) Write a main function to test the previous two equations. ***[2 points]***

(4) Name a folder "**A1_P4_ID1_ID2_ID3**" and put your files inside it (even if it's only one file)

Problem 5 *[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_P5_ID1_ID2_ID3**" 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 6 *[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_P6_ID1_ID2_ID3**" 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.**