CS-1004: Object-Oriented Programming

Assignment 3

(Deadline: 23rdApril, 2022 11:59 PM)

Submission: Header (.h) files are provided where required. Create a .cpp file for each question, name the .cpp file the same as the header. E.g Car.h and Car.cpp.

You need to submit both header file and cpp file for each question.

All cpp files must contain your name, student-id, and assignment # on the top of the file in the comments. Place all your .cpp files (only) in a folder named your ROLLNUM_SECTION (e.g. 21i-0001_A). Compress the folder as a zip file and upload on google classroom.

Deadline: The deadline to submit the assignment is 23rd April, 2022 at 11:59 PM. Correct and timely submission of the assignment is the responsibility of every student.

Plagiarism: This is an individual assignment; any kind of plagiarism will result in a zero.

Restrictions: All functions will be out-of-line.
Evaluation: The assignment is of 200 marks. All submissions will be evaluated on test cases similar to the ones provided. Failure of a test case would result in zero marks for that part.

Question # 1 Rational

A number that can be made by dividing two integers (an integer is a number with no fractional part) is called a rational number. The word comes from "ratio".

Your goal is to overload the operators for a generic “Rational” class. The Rational class will support mixed types in expressions e.g. 0.5 as “1/2”, 0.25 as “1/4” etc. You will need to write two files (Rational.h and Rational.cpp). Your implemented class must fully provide the definitions of following class (interface) functions. You are required to implement the concepts related to operator overloading learned during the course.

```
class Rational {
// think about the private data members
public:
Rational(int n=0,int d=1);
Rational(const Rational &copy);// copy constructor to initialize the Rational form existing
Rational //object

// Binary Operators
// Assignment Operator
Rational operator = (const Rational &x);
// Arithmetic Operators
Rational operator+(const Rational &x) const;
Rational operator-(const Rational &x) const;
Rational operator*(const Rational &x) const;
Rational operator/(const Rational &x) const;
// Compound Arithmetic Operators
Rational operator += (const Rational &x);
Rational operator -= (const Rational &x);
Rational operator *= (const Rational &x);
Rational operator /= (const Rational &x);
// Logical Operators
bool operator == (const Rational & other) const;
bool operator < (const Rational & other) const;
bool operator > (const Rational & other) const;
bool operator <= (const Rational & other) const;
bool operator >= (const Rational & other) const;

// Unary Operator
// Conversion Operator
operator string() const; // returns 2/3 as “2/3”. If the denominator is 1 then only the
numerator is returned, i.e. for 2/1 the operator shall return “2”
~Rational(); // destructor
};

// Operator Overloading as Non-Member Functions
// Stream Insertion and Extraction Operators
ostream& operator<<(ostream& output, const Rational &);    // outputs the Rational
istream& operator>>(istream& input, Rational&);     // inputs the Rational
```

Question # 2 String

Your goal is to overload the operators for the "**String**" class that you have implemented in your first assignment. You will need to write two files (String.h and String.cpp). Your implemented class must fully provide the definitions of following class (interface) functions.

```
class String {
 // think about the private data members
public:
// provide definitions of following functions
String(); // default constructor
String(const char *str); // initializes the string with constant c-string
String(const String &); // copy constructor to initialize the string from the existing
string
String(int x); // initializes a string of predefined size

// Binary Operators
// Sub-script Operators
char &operator[](int i); // returns the character at index [x]
const char operator[](int i) const; // returns the character at index [x]
// Arithmetic Operators
String operator+(const String &str) const; // appends a String at the end of the String
String operator+(const char &str) const; // appends a char at the end of the String
String operator+(char *&str) const; // appends a String at the end of the String
String operator-(const String &substr) const; //removes the substr from the String
String operator-(const string &substr) const; //removes the substr from the String
// Assignment Operators
String& operator=(const String&); // copies one String to another
String& operator=(char*); // copies one c-string to another
String& operator=(const string&); // copies one string to another
// Logical Operators
bool operator==(const String&) const; // returns true if two Strings are equal
bool operator==(const string&) const; // returns true if the string is equal to the String
bool operator==(char *) const; // returns true if the c-string is equal to the String

// Unary Operators
// Boolean Not Operator
bool operator!(); // returns true if the String is empty
// Function-Call Operators
int operator()(char) const; // returns the index of the character being searched
int operator()(const String&) const; // returns the index of the String being searched
int operator()(const string&) const; // returns the index of the string being searched
int operator()(char *) const; // returns the index of the c-string being searched
// Conversion Operator
operator int() const; // returns the length of string
~String(); // destructor
};

ostream& operator<<(ostream& output, const String&); // outputs the string
istream& operator>>(istream& input, String&); // inputs the string
```

Question #3 Array

Your goal is to overload the operators for "Array" class. You will need to write three files (array.h, array.cpp). Your implemented class must fully provide the definitions of following class (interface) functions. Please also write down the test code to drive your class implementation. Please note that we will be running your code against our test code and any segmentation faults or incorrect result will result in loss of marks.

```
class Array{
// think about the private data members...
public:
// provide definitions of following functions...
    Array(); // a default constructor
    Array(int size); // a parametrized constructor initializing an Array of predefined
size
    Array(int* arr, int size); // initializes the Array with a existing Array
    Array(const Array &); // copy constructor
    int& operator[](int i); // returns the integer at index [i] after checking the out
of range error
    int& operator[](int i) const;
    const Array & operator=(const Array&); //copy the array
    Array operator+(const Array&); //adds two Array
    Array operator-(const Array&); //subtracts two Array
    Array operator++();// adds one to each element of Array
    Array operator++(int);// adds one to each element of Array
    Array& operator--(int);// subtracts one from each element of Array
    bool operator==(const Array&)const; // returns true if two arrays are same
    bool operator!(); // returns true if the Array is empty
    void operator+=(const Array&); // adds two Array
    void operator-=(const Array&); // subtracts two Array
    int operator()(int idx, int val); // erases the value val at idx. Returns 1 for a
successful deletion and -1 if idx does not exist or is invalid. Shift the elements after
idx to the left.
    ~Array(); // destructor...
};
// Operator Overloading as Non-Member Functions
// Stream Insertion and Extraction Operators
ostream& operator<<(ostream& input, const Array&); //Inputs the Array
istream& operator>>(istream& output, Array&); //Outputs the Array
```

Question #4 BinaryStore Calculator

A BinaryStore calculator will store bytes "stored in strings" with their addresses, i.e., at each address in the BinaryStore there is a stored Byte. Each address will be 4 characters string and each byte will be 8 characters strings.
Create a class **BinaryStore** and class **Byte** that offer following overloaded operators.
class offers the following methods:

- an overloaded +=operator that will add the address in the list of BinaryStore
- an overloaded + that will add two string bytes for Byte Class
- an overloaded - that will subtract two string bytesByte Class
- an overloaded || operator that will give the string which is bit by bit logical or of act two string bytes for Byte Class
- an overloaded && operator that will give the string which is bit by bit logical and of act two string bytesfor Byte Class
- an overloaded == operator that will give bool value if they are equal or notfor Byte Class

furthermore, implement all the constructors and member functions required against following sample code.

```
int main()
{
// address will be only 4 bit long for this problem. And bytes will be only 8 bit long
// create a 10 locations binary store for storing addresses and values
        BinaryStore b1(10);
        b1+="0011"; // add a new address to the store
        b1.Add("0011",Byte("00000010")); // add the byte at newly added address
        cout<<b1;
        b1+="0110";
        b1.Add("0110",Byte("00000110"));
        b1+="1010";
        Byte nb=b1.Get("0011")+b1.Get("0110"); // add two bytes, bit by bit
        b1.Add("1010",nb);
        Byte nb2=b1.Get("1010")+b1.Get("0110");
        bool r=b1.Get("0011")==b1.Get("0110");
        cout<<endl;
        cout<< "Equal = " << r << endl;
        cout<<b1;
        Byte nb3=b1.Get("1010");
        bool r1=nb3==b1.Get("1010");
        cout<<endl;
        cout<< "Equal = " << r1 << endl;

        Byte nb4=b1.Get("0011") || b1.Get("1010"); // byte1 OR byte2 , bit by bit
        Byte nb5("00001100");
```

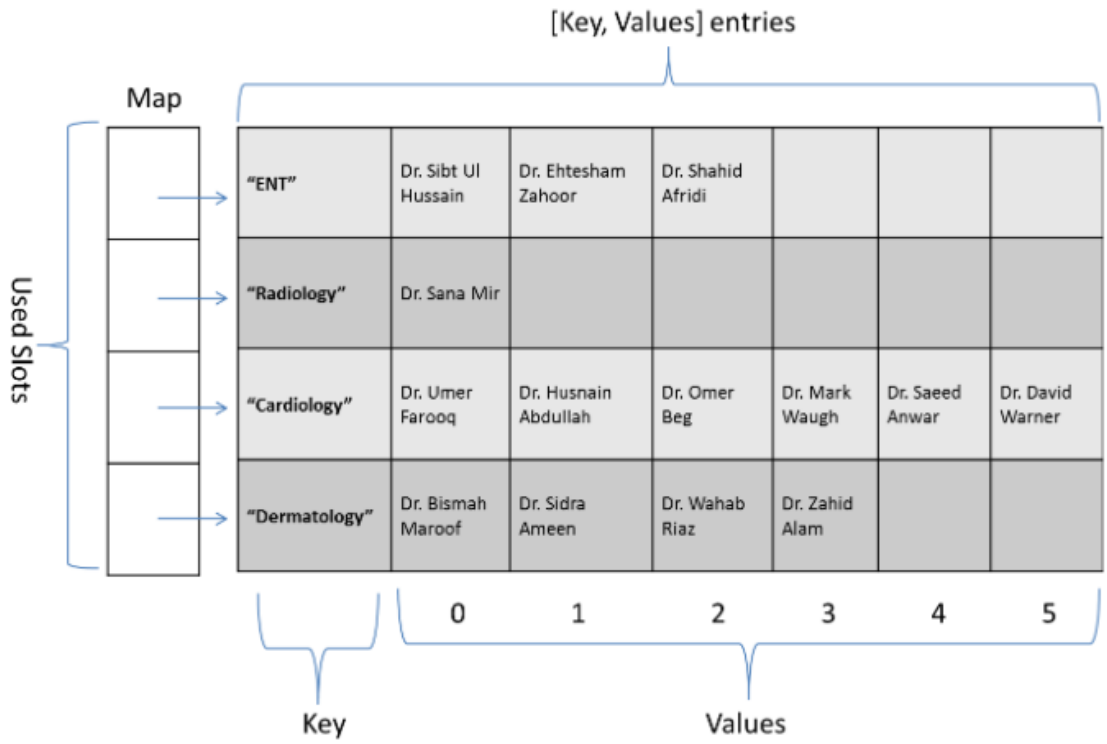
```
        Byte  nb6= nb4 && nb5;// byte1 AND byte2 , bit by bit
        b1+="1011";
        b1.Add("1011",nb6);
        cout<<b1;

}
```

```
output:
0011 00000010
Equal = 0
0011 00000010
0110 00000110
1010 00001000
Equal = 1
0011 00000010
0110 00000110
1010 00001000
1011 00000100
```

Note: Do provide readme.txt file for this question

Question # 5

Consider the following figure



We need a map (or dictionary) to store data about medical practitioners. A map consists of multiple [key, values]entries, where extra space is dynamically allocated when needed. Typically multiple values may be mappedagainst one key. For simplicity, we assume that maximum six (06) values can be stored against one key (as shownin figure). Here key is the specialty of medical practitioner, and values are names of medical practitioners.You are required to design and implement map class (and any other required classes) in C++, where followingcode should run without any syntax/logical errors :

- an overloaded += operator
- an overloaded -= operator

```
void main()
{
    CMap map, map2, map3; // assume CMap is the name of map class

    // operator[] is used to get "entry" against given "key" from map and
```

```
    // can be used in following context with operator= to create/refresh an entry
    map["ENT"] = "Dr. Sibt Ul Hussain";
    map["Radiology"] = "Dr. Sana Mir";
    map2["Cardiology"] = "Dr. Umer Farooq";

    // operator+= with an "entry" is used to add new "value" against a given "key".
    // if entry with given "key" does not exist then new entry is created in map.
    map["ENT"] += "Dr. Ehtesham Zahoor";
    map["ENT"] += "Dr. Shahid Afridi";
    map2["Cardiology"] += "Dr. Husnain Abdullah";
    map2["Cardiology"] += "Dr. Omer Beg";
    map2["Cardiology"] += "Dr. Mark Waugh";
    map2["Cardiology"] += "Dr. Saeed Anwar";
    map2["Cardiology"] += "Dr. David Warner";

    // following statement creates new entry in map, and then add values to it
    map2["Dermatology"] += "Dr. Bisma Maroof";
    map2["Dermatology"] += "Dr. Sidra Ameen";
    map2["Dermatology"] += "Dr. Wahab Riaz";
    map2["Dermatology"] += "Dr. Zahid Alam";

    // operator+ is also used to add two maps and return a new map.
    // operator= is used to assign one map to other(deep copy).
    map3 = map + map2; // map3 now looks like the map in figure above!

    // operator-= is used to remove value from the map against key = ENT.
    // If value does not exist, then nothing changes in the map. If a "key"
    // has no remaining value against it, then the "key" must also be removed.
    map["ENT"] -= "Dr. Ehtesham Zahoor";
}
```

In addition to the above operators, you also need to make a function named toString(), which converts the whole map into a string. Following are the examples demonstrating the output of the toString() function on map, map2, and map3 respectively:

map:

[ENT : { Dr. Sibt Ul Hussain, Dr. Shahid Afridi }, Radiology : { Dr. Sana Mir }]

map2:

[Cardiology : { Dr. Umer Farooq, Dr. Husnain Abdullah, Dr. Omer Beg, Dr. Mark Waugh, Dr. Saeed Anwar, Dr. David Warner }, Dermatology : { Dr. Bisma Maroof, Dr. Sidra Ameen, Dr. Wahab Riaz, Dr. Zahid Alam }]

map3:

[ENT : { Dr. Sibt Ul Hussain, Dr. Ehtesham Zahoor, Dr. Shahid Afridi }, Radiology : { Dr. Sana Mir }, Cardiology : { Dr. Umer Farooq, Dr. Husnain Abdullah, Dr. Omer Beg, Dr. Mark Waugh, Dr. Saeed Anwar, Dr. David Warner }, Dermatology : { Dr. Bisma Maroof, Dr. Sidra Ameen, Dr. Wahab Riaz, Dr. Zahid Alam }]

Note: The evaluation of Q5 will be based on the toString() function as the testcases will work on the output of this function. Therefore, make sure to take care of spaces and punctuations, as well as the function signature: `string toString();`