National University of Computer and Emerging Sciences

School of Computing **Spring 2022** **Islamabad Campus**

CS-1004: Object-Oriented Programming

Assignment 2

(Deadline: 25th March, 2022 11:59 PM)

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

Implement each question in the created .cpp file, you will also need to include the header in your cpp file. **DO NOT change anything in the header file except for declaring data members.**

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 25th March, 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: The use of libraries such as String, cmath is NOT allowed. DO NOT change the function headers/prototypes in the header files or the test cases. 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 10 marks

Implement a structure, **Car**. The structure has the following data member:

1. **int petrolLevel** – that indicates the liters of petrol in the car's petrol tank. PetrolLevel for this car can only be in the range 0 – 45 liters.

The structure has the following member functions:

1. **void setPetrolLevel(int petrolLevelVal)** – a setter for petrol level, cannot set value greater than 45.
2. **int getPetrolLevel()** – a getter for petrolLevel
3. **Car()** – a default constructor
4. **Car(int petrolLevelVal)** – a parametrized constructor
5. **bool MoveCar(int distanceKM)** – a function that takes an integer as an argument and moves the car to the argument value which is in Km. Provided the petrol tank has enough fuel. Successful movement of the car returns true otherwise returns false. Moving for each km causes the petrolLevel to go low by one.
6. **void Refill()** – a function that refills the car tank to the maximum value of petrolLevel
7. **bool isEmpty()** – a function that tells whether the Petrol tank is Empty or not

Question 2 15 marks

Implement a class **Sequence** to store a sequence of non-negative integer values, and the length of the sequence. The class has the following private data members:

1. **int length** – the length of the sequence
2. **int *pseq** – a pointer to a dynamic integer array holding sequence of integers

The class shall provide the following public methods:

1. **Sequence()** – a default constructor that initializes length to 10 and store the sequence of all zeros in an array.
2. **Sequence(int lengthVal, int n1=0,int n2=0,int n3=0, int n4=0, int n5=0, int n6=0, int n7=0, int n8=0, int n9=0, int n10=0)** – another parameterized constructor should initialize the length and array to sequence values passed in the arguments.
3. **Sequence(Sequence &s)** – a copy constructor that creates a copy of a Sequence object.
4. **int getLength()** – a getter for length

5. **int* getSeq()** – a getter for the sequence of numbers
6. **void Sort(int n)** – a function that sorts the first **n** elements in the sequence array. You cannot use Bubble Sort Algorithm
7. **int RemoveDuplicates()** – a function that removes duplicates from the sequence in-place (without creating another array). Since you cannot change the size of the array or create a new one, you must place the result in the **first part** of the array.
   If there are **n** unique elements in the array, then the first **n** elements of array should hold the final result sorted in ascending order. It does not matter what you leave beyond the first **n** elements in the array.

   Return **n** after placing the final result in the first k elements of the array.

   Example

   ```
   Seq array = {3,1,2,2,1,7}

   n = 4

   Sequence array after duplicate removal = {1,2,3,7,dk,dk}

   dk = don't care about values here
   ```

8. **void Rotate(int steps)** – a method that rotates the sequence elements clockwise for the given steps

   Example

   ```
   Seq array = {1,3,4,6,2,6,0}

   steps = 3

   Sequence array after rotation = {2,6,0,1,3,4,6}
   ```

9. **~Sequence()** – a destructor to deallocate the dynamically created array

Question 3 15 marks

A restaurant serves five different kinds of sandwiches.

Implement a class **Sandwich** that includes the following private data members:

1. **char *name –** a sting to store the sandwich name

2. **char *filling –** a string to store the sandwich filling
3. **char *size –** a string for the size of the sandwich (can only be small, medium or large)
4. **bool is_ready –** a boolean to store the status of the sandwich, is it ready or not
5. **double price** - a double to store the price of the sandwich.

The class shall provide the following public methods:

1. **Sandwich()** – a default constructor
2. **Sandwich(char *fillingVal, double priceVal)** – a parametrized constructor
3. **Sandwich(char *fillingVal, double priceVal, char* nameVal, char* sizeVal, bool ready_status)** – a parametrized constructor
4. **Sandwich(const Sandwich &sandwich)** – a copy constructor
5. **void setFilling(char *fillingVal)** – setter for filling
6. **void setPrice(double priceVal)** – setter for price
7. **void setName(char *nameVal)** – setter for name
8. **void setSize(char *sizeVal)** – setter for size
9. **char* getFilling()** – getter for filling
10. **double getPrice()** - getter for price
11. **char* getName()** – getter for name
12. **char* getSize()** – getter for size
13. **void makeSandwich()** – function to make sandwich (check if filling is not NULL then set value of is_ready to true)
14. **bool check_status()** - function to check if sandwich is ready or not

Question 4 30 marks

Implement a structure **Address** with the following data members:

1. **char* address** – a sting to store the address
2. **char* city** – a sting to store the city name
3. **char* state** – a sting to store the state name
4. **int zip_code** – an integer to store the ZIP code

Implement a structure **CustomerAccount** with the following data members:

1. **char* name** – a string to store the name of the customer
2. **Address address** – a nested structure variable to hold the address
3. **long long phoneNum** – a long long to store the phone number of the customer

4. **float balance** – a float to store the customer's balance
5. **char* accountNum** – a string to store the customer's account number, account numbers are in the for "PK001" to "PK100".

The following two will be passed as arguments to all the global functions mentioned below.

1. **CustomerAccount *customers[100]** – an array of 100 CustomerAccount pointers.
2. **int accountsOpen** – an integer to store the current accounts open and can also be used as an index for the customer's array.

Implement the following functions in global scope:

1. **void OpenCustomerAccount (CustomerAccount* customers[], int& accountsOpen, char* NameVal, char*addVal, char*cityVal, char*stateVal, int zipcodeVal, long long phoneVal, float balanceVal)** – a function to create a new customer account and assign it an account number that has not already been taken between PK001 and PK100.
   To dynamically create a new customer account, use an element in the customers array (a pointer) with the *new* keyword.
2. **int SearchCustomer (CustomerAccount* customers[], int accountsOpen, char* accountNum)** – a function that searches for a customer using an account number. If the customer is found it returns the array index otherwise return -1
3. **bool UpdateCustomerAccount (CustomerAccount* customers[], int accountsOpen, char *accountNumVal, Address addressVal)** – an overloaded function that updates a customer's address
4. **bool UpdateCustomerAccount (CustomerAccount* customers[], int accountsOpen, char *accountNumVal, long long phoneVal)** – a function that updates a customer's phone number
5. **bool UpdateCustomerAccount (CustomerAccount* customers[], int accountsOpen, char *accountNumVal, float balanceVal)** – a function that updates a customer's balance
6. **void DisplayAllCustomers(CustomerAccount* customers[], int accountsOpen)** – a function that displays all customer accounts in the bank

Question 5 20 marks

A program calculates performance-based salary of customer support representatives (CSRs) in an organization.

Implement a class **CSR** (Customer support representative) with the following private data members:

1. **int csrID** – a integer to hold employee identification numbers between 1 and 7.
2. **char *csrName** – a string to hold the name of each CSR
3. **int hours** – an integer to hold the number of hours worked by each CSR
4. **int complaintsResolved** – an integer to hold the number of complaints successfully resolved by each CSR
5. **float payrate** – a float to hold the employee's hourly pay rate, where payRate is calculated as following:

   payRate = $25 + 25*(complaintsResolved by each CSR/total complaints resolved)
6. **float wage** – a float to hold the employee's wage, calculated as

   wages = hours * payrate
7. **static int totalComplaintsResolved** - a static integer to hold the total number of complaints resolved by all CSRs

The class should implement the following public methods:

1. **int getCSRID()** – a getter for CSRID
2. **char* getName()** – a getter for employee's name
3. **int getHours()** – a getter for hours worked
4. **int getComplaintsResolved()** – a getter for complaints resolved by the employee
5. **float getPayrate()** – a getter for payrate of the employee
6. **float getWage()** – a getter for the employee's wage

7. **void setCSRID(int ID)** – a setter for CSRID
8. **void setName(char* n)** – a setter for name
9. **void setHours(int hours)** – a setter for hours
10. **void setComplaintsResolved(int cpsResolved)** – a setter for complaintsResolved
11. **static void setTotalCpsResolved(int totalCpsResolved)** – a setter for totalComplaintsResolved
12. **void calcPayrate()** – a function to calculate the employees payrate
13. **void calcWage()** – a function to calculate the employee's wage
14. **static int getTotalCpsResolved()** – a static function to get total complaints resolved

Implement the following functions in global scope. All functions will take an array of CSR objects as an argument.

e.g. CSR employees[7] – an array of 7 CSR objects

Functions:

1. **CSR getCSR_at(CSR employees[7], int index)** – returns the CSR object at the given array index
2. **void calcTotalComplaints (CSR employees[7])** – a function that sums the complaints of resolved by all employees and assigns to the *totalComplaintsResolved* variable.
3. **void calcAllEmployeeWages(CSR employees[7])** – a function that calculates the wages of all employees
4. **void SortByHours(CSR employees[7])** – sorts employees in descending order based on hours worked.
5. **void SortByComplaintsRes(CSR employees[7])** – sorts employees in descending order based on complaints Resolved
6. **void SortByWages(CSR employees[7])** – sorts employees in descending order based on wages.

Note: Ensure all necessary input validation, for example, hours, complaintsResolved cannot be negative numbers. Also assume, no two CSRs will have the same number of hours, complaints Resolved and wages

Question 6 50 marks

Implement your own string class using only primitive data types. Your string class should contain some of the same functionality as the string class in C++.

Your class should have the following member variables:

1. **char *data**: holds the character array that constitutes the string
2. **int length**: the current length of the string that the object contains (number of characters in data)

Your object should have the following constructors and destructor:

1. **String():** default constructor

2. **String(int size)**: alternate constructor that initializes length to size and initializes data to a new char array of size.
3. **String(char* str)**: alternate constructor that initializes length to size of str, initializes data to a new char array of size length and fills the contents of data with the contents of str.
   Note: You can assume that any character array that is sent as str will be terminated by a null character ('\0')
4. **String(const String &str)**: copy constructor
5. **~String()**: destructor in which you should delete your data pointer.

Your class should have the following functions:

1. **int strLength()**:returns the length of the string in the object
2. **void clear()**: should clear the data in your string class (data and length)
3. **bool empty()**: this method should check to see if data within the String class is empty or not.
4. **int charAt(char c)**: this method returns the first index at which this character occurs in the string
5. **char* getdata():** a getter to get the actual string
6. **bool equals(char*str):** this method compares if the data in the calling string object is equal to str.
7. **bool equalsIgnoreCase(char* str):** this method compares the calling string object data with the data in str without considering the case.
8. **char* substring(char* substr, int startIndex)** : this method should search for substr in data within the calling String object starting from the startIndex. The method should search for the substring from the startIndex and return the substring from the position found till the end. For example, if you had the string "awesome" and you wanted to find the substring 'es' starting from the beginning of the string (startIndex = 0). Your function should return "esome". Returns NULL if substring is not found.
9. **char* substring(char* substr, int startIndex, int endIndex)** : this method should search for substr in data within the calling String object between the startIndex and endIndex. For example, if you had the string "awesome" and you wanted to find the substring 'es' starting from startIndex=2 and endIndex=5. Your function should return "esom". Returns NULL if substring is not found.
10. **void print()**: a function that will output the contents of the character array within the object. If the contents of the String are empty (length == 0) then you should print "NULL".

Additional Specifications:

- Your program should have NO memory leaks
- You should not use the built-in C++ string class anywhere in your object.
- You must appropriately use the const keyword (you need to decide which functions will be constant qualified and which will not).
- You may define additional helper functions as needed.
- You will need to submit two files: String.h and String.cpp.

Question 7 60 marks

Implement a class **Matrix** to create matrices of 2x2, 3x3 and 4x4 with the following private data members:

1. int **matrix – a double pointer of type integer to create a 2D array (matrix)
2. int row – an integer to store the rows in the matrix
3. int col – an integer to store the columns in the matrix

The class shall have the following public member functions:

1. **Matrix (int n1, int n2, int n3, int n4, int row = 2, int col = 2)** – a parametrized constructor for a 2x2 matrix
2. **Matrix (int n1, int n2, int n3, int n4, int n5, int n6, int n7, int n8, int n9, int row = 3, int col = 3)** – a parametrized constructor for a 3x3 matrix
3. **Matrix (int n1, int n2, int n3, int n4, int n5, int n6, int n7, int n8, int n9, int n10, int n11, int n12, int n13, int n14, int n15, int n16, int row = 4, int col = 4)** – a parametrized constructor for a 4x4 matrix
4. **Matrix(const Matrix &m)** – a copy constructor
5. **~Matrix()** – a destructor
6. **int getRow()** – a getter for rows

7. **int getCol()** – a getter for columns
8. **int getValue(int row, int col)** – a function to get the value at the given row and column of a matrix
9. **void setValue(int row, int col, int value)** – a function to set the value at a given row and column index of a matrix.
10. **int Total()** - Calculate the total/sum of all the values in the matrix.
11. **double Average()** - Calculates average of all the values in the array.
12. **int RowTotal(int row)** - Calculates total/sum of the values in the specified row.
13. **int ColumnTotal(int col)** - Calculates total/sum of the values in the specified column.
14. **int HighestInRow(int row)** - Finds highest value in the specified row of the array.
15. **int LowestInRow(int row)** - Finds lowest value in the specified row of the array.
16. **Matrix Transpose()** - Find Transpose of array.

17. **int LeftDiagonalTotal()** - Calculates total/sum of the values in the left Diagonal of array.
18. **int RightDiagonalTotal()** - Calculates total/sum of the values in the right Diagonal of array.
19. **Matrix Add(Matrix m**) – adds the calling object matrix with the one passed as an argument and returns a resultant matrix
20. **Matrix Subtract(Matrix m**) – subtracts the calling object matrix with the one passed as an argument and returns a resultant matrix
21. **Matrix Multiply(Matrix m**) – multiplies the calling object matrix with the one passed as an argument and returns a resultant matrix
22. **int FindkSmallest(int k)** – a function that finds the k^{th} smallest element in the matrix. If k is 1, return the smallest element. If k is 2 return the second smallest element and so on.
23. **int FindkLargest(int k)** – a function that finds the k^{th} largest element in the matrix. If k is 1, return the largest element. If k is 2 return the second largest element and so on.
24. **Matrix merge(Matrix m)** - a function to merge two matrices. E.g.

$$m = \begin{Bmatrix} 1, 2, 3, 4 \\ 9, 8, 7, 6 \end{Bmatrix} \text{ and } n = \begin{Bmatrix} 5, 6, 7, 8 \\ 0, 0, 0, 0 \end{Bmatrix}$$

$$\text{after merge} = \begin{Bmatrix} 1, 2, 3, 4, 5, 6, 7, 8 \\ 9, 8, 7, 6, 0, 0, 0, 0 \end{Bmatrix}$$

Good Luck!