National University of Computer and Emerging Sciences

School of Computing **Summer 2023** **Islamabad Campus**

CS-1004: Object-Oriented Programming

Assignment 2

(Deadline: 15th July, 2023 11:59 PM)

Submission: 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. 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 15th July, 2023 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 160 marks.

Q1: Implementation of Family Tree – Create a class Person that should store following attributes of a person **(70 Marks)**

char* name --- name of person
char gender --- gender as ‘M’, ‘F’
Date dob --- an object of Date class

int noOfChildren – total no. of children registered in familyTree so far. Upon each registration this should be incremented by 1

public: Person **children=new Person*[3] – a double pointer pointing to an array of pointers that point to each registered child. Assume that a family tree can register 3 children per person at max.

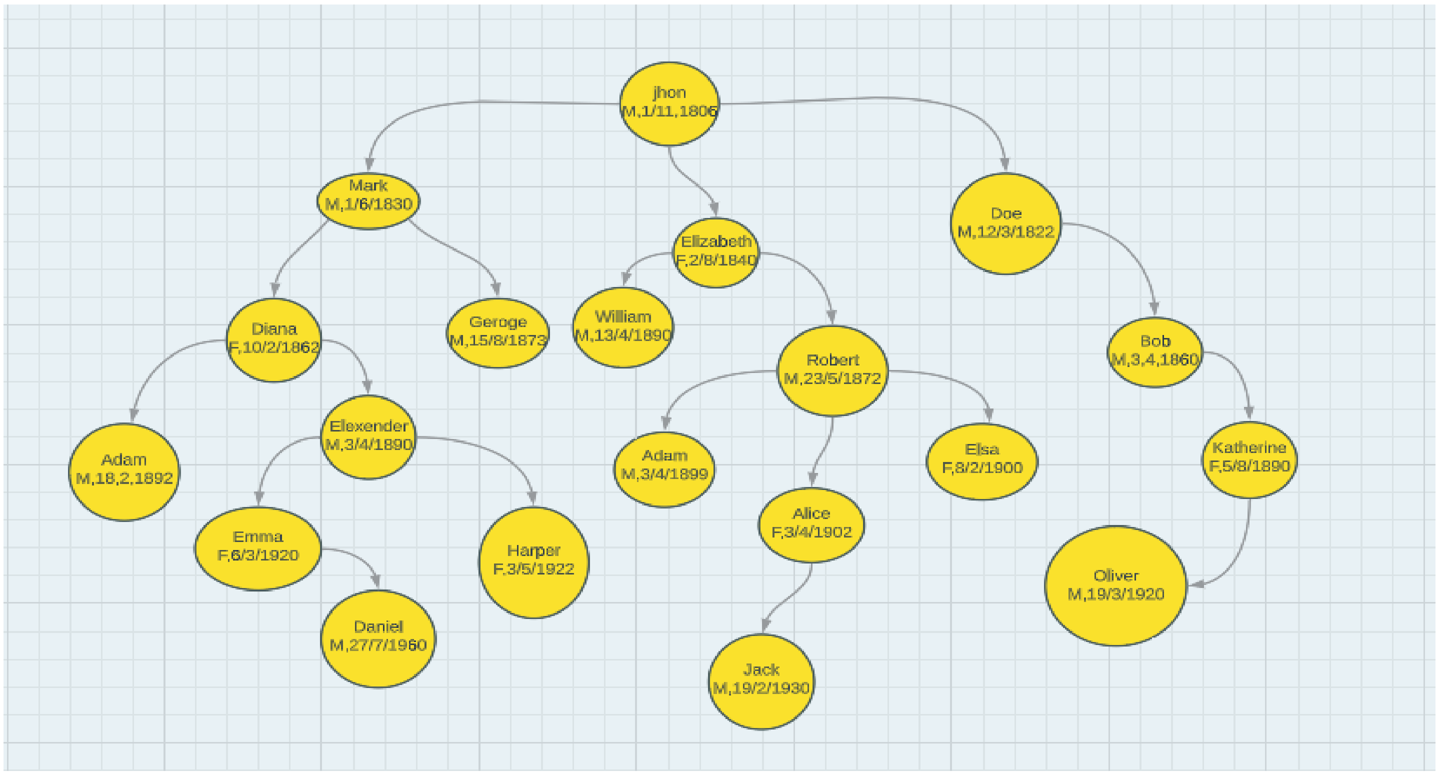
```
class Person{
private:
// think about the private data members...
public:

// provide definitions of following functions...
Person ();// a default constructor
Person(string n, char g, int day, int month, int year);  // a parameterized constructor
//implement mutators for all private data members
//implement accessors for all private data members
//you have to implement the following functions
void displayData();// prints data members
int calcAge(); // subtracts the date of birth from current date (can be taken from user) and
display age in years
 ~Person(); // delete dynamic element
};
```

NADRA is a national database that keeps records of nationals. Family Registration Certificate (FRC) is a mean of being identified with your NADRA’s record. This provides the family composition. You’re required to design a FamilyTree class that keeps record of a person, his children, grandchildren and so on. Each FamilyTree must have an ancestor or forefather who got registered for the first time. The core responsibility of a familyTree class is to register 3 or less children of each person, displaying the data of whole tree , finding other facts about the

family.
Your implemented class must fully provide the definitions of following class (interface) functions.
Please note that you are required to make a menu driven code.

class FamilyTree {
private:
Person* foreFather;
// the first person of a
family
public:
// provide definitions of following functions...
FamilyTree (Person* foreFather);// no default constructor as tree always starts from an ancestor
//implement mutators for all private data members
//implement accessors for all private data members
//you have to implement the following functions

void registerChild(Person &p, Person &child) // a child is always registered against a parent. This function will add an element child to the children array in person p and update the total no of children of person accordingly
Person findYoungestChildOf(Person &p) // a youngest child is the one with lowest age among siblings. This should call calculate age function to get age of each child. For example the youngest child of Robert is Alice as shown in a sample tree below.
void displayFamilyOf(Person p) // this should only display information about all children of a person p
Person* FindEldestGrandsonOf(Person *grandfather) // the function should traverse a tree 2 levels down the grandfather to find grandsons. The eldest among all is the one with greatest age.For example, the eldest grandson of Mark is Elexender as shown in a sample tree below.
void displayTree(Person *p) // the function should traverse the whole tree recursively starting from where person p exists down till the last level. On each level it should first display the father name followed by children names. No loops are allowed in this function
~ FamilyTree (); // delete dynamic element
};



Q2-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++. **(40 Marks)**

String{

private:

char *data;// holds the character array that constitutes the string
int length;// the current length of the string that the object contains (number of characters in data)

public:

String(); //default constructor

String(int size); //alternate constructor that initializes length to size and initializes data to a new char array of size.

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')*/

String(const String &str); // copy constructor

~String(); // destructor in which you should delete your data pointer.

int strLength(); //returns the length of the string in the object

void clear(); // should clear the data in your string class (data and length)

bool empty(); //this method should check to see if data within the String class is empty or not.

int charAt(char c); //this method returns the first index at which this character occurs in the string

char* getdata() ; //a getter to get the actual string

bool equals(char*str);
//this method compares if the data in the calling string object is equal to str.

bool equalsIgnoreCase(char* str); //this method compares the calling string object data with the data in str without considering the case.

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.*/

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*/

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.

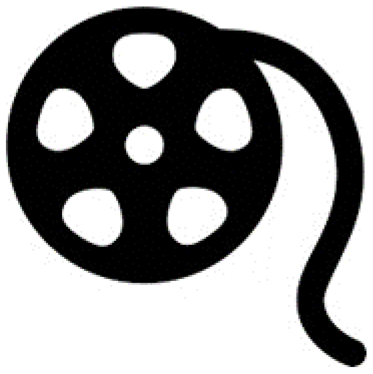
• You must appropriately use the const keyword (you need to decide which functions will be constant qualified and which will not).

• You may define additional helper functions as needed.

• You will need to submit two files: String.h and String.cpp.

Q3: Using an electric pulley is a convenient way to lift heavy luggage. It consists of a rope attached to one end and a weight on the other side. By moving anti-clockwise, the pulley effortlessly raises objects to the roof. The rope forms circles around the pulley, whose size depends on the pulley's circumference or the rope's thickness. Assuming a narrow groove in the pulley, only circles of equal size can fit. **(20 Marks)**

Exploring the relationship between the rope's length, thickness, and the pulley's circumference helps determine the number of rings formed during the pulley's movement. Input values, such as 5 100 2 (5cm initial pulley radius, 100cm rope length, and 2cm rope thickness), reveal that each revolution increases the pulley's radius by 5+2, enlarging the circle's circumference.



1. Define a class Pully consisting of all relevant properties, along with functionalities.
2. Define a class Rope having relevant states and behaviors.
3. Make a class Rotation to implement the functionality of determining the number of rings that the rope will create when moving the pully.
4. Make all data members privately accessible.
5. Provide getter/setter functions and parameterized constructors in each class.

Sample input	**Sample Output**
5 100 2 4 400 2	2 6

Hint: Calculate circumference using the radius and each circle of rope adds +2 in the latest radius value.

Q4: A restaurant serves five different kinds of sandwiches. **(30 Marks)**
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

---