National University of Computer and Emerging Sciences

School of Computing **Spring 2023** **Islamabad Campus**

CS-1004: Object-Oriented Programming

Assignment 2 [110 Marks]

(Deadline: 24th March, 2023 11:59 PM)

Instructions:

1. Assignments are to be done individually. You must complete this assignment by yourself.

You cannot work with anyone else in the class or with someone outside of the class. The code you write must be your own and you must understand each part of coding.

2. The AIM of this assignment is to practice with structures and classes.

3. Plagiarism of any kind (copying from others and copying from the internet, etc.,) is not allowed.

4. Please make CPP file for each question.

National University of Computer and Emerging Sciences

School of Computing **Spring 2023** **Islamabad Campus**

Question # 01 [15 Marks]:

There is a class called **Image**, which is used to represent a digital image with color pixels. The class contains private member variables row, col, depth, and image. row and col represent the dimensions of the image, while depth represents the number of color channels in the image(number of 2d arrays). The image member is a three-dimensional array of Pixel structs, where each struct contains three integers representing the red, green, and blue values of a pixel.

The class includes several member functions that allow for manipulation of the image data. These functions include a **constructor** for creating a new Image, a **copy constructor** for copying an existing Image, a **destructor** for deallocating memory, and various **getter** and **setter** functions for accessing and modifying the pixel data.

The class also includes several image processing functions such as **fill** for filling the entire image with a given color, **clear** for setting all pixels to black, **getAverageBrightness** for calculating the average brightness of the image, **getMaximumBrightness** for finding the maximum brightness value in a specific color channel, **countBrightPixel** for counting the number of bright pixels in the image, and **transposePixel** for transposing the image data along a specific color channel.

Overall, the Image class provides a simple and intuitive way to represent and manipulate digital images in a C++ program. Implement with the following data members.

- int **row**: stores the number of rows in the image.
- Int **col**: stores the number of columns in the image.
- int **depth**: stores the number of color channels in the image.
- Pixel*** **image**: a 3-dimensional array of Pixel objects representing the actual image data.

The class should have the methods:

- **Image**(int d=1, int r=1, int c=1): creates a new Image object with the specified number of color channels (d), rows (r), and columns (c). By default, creates a 1x1x1 image with a single color channel.
- **Image**(const Image& img): creates a new Image object that is a copy of the specified Image object.
- ~**Image**(): frees any dynamically allocated memory used by the Image object.
- int **getRow**(): returns the number of rows in the image.
- int **getCol**(): returns the number of columns in the image.
- int **getDepth**(): returns the number of color channels in the image.
- Pixel **getPixel**(int x, int y, int z): returns the Pixel object at the specified x, y, and z coordinates in the image.
- void **setPixel**(int x, int y, int z, Pixel p): sets the Pixel object at the specified x, y, and z coordinates in the image to the specified Pixel object.

- void **fill**(Pixel p): fills the entire image with the specified Pixel object.
- void **clear**(): fills the entire image with black (i.e., Pixel(0, 0, 0)).
- double **getAverageBrightness**(): calculates and returns the average brightness value of all the pixels in the image.
- int **getMaximumBrightness**(int depth): finds and returns the maximum brightness value of all the pixels in the specified color channel.
- int **countBrightPixel**(): counts and returns the number of pixels in the image that have at least one color channel with a brightness value of 255.
- void **transposePixel**(int depth): transposes the pixels in the specified color channel of the image.

Also draw the image on the screen using graphics library or system library.

Question # 02 [10 Marks]:

PizzaHut serves five different kinds of Pizzas.

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

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

The class shall provide the following public methods:

1. **Pizza()** – a default constructor
2. **Pizza(char *toppingVal, double priceVal)** – a parametrized constructor
3. **Pizza(char *toppingVal, double priceVal, char* nameVal, char* sizeVal, bool ready_status)** – a parametrized constructor
4. **Pizza(const Pizza &pizza)** – a copy constructor
5. **void setTopping(char *toppingVal)** – setter for topping
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* getTopping()** – getter for topping
10. **double getPrice()** - getter for price
11. **char* getName()** – getter for name
12. **char* getSize()** – getter for size
13. **void makePizza()** – function to make pizza (check if topping is not NULL then set value of is_ready to true)
14. **bool check_status()** - function to check if pizza is ready or not

National University of Computer and Emerging Sciences

School of Computing **Spring 2023** **Islamabad Campus**

Question # 03 [15 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 size: 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 length):** 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.
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 stringLength():**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 isEmpty():** 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 isEqual(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".

Question # 04 [20 Marks]:

struct **StudentAccount** – create a class **StudentAccount** with following data members

1. char* name
2. float annualInterestRate
3. double savingBalance
4. char* accountNum // the account numbers will be from "SA00" to "SA99". Each account must have a unique account number that has not been assigned to any other customer before.

The following two will be passed as arguments to some global functions mentioned below

1. **SavingAccount** *savers[100] – an array of 100 SavingAccount 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:

void OpenCustomerAccount (SavingAccount * savers[], int accountsOpen, char* NameVal, double balance) – a function to create a new account and assign it an account number.

float calculateMonthlyInterest (SavingAccount * saver) - that calculates the monthly interest by multiplying the balance by annualInterestRate divided by 12.

void modifyInterestRate(SavingAccount * saver, float newValue)

int SearchCustomer (SavingAccount * savers[], int accountsOpen, char* accountNum) – a function that searches for an account using an account number. If the customer is found it returns the array index otherwise return -1

bool UpdateAccountBalance (SavingAccount * savers[], int accountsOpen, char *accountNumVal, double balanceVal) – a function that updates a customer's balance

Question # 05 [20 Marks]:

Implementation of Integer Class – Your goal is to implement "Integer" class. Your implemented class must fully provide the definitions of following class (interface) functions.

class **Integer**{

int num;

string str;

public:

//include all the necessary checks before performing the operations in the functions

Integer();// a default constructor

Integer(int);// a parametrized constructor

Integer(String); // a parametrized constructor

void set(int);//set value

int get()const; //get value at (i,j)

int bitCount(); //Returns the number of one-bits in the 2's complement binary

int compareTo(Integer); //Compares two Integer objects numerically.

double doubleValue(); //Returns the value of this Integer as a double.

float floatValue(); //Returns the value of this Integer as a float.

Integer plus(const Integer); //adds two Integers and return the result

Integer minus(const Integer); // subtracts two Integers and return the result

Integer multiple(const Integer); //multiplies two Integers and return the result

Integer divide(const Integer); //divides two Integers and return the result

static int numberOfLeadingZeros(int i); /*Returns the number of zero bits preceding the highest-order ("leftmost") one-bit in the two's complement binary representation of the specified int value.*/

static int numberOfTrailingZeros(int i); /*Returns the number of zero bits following the lowest-order ("rightmost") one-bit in the two's complement binary representation of the specified int value.*/

static String toBinaryString(int i); //Returns string representation of i

static String toHexString(int i); //Returns string representation of i in base16

static String toOctString(int i); //Returns string representation of i in base 8

};

Question # 06 [15 Marks]:

Implement a class **Library** with the following private data members:

- char *bookTitle – a string to hold the title of the book
- char *author – a string to hold the name of the author

- int bookID – an integer to hold book identification numbers between 1 and 100.
- int quantity – an integer to hold the number of copies of the book in the library
- float price – a float to hold the price of each book
- static int totalBooks – a static integer to hold the total number of books in the library

The class should implement the following public methods:

- char* getBookTitle() – a getter for book title
- char* getAuthor() – a getter for author name
- int getBookID() – a getter for book ID
- int getQuantity() – a getter for number of copies of the book
- float getPrice() – a getter for the price of each book
- void setBookTitle(char* title) – a setter for book title
- void setAuthor(char* authorName) – a setter for author name
- void setBookID(int bookID) – a setter for book ID
- void setQuantity(int quantity) – a setter for number of copies of the book
- void setPrice(float price) – a setter for the price of each book
- static void setTotalBooks(int totalBooks) – a setter for total number of books
- void calcTotalPrice() – a function to calculate the total price of all copies of the book
- static int getTotalBooks() – a static function to get the total number of books in the library

Implement the following functions in global scope. All functions will take an array of Library objects as an argument. e.g. Library books[10] – an array of 10 Library objects

Functions:

- Library **getBook_at**(Library books[100], int index) – returns the Library object at the given array index
- void **addBook**(Library books[100], Library newBook) – adds a new book to the library array
- void **removeBook**(Library books[100], int bookID) – removes the book with the given book ID from the library array
- void **SortByTitle**(Library books[100]) – sorts the books in ascending order based on title
- void **SortByAuthor**(Library books[100]) – sorts the books in ascending order based on
  author name
- void **SortByPrice**(Library books[100]) – sorts the books in ascending order based on price
- bool **searchByTittle**(Library books[10], char* titlename) - returns true if the book with the titlename is found in the list
- Library **mostExpensiveBook**(Library books[10]) - returns the book with the most expensive price from the list

- Note: Ensure all necessary input validation, for example, quantity cannot be a negative number. Also assume, no two books will have the same book ID.

Question # 07 [15 Marks]:

Design a class called **Student** with the following private data members:

- stdID - an integer to hold the student's identification number
- Name - a string to hold the student's first name
- courseCodes - a dynamic array of strings to hold the course codes of the courses taken by the student
- numCourses - an integer to hold the number of courses taken by the student
- courseGrades - a dynamic array of integers to hold the grades obtained by the student in the courses taken
- gpa - a float to hold the grade point average of the student, calculated as the sum of all course grades divided by the number of courses taken
- The class should implement the following public methods:
- int getStdID() - a getter for the student's ID
- string getName() - a getter for the student's first name
- int getNumCourses() - a getter for the number of courses taken by the student
- string getCourseCode(int i) - a getter for the course code of the course taken at index i of the courseCodes array
- int getCourseGrade(int i) - a getter for the grade obtained by the student in the course taken at index i of the courseGrades array
- float getGPA() - a getter for the student's GPA
- void setStdID(int id) - a setter for the student's ID
- void setName(string firstName) - a setter for the student's first name
- void **setCourseGrade**(string courseCode, int grade) - a function to set the grade obtained by the student in the course with the given course code
- void **addCourse**(string courseCode, int grade) - a function to add a course with the given course code and grade to the student's record
- void **calcGPA**() - a function to calculate the student's GPA

Implement the following functions in global scope. All functions will take an array of Student objects as an argument. e.g. Student students[10] - an array of 10 Student objects

- Student **getStudentAt**(Student students[], int index) - returns the Student object at the given array index
- float **calcClassGPA**(Student students[], int numStudents) - calculates the average GPA of all students in the array
- float **getMaxGPA**(Student students[], int numStudents) - calculates the maximum GPA from all the students in the array

- int **getMinGPA**(Student students[], int numStudents) - calculates the minimumGPA from all the students in the array
- void **printStudentRecord**(Student student) - prints the record of the given student, including the student's ID, name, course codes, grades, and GPA
- void **printAllStudentRecords**(Student students[], int numStudents) - prints the records of all students in the array