**Unit:2**

Single File Programming Question

**Problem Statement**

Vijay wants to write a program for the multiplication of square matrices. Help him write the code.

**Note:** Square matrices mean the number of rows equals the number of columns in the given matrix.

Input format :

The first line of input consists of the matrix size M.

The next M lines of input consist of the M elements in each line separated by space representing the first matrix.

After being separated by a new line, the next M lines of input consist of the M elements in each line separated by space representing the first matrix.

Output format :

The output consists of the product of two matrices.

**Refer to the sample output for the formatting specifications.**

Code constraints :

0 < M <= 10

Sample test cases :

Input 1 :

3

2 3 2

3 2 3

3 3 3

4 5 6

2 3 1

1 2 3

Output 1 :

The product of the two matrices is:

16 23 21

19 27 29

21 30 30

Input 2 :

2

2 2

2 3

5 6

7 8

Output 2 :

The product of the two matrices is:

24 28

31 36

#include <iostream>

#include <vector>

int main() {

int M;

std::cin >> M;

// Initialize matrices

std::vector<std::vector<int>> matrix1(M, std::vector<int>(M));

std::vector<std::vector<int>> matrix2(M, std::vector<int>(M));

std::vector<std::vector<int>> result(M, std::vector<int>(M, 0));

// Input first matrix

for (int i = 0; i < M; i++) {

for (int j = 0; j < M; j++) {

std::cin >> matrix1[i][j];

}

}

// Input second matrix

for (int i = 0; i < M; i++) {

for (int j = 0; j < M; j++) {

std::cin >> matrix2[i][j];

}

}

// Matrix multiplication

for (int i = 0; i < M; i++) {

for (int j = 0; j < M; j++) {

for (int k = 0; k < M; k++) {

result[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

// Output the result

std::cout << "The product of the two matrices is:" << std::endl;

for (int i = 0; i < M; i++) {

for (int j = 0; j < M; j++) {

std::cout << result[i][j] << "\t";

}

std::cout << std::endl;

}

d

return 0;

}

Single File Programming Question

**Problem Statement**

Meena is a fitness enthusiast who wants to track her fitness statistics. She wants to create a program that allows her to input and update her fitness data. She needs to keep track of her name and weight. Meena prefers to use a class called **"FitnessData"** to store and manage her fitness information.

Write a program that utilizes the **"FitnessData"** class and allows Meena to input and update her fitness data. Use a pointer to a data member to access and modify the weight field.

**Note:** This is a sample question asked in a TCS interview.

Input format :

The first input consists of the name as a string.

The second input consists of the initial weight as a double.

The third input consists of the updated weight as a double.

Output format :

The output display the updated fitness information, including Meena's name and weight.

**Refer to the sample output for formatting specifications.**

Code constraints :

The name input will contain no spaces and be a single word.

1.0 <= weight <= 300.0

Sample test cases :

Input 1 :

Meena

60.5

65.2

Output 1 :

Updated fitness information:

Name: Meena

Weight: 65.2

#include <iostream>

#include <string>

class FitnessData {

public:

std::string name;

double weight;

FitnessData(const std::string& n, double w) : name(n), weight(w) {}

void updateWeight(double newWeight) {

weight = newWeight;

}

void displayFitnessInfo() {

std::cout << "Updated fitness information:" << std::endl;

std::cout << "Name: " << name << std::endl;

std::cout << "Weight: " << weight << std::endl;

}

};

int main() {

std::string name;

double initialWeight, updatedWeight;

std::cin >> name >> initialWeight >> updatedWeight;

FitnessData meenaFitness(name, initialWeight);

// Use a pointer to update the weight field

double\* weightPtr = &meenaFitness.weight;

\*weightPtr = updatedWeight;

meenaFitness.displayFitnessInfo();

return 0;

}

**Problem Statement**

Imagine you are developing a weather monitoring system for a meteorological research institute. As part of the system, you need to implement a feature that finds the highest recorded temperature for a specific region. Temperature data is stored in a 2D array, where each row represents a day and each column represents a different region.

The program should prompt the user to enter the temperature data for each region and day. Afterward, it should display the highest recorded temperature.

**Function Name:** int findMaxElement

**Note:** This is a sample question asked in an HCL interview.

Input format :

The first line of the input consists of the values of rows n and columns m, separated by a space.

The next n lines consist of m space-separated integers.

Output format :

The output prints the maximum element in the given 2D array.

Code constraints :

Define m and n = 10

1 <= m, n <= 10

-105 <= Elements of the 2D array <= 105

Sample test cases :

Input 1 :

3 4

34 37 -26 -63

78 95 -61 -44

16 61 45 -16

Output 1 :

95

#include <iostream>

#include <vector>

int findMaxElement(const std::vector<std::vector<int>>& data) {

int maxElement = data[0][0];

for (const auto& row : data) {

for (int element : row) {

if (element > maxElement) {

maxElement = element;

}

}

}

return maxElement;

}

int main() {

int n, m;

std::cin >> n >> m;

std::vector<std::vector<int>> temperatureData(n, std::vector<int>(m));

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

std::cin >> temperatureData[i][j];

}

}

int maxTemperature = findMaxElement(temperatureData);

std::cout << maxTemperature << std::endl;

return 0;

}

Raj wants to build a program that checks the strength of a given password. Implement a function **checkPasswordStrength** that takes a string representing a password as input and evaluates its strength based on criteria such as length, presence of uppercase letters, lowercase letters, numbers, and special characters. Use string objects to manipulate the password and perform the strength check.

* A password is classified as "Weak" if its length is less than 6 characters.
* A password is classified as "Medium" if its length is 6-25 characters, and it contains either an uppercase letter or a lowercase letter, along with at least one numeric digit.
* A password is classified as "Strong" if its length is 8-25 characters and it contains at least one uppercase letter, one lowercase letter, one numeric digit, and one special character.

**Note:** This kind of question will be helpful in clearing Capgemini tests.

Input format :

The input consists of a string representing the password that can be of any length and can contain any combination of characters.

Output format :

The program prints the strength of the password as "Weak," "Medium," or "Strong" based on the criteria mentioned in the question.

Code constraints :

The code assumes that the user will input a single line containing the password without any leading or trailing spaces.

1 <= Length of the password <= 25

Sample test cases :

Input 1 :

abcdef

Output 1 :

Weak

Input 2 :

Abcdef1

Output 2 :

Medium

Input 3 :

Abcdef1!

Output 3 :

Strong

Input 4 :

1123

Output 4 :

Weak

#include <iostream>

#include <string>

#include <cctype>

enum PasswordStrength { WEAK, MEDIUM, STRONG };

PasswordStrength checkPasswordStrength(const std::string& password) {

int length = password.length();

bool hasUpperCase = false;

bool hasLowerCase = false;

bool hasDigit = false;

bool hasSpecialChar = false;

for (char c : password) {

if (std::isupper(c)) {

hasUpperCase = true;

} else if (std::islower(c)) {

hasLowerCase = true;

} else if (std::isdigit(c)) {

hasDigit = true;

} else {

hasSpecialChar = true;

}

}

if (length < 6) {

return WEAK;

} else if (length >= 6 && length <= 25 && (hasUpperCase || hasLowerCase) && hasDigit) {

return MEDIUM;

} else if (length >= 8 && length <= 25 && hasUpperCase && hasLowerCase && hasDigit && hasSpecialChar) {

return STRONG;

} else {

return WEAK;

}

}

int main() {

std::string password;

std::cin >> password;

PasswordStrength strength = checkPasswordStrength(password);

switch (strength) {

case WEAK:

std::cout << "Weak" << std::endl;

break;

case MEDIUM:

std::cout << "Medium" << std::endl;

break;

case STRONG:

std::cout << "Strong" << std::endl;

break;

}

return 0;

}

Single File Programming Question

**Problem Statement**

Imagine you are a student attending a class on matrices, and your teacher has introduced a practical exercise to check whether two batches of matrices are identical or not. Your task is to create a program to solve this problem using an **array of objects** to represent matrices.

Input format :

The first line consists of two space-separated integers: m and n representing the number of rows and columns of the matrices A and B, respectively.

The next m lines each contain n space-separated integers representing the elements of matrix A.

The next m lines each contain n space-separated integers representing the elements of matrix B.

Output format :

The output displays either "Equal" or "Not Equal" based on the comparison of the two matrices A and B.

Code constraints :

MAX\_ROWS = 10

MAX\_COLS =10

MAX\_Matrices = 2

1 < m, n <= 10

-20 <= matrix elements <= 20

Sample test cases :

Input 1 :

3 3

4 9 -2

3 -5 7

-8 1 6

4 9 -2

3 -5 7

-8 1 6

Output 1 :

Equal

Input 2 :

2 3

1 2 3

4 5 6

1 2 3

4 5 6

Output 2 :

Equal

Input 3 :

3 3

4 9 2

3 5 7

8 1 6

4 9 2

3 5 7

5 2 2

Output 3 :

Not Equal

Dharun wants to build a program that generates an acronym for a given phrase. Implement a function that takes a string representing a phrase as input and returns its corresponding acronym. Use string objects to manipulate the input and generate the acronym.

**Function Name:** string generateAcronym

**Note:** This kind of question will be helpful in clearing Accenture tests.

Input format :

The input consists of a string representing the phrase that has to be abbreviated containing only upper-case and lower-case letters.

Output format :

The output prints the acronym of the given phrase in upper-case letters.

Code constraints :

The phrase contains only alphabets. No numbers or special characters should be used.

Length of the phrase < 1000

Sample test cases :

Input 1 :

If you know you know

Output 1 :

IYKYK

Input 2 :

Automated Teller Machine

Output 2 :

ATM

#include <iostream>

#include <string>

#include <sstream>

#include <cctype>

std::string generateAcronym(const std::string& phrase) {

std::string acronym = "";

std::istringstream iss(phrase);

std::string word;

while (iss >> word) {

acronym += std::toupper(word[0]);

}

return acronym;

}

int main() {

std::string phrase;

std::getline(std::cin, phrase);

std::string acronym = generateAcronym(phrase);

std::cout << acronym << std::endl;

return 0;

}

Guna wants to create a program that allows a teacher to input student information and generate their report cards. The program should take the number of students as input, followed by their names, roll numbers, and marks for five subjects. Finally, it should display the report cards for all the students.

He created a class called **Student** to store all the attributes and use an array of objects to solve the problem. Help him meet his expectations.

**Note:** This question is a sample question asked in an Accenture interview.

Input format :

The input begins with an integer, n, representing the number of students

For each student, the following inputs are provided and separated by a space:

-A string representing the student's name.

-An integer representing the student's roll number.

-Five integers representing the marks obtained by the student in each of the five subjects.

Output format :

The output displays the report cards for all the students.

Each report card should be displayed in the following format:

Name: [name]

Roll Number: [rollNumber]

Total Marks: [totalMarks]

where [name] represents the student's name, [rollNumber] represents the student's roll number, and [totalMarks] represents the total marks obtained by the student.

**Refer to the sample output for the exact format.**

Code constraints :

1 <= n <= 100

1 <= Length of the name <= 100

1 <= roll number <= 100

0 <= Each marks <= 100

Sample test cases :

Input 1 :

2

John 1 90 85 95 80 92

Emma 2 88 92 78 85 90

Output 1 :

Student Report Cards:

Name: John

Roll Number: 1

Total Marks: 442

Name: Emma

Roll Number: 2

Total Marks: 433

#include <iostream>

#include <string>

class Student {

public:

std::string name;

int rollNumber;

int marks[5];

int totalMarks;

Student() {

totalMarks = 0;

}

void calculateTotalMarks() {

for (int i = 0; i < 5; i++) {

totalMarks += marks[i];

}

}

void displayReportCard() {

std::cout << "Name: " << name << std::endl;

std::cout << "Roll Number: " << rollNumber << std::endl;

std::cout << "Total Marks: " << totalMarks << std::endl;

}

};

int main() {

int n;

std::cin >> n;

Student students[n];

for (int i = 0; i < n; i++) {

std::cin >> students[i].name;

std::cin >> students[i].rollNumber;

for (int j = 0; j < 5; j++) {

std::cin >> students[i].marks[j];

}

students[i].calculateTotalMarks();

}

std::cout << "Student Report Cards:" << std::endl;

for (int i = 0; i < n; i++) {

students[i].displayReportCard();

}

return 0;

}

2

ingle File Programming Question

**Problem Statement**

Meena wants to write a program that calculates the volume of a cube based on the user-provided values for its length, breadth, and height. Help Meena implement the program using a class named Cube with member functions like:

**void setLength(double l) -** Sets the length of the cube to the provided value l.

**void setBreadth (double b) -** Sets the breadth of the cube to the provided value b.

**void setHeight (double h) -** Sets the height of the cube to the provided value h.

**double getVolume() -** Calculates and returns the volume of the cube based on the set length, breadth, and height values.

**Note:** This is a sample question asked in an IBM interview.

Input format :

The input consists of the three double-point numbers representing the length (l), breadth (b), and height (h) of the cube, separated by a space.

Output format :

The output displays the volume of the cube calculated using the input values.

Code constraints :

1 <= l, b, h <= 99

Sample test cases :

Input 1 :

10.0 10.0 10.0

Output 1 :

1000

Input 2 :

12.3 27.2 32.4

Output 2 :

10839.7

#include <iostream>

class Cube {

private:

double length;

double breadth;

double height;

public:

void setLength(double l) {

length = l;

}

void setBreadth(double b) {

breadth = b;

}

void setHeight(double h) {

height = h;

}

double getVolume() {

return length \* breadth \* height;

}

};

int main() {

Cube;

double l, b, h;

std::cin >> l >> b >> h;

cube.setLength(l);

cube.setBreadth(b);

cube.setHeight(h);

double volume = cube.getVolume();

std::cout << volume << std::endl;

return 0;

}

Harry wants to write a program that takes an integer input from the user and determines the number of digits in the input number. Help Harry implement the program using a class named **Input** using a member function called **count**. Beginning zeros will be ignored while counting the number of digits.

**Note:** This question is a sample question asked in a Wipro interview.

Input format :

The input consists of an integer n.

Output format :

The output displays the count of the number of digits in the entered number.

**Note:** Beginning zeros will be ignored while counting the number of digits.

Code constraints :

0 < n <= 109

Sample test cases :

Input 1 :

12345

Output 1 :

5

Input 2 :

4321

Output 2 :

4

#include <iostream>

class Cube {

private:

double length;

double breadth;

double height;

public:

void setLength(double l) {

length = l;

}

void setBreadth(double b) {

breadth = b;

}

void setHeight(double h) {

height = h;

}

double getVolume() {

return length \* breadth \* height;

}

};

int main() {

Cube;

double l, b, h;

std::cin >> l >> b >> h;

cube.setLength(l);

cube.setBreadth(b);

cube.setHeight(h);

double volume = cube.getVolume();

std::cout << volume << std::endl;

return 0;

}

Single File Programming Question

**Problem Statement**

Akil wants to perform various operations on a string. He wants to append additional characters to the end of the string, insert characters into the string at a specified position, and erase characters from the string at a specified position or within a specified range. Akil needs your help to write a program that performs these operations on a given string.

Write a program that takes a string as input and performs the following operations:

1. **Append -** Append additional characters to the end of the string.
2. **Insert -** Insert additional characters into the string at a specified position.
3. **Erase -** Remove characters from the string at a specified position or within a specified range.

**Note:** This question is a sample question asked in a Mettl interview.

Input format :

The first line of input consists of the initial string.

The second line of input consists of the additional characters to append.

The third line of input consists of the characters to insert.

The fourth line of input consists of the position to insert.

The fifth line of input consists of the starting position to erase.

The sixth line of input consists of the number of characters to erase.

Output format :

The output displays the modified string after each operation (append, insert, and erase).

**Refer to the sample output for the formatting specifications.**

Code constraints :

1 <= Length of the strings < 50

Sample test cases :

Input 1 :

Hello

World

in

4

4

2

Output 1 :

Modified string after append: Hello World

Modified string after insert: Hellino World

Modified string after erase: Hello World

#include <iostream>

class Cube {

private:

double length;

double breadth;

double height;

public:

void setLength(double l) {

length = l;

}

void setBreadth(double b) {

breadth = b;

}

void setHeight(double h) {

height = h;

}

double getVolume() {

return length \* breadth \* height;

}

};

int main() {

Cube;

double l, b, h;

std::cin >> l >> b >> h;

cube.setLength(l);

cube.setBreadth(b);

cube.setHeight(h);

double volume = cube.getVolume();

std::cout << volume << std::endl;

return 0;

}

Charu wants to create a class called **RestaurantMenu** to represent a restaurant menu. The class should have attributes: menu name (string), menu items (array of strings), and prices (array of floats). Help her implement member functions to add menu items and display the menu with prices for the specified item.

**Note:** This question will be helpful in clearing AMCAT tests.

Input format :

The first line of input consists of the menu name.

The second line of input consists of the number of menu items, N.

For each menu item, the following lines of input consist of the name of the item and its price in separate lines.

The last line of input consists of the item name to be searched.

Output format :

If the item is present in the menu, the output prints its respective price rounded off to two decimal places.

If the item is not present in the menu, the output prints "Item not found.".

Code constraints :

1 <= N <= 20

Sample test cases :

Input 1 :

Dinner Menu

4

Grilled Salmon

150.99

Spaghetti Bolognese

120.99

Margherita Pizza

900.99

Caesar Salad

800.99

Caesar Salad

Output 1 :

800.99

Input 2 :

Dinner Menu

4

Grilled Salmon

155.99

Spaghetti Bolognese

120.99

Margherita Pizza

900.99

Caesar Salad

558.99

Salad

Output 2 :

Item not found.

#include <iostream>

class Cube {

private:

double length;

double breadth;

double height;

public:

void setLength(double l) {

length = l;

}

void setBreadth(double b) {

breadth = b;

}

void setHeight(double h) {

height = h;

}

double getVolume() {

return length \* breadth \* height;

}

};

int main() {

Cube;

double l, b, h;

std::cin >> l >> b >> h;

cube.setLength(l);

cube.setBreadth(b);

cube.setHeight(h);

double volume = cube.getVolume();

std::cout << volume << std::endl;

return 0;

}

Single File Programming Question

**Problem Statement**

Sam wants to create a class called **WeatherForecast** that contains weather information. The class should have attributes: location (string), and its temperature (float). He wants to implement member functions to add the weather information and display the location having the lowest temperature. Help him write the code. Assume that the temperature is measured in Celcius.

**Note:** This kind of question helps in clearing Accenture tests.

Input format :

The first line consists of an integer N representing the number of locations.

The following lines consist of the location name and temperature separated by a new line.

Output format :

The output prints the coldest location with the lowest temperature.

Code constraints :

1 <= N <= 20

Sample test cases :

Input 1 :

4

New York

25.5

Los Angeles

20.8

Chicago

15.2

Boston

18.9

Output 1 :

Chicago

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class WeatherForecast {

private:

vector<string> locations;

vector<float> temperatures;

public:

// Function to add weather information for a location

void addLocationWeather(string location, float temperature) {

locations.push\_back(location);

temperatures.push\_back(temperature);

}

// Function to find and display the coldest location

void displayColdestLocation() {

if (locations.empty()) {

cout << "No locations provided." << endl;

return;

}

float lowestTemperature = temperatures[0];

string coldestLocation = locations[0];

for (size\_t i = 1; i < temperatures.size(); ++i) {

if (temperatures[i] < lowestTemperature) {

lowestTemperature = temperatures[i];

coldestLocation = locations[i];

}

}

cout << coldestLocation << endl;

}

};

int main() {

int N;

cin >> N;

cin.ignore(); // Consume the newline character

WeatherForecast weather;

for (int i = 0; i < N; ++i) {

string location;

float temperature;

getline(cin, location);

cin >> temperature;

cin.ignore(); // Consume the newline character

weather.addLocationWeather(location, temperature);

}

weather.displayColdestLocation();

return 0;

}

You have been assigned the task of designing a program that checks whether a given string is a valid phone number or not.

Write a function **isValidPhoneNumber** that takes a phoneNumber (string) as input and determines whether it is a valid phone number. The phone number format should be XXX-XXX-XXXX, where X represents a digit. The number is considered valid if the hyphens are in exact same positions and the length of the string is 12.

Implement the function isValidPhoneNumber and write a main program to test it. The main program should prompt the user to enter a phone number, and then display whether the entered string is a valid phone number or not. If the number is valid, print the numbers without using hyphens.

**Note:** This kind of question helps in clearing Capgemini tests.

Input format :

The input consists of a phone number in the format XXX-XXX-XXXX, where X represents a digit.

Output format :

The output displays whether the entered string is a valid phone number or not as shown in the sample output.

If the number is valid, the output displays the number without hyphens.

Code constraints :

The phone number should be in the format XXX-XXX-XXXX, with a total length of 12 characters.

Each X in the phone number should be a digit (0-9).

No additional characters or spaces should be present in the input.

Sample test cases :

Input 1 :

123-456-7890

Output 1 :

Yes, the phone number is valid.

1234567890

Input 2 :

123-456-789

Output 2 :

No, the phone number is not

#include <iostream>

#include <string>

using namespace std;

// Function to check if a string is a valid phone number

bool isValidPhoneNumber(const string& phoneNumber) {

// Check if the length of the string is exactly 12 characters

if (phoneNumber.length() != 12) {

return false;

}

// Check if hyphens are in the correct positions

if (phoneNumber[3] != '-' || phoneNumber[7] != '-') {

return false;

}

// Check if all characters in positions 0-2 and 4-6 are digits

for (int i = 0; i < 12; ++i) {

if (i == 3 || i == 7) {

continue; // Skip hyphens

}

if (!isdigit(phoneNumber[i])) {

return false;

}

}

return true;

}

int main() {

string phoneNumber;

// Input phone number

cin >> phoneNumber;

if (isValidPhoneNumber(phoneNumber)) {

cout << "Yes, the phone number is valid." << endl;

// Remove hyphens and display the number without hyphens

string numberWithoutHyphens = phoneNumber.substr(0, 3) + phoneNumber.substr(4, 3) + phoneNumber.substr(8);

cout << numberWithoutHyphens << endl;

} else {

cout << "No, the phone number is not valid." << endl;

}

return 0;

}

Ragu wants to perform various operations on a given string. He wants to replace a specific substring with another string, extract a substring from a specified position and length, and resize the string to a new size. Ragu needs your help to write a program that performs these operations on the given string.

Write a program that takes a string as input and performs the following operations:

1. **Replace -** Replace a specific substring in the string with another string.
2. **Substring -** Extract a substring from the original string based on a given starting position and length.
3. **Resize -** Change the length of the string to a new size, either by truncating or padding it with additional characters.

**Note:** This is a sample question asked in a Wipro interview.

Input format :

The first line of input is the initial string.

The second line of input is the substring to be replaced.

The third line of input is the replacement string.

The fourth line of input is the starting position for the substring extraction (a).

The fifth line of input is the length of the substring to be extracted.

The sixth line of input is the new size for resizing the string (b).

Output format :

The output displays the modified string after the replace operation, the extracted substring, and the resized string in separate lines.

**Refer to the sample output for the formatting specifications.**

Code constraints :

1 < a, b < Length of the strings <= 80

Sample test cases :

Input 1 :

Hello, World!

World

Universe

7

5

10

Output 1 :

Modified string after replace: Hello, Universe!

Substring: Unive

Resized string: Hello, Uni

#include <iostream>

#include <string>

using namespace std;

// Function to replace a specific substring in the string

void replaceSubstring(string& str, const string& substringToReplace, const string& replacement) {

size\_t pos = str.find(substringToReplace);

if (pos != string::npos) {

str.replace(pos, substringToReplace.length(), replacement);

}

}

// Function to extract a substring from the string based on starting position and length

string extractSubstring(const string& str, int start, int length) {

if (start >= 0 && start < str.length() && length > 0) {

return str.substr(start, length);

} else {

return "";

}

}

// Function to resize the string to a new size (truncating or padding)

string resizeString(const string& str, int newSize) {

if (newSize > 0) {

if (newSize <= str.length()) {

return str.substr(0, newSize);

} else {

return str + string(newSize - str.length(), ' ');

}

} else {

return "";

}

}

int main() {

string initialString, substringToReplace, replacement;

int start, length, newSize;

// Input initial string

getline(cin, initialString);

// Input substring to be replaced

getline(cin, substringToReplace);

// Input replacement string

getline(cin, replacement);

// Input starting position for substring extraction

cin >> start;

// Input length of the substring to be extracted

cin >> length;

// Input new size for resizing the string

cin >> newSize;

// Perform replace operation

replaceSubstring(initialString, substringToReplace, replacement);

cout << "Modified string after replace: " << initialString << endl;

// Perform substring operation

string extractedSubstring = extractSubstring(initialString, start, length);

cout << "Substring: " << extractedSubstring << endl;

// Perform resize operation

string resizedString = resizeString(initialString, newSize);

cout << "Resized string: " << resizedString << endl;

return 0;

}

**Unit:3**

you are required to create a class called Euclids which compute the greatest common divisor using a scheduled algorithm for to positive integer the easily class take to integer parameter in a set data method and has three method set data computegcd and display

the Eculid algorithm work is followed

. in both the argumnent of positive Eculid algorithm is employed

if any of the argument is negative the algorithm return 0

A first integer is divided by the second and the remainder is stored

then the second is divided by the remainder and the first remainder is divided by the second and so on until a remainder of 0 is obtained

the gcd will be divided that produce a remainder of 0

if either of the input integer is less than or equal to 0 then completeGCD method should return 0 the gcd

input

the input consists of two integer values separated by space

output

the first line of output print the integer separated by space the second line of the output frequency of the integer write a code in c++ and take input from user

Top of Form

#include <iostream>

class Euclids {

private:

int num1;

int num2;

public:

// Method to set data for the two integers

void setData(int a, int b) {

num1 = a;

num2 = b;

}

// Method to compute the GCD using the Euclidean algorithm

int computeGCD() {

if (num1 <= 0 || num2 <= 0) {

return 0; // Return 0 if either input is non-positive

}

int temp;

while (num2 != 0) {

temp = num2;

num2 = num1 % num2;

num1 = temp;

}

return num1; // The GCD is stored in num1

}

// Method to display the GCD

void display() {

std::cout << num1 << " " << num2 << std::endl;

int gcd = computeGCD();

std::cout << "GCD: " << gcd << std::endl;

}

};

int main() {

Euclids euclid;

int a, b;

std::cout << "Enter two positive integers separated by space: ";

std::cin >> a >> b;

euclid.setData(a, b);

euclid.display();

return 0;

}

you are creating a program to manage physical registration information is vehicle is initially identified by its registration number and you need to store detail lies make model and year a manufacturer for each vehicle

implement a class called "Vehicle" with the following figure

a parameterized constructor they take for argument. Registration number (character array), make of the vehicle (character array), model of vehicle(character array), and Year of manufacture(integer).

a Member function name with "displayDetail" display the Registration number, Make ,Model and Year of manufacture

your task is to create the Vehicle class is described and use it to display the registration details program help manage page ka registration efficiency

the input consists of four line is containing

your task is to create the physical class is described and use it to display the registration details program help manage page ka registration efficiency

the input consists of four line is containing

a string representing the registration number

a string representing the make of the vehicle

a string representing the model of the vehicle

a string representing the year of manufacture

output print the following is containing the details in the format

Registration Number :[registration number]

Make:[ make]

Model: [model]

Year of Manufacture:[year of manufacture]

#include <iostream>

#include <cstring>

class Vehicle {

private:

char registrationNumber[20];

char make[50];

char model[50];

int yearOfManufacture;

public:

// Parameterized constructor to initialize vehicle details

Vehicle(const char\* regNum, const char\* vehicleMake, const char\* vehicleModel, int year) {

std::strcpy(registrationNumber, regNum);

std::strcpy(make, vehicleMake);

std::strcpy(model, vehicleModel);

yearOfManufacture = year;

}

// Member function to display vehicle details

void displayDetails() {

std::cout << "Registration Number: " << registrationNumber << std::endl;

std::cout << "Make: " << make << std::endl;

std::cout << "Model: " << model << std::endl;

std::cout << "Year of Manufacture: " << yearOfManufacture << std::endl;

}

};

int main() {

char regNum[20];

char vehicleMake[50];

char vehicleModel[50];

int year;

// Input vehicle details

std::cin.getline(regNum, 20);

std::cin.getline(vehicleMake, 50);

std::cin.getline(vehicleModel, 50);

std::cin >> year;

// Create a Vehicle object with the input details

Vehicle vehicle(regNum, vehicleMake, vehicleModel, year);

// Display vehicle details

vehicle.displayDetails();

return 0;

}

**end**

Hema wants to manage a list of professors at her university. She needs a program that takes input for the number of professors (N) and their details, including their ID and salary. Hema wants to display the details of professors whose salaries are greater than or equal to 20,000.

Write a program to implement this functionality using a class named "Professor" with appropriate member variables and **display()** member functions, and add a constructor with default arguments to the "Professor" class.

**Note:** This is a sample question asked in a Deloitte interview.

**Input format :**

The first line consists of the total number of professors (N).

The next N lines consist of Employee ID (integer) and salary (integer), separated by a space in each line.

**Output format :**

The output prints the details of the professors whose salaries are greater than or equal to 20000 on separate lines.

**Code constraints :**

1 <= N <= 50

0 <= ID <= 105

0 <= salary <= 106

**Sample test cases :**

**Input 1 :**

3

1001 25000

1002 30000

1003 12000

**Output 1 :**

1001 25000

1002 30000

**Input 2 :**

2

100123 100000

100124 200000

**Output 2 :**

100123 100000

100124 200000

#include <iostream>

#include <vector>

class Professor {

public:

Professor(int id = 0, int salary = 0) : id\_(id), salary\_(salary) {}

void display() const {

cout << id\_ << " " << salary\_ << endl;

}

bool isSalaryGreaterThanEqual(int limit) const {

return salary\_ >= limit;

}

private:

int id\_;

int salary\_;

};

int main() {

int N;

cin >> N;

vector<Professor> professors;

for (int i = 0; i < N; ++i) {

int id, salary;

std::cin >> id >> salary;

professors.push\_back(Professor(id, salary));

}

int salaryLimit = 20000;

for (int i = 0; i < N; ++i) {

if (professors[i].isSalaryGreaterThanEqual(salaryLimit)) {

professors[i].display();

}

}

return 0;

}

//output

//2

//100123 100000

//100124 200000

//100123 100000

//100124 200000

Create a class named "Money" with the attributes "rupee" and "paise". Implement a constructor that initializes rupees and paise with default values of 0.

Create member functions setRupee() and setPaise() to set the values, and getRupee() and getPaise() to retrieve them. If paise exceed 99, increment rupees by 1 and adjust paise.

**Note:** This is a sample question asked in an Infosys interview.

**Input format :**

The input consists of two amounts in each line, rupee (integer) and paise (integer), separated by a space in each line.

**Output format :**

The output prints two integers separated by a period (.), representing the sum of rupees and paise.

**Code constraints :**

0 <= rupee <= 105

0 <= paise <= 99

**Sample test cases :**

**Input 1 :**

50 85

42 65

**Output 1 :**

93.50

**Input 2 :**

254 45

845 20

**Output 2 :**

1099.65

**Input 3 :**

50 99

50 9

**Output 3 :**

101.8

**Input 4 :**

10 90

42 50

**Output 4 :**

53.40

#include <iostream>

class Money {

public:

Money() : rupee(0), paise(0) {}

void setRupee(int r) {

rupee = r;

}

void setPaise(int p) {

paise = p;

// Adjust rupees if paise exceeds 99

if (paise > 99) {

int extraRupees = paise / 100;

rupee += extraRupees;

paise %= 100;

}

}

int getRupee() const {

return rupee;

}

int getPaise() const {

return paise;

}

// Function to display the money in the required format

void display() const {

std::cout << rupee << "." << (paise < 10 ? "0" : "") << paise << std::endl;

}

private:

int rupee;

int paise;

};

int main() {

int rupee1, paise1, rupee2, paise2;

while (std::cin >> rupee1 >> paise1) {

Money money1, money2;

money1.setRupee(rupee1);

money1.setPaise(paise1);

std::cin >> rupee2 >> paise2;

money2.setRupee(rupee2);

money2.setPaise(paise2);

Money totalMoney;

totalMoney.setRupee(money1.getRupee() + money2.getRupee());

totalMoney.setPaise(money1.getPaise() + money2.getPaise());

totalMoney.display();

}

return 0;

}

//output

//50 85

//42 65

//93.50

You're creating a program to manage vehicle registration information. Each vehicle is uniquely identified by its registration number, and you need to store details like the make, model, and year of manufacture for each vehicle.

Implement a class called "Vehicle" with the following features:

A **parameterized constructor** that takes four arguments: Registration number (Character array), Make of the vehicle (Character array), Model of the vehicle (Character array), and Year of manufacture (integer).

A **member function** named "displayDetails" displays the Registration number, Make, Model, and Year of manufacture.

Your task is to create the "Vehicle" class as described and use it to display the registration details of a vehicle. This program helps manage vehicle registration efficiently.

**Input format :**

The input consists of four lines, each containing:

* A string representing the registration number.
* A string representing the make of the vehicle.
* A string representing the model of the vehicle.
* An integer representing the year of manufacture.

**Output format :**

The output prints the following, each containing the details of a vehicle in the following format:

Registration Number: [registration number]

Make: [make]

Model: [model]

Year of Manufacture: [year of manufacture]

**Refer to the sample output for the exact format.**

**Code constraints :**

2001 <= year <= 3000

Length of character array <=20

**Sample test cases :**

**Input 1 :**

KA-01-HH-1234

Toyota

Corolla

2022

**Output 1 :**

Registration Number: KA-01-HH-1234

Make: Toyota

Model: Corolla

Year of Manufacture: 2022

#include <iostream>

class Distance {

public:

Distance() : feet\_(0), inches\_(0) {}

Distance(double feet, double inches) : feet\_(feet), inches\_(inches) {}

// Copy constructor

Distance(const Distance& other) : feet\_(other.feet\_), inches\_(other.inches\_) {}

void setDistance(double feet, double inches) {

feet\_ = feet;

inches\_ = inches;

}

void addDistance(const Distance& other) {

feet\_ += other.feet\_;

inches\_ += other.inches\_;

if (inches\_ >= 12.0) {

feet\_ += static\_cast<int>(inches\_ / 12);

inches\_ = inches\_ - 12.0 \* static\_cast<int>(inches\_ / 12);

}

}

void displayDistance() const {

std::cout << "distance = " << feet\_ << "'" << inches\_ << "\"" << std::endl;

}

private:

double feet\_;

double inches\_;

};

int main() {

double feet1, inches1, feet2, inches2;

std::cin >> feet1 >> inches1;

//std::cin >> feet2 >> inches2;

Distance d1(feet1, inches1);

Distance d2 = d1; // Create d2 as a copy of d1 using the copy constructor

d1.addDistance(d2); // Calculate the sum of d1 and d2

d1.displayDistance(); // Display the result

return 0;

}

//output

//10.0 1.0

//distance = 20'2"

**File handling**

**Problem Statement:**

Banu wants to write a program that allows her to enter text and a count value. The program will store the given text multiple times in a file named "**sample.txt**". Later, Banu wants to read the contents of the file and display them on the console.

You should write the main() method where you call the setup code, and if it returns 0, then proceed to do the following:

a) check if the file is open

b) read through the contents of the file

c) check if the eof is reached and if yes, then print "End of file reached"

d) close the file.

If the setup code fails, then return non-zero, upon receiving which the main method will print "Unable to setup data." If you cannot open the file for either reading or writing, then you have to print "Unable to open file."

**Note:** This is a sample question asked in an Accenture interview.

**Input format :**

The first line of input consists of text (a single line).

The second line input consists of a count value, indicating the number of times the text should be repeated in the file.

**Output format :**

The output displays, Create or overwrite the file "sample.txt" and store the given text multiple times, according to the count value.

The output displays the contents of the file and displays each line on the console.

If the end of the file is reached, the output display the message "End of file reached".

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

Length of the text <= 10

**Sample test cases :**

**Input 1 :**

HelloWorld

10

**Output 1 :**

HelloWorld

HelloWorld

HelloWorld

HelloWorld

HelloWorld

HelloWorld

HelloWorld

HelloWorld

HelloWorld

HelloWorld

End of file reached

**Input 2 :**

Universal

4

**Output 2 :**

Universal

Universal

Universal

Universal

End of file reached

#include <iostream>

#include <fstream>

int setup() {

// Open the file for writing (create or overwrite)

std::ofstream outFile("sample.txt");

if (!outFile.is\_open()) {

std::cerr << "Unable to open file." << std::endl;

return 1; // Return non-zero to indicate failure

}

std::string text;

int count;

std::getline(std::cin, text); // Read the text

std::cin >> count; // Read the count

for (int i = 0; i < count; ++i) {

outFile << text << std::endl; // Write text to the file

}

outFile.close(); // Close the file

return 0; // Return zero to indicate success

}

int main() {

if (setup() == 0) {

// File setup was successful, proceed to read the file

std::ifstream inFile("sample.txt");

if (!inFile.is\_open()) {

std::cerr << "Unable to open file." << std::endl;

return 1; // Return non-zero to indicate failure

}

std::string line;

while (std::getline(inFile, line)) {

std::cout << line << std::endl;

}

if (inFile.eof()) {

std::cout << "End of file reached" << std::endl;

} else {

std::cerr << "Error reading file." << std::endl;

}

inFile.close(); // Close the file

} else {

std::cerr << "Unable to setup data." << std::endl;

}

return 0;

}

//output

//HelloWorld

//10

//HelloWorld

//HelloWorld

//HelloWorld

//HelloWorld

//HelloWorld

//HelloWorld

//HelloWorld

//HelloWorld

//HelloWorld

//HelloWorld

//End of file reached

**Problem Statement:**

Praveen has been assigned the task of developing a program that manages a file containing numbers. The program needs to perform the following operations:

**Setup the initial file:** The program should generate a file named "**numbers.txt**" and populate it with a given count of numbers starting from 1. Each number should be written on a separate line.

**Check if the file is empty:** After setting up the initial file, the program needs to check if the file is empty or contains any numbers.

**File manipulation based on emptiness:** Depending on whether the file is empty or not, the program needs to perform the following actions:

* If the file is empty, add a specified number of additional numbers starting from 100 to the file.
* If the file is not empty, consider the operation a failure.

Write a program that takes an integer as input, representing the count of initial numbers to set up the file (**numberCount**). The program should perform the necessary file operations according to the problem statement and output the appropriate success or failure message.

**Note:** This is a sample question asked in an Amcat interview.

**Input format :**

The input contains an integer representing the count of initial numbers to set up the file (**numberCount**).

**Output format :**

If the file is empty the first time, print "File is empty".

Before adding the count of numbers starting from 100, print the message according to the output.

If the file is not empty after completing the writing, then print "Success" otherwise print "Failed"

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

The input value for numberCount should be within the range of integer values.

The program assumes that the file "numbers.txt" is writable and readable in the current directory.

**Sample test cases :**

**Input 1 :**

10

**Output 1 :**

File is empty

Adding 10 numbers starting from 100

Success

#include <iostream>

#include <fstream>

int main() {

int numberCount;

std::cin >> numberCount;

std::fstream file("numbers.txt", std::ios::in | std::ios::out | std::ios::app);

if (!file.is\_open()) {

std::cerr << "Failed to open the file." << std::endl;

return 1; // Return non-zero to indicate failure

}

// Check if the file is empty

file.seekg(0, std::ios::end);

if (file.tellg() == 0) {

std::cout << "File is empty" << std::endl;

// Add numbers starting from 100

for (int i = 0; i < numberCount; ++i) {

file << i + 100 << std::endl;

}

std::cout << "Adding " << numberCount << " numbers starting from 100" << std::endl;

std::cout << "Success" << std::endl;

} else {

std::cerr << "Failed" << std::endl;

}

file.close();

return 0;

}

//output

//10

//File is empty

//Adding 10 numbers starting from 100

//Success

Mohana wants to store an integer value in a binary file and retrieve it later. She wants to write a program that allows her to enter an integer value, stores it in a file named **"student.dat**", and then reads the value from the file.

The methods required are:

a) **getFileHandleForWriting():** This should open the file for writing with appropriate permissions and modes in order to write the instances of the struct student.

b) **getFileHandlerForReading():** This should open the file for reading with appropriate permissions and modes in order to read the instance of the struct student that has been stored.

c) **closeFile()** which can take both the input stream and the output stream and close the file.

If you cannot open the file, print "Cannot open file!".

If you cannot close the file, print "Error occurred at reading/writing time!".

The main method prints "Success" if all is well.

**Note**: This is a sample question asked in a Wipro interview.

**Input format :**

The input consists of an integer value.

**Output format :**

The output displays the integer value in the file "student.dat".

The output displays read the stored value from the file and display it on the console.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

The entered integer value will fit within the range of an integer data type.

The file "student.dat" must be created and accessible for read and write operations.

The program will handle any potential errors during file handling, such as failure to open or close the file.

**Sample test cases :**

**Input 1 :**

10

**Output 1 :**

Stored Value: 10

Success

#include <iostream>

#include <fstream>

// Define a struct to store the integer value

struct Student {

int value;

};

// Function to open the file for writing

std::ofstream getFileHandleForWriting() {

std::ofstream file("student.dat", std::ios::out | std::ios::binary);

return file;

}

// Function to open the file for reading

std::ifstream getFileHandleForReading() {

std::ifstream file("student.dat", std::ios::in | std::ios::binary);

return file;

}

// Function to close the file

void closeFile(std::ifstream& input) {

input.close();

}

// Function to close the file

void closeFile(std::ofstream& output) {

output.close();

}

int main() {

int value;

std::cin >> value;

// Create a struct instance with the entered value

Student;

student.value = value;

// Open the file for writing

std::ofstream outputFile = getFileHandleForWriting();

// Check if the file is open

if (!outputFile.is\_open()) {

std::cerr << "Cannot open file!" << std::endl;

return 1; // Return non-zero to indicate failure

}

// Write the student struct to the file

outputFile.write(reinterpret\_cast<char\*>(&student), sizeof(student));

// Close the file

closeFile(outputFile);

// Open the file for reading

std::ifstream inputFile = getFileHandleForReading();

// Check if the file is open

if (!inputFile.is\_open()) {

std::cerr << "Cannot open file!" << std::endl;

return 1; // Return non-zero to indicate failure

}

// Read the stored value from the file

Student readStudent;

inputFile.read(reinterpret\_cast<char\*>(&readStudent), sizeof(readStudent));

// Close the file

closeFile(inputFile);

// Print the stored value

std::cout << "Stored Value: " << readStudent.value << std::endl;

std::cout << "Success" << std::endl;

return 0;

}

**Single File Programming Question**

**Problem Statement:**

Ishu wants to set up a program that creates a file named "**numbers.txt"** and fills it with numbers from 1 to N, where N is a positive integer provided as input. After setting up the file, Ishu wants to read the numbers from the file and display them on the console. Help ishu to write the program.

**Note:** This is a sample question asked in an Infosys interview.

**Input format :**

The input consists of a single integer, N, representing the maximum number to be written to the file.

**Output format :**

The program reads the numbers from the file "numbers.txt" and outputs each number on a separate space.

**Code constraints :**

1 <= N <= 2000

The input integer N represents the number of elements in the sequence.

The file manipulation operations are performed using the file named "numbers.txt".

The file "numbers.txt" is truncated (emptied) after the setup process.

The program will handle file errors such as failure to open or close the file.

The output format should exactly match the provided samples.

**Sample test cases :**

**Input 1 :**

10

**Output 1 :**

1 2 3 4 5 6 7 8 9 10

#include <iostream>

#include <fstream>

int main() {

int N;

std::cin >> N;

// Open the file for writing

std::ofstream outputFile("numbers.txt");

// Check if the file is open

if (!outputFile.is\_open()) {

std::cerr << "Unable to open file!" << std::endl;

return 1; // Return non-zero to indicate failure

}

// Write numbers from 1 to N to the file

for (int i = 1; i <= N; ++i) {

outputFile << i << " ";

}

// Close the file

outputFile.close();

// Open the file for reading

std::ifstream inputFile("numbers.txt");

// Check if the file is open

if (!inputFile.is\_open()) {

std::cerr << "Unable to open file!" << std::endl;

return 1; // Return non-zero to indicate failure

}

// Read and display numbers from the file

int number;

while (inputFile >> number) {

std::cout << number << " ";

}

// Close the file

inputFile.close();

return 0;

}

//output

//10

//1 2 3 4 5 6 7 8 9 10

You are given a program that saves an employee object to a binary file and calculates the size of the file. The employee object contains an ID. You need to modify the program to correctly calculate the size of the file.

The input shall be an integer. The output should be given as specified in the sample output, where the size might vary based on the input data. If the setup code fails, then print "Unable to setup."

**Input format :**

The input consists of an integer eid representing the employee ID.

**Output format :**

The output displays the size of the file in bytes.

**Code constraints :**

The employee ID (eid) is a non-negative integer.

The file size should be reported in bytes.

**Sample test cases :**

**Input 1 :**

10

**Output 1 :**

size is: 4 bytes.

#include <iostream>

#include <fstream>

class Employee {

public:

int eid;

Employee(int id) : eid(id) {}

};

int main() {

int eid;

std::cin >> eid;

// Create an Employee object

Employee employee(eid);

// Open a binary file for writing

std::ofstream outputFile("employee.dat", std::ios::binary);

// Check if the file is open

if (!outputFile.is\_open()) {

std::cerr << "Unable to setup." << std::endl;

return 1; // Return non-zero to indicate failure

}

// Write the Employee object to the file

outputFile.write(reinterpret\_cast<char\*>(&employee), sizeof(Employee));

// Close the file

outputFile.close();

// Calculate the size of the file

std::ifstream inputFile("employee.dat", std::ios::binary | std::ios::ait);

if (!inputFile.is\_open()) {

std::cerr << "Unable to setup." << std::endl;

return 1; // Return non-zero to indicate failure

}

std::streams file Size = inputFile.tellg();

inputFile.close();

// Print the size of the file in bytes

std::cout << "size is: " << fileSize << " bytes." << std::endl;

return 0;

}

//output

//10

//size is: 4 bytes.

Misha needs a program to open and close files based on user input. The program should ask for the file name, and display messages for successful file opening and closing.

**Input format :**

The input consists of the name of the file they want to open and close and the filename as a single line of text.

**Output format :**

If the file is successfully opened, it displays "File 'filename' opened successfully.".

Regardless of whether the file was opened successfully or not, it displays "File 'filename' closed successfully." to indicate that the file is closed at the end of the program.

**Sample test cases :**

**Input 1 :**

sample.txt

**Output 1 :**

File 'sample.txt' opened successfully.

File 'sample.txt' closed successfully.

**Input 2 :**

input.txt

**Output 2 :**

File 'input.txt' opened successfully.

File 'input.txt' closed succe

#include <iostream>

#include <string>

int main() {

std::string filename;

// Input: Ask the user for the filename

std::cout << "Enter the filename: ";

std::cin >> filename;

// Display a message indicating the file is closed

std::cout << "File '" << filename << "' opened successfully." << std::endl;

std::cout << "File '" << filename << "' closed successfully." << std::endl;

return 0;

}

//output

//Enter the filename: sample.txt

//File 'sample.txt' opened successfully.

//File 'sample.txt' closed successfully.

**Binary search operation**

Imagine you're building a C++ program to assist users in recording and retrieving personal data. Prompt the user to provide an integer (age), a floating-point number (height in meters), and a string (name). Your program should save these details to a file named "**data.txt**". Subsequently, it should read and display the stored information from the file.

**Note:** Write the height value rounded off to one decimal place inside the file and also read the same with one decimal place.

**Input format :**

The first consists of an integer: age.

The second line consists of a floating-point number: height in cm.

The third line consists of a string: name.

**Output format :**

The program displays the age, height, and name of the user in each line.

**Note:**Round off the height value to one decimal place while writing as well as reading from the file.

**Code constraints :**

1 <= age <= 80

45 <= height <= 180

The program assumes that all inputs are valid.

**Sample test cases :**

**Input 1 :**

25

160.55

Maria L

**Output 1 :**

25

160.6

Maria L

**Input 2 :**

3

60.22

Emma G Cooper

**Output 2 :**

3

60.2

Emma G Cooper

#include <iostream>

#include <fstream>

#include <string>

#include <iomanip>

int main() {

int age;

double height;

std::string name;

// Input: Ask the user for age, height, and name

//std::cout << "Enter age: ";

std::cin >> age;

// std::cout << "Enter height (in cm): ";

std::cin >> height;

std::cin.ignore(); // Ignore the newline character

//std::cout << "Enter name: ";

std::getline(std::cin, name);

// Write data to "data.txt"

std::ofstream outFile("data.txt");

if (!outFile.is\_open()) {

std::cerr << "Error: Unable to open data.txt for writing." << std::endl;

return 1;

}

outFile << age << std::endl;

outFile << std::fixed << std::setprecision(1) << height / 1.0 << std::endl; // Convert height to meters and round to one decimal place

outFile << name << std::endl;

outFile.close();

// Read and display data from "data.txt"

std::ifstream inFile("data.txt");

if (!inFile.is\_open()) {

std::cerr << "Error: Unable to open data.txt for reading." << std::endl;

return 1;

}

inFile >> age;

inFile >> height;

inFile.ignore(); // Ignore the newline character

std::getline(inFile, name);

inFile.close();

// Output: Display the user's age, height, and name

std::cout << age << std::endl;

std::cout << std::fixed << std::setprecision(1) << height << std::endl; // Display height in meters

std::cout << name << std::endl;

return 0;

}

//output

//Enter age: 3

//Enter height (in cm): 60.22

//Enter name: Emma G Copper

//3

//0.6

//Emma G Copper

**Unit 4:**

**Single File Programming Question**

**Problem Statement**

Once upon a time in a quaint village, there lived a brilliant young programmer named Alice. She was fascinated by the magical world of Programming and decided to create her very own class called CustomString.

This class had a special power - whenever the ++ operator was applied, it would transform the first letter of the input string into a majestic capital letter. One day, a curious traveler arrived in the village, and Alice eagerly demonstrated the power of her CustomString class.

The traveler was amazed by the enchanting customization and couldn't help but smile as the modified string was displayed, adding a touch of magic to their conversation.

**Input format :**

The input consists of a string consisting of alphanumeric characters (both uppercase and lowercase) and symbols.

**Refer to the sample input for further formatting specifications.**

**Output format :**

The program should display the original and modified strings on separate lines.

**Refer to the sample output for further formatting specifications.**

**Code constraints :**

The input string will contain at most 100 characters.

The input string may consist of spaces, special symbols, and alphanumeric characters.

The first character of the string may be uppercase or lowercase.

**Sample test cases :**

**Input 1 :**

hello World!

**Output 1 :**

Original string: hello World!

Modified string: Hello World!

**Input 2 :**

Apple

**Output 2 :**

Original string: Apple

Modified string: Apple

**Input 3 :**

$10000

**Output 3 :**

Original string: $10000

Modified string: $10000

#include <iostream>

#include <string>

class CustomString {

public:

CustomString(const std::string& inputString) {

originalString = inputString;

}

std::string modifyString() {

if (!originalString.empty()) {

originalString[0] = std::toupper(originalString[0]);

}

return originalString;

}

std::string getOriginalString() const {

return originalString;

}

private:

std::string originalString;

};

int main() {

std::string inputString;

std::getline(std::cin, inputString);

CustomString customString(inputString);

std::cout << "Original string: " << customString.getOriginalString() << std::endl;

std::cout << "Modified string: " << customString.modifyString() << std::endl;

return 0;

}

**Single File Programming Question**

**Problem Statement**

In a picturesque town, there lived a talented artist named Lily, who had a unique way of expressing herself through strings. She created a CustomString class that could magically invert the case of characters using the ! operator.

One day, a curious writer named Mark encountered Lily's creation and decided to put it to the test. Mark entered a captivating sentence into the console, and the CustomString class worked its charm, displaying the original sentence and its inverted counterpart.

Mark couldn't help but marvel at the power of operator overloading, realizing the endless possibilities it offered for crafting artistic and playful text transformations in their literary adventures.

**Note:** Use a character array.

**Input format :**

The input consists of a single line containing a string of characters. The string length does not exceed 100 characters.

**Output format :**

the output displays two lines of output:

The first line should contain the original string.

The second line should contain the inverted string after applying the unary ! operator.

**Refer to the sample output for further formatting specifications.**

**Code constraints :**

The input string will contain only printable ASCII characters (characters with ASCII values from 32 to 126).

The length of the input string will not exceed 100 characters.

**Sample test cases :**

**Input 1 :**

Hello WORLD

**Output 1 :**

Original String: Hello WORLD

Inverted String: hELLO world

#include <iostream>

#include <cstring>

class CustomString {

public:

CustomString(const char\* inputString) {

strcpy(originalString, inputString);

}

CustomString operator!() {

CustomString invertedString(originalString);

for (int i = 0; originalString[i] != '\0'; i++) {

if (std::islower(originalString[i])) {

invertedString.originalString[i] = std::toupper(originalString[i]);

} else if (std::isupper(originalString[i])) {

invertedString.originalString[i] = std::tolower(originalString[i]);

}

}

return invertedString;

}

const char\* getOriginalString() const {

return originalString;

}

private:

char originalString[101]; // Assuming a maximum of 100 characters

};

int main() {

char inputString[101]; // Assuming a maximum of 100 characters

std::cin.getline(inputString, sizeof(inputString));

CustomString customString(inputString);

CustomString invertedString = !customString;

std::cout << "Original String: " << customString.getOriginalString() << std::endl;

std::cout << "Inverted String: " << invertedString.getOriginalString() << std::endl;

return 0;

}

**Single File Programming Question**

**Problem Statement**

Help Aashiq solve the following problem: Create a program that performs string operations using a custom class called MyString.

The program should read two strings and a number from the user, and then perform concatenation and repetition operations on those strings using overloaded operators. Finally, it should display the results.

**Note:** This question helps in clearing HCL tests.

**Input format :**

The input consists of two strings: str1 and str2 in separate lines.

The last line of the input consists of the number of repetitions, n.

**Output format :**

The output prints the following in each line:

a) The concatenated string of first and second.

b) The first string that is repeated n times.

c) The second string that is repeated n times.

**Code constraints :**

The maximum length of each input string is 100 characters.

1 <= n <= 10

**Sample test cases :**

**Input 1 :**

Hello

helllooo

5

**Output 1 :**

Hellohelllooo

HelloHelloHelloHelloHello

helllooohelllooohelllooohelllooohelllooo

**Input 2 :**

Today

Today

5

**Output 2 :**

TodayToday

TodayTodayTodayTodayToday

TodayTodayTodayTodayToday

#include <iostream>

#include <string>

class MyString {

public:

MyString(const std::string& str) : value(str) {}

MyString operator+(const MyString& other) const {

return MyString(value + other.value);

}

MyString operator\*(int n) const {

std::string result;

for (int i = 0; i < n; i++) {

result += value;

}

return MyString(result);

}

std::string getValue() const {

return value;

}

private:

std::string value;

};

int main() {

std::string str1, str2;

int n;

std::getline(std::cin, str1);

std::getline(std::cin, str2);

std::cin >> n;

MyString myStr1(str1);

MyString myStr2(str2);

MyString concatenatedStr = myStr1 + myStr2;

MyString repeatedStr1 = myStr1 \* n;

MyString repeatedStr2 = myStr2 \* n;

std::cout << concatenatedStr.getValue() << std::endl;

std::cout << repeatedStr1.getValue() << std::endl;

std::cout << repeatedStr2.getValue() << std::endl;

return 0;

}

output

Hello

Helllo

5

HelloHelllo

HelloHelloHelloHelloHello

HellloHellloHellloHellloHelllo

**Single File Programming Question**

**Problem Statement**

In the kingdom of Geometria, there were two skilled architects, Lily and Max, known for their exceptional ability to design unique rectangles. They created a remarkable Rectangle class that could compare the size of rectangles using the **overloaded** '**=='** operator.

One day, the kingdom organized a grand architectural competition, and Lily and Max were invited to participate with their awe-inspiring rectangles. As the competition unfolded, they proudly presented their creations, each trying to prove the equality of their rectangles using the operator overloading concept.

The kingdom's people marveled at the elegant display of mathematical prowess, witnessing the magical operator determine whether the rectangles were equal in size or not.

Write a program to accomplish the above-given scenario.

**Input format :**

The first line of input represents the width and height of Rectangle 1, separated by a space.

The second line of input represents the width and height of Rectangle 2, separated by a space.

**Output format :**

The output consists of a single line with one of the following messages:

"Rectangle 1 and Rectangle 2 are equal in size." if Rectangle 1 and Rectangle 2 have the same width and height.

"Rectangle 1 and Rectangle 2 are not equal in size." if Rectangle 1 and Rectangle 2 have different widths or heights.

**Refer to the sample output for further formatting specifications.**

**Code constraints :**

* The width and height of each rectangle are integers.
* The width and height of both rectangles should be non-negative values.
* The width and height of each rectangle should not exceed the range of the integer data type.

**Sample test cases :**

**Input 1 :**

10 20

10 20

**Output 1 :**

Rectangle 1 and Rectangle 2 are equal in size.

**Input 2 :**

10 20

30 40

**Output 2 :**

Rectangle 1 and Rectangle 2 are not equal in size.

**Input 3 :**

10 20

20 10

**Output 3 :**

Rectangle 1 and Rectangle 2 are not equal in size.

#include <iostream>

class Rectangle {

public:

Rectangle(int w, int h) : width(w), height(h) {}

bool operator==(const Rectangle& other) const {

return (width == other.width) && (height == other.height);

}

private:

int width;

int height;

};

int main() {

int w1, h1, w2, h2;

std::cin >> w1 >> h1;

std::cin >> w2 >> h2;

Rectangle rectangle1(w1, h1);

Rectangle rectangle2(w2, h2);

if (rectangle1 == rectangle2) {

std::cout << "Rectangle 1 and Rectangle 2 are equal in size." << std::endl;

} else {

std::cout << "Rectangle 1 and Rectangle 2 are not equal in size." << std::endl;

}

return 0;

}

**Single File Programming Question**

**Problem Statement**

Peter is working on a project, and his task is to create a program that takes an input integer representing a total number of seconds and utilizes **basic to class type conversion** to convert it into a Time object. The Time class, defined in the code, has private member variables for hours, minutes, and seconds and a constructor that converts the input seconds into these time components. After inputting the total seconds, the program constructs a Time object and then calls its display() function to print the time in the format "**hours:minutes:seconds**".

For example, if the user enters 3661 seconds, the program will output "1:1:1" as it corresponds to 1 hour, 1 minute, and 1 second.

**Note:** This kind of question will help in clearing Wipro recruitment.

**Input format :**

The input contains an integer representing the time in seconds.

**Output format :**

The output displays the converted time in the following format: "hours:minutes:seconds"

**Refer to the sample output for formatting specifications.**

**Code constraints :**

0 <= seconds <= 86400

**Sample test cases :**

**Input 1 :**

3661

**Output 1 :**

1:1:1

**Input 2 :**

45034

**Output 2 :**

12:30:34

#include <iostream>

class Time {

public:

Time(int seconds) {

hours = seconds / 3600;

seconds %= 3600;

minutes = seconds / 60;

seconds %= 60;

this->seconds = seconds;

}

void display() const {

std::cout << hours << ":" << minutes << ":" << this->seconds << std::endl;

}

private:

int hours;

int minutes;

int seconds;

};

int main() {

int inputSeconds;

std::cin >> inputSeconds;

Time time(inputSeconds);

time.display();

return 0;

}

**Single File Programming Question**

**Problem Statement**

Jacob is working in banking software, and he wants to convert the currency from US Dollars to Euros. Help him write a program to perform currency conversion from US Dollars (USD) to Euros (EUR) using a class named CurrencyConverter. The program should allow the user to input an amount in US Dollars and a conversion rate (how many Euros one US Dollar is worth), and then it should display the converted amount in Euros (EUR). **Basic to class-type conversion** should be employed to achieve this.

**Input format :**

The first line of the input consists of a double value representing the conversion rate.

The second line consists of a double value representing the amount in US Dollars.

**Output format :**

**The output displays the amount in Euros with two decimal values.**

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

1.0 <= conversion rate <= 100.0

1.0 <= US Dollars <= 105

**Sample test cases :**

**Input 1 :**

30.5

100.0

**Output 1 :**

Converted amount in Euros (EUR): 3050.00 EUR

**Input 2 :**

78.9

1550.0

**Output 2 :**

Converted amount in Euros (EUR): 122295.00 EUR

#include <iostream>

#include <iomanip>

class CurrencyConverter {

public:

CurrencyConverter(double conversionRate) {

rate = conversionRate;

}

double convertToEuros(double usDollars) {

return usDollars \* rate;

}

private:

double rate;

};

int main() {

double conversionRate, usDollars;

std::cin >> conversionRate;

std::cin >> usDollars;

CurrencyConverter converter(conversionRate);

double euros = converter.convertToEuros(usDollars);

std::cout << std::fixed << std::setprecision(2);

std::cout << "Converted amount in Euros (EUR): " << euros << " EUR" << std::endl;

return 0;

}

**Single File Programming Question**

**Problem Statement**

Senthil is working in a grocery store, and he wants to create a program for managing stock items. Help him implement a StockItem class with private member variables for item code, quantity, and price per unit. Use **basic to class type conversion** to enable the creation of a StockItem object by providing input for item code, quantity, and price per unit. The program should calculate and display the total price for the stock item and print its details in a specific format.

**Note:** This kind of question will help in clearing TCS recruitment.

**Input format :**

The first line of the input consists of an integer representing the item code.

The second line consists of an integer representing the quantity.

The third line consists of a double value representing the price per unit.

**Output format :**

The output displays the total price of the stock with two decimal places.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

1 <= item code <= 1000

1 <= quantity <= 1000

1 <= price per unit <= 105

**Sample test cases :**

**Input 1 :**

103

10

150.0

**Output 1 :**

Total Price of the Stock: 1500.00

**Input 2 :**

999

42

1750.6

**Output 2 :**

Total Price of the Stock: 73525.20

#include <iostream>

#include <iomanip>

class StockItem {

public:

StockItem(int code, int qty, double price) {

itemCode = code;

quantity = qty;

pricePerUnit = price;

}

double getTotalPrice() {

return quantity \* pricePerUnit;

}

void display() {

std::cout << std::fixed << std::setprecision(2);

std::cout << "Total Price of the Stock: " << getTotalPrice() << std::endl;

}

private:

int itemCode;

int quantity;

double pricePerUnit;

};

int main() {

int itemCode, quantity;

double pricePerUnit;

std::cin >> itemCode >> quantity >> pricePerUnit;

StockItem stockItem(itemCode, quantity, pricePerUnit);

stockItem.display();

return 0;

}

**Single File Programming Question**

**Problem Statement**

In the enchanting realm of Mathematica, a wise sorcerer named Merlin created a powerful Fraction class capable of performing magical fraction operations. One day, a young wizard apprentice named Arthur stumbled upon Merlin's creation and became intrigued by its potential.

Eager to test its abilities, Arthur devised a spellbinding program that allowed users to input two fractions as basic data types (numerator/denominator) and perform multiplication. Harnessing the magic of **basic to class type conversion**, the fractions were effortlessly transformed into Fraction class objects, and the multiplication result was beautifully displayed in its reduced form.

As Arthur witnessed the seamless transformation, he realized the incredible potential of the basic-to-class-type conversion in simplifying complex mathematical operations in their mystical adventures.

**Input format :**

The input consists of two fractions in the format "numerator denominator".

Each fraction should be provided on a separate line, where each line contains the numerator followed by space and the denominator.

**Output format :**

The output prints the result of multiplication in the format "numerator/denominator."

**Refer to the sample output for further formatting specifications.**

**Code constraints :**

* The numerators and denominators of the fractions are integers.
* The denominators should be non-zero.

**Sample test cases :**

**Input 1 :**

44 88

100 200

**Output 1 :**

The result of multiplication is: 1/4

#include <iostream>

#include <algorithm>

class Fraction {

public:

Fraction(int numerator, int denominator) {

int gcd = std::\_\_gcd(numerator, denominator);

this->numerator = numerator / gcd;

this->denominator = denominator / gcd;

}

Fraction operator\*(const Fraction& other) const {

return Fraction(numerator \* other.numerator, denominator \* other.denominator);

}

void display() const {

std::cout << numerator << "/" << denominator << std::endl;

}

private:

int numerator;

int denominator;

};

int main() {

int numerator1, denominator1, numerator2, denominator2;

std::cin >> numerator1 >> denominator1;

std::cin >> numerator2 >> denominator2;

Fraction fraction1(numerator1, denominator1);

Fraction fraction2(numerator2, denominator2);

Fraction result = fraction1 \* fraction2;

std::cout << "The result of multiplication is: ";

result.display();

return 0;

}

**Single File Programming Question**

**Problem Statement**

Write a program to create a class **Speed** that allows users to input a speed value in kilometers per hour (km/h) and then convert and display it in meters per second (m/s). The program should use class type to basic type conversion to achieve this functionality.

**Formula:**

1km = 1000 meters

1hour = 3600 seconds

**Input format :**

The input consists of a floating-point number representing the speed in kilometers per hour (k/h).

**Output format :**

The output displays a float value after converting to meters per second (m/s), rounded off to three decimal places.

**Refer to the sample output for the exact text.**

**Sample test cases :**

**Input 1 :**

1.1

**Output 1 :**

0.306 m/s

**Input 2 :**

12.9

**Output 2 :**

3.583 m/s

#include <iostream>

#include <iomanip>

class Speed {

public:

Speed() {}

operator double() const {

double kmPerHour;

std::cin >> kmPerHour;

return kmPerHour \* 1000.0 / 3600.0;

}

};

int main() {

Speed;

double speedMetersPerSecond = speed;

std::cout << std::fixed << std::setprecision(3);

std::cout << speedMetersPerSecond << " m/s" << std::endl;

return 0;

}

**Single File Programming Question**

**Problem Statement**

Once upon a time, in a futuristic city, there lived a brilliant programmer named Alex. Alex had developed an innovative Matrix class that could handle large datasets efficiently. The Matrix class could take input, display the matrix, and even convert the matrix to a column-major array.

However, one day, while testing the program, Alex encountered a mysterious bug. Can you help Alex identify the issue and find out how the **class type to basic type conversion** in the Matrix class might be affecting the column-major array generation?

Join Alex on this thrilling coding adventure to uncover the secret of the column-major array!

**Input format :**

The first line of input consists of two integers, 'rows' and 'cols', are separated by a space, representing the dimensions of the matrix.

The next 'rows' lines, each containing 'cols' integers separated by spaces, represent the elements of the matrix.

**Output format :**

The output displays the matrix in row-major order, followed by the column-major array representation of the matrix.

**Refer to the sample output for further formatting specifications.**

**Code constraints :**

* The matrix dimensions (rows and columns) will be positive integers not exceeding 100.
* The matrix elements can be any integer.

**Sample test cases :**

**Input 1 :**

2 3

12 54 72 10 37 15

**Output 1 :**

Matrix:

12 54 72

10 37 15

Column-Major Array:

12 10 54 37 72 15

#include <iostream>

#include <vector>

class Matrix {

public:

Matrix(int rows, int cols) : rows(rows), cols(cols), data(rows, std::vector<int>(cols)) {}

void input() {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

std::cin >> data[i][j];

}

}

}

void display() {

std::cout << "Matrix:" << std::endl;

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

std::cout << data[i][j] << " ";

}

std::cout << std::endl;

}

}

operator std::vector<int>() {

std::vector<int> columnMajorArray;

for (int j = 0; j < cols; j++) {

for (int i = 0; i < rows; i++) {

columnMajorArray.push\_back(data[i][j]);

}

}

return columnMajorArray;

}

private:

int rows;

int cols;

std::vector<std::vector<int>> data;

};

int main() {

int rows, cols;

std::cin >> rows >> cols;

Matrix matrix(rows, cols);

matrix.input();

matrix.display();

std::vector<int> columnMajorArray = matrix;

std::cout << "\nColumn-Major Array:" << std::endl;

for (int i = 0; i < columnMajorArray.size(); i++) {

std::cout << columnMajorArray[i] << " ";

}

std::cout << std::endl;

return 0;

}

**Single File Programming Question**

**Problem Statement**

Once upon a time, in the peaceful town of Willowbrook, there lived a wise old wizard named Merlin. He had a magical book that could convert the birth year of any person into their current age.

However, to use the book, he needed to ask for the name and birth year of the person. As he flipped through the ancient pages of the book, Merlin discovered a young traveler named Alice. Intrigued by her journey, Merlin asked Alice for her name, and she replied, "I am Alice." The wizard then requested her birth year, to which she responded, "I was born in 2001."

Finally, Merlin asked Alice for the current year, and she said, "It's 2023." With all the information gathered, Merlin chanted a spell, and the magical book converted Alice's birth year into her current age using a **class type to basic type conversion**. How old was Alice, and what year was she born?

**Input format :**

The first line of input consists of a string, which represents the name of the person.

The second line consists of an integer representing the birth year of the person.

The third line consists of an integer representing the current year.

**Output format :**

The output displays a single line in the following format:

"Age: [age] years"

**Refer to the sample output for further formatting specifications.**

**Code constraints :**

* The name can be any non-empty string without spaces
* The birth year and current year should be valid integers
* birthYear <= currentYear

**Sample test cases :**

**Input 1 :**

Udhesh

2002

2023

**Output 1 :**

Age: 21 years

#include <iostream>

#include <string>

class Person {

public:

Person(const std::string& name, int birthYear, int currentYear)

: name(name), birthYear(birthYear), currentYear(currentYear) {}

int calculateAge() const {

return currentYear - birthYear;

}

void displayAge() const {

std::cout << "Age: " << calculateAge() << " years" << std::endl;

}

private:

std::string name;

int birthYear;

int currentYear;

};

int main() {

std::string name;

int birthYear, currentYear;

std::cin >> name >> birthYear >> currentYear;

Person person(name, birthYear, currentYear);

person.displayAge();

return 0;

}

**Single File Programming Question**

**Problem Statement**

Nandhini is tasked with creating a program to calculate and display the speed of a car based on the distance covered and the time taken.

Write a program with two classes: **Vehicle** as the base class and **Car** as the derived class, which inherits the properties from the Vehicle class for calculating the speed. The base class fetches the input as a float value, whereas the derived class calculates and prints the output as a float value.

**Note:** Use public inheritance

**Note:** This kind of question will help in clearing Wipro recruitment.

**Input format :**

The input consists of two floating-point numbers separated by a space. The first number represents the distance traveled by the car, and the second number represents the time taken to cover that distance, separated by a space.

**Output format :**

The output displays a single line of output, showing the speed of the car in kilometers per hour.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

The input distance and time should be non-negative floating-point numbers.

The input time should not be zero (to avoid division by zero).

**Sample test cases :**

**Input 1 :**

15.0 5.0

**Output 1 :**

Speed of car: 3 km/hr

**Input 2 :**

100.23 96.0

**Output 2 :**

Speed of car: 1.04406 km/hr

#include <iostream>

#include <iomanip>

class Vehicle {

public:

Vehicle() {

std::cin >> distance >> time;

}

protected:

float distance;

float time;

};

class Car : public Vehicle {

public:

void calculateSpeed() {

if (time == 0.0) {

std::cout << "Time taken cannot be zero." << std::endl;

return;

}

float speed = distance / time;

std::cout << std::fixed << std::setprecision(5);

std::cout << "Speed of car: " << speed << " km/hr" << std::endl;

}

};

int main() {

Car;

car.calculateSpeed();

return 0;

}

**Single File Programming Question**

**Problem statement**

You are working on a bicycle manufacturing project, and your task is to implement a system that handles different types of bicycles. You have a base class called "**Bicycle"**, which contains information about the gear and speed of the bicycle. The class provides methods to apply brakes to reduce speed and speed up the bicycle.

Now, you need to create a specialized class named **"MountainBike"**, which inherits from the Bicycle class. The **"MountainBike"** class represents a specific type of bicycle suitable for off-road mountain biking. Apart from the gear and speed, the mountain bike also has an additional attribute called **"seatHeight"**, which represents the height of the bike's seat.

Your task is to design the classes **"Bicycle"** and **"MountainBike"** using inheritance and implement the necessary member functions. The program should prompt the user to enter the gear, speed, and initial seat height of a mountain bike. Then, it should create a MountainBike object with the provided data and display all the information about the mountain bike, including gear, speed, and seat height.

**Input format :**

The input consists of three integers, which represent the number of gears, speed, and seat height, separated by a space.

**Output format :**

The output displays the number of gears, speed of the bicycle, and seat height.

**Refer to the sample output for format specifications.**

**Code constraints :**

1<=number of gears<=6

0<=speed<=100

0<=height<=60

**Sample test cases :**

**Input 1 :**

2 90 40

**Output 1 :**

No of gears are 2

Speed of bicycle is 90

Seat height is 40

**Input 2 :**

3 60 20

**Output 2 :**

No of gears are 3

Speed of bicycle is 60

Seat height is 20

#include <iostream>

class Bicycle {

protected:

int gears;

int speed;

public:

Bicycle(int \_gears, int \_speed) : gears(\_gears), speed(\_speed) {}

void applyBrakes(int decrement) {

speed -= decrement;

}

void speedUp(int increment) {

speed += increment;

}

void displayInfo() {

std::cout << "No of gears are " << gears << std::endl;

std::cout << "Speed of bicycle is " << speed << std::endl;

}

};

class MountainBike : public Bicycle {

private:

int seatHeight;

public:

MountainBike(int \_gears, int \_speed, int \_seatHeight) : Bicycle(\_gears, \_speed), seatHeight(\_seatHeight) {}

void displayInfo() {

Bicycle::displayInfo();

std::cout << "Seat height is " << seatHeight << std::endl;

}

};

int main() {

int gears, speed, seatHeight;

std::cin >> gears >> speed >> seatHeight;

MountainBike mountainBike(gears, speed, seatHeight);

mountainBike.displayInfo();

return 0;

}

A company maintains a database that has the details of all its employees.

There are two levels of employees:

1. Level 1 is the top management, which has a salary of more than 100 dollars,
2. Level 2 is the staff, which has a salary of less than 100 dollars.

Create a class named "**Employee**" with **empId** and **salary** as attributes.

Create another class, "**empLevel**" that inherits **Employee** and categorizes the employee into various levels.

**Note:** Use single inheritance.

**Input format :**

The input consists of employee ID and salary of the employee, separated by spaces.

Employee ID should be of integer type and salary float type.

**Output format :**

The output displays the employee ID, salary, and level of the employee, in separate lines.

**Refer to the sample output for format specifications.**

**Code constraints :**

1<= empId <=10000

1<= salary <=100000

**Sample test cases :**

**Input 1 :**

253 5.6

**Output 1 :**

Employee ID: 253

Salary: $5.60

Level: 2

#include <iostream>

#include <iomanip>

class Employee {

protected:

int empId;

float salary;

public:

Employee(int id, float sal) : empId(id), salary(sal) {}

void displayDetails() {

std::cout << "Employee ID: " << empId << std::endl;

std::cout << "Salary: $" << std::fixed << std::setprecision(2) << salary << std::endl;

}

};

class empLevel : public Employee {

public:

empLevel(int id, float sal) : Employee(id, sal) {}

void determineLevel() {

if (salary > 100.0) {

std::cout << "Level: 1" << std::endl;

} else {

std::cout << "Level: 2" << std::endl;

}

}

};

int main() {

int empId;

float salary;

std::cin >> empId >> salary;

empLevel emp(empId, salary);

emp.displayDetails();

emp.determineLevel();

return 0;

}

**Unit 5**

Whitelist

Set 1:

virtual

class Investment

class Stock

class Bond

calculateReturns()

You have been assigned the task of developing an investment returns calculation program. The program should allow users to input details about their investments and calculate the returns based on specific formulas. The program should follow these specifications:

Implement an abstract class called Investment with a pure virtual method calculateReturns(). Create concrete classes, such as Stock and Bond, which inherit from the Investment class, to calculate returns for different types of investments.

**Note**: This kind of question will help in clearing Accenture recruitment.

Input format :

For each investment type:

The first line of the input contains the initial investment amount.

The next line contains the final value.

Output format :

For each investment type:

The first line of the output displays the type

The second line displays the Initial Investment.

The next line displays the Final Value.

The last line displays the calculated returns in percentage.

**Refer to the sample output for formatting specifications.**

Code constraints :

The initial investment amount and final value should be non-negative real numbers.

Sample test cases :

Input 1 :

1000

1500

2000

5000

Output 1 :

Type: Stock

Initial Investment: 1000.00

Final Value: 1500.00

Returns: 50.00%

Type: Bond

Initial Investment: 2000.00

Final Value: 5000.00

Returns: 150.00%

Input 2 :

250

120

350

480

Output 2 :

Type: Stock

Initial Investment: 250.00

Final Value: 120.00

Returns: -52.00%

Type: Bond

Initial Investment: 350.00

Final Value: 480.00

Returns: 37.14%

#include <iostream>

#include <iomanip>

class Investment {

public:

Investment(double initial, double final) : initialInvestment(initial), finalValue(final) {}

virtual void calculateReturns() = 0;

void displayInvestmentInfo(const std::string& type) {

std::cout << "Type: " << type << "\n";

std::cout << "Initial Investment: " << std::fixed << std::setprecision(2) << initialInvestment << "\n";

std::cout << "Final Value: " << finalValue << "\n";

std::cout << "Returns: " << std::fixed << std::setprecision(2) << returns << "%" << "\n";

}

protected:

double initialInvestment;

double finalValue;

double returns;

};

class Stock : public Investment {

public:

Stock(double initial, double final) : Investment(initial, final) {}

void calculateReturns() override {

returns = ((finalValue - initialInvestment) / initialInvestment) \* 100;

}

};

class Bond : public Investment {

public:

Bond(double initial, double final) : Investment(initial, final) {}

void calculateReturns() override {

returns = ((finalValue - initialInvestment) / initialInvestment) \* 100;

}

};

int main() {

double initialStock, finalStock, initialBond, finalBond;

std::cin >> initialStock;

std::cin >> finalStock;

std::cin >> initialBond;

std::cin >> finalBond;

Stock stockInvestment(initialStock, finalStock);

Bond bondInvestment(initialBond, finalBond);

stockInvestment.calculateReturns();

bondInvestment.calculateReturns();

stockInvestment.displayInvestmentInfo("Stock");

bondInvestment.displayInvestmentInfo("Bond");

return 0;

}

Satha is working on a project that involves number manipulation using object-oriented programming. Your task is to create a class hierarchy that allows you to perform various operations on an integer and display the result. The base class, "**A**", is an abstract class with a pure virtual function, **print()**, while the derived class, "**B**", implements this function and provides additional functionalities for number operations.

Write a program to find whether the given number(x) is even or odd; if it is even, then print the cube(x)+square(x) otherwise, print cube(x)-square(x).

Create a base class with a pure virtual function, void **print()**. Print the result by implementing this virtual function in the derived class.

Input format :

The input consists of an Integer.

Output format :

If the given number is even, the output will be the sum of the cube and square of the number.

If the given number is odd, the output will be the difference between the cube and square of the number.

Code constraints :

1<=n<=100

Sample test cases :

Input 1 :

5

Output 1 :

100

Input 2 :

4

Output 2 :

80

Note :

The program will be evaluated only after the “Submit Code” is clicked.

Extra spaces and new line characters in the program output will result in the failure of the test case.

Marks : 10

Negative Marks : 0

Whitelist

Set 1:

virtual

Set 2:

class

#include <iostream>

class A {

public:

A(int x) : num(x) {}

virtual void print() = 0;

protected:

int num;

};

class B : public A {

public:

B(int x) : A(x) {}

void print() override {

int result;

if (num % 2 == 0) {

result = (num \* num \* num) + (num \* num);

} else {

result = (num \* num \* num) - (num \* num);

}

std::cout << result << std::endl;

}

};

int main() {

int x;

std::cin >> x;

B obj(x);

obj.print();

return 0;

}

Design a program to manage student information using dynamic constructor and memory allocation. The program should allow users to create and manage student objects with customizable attributes such as name, age, and enrolled courses.

Implement a dynamic constructor that takes the student's name, age, and the number of courses as parameters and allocates memory dynamically to store the course names. The program should handle memory deallocation in the destructor.

Additionally, design methods to set and retrieve the student attributes, check if the student is enrolled in a specific course, and display the student details.

Input format :

The input consists of the following:

First line: The student's name as a string of alphanumeric characters.

Second line: The student's age as a positive integer.

Third line: The number of courses (n) the student is enrolled in as a positive integer.

Fourth line onwards: For each course, The name of the course as a string of alphanumeric characters.

Output format :

The output displays the student's name, age, and courses(space-separated).

The following output displays the results of checking if the student is enrolled in a specific course, which can be "Yes" or "No".

Code constraints :

1 <= n <= 15

Sample test cases :

Input 1 :

Nisha

19

3

Physics

Chemistry

Botany

Zoology

Output 1 :

Student Details:

Name: Nisha

Age: 19

Courses: Physics Chemistry Botany

Is the student enrolled in Zoology? No

Input 2 :

Nisha

19

3

Physics

Chemistry

Botany

Chemistry

Output 2 :

Student Details:

Name: Nisha

Age: 19

Courses: Physics Chemistry Botany

Is the student enrolled in Chemistry? Yes

Note :

The program will be evaluated only after the “Submit Code” is clicked.

Extra spaces and new line characters in the program output will result in the failure of the test case.

Marks : 10

Negative Marks : 0

Whitelist

Set 1:

class Student

Student

#include <iostream>

#include <cstring>

class Student {

public:

Student(const char\* name, int age, int numCourses) {

this->name = new char[strlen(name) + 1];

strcpy(this->name, name);

this->age = age;

this->numCourses = numCourses;

courses = new char\*[numCourses];

}

~Student() {

delete[] name;

for (int i = 0; i < numCourses; i++) {

delete[] courses[i];

}

delete[] courses;

}

void setCourses(const char\* courseName, int index) {

courses[index] = new char[strlen(courseName) + 1];

strcpy(courses[index], courseName);

}

bool isEnrolledInCourse(const char\* courseName) {

for (int i = 0; i < numCourses; i++) {

if (strcmp(courseName, courses[i]) == 0) {

return true;

}

}

return false;

}

void displayDetails() {

std::cout << "Student Details:" << std::endl;

std::cout << "Name: " << name << std::endl;

std::cout << "Age: " << age << std::endl;

std::cout << "Courses: ";

for (int i = 0; i < numCourses; i++) {

std::cout << courses[i] << " ";

}

std::cout << std::endl;

}

private:

char\* name;

int age;

int numCourses;

char\*\* courses;

};

int main() {

char name[100];

int age, numCourses;

std::cin.getline(name, 100);

std::cin >> age;

std::cin >> numCourses;

Student student(name, age, numCourses);

for (int i = 0; i < numCourses; i++) {

char courseName[100];

std::cin >> courseName;

student.setCourses(courseName, i);

}

student.displayDetails();

char courseToCheck[100];

std::cin >> courseToCheck;

if (student.isEnrolledInCourse(courseToCheck)) {

std::cout << "Is the student enrolled in " << courseToCheck << "? Yes" << std::endl;

} else {

std::cout << "Is the student enrolled in " << courseToCheck << "? No" << std::endl;

}

return 0;

}

You are working for a renowned software company that develops various utility applications. One of the applications requires a feature to convert decimal numbers to octal numbers. To achieve this functionality, you are provided with a template that includes a base class "**BaseConverter"** and a derived class "**DecimalToOctalConverter"**.

The "**BaseConverter**" class has a **pure virtual function** convert, which must be overridden in the "**DecimalToOctalConverter**" class. The convert function should take an integer representing a decimal number and return its octal equivalent. The company requires the use of **late** **binding**, also known as dynamic polymorphism, to enable runtime selection of the appropriate convert function based on the object's type.

Your task is to implement the DecimalToOctalConverter class while adhering to late binding principles. Ensure that the conversion from a given decimal number to its octal representation is correct and that the code uses dynamic dispatch to handle different object types.

**Note:** This kind of question will help in clearing Amcat recruitment.

Input format :

The input represents an integer.

Output format :

The output displays the octal number for the given number.

**Refer to the sample output for formatting specifications.**

Code constraints :

-1000<=number<=1000

Sample test cases :

Input 1 :

42

Output 1 :

52

Input 2 :

0

Output 2 :

0

Note :

The program will be evaluated only after the “Submit Code” is clicked.

Extra spaces and new line characters in the program output will result in the failure of the test case.

Marks : 10

Negative Marks : 0

Whitelist

Set 1:

virtual

class

#include <iostream>

#include <string>

#include <sstream>

class BaseConverter {

public:

virtual std::string convert(int number) = 0;

};

class DecimalToOctalConverter : public BaseConverter {

public:

std::string convert(int number) override {

if (number == 0) {

return "0";

}

std::string octal;

int absNumber = std::abs(number);

while (absNumber > 0) {

int remainder = absNumber % 8;

octal = std::to\_string(remainder) + octal;

absNumber /= 8;

}

if (number < 0) {

octal = "-" + octal;

}

return octal;

}

};

int main() {

int number;

std::cin >> number;

DecimalToOctalConverter converter;

std::string octal = converter.convert(number);

std::cout << octal << std::endl;

return 0;

}