**Q1. WAP to find whether the given number is even or odd.**

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (num % 2 == 0)

cout << num << " is an Even number." << endl;

else

cout << num << " is an Odd number." << endl;

return 0;

}

**Q2. WAP to find Given number is prime or composite.**

#include <iostream>

using namespace std;

int main() {

int num, i, count = 0;

cout << "Enter a number: ";

cin >> num;

if (num <= 1) {

cout << num << " is neither Prime nor Composite." << endl;

} else {

for (i = 1; i <= num; i++) {

if (num % i == 0)

count++;

}

if (count == 2)

cout << num << " is a Prime number." << endl;

else

cout << num << " is a Composite number." << endl;

} return 0;

}

**Q3. WAP to print the table of given number upto n multiples.**

#include <iostream>

using namespace std;

int main() {

int num, n;

cout << "Enter a number: ";

cin >> num;

cout << "Enter the number of multiples: ";

cin >> n;

cout << "Multiplication Table of " << num << " up to " << n << " multiples:" << endl;

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

cout << num << " x " << i << " = " << num \* i << endl;

}

return 0;

}

**Q4.WAP to find:**

**i)greater of two numbers**

#include <iostream>

using namespace std;

int main() {

int num1, num2;

cout << "Enter two numbers: ";

cin >> num1 >> num2;

if (num1 > num2)

cout << num1 << " is greater than " << num2 << endl;

else if (num2 > num1)

cout << num2 << " is greater than " << num1 << endl;

else

cout << "Both numbers are equal." << endl;

return 0;

}

**ii)greatest of three numbers.**

#include <iostream>

using namespace std;

int main() {

int num1, num2, num3;

cout << "Enter three numbers: ";

cin >> num1 >> num2 >> num3;

if (num1 >= num2 && num1 >= num3)

cout << num1 << " is the greatest." << endl;

else if (num2 >= num1 && num2 >= num3)

cout << num2 << " is the greatest." << endl;

else

cout << num3 << " is the greatest." << endl;

return 0;

}

**Q5. WAP to find the sum of first n natural numbers.**

#include <iostream>

using namespace std;

int main() {

int n, sum = 0;

cout << "Enter a number: ";

cin >> n;

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

sum += i;

}

cout << "Sum of first " << n << " natural numbers is: " << sum << endl;

return 0;

}

**Q6. WAP to find factorial of given number.**

#include <iostream>

using namespace std;

int main() {

int num;

long long factorial = 1; // Using long long to handle large factorial values

cout << "Enter a number: ";

cin >> num;

if (num < 0) {

cout << "Factorial of a negative number is not defined." << endl;

} else {

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

factorial \*= i;

}

cout << "Factorial of " << num << " is: " << factorial << endl;

}

return 0;

}

**Q7. WAP to find sum of digits of n digit number.**

#include <iostream>

using namespace std;

int main() {

int num, sum = 0, digit;

cout << "Enter a number: ";

cin >> num;

while (num > 0) {

digit = num % 10; // Extract last digit

sum += digit; // Add to sum

num /= 10; // Remove last digit

}

cout << "Sum of digits: " << sum << endl;

return 0;

}

**Q8. WAP to find reverse of a number.**

#include <iostream>

using namespace std;

int main() {

int num, reverseNum = 0, digit;

cout << "Enter a number: ";

cin >> num;

while (num > 0) {

digit = num % 10; // Extract last digit

reverseNum = reverseNum \* 10 + digit;

num /= 10; // Remove last digit

}

cout << "Reversed number: " << reverseNum << endl;

return 0;

}

**Q9. WAP to determine given number is palindrome or not.**

#include <iostream>

using namespace std;

int main() {

int num, originalNum, reverseNum = 0, digit;

cout << "Enter a number: ";

cin >> num;

originalNum = num; // Store original number

while (num > 0) {

digit = num % 10; // Extract last digit

reverseNum = reverseNum \* 10 + digit;

num /= 10; // Remove last digit

}

if (originalNum == reverseNum)

cout << originalNum << " is a Palindrome number." << endl;

else

cout << originalNum << " is NOT a Palindrome number." << endl;

return 0;

}

**Q10. WAP to print Fibonacci series up to n terms.**

#include <iostream>

using namespace std;

int main() {

int n, first = 0, second = 1, next;

cout << "Enter the number of terms: ";

cin >> n;

cout << "Fibonacci Series: ";

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

cout << first << " "; // Print the current term

next = first + second; // Calculate next term

first = second; // Shift first to second

second = next; // Shift second to next

}

cout << endl;

return 0;

}

**Q11. WAP to determine given n digit number is Armstrong or not.**

#include <iostream>

#include <cmath> // For pow() function

using namespace std;

int main() {

int num, originalNum, sum = 0, digit, n = 0;

cout << "Enter a number: ";

cin >> num;

originalNum = num;

// Count the number of digits

int temp = num;

while (temp > 0) {

temp /= 10;

n++;

}

// Calculate sum of nth power of each digit

temp = num;

while (temp > 0) {

digit = temp % 10;

sum += pow(digit, n); // digit^n

temp /= 10;

}

if (sum == originalNum)

cout << originalNum << " is an Armstrong number." << endl;

else

cout << originalNum << " is NOT an Armstrong number." << endl;

return 0;

}

**Q12. WAP to print all even numbers between 100 and 200.**

#include <iostream>

using namespace std;

int main() {

cout << "Even numbers between 100 and 200:" << endl;

for (int i = 100; i <= 200; i += 2) {

cout << i << " ";

}

cout << endl;

return 0;

}

**Q13. WAP to print first 50 prime numbers.**

#include <iostream>

using namespace std;

int main() {

int count = 0, num = 2; // Start from 2

cout << "First 50 Prime Numbers:" << endl;

while (count < 50) {

int i;

for (i = 2; i < num; i++) { // Check if num is prime

if (num % i == 0)

break; // Not prime, exit loop

}

// If loop runs completely, num is prime

if (i == num) {

cout << num << " ";

count++;

}

num++; // Check next number

}

cout << endl;

return 0;

}

**Q14. WAP to print all 4 digit Armstrong number.**

#include <iostream>

#include <cmath>

using namespace std;

int main() {

cout << "4-digit Armstrong numbers:" << endl;

for (int num = 1000; num <= 9999; num++) {

int sum = 0, temp = num, digit;

while (temp > 0) {

digit = temp % 10; // Extract last digit

sum += pow(digit, 4); // Compute 4th power

temp /= 10; // Remove last digit

}

// If Armstrong number, print it

if (sum == num) {

cout << num << " ";

}

}

cout << endl;

return 0;

}

**Q15. WAP to print following patterns:**

**i).**

**\***

**\*\***

**\*\*\***

**\*\*\*\***

**\*\*\*\*\***

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter the number of rows: ";

cin >> n;

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

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

cout << "\*";

}

cout << endl;

}

return 0;

}

**ii)**

**\*\*\*\*\***

**\*\*\*\***

**\*\*\***

**\*\***

**\***

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter the number of rows: ";

cin >> n;

for (int i = n; i >= 1; i--) {

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

cout << "\*";

}

cout << endl;

}

return 0;

}

**iii)**

**\***

**\*\*\***

**\*\*\*\*\***

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter the number of rows: ";

cin >> n;

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

for (int j = 1; j <= n - i; j++) // Print spaces

cout << " ";

for (int j = 1; j <= 2 \* i - 1; j++) // Print stars

cout << "\*";

cout << endl;

}

return 0;

}

**iv)**

**1**

**22**

**333**

**4444**

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter the number of rows: ";

cin >> n;

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

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

cout << i;

}

cout << endl;

}

return 0;

}

**v) Pascal's triangle**

#include <iostream>

using namespace std;

int main() {

int rows, i, j, num;

cout << "Enter the number of rows for Pascal's Triangle: ";

cin >> rows;

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

num = 1; // First number in each row is always 1

for (j = 0; j < rows - i - 1; j++) {

cout << " ";

}

for (j = 0; j <= i; j++) { // Print Pascal's Triangle numbers

cout << num << " "; // Adjust spacing for alignment

num = num \* (i - j) / (j + 1); // Calculate next number in the row

}

cout << endl;

}

return 0;

}

**vi) Floyd's triangle**

#include <iostream>

using namespace std;

int main() {

int n, num = 1;

cout << "Enter the number of rows: ";

cin >> n;

for (int i = 1; i <= n; i++) { // Row loop

for (int j = 1; j <= i; j++) { // Column loop

cout << num << " ";

num++; // Increment number

}

cout << endl;

}

return 0;

}

**Q16. Using functions, write following c++ programs.**

**i) To print all the palindromes for a range 500-1000.**

#include <iostream>

using namespace std;

// Custom function to reverse a number

int reverseNumber(int num) {

int reversed = 0, digit;

while (num > 0) {

digit = num % 10; // Extract last digit

reversed = reversed \* 10 + digit; // Append digit to reversed number

num /= 10; // Remove last digit

}

return reversed;

}

// Custom function to print all palindromes in a given range

void printPalindromes(int start, int end) {

cout << "Palindromes between " << start << " and " << end << ":\n";

for (int i = start; i <= end; i++) {

if (i == reverseNumber(i)) { // Compare number with its reverse

cout << i << " ";

}

}

cout << endl;

}

int main() {

printPalindromes(500, 1000); // Call function with range 500-1000

return 0;

}

**ii) To print first 100 odd numbers.**

#include <iostream>

using namespace std;

// Function to print the first 100 odd numbers

void printOddNumbers(int count) {

int num = 1; // First odd number

cout << "First " << count << " odd numbers:\n";

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

cout << num << " ";

num += 2; // Increment by 2 to get the next odd number

}

cout << endl;

}

int main() {

printOddNumbers(100); // Call function to print first 100 odd numbers

return 0;

}

**v) To calculate geometric sum up to n terms.**

#include <iostream>

#include <cmath> // For pow() function

using namespace std;

// Function to calculate geometric sum

double geometricSum(int n, double a, double r) {

double sum = 0;

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

sum += a \* pow(r, i); // Formula: a \* r^i

}

return sum;

}

int main() {

int n;

double a, r;

// User input

cout << "Enter the first term (a): ";

cin >> a;

cout << "Enter the common ratio (r): ";

cin >> r;

cout << "Enter the number of terms (n): ";

cin >> n;

// Calculate and display the sum

double result = geometricSum(n, a, r);

cout << "Geometric sum up to " << n << " terms: " << result << endl;

return 0;

}

**Q17. Using recursion make a c++ program to:**

**i)To print binary number for a decimal number.**

#include <iostream>

using namespace std;

// Recursive function to convert decimal to binary

void decimalToBinary(int num) {

if (num == 0)

return; // Base case: Stop when num becomes 0

decimalToBinary(num / 2); // Recursive call with quotient

cout << (num % 2); // Print remainder (binary digit)

}

int main() {

int decimal;

cout << "Enter a decimal number: ";

cin >> decimal;

if (decimal == 0)

cout << "0"; // Special case for 0

else

decimalToBinary(decimal); // Call recursive function

cout << endl;

return 0;

}

**ii) To print octal number for a decimal number.**

#include <iostream>

using namespace std;

// Recursive function to convert decimal to octal

void decimalToOctal(int num) {

if (num == 0)

return; // Base case: Stop when num becomes 0

decimalToOctal(num / 8); // Recursive call with quotient

cout << (num % 8); // Print remainder (octal digit)

}

int main() {

int decimal;

cout << "Enter a decimal number: ";

cin >> decimal;

if (decimal == 0)

cout << "0"; // Special case for 0

else

decimalToOctal(decimal); // Call recursive function

cout << endl;

return 0;

}

**iii) To print factorials for a given range.**

#include <iostream>

using namespace std;

// Recursive function to calculate factorial of a number

long long factorial(int num) {

if (num == 0 || num == 1)

return 1; // Base case: Factorial of 0 and 1 is 1

return num \* factorial(num - 1); // Recursive call

}

// Recursive function to print factorials in a given range

void printFactorials(int start, int end) {

if (start > end)

return; // Base case: Stop when start exceeds end

cout << "Factorial of " << start << " is " << factorial(start) << endl;

printFactorials(start + 1, end); // Recursive call for next number

}

int main() {

int start, end;

cout << "Enter the start of range: ";

cin >> start;

cout << "Enter the end of range: ";

cin >> end;

if (start < 0 || end < 0 || start > end) {

cout << "Invalid range! Please enter non-negative values with start <= end." << endl;

} else {

printFactorials(start, end); // Call recursive function

}

return 0;

}

**iv) To print first n terms of Fibonacci series.**

#include <iostream>

using namespace std;

// Recursive function to compute Fibonacci numbers

int fibonacci(int n) {

if (n == 0) return 0; // Base case: F(0) = 0

if (n == 1) return 1; // Base case: F(1) = 1

return fibonacci(n - 1) + fibonacci(n - 2); // Recursive formula

}

// Recursive function to print Fibonacci series up to n terms

void printFibonacci(int count, int index = 0) {

if (index == count)

return; // Base case: Stop when we reach 'count' terms

cout << fibonacci(index) << " "; // Print Fibonacci number

printFibonacci(count, index + 1); // Recursive call for next term

}

int main() {

int n;

cout << "Enter the number of terms: ";

cin >> n;

if (n <= 0) {

cout << "Please enter a positive number." << endl;

} else {

cout << "Fibonacci Series: ";

printFibonacci(n); // Call recursive function

cout << endl;

}

return 0;

}

**Q18. WAP to calculate average of all elements of 1D array.**

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n], sum = 0;

cout << "Enter " << n << " elements: ";

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

cin >> arr[i]; // Input array elements

sum += arr[i]; // Calculate sum

}

double avg = (double)sum / n; // Calculate average

cout << "Average of array elements: " << avg << endl;

return 0;

}

**Q19. WAP to find out minimum and maximum value of a 1D numeric array.**

#include <iostream>

using namespace std;

int main() {

int n;

// Taking input for array size

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

// Taking input for array elements

cout << "Enter " << n << " elements: ";

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

cin >> arr[i];

}

// Initialize min and max with the first element of the array

int minVal = arr[0], maxVal = arr[0];

// Loop through the array to find min and max

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

if (arr[i] < minVal)

minVal = arr[i]; // Update min value

if (arr[i] > maxVal)

maxVal = arr[i]; // Update max value

}

// Display the results

cout << "Minimum value: " << minVal << endl;

cout << "Maximum value: " << maxVal << endl;

return 0;

}

**Q20. WAP to find transpose of a 2D matrix.**

#include <iostream>

using namespace std;

int main() {

int rows, cols;

// Taking input for matrix size

cout << "Enter the number of rows and columns: ";

cin >> rows >> cols;

int matrix[rows][cols], transpose[cols][rows];

// Taking input for matrix elements

cout << "Enter the matrix elements:\n";

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

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

cin >> matrix[i][j];

}

}

// Finding transpose

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

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

transpose[j][i] = matrix[i][j]; // Swapping rows and columns

}

}

// Displaying the transpose matrix

cout << "Transpose of the matrix:\n";

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

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

cout << transpose[i][j] << " ";

}

cout << endl;

}

return 0;

}

**Q21. WAP to add 2D matrices.**

#include <iostream>

using namespace std;

int main() {

int rows, cols;

// Taking input for matrix size

cout << "Enter the number of rows and columns: ";

cin >> rows >> cols;

int matrix1[rows][cols], matrix2[rows][cols], sum[rows][cols];

// Taking input for first matrix

cout << "Enter elements of first matrix:\n";

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

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

cin >> matrix1[i][j];

}

}

// Taking input for second matrix

cout << "Enter elements of second matrix:\n";

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

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

cin >> matrix2[i][j];

}

}

// Adding both matrices

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

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

sum[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

// Displaying the sum matrix

cout << "Sum of matrices:\n";

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

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

cout << sum[i][j] << " ";

}

cout << endl;

}

return 0;

}

**Q22. WAP to multiply 2D matrices.**

#include <iostream>

using namespace std;

int main() {

int rows1, cols1, rows2, cols2;

// Taking input for first matrix size

cout << "Enter rows and columns for first matrix: ";

cin >> rows1 >> cols1;

// Taking input for second matrix size

cout << "Enter rows and columns for second matrix: ";

cin >> rows2 >> cols2;

// Checking if matrix multiplication is possible

if (cols1 != rows2) {

cout << "Matrix multiplication not possible! Number of columns of first matrix must equal rows of second matrix." << endl;

return 0;

}

int matrix1[rows1][cols1], matrix2[rows2][cols2], result[rows1][cols2];

// Taking input for first matrix

cout << "Enter elements of first matrix:\n";

for (int i = 0; i < rows1; i++) {

for (int j = 0; j < cols1; j++) {

cin >> matrix1[i][j];

}

}

// Taking input for second matrix

cout << "Enter elements of second matrix:\n";

for (int i = 0; i < rows2; i++) {

for (int j = 0; j < cols2; j++) {

cin >> matrix2[i][j];

}

}

// Initializing result matrix to 0

for (int i = 0; i < rows1; i++) {

for (int j = 0; j < cols2; j++) {

result[i][j] = 0;

}

}

// Multiplying matrices

for (int i = 0; i < rows1; i++) {

for (int j = 0; j < cols2; j++) {

for (int k = 0; k < cols1; k++) {

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

}

}

}

// Displaying the result matrix

cout << "Resultant matrix after multiplication:\n";

for (int i = 0; i < rows1; i++) {

for (int j = 0; j < cols2; j++) {

cout << result[i][j] << " ";

}

cout << endl;

}

return 0;

}

**Q23. WAP to sort an array in ascending order.**

#include <iostream>

using namespace std;

int main() {

int n;

// Taking input for array size

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

// Taking input for array elements

cout << "Enter " << n << " elements: ";

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

cin >> arr[i];

}

// Sorting using Bubble Sort

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Swap adjacent elements if they are in the wrong order

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

// Displaying the sorted array

cout << "Sorted array in ascending order: ";

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

cout << arr[i] << " ";

}

cout << endl;

return 0;

}

**Q24. WAP to reverse a given string.**

#include <iostream>

#include <cstring> // For strlen()

using namespace std;

int main() {

char str[100];

// Taking input

cout << "Enter a string: ";

cin.getline(str, 100);

int length = strlen(str); // Find string length

// Reversing the string using two-pointer approach

for (int i = 0, j = length - 1; i < j; i++, j--) {

swap(str[i], str[j]); // Swap characters

}

// Display reversed string

cout << "Reversed string: " << str << endl;

return 0;

}

**Q25. WAP to count all vowels in a given string.**

#include <iostream>

using namespace std;

int main() {

string str;

int count = 0;

cout << "Enter a string: ";

cin >> str;

// Checking each character

for (int i = 0; i < str.length(); i++) {

char ch = tolower(str[i]); // Convert to lowercase to handle both cases

if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {

count++;

}

} // Display result

cout << "Number of vowels: " << count << endl;

return 0;

}

**Q26. WAP to check if the given string is palindrome or not.**

#include <iostream>

using namespace std;

int main() {

string str;

int i, n, flag = 1; // flag = 1 assumes palindrome

// Taking input

cout << "Enter a string: ";

cin >> str;

n = str.length();

// Checking for palindrome

for (i = 0; i < n / 2; i++) {

if (str[i] != str[n - i - 1]) {

flag = 0; // Not a palindrome

break;

}

}

// Display result

if (flag == 1)

cout << "The string is a palindrome." << endl;

else

cout << "The string is not a palindrome." << endl;

return 0;

}

**Q27. WAP to check if a given string is anagram or not.**

#include <iostream>

#include <algorithm> // For sort function

using namespace std;

int main() {

string str1, str2;

// Taking input

cout << "Enter first string: ";

cin >> str1;

cout << "Enter second string: ";

cin >> str2;

// Check if lengths are different

if (str1.length() != str2.length()) {

cout << "Strings are not anagrams." << endl;

return 0;

}

// Sorting both strings

sort(str1.begin(), str1.end());

sort(str2.begin(), str2.end());

// Comparing sorted strings

if (str1 == str2)

cout << "Strings are anagrams." << endl;

else

cout << "Strings are not anagrams." << endl;

return 0;

}

**Q28. Define a class called Car with attributes such as make, model, and year. Include member functions to set and get these attributes. Create an object of the Car class and demonstrate the use of its member functions.**

#include <iostream>

using namespace std;

class Car {

private:

string make;

string model;

int year;

public:

// Setter functions

void setMake(string m) {

make = m;

}

void setModel(string m) {

model = m;

}

void setYear(int y) {

year = y;

}

// Getter functions

string getMake() {

return make;

}

string getModel() {

return model;

}

int getYear() {

return year;

}

// Display function

void displayCar() {

cout << "Car Details:\n";

cout << "Make: " << make << "\nModel: " << model << "\nYear: " << year << endl;

}

};

int main() {

// Creating an object of the Car class

Car myCar;

// Setting attributes using setter functions

myCar.setMake("Toyota");

myCar.setModel("Corolla");

myCar.setYear(2022);

// Displaying car details

myCar.displayCar();

// Getting individual attributes

cout << "\nUsing Getters:" << endl;

cout << "Make: " << myCar.getMake() << endl;

cout << "Model: " << myCar.getModel() << endl;

cout << "Year: " << myCar.getYear() << endl;

return 0;

}

**Q29. Define a class called Address with attributes such as street, city, and zipCode. Create a class called Person that has an Address object as a member variable. Demonstrate composition by creating a Person object and accessing its Address attributes.**

#include <iostream>

using namespace std;

// Address class

class Address {

private:

string street;

string city;

int zipCode;

public:

// Constructor

Address(string s, string c, int z) {

street = s;

city = c;

zipCode = z;

}

// Display function

void displayAddress() {

cout << "Street: " << street << "\nCity: " << city << "\nZip Code: " << zipCode << endl;

}

};

// Person class (Has an Address object)

class Person {

private:

string name;

Address address; // Composition: Person "has an" Address

public:

// Constructor

Person(string n, Address addr) : address(addr) {

name = n;

}

// Display function

void displayPerson() {

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

address.displayAddress(); // Access Address attributes

}

};

int main() {

// Creating an Address object

Address myAddress("123 Main St", "New York", 10001);

// Creating a Person object and associating it with an Address

Person myPerson("John Doe", myAddress);

// Displaying Person and Address details

myPerson.displayPerson();

return 0;

}

**Q30. Define a class student with the following specification**

**Private members of class student**

**admno                        integer**

**sname                        20 character**

**eng. math, science       float**

**total                            float**

**Public member function of class student**

**ctotal()                        a function to calculate eng + math + science with float return type.**

**Takedata()                   Function to accept values for admno, sname, eng, science Showdata()                   Function to display all the data members on the screen.**

#include <iostream>

#include <cstring> // For string handling

using namespace std;

class Student {

private:

int admno;

char sname[20];

float eng, math, science;

float total;

// Function to calculate total marks

float ctotal() {

return eng + math + science;

}

public:

// Function to accept values

void Takedata() {

cout << "Enter Admission Number: ";

cin >> admno;

cin.ignore(); // Ignore newline character in buffer

cout << "Enter Student Name: ";

cin.getline(sname, 20);

cout << "Enter marks in English, Math, and Science: ";

cin >> eng >> math >> science;

total = ctotal(); // Calculate total marks

}

// Function to display values

void Showdata() {

cout << "\nStudent Details:\n";

cout << "Admission No: " << admno << endl;

cout << "Name: " << sname << endl;

cout << "English: " << eng << endl;

cout << "Math: " << math << endl;

cout << "Science: " << science << endl;

cout << "Total Marks: " << total << endl;

}

};

int main() {

// Creating a Student object

Student s1;

// Taking input

s1.Takedata();

// Displaying details

s1.Showdata();

return 0;

}

**Q31. Define a class in C++ with following description:**

**Private Members**

**A data member Flight number of type integer**

**A data member Destination of type string**

**A data member Distance of type float**

**A data member Fuel of type float**

**A member function CALFUEL() to calculate the value of Fuel as per the following criteria**

**Distance                                                          Fuel**

**<=1000                                                           500**

**more than 1000  and <=2000                          1100**

**more than 2000                                              2200**

**Public Members**

**A function FEEDINFO() to allow user to enter values for Flight Number, Destination, Distance & call function CALFUEL() to calculate the quantity of Fuel.**

**A function SHOWINFO() to allow user to view the content of all the data members.**

#include <iostream>

using namespace std;

class Flight {

private:

int flightNumber;

string destination;

float distance;

float fuel;

// Function to calculate fuel based on distance

void CALFUEL() {

if (distance <= 1000)

fuel = 500;

else if (distance > 1000 && distance <= 2000)

fuel = 1100;

else

fuel = 2200;

}

public:

// Function to input flight details and calculate fuel

void FEEDINFO() {

cout << "Enter Flight Number: ";

cin >> flightNumber;

cin.ignore(); // Ignore newline character in buffer

cout << "Enter Destination: ";

getline(cin, destination);

cout << "Enter Distance: ";

cin >> distance;

CALFUEL(); // Calculate fuel requirement

}

// Function to display flight details

void SHOWINFO() {

cout << "\nFlight Details:\n";

cout << "Flight Number: " << flightNumber << endl;

cout << "Destination: " << destination << endl;

cout << "Distance: " << distance << " km" << endl;

cout << "Required Fuel: " << fuel << " liters" << endl;

}

};

int main() {

Flight f1; // Creating a Flight object

// Taking input

f1.FEEDINFO();

// Displaying details

f1.SHOWINFO();

return 0;

}

**Q32. Write a menu driven program to perform following:**

**a) Input a matrix**

**b) Display matrix**

**c) Add two matrix**

**d) Multiply two matrixes**

**e) Transpose a matrix**

#include <iostream>

using namespace std;

#define SIZE 10 // Maximum matrix size

class Matrix {

private:

int mat[SIZE][SIZE];

int rows, cols;

public:

// Function to input matrix

void inputMatrix() {

cout << "Enter number of rows and columns: ";

cin >> rows >> cols;

cout << "Enter matrix elements:\n";

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

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

cin >> mat[i][j];

}

}

}

// Function to display matrix

void displayMatrix() {

cout << "Matrix:\n";

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

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

cout << mat[i][j] << " ";

}

cout << endl;

}

}

// Function to add two matrices

Matrix addMatrix(Matrix m2) {

Matrix result;

if (rows != m2.rows || cols != m2.cols) {

cout << "Error: Matrices must have the same dimensions for addition.\n";

return result;

}

result.rows = rows;

result.cols = cols;

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

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

result.mat[i][j] = mat[i][j] + m2.mat[i][j];

}

}

return result;

}

// Function to multiply two matrices

Matrix multiplyMatrix(Matrix m2) {

Matrix result;

if (cols != m2.rows) {

cout << "Error: Column of first matrix must be equal to row of second matrix for multiplication.\n";

return result;

}

result.rows = rows;

result.cols = m2.cols;

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

for (int j = 0; j < m2.cols; j++) {

result.mat[i][j] = 0;

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

result.mat[i][j] += mat[i][k] \* m2.mat[k][j];

}

}

}

return result;

}

// Function to compute transpose

Matrix transposeMatrix() {

Matrix result;

result.rows = cols;

result.cols = rows;

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

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

result.mat[j][i] = mat[i][j];

}

}

return result;

}

};

int main() {

Matrix m1, m2, result;

int choice;

while (true) {

cout << "\nMenu:\n";

cout << "1. Input Matrix\n";

cout << "2. Display Matrix\n";

cout << "3. Add Two Matrices\n";

cout << "4. Multiply Two Matrices\n";

cout << "5. Transpose Matrix\n";

cout << "6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

m1.inputMatrix();

break;

case 2:

m1.displayMatrix();

break;

case 3:

cout << "Enter second matrix:\n";

m2.inputMatrix();

result = m1.addMatrix(m2);

result.displayMatrix();

break;

case 4:

cout << "Enter second matrix:\n";

m2.inputMatrix();

result = m1.multiplyMatrix(m2);

result.displayMatrix();

break;

case 5:

result = m1.transposeMatrix();

result.displayMatrix();

break;

case 6:

cout << "Exiting program.\n";

return 0;

default:

cout << "Invalid choice. Try again.\n";

}

}

}