**OBJECT ORIENTED PROGRAMMING**

**OPERATOR AND CONTROL STRUCTURES**

1. **Write a program to read in two integers and perform the following operations on them: addition, subtraction, multiplication, division, and modulo.**

#include <iostream>

using namespace std;

int main() {

int num1, num2;

// Read two integers from the user

cout << "Enter the first integer: ";

cin >> num1;

cout << "Enter the second integer: ";

cin >> num2;

// Perform addition

int addition = num1 + num2;

cout << "Addition: " << addition << endl;

// Perform subtraction

int subtraction = num1 - num2;

cout << "Subtraction: " << subtraction << endl;

// Perform multiplication

int multiplication = num1 \* num2;

cout << "Multiplication: " << multiplication << endl;

// Perform division (handling division by zero)

if (num2 != 0) {

double division = static\_cast<double>(num1) / num2;

cout << "Division: " << division << endl;

} else {

cout << "Division: Cannot divide by zero" << endl;

}

// Perform modulo (handling division by zero)

if (num2 != 0) {

int modulo = num1 % num2;

cout << "Modulo: " << modulo << endl;

} else {

cout << "Modulo: Cannot perform modulo operation when divisor is zero" << endl;

}

return 0;

}

1. **Program to determine the integer is odd or even**

#include <iostream>

using namespace std;

int main() {

int number;

// Read an integer from the user

cout << "Enter an integer: ";

cin >> number;

// Check if the number is even or odd

if (number % 2 == 0) {

cout << number << " is even." << endl;

} else {

cout << number << " is odd." << endl;

}

return 0;

}

1. **Program to compute the average of three integers**

#include <iostream>

using namespace std;

int main() {

int num1, num2, num3;

double average;

// Read three integers from the user

cout << "Enter the first integer: ";

cin >> num1;

cout << "Enter the second integer: ";

cin >> num2;

cout << "Enter the third integer: ";

cin >> num3;

// Calculate the average

average = (num1 + num2 + num3) / 3.0;

// Display the average

cout << "The average of " << num1 << ", " << num2 << ", and " << num3 << " is: " << average << endl;

return 0;

}

1. **Program to check two numbers are equal or not**

#include <iostream>

using namespace std;

int main() {

int num1, num2;

cout << "Enter the first number: ";

cin >> num1;

cout << "Enter the second number: ";

cin >> num2;

if (num1 == num2) {

cout << "The numbers " << num1 << " and " << num2 << " are equal." << endl;

} else {

cout << "The numbers " << num1 << " and " << num2 << " are not equal." << endl;

}

return 0;

}

1. **Write a program to read in two Floating numbers and perform the following operations on them: addition, subtraction, multiplication, division, and modulo.**

#include <iostream>

#include <cmath>

using namespace std;

int main() {

float num1, num2;

cout << "Enter the first floating-point number: ";

cin >> num1;

cout << "Enter the second floating-point number: ";

cin >> num2;

cout << "Addition: " << num1 + num2 << endl;

cout << "Subtraction: " << num1 - num2 << endl;

cout << "Multiplication: " << num1 \* num2 << endl;

if (num2 != 0) {

cout << "Division: " << num1 / num2 << endl;

} else {

cout << "Division: Cannot divide by zero" << endl;

}

cout << "Modulo: " << fmod(num1, num2) << endl;

return 0;

}

1. **Program to check the character is a vowel or consonant.**

#include <iostream>

using namespace std;

int main() {

char ch;

cout << "Enter a character: ";

cin >> ch;

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

ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U') {

cout << ch << " is a vowel." << endl;

}

else if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z')) {

cout << ch << " is a consonant." << endl;

}

else {

cout << "Invalid input. Please enter an alphabet." << endl;

}

return 0;

}

1. **Program to check the number is positive, negative or zero.**

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (num > 0)

cout << num << " is positive." << endl;

else if (num < 0)

cout << num << " is negative." << endl;

else

cout << "The number is zero." << endl;

return 0;

}

1. **Program to determine which number is greater among two integers.**

#include<iostream>

using namespace std;

int main()

{

int num1, num2;

cout<<"enter num1:";

cin>>num1;

cout<<"enter num2:";

cin>>num2;

if (num1 > num2){

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

} else if (num2 > num1){

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

}

else{

cout<<"incorrect value";

}

return 0;

}

1. **Program to read a floating-number and round it to the nearest integer using the floor an ceil functions.**

#include <iostream>

#include <cmath>

using namespace std;

int main() {

float number;

cout << "Enter a floating-point number: ";

cin >> number;

int roundedDown = floor(number);

int roundedUp = ceil(number);

cout << "Original number: " << number << endl;

cout << "Rounded down: " << roundedDown << endl;

cout << "Rounded up: " << roundedUp << endl;

return 0;

}

1. **Program to swap two numbers using bitwise XOR operator.**

#include <iostream>

using namespace std;

int main() {

int num1, num2;

cout << "Enter two numbers: ";

cin >> num1 >> num2;

cout << "Before swapping: num1 = " << num1 << ", num2 = " << num2 << endl;

num1 = num1 ^ num2;

num2 = num1 ^ num2;

num1 = num1 ^ num2;

cout << "After swapping: num1 = " << num1 << ", num2 = " << num2 << endl;

return 0;

}

1. **Largest among three numbers using ternary conditional operator.**

#include <iostream>

using namespace std;

int main() {

int a, b, c, largest;

cout << "Enter three numbers: ";

cin >> a >> b >> c;

largest = (a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c);

cout << "The largest number is: " << largest << endl;

return 0;

}

1. **smallest among three numbers using ternary conditional operator.**

#include <iostream>

using namespace std;

int main() {

int num1, num2, num3, smallest;

cout << "Enter three numbers: ";

cin >> num1 >> num2 >> num3;

smallest = (num1 < num2) ? ((num1 < num3) ? num1 : num3) : ((num2 < num3) ? num2 : num3);

cout << "The smallest number is: " << smallest << endl;

return 0;

}

1. **Program to check the integer is divisible by 3 or not using ternary conditional operator.**

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter an integer: ";

cin >> num;

string result = (num % 3 == 0) ? "Divisible by 3" : "Not divisible by 3";

cout << result << endl;

return 0;

}

1. **Program to print numbers from 1 to 10 using for loop.**

#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 10; ++i) {

cout << i << " ";

}

return 0;

}

1. **Factorial of a number using for loop.**

#include <iostream>

using namespace std;

int main() {

int n;

long long factorial = 1;

cout << "Enter a positive integer: ";

cin >> n;

if (n < 0)

cout << "Error! Factorial of a negative number doesn't exist.";

else {

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

factorial \*= i;

}

cout << "Factorial of " << n << " = " << factorial;

}

return 0;

}

1. **Print multiplication table using for loop.**

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter a positive integer: ";

cin >> num;

for (int i = 1; i <= 10; ++i) {

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

}

return 0;

}

1. **Fibonacci series using for loop.**

#include <iostream>

using namespace std;

int main() {

int n, t1 = 0, t2 = 1, nextTerm = 0;

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

cin >> n;

cout << "Fibonacci Series: ";

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

if (i == 1) {

cout << t1 << ", ";

continue;

}

if (i == 2) {

cout << t2 << ", ";

continue;

}

nextTerm = t1 + t2;

t1 = t2;

t2 = nextTerm;

cout << nextTerm << ", ";

}

return 0;

}

1. **Prime number using for loop.**

#include <iostream>

using namespace std;

int main() {

int n, i;

bool isPrime = true;

cout << "Enter a positive integer: ";

cin >> n;

if (n == 0 || n == 1) {

isPrime = false;

}

else {

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

if (n % i == 0) {

isPrime = false;

break;

}

}

}

if (isPrime)

cout << n << " is a prime number";

else

cout << n << " is not a prime number";

return 0;

}

1. **Check the string is palindrome or not using while loop.**

#include <iostream>

#include <string>

#include <cctype>

bool isPalindrome(const std::string& str) {

int left = 0;

int right = str.length() - 1;

while (left < right) {

while (!isalnum(str[left]) && left < right) {

left++;

}

while (!isalnum(str[right]) && left < right) {

right--;

}

if (tolower(str[left]) != tolower(str[right])) {

return false;

}

left++;

right--;

}

return true;

}

int main() {

std::string input;

std::cout << "Enter a string: ";

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

if (isPalindrome(input)) {

std::cout << input << " is a palindrome.\n";

} else {

std::cout << input << " is not a palindrome.\n";

}

return 0;

}

1. **Sum of all digits using while loop (n=123 output:1+2+3=6)**

#include <iostream>

int main() {

int n;

std::cout << "Enter a number: ";

std::cin >> n;

int sum = 0;

int remainder;

while (n > 0) {

remainder = n % 10; // Get the last digit

sum += remainder; // Add the last digit to the sum

n /= 10; // Remove the last digit from the number

}

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

return 0;

}

1. **GCD of two numbers using do-while loop.**

#include <iostream>

int main() {

int num1, num2;

std::cout << "Enter first number: ";

std::cin >> num1;

std::cout << "Enter second number: ";

std::cin >> num2;

int gcd;

do {

gcd = num1 % num2;

num1 = num2;

num2 = gcd;

} while (num2 != 0);

std::cout << "GCD of the two numbers is: " << num1 << std::endl;

return 0;

}

1. **Check whether the number is perfect or not.**

#include <iostream>

bool isPerfect(int number) {

int sum = 0;

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

if (number % i == 0) {

sum += i;

}

}

return sum == number;

}

int main() {

int number;

std::cout << "Enter a number: ";

std::cin >> number;

if (isPerfect(number)) {

std::cout << number << " is a perfect number." << std::endl;

} else {

std::cout << number << " is not a perfect number." << std::endl;

}

return 0;

}

1. **Armstrong number.**

#include <iostream>

#include <cmath>

bool isArmstrong(int number) {

int sum = 0;

int numDigits = static\_cast<int>(std::log10(number)) + 1;

int temp = number;

while (temp != 0) {

int digit = temp % 10;

sum += std::pow(digit, numDigits);

temp /= 10;

}

return sum == number;

}

int main() {

int number;

std::cout << "Enter a number: ";

std::cin >> number;

std::cout << number << (isArmstrong(number) ? " is" : " is not") << " an Armstrong number." << std::endl;

return 0;

}

1. **Harshad number.**

#include <iostream>

bool isHarshad(int number) {

int sumOfDigits = 0;

int temp = number;

while (temp != 0) {

sumOfDigits += temp % 10;

temp /= 10;

}

return number % sumOfDigits == 0;

}

int main() {

int number;

std::cout << "Enter a number: ";

std::cin >> number;

if (isHarshad(number))

std::cout << number << " is a Harshad number." << std::endl;

else

std::cout << number << " is not a Harshad number." << std::endl;

return 0;

}

1. Happy number.

#include <iostream>

#include <unordered\_set>

int digitSquareSum(int n) {

int sum = 0;

while (n > 0) {

int digit = n % 10;

sum += digit \* digit;

n /= 10;

}

return sum;

}

bool isHappy(int n) {

std::unordered\_set<int> seen;

while (n != 1 && seen.insert(n).second) {

n = digitSquareSum(n);

}

return n == 1;

}

int main() {

int number;

std::cout << "Enter a number: ";

std::cin >> number;

std::cout << number << (isHappy(number) ? " is" : " is not") << " a Happy number." << std::endl;

return 0;

}

1. **strong number**

#include <iostream>

using namespace std;

bool isStrong(int num) {

int originalNum = num;

int sum = 0;

while (num > 0) {

int digit = num % 10;

int factorial = 1;

for (int i = 1; i <= digit; ++i)

factorial \*= i;

sum += factorial;

num /= 10;

}

return sum == originalNum;

}

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (isStrong(num))

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

else

cout << num << " is not a strong number." << endl;

return 0;

}

1. **buzz number**

#include <iostream>

using namespace std;

bool isBuzz(int num) {

return (num % 7 == 0) || (num % 10 == 7);

}

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (isBuzz(num))

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

else

cout << num << " is not a buzz number." << endl;

return 0;

}

1. **neon number**

#include <iostream>

using namespace std;

bool isNeon(int num) {

int square = num \* num;

int sum = 0;

while (square > 0) {

sum += square % 10;

square /= 10;

}

return sum == num;

}

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (isNeon(num))

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

else

cout << num << " is not a neon number." << endl;

return 0;

}

1. **abundant number**

#include <iostream>

using namespace std;

bool isAbundant(int num) {

int sum = 0;

for (int i = 1; i <= num / 2; ++i) {

if (num % i == 0)

sum += i;

}

return sum > num;

}

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (isAbundant(num))

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

else

cout << num << " is not an abundant number." << endl;

return 0;

}

1. **narcissistic number**

#include <iostream>

#include <cmath>

using namespace std;

bool isNarcissistic(int num) {

int originalNum = num;

int sum = 0;

int numDigits = 0;

while (num > 0) {

numDigits++;

num /= 10;

}

num = originalNum;

while (num > 0) {

int digit = num % 10;

sum += pow(digit, numDigits);

num /= 10;

}

return sum == originalNum;

}

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (isNarcissistic(num))

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

else

cout << num << " is not a narcissistic number." << endl;

return 0;

}

1. **print the pattern 1 22 333 4444 55555**

#include <iostream>

using namespace std;

void printPattern() {

for (int i = 1; i <= 5; ++i) {

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

cout << i;

}

cout << endl;

}

}

int main() {

printPattern();

return 0;

}

1. **print the pattern \* \*\* \*\*\* \*\*\*\* \*\*\*\*\***

#include <iostream>

using namespace std;

void printPattern() {

for (int i = 1; i <= 5; ++i) {

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

cout << "\*";

}

cout << endl;

}

}

int main() {

printPattern();

return 0;

}

1. **Print pascal triangle pattern nested for loop**

#include <iostream>

using namespace std;

void printPascalTriangle(int n) {

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

int C = 1; // used to represent C(line, i)

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

cout << C << " ";

C = C \* (line - i) / i;

}

cout << endl;

}

}

int main() {

int rows;

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

cin >> rows;

printPascalTriangle(rows);

return 0;

}

1. **Print diamond pattern with \* using nested for loop**

#include <iostream>

using namespace std;

void printDiamondPattern(int n) {

int i, j, space = 1;

space = n - 1;

for (j = 1; j <= n; j++) {

for (i = 1; i <= space; i++)

cout << " ";

space--;

for (i = 1; i <= 2 \* j - 1; i++)

cout << "\*";

cout << endl;

}

space = 1;

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

for (i = 1; i <= space; i++)

cout << " ";

space++;

for (i = 1; i <= 2 \* (n - j) - 1; i++)

cout << "\*";

cout << endl;

}

}

int main() {

int rows;

cout << "Enter the number of rows for the diamond pattern: ";

cin >> rows;

printDiamondPattern(rows);

return 0;

}

1. Program to reverse the elements in an array
2. Program to insert an element in an array at a specific position
3. Program to Delete an element in an array at a specific position
4. Find the sum of all elements in an array
5. Find the average of all elements in an array
6. Find the second largest element in an array
7. Find the number of occurrences of a value in an array
8. Merge two array
9. Create a dynamic array using pointers and display the values
10. Create a dynamic 2D (Two dimensional) array using pointers and display the values
11. Add 2 matrices
12. Multiply 2 matrices
13. Find the sum of diagonals of a matrix

**Functions in C++**

1. **Find factorial using function.**

#include <iostream>

unsigned long long factorial(int n) {

if (n == 0 || n == 1) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

int main() {

int number;

cout << "Enter a number: ";

cin >> number;

if (number < 0) {

cout << "Factorial is not defined for negative numbers." << endl;

} else {

unsigned long long result = factorial(number);

cout << "Factorial of " << number << " is: " << result << endl;

}

return 0;

}

1. **Find prime number using function.**

#include <iostream>

bool isPrime(int n) {

if (n <= 1) {

return false; // 0 and 1 are not prime numbers

}

for (int i = 2; i \* i <= n; ++i) {

if (n % i == 0) {

return false; // If n is divisible by any number between 2 and sqrt(n), it's not prime

}

}

return true;

}

int main() {

int number;

cout << "Enter a number: ";

cin >> number;

if (isPrime(number)) {

cout << number << " is a prime number." << endl;

} else {

cout << number << " is not a prime number." <<endl;

}

return 0;

}

1. **Find the reverse of a string using function.**

#include <iostream>

#include <string>

string reverseString(const std::string& str) {

string reversedStr;

for (int i = str.length() - 1; i >= 0; --i) {

reversedStr += str[i];

}

return reversedStr;

}

int main() {

string inputString;

cout << "Enter a string: ";

getline(cin, inputString);

string reversedString = reverseString(inputString);

cout << "Reversed string: " << reversedString << sendl;

return 0;

}

1. **Find minimum and maximum element in an array using function.**

#include <iostream>

void findMinAndMax(int arr[], int size, int& min, int& max) {

min = arr[0];

max = arr[0];

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

if (arr[i] < min) {

min = arr[i];

}

if (arr[i] > max) {

max = arr[i];

}

}

}

int main() {

int size;

std::cout << "Enter the size of the array: ";

std::cin >> size;

int arr[size];

std::cout << "Enter elements of the array: ";

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

std::cin >> arr[i];

}

int min, max;

findMinAndMax(arr, size, min, max);

std::cout << "Minimum element: " << min << std::endl;

std::cout << "Maximum element: " << max << std::endl;

return 0;

}

1. **Find GCD of two number using function.**

#include <iostream>

using namespace std;

// Function to find GCD of two numbers

int gcd(int a, int b) {

if (b == 0) {

return a;

}

return gcd(b, a % b);

}

int main() {

int num1, num2;

cout << "Enter first number: ";

cin >> num1;

cout << "Enter second number: ";

cin >> num2;

int result = gcd(num1, num2);

cout << "GCD of " << num1 << " and " << num2 << " is: " << result << endl;

return 0;

}

1. **Function to count the no of elements in a string.**

#include <iostream>

// Function to count the number of elements in a string

int countElements(const std::string& str) {

return str.length();

}

int main() {

std::string inputString;

std::cout << "Enter a string: ";

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

int count = countElements(inputString);

std::cout << "Number of elements in the string: " << count << std::endl;

return 0;

}

1. **Convert Celsius and Fahrenheit using function.**

#include <iostream>

using namespace std;

double celsiusToFahrenheit(double celsius) {

return (celsius \* 9.0 / 5.0) + 32.0;

}

double fahrenheitToCelsius(double fahrenheit) {

return (fahrenheit - 32.0) \* 5.0 / 9.0;

}

int main() {

double celsius, fahrenheit;

cout << "Enter temperature in Celsius: ";

cin >> celsius;

fahrenheit = celsiusToFahrenheit(celsius);

cout << "Temperature in Fahrenheit: " << fahrenheit << endl;

cout << "Enter temperature in Fahrenheit: ";

cin >> fahrenheit;

celsius = fahrenheitToCelsius(fahrenheit);

cout << "Temperature in Celsius: " << celsius << endl;

return 0;

}

1. **Find the area of a circle using function.**

#include <iostream>

using namespace std;

double calculateArea(double radius) {

return 3.14159 \* radius \* radius; // π \* r^2

}

int main() {

double radius, area;

cout << "Enter the radius of the circle: ";

cin >> radius;

area = calculateArea(radius);

cout << "The area of the circle with radius " << radius << " is: " << area << endl;

return 0;

}

1. **Check whether the string is palindrome or not.**

#include <iostream>

#include <string>

using namespace std;

bool isPalindrome(const string& str) {

int start = 0;

int end = str.length() - 1;

while (start < end) {

if (str[start] != str[end]) {

return false;

}

start++;

end--;

}

return true;

}

int main() {

string inputString;

cout << "Enter a string: ";

getline(cin, inputString);

if (isPalindrome(inputString)) {

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

} else {

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

}

return 0;

}

**Constructor and destructor**

1. **Write a c++ program to create a class for a bank account with a constructor and a destructor.**

#include <iostream>

using namespace std;

class BankAccount {

private:

string accountNumber;

double balance;

public:

BankAccount(string accNum, double initialBalance) : accountNumber(accNum), balance(initialBalance) {

cout << "Bank Account created with account number: " << accountNumber << endl;

}

~BankAccount() {

cout << "Bank Account with account number: " << accountNumber << " is being destroyed." << endl;

}

};

int main() {

BankAccount myAccount("1234567890", 1000.0);

// Other operations with the bank account...

return 0;

}

1. **Write a c++ program to create a class for a car with a constructor and a destructor**

#include <iostream>

using namespace std;

class Car {

private:

string brand;

string model;

public:

Car(string b, string m) : brand(b), model(m) {

cout << "Car object created: " << brand << " " << model << endl;

}

~Car() {

cout << "Car object destroyed: " << brand << " " << model << endl;

}

};

int main() {

Car myCar("Toyota", "Camry");

// Other operations with the car...

return 0;

}

1. **Write a c++ program to create a class for a rectangle with a constructor and a destructor**

#include <iostream>

using namespace std;

class Rectangle {

private:

double length;

double width;

public:

Rectangle(double l, double w) : length(l), width(w) {

cout << "Rectangle object created with length: " << length << " and width: " << width << endl;

}

~Rectangle() {

cout << "Rectangle object destroyed." << endl;

}

};

int main() {

Rectangle myRectangle(5.0, 3.0);

// Other operations with the rectangle...

return 0;

}

1. **Write a c++ program to create a class for a book with a constructor and a destructor**

#include <iostream>

using namespace std;

class Book {

private:

string title;

string author;

public:

Book(string t, string a) : title(t), author(a) {

cout << "Book object created: " << title << " by " << author << endl;

}

~Book() {

cout << "Book object destroyed: " << title << " by " << author << endl;

}

};

int main() {

Book myBook("1984", "George Orwell");

// Other operations with the book...

return 0;

}

1. **Write a c++ program to create a class for student with a constructor and a destructor**

#include <iostream>

using namespace std;

class Student {

private:

string name;

int age;

public:

Student(string n, int a) : name(n), age(a) {

cout << "Student object created: " << name << " (Age: " << age << ")" << endl;

}

~Student() {

cout << "Student object destroyed: " << name << " (Age: " << age << ")" << endl;

}

};

int main() {

Student myStudent("John Doe", 20);

// Other operations with the student...

return 0;

}

**Operator overloading**

1. **Write a c++ program to overload the ++ operator to increment a variable**

#include <iostream>

using namespace std;

class Counter {

private:

int count;

public:

Counter() : count(0) {}

void operator++() {

count++;

}

void display() {

cout << "Count: " << count << endl;

}

};

int main() {

Counter c;

++c;

c.display(); // Output: Count: 1

return 0;

}

1. **Write a c++ program to overload the + operator to add two variables**

#include <iostream>

using namespace std;

class Number {

private:

int value;

public:

Number(int v) : value(v) {}

Number operator+(const Number& other) {

return Number(value + other.value);

}

void display() {

cout << "Value: " << value << endl;

}

};

int main() {

Number num1(5);

Number num2(10);

Number result = num1 + num2;

result.display(); // Output: Value: 15

return 0;

}

1. Write a c++ program to overload the << operator to print contents of a user defined class
2. Write a c++ program to overload the == operator to compare two objects of a user defined class
3. Write a c++ program to overload the \* operator to multiply two matrices
4. rite a c++ program to overload the [] operator to access the elements in an array using index values
5. Write a c++ program to overload the () to call a function with arguments
6. rite a c++ program to overload the – operator to subtract two variables
7. write a c++ program to overload a function to add two integer numbers and two floating point number separately
8. Write a c++ program to overload the += operator to add two objects of a user defined class
9. write a c++ program to overload a function to find the maximum value from two integer numbers and two floating point number, and two characters separately
10. write a c++ program to overload a function to concatenate two strings and two characters arrays separately
11. write a c++ program to overload a function to calculate the sum of two matrices and two arrays separately
12. write a c++ program to overload a function to print an integer array, a double array and a character array separately
13. write a c++ program to overload a function to find a factorial of an integer number and factorial of a floating-point number separately
14. write a c++ program to overload a function to sort an integer array and a double array
15. write a c++ program to overload a function to calculate the power of an integer number and power of a floating-point number separately
16. write a c++ program to overload a function to find an absolute value of an integer number and absolute value of a floating-point number separately

**Inheritance and pointers**

1. **Create a base class called Shape with data members for height and width. Derive two classes Rectangle and Triangle from the base class. Write member functions to calculate the area and perimeter of each class**

#include <iostream>

using namespace std;

class Shape {

protected:

double height;

double width;

public:

Shape(double h, double w) : height(h), width(w) {}

virtual double area() = 0;

virtual double perimeter() = 0;

};

class Rectangle : public Shape {

public:

Rectangle(double h, double w) : Shape(h, w) {}

double area() override {

return height \* width;

}

double perimeter() override {

return 2 \* (height + width);

}

};

class Triangle : public Shape {

private:

double side1;

double side2;

double side3;

public:

Triangle(double h, double w, double s1, double s2, double s3) : Shape(h, w), side1(s1), side2(s2), side3(s3) {}

double area() override {

// Using Heron's formula to calculate the area of a triangle

double s = (side1 + side2 + side3) / 2;

return sqrt(s \* (s - side1) \* (s - side2) \* (s - side3));

}

double perimeter() override {

return side1 + side2 + side3;

}

};

int main() {

Rectangle rect(5, 3);

Triangle tri(3, 4, 3, 4, 5);

cout << "Rectangle:" << endl;

cout << "Area: " << rect.area() << endl; // Output: 15

cout << "Perimeter: " << rect.perimeter() << endl; // Output: 16

cout << "Triangle:" << endl;

cout << "Area: " << tri.area() << endl; // Output: 6

cout << "Perimeter: " << tri.perimeter() << endl; // Output: 12

return 0;

}

1. Create a base class called vehicle with data members for make, model, and year. Derive two classes Car and Truck from the base class. The Car class should have additional data members for seating capacity and fuel type, while the Truck class should have additional data members for payload capacity and towing capacity. Write member functions to get and set the data members for each class

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class Vehicle {**

**protected:**

**string make;**

**string model;**

**int year;**

**public:**

**Vehicle(const string& mk, const string& mdl, int yr) : make(mk), model(mdl), year(yr) {}**

**void setMake(const string& mk) {**

**make = mk;**

**}**

**void setModel(const string& mdl) {**

**model = mdl;**

**}**

**void setYear(int yr) {**

**year = yr;**

**}**

**string getMake() const {**

**return make;**

**}**

**string getModel() const {**

**return model;**

**}**

**int getYear() const {**

**return year;**

**}**

**};**

**class Car : public Vehicle {**

**private:**

**int seatingCapacity;**

**string fuelType;**

**public:**

**Car(const string& mk, const string& mdl, int yr, int capacity, const string& fuel) : Vehicle(mk, mdl, yr), seatingCapacity(capacity), fuelType(fuel) {}**

**int getSeatingCapacity() const {**

**return seatingCapacity;**

**}**

**string getFuelType() const {**

**return fuelType;**

**}**

**};**

**class Truck : public Vehicle {**

**private:**

**double payloadCapacity;**

**double towingCapacity;**

**public:**

**Truck(const string& mk, const string& mdl, int yr, double payload, double towing) : Vehicle(mk, mdl, yr), payloadCapacity(payload), towingCapacity(towing) {}**

**double getPayloadCapacity() const {**

**return payloadCapacity;**

**}**

**double getTowingCapacity() const {**

**return towingCapacity;**

**}**

**};**

**int main() {**

**Car car("Toyota", "Camry", 2020, 5, "Petrol");**

**Truck truck("Ford", "F-150", 2019, 1500, 10000);**

**cout << "Car details:" << endl;**

**cout << "Make: " << car.getMake() << ", Model: " << car.getModel() << ", Year: " << car.getYear() << endl;**

**cout << "Seating Capacity: " << car.getSeatingCapacity() << ", Fuel Type: " << car.getFuelType() << endl;**

**cout << "Truck details:" << endl;**

**cout << "Make: " << truck.getMake() << ", Model: " << truck.getModel() << ", Year: " << truck.getYear() << endl;**

**cout << "Payload Capacity: " << truck.getPayloadCapacity() << " tons, Towing Capacity: " << truck.getTowingCapacity() << " lbs" << endl;**

**return 0;**

**}**

1. Create a base class called Animal with data members for name, species, and age. Derive two classes Cat and Dog from the base class. The Cat class should have additional data members for color and breed, while the Dog class should have additional data members for weight and breed. Write member functions to get and set the data members for each class
2. Create a base class called Employee with data members for name, d, and salary Derive two classes Manager and Engineer from the base class. The Manager class should have additional data members for department and bonus, while the Engineer class should have additional data members for specialty and hours. Write member functions to get and set the data members for each class
3. Create a base class called Person with data members for name, age, and gender. Derive two classes Student and Teacher from the base class. The Student class should have additional data members for roll number and class, while the Teacher class should have additional data members for subject and salary. Write member functions to get and set the data members for each class.
4. **Write a C++ program to create a pointer to an integer and display its value.**

#include <iostream>

using namespace std;

int main() {

int num = 10;

int \*ptr = &num;

cout << "Value of integer pointed by the pointer: " << \*ptr << endl;

return 0;

}

1. **Write a C++ program to create a pointer to a float and display its value**.

#include <iostream>

using namespace std;

int main() {

float num = 3.14;

float \*ptr = &num;

cout << "Value of float pointed by the pointer: " << \*ptr << endl;

return 0;

}

1. **Write a C++ program to create a pointer to a char and display its value**.

#include <iostream>

using namespace std;

int main() {

char ch = 'A';

char \*ptr = &ch;

cout << "Value of char pointed by the pointer: " << \*ptr << endl;

return 0;

}

1. **Write a C++ program to create a pointer to a double and display its value.**

#include <iostream>

using namespace std;

int main() {

double num = 3.14159;

double \*ptr = &num;

cout << "Value of double pointed by the pointer: " << \*ptr << endl;

return 0;

}

1. **Write a C++ program to create a pointer to a string and display its value.**

#include <iostream>

#include <string>

using namespace std;

int main() {

string text = "Hello, World!";

string \*ptr = &text;

cout << "Value of string pointed by the pointer: " << \*ptr << endl;

return 0;

}

1. **Write a C++ program to create a pointer to an array of elements and display its value.**

#include <iostream>

using namespace std;

int main() {

int arr[] = {1, 2, 3, 4, 5};

int \*ptr = arr;

cout << "Elements of the array pointed by the pointer:" << endl;

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

cout << \*(ptr + i) << " ";

}

cout << endl;

return 0;

}

1. **Write a C++ program to create a pointer to an array of character and display its value.**

#include <iostream>

using namespace std;

int main() {

char arr[] = {'H', 'e', 'l', 'l', 'o'};

char \*ptr = arr;

cout << "Elements of the character array pointed by the pointer:" << endl;

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

cout << \*(ptr + i);

}

cout << endl;

return 0;

}

1. **Write a C++ program to create a pointer to an array of floats and display its value.**

#include <iostream>

using namespace std;

int main() {

float arr[] = {1.1, 2.2, 3.3, 4.4, 5.5};

float \*ptr = arr;

cout << "Elements of the float array pointed by the pointer:" << endl;

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

cout << \*(ptr + i) << " ";

}

cout << endl;

return 0;

}

1. **Write a C++ program to create a pointer to an object and display its attributes.**

#include <iostream>

#include <string>

using namespace std;

class MyClass {

public:

int value;

MyClass(int val) : value(val) {}

};

int main() {

MyClass obj(42);

MyClass \*ptr = &obj;

cout << "Value of the object attribute pointed by the pointer: " << ptr->value << endl;

return 0;

}

1. **Write a C++ program to create a pointer to a function and call the function using the pointer.**

#include <iostream>

using namespace std;

void myFunction() {

cout << "This is myFunction." << endl;

}

int main() {

void (\*ptr)() = &myFunction;

cout << "Calling function using pointer: ";

(\*ptr)(); // Calling the function using the pointer

return 0;

}

**Polymorphism**

1. **Create a base class called Person with a virtual function work (). Derive two classes Employee and Manager from the base class. Implement the work () function for each class**

#include <iostream>

using namespace std;

class Person {

public:

virtual void work() {

cout << "Person is working" << endl;

}

};

class Employee : public Person {

public:

void work() override {

cout << "Employee is working" << endl;

}

};

class Manager : public Person {

public:

void work() override {

cout << "Manager is working" << endl;

}

};

int main() {

Employee emp;

Manager mgr;

emp.work();

mgr.work();

return 0;

}

1. **Create a base class called Animal with a virtual function eat (). Derive two classes Herbivore and Carnivore from the base class. Implement the eat function for each class.**

#include <iostream>

using namespace std;

class Animal {

public:

virtual void eat() {

cout << "Animal is eating" << endl;

}

};

class Herbivore : public Animal {

public:

void eat() override {

cout << "Herbivore is eating plants" << endl;

}

};

class Carnivore : public Animal {

public:

void eat() override {

cout << "Carnivore is eating meat" << endl;

}

};

int main() {

Herbivore herb;

Carnivore carn;

herb.eat();

carn.eat();

return 0;

}

1. **Create a base class called Shape with virtual functions area () and volume (). Derive two classes Sphere and Cylinder from the base class. Implement the area and volume () functions for each class**

#include <iostream>

using namespace std;

class Shape {

public:

virtual double area() {

cout << "Area of the shape" << endl;

return 0.0;

}

virtual double volume() {

cout << "Volume of the shape" << endl;

return 0.0;

}

};

class Sphere : public Shape {

public:

double area() override {

cout << "Surface area of the sphere" << endl;

return 0.0;

}

double volume() override {

cout << "Volume of the sphere" << endl;

return 0.0;

}

};

class Cylinder : public Shape {

public:

double area() override {

cout << "Surface area of the cylinder" << endl;

return 0.0;

}

double volume() override {

cout << "Volume of the cylinder" << endl;

return 0.0;

}

};

int main() {

Sphere sphere;

Cylinder cylinder;

sphere.area();

sphere.volume();

cylinder.area();

cylinder.volume();

return 0;

}

1. **Create a base class called Person with a virtual function greet). Derive two classes Student and Teacher from the base class. implement the greet) function for each class**

#include <iostream>

using namespace std;

class Person {

public:

virtual void greet() {

cout << "Hello, I am a person." << endl;

}

};

class Student : public Person {

public:

void greet() override {

cout << "Hello, I am a student." << endl;

}

};

class Teacher : public Person {

public:

void greet() override {

cout << "Hello, I am a teacher." << endl;

}

};

int main() {

Student student;

Teacher teacher;

student.greet(); // Output: Hello, I am a student.

teacher.greet(); // Output: Hello, I am a teacher.

return 0;

}

1. **Create a base class called Shape with virtual functions area( ) and perimeter(). Derive two classes Rectangle and Triangle from the base class. Implement the area () and perimeter () functions for each class.**

#include <iostream>

using namespace std;

class Shape {

public:

virtual double area() {

cout << "Area of the shape" << endl;

return 0.0;

}

virtual double perimeter() {

cout << "Perimeter of the shape" << endl;

return 0.0;

}

};

class Rectangle : public Shape {

private:

double length;

double width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double area() override {

return length \* width;

}

double perimeter() override {

return 2 \* (length + width);

}

};

class Triangle : public Shape {

private:

double side1;

double side2;

double side3;

public:

Triangle(double s1, double s2, double s3) : side1(s1), side2(s2), side3(s3) {}

double area() override {

// Using Heron's formula to calculate the area of a triangle

double s = (side1 + side2 + side3) / 2;

return sqrt(s \* (s - side1) \* (s - side2) \* (s - side3));

}

double perimeter() override {

return side1 + side2 + side3;

}

};

int main() {

Rectangle rect(5, 3);

Triangle tri(3, 4, 5);

cout << "Rectangle:" << endl;

cout << "Area: " << rect.area() << endl; // Output: 15

cout << "Perimeter: " << rect.perimeter() << endl; // Output: 16

cout << "Triangle:" << endl;

cout << "Area: " << tri.area() << endl; // Output: 6

cout << "Perimeter: " << tri.perimeter() << endl; // Output: 12

return 0;

}

1. **Create a base class called Vehicle with a virtual function drive(). Derive two classes Car and Truck from the base class. Implement the drive() function for each class.**

#include <iostream>

using namespace std;

class Vehicle {

public:

virtual void drive() {

cout << "Vehicle is being driven." << endl;

}

};

class Car : public Vehicle {

public:

void drive() override {

cout << "Car is being driven." << endl;

}

};

class Truck : public Vehicle {

public:

void drive() override {

cout << "Truck is being driven." << endl;

}

};

int main() {

Car car;

Truck truck;

car.drive(); // Output: Car is being driven.

truck.drive(); // Output: Truck is being driven.

return 0;

}

1. **Create a base class called Employee with a virtual function calculate Pay(). Derive two classes Manager and Engineer from the base class. Implement the calculatePay () function for each class.**

#include <iostream>

using namespace std;

class Employee {

public:

virtual double calculatePay() {

return 0.0;

}

};

class Manager : public Employee {

public:

double calculatePay() override {

return 1000.0; // Sample pay for manager

}

};

class Engineer : public Employee {

public:

double calculatePay() override {

return 1500.0; // Sample pay for engineer

}

};

int main() {

Manager manager;

Engineer engineer;

cout << "Manager's pay: $" << manager.calculatePay() << endl; // Output: Manager's pay: $1000

cout << "Engineer's pay: $" << engineer.calculatePay() << endl; // Output: Engineer's pay: $1500

return 0;

}

1. **Create a base class called Animal with a virtual function speak(). Derive two classes Cat and Dog from the base class. Implement the speak() function for each class.**

#include <iostream>

using namespace std;

class Animal {

public:

virtual void speak() {

cout << "Animal makes a sound." << endl;

}

};

class Cat : public Animal {

public:

void speak() override {

cout << "Cat meows." << endl;

}

};

class Dog : public Animal {

public:

void speak() override {

cout << "Dog barks." << endl;

}

};

int main() {

Cat cat;

Dog dog;

cat.speak(); // Output: Cat meows.

dog.speak(); // Output: Dog barks.

return 0;

}

1. **Create a base class called Shape with a virtual function area(). Derive two classes Rectangle and Circle from the base class. Implement the area() function for each class.**

#include <iostream>

using namespace std;

class Shape {

public:

virtual double area() {

cout << "Area of the shape" << endl;

return 0.0;

}

};

class Rectangle : public Shape {

private:

double length;

double width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double area() override {

return length \* width;

}

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double area() override {

return 3.14 \* radius \* radius;

}

};

int main() {

Rectangle rect(5, 3);

Circle circle(4);

cout << "Rectangle area: " << rect.area() << endl; // Output: Rectangle area: 15

cout << "Circle area: " << circle.area() << endl; // Output: Circle area: 50.24

return 0;

}

**Exception Handling**

1. **Write a c++ program to demonstrate to use of the finally block for handling exceptions**

#include <iostream>

#include <stdexcept>

using namespace std;

void someFunction() {

try {

throw runtime\_error("An error occurred");

} catch (const exception& e) {

cout << "Caught exception: " << e.what() << endl;

}

cout << "Finally block executed regardless of exception" << endl;

}

int main() {

someFunction();

return 0;

}

1. **Write a c++ program to demonstrate to use of nested try-catch blocks for handling exceptions**

#include <iostream>

#include<stdexcept>

using namespace std;

int main() {

try {

try {

throw runtime\_error("Inner exception");

} catch (const exception& e) {

cout << "Inner catch block: " << e.what() << endl;

throw;

}

} catch (const exception& e) {

cout << "Outer catch block: " << e.what() << endl;

}

return 0;

}

1. **Write a c++ program to demonstrate to use of user-defined exception for handling custom exception**

#include <iostream>

#include <stdexcept>

using namespace std;

class CustomException : public exception {

public:

const char\* what() const noexcept override {

return "Custom exception occurred";

}

};

int main() {

try {

throw CustomException();

} catch (const exception& e) {

cout << "Caught exception: " << e.what() << endl;

}

return 0;

}

1. **Write a c++ program to demonstrate to use of the standard class for handling exceptions**

#include <iostream>

#include <stdexcept>

using namespace std;

int main() {

try {

throw runtime\_error("Standard exception occurred");

} catch (const exception& e) {

cout << "Caught exception: " << e.what() << endl;

}

return 0;

}

1. **Write a c++ program to demonstrate to use of the keyword to throw an exception**

#include <iostream>

#include <stdexcept>

using namespace std;

int main() {

try {

int x = 10;

if (x > 5) {

throw runtime\_error("x is too large");

}

} catch (const exception& e) {

cout << "Caught exception: " << e.what() << endl;

}

return 0;

}

1. **Write a c++ program to demonstrate to use of multiple catch blocks for handling different types of exceptions**

#include <iostream>

#include <stdexcept>

using namespace std;

int main() {

try {

// Code that may throw exceptions

throw runtime\_error("An error occurred");

} catch (const runtime\_error& e) {

cout << "Caught runtime error: " << e.what() << endl;

} catch (const exception& e) {

cout << "Caught exception: " << e.what() << endl;

}

return 0;

}

1. **Write a c++ program to demonstrate to use of try-catch blocks for handling exceptions**

#include <iostream>

#include <stdexcept>

using namespace std;

int main() {

try {

// Code that may throw an exception

throw runtime\_error("An error occurred");

} catch (const exception& e) {

cout << "Caught exception: " << e.what() << endl;

}

return 0;

}