**MODULE: 4.1 (C++ Basic)**

• WAP to print “Hello World” using C++

**#include<iostream>**

**using namespace std;**

**int main()**

**{**

**cout<<"hello world";**

**return 0;**

**}**

• What is OOP? List OOP concepts

**OOPs Concepts:**

* Class
* Objects
* Data Abstraction
* Encapsulation
* Inheritance
* Polymorphism
* Dynamic Binding

**Class:**

A class is a user-defined data type. It consists of data members and member functions, which can be accessed and used by creating an instance of that class. It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.

**Object:**

It is a basic unit of Object-Oriented Programming and represents the real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. An object has an identity, state, and behavior. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other’s data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

**Data Abstraction:**

Data abstraction is one of the most essential and important features of object-oriented programming. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation. Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

**Encapsulation:**

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In Encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.

**Inheritance:**

Inheritance is an important pillar of OOP(Object-Oriented Programming). The capability of a class to derive properties and characteristics from another class is called Inheritance. When we write a class, we inherit properties from other classes. So when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class that possesses it. Inheritance allows the user to reuse the code whenever possible and reduce its redundancy.

**Polymorphism:**

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. For example, A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism.

**Dynamic Binding:**

In dynamic binding, the code to be executed in response to the function call is decided at runtime. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time. Dynamic Method Binding One of the main advantages of inheritance is that some derived class D has all the members of its base class B. Once D is not hiding any of the public members of B, then an object of D can represent B in any context where a B could be used. This feature is known as subtype polymorphism.

• What is the difference between OOP and POP?

|  |  |
| --- | --- |
| **oop** | **pop** |
| [Object oriented](https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/). | [Structure oriented](https://www.geeksforgeeks.org/difference-between-structured-programming-and-object-oriented-programming/). |
| Program is divided into objects | Program is divided into functions. |
| Bottom-up approach. | Top-down approach. |
| Inheritance property is used. | Inheritance is not allowed. |
| It uses access specifier | It doesn’t use access specifier. |
| Encapsulation is used to hide the data. | No data hiding. |

**MODULE: 4.2 (Programing with C++)**

• WAP to create simple calculator using class

**#include <iostream>**

**using namespace std;**

**class Calculator {**

**public:**

**double add(double x, double y) {**

**return x + y;**

**}**

**double subtract(double x, double y) {**

**return x - y;**

**}**

**double multiply(double x, double y) {**

**return x \* y;**

**}**

**double divide(double x, double y) {**

**if (y == 0) {**

**cout << "Cannot divide by zero." << endl;**

**return 0;**

**}**

**return x / y;**

**}**

**};**

**int main() {**

**Calculator calculator;**

**char operation;**

**while (true) {**

**cout << "Options:" << endl;**

**cout << "Enter '+' for addition" << endl;**

**cout << "Enter '-' for subtraction" << endl;**

**cout << "Enter '\*' for multiplication" << endl;**

**cout << "Enter '/' for division" << endl;**

**cout << "Enter 'q' to quit" << endl;**

**cout << "Enter an operation: ";**

**cin >> operation;**

**if (operation == 'q') {**

**break;**

**}**

**double num1, num2;**

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

**cin >> num1;**

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

**cin >> num2;**

**switch (operation) {**

**case '+':**

**cout << "Result: " << calculator.add(num1, num2) << endl;**

**break;**

**case '-':**

**cout << "Result: " << calculator.subtract(num1, num2) << endl;**

**break;**

**case '\*':**

**cout << "Result: " << calculator.multiply(num1, num2) << endl;**

**break;**

**case '/':**

**cout << "Result: " << calculator.divide(num1, num2) << endl;**

**break;**

**default:**

**cout << "Invalid operation. Please enter a valid option." << endl;**

**}**

**}**

**return 0;**

**}**

• Define a class to represent a bank account. Include the following members:

1. Data Member: -Name of the depositor -Account Number -Type of Account -Balance amount in the account

2. Member Functions -To assign values -To deposited an amount -To withdraw an amount after checking balance -To display name and balance

#include <iostream>

#include <string>

using namespace std;

class BankAccount {

private:

string depositorName;

string accountNumber;

string accountType;

double balance;

public:

// Constructor to initialize the account

BankAccount(string name, string accNum, string accType, double initialBalance) {

depositorName = name;

accountNumber = accNum;

accountType = accType;

balance = initialBalance;

}

// Member function to deposit money into the account

void deposit(double amount) {

if (amount > 0) {

balance += amount;

cout << "Deposit of $" << amount << " successful." << endl;

} else {

cout << "Invalid deposit amount." << endl;

}

}

// Member function to withdraw money from the account after checking the balance

void withdraw(double amount) {

if (amount > 0 && amount <= balance) {

balance -= amount;

cout << "Withdrawal of $" << amount << " successful." << endl;

} else {

cout << "Invalid withdrawal amount or insufficient balance." << endl;

}

}

// Member function to display the name and balance

void displayAccountInfo() {

cout << "Depositor Name: " << depositorName << endl;

cout << "Balance: $" << balance << endl;

}

};

int main() {

// Create an instance of the BankAccount class

BankAccount myAccount("John Doe", "123456789", "Savings", 1000.0);

// Deposit and withdraw money

myAccount.deposit(500.0);

myAccount.withdraw(200.0);

// Display name and balance

myAccount.displayAccountInfo();

return 0;

}

• Write a program to find the multiplication values and the cubic values using inline function

#include <iostream>

using namespace std;

// Inline function to calculate the multiplication of a number by itself

inline double multiply(double num) {

return num \* num;

}

// Inline function to calculate the cubic value of a number

inline double cubic(double num) {

return num \* num \* num;

}

int main() {

double number;

cout << "Enter a number: ";

cin >> number;

// Calculate and display the multiplication value

cout << "Multiplication value: " << multiply(number) << endl;

// Calculate and display the cubic value

cout << "Cubic value: " << cubic(number) << endl;

return 0;

}

• Write a program of Addition, Subtraction, Division, Multiplication using constructor.

#include <iostream>

using namespace std;

class Calculator {

private:

double num1;

double num2;

public:

Calculator(double x, double y) {

num1 = x;

num2 = y;

}

double add() {

return num1 + num2;

}

double subtract() {

return num1 - num2;

}

double multiply() {

return num1 \* num2;

}

double divide() {

if (num2 != 0) {

return num1 / num2;

} else {

cout << "Error: Division by zero" << endl;

return 0;

}

}

};

int main() {

double num1, num2;

cout << "Enter the first number: ";

cin >> num1;

cout << "Enter the second number: ";

cin >> num2;

Calculator calculator(num1, num2);

cout << "Addition: " << calculator.add() << endl;

cout << "Subtraction: " << calculator.subtract() << endl;

cout << "Multiplication: " << calculator.multiply() << endl;

cout << "Division: " << calculator.divide() << endl;

return 0;

}

• Write a program of Addition, Subtraction, Division, Multiplication using constructor.

#include <iostream>

using namespace std;

class Calculator {

private:

double num1;

double num2;

public:

// Constructor to initialize the numbers

Calculator(double x, double y) {

num1 = x;

num2 = y;

}

double add() {

return num1 + num2;

}

double subtract() {

return num1 - num2;

}

double multiply() {

return num1 \* num2;

}

double divide() {

if (num2 != 0) {

return num1 / num2;

} else {

cout << "Error: Division by zero" << endl;

return 0;

}

}

};

int main() {

double num1, num2;

cout << "Enter the first number: ";

cin >> num1;

cout << "Enter the second number: ";

cin >> num2;

Calculator calculator(num1, num2);

cout << "Addition: " << calculator.add() << endl;

cout << "Subtraction: " << calculator.subtract() << endl;

cout << "Multiplication: " << calculator.multiply() << endl;

cout << "Division: " << calculator.divide() << endl;

return 0;

}

• Assume a class cricketer is declared. Declare a derived class batsman from cricketer. Data member of batsman. Total runs, Average runs and best performance. Member functions input data, calculate average runs, Display data. (Single Inheritance)

#include <iostream>

#include <string>

using namespace std;

class Cricketer {

protected:

string name;

int age;

public:

void inputData() {

cout << "Enter Cricketer's Name: ";

cin >> name;

cout << "Enter Cricketer's Age: ";

cin >> age;

}

void displayData() {

cout << "Cricketer's Name: " << name << endl;

cout << "Cricketer's Age: " << age << " years" << endl;

}

};

class Batsman : public Cricketer {

private:

int totalRuns;

double averageRuns;

int bestPerformance;

public:

void inputBatsmanData() {

inputData(); // Inherited function to input name and age

cout << "Enter Total Runs: ";

cin >> totalRuns;

cout << "Enter Best Performance: ";

cin >> bestPerformance;

}

void calculateAverageRuns() {

if (totalRuns > 0) {

averageRuns = static\_cast<double>(totalRuns) / 10; // Assuming 10 matches

} else {

averageRuns = 0;

}

}

void displayBatsmanData() {

displayData(); // Inherited function to display name and age

cout << "Total Runs: " << totalRuns << " runs" << endl;

cout << "Average Runs: " << averageRuns << " runs per match" << endl;

cout << "Best Performance: " << bestPerformance << " runs" << endl;

}

};

int main() {

Batsman batsman;

cout << "Enter Batsman's Information:" << endl;

batsman.inputBatsmanData();

batsman.calculateAverageRuns();

cout << "\nBatsman's Data:" << endl;

batsman.displayBatsmanData();

return 0;

}

• Create a class person having members name and age. Derive a class student having member percentage. Derive another class teacher having member salary. Write necessary member function to initialize, read and write data. Write also Main function (Multiple Inheritance)

#include <iostream>

#include <string>

using namespace std;

class Person {

protected:

string name;

int age;

public:

void initializePerson() {

cout << "Enter Person's Name: ";

cin >> name;

cout << "Enter Person's Age: ";

cin >> age;

}

void displayPerson() {

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

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

}

};

class Student {

protected:

float percentage;

public:

void initializeStudent() {

initializePerson(); // Inherited function to initialize name and age

cout << "Enter Student's Percentage: ";

cin >> percentage;

}

void displayStudent() {

displayPerson(); // Inherited function to display name and age

cout << "Percentage: " << percentage << "%" << endl;

}

};

class Teacher {

protected:

float salary;

public:

void initializeTeacher() {

initializePerson(); // Inherited function to initialize name and age

cout << "Enter Teacher's Salary: $";

cin >> salary;

}

void displayTeacher() {

displayPerson(); // Inherited function to display name and age

cout << "Salary: $" << salary << endl;

}

};

int main() {

Student student;

Teacher teacher;

cout << "Enter Student's Information:" << endl;

student.initializeStudent();

cout << "\nStudent's Data:" << endl;

student.displayStudent();

cout << "\nEnter Teacher's Information:" << endl;

teacher.initializeTeacher();

cout << "\nTeacher's Data:" << endl;

teacher.displayTeacher();

return 0;

}

• Assume that the test results of a batch of students are stored in three different classes. Class Students are storing the roll number. Class Test stores the marks obtained in two subjects and class result contains the total marks obtained in the test. The class result can inherit the details of the marks obtained in the test and roll number of students. (Multilevel Inheritance)

#include <iostream>

using namespace std;

class Students {

protected:

int rollNumber;

public:

void inputStudentDetails() {

cout << "Enter Student's Roll Number: ";

cin >> rollNumber;

}

void displayStudentDetails() {

cout << "Student's Roll Number: " << rollNumber << endl;

}

};

class Test : public Students {

protected:

int subject1Marks;

int subject2Marks;

public:

void inputTestMarks() {

inputStudentDetails(); // Inherited function to input roll number

cout << "Enter Marks for Subject 1: ";

cin >> subject1Marks;

cout << "Enter Marks for Subject 2: ";

cin >> subject2Marks;

}

void displayTestMarks() {

displayStudentDetails(); // Inherited function to display roll number

cout << "Marks for Subject 1: " << subject1Marks << endl;

cout << "Marks for Subject 2: " << subject2Marks << endl;

}

};

class Result : public Test {

private:

int totalMarks;

public:

void calculateTotalMarks() {

totalMarks = subject1Marks + subject2Marks;

}

void displayResult() {

displayTestMarks(); // Inherited function to display roll number and test marks

calculateTotalMarks();

cout << "Total Marks Obtained: " << totalMarks << " out of 200" << endl;

}

};

int main() {

Result result;

cout << "Enter Student's Test Details:" << endl;

result.inputTestMarks();

cout << "\nStudent's Test Result:" << endl;

result.displayResult();

return 0;

}

• Write a program to Mathematic operation like Addition, Subtraction, Multiplication, Division Of two number using different parameters and Function Overloading

#include <iostream>

using namespace std;

// Function to perform addition

int mathOperation(int a, int b) {

return a + b;

}

double mathOperation(double a, double b) {

return a + b;

}

// Function to perform subtraction

int mathOperation(int a, int b, int c) {

return a - b - c;

}

double mathOperation(double a, double b, double c) {

return a - b - c;

}

// Function to perform multiplication

int mathOperation(int a, int b, int c, int d) {

return a \* b \* c \* d;

}

double mathOperation(double a, double b, double c, double d) {

return a \* b \* c \* d;

}

// Function to perform division

int mathOperation(int a, int b, int c, int d, int e) {

if (b == 0 || c == 0 || d == 0 || e == 0) {

cout << "Error: Division by zero" << endl;

return 0;

}

return a / (b \* c \* d \* e);

}

double mathOperation(double a, double b, double c, double d, double e) {

if (b == 0.0 || c == 0.0 || d == 0.0 || e == 0.0) {

cout << "Error: Division by zero" << endl;

return 0.0;

}

return a / (b \* c \* d \* e);

}

int main() {

// Using integer parameters

cout << "Integer Results:" << endl;

cout << "Addition: " << mathOperation(5, 3) << endl;

cout << "Subtraction: " << mathOperation(10, 4, 2) << endl;

cout << "Multiplication: " << mathOperation(2, 3, 4, 2) << endl;

cout << "Division: " << mathOperation(20, 2, 2, 5, 2) << endl;

// Using double parameters

cout << "\nDouble Results:" << endl;

cout << "Addition: " << mathOperation(5.5, 3.7) << endl;

cout << "Subtraction: " << mathOperation(10.0, 4.2, 2.1) << endl;

cout << "Multiplication: " << mathOperation(2.0, 3.5, 4.0, 2.5) << endl;

cout << "Division: " << mathOperation(20.0, 2.5, 2.0, 5.0, 2.5) << endl;

return 0;

}

• Write a Program of Two 1D Matrix Addition using Operator Overloading

#include <iostream>

#include <vector>

using namespace std;

class Matrix1D {

private:

vector<int> data;

public:

// Constructor to initialize the matrix

Matrix1D(int size) {

data.resize(size);

}

// Overload the + operator for matrix addition

Matrix1D operator+(const Matrix1D& other) {

if (data.size() != other.data.size()) {

cerr << "Matrix sizes are not equal. Addition not possible." << endl;

return Matrix1D(0);

}

Matrix1D result(data.size());

for (size\_t i = 0; i < data.size(); i++) {

result.data[i] = data[i] + other.data[i];

}

return result;

}

// Display the matrix

void display() {

for (int num : data) {

cout << num << " ";

}

cout << endl;

}

};

int main() {

int size;

cout << "Enter the size of the matrices: ";

cin >> size;

Matrix1D matrix1(size);

Matrix1D matrix2(size);

cout << "Enter the elements of the first matrix:" << endl;

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

cin >> matrix1.data[i];

}

cout << "Enter the elements of the second matrix:" << endl;

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

cin >> matrix2.data[i];

}

Matrix1D result = matrix1 + matrix2;

cout << "Result of matrix addition:" << endl;

result.display();

return 0;

}

• Write a program to concatenate the two strings using Operator Overloading

#include <iostream>

#include <cstring>

using namespace std;

class StringConcatenator {

private:

char\* str;

public:

StringConcatenator(const char\* initialStr = "") {

str = new char[strlen(initialStr) + 1];

strcpy(str, initialStr);

}

StringConcatenator operator+(const StringConcatenator& other) {

char\* resultStr = new char[strlen(str) + strlen(other.str) + 1];

strcpy(resultStr, str);

strcat(resultStr, other.str);

return StringConcatenator(resultStr);

}

void display() {

cout << str << endl;

}

~StringConcatenator() {

delete[] str;

}

};

int main() {

StringConcatenator str1("Hello, ");

StringConcatenator str2("world!");

StringConcatenator result = str1 + str2;

cout << "Concatenated String: ";

result.display();

return 0;

}

• Write a program to calculate the area of circle, rectangle and triangle using Function Overloading ¬ Rectangle: Area \* breadth ¬

Triangle: ½ \*Area\* breadth ¬

Circle: Pi \* Area \*Area

#include <iostream>

#include <cmath>

using namespace std;

const double PI = 3.14159265359; // Value of Pi

// Function to calculate the area of a rectangle

double calculateArea(double length, double breadth) {

return length \* breadth;

}

// Function to calculate the area of a triangle

double calculateArea(double base, double height, char shape = 'T') {

if (shape == 'T') {

return 0.5 \* base \* height; // Triangle

} else {

return -1.0; // Invalid shape

}

}

// Function to calculate the area of a circle

double calculateArea(double radius, char shape = 'C') {

if (shape == 'C') {

return PI \* radius \* radius; // Circle

} else {

return -1.0; // Invalid shape

}

}

int main() {

char shape;

cout << "Enter the shape (R for Rectangle, T for Triangle, C for Circle): ";

cin >> shape;

if (shape == 'R') {

double length, breadth;

cout << "Enter length and breadth of the rectangle: ";

cin >> length >> breadth;

cout << "Area of the rectangle: " << calculateArea(length, breadth) << endl;

} else if (shape == 'T') {

double base, height;

cout << "Enter base and height of the triangle: ";

cin >> base >> height;

cout << "Area of the triangle: " << calculateArea(base, height) << endl;

} else if (shape == 'C') {

double radius;

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

cin >> radius;

cout << "Area of the circle: " << calculateArea(radius) << endl;

} else {

cout << "Invalid shape entered." << endl;

return 0;

}

• Write a program to swap the two numbers using friend function without using third variable

#include <iostream>

using namespace std;

class NumberSwapper; // Forward declaration

class Number {

private:

int value;

public:

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

friend void swapNumbers(Number& num1, Number& num2);

};

void swapNumbers(Number& num1, Number& num2) {

num1.value = num1.value + num2.value;

num2.value = num1.value - num2.value;

num1.value = num1.value - num2.value;

}

int main() {

int num1, num2;

cout << "Enter the first number: ";

cin >> num1;

cout << "Enter the second number: ";

cin >> num2;

Number n1(num1);

Number n2(num2);

cout << "Before swapping: " << n1.value << " " << n2.value << endl;

swapNumbers(n1, n2);

cout << "After swapping: " << n1.value << " " << n2.value << endl;

return 0;

}

• Write a program to find the max number from given two numbers using friend function

#include <iostream>

using namespace std;

class NumberSwapper; // Forward declaration

class Number {

private:

int value;

public:

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

friend void swapNumbers(Number& num1, Number& num2);

};

void swapNumbers(Number& num1, Number& num2) {

num1.value = num1.value + num2.value;

num2.value = num1.value - num2.value;

num1.value = num1.value - num2.value;

}

int main() {

int num1, num2;

cout << "Enter the first number: ";

cin >> num1;

cout << "Enter the second number: ";

cin >> num2;

Number n1(num1);

Number n2(num2);

cout << "Before swapping: " << n1.value << " " << n2.value << endl;

swapNumbers(n1, n2);

cout << "After swapping: " << n1.value << " " << n2.value << endl;

return 0;

}

**MODULE: 4.2 (C, C++ Templates)**

• Write a program of to swap the two values using templates

#include <iostream>

using namespace std;

class NumberSwapper; // Forward declaration

class Number {

private:

int value;

public:

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

friend void swapNumbers(Number& num1, Number& num2);

};

void swapNumbers(Number& num1, Number& num2) {

num1.value = num1.value + num2.value;

num2.value = num1.value - num2.value;

num1.value = num1.value - num2.value;

}

int main() {

int num1, num2;

cout << "Enter the first number: ";

cin >> num1;

cout << "Enter the second number: ";

cin >> num2;

Number n1(num1);

Number n2(num2);

cout << "Before swapping: " << n1.value << " " << n2.value << endl;

swapNumbers(n1, n2);

cout << "After swapping: " << n1.value << " " << n2.value << endl;

return 0;

}

• Write a program of to sort the array using templates.

#include <iostream>

using namespace std;

class NumberSwapper; // Forward declaration

class Number {

private:

int value;

public:

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

friend void swapNumbers(Number& num1, Number& num2);

};

void swapNumbers(Number& num1, Number& num2) {

num1.value = num1.value + num2.value;

num2.value = num1.value - num2.value;

num1.value = num1.value - num2.value;

}

int main() {

int num1, num2;

cout << "Enter the first number: ";

cin >> num1;

cout << "Enter the second number: ";

cin >> num2;

Number n1(num1);

Number n2(num2);

cout << "Before swapping: " << n1.value << " " << n2.value << endl;

swapNumbers(n1, n2);

cout << "After swapping: " << n1.value << " " << n2.value << endl;

return 0;

}