1.Introduction to C++

1.First C++ Program: Hello World

o Write a simple C++ program to display "Hello, World!".

o Objective: Understand the basic structure of a C++ program, including #include, main(), and cout.

#include <iostream> // Required for cout

Using namespace std;

int main() {

cout << "Hello, World!" << std::endl; // Output statement

return 0; // Exit status

}

2.Basic Input/Output

o Write a C++ program that accepts user input for their name and age and then displays a personalized greeting.

o Objective: Practice input/output operations using cin and cout.

#include <iostream>

#include <string>

using namespace std;

int main() {

string name;

int age;

// Ask for user's name

cout << "Enter your name: ";

getline(cin, name); // Read full name with spaces

// Ask for user's age

cout << "Enter your age: ";

cin >> age;

// Display a personalized greeting

cout << "Hello, " << name << "! You are " << age << " years old." << endl;

return 0;

}

3.POP vs. OOP Comparison Program

o Write two small programs: one using Procedural Programming (POP) to calculate the area of a rectangle, and another using Object-Oriented Programming (OOP) with a class and object for the same task.

o Objective: Highlight the difference between POP and OOP approaches.

1.pop

#include <iostream>

using namespace std;

int main() {

float length, width, area;

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

cin >> length;

cout << "Enter width of the rectangle: ";

cin >> width;

area = length \* width;

cout << "Area of the rectangle is: " << area << endl;

return 0;

}

2.oop

#include <iostream>

using namespace std;

class Rectangle {

private:

float length;

float width;

public:

void getData() {

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

cin >> length;

cout << "Enter width of the rectangle: ";

cin >> width;

}

float calculateArea() {

return length \* width;

}

void displayArea() {

cout << "Area of the rectangle is: " << calculateArea() << endl;

}

};

int main() {

Rectangle rect; // Create object

rect.getData(); // Input

rect.displayArea(); // Output

return 0;

}

|  |  |  |
| --- | --- | --- |
| **Feature** | **POP Version** | **OOP Version** |
| Structure | Functions & variables only | Class with data + functions (methods) |
| Data Access | Global/local variables | Private data, accessed via methods |
| Reusability | Low | High (via classes and objects) |
| Modularity | Less modular | Highly modular |

4. Setting Up Development Environment

o Write a program that asks for two numbers and displays their sum. Ensure this is done after setting up the IDE (like Dev C++ or CodeBlocks).

o Objective: Help students understand how to install, configure, and run programs inan IDE.

#include <iostream>

using namespace std;

int main() {

int num1, num2, sum;

cout << "Enter first number: ";

cin >> num1;

cout << "Enter second number: ";

cin >> num2;

sum = num1 + num2;

cout << "The sum of " << num1 << " and " << num2 << " is: " << sum << endl;

return 0;

}

**Dev C++ Setup:**

1. Download Dev C++ from: https://sourceforge.net/projects/orwelldevcpp/
2. Install it and open the program.
3. Click on File > New > Source File, paste the code above.
4. Save the file with .cpp extension (e.g., sum.cpp).
5. Click Execute > Compile and Run to see the result.

2. Variables, Data Types, and Operators

1. Variables and Constants

o Write a C++ program that demonstrates the use of variables and constants. Create variables of different data types and perform operations on them.

o Objective: Understand the difference between variables and constants.

Here's a simple C++ program that demonstrates the use of **variables** and **constants**, and performs basic operations on different data types:

#include <iostream>

using namespace std;

int main() {

// Declare and initialize constants

const float PI = 3.14159; // constant float value

const int DAYS\_IN\_WEEK = 7; // constant integer

// Declare variables of different data types

int age = 20;

float height = 5.9;

char grade = 'A';

bool isPassed = true;

// Display initial values

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

cout << "Height: " << height << " feet" << endl;

cout << "Grade: " << grade << endl;

cout << "Passed: " << isPassed << endl;

cout << "Value of PI (constant): " << PI << endl;

cout << "Days in a week (constant): " << DAYS\_IN\_WEEK << endl;

// Perform operations on variables

age += 1; // increment age

height -= 0.1; // decrease height

grade = 'B'; // change grade

isPassed = false; // update boolean value

// Display updated values

cout << "\nAfter updates:" << endl;

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

cout << "Updated Height: " << height << " feet" << endl;

cout << "Updated Grade: " << grade << endl;

cout << "Passed: " << isPassed << endl;

// Uncommenting the following lines would cause errors because constants can't be changed

// PI = 3.14;

// DAYS\_IN\_WEEK = 8;

return 0;

}

|  |  |  |
| --- | --- | --- |
| **Feature** | **Variable** | **Constant** |
| Value change | Can be changed anytime | Cannot be changed after creation |
| Declaration | int x = 5; | const int x = 5; |
| Purpose | Store dynamic data | Store fixed, unchangeable values |

2. Type Conversion

o Write a C++ program that performs both implicit and explicit type conversions and prints the results.

o Objective: Practice type casting in C++.

#include <iostream>

using namespace std;

int main() {

// Implicit type conversion (automatic)

int intVal = 10;

float floatVal = intVal; // int is implicitly converted to float

cout << "Implicit Conversion:" << endl;

cout << "int value: " << intVal << endl;

cout << "Converted to float: " << floatVal << endl;

cout << endl;

// Explicit type conversion (manual)

float num = 9.75;

int intNum = (int)num; // or use static\_cast<int>(num)

cout << "Explicit Conversion:" << endl;

cout << "float value: " << num << endl;

cout << "Converted to int: " << intNum << endl;

return 0;

}

3. Operator Demonstration

o Write a C++ program that demonstrates arithmetic, relational, logical, and bitwise operators. Perform operations using each type of operator and display the results.

o Objective: Reinforce understanding of different types of operators in C++.

Arithmetic operator

#include <iostream>

using namespace std;

int main() {

int a, b;

a = 7;

b = 2;

// printing the sum of a and b

cout << "a + b = " << (a + b) << endl;

// printing the difference of a and b

cout << "a - b = " << (a - b) << endl;

// printing the product of a and b

cout << "a \* b = " << (a \* b) << endl;

// printing the division of a by b

cout << "a / b = " << (a / b) << endl;

// printing the odulo of a by b

cout << "a % b = " << (a % b) << endl;

return 0;

}

2. relational operator

#include <iostream>

using namespace std;

int main() {

int a, b;

a = 3;

b = 5;

bool result;

result = (a == b); // false

cout << "3 == 5 is " << result << endl;

result = (a != b); // true

cout << "3 != 5 is " << result << endl;

result = a > b; // false

cout << "3 > 5 is " << result << endl;

result = a < b; // true

cout << "3 < 5 is " << result << endl;

result = a >= b; // false

cout << "3 >= 5 is " << result << endl;

result = a <= b; // true

cout << "3 <= 5 is " << result << endl;

return 0;

}

3.logical operator

#include <iostream>

using namespace std;

int main() {

bool result;

result = (3 != 5) && (3 < 5); // true

cout << "(3 != 5) && (3 < 5) is " << result << endl;

result = (3 == 5) && (3 < 5); // false

cout << "(3 == 5) && (3 < 5) is " << result << endl;

result = (3 == 5) && (3 > 5); // false

cout << "(3 == 5) && (3 > 5) is " << result << endl;

result = (3 != 5) || (3 < 5); // true

cout << "(3 != 5) || (3 < 5) is " << result << endl;

result = (3 != 5) || (3 > 5); // true

cout << "(3 != 5) || (3 > 5) is " << result << endl;

result = (3 == 5) || (3 > 5); // false

cout << "(3 == 5) || (3 > 5) is " << result << endl;

result = !(5 == 2); // true

cout << "!(5 == 2) is " << result << endl;

result = !(5 == 5); // false

cout << "!(5 == 5) is " << result << endl;

return 0;

}

4. bitwise operator

#include <stdio.h>

int main()

{

// a = 5 (00000101 in 8-bit binary), b = 9 (00001001 in

// 8-bit binary)

unsigned int a = 5, b = 9;

// The result is 00000001

printf("a = %u, b = %u\n", a, b);

printf("a&b = %u\n", a & b);

// The result is 00001101

printf("a|b = %u\n", a | b);

// The result is 00001100

printf("a^b = %u\n", a ^ b);

// The result is 11111111111111111111111111111010

// (assuming 32-bit unsigned int)

printf("~a = %u\n", a = ~a);

// The result is 00010010

printf("b<<1 = %u\n", b << 1);

// The result is 00000100

printf("b>>1 = %u\n", b >> 1);

return 0;

}

3. Control Flow Statements.

1. Grade Calculator

o Write a C++ program that takes a student’s marks as input and calculates the grade based on if-else conditions.

o Objective: Practice conditional statements (if-else).

#include <iostream>

using namespace std;

int main() {

int marks;

// Input the student's marks

cout << "Enter the student's marks (0-100): ";

cin >> marks;

// Check if the marks are within the valid range (0-100)

if (marks >= 0 && marks <= 100) {

// Determine the grade based on the marks

if (marks >= 90) {

cout << "Grade: A" << endl;

} else if (marks >= 80) {

cout << "Grade: B" << endl;

} else if (marks >= 70) {

cout << "Grade: C" << endl;

} else if (marks >= 60) {

cout << "Grade: D" << endl;

} else {

cout << "Grade: F" << endl;

}

} else {

// Handle invalid input

cout << "Invalid marks. Please enter a value between 0 and 100." << endl;

}

return 0;

}

2. Number Guessing Game

o Write a C++ program that asks the user to guess a number between 1 and 100. The program should provide hints if the guess is too high or too low. Use loops to allow the user multiple attempts.

o Objective: Understand while loops and conditional logic.

#include <iostream>

#include <cstdlib> // for rand() and srand()

#include <ctime> // for time()

using namespace std;

int main() {

int secretNumber, userGuess;

// Seed the random number generator

srand(time(0));

secretNumber = rand() % 100 + 1; // random number between 1 and 100

cout << "Welcome to the Number Guessing Game!" << endl;

cout << "Guess a number between 1 and 100." << endl;

// Loop until the user guesses the correct number

while (true) {

cout << "Enter your guess: ";

cin >> userGuess;

if (userGuess < 1 || userGuess > 100) {

cout << "Please enter a number between 1 and 100!" << endl;

} else if (userGuess < secretNumber) {

cout << "Too low! Try again." << endl;

} else if (userGuess > secretNumber) {

cout << "Too high! Try again." << endl;

} else {

cout << "Congratulations! You guessed the correct number: " << secretNumber << endl;

break; // Exit the loop

}

}

return 0;

}

4. Nested Control Structures

o Write a program that prints a right-angled triangle using stars(\*) with a nested loop.

o Objective: Learn nested control structures.

import java.io.\*;

// Java code to demonstrate right star triangle

public class GeeksForGeeks {

// Function to demonstrate printing pattern

public static void StarRightTriangle(int n)

{

int a, b;

// outer loop to handle number of rows

// k in this case

for (a = 0; a < n; a++) {

// inner loop to handle number of columns

// values changing acc. to outer loop

for (b = 0; b <= a; b++) {

// printing stars

System.out.print("\* ");

}

// end-line

System.out.println();

}

}

// Driver Function

public static void main(String args[])

{

int k = 5;

StarRightTriangle(k);

}

}

4. Functions and Scope.

1. Simple Calculator Using Functions

o Write a C++ program that defines functions for basic arithmetic operations (add, subtract, multiply, divide). The main function should call these based on user input.

o Objective: Practice defining and using functions in C++.

# include <iostream>

using namespace std;

int main() {

char op;

float num1, num2;

cout << "Enter operator: +, -, \*, /: ";

cin >> op;

cout << "Enter two operands: ";

cin >> num1 >> num2;

switch(op) {

case '+':

cout << num1 << " + " << num2 << " = " << num1 + num2;

break;

case '-':

cout << num1 << " - " << num2 << " = " << num1 - num2;

break;

case '\*':

cout << num1 << " \* " << num2 << " = " << num1 \* num2;

break;

case '/':

cout << num1 << " / " << num2 << " = " << num1 / num2;

break;

default:

// If the operator is other than +, -, \* or /, error message is shown

cout << "Error! operator is not correct";

break;

}

return 0;

}

2. Factorial Calculation Using Recursion

o Write a C++ program that calculates the factorial of a number using recursion.

o Objective: Understand recursion in functions.

// C++ program to find factorial of a number using recursion

#include <iostream>

using namespace std;

// Define a function to calculate factorial

// recursively

long long factorial(int n)

{

// Base case - If n is 0 or 1, return 1

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

return 1;

}

// Recursive case - Return n multiplied by

// factorial of (n-1)

return n \* factorial(n - 1);

}

int main()

{

int num = 5;

// printing the factorial

cout << "Factorial of " << num << " is "

<< factorial(num) << endl;

return 0;

}

3. Variable Scope

o Write a program that demonstrates the difference between local and global variables in C++. Use functions to show scope.

o Objective: Reinforce the concept of variable scope.

#include <iostream>

using namespace std;

int x = 10; // Global variable

void showGlobal() {

cout << "Global x = " << x << endl;

}

void showLocal() {

int x = 5; // Local variable

cout << "Local x = " << x << endl;

}

int main() {

showGlobal();

showLocal();

cout << "Back in main - Global x = " << x << endl;

return 0;

}

5. Arrays and Strings

1. Array Sum and Average

o Write a C++ program that accepts an array of integers, calculates the sum and average, and displays the results.

o Objective: Understand basic array manipulation.

#include <iostream>

using namespace std;

int main()

{

cout << "\n\nWelcome to Studytonight :-)\n\n\n";

cout << " ===== Program to find the Sum and Average of the Array elements ===== \n\n";

//i to iterate the outer loop and j for the inner loop

int i, n;

//declaring sum and average as double because average can be a fractional value

double sum=0, average=0;

cout << "\n\nEnter the number integers you want in an array: ";

cin >> n;

//Declaring an array containing 'n' integers

int arr[n];

cout << "\n\n Enter " << n << " integers into an array :\n\n";

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

{

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

cin >> arr[i];

}

cout << "\n\n The Elements of the Array are: \n\n";

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

{

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

sum += arr[i];

}

average = sum/n;

cout << "\n\n The Sum of the Elements of the Array is : " << sum << "\n\n";

cout << "\n\n The Average of the Elements of the Array is : " << average << "\n\n";

cout << "\n\n";

return 0;

}

2. Matrix Addition

o Write a C++ program to perform matrix addition on two 2x2 matrices.

o Objective: Practice multi-dimensional arrays.

// C++ program for addition

// of two matrices

#include <bits/stdc++.h>

using namespace std;

#define N 4

// This function adds A[][] and B[][],

// and stores the result in C[][]

void add(int A[][N], int B[][N], int C[][N])

{

int i, j;

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

for (j = 0; j < N; j++)

C[i][j] = A[i][j] + B[i][j];

}

// Driver code

int main()

{

int A[N][N] = { { 1, 1, 1, 1 },

{ 2, 2, 2, 2 },

{ 3, 3, 3, 3 },

{ 4, 4, 4, 4 } };

int B[N][N] = { { 1, 1, 1, 1 },

{ 2, 2, 2, 2 },

{ 3, 3, 3, 3 },

{ 4, 4, 4, 4 } };

// To store the result

int C[N][N];

int i, j;

add(A, B, C);

cout << "Result matrix is " << endl;

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

for (j = 0; j < N; j++)

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

cout << endl;

}

return 0;

}

3. String Palindrome Check

o Write a C++ program to check if a given string is a palindrome (reads the same forwards and backwards).

o Objective: Practice string operations.

#include <iostream>

#include <string>

using namespace std;

bool isPalindrome(string str) {

int start = 0;

int end = str.length() - 1;

while (start < end) {

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

return false; // Not a palindrome

}

start++;

end--;

}

return true; // It's a palindrome

}

int main() {

string input;

cout << "Enter a string: ";

cin >> input;

if (isPalindrome(input)) {

cout << input << " is a palindrome." << endl;

Introduction to Object-Oriented Programming.

1. Class for a Simple Calculator

} else {

cout << input << " is not a palindrome." << endl;

}

return 0;

}

6. Introduction to Object-Oriented Programming

1. Class for a Simple Calculator

o Write a C++ program that defines a class Calculator with functions for addition, subtraction, multiplication, and division. Create objects to use these functions.

o Objective: Introduce basic class structure.

#include <iostream>

using namespace std;

class Calculator {

public:

// Addition

float add(float a, float b) {

return a + b;

}

// Subtraction

float subtract(float a, float b) {

return a - b;

}

// Multiplication

float multiply(float a, float b) {

return a \* b;

}

// Division

float divide(float a, float b) {

if (b != 0)

return a / b;

else {

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

return 0;

}

}

};

int main() {

Calculator calc;

float num1, num2;

cout << "Enter two numbers: ";

cin >> num1 >> num2;

cout << "Addition: " << calc.add(num1, num2) << endl;

cout << "Subtraction: " << calc.subtract(num1, num2) << endl;

cout << "Multiplication: " << calc.multiply(num1, num2) << endl;

cout << "Division: " << calc.divide(num1, num2) << endl;

return 0;

}

2. Class for Bank Account

o Create a class BankAccount with data members like balance and member functions like deposit and withdraw. Implement encapsulation by keeping the data members private.

o Objective: Understand encapsulation in classes.

#include <iostream>

using namespace std;

class BankAccount {

private:

float balance; // Encapsulated data member

public:

// Constructor to initialize balance

BankAccount(float initialBalance) {

if (initialBalance >= 0)

balance = initialBalance;

else {

cout << "Initial balance can't be negative. Setting balance to 0." << endl;

balance = 0;

}

}

// Function to deposit money

void deposit(float amount) {

if (amount > 0) {

balance += amount;

cout << "Deposited: " << amount << endl;

} else {

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

}

}

// Function to withdraw money

void withdraw(float amount) {

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

balance -= amount;

cout << "Withdrawn: " << amount << endl;

} else {

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

}

}

// Function to check current balance

float getBalance() {

return balance;

}

};

int main() {

BankAccount myAccount(1000.0); // Create account with initial balance

cout << "Initial Balance: " << myAccount.getBalance() << endl;

myAccount.deposit(500);

cout << "Current Balance: " << myAccount.getBalance() << endl;

myAccount.withdraw(200);

cout << "Current Balance: " << myAccount.getBalance() << endl;

myAccount.withdraw(2000); // Attempt invalid withdrawal

return 0;

}

3. Inheritance Example

o Write a program that implements inheritance using a base class Person and derived classes Student and Teacher. Demonstrate reusability through inheritance.

o Objective: Learn the concept of inheritance.

#include <iostream>

#include <string>

using namespace std;

// Base class

class Person {

protected:

string name;

int age;

public:

// Constructor

Person(string n, int a) : name(n), age(a) {}

// Virtual function to display details (can be overridden in derived classes)

virtual void display() {

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

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

}

};

// Derived class Student

class Student : public Person {

private:

string schoolName;

// Constructor

Student(string n, int a, string school) : Person(n, a), schoolName(school) {}

// Overriding the display function

void display() override {

Person::display(); // Calling base class display

cout << "School: " << schoolName << endl;

}

};

// Derived class Teacher

class Teacher : public Person {

private:

string subject;

public:

// Constructor

Teacher(string n, int a, string sub) : Person(n, a), subject(sub) {}

// Overriding the display function

void display() override {

Person::display(); // Calling base class display

cout << "Subject: " << subject << endl;

}

};

int main() {

// Creating objects of Student and Teacher

Student s("John Doe", 20, "XYZ High School");

Teacher t("Dr. Smith", 45, "Mathematics");

// Displaying information of Student and Teacher

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

s.display();

cout << endl;

cout << "Teacher Details:" << endl;

t.display();

return 0;

}