**Module 3 Introduction to OOPS Programming**

**Q-1.Introduction to C++**

**THEORY EXERCISE:**

1. **What are the key differences between Procedural Programming and Object-Oriented Programming (OOP) ?**

* **Procedural Programming (POP) focuses on functions and procedures, following a top-down approach. In contrast, OOP organizes code into objects and classes, using a bottom-up approach. POP lacks features like inheritance and encapsulation, while OOP supports them, making code more modular and reusable. OOP also promotes data hiding and abstraction.**

**2.List and explain the main advantages of OOP over POP ?**

* **OOP offers better code reusability through inheritance and promotes data security via encapsulation. It simplifies complex programs by modeling real-world entities using objects. OOP supports modularity, making programs easier to debug, maintain, and scale. It also enhances code flexibility through polymorphism.**

**3. Explain the steps involved in setting up a C++ development environment.**

* **First, install a C++ compiler like GCC or MSVC. Then, install an IDE or text editor such as Code::Blocks, Dev C++, or VS Code. Configure the compiler in the IDE settings. Optionally, set environment variables if using command line tools. Finally, write and run your first program to test the setup.**

**4. What are the main input/output operations in C++? Provide examples?**

* **C++ uses cin for input and cout for output, both from the <iostream> header. For example: cpp , Copy Edit**

**int age;**

**cin>> age; // Input**

**cout << "Age is: " << age; // Output**

**Q-2. Variables, Data Types, and Operators**

**THEORY EXERCISE:**

1. **What are the different data types available in C++? Explain with examples?**
   * In C++, data types define the type of data a variable can store, helping the compiler allocate memory and determine the operations allowed on that data. They are mainly classified into fundamental, derived, and user-defined types**.**

**(A).Fundamental (Primitive) Data Types – Basic built-in types:**

* + int – Stores whole numbers (e.g., int age = 25;)
  + float – Stores decimal numbers with single precision (e.g., float price = 99.5;)
  + double – Stores decimal numbers with double precision (e.g., double pi = 3.14159;)
  + char – Stores a single character (e.g., char grade = 'A';)
  + bool – Stores true or false (e.g., bool isPassed = true;)
  + void – Represents no value (used for functions without a return type)

**(B).Derived Data Types – Formed from fundamental types:**

* Array – Stores multiple elements of the same type (e.g., int marks[5] = {90, 85, 88, 92, 95};)
* Pointer – Stores memory address of a variable (e.g., int\* ptr = &age;)
* Reference – Alias for another variable (e.g., int& ref = age;)

**(C).User-defined Data Types – Created by the programmer:**

* struct – Groups related variables (e.g., struct Student { int roll; char name[20]; };)
* class – Defines objects with attributes and methods (OOP concept)
* enum – Defines a set of named constants (e.g., enum Day { Mon, Tue, Wed };)

**Example:-**

**#include <iostream>**

**using namespace std;**

**main() {**

**int age = 20; // int type**

**float height = 5.9; // float type**

**char grade = 'A'; // char type**

**bool isPassed = true; // bool type**

**double pi = 3.14159265; // double type**

**cout << "Age: " << age << "\nHeight: " << height**

**<< "\nGrade: " << grade << "\nPassed: " << isPassed**

**<< "\nPi: " << pi;**

**}**

1. **Explain the difference between implicit and explicit type conversion in C++?**

## **1. Implicit Type Conversion (Type Casting)**

Also called **type promotion** or **type coercion**.

* Done **automatically by the compiler**.
* Converts smaller data types to larger ones to avoid data loss (**type promotion**).
* **No special syntax needed**

## **2. Explicit Type Conversion (Type Casting)**

* Done **manually by the programmer**.
* Uses casting operators or constructor syntax.
* You decide exactly how the conversion happens.

**3.What are the different types of operators in C++? Provide examples of each?**

## **1. Arithmetic Operators**

Used for mathematical calculations.

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| + | Addition | a + b |
| - | Subtraction | a - b |
| \* | Multiplication | a \* b |
| / | Division | a / b |
| % | Modulus (remainder) | a % b |

## **2. Relational (Comparison) Operators**

Used to compare values; returns true (1) or false (0).

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| == | Equal to | a == b |
| != | Not equal to | a != b |
| < | Less than | a < b |
| > | Greater than | a > b |
| <= | Less than or equal | a <= b |
| >= | Greater than or equal | a >= b |

## **3. Logical Operators**

Used for logical operations.

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| && | Logical AND | (a > 5 && b < 5) |
| ` |  | ` |
| ! | Logical NOT | !(a > b) |

## **4. Assignment Operators**

Used to assign values to variables.

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| = | Assign | a = 5 |
| += | Add and assign | a += 3 (same as a = a + 3) |
| -= | Subtract and assign | a -= 2 |
| \*= | Multiply and assign | a \*= 2 |
| /= | Divide and assign | a /= 2 |
| %= | Modulus and assign | a %= 2 |

## **5. Increment/Decrement Operators**

Increase or decrease a variable by 1.

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| ++ | Increment | a++ or ++a |
| -- | Decrement | a-- or --a |

## **6. Conditional (Ternary) Operator**

Shorthand for if-else.

**Syntax:**

cpp

(condition) ? expression1 : expression2;

## **7. Bitwise Operators**

Operate on data at the **bit level**.

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| & | AND | a & b |
| ` | ` | OR |
| ^ | XOR | a ^ b |
| ~ | NOT | ~a |
| << | Left shift | a << 1 |
| >> | Right shift | a >> 1 |

## **8. Special Operators**

* **sizeof** – Returns size of data type/variable.

cpp

cout << sizeof(int);

* **typeid** – Returns type information (requires <typeinfo>).

cpp

cout << typeid(a).name();

* **Scope Resolution (::)** – Access global variables or class members.
* **Member Access (. and ->)** – Access object members.

## **Example Program Using All Types**

**#include <iostream>**

**using namespace std;**

**main(){**

**int a = 10, b = 3;**

**// Arithmetic**

**cout << "Add: " << a + b << endl;**

**// Relational**

**cout << (a > b) << endl;**

**// Logical**

**cout << (a > 5 && b < 5) << endl;**

**// Assignment**

**a += 2;**

**cout << "a after += 2: " << a << endl;**

**// Increment**

**cout << "Pre-increment: " << ++b << endl;**

**// Ternary**

**int max = (a > b) ? a : b;**

**cout << "Max: " << max << endl;**

**// Bitwise**

**cout << "a & b: " << (a & b) << endl;**

**// Special**

**cout << "Size of int: " << sizeof(int) << endl;**

**}**

**4. Explain the purpose and use of constants and literals in C++?**