**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++?**

* In C++, constants and literals are used to represent fixed values in a program that do not change during execution. A constant is a variable whose value is fixed after initialization, declared using the const keyword or #define preprocessor directive. For example, const int MAX = 100; ensures that MAX cannot be modified later, which improves code reliability and readability. Literals are the actual fixed values used directly in code, such as numbers (42), characters ('A'), strings ("Hello"), or boolean values (true). Literals are the source values assigned to variables or constants. Using constants makes code more maintainable, as you can change the value in one place without modifying multiple occurrences, while literals provide direct representation of data in the program. Together, they help write clear, predictable, and less error-prone programs**.**

**Q-3. Control Flow Statements?**

**THEORY EXERCISE:**

**1. What are conditional statements in C++? Explain the if-else and switch statements?**

**Ans:-**

* In C++, conditional statements allow the program to make decisions and execute different code based on conditions.

1.If-else Statement:-

Used to execute one block of code if a condition is true, and another block if it is false.

**2.switch Statement**:-

Used when you want to choose between multiple options based on a single variable’s value. It uses case labels and an optional default.

**2. What is the difference between for, while, and do-while loops in C++?**

**1.for loop:- Used when the number of iterations is known in advance. Initialization, condition, and update are in one line.**

**2.while loop:-** **Used when the number of iterations is not known beforehand; condition is checked before each iteration.**

**3.do-while loop:-** **Similar to while, but condition is checked after executing the loop body, so it runs at least once.**

**Key difference:- for is compact for counted loops, while checks before execution, and do-while guarantees at least one run.**

1. **How are break and continue statements used in loops? Provide examples.?**

* In C++, break and continue are control statements used inside loops to alter their normal flow:

Break:-

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

if(i == 3) break;

cout << i << " "; // Output: 1 2

}

Continue:-

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

if(i == 3) continue;

cout << i << " "; // Output: 1 2 4 5

}

1. **Explain nested control structures with an example?**

* In C++, nested control structures occur when one control structure (like if, for, while, or switch) is placed inside another. They allow more complex decision-making or looping by combining multiple conditions or iterations.
* **Example – Nested if in a for loop:**

for(int i = 1; i <= 3; i++) { // Outer loop

if(i % 2 == 0) { // Inner if condition

cout << i << " is even\n";

} else {

cout << i << " is odd\n";

}

}

**How it works:**

* The outer for loop runs from 1 to 3.
* Inside it, the if-else checks whether the current number is even or odd.

Nested control structures are useful when you need multiple levels of logic, such as loops within loops, or conditions inside loops, to handle complex program flows.