**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.

**Q-4 Function And Scope?**

**THEORY EXERCISE:**

**1.What is a function in C++? Explain the concept of function declaration, definition, and calling?**

In C++, a function is a self-contained block of statements that performs a specific task. It allows code to be organized into reusable units, reducing repetition and improving readability. Functions can take input values (parameters) and may return a result.

* Function Declaration (Prototype): This is a statement that tells the compiler the function’s name, return type, and parameters before it is used in the program. It acts as a promise that the function exists somewhere in the code.
* Function Definition: This part contains the actual body of the function where the instructions are written. It explains how the function will perform its task.
* Function Calling: This is the process of using a function in a program. When a function is called, the program control transfers to it, executes the instructions, and then returns to the point where it was called.

Together, these steps ensure that functions can be properly recognized, executed, and reused in a C++ program.

**2.What is the scope of variables in C++? Differentiate between local and global scope?**

**Ans:-**In C++, the scope of a variable refers to the region of a program where the variable is accessible. Scope determines the lifetime and visibility of a variable within the program. Variables can have different scopes depending on where they are declared.

* Local Scope: A variable declared inside a function, loop, or block has local scope. It is accessible only within that specific block and is created when the block starts executing, then destroyed when the block ends. Local variables help prevent interference with other parts of the program.
* Global Scope: A variable declared outside all functions has global scope. It is accessible from any function or block in the program. Global variables exist for the entire duration of the program and retain their values between function calls**.**

**3. Explain recursion in C++ with an example.?**

**Ans:-** In C++, recursion is a programming technique where a function calls itself directly or indirectly to solve a problem. Recursion is typically used for problems that can be divided into smaller, similar subproblems. Every recursive function must have:

**1.Base Case**: A condition that stops the recursion to prevent infinite calls.

**2.Recursive Case**: The part where the function calls itself with modified arguments to gradually approach the base case.

>The process works like repeatedly breaking down a task into simpler forms until it reaches the simplest form that can be solved directly. Recursion is often used in tasks like calculating factorials, traversing trees, and solving mathematical sequences.

**For example**, in calculating a factorial, the function keeps calling itself with a smaller number until it reaches 1, then the results are combined as the calls return. Without a base case, recursion would continue indefinitely, causing a stack overflow.

**4. What are function prototypes in C++? Why are they used?**

**Ans:-** In C++, a function prototype is a declaration of a function that tells the compiler about the function’s name, return type, and parameters before its actual definition appears. It does not include the function body.

Purpose and Use:

* Introduces the function to the compiler so it can be called before it is defined in the code.
* Helps with type checking by ensuring the correct number and type of arguments are passed when the function is called.
* Improves code organization, especially when functions are defined after the main() function or in separate files.
* Prevents errors related to calling functions that the compiler has not yet seen.

**Q-5. Arrays and Strings?**

**THEORY EXERCISE:**

**1.What are arrays in C++? Explain the difference between single-dimensional and multi- dimensional arrays.?**

**Ans:-**In C++, an array is a collection of elements of the same data type stored in contiguous memory locations and accessed using an index. Arrays allow storing multiple values under a single variable name, making data handling easier. The index of an array starts from 0 for the first element.

* **Single-Dimensional Array**: This is a linear list of elements arranged in a single row. It is useful for storing simple lists like marks, ages, or names. Elements are accessed using one index, such as **arr[0], arr[1].**
* **Multi-Dimensional Array**: This is an array of arrays, where elements are stored in a table-like format with rows and columns. The most common form is the two-dimensional array, which can store data in a matrix form. Elements are accessed using multiple indices, such as **arr[2][3].**

**2.Explain string handling in C++ with examples.?**

* In C++, string handling refers to the methods and operations used to create, store, and manipulate sequences of characters. Strings can be managed in two main ways: using C-style strings (character arrays) or the C++ string class from the Standard Template Library (STL).

**1.C-Style Strings:-**

* These are arrays of characters terminated by a null character '\0'. They are handled using functions from the <c string> library like strlen(), strcpy(), and strcat(). However, they require manual handling of memory and null termination

**2.C++ string Class:-**

* Provided by the <string> header, this class allows easy manipulation of strings with features like dynamic sizing and built-in operators. Common operations include concatenation (+), comparison (==), finding substrings (find()), and getting length (length()).

**3. How are arrays initialized in C++? Provide examples of both 1D and 2D arrays?**

* In C++, arrays can be initialized when declared or later by assigning values

**1D Array:**

**int a[5] = {1, 2, 3, 4, 5}; // full initialization**

**int b[5] = {1, 2}; // rest become 0**

**int c[] = {10, 20, 30}; // size auto-calculated**

**2D Array:**

**int m[2][3] = {{1, 2, 3}, {4, 5, 6}}; // full initialization**

**int n[2][3] = {{1, 2}, {4}}; // rest become 0**

**int p[2][3] = {1, 2, 3, 4, 5, 6}; // flat list**

**4. Explain string operations and functions in C++?**

* In C++, string operations and functions allow you to create, modify, and analysis strings easily, especially using the string class from the <string> header.

**Common String Operations**

1. **Concatenation:** Joining two strings using + or append().
2. **Comparison:** Comparing strings using ==, <, >, or compare().
3. **Accessing Characters**: Using at(index) or [].
4. **Substring Extraction:** Using substr(position, length).
5. **Finding Substring:** Using find() or rfind().
6. **Modifying**: Using replace(), insert(), erase(), or clear().
7. **Length:** Using length() or size()

**Q-6 Introduction to Object-Oriented Programming(OOP)?**

**THEORY EXERCISE:**

**1. Explain the key concepts of Object-Oriented Programming (OOP)?**

**Ans:- Key concepts of Object-Oriented Programming (OOP) in C++:**

1. **Class**: A blueprint or template for creating objects, defining attributes (data members) and behaviors (member functions).
2. **Object**: An instance of a class representing real-world entities, with its own state and behavior.
3. **Data Encapsulation & Abstraction**: Together, they involve bundling data and methods inside a class while hiding unnecessary implementation details. Access to data is controlled using access specifiers (private, public, protected), and only the essential features are exposed to the user.
4. **Inheritance**: Allows a derived class to acquire the properties and methods of a base class, promoting code reusability.
5. **Polymorphism**: Enables the same function or operator to work in different ways (function overloading, operator overloading, virtual functions).
6. **Dynamic Binding (Late Binding):** The method to be executed is decided at runtime, mainly through virtual functions, enabling runtime polymorphism**.**

**2. What are classes and objects in C++? Provide an example?**

In C++, a class is a user-defined blueprint or template that defines the data (attributes) and the functions (methods) that operate on that data. An object is an instance of a class that has its own copy of the data members and can use the class’s methods.

* **Class**: Defines the structure and behavior of objects.
* **Object**: A real-world entity created from a class, which stores its own state and can perform actions defined in the class.

**Example**:-

**#include <iostream>**

**using namespace std;**

**// Class definition**

**class Car {**

**public:**

**string brand;**

**int year;**

**void display() {**

**cout << "Brand: " << brand << ", Year: " << year << endl;**

**}**

**};**

**main() {**

**// Creating objects**

**Car car1;**

**car1.brand = "Toyota";**

**car1.year = 2020;**

**Car car2;**

**car2.brand = "Honda";**

**car2.year = 2022;**

**// Calling methods**

**car1.display();**

**car2.display();**

**}**

**3. What is inheritance in C++? Explain with an example?**

* In C++, inheritance is an OOP concept that allows one class (derived class) to acquire the properties and behaviors (data members and member functions) of another class (base class). It promotes code reusability and creates a hierarchical relationship between classes.

**Types of Inheritance in C++:-**

* Single (one base → one derived)
* Multiple (multiple bases → one derived)
* Multilevel (derived from another derived)
* Hierarchical (one base → multiple derived)
* Hybrid (combination of types)

**Example:- Single Inheritance**

**#include <iostream>**

**using namespace std;**

**// Base class**

**class Vehicle {**

**public:**

**string brand;**

**void showBrand() {**

**cout << "Brand: " << brand << endl;**

**}**

**};**

**// Derived class (inherits from Vehicle)**

**class Car : public Vehicle {**

**public:**

**int year;**

**void showYear() {**

**cout << "Year: " << year << endl;**

**}**

**};**

**main() {**

**Car c1;**

**c1.brand = "Toyota"; // inherited from Vehicle**

**c1.year = 2022; // own member**

**c1.showBrand(); // base class method**

**c1.showYear(); // derived class method**

**}**

**4. What is encapsulation in C++? How?**

* In C++, encapsulation is the process of bundling data (variables) and methods (functions) into a single unit called a class, and restricting direct access to the data from outside the class.  
  This is done using access specifiers:

**1.private:** accessible only inside the class.

**2.protected**: accessible inside the class and by derived classes.

**3.public:** accessible from anywhere in the program.

**How Encapsulation Works**:-

* Declare class variables as private so they cannot be directly modified.
* Provide public getter and setter functions to control access and modification of the data.
* This ensures data hiding and security, preventing unintended changes.