Module 3

Introduction to OOPS Programming

THEORY EXERCISE:

1. Introduction to C++

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

ANS.

Feature	**Procedural Programming**	**Object-Oriented Programming (OOP)**
Approach	Follows a linear, step-by-step approach	Follows a modular, object-based approach
Structure	Program is divided into functions	Program is divided into objects & classes
Data Handling	Data is global and shared among functions	Data is encapsulated within objects
Code Reusability	Limited reuse (functions can be reused)	High reuse through inheritance & polymorphism
Encapsulation	No built-in data hiding mechanism	Uses encapsulation to protect data
Security	Less secure (global data can be accessed from anywhere)	More secure (private/protected access control)
Examples	C, Pascal, FORTRAN	Java, Python, C++, C#

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

ANS.

1.Code Reusability (Inheritance):OOP allows classes to inherit properties and methods from other classes, reducing code duplication and improving efficiency.

2. Encapsulation: Data is bundled inside objects using access modifiers (private, protected, public), preventing unintended modification and improving security.

3. Data Hiding: Unlike POP, where data is globally accessible, OOP restricts access to critical data, ensuring better control and protection.

4. Modularity: OOP organizes programs into smaller, manageable modules (objects), making complex applications easier to develop and maintain.

5. Maintainability: Since OOP code is structured and modular, debugging, updating, and scaling applications is much easier compared to POP.

6. Polymorphism: OOP supports method overloading and overriding, allowing a single function name to be used for different functionalities, improving flexibility.

7. Extensibility & Scalability: New features and functionalities can be added easily without modifying the existing code, making OOP ideal for large applications.

8. Real-world Modelling: OOP is closer to real-world concepts (e.g., objects, attributes, behaviors), making it easier to design software that mirrors real-world scenarios.

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

ANS.

Step 1: Install a C++ Compiler

A compiler translates C++ code into machine code. Popular C++ compilers include:

- **GCC (GNU Compiler Collection)** – Available on Linux/macOS and can be installed on Windows via MinGW.
- **MSVC (Microsoft Visual C++ Compiler)** – Comes with Visual Studio.
- **Clang** – Used in macOS and Linux.

🔹 **Installing GCC on Windows (via MinGW-w64):**

1. Download **MinGW-w64** from MinGW-w64 official site.
2. Install it and add the bin folder to the system **PATH** (Environment Variables).
3. Verify installation by running g++ --version in the command prompt.

🔹 **Installing GCC on Linux/macOS:**

- Linux: Run sudo apt install g++ (Ubuntu/Debian) or sudo yum install gcc-c++ (CentOS).
- macOS: Install **Xcode Command Line Tools** using xcode-select --install.

Step 2: Choose a Code Editor or IDE

A good editor/IDE improves productivity. Some popular choices are:

- **Visual Studio Code (VS Code)** – Lightweight, supports extensions.
- **Code::Blocks** – Simple IDE with built-in compiler.
- **Dev-C++** – Good for beginners.
- **Eclipse CDT** – Feature-rich for large projects.
- **CLion (JetBrains)** – Powerful, but paid.

✅ **Recommended:** VS Code with C++ extensions for a smooth experience.

Step 3: Configure Your IDE or Editor

If using **VS Code**, set up C++ as follows:

1. Install **C/C++ extension** from the VS Code marketplace.
2. Install **MinGW-w64** (if using Windows).
3. Configure **tasks.json** and **launch.json** for compiling and running C++ programs.
4. Test with a simple **Hello World** program.

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

Ans. **1. Input using** `std::cin`

`std::cin` is used to read data from the standard input (keyboard).

code:

```
#include <iostream>

using namespace std;

int main() {
```

```
    int number;
    cout << "Enter a number: ";
    cin >> number; // Reads an integer from the user
    cout << "You entered: " << number << endl;
    return 0;
}
```

2. **Output using `std::cout`**

`std::cout` is used to write data to the standard output (console).

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, World!" << endl; // Outputs a string
    int x = 10;
    cout << "The value of x is: " << x << endl; // Outputs a variable
    return 0;
}
```

3. **Error output using std::cerr**

std::cerr is used to output error messages. It is unbuffered, meaning the output is immediately displayed.

```
#include <iostream>
using namespace std;
int main() {
    int age;
    cout << "Enter your age: ";
    cin >> age;
    if (age < 0) {
        cerr << "Error: Age cannot be negative!" << endl; // Error message
    } else {
        cout << "Your age is: " << age << endl;
```

```
    }

    return 0;
```

4. **Logging output using std::clog**

std::clog is used for logging purposes. It is buffered, meaning the output may be delayed until the buffer is flushed.

```
#include <iostream>

using namespace std;

int main() {

    clog << "This is a log message." << endl; // Logging message

    return 0;

}
```

2. Variables, Data Types, and Operators

1. What are the different data types available in C++? Explain with examples.

ANS. 1. **Primitive Data Types**

These are the basic data types provided by the language.

Integer Types

EX: int: Used to store whole numbers (positive or negative).

```
int age = 25;
```

EX: short: A shorter integer, typically uses less memory than int.

```
short temperature = -10;
```

EX: long: A longer integer, typically uses more memory than int.

```
long population = 7800000000L;
```

EX: long long: An even longer integer, used for very large numbers.

```
EX : long long bigNumber = 922337203685477580 7LL;
```

b. **Floating-Point Types**

EX: float: Used to store single-precision floating-point numbers.

float pi = 3.14f;

EX: double: Used to store double-precision floating-point numbers.

double precisePi = 3.1415926535;

EX: long double: Used for extended precision floating-point numbers.

long double veryPrecisePi = 3.141592653589793238L;

2. Explain the difference between implicit and explicit type conversion in C++.

ANS.Implicit Type Conversion (Automatic Type Conversion):

Definition: Implicit type conversion is automatically performed by the compiler when a value of one data type is assigned to a variable of another compatible data type.

Example:

int a = 10;

double b = a; // Implicit conversion from int to double

Key Points:

No explicit instruction is required from the programmer.

Occurs when there is no loss of data (e.g., int to double).

Also known as "type coercion."

Explicit Type Conversion (Type Casting):

Definition: Explicit type conversion is manually performed by the programmer using casting operators to convert a value from one data type to another.

Example:

double a = 10.5;

int b = (int)a; // Explicit conversion from double to int

Key Points:

Requires explicit instruction using casting operators like (int), (double), etc.

Can lead to data loss if not used carefully (e.g., converting double to int truncates the decimal part).

Also known as "type casting."

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

ANS.1. Arithmetic Operators:

Used for mathematical operations.

Examples:

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

int sum = a + b; // Addition (+)

int diff = a - b; // Subtraction (-)

int product = a * b; // Multiplication (*)

int quotient = a / b; // Division (/)

int remainder = a % b; // Modulus (%)
```

2. Relational Operators:

Used to compare two values.

Examples:

```
int a = 10, b = 20;

bool isEqual = (a == b); // Equal to (==)

bool isNotEqual = (a != b); // Not equal to (!=)

bool isGreater = (a > b); // Greater than (>)

bool isLess = (a < b); // Less than (<)

bool isGreaterOrEqual = (a >= b); // Greater than or equal to (>=)

bool isLessOrEqual = (a <= b); // Less than or equal to (<=)
```

3. Logical Operators:

Used to combine multiple conditions.

Examples:

```
bool x = true, y = false;

bool result1 = x && y; // Logical AND (&&)

bool result2 = x || y; // Logical OR (||)

bool result3 = !x; // Logical NOT (!)
```

4. Assignment Operators:

Used to assign values to variables.

Examples:

```
int a = 10;

a += 5; // Equivalent to a = a + 5

a -= 3; // Equivalent to a = a - 3

a *= 2; // Equivalent to a = a * 2

a /= 4; // Equivalent to a = a / 4

a %= 3; // Equivalent to a = a % 3
```

5. Increment and Decrement Operators:

Used to increase or decrease the value of a variable by 1.

Examples:

int a = 10;

a++; // Post-increment (a = a + 1)

++a; // Pre-increment (a = a + 1)

a--; // Post-decrement (a = a - 1)

--a; // Pre-decrement (a = a - 1)

6. Bitwise Operators:

Used to perform operations on binary representations of data.

Examples:

int a = 5, b = 3; // 5 = 0101, 3 = 0011

int result1 = a & b; // Bitwise AND (0101 & 0011 = 0001)

int result2 = a | b; // Bitwise OR (0101 | 0011 = 0111)

int result3 = a ^ b; // Bitwise XOR (0101 ^ 0011 = 0110)

int result4 = ~a; // Bitwise NOT (~0101 = 1010)

int result5 = a << 1; // Left shift (0101 << 1 = 1010)

int result6 = a >> 1; // Right shift (0101 >> 1 = 0010)

7. Ternary Operator:

A shorthand for if-else statements.

Example:

int a = 10, b = 20;

int max = (a > b) ? a : b; // If a > b, max = a; else max = b

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

ANS.Constants:

Definition: Constants are variables whose values cannot be changed after initialization.

Purpose:

- To define fixed values that should not be modified during program execution.
- Improves code readability and maintainability.

Syntax:

const int MAX_VALUE = 100; // MAX_VALUE is a constant

Use Case:

```
const double PI = 3.14159;

double radius = 5.0;

double area = PI * radius * radius; // PI cannot be changed
```

Literals:

1. **Definition**: Literals are fixed values directly used in the code.
2. **Types**:
   - **Integer Literals**: int a = 10;
   - **Floating-Point Literals**: double b = 3.14;
   - **Character Literals**: char c = 'A';
   - **String Literals**: string s = "Hello";
   - **Boolean Literals**: bool flag = true;
3. **Purpose**:
   - To represent fixed values in the code.
   - Used directly in expressions or assignments.
4. **Example**:

- int age = 25; // 25 is an integer literal
- double price = 99.99; // 99.99 is a floating-point literal
- char grade = 'A'; // 'A' is a character literal
- string name = "Alice"; // "Alice" is a string literal
- bool isActive = true; // true is a boolean literal

3. Control Flow Statements

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

ANS.Conditional Statements in C++

Conditional statements in C++ are used to execute different blocks of code based on whether a specified condition is true or false. They allow the program to make decisions and perform actions accordingly. The two most commonly used conditional statements in C++ are if-else and switch.

1. if-else Statement

The if-else statement is used to execute a block of code if a condition is true. If the condition is false, an alternative block of code (in the else part) can be executed.

Syntax

```
if (condition) {

   // Code to execute if the condition is true

} else {

   // Code to execute if the condition is false

}
```

2. switch Statement

The switch statement is used to select one of many code blocks to execute based on the value of a variable or expression. It is often used as an alternative to multiple if-else statements when dealing with a single variable that can take multiple values.

Syntax:

```
switch (expression) {
```

```
  case value1:
    // Code to execute if expression == value1
    break;
  case value2:
    // Code to execute if expression == value2
    break;
  default:
    // Code to execute if expression doesn't match any case
}
```

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

ANS. 1. for Loop

1.Initialization: The initialization of the loop variable is done **inside the loop syntax**.

for (int i = 0; i < 5; i++)

2.Condition Check: The condition is checked **before each iteration**. If false, the loop terminates.

3.Update: The loop variable is updated **automatically** after each iteration (e.g., i++).

4.Use Case: Best for situations where the **number of iterations is known** (e.g., iterating over an array).

5.Execution: The loop body **may not execute** if the condition is false initially.

2. while Loop

1.Initialization: The loop variable is initialized **outside the loop**.

int i = 0;

while (i < 5)

2.Condition Check: The condition is checked **before each iteration**. If false, the loop terminates.

3.Update: The loop variable is updated **manually inside the loop body**.

i++;

4.Use Case: Best for situations where the **number of iterations is unknown** (e.g., reading input until a condition is met).

5.Execution: The loop body **may not execute** if the condition is false initially.

3. do-while Loop

1.Initialization: The loop variable is initialized **outside the loop**.

int i = 0;

do { ... } while (i < 5);

2.Condition Check: The condition is checked **after each iteration**. If false, the loop terminates.

3.Update: The loop variable is updated **manually inside the loop body**.

i++;

4.Use Case: Best for situations where the loop body **must execute at least once** (e.g., displaying a menu).

5.Execution: The loop body **always executes at least once**, even if the condition is false initially.

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

ANS. **1. break Statement**

The break statement is used to exit a loop immediately when a certain condition is met.

Example of break in a for loop:

```cpp
#include <iostream>
using namespace std;

int main() {
    for (int i = 0; i < 10; i++) {
        if (i == 5) {
            break; // Exit the loop when i equals 5
        }
        cout << i << endl;
    }
    return 0;
}
```

Output:

Copy

```
0
1
2
3
4
```

2. continue Statement

The continue statement skips the remaining code in the loop for the current iteration and moves to the next iteration.

Example of continue in a for loop:

```cpp
#include <iostream>
using namespace std;

int main() {
    for (int i = 0; i < 10; i++) {
        if (i % 2 == 0) {
            continue; // Skip even numbers
        }
        cout << i << endl;
    }
    return 0;
}
```

Output:

Copy

```
1
3
5
7
9
```

4. Explain nested control structures with an example.

ANS. **Nested Control Structures**

A **nested control structure** occurs when one control structure (such as loops, conditional statements, or switch cases) is placed inside another. These structures allow for more complex decision-making and iterative processes in programming.

Example: Nested If-Else

C ++

```
#include <iostream>

using namespace std;

int main() {

   int age;

   char gender;

   cout << "Enter age: ";

   cin >> age;

   cout << "Enter gender (M/F): ";

   cin >> gender;

   if (age >= 18) {  // Outer if condition

      if (gender == 'M') {  // Inner if condition

         cout << "You are an adult male." << endl;

      } else if (gender == 'F') {

         cout << "You are an adult female." << endl;

      } else {

         cout << "Invalid gender input." << endl;

      }

   } else {

      cout << "You are a minor." << endl;

   }

   return 0;

}
```

Explanation

1. The **outer if condition** checks if the user is 18 or older.
2. If the condition is **true**, an **inner if-else structure** further checks the gender and displays an appropriate message.
3. If the outer condition is **false**, the program directly prints that the user is a minor.

4. Functions and Scope

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

ANS.A function in C++ is a block of code that performs a specific task. It helps in code reusability and modular programming.

Function Components:

1. **Function Declaration (Prototype)** – It tells the compiler about the function name, return type, and parameters before it is defined.
2. **Function Definition** – It contains the actual implementation of the function.
3. **Function Call** – It is used to execute the function in the program.

Explanation:

- The function add() is **declared** before main().
- The function is **defined** after main(), containing the logic.
- The function is **called** inside main() to execute and return the sum.

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

ANS. The scope of a variable refers to the region of the program where it is accessible.

Types of Scope:

1. **Local Scope:** Variables declared inside a function or block. They can be accessed only within that block.
2. **Global Scope:** Variables declared outside all functions. They can be accessed throughout the program.

Key Differences:

Feature	Local Scope	Global Scope
Declaration	Inside a function/block	Outside all functions
Accessibility	Limited to the block	Accessible throughout the program
Lifetime	Created when function is called, destroyed when it ends	Exists throughout program execution
Memory	Stored in stack	Stored in data segment

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

ANS. Recursion is a technique where a function calls itself to solve a problem.

```cpp
#include <iostream>
using namespace std;

int factorial(int n) {
    if (n == 0 || n == 1)  // Base Case
        return 1;
    else
        return n * factorial(n - 1);  // Recursive Call
}

int main() {
    int num = 5;
    cout << "Factorial of " << num << " is: " << factorial(num) << endl;
    return 0;
}
```

Explanation:

- The **base case** stops recursion when n == 0 or n == 1.
- The **recursive case** calls the function with n-1 until it reaches the base case.
- This leads to **factorial(5) → 5 × 4 × 3 × 2 × 1 = 120**.

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

ANS. A function prototype is a declaration of a function before its definition. It helps the compiler understand the function's name, return type, and parameters before its actual definition.

Why are Function Prototypes Used?

1. **Ensures Proper Function Calls:** The compiler knows the function's signature before encountering its definition.
2. **Enables Top-Down Compilation:** Functions can be called before their definition.
3. **Supports Separate Compilation:** Prototypes allow function definitions in separate files (header files).

5. Arrays and Strings

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

ANS. Arrays are collections of elements of the same data type stored in contiguous memory locations.

Arrays in C++:

- Single-dimensional: Stores elements in a single row.

code int arr[5] = {1, 2, 3, 4, 5};

- Multi-dimensional: Stores elements in a grid (rows and columns).

code int matrix[2][2] = {{1, 2}, {3, 4}};

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

ANS. String Handling in C++:

- Using char arrays:

char str[20] = "Hello"; cout << str;

- Using string class:

#include string s = "World"; cout << s;

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

ANS. Array Initialization:

- 1D Array:

int arr[5] = {1, 2, 3, 4, 5};

- 2D Array:

int matrix[2][2] = {{1, 2}, {3, 4}};

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

ANS.

- Length:

string s = "Hello"; cout << s.length();

- Concatenation:

string a = "Hi " , b = "there!"; cout << a + b;

- Substring:

string s = "HelloWorld"; cout << s.substr(0, 5); // Output: Hello

- Compare:

string s1 = "abc" , s2 = "xyz"; if (s1 == s2) cout << "Equal"; else cout << "Not Equal";

6. Introduction to Object-Oriented Programming

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

ANS. Key Concepts of OOP: Encapsulation: Wrapping data and methods into a single unit (class). Inheritance: Deriving new classes from existing ones. Polymorphism: Same function behaves differently based on context. Abstraction: Hiding complex details and showing only essentials.

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

ANS. Classes and Objects: Class: Blueprint for creating objects. Object: Instance of a class. code : class Car { public: string brand; void showBrand() { cout << brand; } }; Car myCar; myCar.brand = "Toyota"; myCar.showBrand();

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

ANS. Inheritance: One class acquires properties of another.

code:

```
class Animal {
 public:
 void sound() { cout << "Animal Sound";
}
 };
class Dog : public Animal {}; // Dog inherits Animal
Dog d;
d.sound(); // Inherited method
```

4. What is encapsulation in C++? How is it achieved in classes?

ANS. Encapsulation: Protects data by keeping it private and accessing it through public methods.

code :

```
  class BankAccount {
private:
 int balance;
public:
 void setBalance(int b) { balance = b; }
  int getBalance() { return balance; }
 };
```