Module 3
Introduction to OOPS Programming

Q-1. List and explain the main advantages of OOP over POP.

Ans:-

1. **Modularity**
    - In OOP, code is organized into classes/objects, making programs easier to manage and update compared to POP, where everything is function-based.
2. **Reusability**
    - OOP supports **inheritance**, so existing classes can be reused and extended.
    - POP requires rewriting code or copying functions.
3. **Encapsulation (Data Hiding)**
    - OOP allows data to be hidden inside objects and accessed only through methods, ensuring **better security**.
    - In POP, data is usually global and less protected.
4. **Abstraction**
    - OOP lets you focus on **what** an object does rather than **how** it does it (using abstract classes/interfaces).
    - POP mixes logic and data, making it harder to abstract details.
5. **Polymorphism (Flexibility)**
    - In OOP, the same function or operator can work differently depending on the object (method overloading/overriding).

- POP does not support this level of flexibility.

6. **Easier Maintenance**
   - OOP code is modular and reusable, so debugging and modifying are simpler.
   - POP code becomes lengthy and hard to maintain for large projects.
7. **Real-World Modeling**
   - OOP is based on objects, which represent real-world entities (like Student, Car, Employee).
   - POP focuses only on functions, which is less natural for complex systems.

Q- 2. What are the different data types available in C++? Explain with examples.

Ans:-

Basic (Fundamental) Data Types

1.int → used for integer values (whole numbers)

Example:

```
int age = 20;
```

2.float → used for decimal numbers (single precision)

Example:

```
float price = 99.5;
```

3.double → used for decimal numbers (double precision, more accurate)

Example:

```
double pi = 3.14159;
```

4.char → used to store a single character

Example:

```
char grade = 'A';
```

5.bool → used for logical values (true/false)

Example:

```
bool isPassed = true;
```

6.void → indicates no return value (commonly used in functions)

Example:

```
void display() { cout << "Hello"; }
```

Derived Data Types

1.Array → collection of data items of the same type

Example:

```
int marks[5] = {10, 20, 30, 40, 50};
```

2.Pointer → stores the memory address of another variable

Example:

```
int x = 10;
int* ptr = &x;
```

3.Function → a block of code that can be reused multiple times

Example:

```
int add(int a, int b) { return a + b; }
```

User-Defined Data Types

1.struct → groups different types of data together

Example:

```
struct Student {
int id;
char name[20];
};
```

2.class → OOP concept; encapsulates data and functions together

Example:

```
class Car
 {
        Public:
        string brand;
         int speed;
  };
```

Q-3 Explain the difference between implicit and explicit type conversion in C++.

Ans:-

Difference Between Implicit and Explicit Type Conversion in C++

- **Implicit Type Conversion:**

1. This conversion is done automatically by the compiler.
2. It happens when we assign one data type to another compatible type.
3. Usually converts from smaller to bigger type, like int to double.
4. No special syntax is needed.
5. It is safe in most cases, but sometimes can give unexpected results.

Example:

```
int a = 10;
double b = a;
```

- **Explicit Type Conversion:**

1. This conversion is done manually by the programmer.
2. It is used when we want to forcefully convert one type into another.
3. Can convert from bigger type to smaller type, like double to int.
4. Needs casting syntax like (type) or static_cast<type>().
5. It may cause data loss, like losing decimal part.

Example:

```
double x = 9.8;
int y = (int)x;
```

Q-4 How are break and continue statements used in loops? Provide examples.

Ans:-

- **Break Statement**

- **Purpose:** Used to **immediately stop the loop**, even if the loop condition is still true.
- When break is encountered, the program exits the loop and moves to the next statement after the loop.
- Mostly used when a specific condition is met and you want to end the loop early.

Example:
If you are searching for the first zero in a list of numbers, as soon as you find zero, you use break to stop the loop immediately.

- **Continue Statement**

- **Purpose:** Used to **skip the current iteration** of the loop but keep the loop running for the next iterations.
- When continue is encountered, the remaining code inside the loop for the current iteration is skipped, and control moves to the next iteration's condition check.
- Used when you want to ignore certain conditions but continue looping.

Example:
If you are printing numbers from 1 to 10 but want to skip odd numbers, whenever an odd number comes, use continue to skip that iteration and move to the next number.

Q-5 Explain nested control structures with an example.

Ans:-

Nested Control Structures in C++

- **Definition:** When one control structure (like if, for, while) is placed **inside another control structure**, it's called *nested*.
- This allows more complex decisions or repeated actions inside other decisions or loops.
- Commonly used to check multiple conditions or perform tasks that depend on multiple levels of logic.

Example Explanation (without code):

Imagine you want to check students' marks and print:

- If marks are above 90, print "Excellent"
- But only if attendance is also above 75%, print "Eligible for prize"
- Otherwise, print "Keep trying"

Q-6-Explain recursion in C++ with an example.

Ans:

What is Recursion in C++?

Recursion in C++ is a programming technique where a function calls itself directly or indirectly to solve a problem.

Recursion is used to solve problems that can be broken down into smaller sub-problems of the same type, such as calculating factorial, Fibonacci numbers, or solving puzzles like Tower of Hanoi.

Key Components of Recursion

1.Base Case:

The condition under which the recursion stops. It prevents infinite calls.

2.Recursive Case:

The part where the function calls itself to break the problem into smaller subproblems.

Example:

```
#include <iostream>

using namespace std;

int factorial(int n)

{

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

    {
```

```
        return 1;
    }
   else
    {
      return n * factorial(n - 1);
    }
int main()
 {
   int num = 5;
   cout << "Factorial of " << num << " is: " << factorial(num);
   return 0;
}
```

Q-7-What are arrays in C++? Explain the difference between single-dimensional and multi-dimensional arrays.

Ans:

What are Arrays in C++?

An array in C++ is a collection of fixed-size elements of the same data type, stored in contiguous memory locations.

Arrays allow you to store and access multiple values using a single variable name with an index.

🔁 Difference Between Single-Dimensional and Multi-Dimensional Arrays

Feature	Single-Dimensional Array	Multi-Dimensional Array
Structure	Linear (like a list)	Tabular or higher-dimensional
Indexing	One index (e.g., `arr[i]`)	Multiple indices (e.g., `arr[i][j]`)
Memory Layout	Sequential in one direction	Stored in row-major order (by default)
Use Case	Simple lists (e.g., names, scores)	Grids, matrices (e.g., tables, images)
Complexity	Easy to declare and access	More complex to manage and navigate

Q-8-Explain string handling in C++ with examples

Ans:

In C++, strings are used to store and manipulate sequences of characters. C++ provides two main ways to handle strings:

C-style strings (character arrays)

C++ string class (from the Standard Template Library - STL)

1. C-style Strings (Character Arrays)

Declaration:

char name[10] = "John";

Common Operations:

Using cin and cout

Using standard functions like strlen(), strcpy(), strcat(), strcmp() from <cstring>

Example:

#include <iostream>

```
#include <string>

using namespace std;

int main()

{

   char str1[20] = "Hello";

   char str2[20] = "World";


   strcat(str1, str2);

   cout << "Combined String: " << str1 << endl;

   cout << "Length: " << strlen(str1) << endl;

   return 0;

}
```

2. C++ Strings (std::string)

The string class in C++ is more powerful and easier to use than C-style strings. It comes from the <string> header.

Declaration:

```
#include <string>

string name = "Alice";
```

Common Operations:

Length: str.length()

Concatenation: str1 + str2

Substring: str.substr(start, length)

Comparison: ==, !=, <, >

Finding: str.find("text")

Insertion/Deletion: insert(), erase()

Example:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
   string str1 = "Hello";
   string str2 = "World";

   string combined = str1 + " " + str2;
   cout << "Combined String: " << combined << endl;
   cout << "Length: " << combined.length() << endl;
   cout << "Substring: " << combined.substr(0, 5) << endl;
   return 0;
}
```

Q-9 How are arrays initialized in C++? Provide examples of both 1D and 2D arrays.

Ans:-

Array Initialization in C++

In C++, arrays can be initialized at the time of declaration using curly braces {}. Arrays must be of a fixed size and contain elements of the same data type.

1. One-Dimensional Array (1D Array)

Syntax:

```
dataType arrayName[size] = {value1, value2, ..., valueN};
```

Examples:

a) Full Initialization

```
int numbers[5] = {10, 20, 30, 40, 50};

int numbers[] = {1, 2, 3};  // Size = 3
```

2. Two-Dimensional Array (2D Array)

Syntax:

```
dataType arrayName[rows][columns] = {
   {value1, value2, ...},
   {value1, value2, ...},
   ...
```

};

Examples:

 a) Full Initialization

```
int matrix[2][3] = {
   {1, 2, 3},
   {4, 5, 6}
};
```

Q-10 Explain the key concepts of Object-Oriented Programming (OOP).

Ans:-

Key Concepts of Object-Oriented Programming (OOP)

1. Class

Blueprint of an object.

Defines data (variables) and functions together.

```
class Car
 {
   public:
    string color;
    void drive();
};
```

2. Object

Instance of a class.

Used to access members of the class.

Car c1;

c1.drive();

3. Encapsulation

Wrapping data and functions into a single unit (class).

Helps in data protection.

4. Abstraction

Hiding complex details and showing only important features.

User sees only what is necessary.

5. Inheritance

One class can inherit properties of another class.

Promotes code reusability.

```
class Vehicle
{
  public: void start();
};
class Car : public Vehicle {};
```

6. Polymorphism

One function or object behaves in different ways.

Types:

Compile-time (Function Overloading)

Run-time (Function Overriding)

7. Constructor & Destructor

Constructor: Automatically called when object is created.

Destructor: Called when object is destroyed.

Q-11 What isinheritance in C++? Explain with an example.

Ans:-

Inheritance is one of the core features of Object-Oriented Programming (OOP). In C++, inheritance allows a class (derived class) to acquire the properties and behaviors (i.e., data members and member functions) of another class (base class). This promotes code reuse, extensibility, and supports polymorphism.

Types of Inheritance in C++

Single Inheritance – One derived class inherits from one base class.

Multiple Inheritance – One derived class inherits from more than one base class.

Multilevel Inheritance – A class is derived from a derived class.

Hierarchical Inheritance – Multiple derived classes inherit from a single base class.

Hybrid Inheritance – A combination of more than one type of inheritance.

Syntax of Inheritance

```
class Base
{
   // base class members
};

class Derived : access_specifier Base
{
   // derived class members
};
```

Q-12 What is encapsulation in C++? How isit achieved in classes?

Ans:-

Encapsulation is one of the fundamental concepts of Object-Oriented Programming (OOP). It refers to the bundling of data and functions that operate on that data within a single unit (class), and restricting direct access to some of the object's components.

- This helps in:

Protecting data from unauthorized access or modification

Making the code secure, modular, and maintainable

- **Encapsulation in C++ is achieved by:**

Using Classes – To group data members and functions together.

- **Access Specifiers – To control visibility:**

private – Members are accessible only inside the class.

protected – Members are accessible in the class and derived classes.

public – Members are accessible from anywhere.

Getters and Setters – Public methods used to access and update private data.

Example of Encapsulation in C++

```
#include <iostream>

using namespace std;


class Employee

 {

private:

   int salary;

public:

   // Setter function

   void setSalary(int s)

{

       if (s >= 0)

           salary = s;

       else

           cout << "Invalid salary!" << endl;

   }

   // Getter function
```

```
    int getSalary()
 {
        return salary;
    }
};
int main()
 {
    Employee emp;
    emp.setSalary(50000);            // Set salary using setter
    cout << emp.getSalary() << endl;  // Get salary using getter
    emp.setSalary(-1000);            // Invalid salary input
return 0;
}
```