Module 3– Introduction to OOPS Programming

1. Introduction to C++

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

Ans:

Procedural Programming

- Program divided into functions/procedures.
- Uses top-down approach.
- Data is usually global/shared.
- Focus on functions (logic).
- Less secure, no strict data hiding.
- Languages: C, Pascal, Fortran.

Object-Oriented Programming (OOP)

- Program divided into objects (data + methods).
- Uses bottom-up approach.
- Data is encapsulated inside objects.
- Focus on objects (real-world entities).
- More secure with encapsulation and access control.
- Supports inheritance & polymorphism (code reuse).
- Languages: Java, C++, Python, C#.

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

Ans:

1.Encapsulation (Data Security)

- OOP: Data is hidden inside objects and accessed only through methods → prevents accidental changes.
- POP: Data is often global → any function can modify it, less secure.

2.Reusability (Code Reuse)

- OOP: Supports inheritance, so existing classes can be reused and extended.
- POP: Code reuse only through functions, less flexible.

3. Modularity (Better Organization)

- OOP: Program is divided into objects (independent modules), making large projects easier to manage.
- POP: Divided into functions, but data and logic are often tangled.

4. Polymorphism (Flexibility)

- OOP: Same function/method can work in different ways depending on context (e.g., method overloading/overriding).
- POP: Doesn't support polymorphism, so less flexible.

5. Real-World Modeling

- OOP: Objects represent real-world entities → easy to understand and design.
- POP: Works as step-by-step instructions → harder to relate to real-world systems.

6. Maintainability (Easy Updates)

- OOP: Changes in one class/object usually don't affect others → easier to maintain.
- POP: Changes in global data may break many functions.

7. Scalability (Good for Large Projects)

- OOP: Best suited for large, complex applications (banking systems, games, etc.).
- POP: Works fine for small programs, but becomes difficult to manage as project grows.

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

Ans:

1. Encapsulation (Data Security)

- In OOP, data is hidden inside objects and accessed through methods. This prevents accidental or unauthorized changes, unlike POP where data is often global.

2. Code Reusability

- OOP supports inheritance, allowing classes to reuse and extend existing code. POP only reuses code through functions, which is less flexible.

3. Modularity (Better Organization)

- OOP divides programs into independent objects, making large projects easier to understand and manage. In POP, functions and data are mixed, reducing clarity.

4. Polymorphism (Flexibility)

- OOP allows the same function name to perform different tasks depending on the context (method overloading/overriding). POP does not support this flexibility.

5. Real-World Modeling

- OOP models real-world entities as objects, making systems more intuitive. POP follows a step-by-step approach, which is harder to relate to real-world problems.

6. Maintainability

- In OOP, changes in one object/class usually do not affect others, making maintenance easier. POP's use of global data makes programs harder to update.

7. Scalability

- OOP is highly suitable for large and complex applications (like banking systems, games, enterprise software). POP is better suited only for small programs.

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

Ans:

1. Input Operation

- Used to take data from the user.
- The cin (stands for *character input*) object is used for input.

Example:

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

int main() {
   int age;
   cout << "Enter your age: ";  // Output message
   cin >> age;              // Take input from user
   cout << "Your age is " << age; // Display input value
   return 0;
}
```

Explanation:

- cin reads input from the keyboard.
- >> is the extraction operator (used to take input).

2. Output Operation

- Used to display data on the screen.
- The cout (stands for *character output*) object is used for output.

Example:

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

int main() {
   string name = "Ved";
   cout << "Hello, " << name << "!";  // Display output
   return 0;
```

}

Explanation:

- cout sends data to the output screen.
- << is the insertion operator (used to print output).

3. Other Common I/O Functions

- getline() – reads a full line of text (including spaces).
- string fullName;
- cout << "Enter your full name: ";
- getline(cin, fullName);

cout << "Hello, " << fullName;

2. Variables, Data Types, and Operators'

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

Ans:

Primitive (Basic) Data Types
These are the fundamental built-in types.

- **int** → Stores integers.
- int age = 20;
- **float** → Stores single-precision decimal numbers.
- float pi = 3.14;
- **double** → Stores double-precision decimal numbers.
- double g = 9.81;
- **char** → Stores a single character.
- char grade = 'A';
- **bool** → Stores true or false.
- bool isPass = true;

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

Ans:

1. Implicit Type Conversion (Type Casting / Type Promotion)

- Also called Type Promotion or Type Casting by Compiler.
- Happens automatically when a smaller data type is converted into a larger data type.
- No data loss (in most cases).
- Controlled by the compiler, not the programmer.

2. Explicit Type Conversion (Type Casting by Programmer)

- Also called Type Casting.
- Done manually by the programmer using casting operators.
- **Syntax:** (type) expression
- May cause data loss if the target type is smaller.

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

Ans:

1. Arithmetic Operators

Used for mathematical operations.

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

2. Relational (Comparison) Operators

Used to compare two values, result is true or false.

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

3. Logical Operators

Used to combine conditions.

Operator	Example	Meaning
&&	(a > 0 && b > 0)	Logical AND
!	!(a > b)	Logical NOT

4. Assignment Operators

Used to assign values to variables.

Operator	Example	Equivalent To
=	x = 5	Assign 5 to x
+=	x += 2	x = x + 2
-=	x -= 3	x = x - 3
*=	x *= 4	x = x * 4
/=	x /= 2	x = x / 2
%=	x %= 3	x = x % 3

5. Increment and Decrement Operators

Used to increase/decrease a value by 1.

Operator	Example	Meaning
++	x++	Post-increment
--	--x	Pre-decrement

6. Bitwise Operators

Work at the bit level.

Operator	Example	Meaning
&	a & b	Bitwise AND
^	a ^ b	Bitwise XOR
~	~a	Bitwise NOT
<<	a << 1	Left shift
>>	a >> 1	Right shift

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

Ans:

1. Constants:

- **Definition:**
  Constants are variables whose values cannot be changed once they are assigned.
- **Purpose:**
  They are used when a fixed value is needed throughout the program — for example, mathematical values like π (pi) or configuration limits.
- **How to Declare Constants:**
  - Using the const keyword
  - const int MAX = 100;

(Here, MAX is a constant integer with a value of 100.)

  - Using the #define preprocessor directive
  - #define PI 3.14159

(This defines PI as a constant value 3.14159.)

2. Literals:

- **Definition:**
  Literals are the actual fixed values that appear directly in the code.
- **Examples:**
- int a = 10; // 10 is an integer literal
- char grade = 'A'; // 'A' is a character literal
- float pi = 3.14; // 3.14 is a floating-point literal

string name = "Ved"; // "Ved" is a string literal

3. Control Flow Statements

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

Ans:

1. Conditional Statements in C++

Conditional statements are used to make decisions in a program.
They allow the program to execute different code blocks based on whether a condition is true or false.

2. if-else Statement

Purpose:
Used to perform different actions depending on whether a condition is true or false.
Syntax:

```
if (condition) {
   // code executes if condition is true
}
else {
   // code executes if condition is false
}
```

Example:

```
int age = 18;

if (age >= 18) {
   cout << "You are an adult.";
} else {
   cout << "You are not an adult.";
}
```

Output:
You are an adult.

3. switch Statement

Purpose:
The switch statement is used when you want to compare a single variable against

multiple possible values.

Syntax:

```
switch (expression) {
   case value1:
      // code for value1
      break;
   case value2:
      // code for value2
      break;
   default:
      // code if no case matches
}
```

Example:

```
int day = 3;

switch (day) {
   case 1: cout << "Monday"; break;
   case 2: cout << "Tuesday"; break;
   case 3: cout << "Wednesday"; break;
   default: cout << "Invalid day";
}
```

Output:

Wednesday

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

Ans:

Feature	for Loop	while Loop	do-while Loop
Condition Checked	Before loop	Before loop	After loop
Use When	Number of iterations known	Iterations unknown	Must run at least once
Syntax Structure	Compact (init, condition, update in one line)	Initialization and update outside loop	Condition checked at end
Minimum Execution	0 times	0 times	1 time

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

Ans:

1. break Statement

Purpose:

The break statement is used to exit (terminate) the loop immediately, even if the condition is still true.

Use:

Often used when a certain condition is met and no further looping is needed.

Example:

```
for (int i = 1; i <= 10; i++) {
   if (i == 5)
      break;    // loop stops when i = 5
   cout << i << " ";
}
```

Output:

1 2 3 4

(The loop stops as soon as i becomes 5.)

2. continue Statement

Purpose:

The continue statement is used to skip the current iteration of the loop and move to the next one.

Use:

Useful when you want to skip certain values but continue looping.

Example:

```
for (int i = 1; i <= 5; i++) {
   if (i == 3)
      continue;   // skip when i = 3
   cout << i << " ";
}
```

Output:

1 2 4 5

(When i is 3, that iteration is skipped.)

4. Explain nested control structures with an example.

Ans:

Meaning:
When one control structure (like an if, for, or while statement) is placed inside another, it is called a nested control structure.
It means using a loop or decision statement within another loop or decision statement.

Example 1: Nested if-else

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

int main() {
   int age = 20;
   int marks = 85;

   if (age >= 18) {            // outer if
      if (marks >= 50) {       // inner if
         cout << "You are eligible for admission.";
      } else {
         cout << "You failed to meet the marks requirement.";
      }
   } else {
      cout << "You are not old enough to apply.";
   }

   return 0;
}
```

Output:
You are eligible for admission.

4.Functions and Scope

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

Ans:

What is a Function in C++?

A function in C++ is a block of code that performs a specific task.
It helps to divide a program into smaller, reusable parts, making the code easier to read, debug, and manage.

➢ **Purpose of a Function:**

- To avoid repeating code
- To organize the program logically
- To **reuse** code whenever needed

➢ **Types of Functions**

1. **Built-in functions** → Provided by C++ (e.g., sqrt(), pow(), cout)
2. **User-defined functions** → Created by the programmer

1. Function Declaration (Prototype)

It tells the compiler about the function name, return type, and parameters — before it is used.

Syntax:

```
returnType functionName(parameter1, parameter2, ...);
```

Example:

```
int add(int a, int b);   // function declaration
```

2. Function Definition

It contains the actual code (body) that runs when the function is called.

Syntax:

```
returnType functionName(parameter1, parameter2, ...) {
   // function body
   return value;  // if needed
}
```

Example:

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

3. Function Calling

When you use the function in your program to execute it.

Syntax:

```
functionName(arguments);
```

Example:

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

int add(int a, int b);   // declaration

int main() {
   int sum = add(5, 10);   // function call
   cout << "Sum = " << sum;
   return 0;
}

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

Output:

Sum = 15

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

Ans:

Scope of Variables in C++

The scope of a variable in C++ refers to the region or part of the program where that variable can be accessed or used.

In other words, it defines the lifetime and visibility of a variable.

There are mainly two types of scopes in C++:

1. **Local Scope**
2. **Global Scope**

1. Local Scope

- A local variable is declared inside a function, block, or loop.
- It can only be accessed within that block where it is declared.
- Once the block ends, the variable is destroyed and cannot be used anymore.

Example:

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

void show() {
   int x = 10;   // Local variable
   cout << "Local x = " << x << endl;
}

int main() {
   show();
   // cout << x;  // Error: x is not accessible here
   return 0;
}
```

Explanation:

x is local to the function show(). It cannot be used outside that function.

2. Global Scope

- A global variable is declared outside all functions.
- It can be accessed from any function in the program.
- It exists throughout the execution of the program.

Example:

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

int x = 20;   // Global variable

void display() {
   cout << "Global x = " << x << endl;
}
int main() {
   cout << "Global x in main = " << x << endl;
   display();
   return 0;
}
```

Explanation:

x is a global variable, so it is accessible both in main() and display().

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

Recursion means a function calling itself again and again until a certain condition is met.
It keeps doing the same work with a smaller value each time.
Simple Example: Counting Down Numbers

Program:

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

void countDown(int n) {
   if (n == 0) {          // Base case (stop when n becomes 0)
      cout << "Time's up!";
      return;
   }
   cout << n << " ";       // Print the current number
   countDown(n - 1);       // Call the same function with smaller value
}

int main() {
   countDown(5);          // Start counting from 5
   return 0;
}
```

Output:

5 4 3 2 1 Time's up!

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

Ans:

A function prototype tells the compiler about a function before it is used — its name, return type, and parameters.

Syntax:

return_type function_name(parameter_list);

Example:

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

void greet();   // Function prototype

int main() {
   greet();    // Function call
}

void greet() {  // Function definition
   cout << "Hello!";
}
```

5. Arrays and Strings

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

ANS:

array is a collection of elements of the same data type stored in continuous memory locations. It is used to store multiple values under a single name, and each element can be accessed using an index. The index of the first element is always 0. Arrays help make the code simple and easy to manage when dealing with many values of the same type.

Example:

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

Here, numbers is an array of 5 integers. We can access each value by its position, like numbers[0] will give 10, and numbers[3] will give 40.

Difference between Single-Dimensional and Multi-Dimensional Arrays:

- **Single-Dimensional Array:**
  This type of array stores data in a single row. It uses only one index to access elements.
  Example:

  ```
  int marks[4] = {85, 90, 78, 92};
  ```

- **Multi-Dimensional Array:**
  This type of array stores data in more than one dimension, usually in rows and columns, like a table. It uses more than one index to access elements.
  Example:

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

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

ANS:

strings are used to store text. A string is a sequence of characters. There are two main ways to handle strings in C++:

1. C-style strings — arrays of characters ending with a special null character ('\0').
2. C++ string class — part of the Standard Template Library (STL) that makes string handling easier.

---

1. C-Style Strings

C-style strings are stored as a character array.

Example:

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

int main() {
   char name[20] = "Ved Jani";
   cout << "Name: " << name;
   return 0;
}
```

Here, name is a character array storing the text "Ved Jani". The last character '\0' marks the end of the string.

2. C++ String Class

The C++ string class is part of the <string> library. It makes string operations easier, such as finding length, concatenation, comparison, etc.

Example:

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

int main() {
   string name = "Ved";
   string surname = " Jani";
   string fullName = name + surname;

   cout << "Full Name: " << fullName << endl;
   cout << "Length: " << fullName.length();
   return 0;
}
```

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

ANS;

arrays can be initialized when they are declared. Initialization means giving values to the elements of the array.

1. One-Dimensional (1D) Array Initialization

A 1D array stores values in a single row.

Example:

```
#include <iostream>
using namespace std;
int main() {
   int numbers[5] = {10, 20, 30, 40, 50}; // Initialization at declaration

   // Accessing array elements
   for (int i = 0; i < 5; i++) {
      cout << "numbers[" << i << "] = " << numbers[i] << endl;
   }
   return 0;
}
```

Here:

- numbers[5] means an array of size 5.
- Values are stored directly in the array during declaration.

2. Two-Dimensional (2D) Array Initialization

A 2D array stores values in a table format (rows and columns).

Example:

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

int main() {
   int matrix[2][3] = { {1, 2, 3}, {4, 5, 6} }; // Initialization

   // Accessing array elements
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 3; j++) {
         cout << "matrix[" << i << "][" << j << "] = " << matrix[i][j] << endl;
      }
   }
   return 0;}
```

Here:

- matrix[2][3] means 2 rows and 3 columns.
- Values are provided row by row in braces {}.

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

ANS:

1 Creating a String:

string name = "Ved";
This creates a string variable named name.

2 Reading a String:

```
string str;
cout << "Enter a string: ";
getline(cin, str);
```

getline() allows you to take input with spaces.

3 Concatenation (Joining Strings):

```
string a = "Hello ";
string b = "World";
string c = a + b;   // Result: "Hello World"
```

4 Finding Length of String:

```
string s = "C++";
cout << s.length();  // Output: 3
```

5 Accessing Characters:

```
string s = "ChatGPT";
cout << s[0];  // Output: C
```

Common String Functions

Function	Description	Example
s.length()	Returns number of characters	s.length();
s.empty()	Checks if string is empty	if(s.empty())
s.append("text")	Adds text to end	s.append(" Code");
s.insert(pos, "word")	Inserts text at position	s.insert(3, "AI");
s.erase(pos, len)	Removes part of string	s.erase(2, 3);
s.substr(pos, len)	Returns substring	s.substr(0, 4);
s.find("word")	Finds position of word	s.find("GPT");
s.compare(str2)	Compares two strings	s.compare("Hello");

6.Introduction to Object-Oriented Programming

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

Ans:

1. Class and Object

- Class is a blueprint or template that defines how an object will behave.
- Object is an instance of a class that represents a real-world entity.

Example:

```
class Car {
public:
   string brand;
   void show() {
      cout << "Brand: " << brand;
   }
};

int main() {
   Car c1;
   c1.brand = "Tesla";
   c1.show();
   return 0;
}
```

2. Encapsulation

It means hiding data inside a class and allowing access only through functions.
This protects data from being directly modified.

Example:

```
class Student {
private:
   int marks;
public:
   void setMarks(int m) { marks = m; }
   int getMarks() { return marks; }
};
```

3. Inheritance

It allows one class to inherit properties and methods from another class.
This helps in code reusability**.**

Example:

```
class Animal {
public:
  void eat() { cout << "Eating..."; }
};
class Dog : public Animal {
public:
  void bark() { cout << " Barking..."; }
};
```

4. Polymorphism

It means many forms — the same function can behave differently depending on the object.

It is done using function overloading and overriding.

Example:

```
class Shape {
public:
  void draw() { cout << "Drawing Shape\n"; }
};
class Circle : public Shape {
public:
  void draw() { cout << "Drawing Circle\n"; }
};
```

5. Abstraction

It means showing only important details and hiding unnecessary information.

This makes complex systems easier to use.

Example:

```
class Car {
public:
  void start() { cout << "Car Started"; }
};
```

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

Ans:

In C++, a class is a blueprint or template that defines the properties (data) and behaviors (functions) of an object.
An object is an instance of a class — it represents a real-world entity like a car, student, or mobile phone.

Example:

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

class Car {      // Class declaration
public:
  string brand;
  int speed;

  void showDetails() {
    cout << "Brand: " << brand << endl;
    cout << "Speed: " << speed << " km/h" << endl;
  }
};

int main() {
  Car c1;          // Object creation
  c1.brand = "Tesla";
  c1.speed = 200;
  c1.showDetails();   // Calling function

  return 0;
}
```

Explanation:

- **Class:** Car defines data members (brand, speed) and a function (showDetails()).
- **Object:** c1 is an object of the class Car. It stores and accesses the data using the class's structure.

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

Ans:

Inheritance is an important feature of Object-Oriented Programming that allows one class to use the properties and functions of another class.
It helps in code reusability and makes programs easier to maintain.
The class that gives its properties is called the base class **(**parent**)**, and the class that receives them is called the derived class **(**child**)**.

Types of Inheritance:

1. **Single Inheritance** – One class inherits from another.
2. **Multiple Inheritance** – A class inherits from more than one class.
3. **Multilevel Inheritance** – A class inherits from a derived class.
4. **Hierarchical Inheritance** – Many classes inherit from one base class.
5. **Hybrid Inheritance** – A combination of different types.

Example:

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

class Animal {      // Base class
public:
  void eat() {
    cout << "Eating..." << endl;
  }
};

class Dog : public Animal {   // Derived class
public:
  void bark() {
    cout << "Barking..." << endl;
  }
};

int main() {
  Dog d1;
  d1.eat();    // Inherited function
  d1.bark();   // Function of derived class
  return 0;
}
```

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

Ans:

Encapsulation is one of the main principles of Object-Oriented Programming.
It means binding data and functions into a single unit (class) and hiding the internal details of how things work.
This helps to protect data from being changed accidentally.

How Encapsulation is Achieved:

1. **By using classes** – Data (variables) and functions (methods) are written inside a class.
2. **By using access specifiers** – private, public, and protected keywords control access to data.
    - **Private:** Data cannot be accessed directly from outside the class.
    - **Public:** Data or functions can be accessed from outside the class.

Example:

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

class Student {
private:
  int marks;    // private data (hidden from outside)

public:
  void setMarks(int m) {   // public function to set data
    marks = m;
  }

  int getMarks() {       // public function to access data
    return marks;
  }
};

int main() {
  Student s1;
  s1.setMarks(90);       // setting value using function
  cout << "Marks: " << s1.getMarks();
  return 0;
}
```

Thank you 😁