# MODULE – 3

**Introduction to OOPS Programming**

1. **What are the key differences between Procedural Programming and ObjectOrientedProgramming (OOP)?**

* **Procedural Programming (POP**) is a programming approach where the program is divided into functions (procedures). The focus is mainly on how to perform a task (step-by-step instructions). In POP, data and functions are treated separately, so data is less secure. Example: C language.
* **Object-Oriented Programming (OOP)**, on the other hand, is based on **objects and classes**. Here, the focus is on **what entities (objects) are** and **how they interact**. Data and functions are combined inside objects, which makes data **secure and reusable**. OOP supports important concepts like **Encapsulation, Inheritance, and Polymorphism**. Example: **C++, Java**

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

**Advantages of OOP over POP**

-> Object-Oriented Programming (OOP) has several advantages compared to Procedural Oriented Programming (POP):

1. Data Security (Encapsulation)

In OOP, data and functions are combined inside objects, so data is protected from direct access.

2. Reusability (Inheritance)

Existing classes can be reused in new programs by extending them. This reduces code duplication.

3. Modularity

Program is divided into objects, each object represents a real-world entity, so program is easy to understand and manage.

4. Extensibility

New features can be easily added without changing existing code.

5. Maintainability

Due to modular and reusable nature, OOP programs are easy to modify and maintain.

6. Better Problem Solving (Real-world mapping)

OOP is based on objects, which are similar to real-world entities, so it matches real-life problems better than POP.

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

-> Steps to Set Up a C++ Development Environment.

**1. Install a Compiler**

C++ is a compiled language, so first we need a **compiler** (like GCC/g++ or MSVC).

**2. Install an IDE or Text Editor**

We need a software to write and edit C++ code.  
 Options:

* IDE (Integrated Development Environment) like **Code::Blocks, Dev C++, Visual Studio, CLion**.
* Text Editors like **VS Code, Sublime Text, Notepad++** (with compiler integration).

**3. Configure Compiler with IDE**

In IDEs like Code::Blocks/Dev C++, compiler comes pre-installed.  
In VS Code, you must **link MinGW compiler** in settings.

**4. Write a Simple Program**

Create a new C++ file (e.g., hello.cpp) and type a basic program:

#include <iostream>

using namespace std;

int main() {

cout << "Hello, World!";

return 0;

}

**5. Compile the Program**

Use compiler (g++ hello.cpp -o hello) or IDE’s **Build/Run** button.

**6. Run the Program**

After successful compilation, run the executable file and check output.

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

* main input/output operations in C++

In C++, input/output is handled using the **iostream** library (#include<iostream>). The two most common operators are:

1. **Output (Display Something on screen)**

Using **cout** (Character Output).

**<<** is Calles the insertion operator (it inserts data into the output stream).

Example:

#include<iostream>

using namespace std;

int main(){

cout <<"Hello World !"<<endl;

cout<<"My Age Is :"<<16 <<endl;

return 0

}

**Output is**

Hello World !

My age is :16

**Input (Taking Data User)**

Using **cin** (Character Input).

**>>** is called the extraction operator (it take data from user).

**Example:**

#include<iostream>

using namespace std;

int main(){

string name;

int age;

cout<< "Enter Your Name :";

cin>>name;

cout<< "Enter Your Age :";

cin>>age;

cout<< "Your Name is :"<<name<<endl<<"Your Age is :"<<age;

return 0;

}

Input is:

Chirag

16

Output is:

Your Name is : Chirag

Your Age is : 16

1. **Using getline() for Full Ling Input.**

Want to take **input with spaces** (like full name)

Example :

#include<iostream>

#include<string>

using namespace std;

int main(){

string fullName;

cout<< "Enter Your Full Name :";

getline(cin, fullName);

cout<< "Hello "<<fullName<< " ! "<<endl;

return 0;

}

Input is:

Gopal Bawaskar

Output is:

Hello Gopal Bawaskar !

1. **Variable, Data Types, and Operators.**
2. **What are the different data types available in C++? Explain with examples.**

In C++, Data Types define the type of data a variable can hold.

1. **Primary (Basic) Data Types.**

These are the fundamental data types.

1. **int –** Stores integers (Whole numbers).

int age = 21;

Example: 21, -10, 1000

1. **float –** Stores decimal numbers (single precision).

float price = 90.50;

Example: 3.14, -2.5

1. **double –** Stores decimal numbers (double precision, more accurate than float).

Double pi = 3.1415926535;

1. **char –** Store a single character.

char grade = ‘A’;

1. **bool -**  Store a True or False.

bool isPass = true;

1. **void –** Means “no value”, Used with Function.

void display() {

cout << "Hello!";

}

1. **Derived Data Types**

These are created from basic data types.  
1) **Array –** Collection of same types of data.  
int marks [5] = {90, 85, 88, 92, 95};

1. **Pointer** – Stores memory address of another variable.

int x = 10;

int \*ptr = &x; // pointer stores address of x

1. **Function** - Block of code that performs a task.

int add(int add(int a, int b){

return a + b;

1. **User – Defined Data Types**

Defined by programmer.

1. **struct** – Collection of different data types.

struct Student {

int rollNo;

char name[20];

float marks;

};

1. **class –** Ised in Object – Oriented Programming (OOP).

**class Car {**

**public:**

**string brand;**

**int speed;**

**};**

1. **enum –** Collection of named constants.

enum Week {Mon, Tue, Wed, Thu, Fri, Sat, Sun};

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

**Type Conversion in C++**

When we change the **data type** of a variable/value into another type, it is called **type conversion**.  
There are two main types: **Implicit (Type Casting by Compiler)** and **Explicit (Type Casting by Programmer).**

**1. Implicit Type Conversion (Type Casting by Compiler)**

* Also called **Type Casting / Type Promotion**.
* Happens **automatically** by the C++ compiler.
* The compiler **converts smaller data type → larger data type** to avoid data loss.
* No special syntax is required.

**Example:**

#include <iostream>

using namespace std;

int main() {

int a = 10;

float b = 5.5;

float result = a + b; // 'a' (int) is converted into float automatically

cout << "Result = " << result;

return 0;

}

Here, a (int) is automatically converted into float.

**2. Explicit Type Conversion (Type Casting by Programmer)**

* Also called **Type Casting by Programmer**.
* Done **manually** by the programmer using **type casting operators**.
* We decide which type to convert into, even if there is a chance of **data loss**.

**Example:**

#include <iostream>

using namespace std;

int main() {

float pi = 3.14159;

int x = (int)pi; // Explicit conversion (float → int)

cout << "Value of x = " << x;

return 0;

}

**Simple memory tip:**

* **Implicit** = “Compiler” (Automatic)
* **Explicit** = “Programmer” (Manual)

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

**Operators in C++**

-Operators are special symbols used to perform operations on variables and values.

**-Types of Operators in C++**

**1. Arithmetic Operators**

Used for mathematical calculations.  
**Examples:**

int a = 10, b = 3;

cout << a + b; // Addition (10 + 3 = 13)

cout << a - b; // Subtraction (7)

cout << a \* b; // Multiplication (30)

cout << a / b; // Division (3) → int division

cout << a % b; // Modulus (remainder = 1)

**2. Relational Operators**

Used to compare two values, result is **true(1) / false(0)**.  
**Examples:**

int x = 5, y = 10;

cout << (x == y); // Equal to (false)

cout << (x != y); // Not equal (true)

cout << (x > y); // Greater than (false)

cout << (x < y); // Less than (true)

cout << (x >= y); // Greater or equal (false)

cout << (x <= y); // Less or equal (true)

**3. Logical Operators:**

Used in conditions (AND, OR, NOT).  
**Examples:**

int p = 1, q = 0;

cout << (p && q); // AND (0)

cout << (p || q); // OR (1)

cout << (!p); // NOT (0)

**4. Assignment Operators**

Used to assign values.  
**Examples:**

int a = 5;

a += 3; // same as a = a + 3 → 8

a -= 2; // same as a = a - 2 → 6

a \*= 4; // same as a = a \* 4 → 24

a /= 6; // same as a = a / 6 → 4

a %= 3; // same as a = a % 3 → 1

**5. Increment & Decrement Operators**

**Examples:**

int n = 5;

cout << ++n; // Pre-increment → 6

cout << n++; // Post-increment → 6 (then n=7)

cout << --n; // Pre-decrement → 6

cout << n--; // Post-decrement → 6 (then n=5)

**6. Bitwise Operators**

Work on bits (0,1).  
**Examples:**

int a = 5, b = 3;

// 5 = 0101, 3 = 0011

cout << (a & b); // AND (1)

cout << (a | b); // OR (7)

cout << (a ^ b); // XOR (6)

cout << (~a); // NOT (-6)

cout << (a << 1); // Left shift (10)

cout << (a >> 1); // Right shift (2)

**7. Conditional / Ternary Operator**

**Example:**

int age = 20;

string result = (age >= 18) ? "Adult" : "Minor";

cout << result; // Adult

**8. Special Operators**

* **sizeof** → size of data type/variable
* **typeid** → type info
* **comma ( , )** → multiple expressions
* **pointer (\*) & reference (&)**

**Example:**

int x = 10;

cout << sizeof(x); // 4 (on most systems)

cout << &x; // prints address of xTop of Form

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

**Constants and Literals in C++**

**1. Constants in C++**

A **constant** is a value that **cannot be changed** during program execution.

**Purpose of Constants :**

* To make program **more readable** and meaningful.
* To avoid accidental value changes.
* To improve program **security and reliability**.

**Ways to Define Constants :**

**(a) Using const keyword :**

#include <iostream>

using namespace std;

int main() {

const float PI = 3.14159; // constant variable

cout << "PI = " << PI;

// PI = 3.14; error (cannot change value)

return 0;

}

**(b) Using #define preprocessor**

#include <iostream>

#define MAX 100

using namespace std;

int main() {

cout << "Maximum = " << MAX;

return 0;

}

**2. Literals in C++**

**Literals are fixed values** which are directly written in the program.  
 Example: numbers (10), characters ('A'), strings ("Hello"), etc.

**Types of Literals :**

1. **Integer Literals:**
   * Example: 10, -45, 0
2. **Floating-point Literals:**
   * Example: 3.14, -0.99, 2.5e3
3. **Character Literals:**
   * Example: 'A', 'b', '9'
4. **String Literals :**
   * Example: "Hello", "C++", "123"
5. **Boolean Literals (true/false)**
   * Example: true, false
6. **Escape Sequence Literals** (special meaning)
   * Example: \n (new line), \t (tab), \\ (backslash), \" (double quote).

**Key Difference between Constants & Literals**

| **Feature** | **Constants** | **Literals** |
| --- | --- | --- |
| **Definition** | Named fixed value | Actual fixed value |
| **Example** | const int max = 100; | 100 itself |
| **Purpose** | To reuse values with a name | Direct values in program |
| **Changeable?** | Not changeable | Not changeable |

**Short Memory Tip**

* **Constant = Named Fixed Value (PI, MAX, RATE)**
* **Literal = Actual Fixed Value (3.14, 100, 'A')**Bottom of Form

**3.Control Flow Statement:**

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

Conditional statements allow the program to make decisions based on conditions.

**(a) if-else statement**

* Syntax:

if (condition) {

// code if condition is true

} else {

// code if condition is false

}

* Example:

int age = 20;

if (age >= 18) {

cout << "You are eligible to vote.";

} else {

cout << "You are not eligible to vote.";

}

**(b) switch statement**

* Syntax:

switch (expression) {

case value1:

// code

break;

case value2:

// code

break;

default:

// code

}

* 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";

}

**2. Difference between for, while, and do-while loops**

**(a) for loop**

Used when number of iterations is known.

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

cout << i << " ";

}

**(b) while loop**

Used when condition is checked **before** execution.

int i = 1;

while (i <= 5) {

cout << i << " ";

i++;

}

**(c) do-while loop**

Used when loop must run **at least once**.

int i = 1;

do {

cout << i << " ";

i++;

} while (i <= 5);

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

**(a) break statement**

Stops the loop immediately.

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

if (i == 3) break;

cout << i << " ";

}

// Output: 1 2

**(b) continue statement**

Skips current iteration and moves to the next.

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.**

When one control structure is placed inside another (like loop inside loop, or if inside loop).

**Example: Multiplication Table using Nested for loop**

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

for (int j = 1; j <= 3; j++) {

cout << i << " x " << j << " = " << i \* j << endl;

}

cout << endl;

}

**4.Function And Scope**

**1. What is a function in C++?**

A function is a block of code that performs a specific task and can be reused.

**Parts of a function**

* **Declaration (Prototype):** tells compiler about function name, return type, and parameters.
* **Definition:** actual body of the function (what it does).
* **Calling:** using the function in main() or another function.

**Example:**

#include <iostream>

using namespace std;

int add(int, int); // Declaration

int main() {

int result = add(5, 10); // Calling

cout << "Sum = " << result;

return 0;

}

int add(int a, int b) { // Definition

return a + b;

}

**2. Scope of variables in C++**

Scope means where a variable can be accessed.

**Local Scope:**

Declared inside a function or block. Accessible only there.

void test() {

int x = 10; // local variable

}

**Global Scope:**

Declared outside all functions. Accessible everywhere.

int g = 50; // global variable

int main() {

cout << g; // accessible

}

**3. Recursion in C++**

Recursion means a function calling itself. Used in problems like factorial, Fibonacci, etc.

**Example: Factorial**

#include <iostream>

using namespace std;

int factorial(int n) {

if (n == 0) return 1; // base case

return n \* factorial(n-1); // recursive call

}

int main() {

cout << "Factorial of 5 = " << factorial(5);

return 0;

}

**4. Function Prototypes in C++**

A function prototype tells the compiler about the function **before its use**.  
It includes function name, return type, and parameter list, but **not the body**.

**Example:**

#include <iostream>

using namespace std;

int add(int, int); // Function prototype

int main() {

cout << add(3, 7);

return 0;

}

int add(int a, int b) { // Function definition

return a + b;

}Top of Form

**5.Arrays and Strings**

**1. What are Arrays in C++?**

**Definition:**  
An **array** is a collection of elements of the **same data type**, stored in **contiguous memory locations**.

* Each element is accessed using an **index**.
* Indexing in C++ starts from **0**.

Example:

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

cout << marks[0]; // prints 10

**Single-Dimensional Array**

* It is a **list** of elements.
* Like → [10, 20, 30, 40].

Example:

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

**Multi-Dimensional Array**

* It is like a **table (matrix)** of rows and columns.
* Example:

2D array = rows × columns.

Example:

int matrix[2][3] = {

{1, 2, 3},

{4, 5, 6}

};

**2. String Handling in C++**

String = collection of characters.  
C++ ma **2 types** na string handle karva na ways che:

1. **C-style strings** (character arrays):

char name[10] = "Gopu";

cout << name;

1. **C++ string class (modern way)**:

#include<iostream>

#include <string>

using namespace std;

int main() {

string name = "Gopu";

cout << "Hello " << name;

}

String handling operations (C++ string class sathe easy che):

* name.length() → length
* name.append(" Bawaskar") → add
* name.substr(0, 2) → substring

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

**Array Initialization in C++**

**Initialization** means giving values to an array at the time of declaration.  
Arrays can be **1D (single row/line)** or **2D (rows & columns)**.

**1. One-Dimensional (1D) Array**

Syntax:

datatype array\_name[size] = {value1, value2, ...};

Examples:

#include <iostream>

using namespace std;

int main() {

// 1D Array Initialization

int arr1[5] = {1, 2, 3, 4, 5}; // full initialization

int arr2[5] = {10, 20}; // remaining become 0

int arr3[] = {7, 8, 9}; // size automatically = 3

cout << "1D Arrays:" << endl;

cout << "arr1: ";

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

cout << arr1[i] << " ";

}

cout << endl;

cout << "arr2: ";

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

cout << arr2[i] << " ";

}

cout << endl;

cout << "arr3: ";

for (int i = 0; i < 3; i++) {

cout << arr3[i] << " ";

}

cout << endl;

return 0;

}

**2. Two-Dimensional (2D) Array**

Syntax:

datatype array\_name[rows][cols] = { {row1}, {row2}, ... };

Examples:

#include <iostream>

using namespace std;

int main() {

// 2D Array Initialization

int matrix[2][3] = {

{1, 2, 3},

{4, 5, 6}

};

int matrix2[2][3] = {1, 2, 3, 4, 5, 6}; // flat initialization

int matrix3[2][3] = { {1, 2}, {4} }; // partial initialization

cout << "2D Arrays:" << endl;

cout << "matrix:" << endl;

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 3; j++) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

cout << "matrix2:" << endl;

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 3; j++) {

cout << matrix2[i][j] << " ";

}

cout << endl;

}

cout << "matrix3:" << endl;

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 3; j++) {

cout << matrix3[i][j] << " ";

}

cout << endl;

}

return 0;

}

**Key Points to Remember:**

* 1D = line of elements.
* 2D = table (rows × columns).
* If fewer values are provided → the rest are automatically set to 0.

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

**1. String in C++**

In C++, strings can be handled in two ways:

1. **C-style strings** → using char arrays.
2. **C++ string class** → using <string> library (modern and easy).

**2. Common String Operations in C++ (using string class)**

**a. Input and Output**

#include <iostream>

#include <string>

using namespace std;

int main() {

string name;

cout << "Enter your name: ";

getline(cin, name); // takes full line input

cout << "Hello " << name << endl;

}

**b. Length of String**

string str = "Gopu";

cout << "Length: " << str.length();

// Output: 4

**c. Concatenation (Joining Strings)**

string first = "Gopu";

string last = " Bawaskar";

string full = first + last; // using +

cout << full;

OR

first.append(last);

cout << first;

**d. Substring**

string word = "Programming";

cout << word.substr(0, 4); // "Prog"

cout << word.substr(4, 7); // "ramming"

**e. Comparison**

string a = "Apple";

string b = "Banana";

if (a == b)

cout << "Equal";

else if (a < b)

cout << "Apple comes before Banana";

else

cout << "Banana comes before Apple";

**f. Modify / Replace**

string str = "Hello Gopu";

str.replace(6, 4, "World"); // replace from index 6, 4 letters

cout << str;

// Output: Hello World

**g. Find (Search inside string)**

string str = "Hello World";

cout << str.find("World"); // Output: 6 (starting index)

**h. Erase**

string str = "Programming";

str.erase(3, 4); // remove 4 characters from index 3

cout << str; // Output: "Proaming"

**3. Functions Summary Table**

| **Function** | **Example** | **Output** |
| --- | --- | --- |
| length() | "Hello".length() | 5 |
| append() | "Hi".append(" Gopu") | Hi Gopu |
| substr(pos,n) | "Gopal".substr(0,3) | Gop |
| find() | "Hello".find("lo") | 3 |
| erase() | "World".erase(1,2) | Wld |
| replace() | "Hi Gopu".replace(3,4,"All") | Hi All |

**In Short:**

* String operations in C++ are **easy** because of the string class.
* You can **input, join, compare, search, erase, replace** using built-in functions.

Top of Form

**6. Introduction to Object-Oriented Programming**

**1. Key Concepts of OOP (Object-Oriented Programming)**

OOP divides programming into **real-world objects**.  
The main 4 concepts are:

1. **Encapsulation** → Keep data and functions together in a class.
2. **Abstraction** → Hide unnecessary details; provide only required functionality.
3. **Inheritance** → One class can use properties and functions of another class (reusability).
4. **Polymorphism** → Same function name or operator behaves differently depending on context.

Why use OOP?

* Code becomes **reusable, secure, and easy to maintain**.

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

* **Class** → Blueprint or template containing variables (data members) and functions (methods).
* **Object** → Real instance of a class.

**Example:**

#include <iostream>

using namespace std;

class Car {

public:

string brand;

int speed;

void showDetails() {

cout << "Brand: " << brand << ", Speed: " << speed << " km/h" << endl;

}

};

int main() {

Car c1; // Object creation

c1.brand = "BMW";

c1.speed = 220;

c1.showDetails(); // Function call

return 0;

}

**Output:**

Brand: BMW, Speed: 220 km/h

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

Inheritance allows a **child (derived) class** to use properties and functions of a **parent (base) class**.

**Example:**

#include <iostream>

using namespace std;

// Base class

class Animal {

public:

void eat() {

cout << "This animal eats food." << endl;

}

};

// Derived class

class Dog : public Animal {

public:

void bark() {

cout << "Dog barks!" << endl;

}

};

int main() {

Dog d1;

d1.eat(); // Inherited from Animal

d1.bark(); // Own function

return 0;

}

**Output:**

This animal eats food.

Dog barks!

**4. Encapsulation in C++? How is it achived in classes?**

Encapsulation means **keeping data members private** and providing **public functions (getters/setters)** to access them.  
 It ensures data is **secure** and cannot be accessed directly from outside the class.

**Example:**

#include <iostream>

using namespace std;

class BankAccount {

private:

double balance; // Private data

public:

BankAccount() { balance = 0; }

void deposit(double amount) {

balance += amount;

}

void withdraw(double amount) {

if (amount <= balance)

balance -= amount;

else

cout << "Insufficient balance!" << endl;

}

double getBalance() {

return balance;

}

};

int main() {

BankAccount acc;

acc.deposit(1000);

acc.withdraw(300);

cout << "Balance: " << acc.getBalance();

return 0;

}

**Output:**

Balance: 700

**Quick Summary**

* **OOP Concepts** → Encapsulation, Abstraction, Inheritance, Polymorphism
* **Class = blueprint**, **Object = real instance**
* **Inheritance** → Child class uses parent class properties
* **Encapsulation** → Data is hidden and accessed securely through functions.Bottom of Form

Bottom of Form