**Module 3 – Introduction to OOPS programming**

1. **What is the key difference between procedural language and object-oriented programming language.**

| * **Feature** | * **Procedural Programming** | * **Object-Oriented Programming** |
| --- | --- | --- |
| * Approach | * Follows a top-down approach | * Follows a bottom-up approach |
| * Focus | * Focuses on functions and procedures (step-by-step instructions) | * Focuses on objects and classes (real-world entities) |
| * Data Handling | * Data is global and can be accessed by any function | * Data is encapsulated within objects (data hiding) |
| * Modularity | * Less modular; code reuse is limited | * Highly modular; supports code reuse through inheritance |
| * Security | * Less secure, since global data can be modified anywhere | * More secure due to encapsulation and access modifiers |

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

* Advantages are below:
* **Modularity** - Code is organized into classes and objects for better structure.
* **Encapsulation** - Data is hidden within objects to protect it from outside interference.
* **Reusability (Inheritance)** - Existing code (classes) can be reused in new programs.
* **Polymorphism** - Same function behaves differently based on the object.
* **Abstraction** - Only necessary details are shown, hiding complexity.

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

* Steps are below:
* Install a C++ Compiler  
  Download and install a C++ compiler like MinGW (for Windows), g++ (Linux), or Xcode command line tools (Mac).
* Install a Text Editor or IDE  
  Choose an editor or IDE like Code: Blocks, Dev C++, Visual Studio, or VS Code to write and manage your code.
* Set Environment Variables (if needed)  
  Add the compiler's bin folder path to the system environment variables (mainly for Windows users) so you can compile from anywhere using the terminal.
* Write Your First C++ Program  
  Open your editor and write a simple program using basic C++ syntax to test the setup.
* Compile and Run the Program  
  Use the build/run option in the IDE or use a terminal command like g++ file.cpp -o file and./file to run it.
* Debug and Fix Errors  
  Use the IDE’s output and debug features to find and fix any errors or bugs in your code.

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

* Cin (Common Input)  
  cin is used to take input from the user through the keyboard.  
  It reads data and stores it in a variable.

- Example:

cin >> name;

* **cout (Common Output)**  
  cout is used to display output on the screen.  
  It prints text or variables to the console.

- Example:

cout << "Hello, World!";

* Example

#include <iostream>

using namespace std;

main () {

int a;

cout << "Enter number: ";

cin >> a;

}

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

* data types explain below in detail
* Integer (int)  
  Used to store whole numbers (positive or negative).  
  Example:int age = 25;
* Floating Point (float)  
  Used to store decimal numbers (single precision).  
  Example: float price = 99.99;
* Double (double)  
  Used to store decimal numbers with more precision (double precision).  
  Example: double pi = 3.1415926535;
* Character (char)  
  Used to store a single character.  
  Always enclosed in single quotes.  
  Example: char grade = 'A';
* Boolean (bool)  
  Stores true or false values.  
  Used for conditions and logic.  
  Example: bool isPassed = true;
* String (string)  
  Used to store a sequence of characters (text).  
  You must include the #include <string> header.  
  Example: string name = "Khushi";

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

* **Implicit Type Conversion (Type Promotion)**
* Also known as **automatic type conversion**.
* Done by the **compiler automatically** when needed.
* Happens when you assign a smaller data type to a larger one (e.g., int to float).
* No data is lost in safe conversions.
* **Example:**

**Int a= 10;**

**Float b= a; // int automatically converted to float**

* **Explicit Type Conversion (Type casting)**
* Also known as manual type conversion.
* Done by the programmer using a cast operator.
* Used when converting from a larger data type to a smaller one, or when precise control is needed.
* May lead to data loss or truncation if not handled carefully.

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

* Here is a simple explanation of the different types of operators in C++.
* **Arithmetic Operators**  
  Used to perform mathematical operations.  
  Examples:

int a = 10, b = 3;

cout << a + b; // Addition

cout << a - b; // Subtraction

cout << a \* b; // Multiplication

cout << a / b; // Division

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

* **Relational (Comparison) Operators**  
  Used to compare two values and return true or false.  
  Examples:

a == b // Equal to

a! = b // Not equal toa > b // Greater than

a < b // Less thana >= b // Greater than or equal to

a <= b // Less than or equal to

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

* Constants and litrals.
* Constants – Purpose and Use

- To store fixed values that should not change during program execution.

- To make code more readable, maintainable, and secure

* Literals – Purpose and Use

- To represent actual constant values used directly in the code (e.g., 5, "Hello")

- To assign initial or fixed values to variables or use them in expressions

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

* Conditional statements are used to make decisions in a program based on conditions (true or false).  
  They help the program choose different paths of execution.
* if-else Statement

The if-else statement checks a condition.  
If the condition is true, it runs one block of code; otherwise, it runs another.

- Syntax:

if (condition) {

// code if true

} else {

// code if false

}

- Example:

int age = 18;

if (age >= 18) {

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

} else {

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

}

* **switch Statement**

The switch statement is used to check multiple values of a variable.  
It is often used as an alternative to multiple if-else statements.

- Syntax:

switch (expression) {

case value1:

// code for value1

break;

case value2:

// code for value2

break;

default:

// code if no match

}

- 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";

}

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

| Feature | for Loop | while Loop | do-while Loop |
| --- | --- | --- | --- |
| Use case | When number of iterations is known | When condition is checked before execution | When loop must run at least once |
| Condition check | At the beginning | At the beginning | At the end |
| Syntax style | Compact: init, condition, increment in one line | Split: init outside, condition on top | Condition comes after loop body |
| Execution guarantee | May not execute if condition is false | May not execute if condition is false | Always executes once, even if false |
| Example | for (int i = 0; i < 5; i++) | while (i < 5) | do { ... } while (i < 5); |

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

* Break & continue statements
* break Statement in C++

- Used to exit the loop immediately, even if the loop condition is still true.

- Commonly used to stop the loop when a certain condition is met

- Example:

for (int i = 1; i <= 10; i++) {

if (i == 5) {

break; // loop stops when i is 5

}

cout << i << " ";

}

* continue Statement in C++

- Used to skip the current iteration and jump to the next one Loop does not stop;

- only that iteration is skipped

- Example:

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

If (i == 3) {

continue; // skip when i is 3

}

cout << i << " ";

}

1. **Explain nested control structures with an example.**

* **Nested control structures** mean placing one control structure **inside another**, like putting an if inside a for, or a for inside another for. They are used when **multiple conditions or loops** need to be handled together.
* Example: Nested if Statement

int age = 20;

char gender = 'F';

if (age >= 18) {

if (gender == 'F') {

cout << "Eligible woman.";

} else {

cout << "Eligible man.";

}

} else {

cout << "Not eligible.";

}

* **Example: Nested for Loop**

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

for (int j = 1; j <= 2; j++) {

cout << "i = " << i << ", j = " << j << endl;

}

}

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

* function
* What is a Function in C++?

- A function is a block of code designed to perform a specific task.  
- It helps to reuse code, organize programs better, and make code easier to read and maintain.

* Function Declaration Also called function prototype.  
  - It tells the compiler about the function’s name, return type, and parameters before its actual body is defined.  
  - Helps in compiling when function is called before definition.

- Example:

int add (int, int); // Declaration

* Function Definition

- This is where the actual body of the function is written.  
- It specifies what the function does.

- Example:

int add (int a, int b) {

return a + b;

}

* Function Calling

- To use the function, you call it by its name and provide arguments (if any).  
- The program executes the function’s code and returns the result (if any).

- Example:

int result = add (5, 3);

cout << result; // Output: 8

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

* Scope variables
* What is Scope of Variables?

- The scope of a variable refers to the part of the program where the variable can be accessed or used.  
- C++ has two main types of variable scope: Local and Global.

* Local Scope

- A variable declared inside a function or block  
- Can be accessed only within that function or block  
- Memory is released after the block ends

- Example:

void show () {

int x = 10; // local variable

cout << x;

}

* Global Scope

- A variable declared outside all functions  
- Can be accessed from any function in the same file  
- Exists throughout the program

- Example:

int x = 100; // global variable

void display () {

cout << x; // accessible here

}

* Difference

1. **Explain recursion in C++ with an example.**

* Recursion in C++

- Recursion is a programming technique where a function calls itself to solve a smaller part of a larger problem.

- It is commonly used in problems like factorial, Fibonacci series, tree traversal, etc.

* Key Parts of Recursion

- Base Case: Condition to stop recursion (prevents infinite calls)  
- Recursive Call: Function calls itself with a smaller/simpler input

* Example: Factorial Using Recursion

#include<iostream>

using namespace std;

int factorial (int n) {

if (n == 0) // base case

return 1;

else

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

}

int main () {

int num = 5;

cout << "Factorial of " << num << " is " << factorial(num);

return 0;

}

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

* Function prototype
* What is a Function Prototype?

- A function prototype is a declaration of a function that tells the compiler about the functions:

Name

Return type

Number and types of parameters

- It ends with a semicolon and is usually written before main ().

- Example:

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

* Why are Function Prototypes Used?

- To inform the compiler about a function before it is used in the program  
- Helps the compiler check for correct function calls (number/type of arguments)  
- Allows defining the function after main (), improving program structure

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

* Key concepts
* Class

- A blueprint or template for creating objects  
- Defines data members (variables) and member functions (methods)

- Example:

class Car {

public:

string brand;

void start () {

cout << "Car started";

}

};

* Object

- A real-world entity created from a class  
- Has its own identity, state, and behaviour

- Example:

Car my Car;

myCar.brand = "Toyota";

myCar.start();

* Encapsulation

- Binding of data and functions into one unit (class)  
- Data is kept safe from outside access using private access modifiers

* Abstraction

- Hiding complex details and showing only essential features  
- Helps in reducing complexity and increasing readability

* Inheritance

→ One class (child) can inherit properties and behavior from another (parent)  
→ Promotes code reusability

Example:

class Vehicle {

public:

void run () {

cout << "Running";

}

};

class Bike: public Vehicle {}; // Bike inherits Vehicle

* Polymorphism

- One name can have many forms  
- Allows functions or methods to behave differently based on context

Types:

Compile-time (Function Overloading)

Run-time (Function Overriding)

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

* Class a& object
* What is a Class in C++?

- A class is a user-defined data type that acts as a blueprint for creating objects.  
- It contains data members (variables) and member functions (methods) to define the behaviour of the object.

* What is an Object in C++?

- An object is a real-world entity created from a class.  
- It has its own data and behaviour as defined in the class.

* Example

#include<iostream>

using namespace std;

class Student {

public:

void show () {

cout << "Hello, I am a student.";

}

};

int main () {

Student s1; // Object created

s1. show(); // Function called using object

return 0;

}

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

* Inheritance

- Inheritance in C++ is a feature that allows one class (child/derived class) to acquire properties and behaviours (data and functions) from another class (parent/base class).  
It helps in code reusability and building a relationship between classes.

- Types of Inheritance:

* + Single
  + Multiple
  + Multilevel
  + Hierarchical
  + Hybrid

- Example

#include<iostream>

using namespace std;

class Animal {

public:

eat () {

cout << "Eating\n";

}

};

class Dog : public Animal {

public:

bark () {

cout << "Barking\n";

}

};

int main () {

Dog d;

d.eat (); // Inherited function

d.bark (); // Own function

return 0;

}

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

* Encapsulation
* What is Encapsulation in C++?

- Encapsulation is an object-oriented programming concept where data (variables) and functions (methods) that operate on the data are bundled together into a single unit — called a class.

- It helps in protecting data from outside interference and misuse by restricting direct access.

* How is Encapsulation Achieved in C++?

- Encapsulation is achieved using:

- Classes – to wrap data and functions together

- Access Specifiers – like private, public, and protected to control access