Module-3 Introduction to OOPS Programming

1.Introduction to C++ Ans:

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

Aspect	Procedural Programming (POP)	Object-Oriented Programming (OOP)
Approach	Follows **top-down** approach (focuses on functions/procedures).	Follows **bottom-up** approach (focuses on objects).
Focus	Focus is on **functions** that operate on data.	Focus is on **objects** that contain data and methods.
Data Handling	Data is often **global** and can be accessed by any function.	Data is **encapsulated** within objects (private/protected).
Reusability	Limited – functions may need modification for reuse.	High – objects and classes can be reused (inheritance).
Security	Less secure (global data can be changed easily).	More secure (access specifiers like private, protected, public).
Examples	C, Pascal	C++, Java, Python (OOP style)

1. Main Advantages of OOP over POP:

1. **Encapsulation** – Bundling data and methods inside a class protects data from accidental modification.
2. **Inheritance** – Reusability of code by deriving new classes from existing ones.
3. **Polymorphism** – Ability to use a single function/operator in multiple ways (e.g., function overloading).
4. **Abstraction** – Hiding implementation details and showing only necessary features.
5. **Reusability** – Classes and objects can be reused across different projects.

6. **Modularity** – Code is organized into objects, making it easier to understand, debug, and maintain.
7. **Security** – Access specifiers (private, protected, public) provide controlled access.

---

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

1. **Install a Compiler** ○ Windows: Install **MinGW (GCC)** or use **Visual Studio**. ○ Linux:

   Already comes with g++. Install with:

   - sudo apt-get install g++ ○ Mac: Install **Xcode Command**

   Line Tools (xcode-select --install).

2. **Install an IDE or Editor** ○ Popular IDEs: **Code::Blocks**, **Dev C++**, **Visual Studio**, **CLion**.
   - Lightweight editors: **VS Code**, **Sublime**, **Atom**.
3. **Set Environment Path (if needed)**
   - On Windows, add compiler path (e.g., C:\MinGW\bin) to system environment variables.
4. **Write Your First Program**
5. #include <iostream>
6. using namespace std;
7. int main() {
8. cout << "Hello, C++!" << endl;
9. return 0;
10. }
11. **Compile and Run** ○ Command line:
    - g++ program.cpp -o program ○

    ./program

---

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

1. Install a C++ Compiler

- A compiler converts your C++ source code (.cpp) into machine code.
- Popular compilers:
  - GCC (GNU Compiler Collection) – common on Linux, also available on Windows via MinGW or Cygwin.
  - Clang – widely used on macOS and Linux.

- MSVC (Microsoft Visual C++ Compiler) – comes with Visual Studio on Windows.

👉 Without a compiler, you cannot run C++ programs.

---

2. Choose an Editor or IDE

- An IDE (Integrated Development Environment) provides code editing, compiling, debugging, and running in one place.
- Popular choices:
  - Visual Studio Code (VS Code) – lightweight, cross-platform (requires extensions).
  - Code::Blocks – beginner-friendly IDE with built-in compiler support.
  - Dev C++ – simple IDE for learning basics.
  - CLion (JetBrains) – professional, advanced features (paid).
  - Visual Studio (Windows) – powerful, widely used.

👉 If you want a lightweight setup, you can also use a text editor (like Notepad++, Sublime) with command-line compilation.

---

3. Install and Configure Build Tools

- On Windows, install MinGW or use the compiler bundled with your IDE.
- On Linux, install GCC/Clang using:
- sudo apt-get install g++
- On macOS, install Xcode Command Line Tools:
- xcode-select --install

---

4. Write a Simple C++ Program

Create a file hello.cpp:

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

int main() {
   cout << "Hello, World!" << endl;
   return 0;
}
```

---

5. Compile the Program

- Windows (MinGW):
- g++ hello.cpp -o hello.exe
- Linux/macOS:
- g++ hello.cpp -o hello

---

6. Run the Program

- Windows:
- hello.exe

- Linux/macOS:
- ./hello

---

7. (Optional) Add Debugging and Extensions
   - Install GDB (GNU Debugger) to debug code.
   - In VS Code, install the C/C++ extension for IntelliSense, debugging, and code navigation.
   - Configure build tasks (tasks.json) and debugger (launch.json).

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

Standard Output (cout) – Displays output on the screen.

- **Standard Error (cerr)** – Prints error messages (unbuffered).
- **Standard Log (clog)** – Prints log messages (buffered).

Examples:

```
#include <iostream> using

namespace std;


int main() {

int age;


  // Input    cout << "Enter

your age: ";    cin >> age;


  // Output    cout << "You are " << age << " years

old." << endl;


  // Error message    cerr << "This is an error

message." << endl;


  // Log message    clog << "This is a log

message." << endl;
```

```
    return 0;
}
```

2. Variables, Data Types, and Operators Ans:

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

C++ data types are mainly classified into 3 categories:

(a) Primitive (Basic) Data Types

- int → integers (whole numbers)
- int age = 20;
- float → floating point (decimal values with single precision)
- float pi = 3.14f;
- double → double precision floating point
- double price = 99.99;
- char → single character
- char grade = 'A';
- bool → Boolean (true/false)
- bool isPassed = true;
- void → no value (mainly used in functions)
- void showMessage() { cout << "Hello"; }

(b) Derived Data Types

- Arrays
- int marks[5] = {90, 85, 76, 88, 92};
- Pointers
- int a = 5;
- int* ptr = &a;
- Functions
- int add(int x, int y) { return x + y; }

(c) User-Defined Data Types

- Structure
- struct Student {
- int id;
- char name[20];
- };
- Class • class Car {
- public:
- string model;
- };
- Enum
- enum Days {Mon, Tue, Wed, Thu, Fri};

---

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

Implicit Type Conversion (Type Casting / Type Promotion)

- Done automatically by the compiler.
- Converts smaller data type → larger data type (to avoid data loss).

Example:

int x = 10; double y = x; // Implicitly converts int

→ double

cout << y; // Output: 10

Explicit Type Conversion (Type Casting)

- Done manually by the programmer.
- Uses casting operators like (type) or static_cast<type>().

Example:

double pi = 3.14159;

int x = (int)pi; // Old style casting int y =

static_cast<int>(pi); // Modern C++ casting cout

<< x << " " << y; // Output: 3 3

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

(a) Arithmetic Operators int a = 10, b = 3; cout << a + b; // 13 cout << a - b; // 7 cout << a * b; // 30 cout << a / b; // 3 cout << a % b; // 1

(b) Relational Operators int x = 5, y = 10; cout << (x == y); // 0 (false) cout << (x != y); // 1 (true) cout << (x < y); // 1 cout << (x > y); // 0

(c) Logical Operators bool a = true, b = false; cout << (a && b); // 0 cout << (a || b); // 1 cout << (!a); // 0

(d) Assignment Operators int num = 10; num += 5; // num = num + 5 → 15 num -= 3; // 12 num *= 2; // 24 num /= 4; // 6 num %= 2; // 0

(e) Increment/Decrement Operators

int i = 5;

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

(f) Bitwise Operators int a = 5, b = 3; cout << (a & b); // AND → 1 cout << (a | b); // OR → 7

cout << (a ^ b); // XOR → 6 cout << (a << 1); // Left shift → 10 cout << (a >> 1); // Right shift → 2

(g) Conditional (Ternary) Operator int age = 20; string result = (age >= 18)

? "Adult" : "Minor";

cout << result;

(h) Special Operators

- sizeof
- cout << sizeof(int); // 4 (depends on system)
- typeid
- cout << typeid(age).name();
- Pointer operators (&, *)
- int a = 10;
- int* p = &a; // address-of
- cout << *p; // dereference → 10

---

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

A constant is a variable whose value cannot be changed once assigned.

- Declared using const or #define.

Example: const double PI

= 3.14159;

#define MAX 100

- Prevents accidental modification.
- Improves code readability and maintainability.

Literals

- Fixed values assigned directly to variables.
- Types of literals:
    - Integer literals: 10, -25, 0 ○ Floating-point literals: 3.14, -0.5 ○ Character literals: 'A', 'b' ○ String literals: "Hello" ○ Boolean literals: true, false

Example:

int x = 100; // 100 is an integer literal char

grade = 'A'; // 'A' is a character literal string name

= "Raj"; // "Raj" is a string literal

3. Control Flow Statements Ans:

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

Conditional statements are used to make decisions in programs. They allow the program to execute certain parts of code based on conditions.

(a) if-else statement

- if → executes a block when the condition is true.
- else → executes a block when the condition is false.

Example:

#include <iostream> using

namespace std;

int main() { int age;

cout << "Enter age: ";

cin >> age;

  if (age >= 18) { cout << "You are eligible

to vote." << endl;

  } else { cout << "You are not eligible to

vote." << endl;

  }

  return 0;

}

(b) switch statement

- Used when multiple choices are available for a single variable.
- Only works with integral/enum/char types (not floats or strings).

Example:

```
#include <iostream> using

namespace std;


int main() {    int day;    cout

<< "Enter day (1-3): ";    cin

>> day;


   switch (day) {       case 1: cout <<

"Monday"; break;       case 2: cout <<

"Tuesday"; break;       case 3: cout <<

"Wednesday"; break;       default: cout

<< "Invalid day"; break;

   }

   return 0;

}
```

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

Loop Type	Syntax	When to Use	Example
for loop	for(init; condition; update)	When number of iterations is known	for(int i=1; i<=5; i++) cout<<i;
while loop	while(condition)	When number of iterations is unknown, depends on condition	while(x>0) { x--; }
do-while loop	do { } while(condition);	When loop should execute at least once	do { cin>>n; } while(n<=0);

-> Key difference: do-while always runs once, even if condition is false.

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

- Immediately exits the loop (or switch).
- Control goes outside the loop.

Example:

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

(i == 3) break; cout

<< i << " ";

}

// Output: 1 2

---

continue

- Skips the current iteration and continues with the next one.

Example:

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

if (i == 3) continue; cout

<< i << " ";

}

// Output: 1 2 4 5

4.Explain nested control structures with an example.

- Nested control structures = using one control structure inside another (if, loop, switch, etc.).
- Useful for handling complex decision-making or looping.

Example: Nested if + loop

#include <iostream> using

namespace std;

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

i++) { if (i

% 2 == 0) { cout << i << " is

Even" << endl;

```
    } else {            cout << i << " is
Odd" << endl;
    }
  }
  return 0;
}
```

Output:

1 is Odd

2 is Even

3 is Odd

4. Functions and Scope

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

-> A function is a block of code that performs a specific task.

- It helps in modularity, reusability, and readability of code.
- Instead of repeating code, we write it once inside a function and call it whenever needed.

Function Components:

1. Declaration (Prototype) – Introduces the function to the compiler.
2. Definition – Contains the actual code (body) of the function.
3. Calling – Invoking the function to execute.

Example:

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

// 1. Declaration (Prototype)
int add(int, int);
```

```
int main() {    // 3. Calling
int sum = add(5, 3);    cout <<
"Sum = " << sum;
   return 0;
}

// 2. Definition int
add(int a, int b) {
return a + b;
}
```

---

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

-> The scope of a variable defines where in the program the variable is accessible.

Types of Scope:

1. Local Scope ○ Declared inside a function/block.
   ○ Accessible only within that block.
2. void demo() {
3. int x = 10; // local variable
4. cout << x;
5. }
6. Global Scope ○ Declared outside all functions.
   ○ Accessible throughout the program (unless shadowed by a local variable). 7. int g = 50; // global variable
8. int main() {
9. cout << g;
10. }

-> Key Difference:

Local Variable	Global Variable
Declared inside a function/block	Declared outside all functions

Lifetime ends when function exits Lifetime exists throughout program

Accessible only within function Accessible anywhere

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

-> Recursion = A function that calls itself to solve a smaller version of the same problem.

- Requires a base condition to stop infinite calls.
- Useful in problems like factorial, Fibonacci, tree traversal.

Example: Factorial using Recursion

```
#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

<< " = " << factorial(num);    return 0;

}
```

Output:

Factorial of 5 = 120

---

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

-> A function prototype is a declaration of a function before it is defined.

- Tells the compiler function name, return type, and parameters.
- Helps in top-down programming where main() is written first.

Why use Function Prototypes?

1. Ensures compiler knows about the function before use.
2. Prevents errors due to implicit declarations.

3. Supports modular programming (functions can be defined later).

Example:

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

// Function Prototype int
multiply(int, int);

int main() {    cout <<
multiply(4, 5);    return
0;
}

// Function Definition
int multiply(int a, int b) {
return a * b;
}
```

5. Arrays and Strings Ans:

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

-> An array is a collection of elements of the same data type stored in contiguous memory locations.

- Each element is accessed using an index (starting from 0).

Types:

- Single-dimensional (1D) array → stores elements in a linear form.
- Multi-dimensional (2D, 3D, …) → stores data in tabular or matrix form.

Difference between 1D and 2D Arrays:

Feature	1D Array	2D Array
Structure	Linear (list of elements)	Tabular (rows × columns)
Declaration	int arr[5];	int arr[3][3];

Access	arr[i]	arr[i][j]
Example	Marks of 5 students	Matrix representation

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

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

1. C-style strings (char arrays)
2. char name[10] = "Hello";
3. cout << name; ○ Ends with a null character ('\0'). ○ Use functions from <cstring> (like strlen, strcpy, strcmp).
4. C++ Strings (using string class from <string> library)
5. #include <iostream>
6. #include <string> 7. using namespace std;
8.

9. int main() {
10. string name = "OpenAI";
11. cout << "Length: " << name.length() << endl;
12. name.append(" GPT");
13. cout << "Updated: " << name << endl;
14. return 0;
15. } ○ Easier to use and more powerful than char arrays. ○ Supports operations like concatenation (+), comparison, substring, etc.

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

1D Array Initialization

// Method 1: Direct initialization int arr1[5]

= {10, 20, 30, 40, 50};

// Method 2: Partial initialization (remaining elements = 0) int arr2[5]

= {1, 2};

```
// Method 3: Compiler counts elements automatically int
arr3[] = {5, 10, 15};
```

2D Array Initialization

```
// Method 1: Row-wise initialization int
matrix[2][3] = {
   {1, 2, 3},
   {4, 5, 6}
};

// Method 2: Single line  int
matrix2[2][3] = {1, 2, 3, 4, 5, 6};
```

---

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

(a) C-style string functions (<cstring>)

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

int main() {     char
str1[20] = "Hello";     char
str2[20] = "World";     cout
<< strlen(str1) << endl;
// Length = 5     cout <<
strcmp(str1, str2) << endl;  //
Compare     strcpy(str1,
str2);                // Copy
str2 into str1     strcat(str1,
" C++");
// Concatenate     cout << str1
<< endl;     return 0;
```

}

(b) C++ string class operations (<string>)

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

int main() {    string
s1 = "Hello";    string
s2 = "World";

   cout << s1 + " " + s2 << endl;   // Concatenation
cout << s1.length() << endl;    // Length    cout <<
s1.substr(1, 3) << endl; // Substring (ell)    cout <<
s1.find("lo") << endl;   // Find position (3)    cout <<
(s1 == s2) << endl;     // Compare (0 = false)    return
0;
}
```

6. Introduction to Object-Oriented Programming

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

OOP is a programming paradigm based on the concept of objects. The
4 main pillars of OOP are:

1. Encapsulation – Wrapping data and functions into a single unit (class).
2. Abstraction – Hiding unnecessary implementation details and showing only the required features.
3. Inheritance – Creating new classes from existing ones to reuse code.
4. Polymorphism – Ability of the same function/operator to behave differently (overloading, overriding).

Together, these make programs more modular, reusable, secure, and maintainable.

---

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

- Class → A blueprint/template for creating objects. It defines data members (variables) and member functions (methods).
- Object → An instance of a class.

Example:

```
#include <iostream> using

namespace std;


// Class definition

class Car { public:

string brand;    int

speed;

   void display() {        cout << "Brand: " << brand << ", Speed: " << speed

<< " km/h" << endl;

   }

};


int main() {

   // Creating objects

   Car car1, car2;


   car1.brand          =          "BMW";

car1.speed = 220;    car2.brand =

"Audi";    car2.speed = 200;


   // Calling member function

car1.display();    car2.display();
```

```
    return 0;
}
```

Output:

Brand: BMW, Speed: 220 km/h

Brand: Audi, Speed: 200 km/h

---

3. What isinheritance in C++? Explain with an example.

-> Inheritance allows a new class (child/derived class) to reuse the properties and methods of an existing class (parent/base class).

Types:

- Single, Multiple, Multilevel, Hierarchical, Hybrid.

Example (Single Inheritance): #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 d;
    d.eat();   // Inherited function
```

```
    d.bark();  // Own function     return
0;
}
```

Output:

This animal eats food.

Dog barks.

Inheritance promotes reusability and reduces code duplication.

---

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

-> Encapsulation means binding data (variables) and methods (functions) together into a class and restricting direct access.

- Achieved using access specifiers:
    - private → accessible only within the class.
    - protected → accessible in class + derived classes.
    - public → accessible everywhere.

Example:

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

class BankAccount {  private:   // Data
hidden (Encapsulation)     double
balance;

public:
    BankAccount(double b) { balance = b; }

    void deposit(double amount) {
balance += amount;
    }
```

```
    double getBalance() {   // Public method to access private data        return
balance;
    }
};

int main() {
    BankAccount acc(1000);

    acc.deposit(500);
    // cout << acc.balance;  ERROR (balance is private)
cout << "Balance: " << acc.getBalance();

    return 0;
}
```

Output:

Balance: 1500