# Module-3

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

Ans.

| **Procedural Programming** | **Object-Oriented Programming (OOP)** |
|---|---|
| Program is divided into functions/procedures. | Program is divided into objects that combine data and behavior. |
| Focuses on functions and the sequence of steps to be carried out. | Focuses on objects that represent real-world entities and their interactions |
| Data is usually separate from functions. Functions operate on data by passing it around. | Data and functions are encapsulated within objects, improving security. |

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

Ans:

**Data Security (Encapsulation)**

**OOP:** object oriented program hides internal details and prevents accidental data manipulation.

**POP:** Data is usually global and functions can access/modify it freely, which can lead to security issues and unintended side effects.

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

Ans:

STEPS:

**Install a C++ Compiler**

A compiler converts C++ source code (.cpp files) into machine code. Popular compilers include:

. MinGW (Minimalist GNU for Windows**)** → for Windows

**Install a Code Editor / IDE**

Text Editor + Compiler → Lightweight setup.

- Examples: VS Code, Sublime Text, Notepad++

**Configure the Compiler with the IDE/Editor**

If you use **VS Code or a text editor**, you must:

1. Install compiler (e.g., MinGW).
2. Add compiler path to the system environment variables (so commands like g++ are recognized in terminal).
3. Install C++ extensions in VS Code.

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

Ans:

INPUT:

```
#Included<iostream>

Using namespace std;

Int main(){

     Int num;

      Cout<< “enter your number to print on screen:” ;

      Cin >> num ;

      Cout << “your enter number is: “ num ;

}
```

OUTPUT:

```
enter your number to print on screen: 10

your enter number is: 10
```

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

Data types:

- Int
- Char
- Float
- Double

Examples:

```
 #include <iostream>

using namespace std;
```

```
int main() {
    int age = 20;
    float pi = 3.14;
    double price = 99.99;
    char grade = 'A';
    bool isPassed = true;

    cout << "Age: " << age << endl;
    cout << "Pi: " << pi << endl;
    cout << "Price: " << price << endl;
    cout << "Grade: " << grade << endl;
    cout << "Passed? " << isPassed << endl;
    return 0;
}
```

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

Ans:

**Implicit Type Conversion (Type Promotion)**

Done automatically by the compiler without programmer intervention.

Assigning a smaller type to a larger type (e.g., int → float).

During expressions involving mixed data types ( int + double).

**Explicit Type Conversion (Type Casting)**

Done **manually by the programmer** to force a conversion.

Used when you want to control the conversion explicitly.

Can be done using:

**C-style cast:** (type)expression

**Function-style cast:** type(expression)

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

Ans:

There are various type of operation :

- Arthimatic operation
- Relational operator
- Logical operator
- Assignment operator
- Increment and decrement operator

Examples.

- Arthimatic operation

```
int a = 10, b = 3;

cout << a + b; // 13

cout << a % b; // 1
```

- Relational operator

```
int a = 5, b = 10;

cout << (a < b);    // 1 (true)
```

- Logical operator

  ```
  bool x = true, y = false;
  cout << (x && y); // 0
  cout << (x || y); // 1
  ```

- Assignment operator

  ```
  int a = 5;

  a += 3; // a = 8
  ```

- Increment and decrement operator

  ```
  int a = 5;

  cout << ++a; // 6 (pre-increment)

  cout << a--; // 6 (prints then decrements to 5)
  ```

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

Ans:

- **Constants in C++**

A constant is a value that cannot be changed after it is defined.
They help make code more readable, maintainable, and safe.

- **Literals in C++**

A literal is a fixed value written directly in the code (like 10, 3.14, 'A', "Hello").
They are the actual constant values used in expressions.

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

Ans:

Conditional statements in **C++** are control structures that allow a program to make decisions based on certain conditions

**if-else Statement**

The **if-else** statement checks a condition and executes one block of code if the condition is true, and another block if it is false.

```
#include <iostream>

using namespace std;


int main() {

   int age;

   cout << "Enter your age: ";

   cin >> age;


   if (age >= 18) {

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

   } else {

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

   }


   return 0;}
```

## switch Statement

The **switch** statement is used when you want to compare a single variable against multiple possible values. It is often more readable than writing many else if statements.

```
#include <iostream>

using namespace std;

int main() {

   int day;

   cout << "Enter a number (1-7): ";

   cin >> day;


   switch (day) {

      case 1: cout << "Monday"; break;

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

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

      case 4: cout << "Thursday"; break;

      case 5: cout << "Friday"; break;

      case 6: cout << "Saturday"; break;

      case 7: cout << "Sunday"; break;

      default: cout << "Invalid input";

   }


   return 0;  }
```

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

Ans:

**for Loop**

The for loop is generally used when you **know in advance** how many times you want to repeat the code.

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

int main() {
   for (int i = 1; i <= 5; i++) {
      cout << "i = " << i << endl;
   }
   return 0;
}
```

OUTPUT:

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

**while Loop**

The while loop is used when you may not know in advance how many times you need to repeat the code. It checks the condition before each iteration.

```
#include <iostream>

using namespace std;


int main() {

   int i = 1;

   while (i <= 5) {

      cout << "i = " << i << endl;

      i++;

   }

   return 0;

}
```

OUTPUT:

```
i = 1

i = 2

i = 3

i = 4

i = 5
```

**do-while Loop**

The do-while loop is similar to the while loop, but it executes the code at least once because the condition is checked after the loop body.

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

int main() {
    int i = 1;
    do {
        cout << "i = " << i << endl;
        i++;
    } while (i <= 5);
    return 0;
}
```

OUTPUT:

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

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

Ans:

**break Statement**

The break statement immediately exits the loop, regardless of the loop condition. The control jumps to the statement after the loop.

```
#include <iostream>

using namespace std;


int main() {

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

        if (i == 5) {

            break; // exit the loop when i is 5

        }

        cout << i << " ";

    }

    return 0;

}
```

OUTPUT:

```
1 2 3 4
```

**continue Statement**

The continue statement skips the current iteration of the loop and moves to the next iteration.

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

int main() {
   for (int i = 1; i <= 10; i++) {
      if (i % 2 == 0) {
         continue; // skip even numbers
      }
      cout << i << " ";
   }
   return 0;
}
```

OUTPUT:

```
1 3 5 7 9
```

12. . Explain nested control structures with an example.

Ans:

**Nested if-else**

You can place an if-else inside another if or else block. Is know as nested if-else statement.

```
#include <iostream>

using namespace std;


int main() {

    int num;

    cout << "Enter a number: ";

    cin >> num;


    if (num > 0) {

        cout << "Number is positive." << endl;

        if (num % 2 == 0) {

            cout << "It is also even." << endl;

        } else {

            cout << "It is also odd." << endl;

        }

    } else if (num < 0) {

        cout << "Number is negative." << endl;

    } else {

        cout << "Number is zero." << endl; }    return 0; }
```

OUTPUT:

```
Enter your number: 10

Number is positive.
It is also even.
```

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

Ans:

a function is a block of code designed to perform a specific task. Functions help make programs modular, readable, and reusable by avoiding repetition of code.

A function usually has a name, return type, parameters (optional), and a body.

Function Declaration (Prototype)

A function declaration tells the compiler about the function’s name, return type, and parameters without defining its body. It is usually placed before main().

return_type function_name(parameter_list);

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

 int → return type

 add → function name

 (int a, int b) → parameters

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

Ans:

**Local Variables**

- Declared inside a function or a block (e.g., within { }).
- Accessible only within that function or block.
- Destroyed automatically when the function/block ends.

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

int main() {
    int x = 10; // local variable
    cout << "x = " << x << endl;
    return 0;
}
// x is NOT accessible here
```

**Global Variables**

- Declared outside all functions, usually at the top of the program.
- Accessible from any function in the same file (or with extern in other files).
- Exist throughout the program's execution.

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

int y = 20; // global variable

int main() {
   cout << "y = " << y << endl; // accessible inside main
   return 0;
}

void display() {
   cout << "y = " << y << endl; // accessible inside another function
}
```

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

Ans:

recursion is a programming technique where a function calls itself to solve a problem. It is often used to solve problems that can be broken down into smaller, similar subproblems.

A **recursive function** must have:

1. **Base case** – the condition under which the function stops calling itself.

2. **Recursive case** – the part where the function calls itself with modified parameters

```
#include <iostream>

using namespace std;

// Recursive function to calculate factorial

int factorial(int n) {

if (n == 0) {        // base case

return 1;

} else {             // recursive case

return n * factorial(n - 1);

}

}


int main() {

int num;

cout << "Enter a number: ";

cin >> num;


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

return 0;

}
```

OUTPUT;

```
 Enter a number: 5

Factorial of 5 is 120
```

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

Ans:

A function prototype is a declaration of a function that informs the compiler about the function’s name, return type, and parameters without providing its body.

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

▯ int → return type

▯ add → function name

▯ (int a, int b) → parameters

WHY FUNCTION IS USED:

Improves code readability and organization:
You can place main() at the top and define functions later in the program.

Allows calling functions before their definition:
Without a prototype, the compiler would throw an error if a function is called before its definition.

```
#include <iostream>

using namespace std;


// Function prototype

int add(int a, int b);
int main() {
    int result = add(5, 3); // Function called before definition
   cout << “total number is “ result <<
    //  cout << "Sum = " << result << endl; //
    return 0;
}

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

OUTPUT:

```
Total number is = 8
```

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

Ans:

Array is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow you to store multiple values under a single variable name, with each element accessed using an index.

**Single-Dimensional Arrays**

A single-dimensional (1D) array is like a list of elements.

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

int main() {
    int numbers[5] = {10, 20, 30, 40, 50}; // 1D array

    for (int i = 0; i < 5; i++) {
        cout << "numbers[" << i << "] = " << numbers[i] << endl;
    }

    return 0;
}
```

OUTPUT:

```
numbers[0] = 10
numbers[1] = 20
numbers[2] = 30
numbers[3] = 40
numbers[4] = 50
```

## Multidimensional Arrays

A multidimensional array has more than one index, like a matrix or table. The most common is a 2D array (rows × columns).

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

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

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

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

Ans;

**C++ std::string**

C++ offers a safer and more flexible way of handling strings via the std::string class from <string>.

```
#include <iostream>

#include <string>

using namespace std;


int main() {

   string str1 = "Hello";

   string str2 = "World";


   // Concatenation

   string str3 = str1 + " " + str2;

   cout << str3 << endl;


   return 0;

}
```

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

Ans:

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

Declaration and Initialization

```
#include <iostream>

using namespace std;


int main() {

    int arr[5] = {1, 2, 3, 4, 5};

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

        cout << arr[i] << " ";

    return 0;

}
```

OUTPUT:

```
1 2 3 4 5
```

**Two-Dimensional (2D) Arrays**

2D arrays can be thought of as arrays of arrays

```
#include <iostream>

using namespace std;
```

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

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

OUTPUT:

```
1 2 3
4 5 6
```

KEY POINTS OF 1D AND 2D:

 **1D array:** type name[size] = {values};

 **2D array:** type name[rows][cols] = {{row1_values},{row2_values},...};

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

Ans:

OOP is a programming paradigm that organizes code around objects and classes rather than just procedures or functions.

**Classes and Objects**

Class

- A blueprint for creating objects.
- Defines attributes (data members) and behaviors (member functions).

```
#include <iostream>

using namespace std;


class Car {

public:

   string brand;

   int year;


   void display() {

      cout << brand << " - " << year << endl; } }

int main() {

   Car car1;         // object creation

   car1.brand = "Toyota";

   car1.year = 2020;

   car1.display();    return 0; }
```

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

Ans;

**Classes and Objects in C++**

**Class**

- A class is a blueprint or template for creating objects.
- It defines attributes (data members) and behaviors (member functions or methods) of objects.

Syntax:

```
class ClassName {
   access_specifier:
      data_members;
      member_functions;
};
```

**Object**

- An object is an instance of a class.
- It contains actual values and can use the member functions of the class.

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

// Class definition
class Car {
 public:
    string brand;      // Attribute
    int year;          // Attribute

    void display() {   // Method
        cout << brand << " - " << year << endl;
    }
 };

 int main() {
    Car car1;          // Object creation
    Car car2;          // Another object

    // Assign values to attributes
    car1.brand = "Toyota";
    car1.year = 2020;

    car2.brand = "Honda";
    car2.year = 2018;
// Call method to display
    car1.display();    // Output: Toyota - 2020
    car2.display();    // Output: Honda - 2018

    return 0; }
```

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

Ans:

**What is Inheritance?**

- Inheritance is an OOP concept where a class (derived/child class) acquires properties and behaviors (data members and member functions) from another class (base/parent class).
- Purpose:
    - Code reusability
    - Establish hierarchical relationships
    - Allows extending functionality without modifying existing code

**Types of Inheritance in C++**

1. Single Inheritance: One base → one derived class
2. Multiple Inheritance: Multiple base classes → one derived class
3. Multilevel Inheritance: Base → Derived → More Derived
4. Hierarchical Inheritance: One base → multiple derived classes
5. Hybrid Inheritance: Combination of the above

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

// Base class
class Vehicle {
```

```
public:
   string brand;
   void honk() {
      cout << "Beep Beep!" << endl;
   }
};

// Derived class
class Car : public Vehicle {  // Car inherits Vehicle
public:
   string model;
};

int main() {
   Car car1;
   car1.brand = "Toyota";  // inherited from Vehicle
   car1.model = "Corolla"; // own member
   car1.honk();            // inherited method

   cout << car1.brand << " " << car1.model << endl; // Output: Toyota
Corolla
   return 0;
}
```

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

Ans:

**What is Encapsulation**

- Encapsulation means binding data (variables) and functions (methods) together into a single unit called a class.
- It is also known as data hiding, because it allows restricting direct access to class data

**How Encapsulation is Achieved in C++**

1. Use of Access Specifiers:
    - private → Data hidden from outside the class.
    - public → Functions (methods) provide controlled access.
    - protected → Data/methods accessible to derived classes (used in inheritance).
2. Data Members (private) + Methods (public).