Module-3 Introduction to oops programming.

1.Introduction to C++

Q1.What are the key differences between procedural programming and object-oriented programming(OOP)?

- The difference between POP and OOP lie in their programming paradigms ,structure, and approach to solving problems.
- 1. Programming Approach:

  OOP:- Focuses on objects,which represent real-world entities.Objects encapsulate data and behaviour.

  POP:- Focuses on procedures or functions. The program is divided into a set functions that operate on data.

- 2. Key concept:
  OOP:- Based on principles like;

  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism

  POP:- Based on dividing the program into procedures or functions, which are called in sequence.

- 3. Data Security:
  OOP:- Provides better data security because data is hidden using encapsulation and accessed via methods.

  POP:- Data is global and can be accessed freely by any functions,leading to less security.

- 4. Code Reusability:
  OOP:- Promotes code reusability through inheritance and polymorphism.

  POP:- Code reusability is limited to function reuse.

Q2 .List and explain the main advantages of OOP over POP.

- Advantages of OOP:-
  1. Modularity and reusability :- OOP emphasizes creating reusable code through the use of classes and objects. This modular approach allows developers to create libraries or componants thet can be reused across multiple projects.
  2. Encapsulation :- OOP groups data and methods that operate on that data within objects, protecting the internal state of the object. This promotes better security and data integrity by exposing only necessary details to other parts of the program.
  3. Inheritance:- OOP enables the creation of new classes based on existing ones, reducing redundancy and improving maintainability.
  4. Polymorphism:- This allows objects of different classes to be treated as objects of a common superclass, enabling flexibility in code. It supports overriding methods to define specific behaviours for derived classes while maintaining a consistent interface.
- Advantages of POP:-
  1. Simplicity :- POP follows a straightforward, step-by-step approach where tasks are broken down into smaller functions or procedures, making it easy to understand for beginners.
  2. Efficient for smaller programs:- POP works well for small to medium-sized programs where the overhead of OOP may not be necessary.
  3. Ease of implementation:- Since POP relies on procedurals and sequential logic, it is often easier to implement and excecute, especially for algorithms or computations.
  4. Low resources Requirements:- Compared to OOP , POP tends to have lower memory and processing overhead, as it doesn’t required object creation or class management.

Q3. Explain the steps involved in setting up a c++ development environment.

- Setting up a C++ development environment involves several steps to ensure you can write, compile, and execute C++ programs seamlessly. Below is a step-by-step guide:

1. Choose an IDE or Text Editor

- IDE (Integrated Development Environment): Provides tools like a text editor, compiler, and debugger in one package.
  - Popular IDEs for C++:
  - Visual Studio (Windows)
  - Code::Blocks (Cross-platform)
  - CLion (Cross-platform, paid but powerful)

2. Install a C++ Compiler

  - A C++ compiler translates C++ code into machine code. Common compilers include:
  - GCC (GNU Compiler Collection): Available for Linux, macOS, and Windows (via MinGW).
  - MSVC (Microsoft C++ Compiler): Comes with Visual Studio for Windows.
  - Clang: A modern compiler available for macOS, Linux, and Windows.
  - Installation Options:
  - On Windows:
  - Install MinGW or TDM-GCC for GCC.
  - Use Visual Studio to install MSVC.

Q4. What are the main input/output operations in C++? Provide example.

In C++, input and output operations are primarily handled using streams provided by the iostream library. The main input/output operations are:

1. Input Operations

C++ uses the std::cin object for input operations, which allows you to receive data from the user through the standard input (keyboard).

The >> operator is called the extraction operator.

- It reads and stores user input into the specified variable.

Example: Input Operation

```
#include <iostream>
using namespace std;
int main()
{
   int age;
   cout << "Enter your age: ";
   cin >> age;  // Takes input from the user
   cout << "You entered: " << age << endl;
   return 0;
}
```

2. Output Operations

C++ uses the std::cout object for output operations, which allows you to display data on the standard output (console).

The << operator is called the insertion operator.

- It outputs data to the console.

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

int main() {
   cout << "Hello, World!" << endl;  // Outputs text
   cout << "The result is: " << 42 << endl;  // Outputs text and numbers
   return 0;
}
```

2. Variables,Data types and operators:

Q1. What are the different data types available in C++? Explain with example.

- In C++, data types specify the type of data a variable can hold. They are broadly categorized into Primitive, Derived, and User-defined types.
- 1) Primitive Data Types
-
- a) Integer Types
-     • int: Stores integers (whole numbers).
-     • short, long, long long: Variants with different ranges.
- Example : int age = 25;
- long population = 7800000000;
- short small Number = 300;

- b) Floating-Point Types
-     • float: Stores single-precision decimal numbers.
-     • double: Stores double-precision decimal numbers.
-     • long double: Stores extended-precision decimal numbers.
- Example : float price = 10.5;
- double pi = 3.14159;
- long double largeDecimal = 2.71828182845

- c) Character Types
- char: Stores a single character enclosed in single quotes.
- Example:
- char grade = 'A';

- d) Boolean Type
- bool: Stores true or false.

- Example: bool isAvailable = true;

- 2. Derived Data Type:

- a) Arrays: Collection of elements of the same type.
- Example: int numbers[5] = {1, 2, 3, 4, 5};

- [b) Pointers

- Stores the address of another variable.

Example:

- int x = 10;
- int* ptr = &x;

- c) References

- Acts as an alias to another variable.

- Example:
- int a = 10;
- int& ref = a;

- 3. User-Defined Data Types

- a) Structures

- Groups variables of different types.

```
 struct Student {
    int id;
    char name[50];
    float grade;
};
Student s1 = {1, "John", 9.5};
```

b) Classes

- Used in object-oriented programming to define objects

Example:

```
class Car {
public:
    string brand;
    int year;
};
Car car1 = {"Toyota", 2022};
```

Q2. Explain the difference between implicit and explicit type conversion in C++.

- ➢ Implicit Type Conversion (Type Promotion)
  - Automatically performed by the compiler.
  - Happens when a smaller or lower-precision type is converted to a larger or higher-precision type.
  - No loss of data (in most cases).
- ➢ Explicit Type Conversion (Type Casting)
  - Manually specified by the programmer.
  - Done using casting operators like (type) or static_cast.
  - May result in data loss or precision issues.

Q3. What are the different types of operators in C++? Provide example of each.

- ➢ C++ support wide range of operator some basic categories:

1. Assignment operator
2. Arithmetic operator
3. Increment and decrement operator
4. Relational operator
5. Logical operator
6. Conditional operator
7. Bitwise operator
8. Scope resolution operator

➢ 1. Assignment operator :-

- ➔ The assignment operator assign value to a variable. Using assignment operator you can assign a constant value, value of an expression or you can assign value of one variable to another variable.
- ➔ Example: A=5

➢ 2. Arithmetic operator:-

- ➔ Arithmetic operator are use of perform arithmetic operation on numeric data. C++ support five arithmetic operator.
- ➔ 1.Addition
- ➔ 2.subtraction
- ➔ 3.Multiplication
- ➔ 4.division
- ➔ 5.Modulo

1.Addition operator:

Example :- a=5; b=5;

C=a+b;

2.subtraction operator:

Example : - a=5; b=2;

C=a-b;

3.Multiplication operator:

Example : - a=5; b=5;

C=a*b;

4.Division operator:

Example: - a=5; b=10;

C=a/b;

5. Modulo operator:
   Example: - a=5; b=10;
   C=b%a;

- 3. Increment and decrement operator:-

➔ If you want to perform increment or decrement 1value of any variable you can use increment operator(++) or decrement operator(--). It consider as unary operator because it perform operation on single operand.

```
Example:-  a++ is equivalent to a = a+1;
           a—is equivalent to a = a-1;
```

- 4. Relational operator:

➔ To perform comparison between two expression you can use the relational operator. Relational operators are == (equal to) , !=(not equal to), >(greater than), <(less than), >== (greater than or equal to) and <= (less than equal to).

```
Example:-  A>B;
           A <B;
           A<=B;
           A>=B;
           A==B;
           A!=B;
```

- 5. Logical operator:-

➔ Used to combine conditional expressions.

➔ Operators: && (AND), || (OR), ! (NOT)

Example: - AND :- A && B
OR :- A || B

- 6. Conditional operator:-

➔ Used for concise conditional expressions.
➔ Operator: condition ? expr1 : expr2

Example :- a=5; b=10;
Max = a>b ? a:b

- 7. Bitwise operator:-

➔ Bitwise operator works on bits and performs bit by bit operation.
➔ & - AND
➔ | - OR
➔ ^ - XOR
➔ ~ - NOT

Example:- A & B

A | B

A ^ B

~A

- 8. Scope resolution operator:-

➔ Scope resolution operator is newly introduced in C++. The :: operator is unary operator. It is used to access global variable.

```
Example:- int no = 10;
          Int main()
          {
                Int no = 5;
                Cout<<"\n No";
                 Cout<<::No";
           }
```

Q4. Explain the purpose and use of constant and literals in C++.

➔ 1. Constants:
➔ Purpose: Represent fixed values that cannot be changed during the program's execution.
➔ Use:
➔ Prevent accidental modification of important values.
➔ Improve code readability and maintainability.
➔ Example: const double PI = 3.14159;
➔ 2. Literals:
➔ Purpose: Directly specify fixed values in the code.

➔ Use:
➔ Represent hardcoded values like numbers, characters, or strings.
➔ Initialize variables or use in expressions.
➔ Example: int age = 25;

3. Control Flow statements:-

Q1. What are conditional statements in C++? Explain the if-else and switch statements.

➔ Conditional statements control the flow of a program by executing certain blocks of code based on conditions.
➔ 1. if-else Statement
➔ Purpose: Executes one block of code if the condition is true and another if false.
➔ Syntax:
➔ if (condition) {
➔   // Code for true condition
➔ } else {
➔   // Code for false condition
➔ }
➔ 2. switch Statement
➔ Purpose: Evaluates an expression and executes the matching case block.
➔ Syntax:
➔ switch (expression) {
➔   case value1: // Code for value1; break;
➔   case value2: // Code for value2; break;
➔   default: // Code if no case matches
➔ }

Q2. What is the difference between for, while, and do-while loops in C++?

Q3. How are break and continue statements used in loops? Provide example.

- ➔ 1. break Statement
- ➔ Purpose: Exits the loop immediately, skipping all remaining iterations.
- ➔ Usage: Often used to terminate a loop when a specific condition is met.
- ➔ Example:
- ➔ for (int i = 0; i < 5; i++) {
- ➔ if (i == 3) break; // Exits loop when i equals 3
- ➔ cout << i << " ";
- ➔ }
- ➔ // Output: 0 1 2
- ➔ 2. continue Statement
- ➔ Purpose: Skips the current iteration and moves to the next iteration of the loop.
- ➔ Usage: Used to skip specific iterations based on a condition.
- ➔ Example:
- ➔ for (int i = 0; i < 5; i++) {
- ➔ if (i == 3) continue; // Skips iteration when i equals 3
- ➔ cout << i << " ";
- ➔ }
- ➔ // Output: 0 1 2 4

Q4. Explain nested control structure with an example.

- ➔ A nested control structure is when one control statement (like if, for, while, etc.) is placed inside another control statement. This allows more complex decision-making and control flows.
- ➔ Types of Nested Control Structures:

1.Nested if Statement 2. Nested Loops. 3. Combination of if Loops

Combination of if and Loops

You can also combine if statements and loops for more complex control flows.Nested control structures allow for more complex decision-making and iterations. You can nest multiple loops, if conditions, or combinations of both to handle intricate logic.

4.Functions and scopes:-

Q1. What is a function in C++? Explain the concept of function declaration, definition, and calling.

➔ A function in C++ is a block of code that performs a specific task and can be called multiple times with different arguments. Functions help in organizing and reusing code.

a) Function Declaration

•Purpose: Tells the compiler about the function's name, return type, and parameters without providing the function's implementation.

•Syntax:

```
return_type function_name(parameter_list);
```

Example: int add(int, int); // Declaration of a function that adds two integers

b) Function Definition

•Purpose: Provides the actual implementation of the function, where the task is performed.

•Syntax:

```
return_type function_name(parameter_list) {
  // Function body
}
```

Example:

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

c) Function Calling

•Purpose: To execute a function and get the result.

•Syntax:

```
unction_name(arguments);
```

Example:

```
int result = add(5, 3);  // Function call, passing arguments 5 and 3
cout << result;          // Output: 8
```

Q2. What is the scope of variables in C++? Differentciate between local and global scope.

➔ The scope of a variable defines the region of the program where it can be accessed or modified.

➔ Local Scope vs. Global Scope

1.Local Scope:

•Definition: Variables declared inside a function or block.

•Access: Accessible only within the function or block where it is declared.
•Lifetime: Exists only during the execution of that function or block.

2. Global Scope:
•Definition: Variables declared outside all functions, usually at the top of the program.
•Access: Accessible by any function throughout the program.
•Lifetime: Exists for the entire program execution.

Q3. Explain recursion in C++ with an example. Recursion in C++

→ Recursion is a programming technique where a function calls itself in order to solve a problem. A recursive function typically has two parts:

1.Base Case: A condition that stops the recursion.

2.Recursive Case: The part where the function calls itself.

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;
```

}

Explanation:

•The function factorial() calls itself with a smaller value of n until it reaches the base case (n == 0).

•Output: Factorial of 5 is: 1l\\20

Recursion is useful for problems that can be broken down into smaller sub-problems, like tree traversal, sorting algorithms, etc.

Q4. What are function prototype in C++? Why are they used?

➔ function prototype is a declaration of a function that specifies the function's name, return type, and parameters, but does not provide the function's body. It informs the compiler about the function's interface before its actual definition is encountered in the code.
➔ Syntax of Function Prototype:
➔ _type function_name(parameter_list);

Example:

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

→ Why Function Prototypes are Used:

1.Forward Declaration: Allows the function to be used before it

definition.

2.Type Checking: Ensures that function calls are made with the correct number and type of arguments.

3.Modular Programming: Helps in organizing code, especially in large programs with functions defined later in the code.

5. Arrays and string:

Q1. What are arrays in C++? Explain the difference single-dimensional and multi-dimensional arrays.

➔ An array in C++ is a collection of elements of the same data types stored in contiguous memory locations. Arrays are used to stored multiple values under a single variable name.

Feature	Single-dimensional array	Multi-dimensional array
Structure	Stores data in a single row or linear format.	Stores data in a grid or tablular format
Declaration	data_type array_name[size];	Data_type Array_name[row_size] [col_size];
Access	Accessed with one index	Accessed with multiple index
Use case	Used for linear lists marks or temparatures.	Used for matrics,tables,or grids.

Q2. Explain string handling in C++ with example.

➔ In C++, strings are sequences of characters. String handling can be done using:

1.C-style Strings (character arrays).

2.std::string Class from the Standard Template Library (STL).

➔ 1. C-Style Strings

•Represented as a character array terminated by a null character ('\0').

•Example:

- char str[20] = "Hello, C++";
- cout << str; // Output: Hello, C++

➔ 2. std::string Class

•Provides advanced string handling with built-in functions for manipulation.

•Example:

```
#include <iostream>
#include <string>
using namespace std;
int main() {

 string str1 = "Hello";

 string str2 = "World";

  string result = str1 + " " + str2;  // Concatenation

  cout << result << endl;  // Output: Hello World

   cout << "Length: " << result.length();  // Output: Length: 11
  return 0;
```

}

➔ Common String Functions:
- length(): Returns the length of the string.
- substr(start, length): Extracts a substring.
- find(substring): Finds the position of a substring.
- compare(str): Compares two strings.

➔ Using std::string is recommended for better flexibility and ease of use.

Q3. How are array initialized in C++provide examples of both 1D and 2D arrays.

➔ Arrays in C++ can be initialized during declaration or later in the program. below are example for 1D and 2D array.

➔ D Array Initialization

➔ A one-dimensional array stores elements in a linear format.

Syntax:

➔ data_type array_name[size] = {values};

Example:

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

 int main() {

 // Declaration and Initialization

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

// Accessing elements

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

cout << arr[i] << " ";  // Output: 1 2 3 4 5

}

return 0;

}
```

2D Array Initialization

➔ A two-dimensional array stores data in rows and columns.

Syntax:

➔ data_type array_name[rows][columns] = {{row1_values}, {row2_values}, ...};

Example:

```
#include <iostream>
using namespace std;
 int main() {
// Declaration and Initialization
 int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
// Accessing elements
 for (int i = 0; i < 2; i++) {          // Rows
for (int j = 0; j < 3; j++) {      // Columns
cout << matrix[i][j] << " ";  // Output: 1 2 3 4 5 6
}cout << endl;  // New line for each row
}
   return 0;
}
```

Q4. Explain string operations and functions in C++?

➔ In C++, String can be handled using:

1. C- style strings.
2. Std::string class from the standard template library which provides more functionality.

➔ Common string operation in C++:

1. Using std::string class
    - Concatenations
    - Accessing characters
    - Comparison
    - Substring Extraction
2. Common std::string functions
    - Length()/size()
    - Append(str)
    - Find(substring)
    - Replace(pos,len,str)
    - Erase(pos,len)

➔ Using the std::string class is preferred for modern C++ due to its flexibility and built-in functions for efficient string handling.

6.Introduction to object – oriented programming:

Q1. Explain the key concept of OOP.

➔ Key Concepts of OOP (Object-Oriented Programming)

1.Encapsulation:
  •Bundling data and methods into a single unit (class).
  •Example:

2. Inheritance:
  •Creating new classes (derived classes) based on existing ones (base classes).
  •Promotes code reusability.
  •Example:

3. Polymorphism:
  •Ability to use a single interface for different types.
  •Example: Method Overloading and Overriding.

4. Abstraction:
  •Hiding complex implementation details and showing only the essentials.
  •Example:

Encapsulation: Data hiding.

Inheritance: Code reusability.

Polymorphism: Multiple forms of the same function.
Abstraction: Focus on "what" instead of "how."

Q2. What are classes and objects in C++? Explain with an example.

➔ Class: A blueprint for creating objects. It defines properties (data members) and behaviors (member functions).

➔ Object: An instance of a class that uses the blueprint to hold specific data and perform actions.

Syntax of Class:

```
class ClassName {

 public:

  data_members;    // Variables
  member_functions; // Methods

};
```

Example:

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

    // Class definition

    class Car {
     public:

      string brand;    // Data member

     int speed;       // Data member

  void display() { // Member function

   cout << "Brand: " << brand << ", Speed: " << speed << endl;

    }

};

    int main() {

   // Creating an object

   Car car1;

   car1.brand = "Toyota";  // Assign values to data members

    car1.speed = 120;
    car1.display();  // Call member function

return 0;

}
```

Q3. What is inheritance in C++? Explain with an example.

➔ Inheritance is a mechanism in C++ where one class inherits properties and behaviour .it promotes code reusability and helps in creating hierarchical relationship.

➔ Types of inheritance:

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

Example:

```
#include<iostream>
Using namespace std;
Class vehical
{
        Public:
                Void start()
                        {
                                Cout<<"\nVehical started." <<endl;
                        }
};
Class car : public vehical
{
        Public:
                Void drive()
                {
                        Cout<<"\nCar is driving" <<endl;
                }
};
Int main()
{
Car c;
c.start();
c.drive();
return 0;
}
```

Q4. What is encapsulation in C++? How is it achieved in classes?

➔ Encapsulation is the concept of bundling data (attributes) and methods (functions) that operate on the data into a single unit (class). It also restricts direct access to some of an object’s components, ensuring controlled access through methods.

→ How is Encapsulation Achieved?

1.Access Specifiers: private, protected, and public control access to class members.

•Private: Members are accessible only within the class.

•Public: Members are accessible from outside the class.

2.Getter and Setter Methods: Used to access and modify private members.