**Module – 3**

**Introduction To OOPS Programming**

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

**Ans. The Differences between POP & OOP is as follows**

**POP:**

1. **Top to down approach: The program is broken down into functions.**
2. **No access specifiers: There are no access Like private, public, or protected.**
3. **Less Secure: Data hiding is not possible, making it less secure**
4. **No Inheritance: Inheritance is not supported.**
5. **Function-oriented: Functions are more important than data.**

**OOP:**

1. **Bottom-Up Approach: The program is broken down into objects.**
2. **Access specifiers: support access specifiers like private, public, & protected.**
3. **More secure: Data hiding is possible, making it more secure.**
4. **Supports inheritance: Inheritance allows new objects to take on properties of existing objects.**
5. **Data-Oriented: Data is more important than functions.**

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

**Ans.**

1. **Modularity**

* **OOP Advantage: In POP, the program is divided into objects, making it modular. Each objects represents a specific part of the application with its own data & behavior.**
* **Why it’s better than POP: In POP, code is divided into procedures or functions, but data is shared across functions, making modularity harder to maintain & increasing the chance of unintended side effects.**

1. **Reusability**

* **Through inheritance, OOP allows classes to reuse properties &methods of other classes.**
* **Advantage: Reduce code duplication & promotes code reuse, saving time & effort.**

1. **Encapsulation**

* **Encapsulation means hiding internal state & requiring all interaction to be performed through objects methods.**
* **Advantage: Enhance security & prevents accidental interference with data by restricting direct access.**

1. **Abstraction**

* **OOP allows you to hide complex implementation details & show only essential features.**
* **Advantage: Simplifies the interface for the users & helps in managing complexity.**

1. **Polymorphism**

* **Polymorphism allows objects to be treated as instances of their parent class rather than their actual class.**
* **Advantage: Promotes flexibility & integration by enabling the same interface to behave differently with different lasses.**

1. **Maintainability**

* **Because of the modular structure, code changes in one part typically don’t affect others.**
* **Advantage: Makes it easier to update, scale, & maintain applications.**

1. **Scalability**

* **OOP system can be more easily expanded by adding new classes or objects.**
* **Advantage: Better suited for large software projects.**

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

**Ans.**

1. **Choose and install a C++ compiler.**
2. **Choose and ide or text editor.**
3. **Setup the complier in your IDE.**
4. **Write your first C++ program.**
5. **Compile & run.**
6. **Optional: Install a debugger.**
7. **Test your setup.**
8. **What are the main input/output operations in C++? Provide examples.**

**Ans. Cin – for input**

**Cout – for output**

**Explanation: cin>>variable; reads input into variable.**

**Cout<<value; prints a value to the screen.**

**Endl moves the output to a new line(like \n).**

**Example:**

**#include<iostream>**

**Using namespace std;**

**Main()**

**{**

**Int n;**

**Cout<< “Enter No Of Student: ”;**

**Cin>>n**

**}**

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

**Ans.**

1. **Fundamental primitive data types**

**Int – Integer Number – Age=25**

**Float – Float point number – float pi=3.14**

**Double – double precision floating point – g=9.81**

**Char – single character – char grade = ‘A’**

**Bool – Boolean Number – bool is on = true**

**Void – no value – void greet (){}**

1. **Derived data types**

**Pointers – Stores add. of another vari – int\*p=&age**

**Array – Colle. of element of same type –**

**Int scores [3]={10,20,30}**

**Functions – Block of reusable code – int sum(int a, int b)**

**Reference – Alias for another variable – int& ref =age**

1. **User defined data types**

**Struct - Collection of variables - struct Person {string nm; int n; };**

**Union - Like struct but shares memory – union data {int i, float f}**

**Enum colour – blue print for object (oop) - class {RED, GREEN, BLUE}**

1. **Abstract or Special Data Type**

**Std::string – string class from standard library - std::string name = "Alice";**

**Std::vector - Dynamic array (part of STL) - std::vector<int> nums = {1, 2};**

**Std::map - Key-value pair storage - std::map<string, int> ageMap;**

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

**Ans. Implicit Type Conversion (Type Promotion):**

**Also called automatic type conversion, this happens automatically when:**

* **Assigning a value to a variable of a different type.**
* **Using mixed-type expressions (e.g., int + float).**

**Features:**

* **Done by the compiler.**
* **Safe (but might cause data loss in some cases).**
* **Usually converts smaller to larger data types (e.g., int → float).**

**Explicit Type Conversion (Type Casting):**

**Also called manual type conversion, it’s done by the programmer using casting operators.**

**Features:**

* **You force the conversion from one type to another.**
* **Useful when precision or specific behavior is needed.**
* **Can lead to data loss if not used carefully.**

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

**Ans.**

1. **Arithmetic Operators:**

**Operator Description Example Result**

**+ Addition 5 + 3 8**

**- Subtraction 5 - 3 2**

**\* Multiplication 5 \* 3 15**

**/ Division 10 / 2 5**

**% Modulus 10 % 3 1**

1. **Relational (Comparison) Operators:**

**Operator Description Example Result**

**== Equal to 5==5 true**

**!= Not equal to 5!=3 true**

**> Greater than 5>3 true**

**< Less than 5<3 true**

**>= Greater or equal 5>=5 true**

**<= Less or equal 3<=5 true**

1. **Logical Operator:**

**Operator Description Example Result**

**&& Logical AND true&&false false**

**! Logical NOT !true false**

1. **Assignments Operator:**

**Operator Description Example**

**= Assign int x=10**

**+= Add & assign x +=5//x=x+5**

**-= Subtract & assign x-=3**

**\*= Multiply & assign x \*=2**

**/= Divide & assign x/=4**

**%= Modulus & assign x%=3**

1. **Increment & Decrement Operator:**

**Operator Description Example**

**++ Increment x++ or ++x**

**-- Decrement x- -or - x**

1. **Bitwise Operator:**

**Operator Description Example**

**& Bitwise AND 5&3 – 1**

**^ Bitwise XOR 5^3 – 6**

**~ Bitwise NOT ~5 – -6**

**<< Left shift 5<<1 – 10**

**>> Right shift 5>>1 – 2**

1. **Conditional (ternary ) Operator:**

**A shorthand for if - else**

**(condition) ? expression 1 : expression 2**

1. **Sizeof Operator:**

**Return the size of a data type or variable (in bytes).**

**Sizeof(int);**

**Sizeof (double);**

1. **Type Casting Operator:**

**Used to convert one type into another.**

**Float x=10.5;**

**Int y = (int) x;**

**Int z = static\_cast <int>(x);**

**10) Scope Resolution Operator:**

**Used to define a function outside a class or access global variable.**

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

**Ans.**

1. **Constants in C++: Constants are fixed values that do not change during the execution of a program. They make code more readable, safe, and maintainable.**

**Use:**

* **Prevent accidental changes to important values.**
* **Improve code clarity (e.g., PI is more meaningful than 3.14).**
* **Makes it easier to update values in one place.**

1. **Literals in C++: Literals are fixed values directly written in code. They're the actual values assigned to variables or constants.**

**Purpose of constants is readability & maintainability.**

**Purpose of literals is to represent actual data in code.**

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

**Ans.**

1. **if, else if, and else Statement:**

* **The program checks each condition in order.**
* **As soon as it finds a true condition, it runs the corresponding block and skips the rest.**

1. **Switch Statement:**

**The switch statement is useful when you need to compare a single value against multiple constants.**

* **The switch compares day to each case.**
* **Executes the matching case and stops at break.**
* **If no match is found, the default block runs.**

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

**Ans.**

1. **for Loop: When you know in advance how many times you want to repeat something.**

* **Counting loops**
* **Iterating over arrays with an index.**

1. **while Loop: When you don’t know beforehand how many times the loop will run.**

* **Looping until a certain condition is met**
* **Input validation.**

1. **do-while Loop: When you want the loop to run at least once, no matter what.**

* **Menu systems**
* **Asking the user if they want to repeat an action.**

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

**Ans.**

**break Statement:**

* **Used to exit a loop immediately, even if the loop condition is still true.**
* **Often used when a certain condition is met, and there’s no need to continue the loop.**

**Use of break:**

* **Exiting early from loops.**
* **Searching algorithms (exit when found)**
* **Menu-driven programs (exit on choice)**

**continue Statement:**

* **Used to skip the current iteration of a loop and move to the next one.**
* **The rest of the loop body after continues is ignored for that iteration.**

**Use of continue:**

* **Ignoring specific conditions.**

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

**Ans. Nested control structures mean placing one control structure (like if, for, while, etc.) inside another.**

**This allows more complex decision-making or looping logic by combining multiple conditions or iterations.**

**Types of Nesting You Can Do:**

* **if inside if**
* **for inside for**
* **while inside if**
* **switch inside for, etc.**

**Example:**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

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

**{**

**if (i % 2 == 0)**

**{**

**cout << i << " is even" << endl;**

**}**

**else**

**{**

**cout << i << " is odd" << endl;**

**}**

**}**

**}**

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

**Ans. A function is a block of code that performs a specific task.**

**Instead of writing the same code over and over, you can define it once in a function and call it whenever needed.**

**Function Declaration:**

**return\_type function\_name(parameter\_list);**

**For Example:**

**Void getdata();**

**Function Definition:**

**void getdata()**

**{**

**cout << "Hello, world!" << endl;**

**}**

**Function Call:**

**Getdata();**

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

**Ans. There are mainly two types.**

* **Local scope**
* **Global scope**

**Local scope:**

* **Declared inside a function, block, or loop.**
* **Only accessible within that block.**
* **Destroyed when the block ends.**

**Global scope:**

* **Declared outside all functions.**
* **Can be accessed from any function in the file.**
* **Exists for the lifetime of the program.**

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

**Ans. Recursion is a process where a function calls itself directly or indirectly to solve a problem.**

**It’s like breaking a big problem into smaller sub problems of the same type.**

**Example:**

**#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**

**}**

**}**

**main()**

**{**

**int num = 5;**

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

**return 0;**

**}**

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

**Ans. A function prototype is a declaration of a function that tells the compiler:**

* **The function’s name**
* **Its return type**
* **The number and types of parameters**

**It appears before the function is actually defined (usually at the top of the program or in a header file).**

**Uses:**

* **Tells the compiler early**
* **Allows flexible structure**
* **Supports modular code**
* **Helps with error checking**

1. **Theory Exe:**
2. **What are arrays in C++? Explain the difference**

**between single-dimensional and multi-dimensional arrays.**

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

**Instead of creating separate variables for each element, you can use an array to store multiple values under one name, accessed by an index.**

**Single-Dimensional Array:**

* **One row of elements**
* **One index [i]**
* **Like a list**

**Multi-Dimensional Array:**

* **Elements arranged in rows and columns**
* **Two (or more) indices [i][j]**
* **Like a table or matrix**
* **Grids, matrices, tables (e.g., chessboard)**

1. **Explain string handling in C++ with examples**

**Ans. In C++, strings can be handled in two main ways:**

* **Using C-style strings (character arrays)**
* **Using the C++ string class (from the Standard Library)**

**C-Style String Example:**

**#include <stdio.h>**

**main()**

**{**

**char name[] = "John";**

**cout << "Name: " << name << endl;**

**}**

**C++ String Example:**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**main()**

**{**

**string name = "Alice";**

**cout << "Hello, " << name << “/n”;**

**}**

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

**Ans. One-dimensional array: A 1D array is essentially a list of values that are stored in a single row.**

**Example: 1D Array elements: 1 2 3 4 5**

**Two-dimensional array: A 2D array can be thought of as a table with rows and columns (like a matrix).**

**Example: 2D Array elements:**

**1 2 3**

**4 5 6**

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

**Ans. In C++, string handling is an essential aspect of programming, and the C++ Standard Library provides a powerful string class to make working with strings easier. In addition to the basic operations, there are several built-in functions that allow efficient manipulation of strings.**

**String operation in C++: The string class in C++ provides several member functions to perform various string operations like concatenation, substring extraction, comparison, etc.**

**C-style String Operations: C-style strings are arrays of characters and require functions from <cstring> for manipulation.**

* **The string class is a high-level abstraction for string manipulation, providing functions like append(), find(), substr(), replace(), etc.**
* **C-style strings are arrays of characters and use functions like strcpy(), strcat(), strlen(), etc.**
* **The string class is generally preferred for ease of use and safety, but C-style strings are still widely used, especially in legacy code or system programming.**

1. **Theory Exe:**
2. **Explain the key concepts of Object-Oriented Programming (OOP).**

**Ans. Object-Oriented Programming (OOP) is a programming paradigm centered around objects rather than just functions and logic. In C++, OOP is a foundational concept that allows you to model real-world entities using classes and objects.**

**Class & Object:**

* **Class: A blueprint for creating objects (like a template).**
* **Object: An instance of a class.**

**Encapsulation:**

* **Wrapping data and functions into a single unit (class).**
* **Use of access specifiers: private, public, and protected.**
* **Protects data from unauthorized access.**

**Abstraction:**

* **Hiding complex implementation details and showing only the necessary features.**
* **Achieved using classes, access specifiers, and interfaces (abstract classes).**

**Inheritance:**

* **One class can inherit properties and behaviors (methods) from another class.**
* **Promotes code reusability.**

**Polymorphism:**

* **Polymorphism = many forms**
* **A function or method behaves differently based on the object.**

**There are two types:**

* **Method overloading**
* **Method overriding**

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

**Ans. In C++, classes and objects are the core building blocks of Object-Oriented Programming (OOP).**

**Class: A class is a user-defined blueprint or template from which objects are created. It can contain:**

* **Data members (variables)**
* **Member Functions (methods)**

**Object: An object is an instance of a class. Once a class is defined, you can create objects based on that class.**

**Each object has its own copies of the class’s data members, and it can use the functions defined in the class.**

**Example:**

**#include<iostream>**

**Using namespace std;**

**Class data**

**{**

**Pulic:**

**Int id;**

**String str;**

**Void getdata()**

**{**

**Cout<< “Enter a ID: ”;**

**Cin>>id;**

**Cout<< “Enter your Name:”;**

**Cin>>str;**

**Cout<< “ID:”<<id;**

**Cout<< “String:”<<str;**

**}**

**}**

**Main()**

**{**

**Data dt;**

**Dt. Getdata();**

**}**

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

**Ans. Inheritance is one of the fundamental principles of Object-Oriented Programming (OOP).**

**It allows a class (derived class) to inherit properties and behaviors (data members and member functions) from another class (base class).**

**This promotes:**

* **Code reusability**
* **Hierarchy/relationship modeling (e.g., a Dog is a type of Animal)**
* **Extensibility (you can add features to existing classes without modifying them)**

**Types of inheritance in C++:**

* **Single inheritance – One base class → One derived class**
* **Multiple inheritance – One derived class inherits from multiple base classes**
* **Multilevel inheritance – A class is derived from another derived class**
* **Hierarchical inheritance – Multiple classes inherit from a single base class**
* **Hybrid inheritance – Combination of above types**

**Example:**

**#include<iostream>**

**Using namespace std;**

**Class data**

**{**

**Pulic:**

**Int id;**

**String str;**

**Void getdata()**

**{**

**Cout<< “Enter a ID: ”;**

**Cin>>id;**

**Cout<< “Enter your Name:”;**

**Cin>>str;**

**}**

**}**

**Class printdata: public data**

**{**

**Void print()**

**{**

**Cout<< “ID:”<<id;**

**Cout<< “String:”<<str;**

**}**

**}**

**Main()**

**{**

**Data dt;**

**Dt. Getdata();**

**Dt. printdata**

**}**

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

**Ans. Encapsulation is one of the core principles of Object-Oriented Programming (OOP).**

**It refers to the bundling of data (variables) and the functions (methods) that operate on that data into a single unit, typically a class.**

**It also helps in data hiding, meaning the internal state of an object is protected from unintended or unauthorized access.**

**Encapsulation is achieved using access specifiers inside a class:**

* **private: Members cannot be accessed from outside the class.**
* **public: Members can be accessed from anywhere.**
* **protected: Used in inheritance (accessible in derived classes).**