**Module #3 Introduction to OOPS Programming**

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

Procedural Programming focuses on functions and follows a linear, step-by-step approach. Data and logic are separate.  
Object-Oriented Programming (OOP) organizes code using objects and classes, emphasizing encapsulation, inheritance, and reusability.

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

the main advantages of Object-Oriented Programming (OOP) over Procedural-Oriented Programming (POP):

1. **Encapsulation**: OOP binds data and methods together in classes, protecting data from unauthorized access. POP does not offer this kind of protection.
2. **Reusability**: OOP allows code reuse through inheritance, while POP requires rewriting code for similar functionality.
3. **Modularity**: OOP divides programs into objects (modules), making code easier to maintain and debug. POP can become complex as the code grows.
4. **Scalability**: OOP supports large and complex systems better by organizing them into objects and hierarchies. POP struggles with large systems due to lack of structure.
5. **Abstraction**: OOP hides complex implementation details and provides a simple interface. POP lacks abstraction mechanisms.
6. **Polymorphism**: OOP allows methods to behave differently based on the object, enabling flexibility. POP does not support this feature directly.

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

In C++, input/output (I/O) operations are mainly handled using the iostream library. The two primary objects are:

1. cin: For input from the user (standard input).
2. cout: For output to the console (standard output).

#include <iostream> using namespace std; main() string name; int age;

cout << "Enter your name: "; cin >> name; // Input for name

cout << "Enter your age: "; cin >> age; // Input for age

cout << "Hello " << name << "! You are " << age << " years old." << endl;

; }

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

**#include <iostream>**

**using namespace std;**

**main() {**

**int age = 25; // Integer**

**float height = 5.9; // Float**

**char grade = 'A'; // Character**

**bool isPassed = true; // Boolean**

**double pi = 3.14159; // Double**

**int arr[3] = {1, 2, 3}; // Array**

**cout << "Age: " << age << endl;**

**cout << "Height: " << height << endl;**

**cout << "Grade: " << grade << endl;**

**cout << "Passed: " << isPassed << endl;**

**cout << "Pi: " << pi << endl;**

**cout << "Array First Element: " << arr[0] << endl; ;}**

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

**Implicit Conversion is an automatic conversion performed by the compiler without requiring special syntax, with no control by the programmer. An example is float x = 10; (int to float).**

**Explicit Conversion is a manual conversion specified by the programmer using (type) or static\_cast<type>(), giving full control over the conversion. An example is int x = (int)5.75; (float to int).**

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

1. **Arithmetic Operators: Used for basic math operations (+, -, \*, /, %).**
2. **Relational Operators: Used to compare values (==, !=, >, <, >=, <=).**
3. **Logical Operators: Used for logical operations (&&, ||, !).**
4. **Assignment Operators: Used to assign values (=, +=, -=, \*=, /=, %=).**
5. **Increment/Decrement Operators: Used to increase or decrease by 1 (++, --).**
6. **Bitwise Operators: Used for bit-level operations (&, |, ^, ~, <<, >>).**
7. **Conditional (Ternary) Operator: A shorthand for if-else (?:).**
8. **Pointer Operators: Used with pointers (\* for dereferencing, & for address-of).**
9. **Comma Operator: Allows multiple expressions in a single statement (,).**
10. **Type-Casting Operators: Used for explicit type conversion ((type), static\_cast<>()).**

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

### **Constants and Literals in C++:**

* **Literal Constants: Fixed values directly written in the code, such as numbers (10), floating-point values (3.14), characters ('A'), and strings ("Hello").**
* **Symbolic Constants: Named constants using const or #define, which cannot be changed after initialization.**
  + **Example: const int MAX\_AGE = 100; or #define PI 3.14159;**

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

### **Conditional Statements in C++**

**Conditional statements in C++ allow you to make decisions in your program based on certain conditions. These statements help control the flow of the program by executing different blocks of code depending on whether a condition is true or false.**

### **1. if-else Statement**

**The if-else statement evaluates a condition and executes the corresponding block of code based on whether the condition is true or false.**

**if (condition) { // Block of code executed if the condition is true**

**} else { // Block of code executed if the condition is false }**

### **3. switch Statement**

**The switch statement allows you to test multiple conditions based on a single variable. It is often used when you have many possible values to check for a single variable.**

**switch (variable) {**

**case value1: // Block of code executed if variable equals value1**

**break;**

**case value2: // Block of code executed if variable equals value2**

**break;**

**default: // Block of code executed if variable doesn't match any case }**

### **Difference Between for, while, and do-while Loops in C++:**

1. **for loop:**

**Purpose: Used when the number of iterations is known in advance.**

**for (initialization; condition; increment/decrement) {**

**// Code to execute }**

1. **while loop:**

**Purpose: Used when the number of iterations is not known, and the loop runs as long as the condition is true.**

**while (condition) { // Code to execute }**

**int i = 0;**

1. **do-while loop:**

**Purpose: Similar to while, but it guarantees at least one execution of the loop, as the condition is checked after the loop.**

**do { // Code to execute } while (condition);**

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

### **break and continue Statements in Loops**

1. **break Statement:**

**Purpose: Exits the loop immediately, terminating further iterations.**

**Usage: Typically used when a specific condition is met, and no further iterations are needed.**

**for (int i = 0; i < 10; i++) {**

**if (i == 5) {**

**break; // Exit the loop when i equals 5**

**}**

**cout << i << " ";**

**}**

**// Output: 0 1 2 3 4**

1. **continue Statement:**

**Purpose: Skips the current iteration and moves to the next iteration of the loop.**

**Usage: Often used when certain conditions are met, and you want to skip the rest of the code in the current iteration.**

**for (int i = 0; i < 10; i++) { if (i == 5) {**

**continue; // Skip the iteration when i equals 5}**

**cout << i << " "; } // Output: 0 1 2 3 4 6 7 8 9**

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

### **Nested Control Structures in C++**

**Nested control structures are control statements (like if, for, or while) inside other control statements.**

### **Nested if-else**

**if (a > b) {**

**if (a > c) {**

**// a is greatest**

**} else {**

**// c is greatest**

**}**

**} else {**

**// similar nested logic**

**}**

### **Nested Loops**

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

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

**// Code executes for each combination of i and j**

**}**

**}**

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

### **Functions in C++:**

**A function in C++ is a block of code that performs a specific task. It is designed to be reusable and modular, allowing you to break down a large program into smaller, manageable pieces.**

**Function Declaration: A prototype that tells the compiler about the function's name, return type, and parameters.  
cpp  
Copy code  
int add(int, int);**

**Function Definition: Provides the actual implementation of the function.  
  
int add(int a, int b) {**

**return a + b;}**

**Function Calling: Executes the function by passing arguments.  
  
int result = add(5, 3); // Calls the function**

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

### **Scope of Variables in C++**

**Local Scope: A variable declared inside a function or block is only accessible within that function or block. It is destroyed when the execution leaves the scope.  
  
void function() {**

**int localVar = 10; // Only accessible inside function }**

**Global Scope: A variable declared outside all functions is accessible throughout the program.  
int globalVar = 20; // Accessible everywhere in the program**

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

### **Recursion in C++**

**Recursion is when a function calls itself to solve a problem.**

### **Example: Factorial Function**

**#include <iostream>**

**using namespace std;**

**int factorial(int n) {**

**if (n == 0) return 1; // Base case**

**else return n \* factorial(n - 1); // Recursive case }**

**main() {**

**cout << factorial(5); // Output: 120 }**

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

### **Function Prototypes in C++**

**A function prototype in C++ is a declaration of a function that specifies its name, return type, and parameters, but does not include the function's body. It informs the compiler about the function before it is used in the code.**

**return\_type function\_name(parameter1\_type, parameter2\_type, ...);**

### **Example:**

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

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

### **Arrays in C++**

**Single-Dimensional Array: A list of elements accessed with one index.  
  
int arr[5] = {1, 2, 3, 4, 5};**

**cout << arr[2]; // Output: 3**

**Multi-Dimensional Array: An array of arrays, accessed with multiple indices.  
  
int arr[2][3] = {{1, 2, 3}, {4, 5, 6}};**

**cout << arr[1][2]; // Output: 6**

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

### **String Handling in C++**

**C-style Strings: Arrays of characters ending with a null character ('\0').  
  
char str[] = "Hello";**

**cout << str; // Output: Hello**

**C++ string Class: A flexible class with built-in methods like length(), append(), etc.  
  
string str = "Hello";**

**str.append(" World");**

**cout << str; // Output: Hello World**

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

### **Array Initialization in C++**

1. **1D Array Initialization:**

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

**Dynamic (using loop):  
int arr[5];**

**for (int i = 0; i < 5; i++) arr[i] = i + 1;**

1. **2D Array Initialization:**

**Static:  
  
int arr[2][3] = {{1, 2, 3}, {4, 5, 6}};**

**Dynamic (using loops):  
c  
int arr[2][3];**

**int value = 1;**

**for (int i = 0; i < 2; i++) for (int j = 0; j < 3; j++) arr[i][j] = value++;**

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

### **String Operations and Functions in C++**

**String Initialization:**

**C++ string: string str = "Hello";**

**Common C++ String Functions:**

**length(): Returns the length of the string.  
  
string str = "Hello";**

**cout << str.length(); // Output: 5**

**append(): Adds text to the end of the string.  
  
str.append(" World");**

**substr(): Extracts a part of the string.  
  
string sub = str.substr(0, 5); // "Hello"**

**find(): Finds a substring within the string.  
  
int pos = str.find("World"); // Returns position of "World"**

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

**Class: Blueprint for creating objects with properties and methods.  
  
class Car { ... };**

**Object: Instance of a class.  
  
Car myCar;**

**Encapsulation: Bundling data and methods, restricting access using access modifiers.  
  
class Car { private: int speed; public: void setSpeed() { ... }; };**

**Abstraction: Hiding complex implementation, exposing only essential features.  
  
class Shape { virtual void draw() = 0; };**

**Inheritance: A class inherits properties and methods from another class.  
  
class Car : public Vehicle { ... };**

**Polymorphism: Methods behave differently based on the object.  
  
class Dog : public Animal { void sound() { ... }; };**

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

**Class: A blueprint that defines data and functions.**

**class Car {**

**public:**

**string color;**

**int year;**

**void start() {**

**cout << "Car is starting";**

**}**

**};**

**Object: An instance of a class.  
  
Car car1; // Object of class Car**

**car1.color = "Red";**

**car1.year = 2020;**

**car1.start(); // Calling function**

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

### **Inheritance in C++**

**Inheritance is a mechanism in C++ where a new class (derived class) inherits properties and methods from an existing class (base class). It allows code reuse and establishes a relationship between classes.**

### **Example:**

**#include <iostream>**

**using namespace std;**

**// Base class**

**class Vehicle {**

**public:**

**void move() {**

**cout << "Vehicle is moving" << endl; }};**

**// Derived class**

**class Car : public Vehicle {**

**public:**

**void start() {**

**cout << "Car is starting" << endl; } };**

**main() {**

**Car myCar;**

**myCar.move(); // Inherited function from Vehicle**

**myCar.start(); // Function of Car; }**

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

**Encapsulation is the concept of bundling data (variables) and methods (functions) that operate on the data within a single unit (class). It restricts direct access to some of the object's components, making the data protected and controlling how it is accessed and modified.**

### **How it is Achieved:**

* **Private access modifier: Data members are usually made private to prevent direct access.**
* **Public methods: Provide getter and setter functions to access or modify the private data.**

### **Example:**

**#include <iostream>**

**using namespace std;**

**class Car {**

**private: int speed; // Private data member**

**public:**

**// Setter method**

**void setSpeed(int s) {**

**speed = s;**

**}**

**// Getter method**

**int getSpeed() {**

**return speed;**

**}**

**};**

**main() {**

**Car myCar;**

**myCar.setSpeed(100); // Accessing speed via setter**

**cout << "Speed: " << myCar.getSpeed() << endl; // Accessing speed via getter }**