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

**Ans.**

|  |  |  |
| --- | --- | --- |
| Key Point | Procedural Programming | Object-Oriented Programming (OOP) |
| Approach | **Top-down approach.** | **Bottom-up approach.** |
| Basic Building Block | **Functions / Procedures.** | **Objects / Classes.** |
| Data Access | **Data is global and accessible directly** | **Data is hidden (encapsulation); accessed via methods** |
| Modularity | **Achieved using functions** | **Achieved using classes and objects** |
| Code Reusability | **Limited** | **High (through inheritance and polymorphism)** |
| Security | **Less secure (no data protection)** | **More secure (access control and data hiding)** |
| Real-world Modeling | **Difficult to model** | **Easier to model** |
| Examples | **C, Pascal, Fortran** | **Java, C++, Python (OOP), C#** |

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

|  |  |
| --- | --- |
| Advantage | Explanation |
| 1. Modularity | In OOP, programs are divided into classes and objects, making code organized and modular. Each part of the program can be developed and tested independently. |
| 2. Reusability | OOP promotes code reuse through inheritance, allowing new classes to reuse existing code with little or no modification. |
| 3. Data Security | OOP uses encapsulation to protect data by restricting access through access modifiers (private, public, protected). |
| 4. Easy Maintenance | Modular structure makes it easier to update, modify, and maintain large programs without affecting other parts. |
| 5. Real-world Modeling | OOP allows better modeling of real-world entities using objects, which improves program design and problem-solving. |
| 6. Polymorphism | Enables one interface to be used for different data types, simplifying code and allowing flexibility through method overloading and overriding. |
| 7. Scalability | OOP is better suited for large, complex, and scalable software projects due to its structured design. |

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

**Ans. Steps to set up a C++ Development Environment**

**1. Install a C++ Compiler**

**· A compiler is needed to convert C++ code into machine language.**

**· Popular compilers:**

**· GCC (g++) – for Linux/Windows (via MinGW).**

**· MSVC – comes with Visual Studio (Windows).**

**· Clang – for macOS/Linux.**

1. **Install an IDE or Text Editor**

**· IDE = Integrated Development Environment (helps write, compile, debug easily).**

**· Popular choices:**

**· Code::Blocks**

**· Dev C++**

**· Visual Studio**

**· Visual Studio Code (VS Code) + C++ extensions**

1. **Configure the Compiler in IDE**

**a. Some IDEs (like Code::Blocks with MinGW or Dev C++) come pre-configured.**

**b. In VS Code:**

**i. Install C/C++ extension (by Microsoft).**

**ii. Configure compiler path (g++ or clang++).**

**iii. Create a tasks.json file to build and launch.json to run/debug.**

1. **Write Your First Program**

**Example: hello.cpp:-**

**#include <iostream>**

**using namespace std;**

**int main() {**

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

**return 0;**

**}**

1. **Compile the Program**

**o Using terminal/command prompt:**

**o g++ hello.cpp -o hello**

**o This creates an executable file (hello.exe on Windows, ./hello on Linux/macOS).**

1. **Run the Program**

**· Windows:**

**Hello**

**· Linux/macOS:**

**./hello**

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

**Ans. Main Input/Output in C++**

**1. cout → Output (print on screen)**

**2. cout << "Hello";**

**3. cin → Input (take from user)**

**4. int x;**

**5. cin >> x;**

**6. getline() → Input a whole line (with spaces)**

**7. string name;**

**getline(cin, name);**

1. **Variables, Data Types, and Operators:-**
2. **What are the different data types available in C++? Explain with examples.**

**Ans. 1. Fundamental (Basic) Data Types**

**These are the most commonly used data types in C++.**

|  |  |  |
| --- | --- | --- |
| Data Type | Description | Example |
| int | **Stores integers (whole numbers, positive/negative)** | **int age = 21;** |
| float | **Stores decimal numbers (single precision)** | **float pi = 3.14;** |
| double | **Stores decimal numbers (double precision, more accurate)** | **double price = 99.99;** |
| char | **Stores a single character (inside ' ')** | **char grade = 'A';** |
| bool | **Stores Boolean values (true or false)** | **bool isPassed = true;** |
| void | **Represents no value (used with functions)** | **void display();** |

**2. Derived Data Types**

**These are built from fundamental types.**

* **Arrays → Collection of similar data types.**
* **int marks[5] = {90, 85, 88, 92, 75};**
* **Pointers → Store memory addresses of variables.**
* **int num = 10;**
* **int\* ptr = &num; // pointer pointing to num**
* **Functions → Return values and perform tasks.**
* **int add(int a, int b) {**

**return a + b;**

**}**

**3. User-Defined Data Types**

**C++ allows programmers to define their own data types.**

* **Structures (struct)**
* **struct Student {**

**int rollNo;**

**char name[20];**

**};**

* **Classes (class) → Used in OOP for defining objects.**
* **class Car {**

**public:**

**string brand;**

**int speed;**

**};**

**✅ Summary**

* **Basic Types: int, float, double, char, bool, void.**
* **Derived Types: Arrays, Pointers, Functions.**
* **User-defined Types: struct, class.**

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

|  |  |  |
| --- | --- | --- |
| Feature | Implicit Type Conversion | Explicit Type Conversion |
| Also Known As | **Type Promotion | Automatic Conversion** | **Type Casting | Manual Conversion** |
| How It's Done | **Automatically by the compiler** | **Manually by the programmer** |
| Syntax | **No special syntax required** | **Use casting syntax like (type) or static\_cast<type>()** |
| Control | **Less control over the conversion** | **Full control over how and what to convert** |
| Risk of Data Loss | **May cause unexpected data loss or errors silently.** | **Programmer is aware of potential data loss.** |
| Common Use Case | **Converting int to float, char to int, etc.** | **Converting float to int, pointers, or class types.** |

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

**Ans. Types of Operators in C++**

* 1. **Arithmetic Operators**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int a = 5, b = 4;**

**cout << "Addition: " << a + b << endl; // 9**

**cout << "Subtraction: " << a - b << endl; //1**

**cout << "Multiplication: " << a \* b << endl; // 20**

**cout << "Division: " << a / b << endl; // 1**

**cout << "Modulus: " << a % b << endl; // 1**

**}**

**2. Relational (Comparison) Operators**

**int x = 6, y = 7;**

**cout << (x == y) << endl; // 0 (false)**

**cout << (x != y) << endl; // 1 (true)**

**cout << (x > y) << endl; // 0**

**cout << (x < y) << endl; // 1**

**3. Logical Operators**

**int a = 6, b = 11;**

**cout << (a > 0 && b > 0) << endl; // 1 (true)**

**cout << (a > 0 || b < 0) << endl; // 1 (true)**

**cout << !(a > 0) << endl; // 0 (false)**

**4. Assignment Operators**

**int a = 12;**

**a += 6; // a = a + 6 → 18**

**a -= 4; // a = a - 4 → 8**

**a \*= 3; // a = a \* 3 → 36**

**a /= 4; // a = a / 4 → 3**

**5. Increment & Decrement Operators**

**int x = 4;**

**cout << x++ << endl; // 4 (then x=5)**

**cout << ++x << endl; // 6**

**cout << x-- << endl; // 6 (then x=5)**

**cout << --x << endl; // 4**

**6. Bitwise Operators**

**int a = 5, b = 3; // 5 = 0101, 3 = 0011**

**cout << (a & b) << endl; // AND → 1**

**cout << (a | b) << endl; // OR → 7**

**cout << (a ^ b) << endl; // XOR → 6**

**cout << (a << 1) << endl; // Left shift → 10**

**cout << (a >> 1) << endl; // Right shift → 2**

**7. Special Operators**

**· sizeof → size of data type.**

**· typeid → type information.**

**· comma ( , ) → multiple expressions.**

**cout << sizeof(int) << endl; // usually 4**

**int x, y;**

**y = (x = 10, x + 5); // first x=10, then y=15**

**cout << y << endl;**

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

**Constants are values that cannot be changed once defined.**

**· Declared using const keyword.**

**· Used when we want a fixed value throughout the program (like pi = 3.14159, year = 2025).**

**· Example:-**

**#include <iostream>**

**using namespace std;**

**int main() {**

**const double PI = 3.14159; // constant**

**int r = 5;**

**double area = PI \* r \* r; // using constant**

**cout << "Area: " << area;**

**}**

* 1. **Literals:-**

**A literal is a fixed value directly written in the code. It represents a constant that does not change during execution.**

* **To assign hardcoded values to variables.**
* **To express constants like numbers, characters, strings, etc.**
* **Used in initialization, comparisons, and output formatting.**
* **Example:-**

**int x = 10; // 10 is an integer literal**

**char ch = 'A'; // 'A' is a character literal**

**float pi = 3.14f; // 3.14 is a floating-point literal**

* **Key Points are:-**

1. **Constants: Named fixed values; cannot be changed after declaration.**
2. **Literals: Fixed values written directly in the code; can be assigned to variables or used directly.**
3. **Using both improves code readability, safety, and maintainability.**
4. Control Flow Statements:-
   1. **What are conditional statements in C++? Explain the if else and switch statements.**

**Ans.**

* 1. **Conditional Statements:-A conditional statement evaluates a condition (an expression that results in true or false) and executes a block of code depending on that result.**

**Types of conditional statements in C++:**

1. **if statement**
2. **if-else statement**
3. **if-else if-else ladder**
4. **switch statement**
   1. **If – else statement:- Used when you want to check a condition and run code accordingly.**

**· Example:-**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int age = 18;**

**if (age >= 18) {**

**cout << "You are an Adult.";**

**} else {**

**cout << "You are a Minor.";**

**}**

**return 0;**

**}**

* 1. **Switch statement:-Used when you have multiple cases to check (instead of many if-else).**

**Works only with integers, characters, and enums (not with strings or floats).**

**· Example:-**

**include <iostream>**

**using namespace std;**

**int main() {**

**int day = 3;**

**switch (day) {**

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

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

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

**default: cout << "Invalid Day";**

**}**

**return 0;**

**}**

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | for Loop | while Loop | do-while Loop |
| ****Control Type**** | **Entry-controlled** | **Entry-controlled** | **Exit-controlled** |
| ****Condition Checked**** | **Before each iteration** | **Before each iteration** | **After each iteration** |
| ****Minimum Executions**** | **0 (if condition is false initially)** | **0 (if condition is false initially)** | **1 (always executes at least once)** |

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

**Syntax & Examples:-**

**for Loop:-Used when you know how many times to repeat.**

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

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

**}**

**while Loop:-Used when the condition is evaluated before each iteration.**

**int i = 0;**

**while (i < 5) {**

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

**i++;**

**}**

**do-while Loop:-Used when the loop must run at least once.**

**int i = 0;**

**do {**

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

**i++;**

**} while (i < 5);**

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

**Ans.**

1. **break Statement**

* **Purpose: Immediately exits the loop.**
* **Use case: When a condition is met and no further iteration is needed.**

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

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

**break; // Loop stops when i is 5**

**}**

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

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

1. **continue Statement**

* **Purpose: Skips the current iteration and moves to the next.**
* **Use case: When you want to ignore specific cases but continue looping.**

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

**if (i == 2){**

**continue; // Skips printing 2**

**}**

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

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

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

**Ans.** **Nested Control Structures:- Nested control structures are control statements placed inside other control statements—like a loop inside a loop, or an if inside a for. They allow for multi-level decision-making or iteration, essential for tasks like matrix traversal, pattern printing, or complex condition checks.**

**Example: Pattern Printing with Nested Loops**

**for (int i = 1; i <= 3; i++) { // Outer loop: rows**

**for (int j = 1; j <= i; j++) { // Inner loop: columns**

**cout << "\* ";**

**}**

**cout << endl;**

**}//Output: \***

**\* \***

**\* \* \***

* **The outer loop controls the number of lines.**
* **The inner loop prints stars based on the current line number.**

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

**Ans.** **In C++, a function is a block of code that performs a specific task and can be reused multiple times. Functions help in reducing code duplication and improving readability.**

**1. Function Declaration (Prototype)**

* **Tells the compiler about the function’s name, return type, and parameters (before main).**

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

**2. Function Definition**

* **Contains the actual code (body) of the function.**

**int add(int a, int b) { // definition**

**return a + b;**

**}**

**3. Function Calling**

* **When the function is used inside main() or another function.**

**#include <iostream>**

**using namespace std;**

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

**int main() {**

**int sum = add(5, 3); // calling**

**cout << "Sum = " << sum;**

**return 0;**

**}**

**int add(int a, int b) { // definition**

**return a + b;**

**}**

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

**Ans. Scope of Variables in C++**

**The scope of a variable defines the part of the program where the variable can be accessed or used.**

**There are mainly two types:**

1. **Local Scope → Variable declared inside a function/block.**
2. **Global Scope → Variable declared outside all functions (accessible everywhere).**

| **Feature** | **Local Variable** | **Global Variable** |
| --- | --- | --- |
| **Declaration** | **Declared inside a function/block.** | **Declared outside all functions.** |
| **Scope/Access** | **Accessible only within that function/block.** | **Accessible from any function in the program.** |
| **Lifetime** | **Exists only while the function is running.** | **Exists throughout the program execution.** |
| **Default Value** | **Contains garbage value (if not initialized).** | **Initialized to zero by default.** |
| **Memory Allocation** | **Stored in stack memory.** | **Stored in data segment memory.** |

* 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 is often used to solve problems that can be broken into smaller subproblems (like factorial, Fibonacci, searching, etc.).**

**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**

* **Recursion = function calling itself.**
* **Needs a base case (stopping condition) to avoid infinite calls.**
* **Useful for problems like factorial, Fibonacci, and tree traversal.**
  1. **What are function prototypes in C++? Why are they used?**

**Ans. Function Prototypes:-**

**A function prototype is just a declaration of a function, written before it is used in the program. It tells the compiler:**

**· Function name**

**· Return type**

**· Number and type of parameters**

**It does not contain the function body.**

**· Example:-**

**#include <iostream>**

**using namespace std;**

**// Function prototype**

**int add(int x, int y);**

**int main() {**

**int result = add(5, 3); // Function call**

**cout << "Sum = " << result;**

**return 0;**

**}**

**// Function definition**

**int add(int x, int y) {**

**return x + y;**

**}**

* 1. Arrays and Strings:-

1. **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 the contiguous memory locations.**

**It allows us to store and access the multiple values using a single name with the help of an index.  
Types of Array:-**

* 1. **Single-Dimensional Array(1D):-**
* **Stores data in a single row.**
* **Accessed using one index.**
* **Example :-**

**#include<iostream>**

**using namespace std;  
int main()  
{**

**int arr[5]={19,21,23,25,27}; //1D Array..  
cout << “Element[0]: “ <<arr[0];// Accessing element**

**}**

* 1. **Multi-Dimensional Array:-**
* Stores data in rows and columns (like a table).
* Accessed using two or more indices.
* Example:-

**#include <iostream>**

**using namespace std;**

**int main() {**

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

**cout << "Element at row 1, col 2 = " << matrix[0][1];**

**return 0;**

**}**

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

**Ans. A string is a sequence of characters used to represent text.**

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

**1. C - Style Strings:-**

**· Defined using char arrays.**

**· Always end with a null character ('\0').**

**· Functions from <cstring> (like strcpy, strlen, strcmp) are used.**

**· Example:-**

**#include <iostream>**

**#include <cstring>**

**using namespace std;**

**int main() {**

**char name[20] = "Jainam"; // C-style string**

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

**cout << "Length: " << strlen(name) << endl; // string length**

**return 0;**

**}**

**2. C++ Strings (using <string> class)**

**Easier and safer than C-style strings.**

**Provides many built-in functions like .length(), .append(), .substr(), etc.**

**· Example:**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**int main() {**

**string str1 = "Hello";**

**string str2 = "World";**

**string result = str1 + " " + str2; // concatenation**

**cout << "Combined: " << result << endl;**

**cout << "Length: " << result.length() << endl;**

**return 0;**

**}**

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

**and 2D arrays.**

**Ans.**  **Array initialization:-**

**In C++, arrays can be initialized (given values) at the time of declaration.**

**Example:- Single dimensional array:**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int numbers[5] = {10, 20, 30, 40, 50}; // 1D array initialization**

**cout << "First element: " << numbers[0] << endl;**

**cout << "Last element: " << numbers[4] << endl;**

**return 0;**

**}**

**Example:- Multi Dimensional array:-**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int matrix[2][3] = { {1, 2, 3}, {4, 5, 6} }; // 2x3 array**

**cout << "Element at [1][2]: " << matrix[1][2] << endl; // 6**

**return 0;**

**}**

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

**Ans. In C++, we can handle strings in two ways:**

**1. C-style strings (character arrays with <cstring>)**

**2. C++ string class (from <string>)**

**The C++ string class is much easier and safer, so we’ll focus more on that.**

**In short:-**

**· Strings are used to store text.**

**· With C++ string class, operations become very easy (just like in Python/Java).**

1. Introduction to Object-Oriented Programming:-
2. **Explain the key concepts of Object-Oriented Programming**

**(OOP).**

**Ans. Object-Oriented Programming is a way of writing programs by organizing data and functions into objects. The main concepts of OOP are:**

**1. Class:-**

**· A class is like a blueprint or template.**

**· It defines how objects will look and behave.**

**· It contains data (variables) and methods (functions).**

**· Example:-**

**class Car {**

**public:**

**string brand;**

**void drive() {**

**cout << "Car is driving" <<endl;**

**}**

**};**

**2. Object:-**

**· An object is a real thing created from a class.**

**· Example: if class = blueprint of a car, then object = actual car.**

**· Example:-**

**Car myCar; // object of class Car**

**myCar.brand = "BMW";**

**myCar.drive();**

**3. Encapsulation:-**

**· Wrapping data and functions together in a class.**

**· It helps to protect data using access specifiers (private, public).**

**· Example:-**

**class Student {**

**private:**

**int marks; // hidden**

**public:**

**void setMarks(int m) { marks = m; }**

**int getMarks() { return marks; }**

**};**

**4. Abstraction:-**

**· Showing only essential details and hiding background details.**

**· Example: You use a TV remote without knowing the internal circuits.**

**· Example:-**

**class Remote {**

**public:**

**void powerOn() {**

**cout << "TV On" << endl;**

**} // hides inner details**

**};**

**5. Inheritance:-**

**· One class can reuse features of another class.**

**· Promotes code reusability.**

**· Example:-**

**class Animal {**

**public:**

**void eat() {**

**cout << "Eating" << endl;**

**}**

**};**

**class Dog : public Animal {**

**public:**

**void bark() {**

**cout << "Barking" << endl;**

**}**

**};**

**6. Polymorphism:-**

**· One name, many forms (same function behaves differently).**

**· Types:**

**· Compile-time (function overloading, operator overloading)**

**· Run-time (function overriding using inheritance)**

**· Example:-**

**class Shape {**

**public:**

**void area(int r) {**

**cout << "Circle area: " << 3.14\*r\*r << endl;**

**}**

**void area(int l, int b) {**

**cout << "Rectangle area: " << l\*b << endl;**

**}**

**};**

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

**Ans. ✅ Class**

**A class is a blueprint or template for creating objects. It defines the data members (variables) and member functions (methods) that the objects created from the class will have.**

**✅ Object**

**An object is an instance of a class. When a class is defined, no memory is allocated until an object of that class is created.**

**🔹 Example of Class and Object:-**

**#include <iostream>**

**using namespace std;**

**// Define a class**

**class Car {**

**public:**

**// Data members**

**string brand;**

**int year;**

**// Member function**

**void displayInfo() {**

**cout << "Brand: " << brand << ", Year: " << year << endl;**

**}**

**};**

**int main() {**

**// Create an object of the class**

**Car car1:**

**// Assign values to the object's members**

**car1.brand = "Toyota";**

**car1.year = 2020;**

**// Call a member function using the object**

**car1.displayInfo();**

**return 0;**

**}//Output:- Brand: Toyota, Year: 2020**

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

**Ans. Inheritance:-**

**Inheritance is an OOP concept in C++ where one class (derived/child class) acquires the properties and behaviors (data members & member functions) of another class (base/parent class).**

**👉 It promotes code reusability and represents "is-a" relationship.**

**Syntax:- class Derived : access-specifier Base {**

**// additional members**

**};**

**Example:-**

**#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 << "The dog barks." << endl;**

**}**

**};**

**int main() {**

**Dog d;**

**d.eat(); // inherited from Animal**

**d.bark(); // Dog's own function**

**return 0;**

**}**

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

**Ans. Encapsulation:- Encapsulation is a fundamental object-oriented concept that means wrapping data and the functions that operate on it into a single unit, typically a class, and restricting direct access to some of the object's components.**

**✅ How it's achieved in classes:**

* **Access Specifiers:**
  + **private: hides data from outside access**
  + **public: exposes methods to interact with the data**
  + **protected: allows access within the class and its subclasses**
* **Data Hiding:**
  + **Internal variables are kept private**
  + **External access is provided through public methods (getters/setters)**

**This ensures data integrity, promotes modularity, and makes the code easier to maintain and debug. It's like locking your valuables in a safe and only giving access through a secure keypad.**

**….**