Name :- Gajera Pal

**Introduction to C++**

**1. Key Differences Between Procedural Programming (POP) and Object-Oriented Programming (OOP)**

**ANS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | **Feature** |  |  | | --- | |  | | **POP** | **OOP** |
| **Approach** | |  | | --- | | Follows a top-down approach |  |  | | --- | |  | | |  | | --- | | Follows a bottom-up approach |  |  | | --- | |  | |
| **Structure** | |  | | --- | | Based on functions and procedures |  |  | | --- | |  | | |  | | --- | | Based on objects and classes |  |  | | --- | |  | |
| **Data Security** | Less secure, as data is globally accessible | More secure, as data is encapsulated within objects |
| **Examples** | C, Pascal | C++, Java, Python (OOP) |

**2. Advantages of OOP Over POP**

**ANS**

1. **Encapsulation** – Protects data by restricting direct access.
2. **Inheritance** – Enables code reuse and hierarchy creation.
3. **Polymorphism** – Enhances flexibility by allowing multiple behaviors.
4. **Abstraction** – Hides complexity and enhances modularity.
5. **Reusability** – Encourages the use of existing code.
6. **Scalability** – Facilitates easy expansion and maintenance.

**3. Steps to Set Up a C++ Development Environment**

**ANS**

1. **Download & Install Compiler** – Install GCC (MinGW) or MSVC.
2. **Choose an IDE** – Use Code::Blocks, Dev-C++, or Visual Studio.
3. **Set Up Environment Variables** – Configure system paths for compiler access.
4. **Write & Save a C++ Program** – Create a .cpp file.
5. **Compile & Run** – Use the terminal/IDE to compile (g++ program.cpp -o program) and execute (./program).

**4. Main Input/Output Operations in C++ (Examples)**

1. **Input (ci**int num;

std::cin >> num;

1. **Output (cout)** – Used to display output.

std::cout << "Hello, World!";

1. **File Input/Output (ifstream / ofstream)** – Used to read/write files.

std::ofstream file("example.txt");

file << "Writing to file!";

file.close();

**Variables, Data Types, and Operators**

**1. Different Data Types in C++ (With Examples)**

**ANS**

C++ provides several data types categorized as follows:

* **Basic Data Types**:
  + int (Integer): int num = 10;
  + float (Floating point): float pi = 3.14;
  + double (Double precision float): double pi = 3.14159;
  + char (Character): char letter = 'A';
  + bool (Boolean): bool isTrue = true;
* **Derived Data Types**:
  + array: int arr[5] = {1, 2, 3, 4, 5};
  + pointer: int \*ptr;
  + reference: int &ref = num;
* **User-defined Data Types**:
  + struct, class, enum, union

2. **Implicit vs. Explicit Type Conversion in C++**

**ANS**

int a = 10;

double b = a; // Implicit conversion (int → double)

double x = 5.75;

int y = (int)x; // Explicit conversion (double → int)

**3. Types of Operators in C++ (With Examples)**

**ANS**

**Arithmetic Operators** (+, -, \*, /, %):

int sum = 5 + 3; // Output: 8

**Relational Operators** (==, !=, >, <, >=, <=):

if (5 > 3) { /\* True \*/ }

**Logical Operators** (&&, ||, !):

if (a > 0 && b > 0) { /\* True if both are positive \*/ }

**Bitwise Operators** (&, |, ^, ~, <<, >>):

int result = 5 & 3; // Bitwise AND

**Assignment Operators** (=, +=, -=, \*=, /=, %=, etc.):

a += 5; // Equivalent to: a = a + 5;

**Ternary Operator** (?:):

int min = (a < b) ? a : b;

**Increment/Decrement Operators** (++, --):

int x = 5;

x++; // x becomes 6

**4. Constants and Literals in C++**

**ANS**

* **Constants**: Fixed values that do not change during execution.
  + Declared using const or #define.

const float PI = 3.14159;

* **Literals**: Directly specified constant values in code.
  + Integer: 10, -5
  + Floating-point: 3.14
  + Character: 'A'
  + String: "Hello"
  + Boolean: true, false

**Control Flow Statements**

**1. Conditional Statements in C++**

**ANS**

**if-else Statement**: Executes different blocks of code based on conditions.

int num = 10;

if (num > 0) {

std::cout << "Positive";

} else {

std::cout << "Negative";

}

**switch Statement**: Used for multiple conditions.

int choice = 2;

switch (choice) {

case 1: std::cout << "One"; break;

case 2: std::cout << "Two"; break;

default: std::cout << "Invalid";

}

2. **Difference Between for, while, and do-while Loops**

**ANS**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Loop Type** |  |  | | --- | |  | | Syntax | |  | | --- | | **Condition Check** |  |  | | --- | |  | | | **Use Case** | | --- |  |  | | --- | |  | |
| For | |  | | --- | | for(init; condition; update) |  |  | | --- | |  | | |  | | --- | | Before execution |  |  | | --- | |  | | |  | | --- | | When loop runs a known number of times |  |  | | --- | |  | |
| While | |  | | --- | | while(condition) |  |  | | --- | |  | | |  | | --- | | Before execution |  |  | | --- | |  | | |  | | --- | | When loop runs based on a condition |  |  | | --- | |  | |
| do-while | |  | | --- | | do { } while(condition); |  |  | | --- | |  | | |  | | --- | | After execution |  |  | | --- | |  | | Ensures at least one execution |

**3. Break and Continue Statements in Loops**

**ANS**

**Break**

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

if (i == 3) break; // Stops loop when i = 3

std::cout << i << " ";

}

**Continue**

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

if (i == 3) continue; // Skips printing 3

std::cout << i << " ";

}

**4. Nested Control Structures**

**ANS**

Code:

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

for (int j = 1; j <= 3; j++) {

std::cout << i << "," << j << " ";

}

std::cout << std::endl;

}

Output:

1,1 1,2 1,3

2,1 2,2 2,3

3,1 3,2 3,3

**Functions and Scope**

**1. Function in C++**

**ANS**

**Function Declaration**

int add(int, int); // Declaration

**Function Definition**

int add(int a, int b) {

return a + b;

}

**Function Call**

int result = add(5, 3); // Calling function

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

**ANS**

**Local Scope:**

void func() {

int x = 10; // Local variable

}

**Global Scope:**

int y = 20; // Global variable

void func() {

std::cout << y; // Accessible here

}

**3. Recursion in C++**

**ANS**

int factorial(int n) {

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

return n \* factorial(n - 1); // Recursive call

}

int main() {

std::cout << factorial(5); // Output: 120

}

**4. Function Prototypes in C++**

**ANS**

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

int main() {

std::cout << add(3, 4);

}

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

return a + b;

}

**Arrays and Strings**

**1. Arrays in C++**

**ANS**

**Single-Dimensional Array**

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

**Multi-Dimensional Array**:

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

2. **String Handling in C++**

**ANS**

#include <iostream>

#include <string>

int main() {

std::string s = "Hello, World!";

std::cout << s;

}

**3. Array Initialization in C++**

**ANS**

**1D Array Initialization**:

int arr1[] = {1, 2, 3, 4, 5};

int arr2[5] = {0}; // All elements set to 0

**2D Array Initialization**:

int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};

**4. String Operations and Functions in C++**

**ANS**

**Concatenation**:

std::string s1 = "Hello", s2 = " World";

std::string result = s1 + s2;

**Length of a string**:

int len = s1.length();

**Substring Extraction**:

std::string sub = s1.substr(0, 3); // "Hel"

**String Comparison**:

if (s1 == s2) { std::cout << "Equal"; }

**Introduction to OOP**

**1. Key Concepts of Object-Oriented Programming (OOP)**

**ANS**

1. **Encapsulation** – Hiding data using access specifiers.
2. **Abstraction** – Hiding complex implementation details.
3. **Inheritance** – Reusing properties and behaviors from a base class.
4. **Polymorphism** – Using a single interface for multiple behaviors.

2. **Classes and Objects in C++**

**ANS**

**Class**: A blueprint for creating objects.

**Object**: An instance of a class.

class Car {

public:

std::string brand;

void display() {

std::cout << "Brand: " << brand;

}

};

int main() {

Car myCar;

myCar.brand = "Tesla";

myCar.display();

}

**3. Inheritance in C++**

**ANS**

class Animal {

public:

void makeSound() { std::cout << "Animal Sound"; }

};

class Dog : public Animal {

public:

void bark() { std::cout << "Bark"; }

};

int main() {

Dog d;

d.makeSound(); // Inherited function

d.bark();

}

**4. Encapsulation in C++**

**ANS**

class BankAccount {

private:

double balance;

public:

void setBalance(double amt) { balance = amt; }

double getBalance() { return balance; }

};

int main() {

BankAccount acc;

acc.setBalance(5000);

std::cout << acc.getBalance();

}