**Module – 3**

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

Procedural vs Object-Oriented Programming (OOP)

| **Procedural Programming (POP)** | **Object-Oriented Programming (OOP)** |
| --- | --- |
| Focus on functions | Focus on objects |
| Data is not secure (global access) | Data is secure (using private/public keywords) |
| Example: C | Example: C++, Java |
| Functions perform actions | Objects hold data + functions (methods) |
| Hard to maintain as program grows | Easy to maintain, reuse and extend |

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

Advantages of OOP over POP

1. **Reusability**: Classes can be reused.
2. **Security**: Data is hidden using **private** members.
3. **Scalability**: Easy to manage large programs.
4. **Modularity**: Code is divided into objects.

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

Steps to Setup C++ Development Environment

1. Install C++ Compiler (like **GCC** or **MinGW**).
2. Install IDE (like **Code::Blocks**, **Dev C++**, **Visual Studio**).
3. Set **environment variables** (for Windows).
4. Write code in IDE → Compile → Run.

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

Input/Output in C++ (Examples)

#include<iostream>

using namespace std;

int main() {

int x;

cout << "Enter number: "; // Output

cin >> x; // Input

cout << "You entered: " << x;

return 0;

}

**1. Data Types in C++**

| **Type** | **Example** |
| --- | --- |
| int | int x = 5; |
| float | float y = 3.14; |
| double | double z = 6.022; |
| char | char ch = 'A'; |
| bool | bool flag = true; |
| string | string s = "Hello"; |

**2. Implicit vs Explicit Type Conversion**

* **Implicit**: Auto conversion

int x = 5;

float y = x; // int to float automatically

* **Explicit**: Manual conversion

float x = 5.6;

int y = (int)x; // float to int manually

**3. Operators in C++ (Examples)**

| **Type** | **Example** |
| --- | --- |
| Arithmetic | +, -, \*, /, % |
| Relational | ==, !=, >, < |
| Logical | &&, ||, ! |
| Assignment | =, +=, -= |
| Increment/Decrement | ++, -- |

**4. Constants and Literals**

* **Constants**: Value can't change.

const int x = 10;

* **Literals**: Fixed values in code.

int x = 5; // 5 is literal

**1. Conditional Statements**

* **if-else**:

if(a > b)

cout << "A is big";

else

cout << "B is big";

* **switch**:

switch(choice) {

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

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

}

**2. Loops Differences**

| **Loop** | **Condition Check** | **Runs at least once?** |
| --- | --- | --- |
| for | Start | No |
| while | Before | No |
| do-while | After | Yes |

**3. break & continue**

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

if(i==3) continue; // skips 3

cout<<i;

}

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

if(i==3) break; // stops at 3

cout<<i;

}

**4. Nested Control Structure**

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

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

cout << i << j << " ";

}

}

**1. Function in C++**

* **Declaration**: Before main

int add(int, int);

* **Definition**:

int add(int a, int b){

return a + b;

}

* **Calling**:

int c = add(2, 3);

**2. Scope of Variables**

* **Local**: Inside function
* **Global**: Outside all functions

**3. Recursion**

A function calling itself:

int fact(int n){

if(n==0) return 1;

else return n \* fact(n-1);

}

**4. Function Prototype**

Tells compiler about function before using it:

int sum(int, int); // Prototype

**1. Arrays**

* **1D Array**: int a[5];
* **2D Array**: int b[3][3];

**2. String Handling**

string s = "Hello";

cout << s.length();

**3. Array Initialization**

int a[3] = {1,2,3}; // 1D

int b[2][2] = {{1,2},{3,4}}; // 2D

**4. String Functions**

| **Function** | **Example** |
| --- | --- |
| length() | s.length(); |
| append() | s1.append(s2); |
| compare() | s1.compare(s2); |

**1. OOP Concepts**

* **Class**
* **Object**
* **Encapsulation**
* **Inheritance**
* **Polymorphism**
* **abstraction**

**2. Classes & Objects**

class Car {

public:

int speed;

void run(){ cout << "Running"; }

};

Car c1; // Object

c1.run();

**3. Inheritance**

class A {

public:

void display(){ cout<<"A"; }

};

class B : public A {}; // B inherits A

B obj;

obj.display(); // A

**4. Encapsulation**

Hiding data using private:

class Student {

private:

int marks;

public:

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

int getMarks(){ return marks; }

};