# MODULE :-3 INTRODUCTION TO OOPS PROGRAMMING

# [THEORY EXERCISE]

## 1. Introduction to C++

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

   **Ans :-**

| Feature | Procedural Programming (POP) | Object-Oriented Programming (OOP) |
|---|---|---|
| Approach | Follows a top-down approach | Follows a bottom-up approach |
| Structure | Program is divided into functions | Program is divided into objects and classes |
| Data Handling | Data is exposed and shared among functions | Data is hidden inside objects (encapsulation) |
| Reusability | Limited code reusability | Promotes reusability through inheritance |
| Security | Less secure (no access modifiers) | More secure (data hiding and abstraction) |
| Example Languages | C, Pascal, FORTRAN | C++, Java, Python (OOP style) |
| Real-World Mapping | Difficult to map to real-world entities | Maps directly to real-world objects |

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

**Ans :-**

**Modularity**

- **Programs are divided into objects which are easier to manage.**

**Reusability**

- **Classes can be reused in other programs using inheritance.**

**Data Security**

- **Data is hidden using private access, and access is given through methods only.**

**Easy to Maintain**

- **Programs are easy to update, test, and debug.**

**Real-world Modeling**

- **Objects in programs represent real-world things (like Car, Employee).**

**Extensibility**

- **New features can be added easily using inheritance and polymorphism.**

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

**Ans :- To write and run C++ programs, follow these steps:**

◇ **Step 1: Install a C++ Compiler**

**1.For Windows: Use MinGW, Code::Blocks, or Dev-C++**

**2.For Linux: Install g++ using terminal**

```
sudo apt install g++
```

**Step 2: Choose an IDE or Editor**

**1.IDEs: Code::Blocks, Dev-C++, Visual Studio**

**2.Editors: VS Code, Notepad++**

◇ **Step 3: Write Your Program**

**Save your code with .cpp extension.**

◇ **Step 4: Compile the Program**

**Using terminal:**

```
g++ hello.cpp -o hello
```

◇ **Step 5: Run the Program**

```
./hello
```

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

**Ans :- C++ uses cin for input and cout for output, from the iostream header.**

- **Input (cin)**

**Used to get values from the user.**

```
int age;
cout << "Enter your age: ";
cin >> age;
```

- **Output (cout)**

**Used to print messages or values.**

```
cout << "You entered age: " << age;
```

- **Full Example:**

```
#include <iostream>
using namespace std;

int main() {
    int age;
    cout << "Enter your age: ";
    cin >> age;
    cout << "Your age is " << age;
    return 0;
}
```

## 2. Variables, Data Types, and Operators

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

**Ans :- C++ has two main types of data types:**

**1. Built-in (Primitive)**

**2. User-defined (like struct, class)**

- **Built-in Data Types:**

| Data Type | Description | Example |
|---|---|---|
| int | Stores integers | int age = 20; |
| float | Stores decimal numbers (single precision) | float price = 10.5; |
| double | Stores decimal numbers (double precision) | double pi = 3.14159; |
| char | Stores a single character | char grade = 'A'; |
| bool | Stores true or false | bool pass = true; |
| void | No value (used for functions) | void show(); |

- **User-Defined Data Types:**

**struct, class, enum**

- **Derived Data Types:**

**Arrays, Pointers, References**

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

**Ans :-** ◇ **Implicit Type Conversion (Type Promotion):**

**1.Done automatically by the compiler.**

**2.Happens when we assign one data type to another type with higher range.**

```
int a = 5;
float b = a;  // int is automatically converted to float
```

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

- **Done manually by the programmer.**
- **Use casting syntax to convert one type to another.**

**Example:**

```
float f = 5.67;
int a = (int)f;  // float manually converted to int
```

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

**Ans :- C++ provides many types of operators:**

| Operator Type | Description | Example |
|---|---|---|
| Arithmetic | +, -, *, /, % | a + b, a * b |
| Relational | ==, !=, >, <, >=, <= | a > b, a == b |
| Logical | &&, \|\|, ! | (a > b) && (a < c) |
| Assignment | =, +=, -=, *=, /= | a = 5;, a += 2; |
| Increment/Decrement | ++, -- | a++, --b |
| Bitwise | &, | , ^, ~, <<, >> |
| Conditional (Ternary) | ?: | (a > b) ? a : b |
| Sizeof | Returns size of a variable | sizeof(int) |

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

**Ans :-** ◇ **Constants:**

**Values that do not change during program execution.**

**Declared using the const keyword.**

**Example:**

```
const float PI = 3.14;
```

◇ **Literals:**

- **Fixed values written directly in code.**

**Types of Literals:**

| Type | Example |
|---|---|
| Integer Literal | 10, -99 |
| Float Literal | 3.14, 0.005 |
| Character Literal | 'A', '@' |
| String Literal | "Hello", "C++" |
| Boolean Literal | true, false |

**Use:**

**Constants and literals make code clear, safe, and easy to maintain. They prevent accidental changes to important values.**

## 3. Control Flow Statements

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

**Ans :- Conditional statements are used to make decisions in a program. They help the program choose different paths based on conditions.**

- **if-else Statement:**

**Used when you want to check a condition and execute code accordingly.**

**Syntax:**

```
if (condition) {
    // code if true
} else {
    // code if false
}
```

**Example:**

```
int age = 18;
if (age >= 18) {
    cout << "Eligible to vote.";
} else {
    cout << "Not eligible.";
}
```

- **switch Statement:**

**Used when you want to check multiple possible values of a variable.**

**Syntax:**

```
switch (expression) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;
    default:
        // code if no match
}
```

**Example:**

```
int day = 2;
switch (day) {
    case 1: cout << "Monday"; break;
    case 2: cout << "Tuesday"; break;
    default: cout << "Other day";
}
```

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

**Ans :- Loops are used to repeat a block of code multiple times.**

| Loop Type | Description | Syntax Example | Executes at least once? |
|---|---|---|---|
| for | Use when you know how many times to repeat | for (int i=0; i<5; i++) | ✗ No |
| while | Use when you want to repeat while a condition is true | while (i < 5) | ✗ No |

| Loop Type | Description | Syntax Example | Executes at least once? |
|---|---|---|---|
| do-while | Same as while but runs at least once | do { } while (i < 5); | ☑ Yes |

**Example:**

```
// for loop
for (int i = 1; i <= 3; i++) {
    cout << i << " ";
}
```

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

**Ans:-** ◇ **break Statement:**

**Used to exit the loop immediately.**

**Example:**

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) break;
    cout << i << " ";
}
// Output: 1 2
```

◇ **continue Statement:**

**Skips the current iteration and continues with the next loop cycle.**

**Example:**

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) continue;
    cout << i << " ";
}
// Output: 1 2 4 5
```

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

**Ans :- Nested control structures mean using one control statement inside another.**

**You can have:**

**1. if inside if (nested if)**

**2. loops inside loops (nested loops)**

**3.if inside loops, and vice versa**

**Example: Nested for loop**

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 2; j++) {
        cout << "i=" << i << ", j=" << j << endl;
    }
}
```

**Output:**

```
i=1, j=1
i=1, j=2
i=2, j=1
i=2, j=2
i=3, j=1
i=3, j=2
```

**Nested structures are useful in patterns, tables, matrix programs, etc.**

## 4. Functions and Scope

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

**Ans:- A function in C++ is a block of code that performs a specific task. It helps to organize code, avoid repetition, and make programs easier to read and maintain.**

- **3 Main Parts of a Function:**

1. **Function Declaration (Prototype)**
    - **Tells the compiler about the function's name, return type, and parameters.**
    - **Written before main() or in header files.**

**Example:**

```
int add(int a, int b);  // Declaration
```

2. **Function Definition**

- **Contains the actual code (logic) of the function.**

**Example:**

```
int add(int a, int b) {
    return a + b;
}
```

3. **Function Call**

- **Used to execute the function.**

**Example:**

```
int result = add(5, 3);
```

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

**Ans :- Scope refers to the area of the program where a variable can be accessed or used.**

◇ **Local Scope:**

- **Variable declared inside a function or block.**
- **Only accessible within that block.**

**Example:**

```
void show() {
    int x = 10;  // local variable
    cout << x;
}
```

◇ **Global Scope:**

- **Variable declared outside all functions.**
- **Accessible by all functions in the same file.**

**Example:**

```
int x = 100;  // global variable

void show() {
    cout << x;
}
```

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

**Ans :- Recursion means a function calling itself to solve a smaller version of the problem.**

**It's useful in:**

- **Factorial**
- **Fibonacci**
- **Tree traversals**
- **Backtracking**

**Example: Factorial using recursion**

```
int factorial(int n) {
    if (n == 0) return 1;
    else return n * factorial(n - 1);
}
```

**Explanation:**

- **If n = 3, it works like:**
  **3 * factorial(2) → 3 * 2 * factorial(1) → 3 * 2 * 1 * factorial(0) → returns 1**

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

**Ans . :- A function prototype is a declaration of a function before it is used.**
**It tells the compiler:**

- **Function name**
- **Return type**
- **Number and type of arguments**

◇ **Syntax:**

```
return_type function_name(parameter_list);
```

◇ **Example:**

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

◇ **Why Use Function Prototypes?**

- **Helps the compiler check for correct function usage**
- **Supports modular programming (functions can be written after main)**
- **Reduces errors due to wrong function calls**

## 5. 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 at contiguous memory locations.**

◇ **Features:**

- **Fixed size**
- **Elements are accessed using index (starts from 0)**

---

◇ **Single-Dimensional Array (1D):**

- **Stores elements in a single row (like a list)**

**Example:**

```
int marks[5] = {80, 90, 85, 75, 95};
```

- **marks[0] is 80, marks[1] is 90, etc.**

◇ **Multi-Dimensional Array (2D or more):**

- **Stores data in tables (rows and columns)**

**Example (2D Array):**

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

- **Access element like matrix[1][2] → 6**

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

**Ans :- C++ supports two types of strings:**

- **(a) C-style strings (char arrays):**
  - **An array of characters ending with '\0' (null character)**

**Example:**

```
char name[10] = "Jay";
cout << name;
```

- **(b) C++ strings (using <string> library):**
  - **Easier and safer than C-style strings.**

**Example:**

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string name = "Jay";
    cout << "Name is " << name;
    return 0;
}
```

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

**Ans :-** **1D Array Initialization:**

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

**Or**

```
int nums[] = {10, 20, 30};
```

◇ **2D Array Initialization:**

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

**Accessing:**

```
cout << table[1][2];  // Output: 6
```

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

**Ans :- When using C++ strings (<string>), we can perform many operations easily.**

◇ **Common String Functions and Operations:**

| Function/Operation | Description | Example |
|---|---|---|
| length() or size() | Returns number of characters | str.length() |
| + | Concatenates strings | "Hello " + "World" |
| ==, != | Compares strings | if (str1 == str2) |
| substr(start, len) | Gets part of string | str.substr(1,3) |
| append() | Adds string at end | str1.append("test") |
| at(index) | Gets character at index | str.at(0) |
| empty() | Checks if string is empty | str.empty() |
| find("word") | Finds position of word | str.find("a") |

**Example:**

```
string str = "Hello";
string str2 = "World";
string result = str + " " + str2;  // "Hello World"
cout << result.length();           // Output: 11
```

## 6. Introduction to Object-Oriented Programming

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

**Ans :-** **OOP (Object-Oriented Programming) is a method of programming where the focus is on objects and classes instead of just functions and logic. It helps organize code in a modular and reusable way.**

- **Main OOP Concepts (4 Pillars):**

| Concept | Description |
|---|---|
| Encapsulation | Hiding internal details and only exposing necessary parts (using classes) |
| Abstraction | Showing only essential features and hiding background details |
| Inheritance | Reusing existing class features in a new class |
| Polymorphism | One function or operator behaves differently in different situations |

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

**Ans :-** ◇ **Class:**

**A class is a blueprint or template for creating objects. It defines properties (data) and behaviors (functions).**

◇ **Object:**

**An object is an actual instance of a class. It represents a real-world entity.**

**Example:**

```cpp
#include <iostream>
using namespace std;

class Car {
public:
    string brand;
    int year;

    void showDetails() {
        cout << brand << " - " << year << endl;
    }
};

int main() {
    Car myCar;              // Object creation
    myCar.brand = "Toyota";
    myCar.year = 2020;
    myCar.showDetails();    // Calling class function
    return 0;
}
```

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

**Ans :-** **Inheritance means one class (child) can acquire the properties and functions of another class (parent).**
**It allows code reuse and is a key feature of OOP.**

- **Types of Inheritance:**
  - **Single**
  - **Multiple**
  - **Multilevel**
  - **Hierarchical**
  - **Hybrid**

**Example: Single Inheritance**

```cpp
#include <iostream>
using namespace std;

class Animal {
public:
    void eat() {
        cout << "Eating..." << endl;
    }
};

class Dog : public Animal {
public:
    void bark() {
        cout << "Barking..." << endl;
    }
};

int main() {
    Dog d;
    d.eat();   // Inherited from Animal
    d.bark();  // Own function
    return 0;
}
```

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

**Ans :- Encapsulation means binding data and methods together in a class and hiding the internal details from outside code.**

- **How it is achieved:**
  - **By making data members private**
  - **Providing access through public functions (getters/setters)**

**Example:**

```
class Student {
private:
    int rollNo;  // Private: can't access directly

public:
    void setRollNo(int r) {
        rollNo = r;
    }

    int getRollNo() {
        return rollNo;
    }
};

int main() {
    Student s;
    s.setRollNo(101);
    cout << s.getRollNo();  // Accessing private data via public function
}
```

**Encapsulation protects data from unauthorized access and makes the program more secure and manageable.**