

Module:3 Introduction to OOPS Programming

NAME : - Rahul Zapadiya

Subject : - oops

1. Introduction to C++

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

Ans :

1. Procedural Programming (POP)

Approach : Top-down

Focus : functions and procedures

Data access : data is exposed to all functions

Modularity : Less modular ; functions depend on global data.

Reusability : limited

Examples : C, pascal

2.Object-Oriented Programming (OOP)

Approach : Bottom-up

Focus : Objects and classes

Data access : Data is hidden and accessed via methods

Modularity : Highly modular using encapsulated classes

Reusability : promoter code reuse via inheritance

Examples : C++, Java, Python

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

Ans : Encapsulation: Data and functions are bundled together inside objects, improving security and modularity.

Inheritance: Allows new classes to acquire properties and behaviors of existing ones, promoting code reuse.

Polymorphism: Same operation can behave differently on different classes, making code flexible and scalable.

Abstraction: Focuses on essential features, hiding unnecessary details, which simplifies complexity.

Maintainability: Code is easier to manage, update, and debug due to better structure.

Modularity: Projects can be divided into objects, making them more manageable and team-friendly.

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

Ans : Install a Compiler: Examples include GCC (MinGW on Windows), Clang, or MSVC.

Install an IDE or Text Editor: Such as Code::Blocks, Dev C++, Visual Studio, or VS Code.

Set Environment Variables: Add compiler path to system PATH.

Create a New Project or File: Write .CPP files inside the IDE or editor.

Compile the Program: Use a build command or IDE build button.

Run the Executable: After successful compilation, run the output file to see results.

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

Ans : input : Used to take input from the user.

Int age;

```
cin>>age;
```

Output: cout

Used to print output to the screen.

```
cout << "Enter your age: ";
```

Example :

```
#include <iostream>
void main() {
    int age;
    cout << "Enter your age: ";
    cin >> age;
    cout << "You are " << age << " years old.";
    getch();
}
```

2. Variables, DataTypes, and Operators

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

Ans : C++ provides several data types, broadly classified into:

1. Fundamental Data Types:

- Int for integers
`int age = 21;`
- Float for floating-point numbers
`float height = 5.9;`
- Double for higher precision floating numbers
`double pi = 3.14159;`
- char for single characters
`char grade = 'A';`
- bool for Boolean values(true/false)
`bool is Passed = true;`

2. Derived Data Types:

- Array:
`int marks[5];`
- Pointer:
`int *ptr = &age;`

- Function:
Functions that return or take data types.

3. User-defined Data Types:

- struct, class, enum:

```
struct Student {  
    int id;  
    char name[20];  
};
```

4. Void Type:

- Used for functions that return nothing
`void display();`

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

Ans : Implicit Type Conversion:

- Done automatically by the compiler.
- Example:

```
int a = 10;
```

```
float b = a; // int to float automatically
```

Explicit Type Conversion:

- Done manually by the programmer using casting.
- Example:

```
float a = 10.5;
```

```
int b = (int)a; // float to int using casting
```

3.What are the different types of operators in C++?

Provide examples of each.

Ans : 1. Arithmetic Operators

Used for mathematical operations:

+ , - , * , / , %

Example: a + b

2. Relational Operators

Used to compare values:

==, !=, <, >, <=, >=

Example: a < b

3. Logical Operators

Used for logical conditions:

&& (AND), || (OR), ! (NOT)

Example: `a > 0 && b < 10`

4. Assignment Operators

Used to assign or update values:

=, +=, -=, *=, /=, %=

Example: `a += 5`

5. Increment/Decrement Operators

Increase or decrease value by 1:

++, --

Example: `a++, --b`

6. Bitwise Operators

Operate on bits:

&, |, ^, ~, <<, >>

Example: `a & b`

7. Ternary Operator

Short form of if-else:

? :

Example: $(a > b) ? a : b$

8. Scope Resolution Operator

Access global or class members:

::

Example: ::x

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

Ans : Constants:

- Fixed values that cannot be changed during program execution.
- Declared using const keyword.

`const float PI = 3.14159;`

Literals:

- Constant values assigned directly to variables.

```
int num = 100;      // 100 is an integer literal  
char ch = 'A';     // 'A' is a character literal  
float f = 12.34;    // 12.34 is a float literal
```

Purpose:

- Improve readability and maintainability.
- Prevent accidental modification of fixed values.
- Clarify intent of the code.

3. Control Flow Statements

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

Ans : Conditional statements allow a program to make decisions based on certain conditions.

if-else Statement:

It checks a condition; if true, a block of code runs.
Otherwise, another block runs.

cpp

CopyEdit

```
int num = 10;  
  
if (num > 0) {  
    cout << "Positive";  
  
}  
  
else  
  
{  
    cout << "Non-positive";  
  
}
```

switch Statement:

Used to select one block of code among many options, based on a value.

cpp

CopyEdit

```
int choice = 2;  
  
switch(choice) {  
  
    case 1: cout << "One"; break;  
  
    case 2: cout << "Two"; break;  
  
    default: cout << "Invalid";  
  
}
```

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

break Statement:

Used to exit the loop immediately.

```
for (int i = 1; i <= 5; i++)  
  
{  
  
    if (i == 3)  
  
        break;
```

```
cout << i << " ";
}

Output: 1 2
```

continue Statement:

Skips the current iteration and moves to the next.

```
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 : A nested control structure means using one control structure (like a loop or if-else) inside another.

Example: Nested loops (multiplication table)

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

Output:

```
1 2 3  
2 4 6  
3 6 9
```

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. Functions help in code reusability and modular programming.

- Function Declaration (or Prototype):

It tells the compiler about the function's name, return type, and parameters before its actual definition.

Example: int add(int, int);

- Function Definition:

This is where the function's logic is written.

Example:

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

- Function Calling:

This is how you execute the function.

Example: int sum = add(5, 3);

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

Ans : Scope defines where a variable can be accessed within a program.

- Local Scope:

A variable declared inside a function or block is accessible only within that block.

Example:

```
void fun() {  
    int x = 10; // local variable  
}
```

- Global Scope:

A variable declared outside all functions is

accessible from any function in the program.

Example:

```
int x = 20; // global variable
```

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

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

Ans : Recursion is a process where a function calls itself to solve a smaller instance of the same problem.

Example: Factorial using recursion

```
int factorial(int n) {  
    if (n == 0)  
        return 1;
```

```
else  
    return n * factorial(n - 1);  
}
```

Explanation: The function keeps calling itself with a reduced value of n until it reaches 0, then returns the result back through the chain.

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

Ans : A function prototype is a declaration of a function that tells the compiler about the function's name, return type, and parameters before its actual definition.

Syntax Example:

```
int sum(int, int);
```

Why used:

- To inform the compiler about a function before it is used.
- Helps in type checking of parameters and return types.
- Necessary when the function definition is written after main().

5.Arrays and Strings

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

Ans : An array in C++ is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow easy access using index numbers.

- Single-Dimensional Array (1D):
Stores data in a linear form using a single

index.

Example:

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

- Multi-Dimensional Array (2D or more):
Stores data in tabular form using multiple indices.

Example (2D array):

```
int matrix[2][3] = {
```

```
    {1, 2, 3},
```

```
    {4, 5, 6}
```

```
};
```

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

Ans : In C++, strings can be handled using character arrays or the string class from the Standard Library.

- Using Character Arrays:

```
char name[] = "Dhvanit";  
cout << name;
```

- Using string Class:

```
#include <iostream.h>  
#include<conio.h>  
#include <string>
```

```
void main() {  
    string greeting = "Hello";  
    cout << greeting.length(); // Outputs 5  
    getch();  
}
```

String handling functions:

- length(), append(), substr(), compare(), etc.

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

Ans : Arrays in C++ can be initialized at the time of declaration:

- 1D Array Initialization:

```
int marks[4] = {90, 85, 70, 95};
```

- 2D Array Initialization:

```
int table[2][2] = {
```

```
    {1, 2},
```

```
    {3, 4}
```

```
};
```

You can also leave out the size if you provide the values:

```
int nums[] = {10, 20, 30}; // Size is auto-determined as 3
```

6. Introduction to Object-Oriented Programming

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

Ans : Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around data, or objects, rather than functions. The key OOP concepts are:

- Class: Blueprint for creating objects.
- Object: Instance of a class.
- Encapsulation: Bundling data and methods into a single unit (class).
- Abstraction: Hiding internal details and showing only essential features.
- Inheritance: One class acquires the properties of another.
- Polymorphism: Ability to use functions in multiple forms.

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

Ans : A class in C++ is a user-defined data type that groups data and functions. An object is an instance of a class.

Example:

```
#include <iostream.h>
```

```
#include<conio.h>
```

```
class Car {
```

```
public:
```

```
    string brand;
```

```
    void display() {
```

```
        cout << "Brand: " << brand << endl;
```

```
}
```

```
};
```

```
void main() {  
    Car c1;  
    c1.brand = "Toyota";  
    c1.display();  
    getch();  
}
```

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

Ans : Inheritance allows a class (derived class) to inherit members from another class (base class). It promotes code reusability.

Example:

```
#include <iostream.h>
```

```
#include<conio.h>

class Animal {
public:
    void sound() {
        cout << "Animal makes a sound." << endl;
    }
};

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

```
void main() {  
    Dog d;  
    d.sound();  
    d.bark();  
    getch();  
}
```

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

Ans : Encapsulation is the concept of wrapping data (variables) and functions into a single unit (class) and restricting access to some of the object's components.

In C++, it's achieved using access specifiers:

- private: Accessible only inside the class.
- public: Accessible from outside the class.

Example:

```
class Student {
```

private:

```
    int rollNo;
```

public:

```
    void setRollNo(int r) {
```

```
        rollNo = r;
```

```
}
```

```
    int getRollNo() {
```

```
        return rollNo;
```

```
}
```

```
};
```

