MODULE #3 INTRODUCTION TO OOPS PROGRAMMING

1. Introduction to C++:

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

Answer:

Feature	Procedural Programming(pop)	Object-Oriented Programming(oop)
Approach	Follow a step-by-step procedure(like a recipe).	Organizes code using objects(like real-world things).
Focus	Focuses on functions(what to do).	Focuses on objects and classes(who does it).
Data	Data is not secure(can be accessed from anywhere).	Data is protected using encapsulation(hidden inside object).
Reusability	Less code reuse.	Promotes code reuse using inheritance and polymorphism.
Language	C,pascal.	C++,Java,Python.

Example:

- Procedural:

  "Boil water → Add tea leaves → Pour into cup" (focus on steps)

- OOP:

  "Create a **TeaMaker** object → Call makeTea() method" (focus on object and behavior)

Created by: Hensi Vaghela

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

Answer:

main advantages of Object-Oriented Programming (OOP) over Procedural-Oriented Programming (POP)**:**

1. Reusability (Using Inheritance):

- You can reuse code by creating new classes from existing ones using *inheritance*.

  Benefit: Saves time and reduces code duplication.

2. Data Security (Encapsulation):

- Keeps data safe by hiding it inside objects and only allowing access through methods.

  Benefit: Prevents accidental changes and keeps data safe.

3. Easy to Manage (Modularity):

- Code is divided into objects (modules), each handling its own task.

  Benefit: Easier to understand, test, and fix bugs.

4.Real-World Mapping:

- Uses classes and objects which are like real-world things (e.g., Car, Employee).

  Benefit: Makes programs easier to design and relate to real-life problems.

5. Polymorphism (One Name, Many Behaviors):

- Same method can work differently for different objects.

  Benefit: Code becomes flexible and easier to extend.

6. Scalability and Maintainability:

- Easier to update or scale up as the project grows.

  Benefit: Long-term projects are easier to manage and maintain.

Created by: Hensi Vaghela

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

Answer:

1. Install a C++ Compiler:

A **compiler** converts your C++ code into machine code that the computer can understand.

- **Popular C++ compilers:**
  - **Windows:** MinGW, GCC (via MSYS2 or WSL)
  - **Linux:** GCC is usually pre-installed
  - **Mac:** Xcode includes a C++ compiler

2. Choose and Install a Code Editor or IDE:

An **IDE (Integrated Development Environment)** or **editor** helps you write and run C++ code easily.

- **Popular IDEs/editors:**
  - **Code::Blocks** (beginner-friendly)
  - **Dev C++**
  - **Visual Studio (Windows)**
  - **VS Code** (lightweight, cross-platform)

3. Set Up the Compiler in the IDE (if needed):

- Example: In Code::Blocks, go to **Settings → Compiler** and check if it detects your compiler.

➢ **Setup with Dev C++ :**

1. **Download Dev C++:**
   - Go to https://sourceforge.net/projects/orwelldevcpp/
   - Download and install it.

2. **Open Dev C++ and create a new file:**
   - Click **File → New → Source File**

3. **Write your C++ code:**

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

 int main() {
```


Created by: Hensi Vaghela

```
    cout << "Hello, World!";
  return 0;
}
```

4. **Save the file:**
   - File → Save as program.cpp

5. **Compile and Run:**
   - Press **F11** or click **Execute → Compile & Run**

- **Setup with VS Code:**

1. **Install VS Code:**
   - From: https://code.visualstudio.com/

2. **Install MinGW Compiler:**
   - Download from https://www.mingw-w64.org/
   - Install it and note the install path (like C:\mingw-w64\bin)
   - Add this path to **System Environment Variables → Path**

3. **Open VS Code → Install Extensions:**
   - Click Extensions icon
   - Install:
     - **C/C++ by Microsoft**
     - **Code Runner** (optional for easy running)

4. **Write your code in a .cpp file**

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

int main() {
   cout << "Hello from VS Code!";
   return 0;
}
```

5. **Run the Program:**
   - Open Terminal in VS Code: Ctrl + ~
   - Compile: g++ main.cpp -o main
   - Run: ./main

Created by: Hensi Vaghela

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

Answer:

In C++, the **main input/output (I/O) operations** are performed using the **iostream** library, which provides the following key stream objects:

- **Input Operation: cin**

  Used to take input from the user (usually from the keyboard).

  Use **extraction (>>)** operator.

- **Output Operation: cout**

  Used to display output to the screen (usually to the console).

  Use **insertion (<<)** operator.

Example:

- **Input & Output in C++**

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

int main()
{
   int age;
   string name;

   cout << "Enter your name: ";   // Output to console
   cin >> name;                  // Input from user

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

   cout << "Hello, " << name << "! You are " << age << " years old." << endl;

   return 0;
}
```


Created by: Hensi Vaghela

2. Variables, Data Types, and Operators:

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

Answer:

There are basic **7 datatype** available in c++.

1. int (Integer):

- Used to store whole numbers (without decimals).

Example:

```
int age = 25;
```

2. float (Floating Point):

- Used to store numbers with decimals (small size).

Example:

```
float average = 99.50;
```

3. double (Double Floating Point):

- Used to store larger decimal numbers.

Example:

```
double pi = 3.14159;
```

4. char (Character):

- Used to store only single character.

Example:

```
char grade = 'A';
```

5. string (Text):

- Used to store a group of characters (a word or sentence).
- string needs #include <string> at the top.

Example:

```
string name = "Hensi";
```

6. bool (Boolean):

- Used to store true or false values.

Example:

```
bool isPassed = true;
```

7. unsigned int:

- Stores only positive whole numbers (no negative).

Example:

```
unsigned int marks = 80;
```

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

Answer:

1. Implicit Type Conversion (Type Promotion):

Done automatically by C++.

C++ changes one data type to another **on its own** when needed.

Example:

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

- Here, a is an int, but it becomes float when assigned to b.
  You don't need to write anything special — it happens **automatically**.

2. Explicit Type Conversion (Type Casting):

Done manually by the programmer.

You tell C++ **exactly** how to convert the type.

Example:

```
float a = 5.75;
int b = (int)a;  // float is manually converted to int
```

- Here, the decimal value 5.75 becomes 5 — the decimal part is cut off because we forced it to become an int.

Feature	Implicit Conversion	Explicit Conversion
Who does it?	C++ automatically	You (the programmer)
Syntax	No special code needed	Use type in brackets `(type)`
Example	`float b = a;`	`int b = (int)a;`
Control	Less control	More control

Created by: Hensi Vaghela

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

Answer:

1. Arithmetic Operators:

- Used to do math calculations.

Operator	Meaning	Example (a = 10,b =5)	Result
+	Addition	a + b	15
-	Subtraction	a - b	5
*	Multiplication	a * b	50
/	Division	a / b	2
%	Modulus (Remainder)	a % b	0

2. Relational (Comparison) Operators:

- Used to compare two values. Result is **true (1)** or **false (0)**.

Operator	Meaning	Example	Result
==	Equal to	a == b	0
!=	Not equal to	a != b	1
>	Greater than	a > b	1
<	Less than	a < b	0
>=	Greater or equal	a >= b	1
<=	Less or equal	a <= b	0

3. Logical Operators:

- Used to combine conditions.

Operator	Meaning	Example	Result
&&	Logical AND	a > 5 && b < 10	true
||	Logical OR	A == 5 || A == 10	true
!	Logical NOT	!(a == b)	true

Created by: Hensi Vaghela

4. Assignment Operators:

- Used to assign values to variables.

Operator	Example	Meaning
=	a = 5	Assigns 5 to a
+=	a += 3	a = a + 3
-=	a -= 2	a = a - 2
*=	a *= 2	a = a * 2
/=	a /= 2	a = a / 2
%=	a %= 2	a = a % 2

5. Increment and Decrement Operators

- Used to increase or decrease a value by 1.

Operator	Meaning	Example	Result
++	Increment	a++ or ++a	a = a + 1
--	Decrement	a-- or --a	a = a - 1

6. Bitwise Operators:

- Work on bits (0s and 1s) of numbers.

Operator	Meaning	Example
&	AND	a & b
`	`	OR
^	XOR	a ^ b
~	NOT	~a
<<	Left Shift	a << 1
>>	Right Shift	a >> 1

Created by: Hensi Vaghela

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

Answer:

What are Constants?

- **Constants** are values that **do not change** during the program.

Why use constants?

1. To protect values from being changed by mistake
2. To make your code easier to read and understand

How to define a constant:

```
const int maxMarks = 100;
```

Now maxMarks cannot be changed anywhere in the program.

What are Literals?

- **Literals** are the **actual fixed values** used directly in the code.

Examples of literals:

```
int  age = 18;         // 18 is an integer literal
float  pi = 3.14;      // 3.14 is a float literal
char  grade = 'A';     // 'A' is a character literal
string name = "Hensi"; // "Hensi" is a string literal
```

Types of literals:

- **Integer literal**: 10, -5
- **Float literal**: 3.14, -0.01
- **Character literal**: 'A', 'B'
- **String literal**: "Hello", "World"
- **Boolean literal**: true, false

3. Control Flow Statements:

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

Answer:

- Conditional statements are **decision-making statements** in C++ that allow your program to take **different actions** based on **different conditions**.

- They help control the **flow of execution** depending on whether a **certain condition is true or false**.

Types of Conditional Statements:

1. **if statement**
2. **if-else statement**
3. **if-else if-else ladder**
4. **switch statement**

1.if-else Statement:

- The if-else statement is used when we want to execute **one block of code if a condition is true** and another block if it is false.

Syntax:

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

Example:

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

int main()
{
   int age = 18;
```

```
   if (age >= 18)
 {
      cout << "You are eligible to vote.";
   }
else
{
      cout << "You are not eligible to vote.";
}

   return 0;
}
```

2.switch Statement:

- The switch statement is used to **select one of many blocks of code to be executed**.
- It works with **integers, characters**, and **enumerated types**.

Syntax:

```
switch(expression)
 {
   case value1:
      // code block
      break;

   case value2:
      // code block
      break;

   ...
   default:
      // default code block
}
```

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;

}
```

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

Answer:

Loops are used in C++ to execute a block of code **repeatedly** based on a condition.

1. for Loop:

- Used when the **number of iterations is known** in advance.

Syntax:

```
for (initialization; condition; incre/decre)
{
   // code to repeat
}
```

2. while Loop:

- Used when the **number of iterations is not known**, and you want to loop **as long as a condition is true**.

Syntax:

```
while (condition)
{
   // code to repeat
}
```

3. do-while Loop:

- Like while, but the **code runs at least once**, even if the condition is false. This is because the **condition is checked after** the loop body.

Syntax:

```
Do
 {
   // code to repeat
} while (condition);
```

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

Answer:

- Both break and continue are **loop control statements** in C++. They help **change the flow** of loops in specific situations.

1.break Statement:

- **Purpose:** Immediately **terminates the loop** (or switch) when executed.
- **Used when:** You want to **exit a loop early**, even if the loop condition is still true.

Example:

```
#include <iostream>
using namespace std;
int main(){
   for (int i = 1; i <= 10; i++) {
      if (i == 5){
         break;  // Loop stops when i is 5
      }
      cout << i << endl;
   }
   return 0;
}
```

2.continue Statement:

- **Purpose: Skips the current iteration** and goes to the next one.
- **Used when:** You want to **skip some part of the loop** but continue looping.

Example:

```
#include <iostream>
using namespace std;
int main(){
   for (int i = 1; i <= 5; i++){
      if (i == 3) {
         continue;  // Skips printing when i is 3
      }cout << i << endl;
   }
   return 0;
}
```

4.Explain nested control structures with an example.

Answer:

- Nested control structures are control statements (like if, for, while, switch, etc.) placed **inside another control structure**.
- They help handle more complex decision-making or looping.

Example 1: Nested if Statement:

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

int main() {
   int marks;
   cout << "Enter your marks: ";
   cin >> marks;
if (marks >= 50) {
      if (marks >= 75) {
         cout << "Distinction" << endl;
      } else {
         cout << "Pass" << endl;
      }
   } else {
      cout << "Fail" << endl;
   }
   return 0;
}
```

Example 2: Nested for Loop:

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

int main() {
   for (int i = 1; i <= 3; i++) {
      for (int j = 1; j <= 3; j++) {
         cout << "(" << i << "," << j << ") ";
      }
      cout << endl;
   }
   return 0;
}
```

4. Functions and Scope:

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

Answer:

- A **function** in C++ is a **block of reusable code** that performs a specific task.
- It helps in dividing a large program into smaller, manageable parts.

➢ **Types of Functions:**

- **Built-in functions** :

  1.Math funcations.
  2.String funcations.

- **User-defined functions**:

  1.with parameter with returntype.
  2.with parameter without returntype.
  3.without parameter with returntype.
  4.without parameter without returntype.

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

  1. **Function Declaration (or Prototype):**

  - Declares the function **before main()**.
  - Tells the compiler the return type ,function name, and parameters.

  Ex., int add(int, int); // Function declaration

  2. **Function Definition:**

  - Contains the actual **code (body)** of the function.
  - It tells what the function will do.

  Ex., int add(int a, int b) { // Function definition
  return a + b;

}

3. Function Call:

- Executes the function by **calling it in main() or another function.**

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

Example:

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

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

int main()
{
   int sum = add(10, 20); // Function call
   cout << "Sum = " << sum << endl;
   return 0;
}

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

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

Answer:

- The **scope** of a variable refers to the **region of the program** where the variable is **defined** and can be **accessed or used**.

There are mainly two types of scope in C++:

1. Local Scope
    - A variable declared **inside** a function or block that is called **local scope**.
2. Global Scope
    - A variable declared **outside all functions** that is called **global scope**.

➢ Difference Between Local and Global Scope:

Feature	Local Variable	Global Variable
Declaration	Inside a function or block	Outside all functions
Access	Only within that block/function	Anywhere in the program
Lifetime	Created and destroyed with block	Exists throughout program execution
Memory	Uses stack memory	Uses static/global memory
Use case	For temporary or function-specific data	For shared data across functions

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

Answer:

- Recursion is a programming technique where a function calls itself to solve a smaller instance of the same problem until it reaches a **base case** (a condition where the function stops calling itself).

Key Concepts:

1. **Recursive Function** – A function that calls itself.
2. **Base Case** – Condition to stop the recursion.
3. **Recursive Case** – The part where the function calls itself.

Example:

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

// Recursive function to calculate factorial
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;
   cout << "Enter a number: ";
   cin >> num;

   cout << "Factorial of " << num << " is: " << factorial(num);
   return 0;
}
```

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

Answer:

- A **function prototype** is a declaration of a function that tells the compiler about the function's name, return type, and parameters **before** the function is actually defined.

Syntax:

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

Example:

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

int main() {
   int result = add(5, 3);
   cout << "Result: " << result;
   return 0;
}
int add(int a, int b) {  // Function definition
   return a + b;
}
```

Why Are Function Prototypes Used?

1. **Tell the Compiler in Advance:**
   - The prototype gives the compiler enough information to compile calls to the function even if the definition comes later in the code.
2. **Enable Modular Coding:**
   - You can define functions after main() or in separate files.
3. **Helps in Type Checking:**
   - The compiler checks whether the function is called with the correct number and type of arguments.
4. **Improve Code Readability:**

- It provides a quick overview of all the functions used in the program at the beginning.

5. Arrays and Strings:

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

Answer:

- An **array** in C++ is a **collection of elements** of the **same data type**, stored in **contiguous memory locations**.

- Arrays are used to store multiple values in a single variable instead of declaring separate variables for each value.

Syntax:

```
int numbers[5] = {10, 20, 30, 40, 50};
```

Types of Array:

1. single-dimensional Array
2. multi-dimensional Array

Difference:

Feature	Single-Dimensional Array	Multidimensional Array
Structure	Linear (1 row)	Tabular (rows and columns)
Declaration	int arr[5];	int arr[2][3];
Access Example	arr[2]	arr[1][2]
Memory Layout	Continuous in one direction	Continuous in nested structure
Use Case	List of items	Matrices, tables, grids

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

Answer:

- In C++, **string handling** refers to working with sequences of characters.

C++ provides two main ways to handle strings:

1. C-Style Strings (Character Arrays)

- These are arrays of characters ending with a null character '\0'.
- They are part of the C standard and are supported in C++.

2. C++ Strings (using std::string)

- Provided by the C++ Standard Library.
- Easier and safer to use than C-style strings.

Example:

```
#include <iostream>
#include <string>  // Required for using string
using namespace std;

int main()
{
   string greeting = "Hello, World!";
   cout << "C++ String: " << greeting << endl;


   // String operations

   greeting += " Have a nice day."; // Concatenation
   cout << "After Concatenation: " << greeting << endl;

   cout << "Length: " << greeting.length() << endl;

   return 0;
}
```

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

Answer:

- **arrays** can be initialized in several ways at the time of declaration.
- Here's how you can initialize both **1D (one-dimensional)** and **2D (two-dimensional)** arrays.

1. One-Dimensional Array (1D) Initialization

Syntax:

datatype arrayName[size] = {value1, value2, ..., valueN};

Example:

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

int main() {
   int a[5] = {1, 2, 3, 4, 5}; // Initialize all elements
   int b[5] = {1, 2};         // Remaining elements set to 0 → {1, 2, 0, 0, 0}
   int c[] = {10, 20, 30};     // Size auto-determined as 3

   for(int i = 0; i < 5; i++) {
      cout << "a[" << i << "] = " << a[i] << endl;
   }

   return 0;
}
```

2. Two-Dimensional Array (2D) Initialization

Syntax:

datatype arrayName[rows][columns] = {{row1_val1, row1_val2},{row2_val1, row2_val2}};

Example:

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

int main() {
```

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

    // Output 2D array
    for(int i = 0; i < 2; i++) {
        for(int j = 0; j < 3; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

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

Answer:

- A **string** is a sequence of characters like "Hello" or "C++ is fun".

➢ Basic String Operations:

1. Create a string:

```
string name = "Alice";
```

2. Input a string:

```
string city;
cin >> city;   // reads one word
```

3. Input full line:

```
getline(cin, city);  // reads full line with spaces
```

4. Concatenation (joining strings):

```
string first = "Hello";
string second = "World";
string result = first + " " + second;  // Hello World
```

5. Length of string:

```
string s = "Coding";
cout << s.length();  // Output: 6
```

➢ Common String Functions:

1. strlen() – String Length:

- Returns the **length** of the string (excluding the null character \0).

2. strcpy() – String Copy:

- Copies the content of one string into another.

3. strcat() – String Concatenation:

- Appends one string to the end of another.

4. strcmp() – String Compare:

- Compares two strings **lexicographically**.

5. strchr() – Search Character in String:

- Finds the **first occurrence** of a character in a string.

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

int main() {
   char str1[50] = "Hello";
   char str2[50] = "World";
   char str3[50];

   // strlen() - String Length
   cout << "Length of str1: " << strlen(str1) << endl;

   // strcpy() - String Copy
   strcpy(str3, str1);
   cout << "Copied str1 to str3: " << str3 << endl;

   // strcat() - String Concatenation
   strcat(str1, str2);  // Now str1 = "HelloWorld"
   cout << "After concatenation: " << str1 << endl;

   return 0;
}
```

6. Introduction to Object-Oriented Programming:

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

Answer:

1)object :

- **Any Entity which has own state and behavior that is called object.**
  ex:pen,paper,chair etc..

2)class :

- **collection of objects or it's have a data memebers and method's collection.**
  ex: human body

3)abstraction :

- **Hiding internal details and showing functionalities**
  ex:login page

4)encapsulation :

- **Wrapping up of data or binding of data**
  ex:capsule

5)inheritance :

- **When One object acquire all the properties and behavior of parent class**
  ex: father-son

6)polymorphism :

- **Many ways to perform anything**
  ex:road ways

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

Answer:

Class:

- A **class** is a **user-defined blueprint** or template that groups **data members** (variables) and **member functions** (methods) into a single unit.
- It defines the **structure** and **behavior** of an object.

Object:

- An **object** is a **real instance** of a class.
- When an object is created, memory is allocated and you can use it to access class members.

Example:

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

// Define a class
class Student {
public:
   string name;
   int age;

   void display() {
      cout << "Name: " << name << endl;
      cout << "Age: " << age << endl;
   }
};

int main() {
   // Create an object of class Student
   Student s1;

   s1.name = "Alice";
   s1.age = 20;

   // Call the member function
   s1.display();

   return 0;
}
```

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

Answer:

- **Inheritance is an Object-Oriented Programming (OOP) concept where one class (called the child or derived class) inherits the properties (data members and member functions) of another class (called the parent or base class).**

Types of Inheritance in C++:

1.Single Inheritance:

- One derived class inherits from **one base class**.

  Example: *B* inherits from *A*.

2.Multiple Inheritance:

- One derived class inherits from **more than one base class**.

  Example: C inherits from A and B.

3.Multilevel Inheritance:

- A class is derived from another **derived class** (like a chain).

  Example: C inherits from B, and B inherits from A

4.Hierarchical Inheritance:

- **Multiple classes** inherit from a **single base class**.

  Example: *B* and *C* both inherit from *A*.

5.Hybrid Inheritance:

- A combination of **two or more types** of inheritance.

  Example: Mix of multilevel and multiple inheritance.

Example: Single Inheritance:

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

class A{
public :
		void a()
		{
			cout<<"A called"<<endl;
		}

};

class B : public A
{
		public :
		void b()
		{
			cout<<"B called"<<endl;
		}
};
int main()
{
	B b1;
	b1.a();
	b1.b();

	return 0;
}
```

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

Answer:

- **Encapsulation in C++** is one of the fundamental principles of Object-Oriented Programming (OOP).

- It refers to **the bundling of data (variables) and the methods (functions) that operate on that data into a single unit — a class — and restricting direct access to some of the object's components**.

Key Idea:

- Encapsulation **protects the internal state** of an object by making variables private and exposing only necessary functions (methods) as public.

➢ **How Encapsulation is Achieved in C++:**

1. **By using access specifiers**:

    - **private**:
      Members are accessible only within the class.

    - **public**:
      Members are accessible from outside the class.

    - **protected**:
      (Used in inheritance) Accessible in the class and its derived classes.

2. **Using getter and setter methods**:

    - These allow controlled access to private data.

Example:

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

class Student {
private:
   int rollNumber;  // private data member
   string name;

public:
   // Setter methods
   void setRollNumber(int r) {
      rollNumber = r;
   }

   void setName(string n) {
      name = n;
   }

   // Getter methods
   int getRollNumber() {
      return rollNumber;
   }

   string getName() {
      return name;
   }
};

int main() {
   Student s;
   s.setRollNumber(101);
   s.setName("Amit");

   cout << "Roll Number: " << s.getRollNumber() << endl;
   cout << "Name: " << s.getName() << endl;

   return 0;
}
```