Assignment

Module 3 – Introduction to OOPS Programming

(1) What Are The Key Differences Between Procedural Programming And Objectorientedprogramming (OOP)?,List And Explain The Main Advantages Of OOP Over POP,Explain The Steps Involved In Setting Up A C++ Development Environment, What Are The Main Input/Output Operations In C++? Provide Examples.

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

Point	Procedural Programming	Object-Oriented Programming (OOP)
1. Approach	Focuses on functions and steps to perform a task.	Focuses on objects and classes to solve a problem.
2. Structure	Program is divided into functions.	Program is divided into objects.
3. Data	Data is not secure; any function can access it.	Data is secured using encapsulation.
4. Reusability	Code reusability is less.	Code can be reused using inheritance and polymorphism.
5. Real-world Mapping	Hard to relate with real-world things.	Easy to relate with real-world objects.
6. Example Languages	C, Pascal	C++, Java, Python

- **List and explain the main advantages of OOP over POP**

1. Code Reusability

- OOP allows us to reuse code using inheritance.
- We can create new classes from old ones without writing all the code again.

2. Data Security (Encapsulation)

- In OOP, data is hidden inside objects.
- Only specific functions (methods) can access or change the data.
- This keeps the data safe and secure.
- In POP, data is open and any function can change it.

3. Easy to Understand (Real-world Mapping)

- OOP uses real-world objects (like Car, Student, Account),
- So, it's easier to design and understand programs.

4. Modularity

- OOP breaks the program into small parts called objects or classes.
- Each object has its own data and functions.
- This makes the program clean and organized**.**

5. Easy to Maintain and Update

- If we want to make changes, we can do it in one class or object.
- We don't need to touch the whole program.
- This saves time and reduces errors.

6. Polymorphism

- In OOP, the same function or operator can work in different ways.
- This makes the code more flexible and powerful.

7. Good for Big Projects

- OOP is best for making large and complex applications like games, banking systems, etc.
- POP becomes hard to manage in big projects.

OOP is better than POP because it gives **reusability, security, easy maintenance, and real-world design**.

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

Step 1: Install a C++ Compiler

- A compiler converts your C++ code into machine language (like .exe).
- Most popular compiler: MinGW (Windows) or GCC (Linux).

👉 **For Windows:**

- Download MinGW from https://www.mingw-w64.org/
- Install it and add its path to the Environment Variables.

Step 2: Choose and Install a Code Editor or IDE

- IDE means Integrated Development Environment – it helps you write, compile, and run code easily.
- **Some popular options:**
    - Code::Blocks (Simple and beginner friendly)
    - Visual Studio Code (VS Code) (Lightweight and popular)
    - Dev C++ (Simple and offline)
    - Turbo C++ (Old but still used in schools)

Step 3: Set Up the Editor with Compiler (if needed)

👉 **For example, in VS Code:**

- Install VS Code from https://code.visualstudio.com

- Install C/C++ extension from the Extensions tab.
- Make sure MinGW is installed and path is set.
- Create a .cpp file, and use the terminal to compile:

  g++ file.cpp -o file

  ./file

Step 4: Write Your First C++ Program

Example:

#include<iostream>

using namespace std;

int main() {

cout << "Hello, C++!" << endl;

return 0;

} Save this as hello.cpp.

Step 5: Compile and Run the Program

- If you are using an IDE like **Code::Blocks**, press **F9** to compile and run.
- If using terminal:

Step 5: Compile and Run the Program

- If you are using an IDE like **Code::Blocks**, press **F9** to compile and run.
- If using terminal:

  g++ hello.cpp -o hello

  ./hello

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

In C++, we use:

- cin → for input (getting data from user)
- cout → for output (showing data on screen)

Both are part of the iostream library.

1. Output Operation – cout

- Used to print text or values on the screen.
- cout stands for console output.
- Uses << (insertion operator).

Example:

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

int main() {
   cout << "Hello, World!";
   return 0;
} //output- Hello, World!
```

2. Input Operation – cin

- Used to take input from the user.
- cin stands for console input.
- Uses >> (extraction operator).

Example:

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

```
int main() {
    int age;
    cout << "Enter your age: "; // input 18
    cin >> age;
    cout << "Your age is: " << age;
    return 0;
} // output - Your age is: 18
```

Multiple Inputs/Outputs Example:

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

int main() {
    int a, b;
    cout << "Enter two numbers: ";
    cin >> a >> b;

    cout << "Sum is: " << (a + b);
    return 0;
}
```

(2) What Are The Different Data Types Available In C++? Explain With Examples,Explain The Difference Between Implicit And Explicit Type Conversion In C++,What Are The Different Types Of Operators In C++? Provide Examples Of Each,Explain The Purpose And Use Of Constants And Literals In C++.

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

  1. Basic (Primary) Data Types:

Data Type	Use	Example
Int	Stores whole numbers	int age = 20;
Float	Stores decimal numbers (small)	float weight = 55.5;
Double	Stores decimal numbers (more precision)	double pi = 3.14159;
char	Stores a single character	char grade = 'A';
bool	Stores true/false values	bool isPassed = true;

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

In C++, type conversion means changing one data type into another. There are two types of type conversion:

1. Implicit Type Conversion:
   This type of conversion is done automatically by the compiler.
   When we assign a smaller data type to a bigger one (like int to float), C++ converts it on its own.

   Example:

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

2. Explicit Type Conversion:
   In this type, the programmer manually converts one data type into another using type casting.
   It is used when we want full control over conversion.

   Example:

   float x = 9.7;

   int y = (int)x; // x is manually converted to int

- **<u>What are the different types of operators in C++? Provide examples.</u>**

  In C++, operators are symbols that are used to perform different operations on variables and values. They help in calculations, comparisons, decisions, etc.

  1. **Arithmetic Operators**

Operator	Meaning	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus (Reminder)	a%b

2. **Relational (Comparison) Operators**

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

3. **Logical Operators**

Operator	**Meaning**	**Example**
&&	AND	if(a > 0 && b > 0)
||	OUR	`
!	AND	if(!(a == b))

4)Assignment Op: =

5)Shorthand Op/Assignment Op: +=, -=, *=, /=

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

  In C++, constants and literals are used to store fixed values that do not change while the program is running.

Constants:

- A **constant** is a variable whose value **cannot be changed** after it is defined.
- It helps to make the program **safe** and **more readable**.
- We use the const keyword to define a constant.

Example:

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

Here, PI is a constant and its value will always remain 3.14.

Why use constants?

- To prevent accidental changes in important values.
- To make the program easier to understand.
- Useful in places where values are fixed (like pi, tax rate, etc).

Literals:

- A **literal** is a **fixed value** that is directly written in the code.
- It can be a number, character, string, or boolean.

Examples of literals:

```
int age = 20;          // 20 is an integer literal
char grade = 'A';      // 'A' is a character literal
float pi = 3.14;       // 3.14 is a float literal
bool isTrue = true;    // true is a boolean literal
```

Types of literals in C++:

- Integer literals (e.g., 100)
- Floating-point literals (e.g., 3.14)
- Character literals (e.g., 'A')
- String literals (e.g., "Hello")
- Boolean literals (true, false)

(3) What are conditional statements in C++? Explain the if-else and switch statements,What is the difference between for, while, and do-while loops in C++? ,How are break and continue statements used in loops? Provide examples,Explain nested control structures with an example.

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

  In C++, conditional statements are used to make decisions in a program. They check whether a condition is true or false, and based on that, different blocks of code are executed.

if-else Statement:

- The if statement checks a condition.
- If the condition is true, it runs a block of code.
- If the condition is false, the else block runs.

Syntax:

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

switch Statement:

- The switch statement is used when we have **many options** based on a single value.
- It checks the value and runs the matching case.

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

```
    case value2:
        // code
        break;
    default:
        // code if no case matches
}
```

- **<u>What is the difference between for, while, and do-while loops in C++?</u>**

In C++, loops are used to **repeat a block of code** multiple times. The three main types of loops are: for, while, and do-while. All three are used for looping, but their working is slightly different.

◈ for loop

- Used when we **know in advance** how many times we want to run the loop.
- Initialization, condition, and update are written in one line.

Syntax:

```
for (int i = 0; i < 5; i++) {
    cout << i << " ";
} // This loop will print: 0 1 2 3 4
```

◈ while loop

- Used when we **don't know how many times** the loop will run — only condition is checked.
- It checks the condition **before** running the loop.

Syntax:

```
int i = 0;
while (i < 5) {
    cout << i << " ";
```

```
    i++;
} // It will also print: 0 1 2 3 4
```

◈ do-while loop

- It is similar to while loop, **but it runs the code at least once**, even if the condition is false.
- Condition is checked **after** running the loop.

Syntax:

```
int i = 0;
do {
    cout << i << " ";
    i++;
} while (i < 5);    // Output: 0 1 2 3 4
```

- **How are break and continue statements used in loops? Provide**

Break and continue statements are used to control the flow of loops.

- break is used to stop the loop completely. When break runs, the loop ends, and the program moves to the next part after the loop.
- continue is used to skip the current step in the loop and move to the next step directly.

Example:

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        continue;  // it skips 3
    }
    if (i == 5) {
        break;     // it stops when i is 5
    } System.out.println(i);
}   //output      1 2 4
```

- **<u>Explain nested control structures with an example.</u>**
- Nested control structures mean using one control structure inside another.
  For example, a loop inside another loop, or an if-statement inside a loop.
- **Example:**

```
for (int i = 1; i <= 2; i++) {
   for (int j = 1; j <= 3; j++) {
      System.out.println("i = " + i + ", j = " + j);
   }
}
```

Output

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

Explanation:

Here, one for loop is inside another for loop.
This is a nested loop, which is a type of nested control structure.

In short:
Nested control structures help to do more complex tasks, like working with patterns, tables, or multiple conditions.

(4) What is a function in C++? Explain the concept of function declaration, definition, and calling, What is the scope of variables in C++? Differentiate between local and global scope, Explain recursion in C++ with an example, What are function prototypes in C++? Why are they used?

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

What is a function in C++?

A function is a block of code that performs a specific task.
Instead of writing the same code again and again, we write it once in a function and use it whenever needed.

1. **Function Declaration:**

This tells the compiler about the function name, return type, and parameters (if any).
It is written before main().

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

2. **Function Definition:**

This is where we write the **actual code** of the function.
It tells what the function will **do**.

```
cpp

CopyEdit

int add(int a, int b) {

   return a + b;

}
```

3**. Function Calling:**

This is how we **use** the function inside main() or another function.

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

Full Example:

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

int add(int a, int b);  // declaration

int main() {
   int sum = add(4, 6);  // calling
   cout << "Sum is: " << sum;
   return 0;
}

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

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

What is the scope of variables in C++?

Scope means the area in the program where a variable can be used or accessed.

Types of Scope:

1. Local Scope:

- A local variable is declared inside a function or block.
- It can be used only inside that function/block.
- It is destroyed when the function ends.

```
   void show() {
   int a = 10;  // local variable
```

```
    cout << a;
}
```

2. Global Scope:

- A global variable is declared outside all functions, usually at the top.
- It can be used in any function of the program.

```
int a = 10;  // global variable

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


```
int a = 10;  // global variable

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


Difference Between Local and Global Scope:

Feature	**Local Variable**	**Global Variable**
Declared Inside	A function or block	Outside all functions
Accessible In	Only that function/block	All functions
Lifetime	Ends when function ends	Till program ends
Memory Usage	Created during function call	Created at program start

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

What is Recursion in C++?

Recursion is when a function **calls itself** to solve a problem.
It is used when a task can be broken into **smaller sub-tasks** of the same type.

Example: Find Factorial Using Recursion

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

int factorial(int n) {
    if (n == 0 || n == 1)
        return 1;  // base case
    else
        return n * factorial(n - 1);  // recursive call
}

int main() {
    int num = 5;
    cout << "Factorial of " << num << " is: " << factorial(num);
    return 0;
}
```

How it works:

For factorial(5) →
5 * factorial(4)
5 * 4 * factorial(3)
5 * 4 * 3 * factorial(2)
5 * 4 * 3 * 2 * factorial(1)
5 * 4 * 3 * 2 * 1 = 120

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

What are Function Prototypes in C++?

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

Syntax:

return_type function_name(parameter_list);

Example:

int add(int a, int b); // function prototype

Why are Function Prototypes Used?

- To tell the compiler in advance about the function.
- So that we can call the function before its definition.
- Helps to avoid errors related to unknown functions.

Example Program:

```
#include <iostream>

using namespace std;


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


int main() {
   cout << "Sum: " << add(3, 4);  // function call
   return 0;
}


int add(int a, int b) {  // function definition
```

```
    return a + b;
}
```

(5) What are arrays in C++? Explain the difference between single-dimensional and multi-dimensional arrays, Explain string handling in C++ with examples, How are arrays initialized in C++? Provide examples of both 1D and 2D arrays, Explain string operations and functions in C++.

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

Array in C++ is a data structure where we can store many values of the same type together. For example, if we want to store 10 numbers, instead of making 10 separate variables, we can use one array.

Single-dimensional array:
It has only one row. It looks like a simple list. Example: marks[5] → we can store 5 students' marks.

Multi-dimensional array:
It has rows and columns. It looks like a table. Example: int matrix[3][3] → it has 3 rows and 3 columns. Mostly used for tables or matrices.

In short: single-dimensional array is like a line, and multi-dimensional array is like a table.

- **Explain string handling in C++ with examples**

In C++, string handling means working with text (group of characters). A string is just a collection of characters, like "Hello" or "C++ is fun".

There are two ways to handle strings in C++:

1. Using character arrays (C-style strings):
    - String is stored as an array of characters ending with \0 (null character).
    - Example:

      #include <iostream>

      using namespace std;

```
int main() {
    char name[20] = "Sanam";
    cout << "Name is: " << name;
    return 0;
}
```

2. Using string class (C++ style strings):

- C++ has a string class in <string> header, which makes handling easier.
- Example:

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

int main() {
    string name = "Sanam Bhoraniya";
    cout << "Name is: " << name << endl;
    cout << "Length of string: " << name.length();
    return 0;
}
```

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

In C++, arrays can be initialized when we declare them. Initialization means giving values to the array.

1D Array (One-dimensional):
This is like a list of elements.

Example:

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

int main() {
   // Method 1: Direct initialization
   int numbers[5] = {10, 20, 30, 40, 50};

   // Method 2: Partial initialization (rest will be 0)
   int marks[5] = {90, 80};

   // Printing
   for(int i=0; i<5; i++) {
      cout << numbers[i] << " ";
   }
   return 0;
}
```

2D Array (Two-dimensional):
This is like a table (rows and columns).
Example:

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

int main() {
   // Direct initialization
   int matrix[2][3] = {
```

```
        {1, 2, 3},
        {4, 5, 6}
    };

    // Printing
    for(int i=0; i<2; i++) {
        for(int j=0; j<3; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

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

In C++, strings are used to store and work with text. The **string class** in C++ provides many operations and functions that make it easy to handle strings.

Common String Operations / Functions:

1. Length of string → .length() or .size()
   Example:

   ```
   string s = "Hello";
   cout << s.length();   // Output: 5
   ```

2. Concatenation (joining strings) → + or .append()

   ```
   string a = "Good ";
   string b = "Morning";
   string c = a + b;       // Using +
   cout << c << endl;

   a.append(" Night");     // Using append
   ```

```
cout << a;
```

3. Access character → []

```
string s = "World";
cout << s[0];   // Output: W
```

4. Compare strings → .compare()

```
string x = "Apple";
string y = "Banana";
if(x.compare(y) == 0)
   cout << "Equal";
else
   cout << "Not Equal";
```

5. Substring → .substr(start, length)

```
string s = "Programming";
cout << s.substr(0, 4);   // Output: Prog
```

6. Insert and Erase

```
string s = "Hello";
s.insert(5, " World");   // Insert
cout << s << endl;       // Output: Hello World

s.erase(5, 6);          // Erase " World"
cout << s;              // Output: Hello
```

7. Find position of word/character → .find()

```
string s = "I love C++";
cout << s.find("love");   // Output: 2
```

(6) Explain the key concepts of Object-Oriented Programming (OOP), What are classes and objectsin C++? Provide an example, What isinheritance in C++? Explain with an example, What is encapsulation in C++? How isit achieved in classes?

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

Object-Oriented Programming (OOP) is a way of writing programs where we organize everything into objects. Objects are created from classes, and they have data (variables) and actions (functions).

The main concepts of OOP are:

1. Class:
   A class is like a blueprint or template. It defines how an object will look and behave.
   Example: class Car { ... };
2. Object:
   An object is a real thing created from a class. If class is a blueprint, then object is the house built from it.
   Example: Car c1;
3. Encapsulation:
   Wrapping data (variables) and functions into a single unit (class). It also means hiding data using private, public, and protected.
   Example: private balance in a Bank class.
4. Abstraction:
   Showing only the important things and hiding the details.
   Example: When we drive a car, we use the steering and pedals, but we don't see the internal engine work.
5. Inheritance:
   One class can get properties and functions of another class.
   Example: A "Student" class can inherit from a "Person" class.
6. Polymorphism:
   It means "many forms." A function or operator can behave differently depending on the situation.
   Example: + can add numbers, and also join (concatenate) strings.

- **What are classes and objectsin C++? Provide an example.**

In C++, a **class** is a blueprint that defines how an object will look and work. It contains data (variables) and functions.
An **object** is a real thing created from a class. We can create many objects from one class.

Example:

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

// Class definition
class Car {
public:
    string brand;
    int year;

    void show() {
        cout << "Brand: " << brand << ", Year: " << year << endl;
    }
};

int main() {
    // Creating objects from class
    Car car1;
    car1.brand = "BMW";
    car1.year = 2022;

    Car car2;
```

```
    car2.brand = "Toyota";
    car2.year = 2020;

    // Using objects
    car1.show();
    car2.show();

    return 0;
}
```

Output

Brand: BMW, Year: 2022

Brand: Toyota, Year: 2020

- **What isinheritance in C++? Explain with an example.**

Inheritance in C++ means that one class can take properties and functions from another class.

- The class that gives its features is called base class.
- The class that receives features is called derived class.

Inheritance helps in code reusability, because we don't have to write the same code again and again.

Example:

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

// Base class
class Person {
public:
    string name;
```

```
    int age;

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

// Derived class (inheriting Person)
class Student : public Person {
public:
    int rollNo;

    void showStudent() {
        cout << "Roll No: " << rollNo << endl;
    }
};

int main() {
    Student s1;
    s1.name = "BADI MUKHTAR";
    s1.age = 21;
    s1.rollNo = 05;

    s1.showPerson();   // From base class
    s1.showStudent();  // From derived class
```

```
    return 0;
}
```

Output

```
Name: BADI MUKHTAR, Age: 21
Roll No: 05
```

- **What is encapsulation in C++? How isit achieved in classes?**

Encapsulation in C++ means wrapping data and functions together in a single unit (class).
It also means hiding the data so that it cannot be accessed directly from outside the class. Instead, we use functions (getters/setters) to access or change the data.

This is achieved in C++ by using access specifiers:

- private → data hidden from outside.
- public → functions to access/modify data.

Example:

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

class BankAccount {
private:
    int balance;   // private data (hidden)

public:
    // function to set balance
    void setBalance(int b) {
        balance = b;
    }
```

```
    // function to get balance
    int getBalance() {
        return balance;
    }
};

int main() {
    BankAccount acc;
    acc.setBalance(5000);         // setting balance using function
    cout << "Balance: " << acc.getBalance();  // accessing using function

    return 0;
}
```

OUTPUT

Balance: 5000