Module-3 Introduction to OOPS Programming

THEORY EXERCISE:-

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

- **Ans:-**

Aspect	**Procedural Programming**	**Object-Oriented Programming (OOP)**
Definition	Programming paradigm based on functions or procedures.	Programming paradigm based on objects that encapsulate data and behavior.
Focus	Focuses on **functions/procedures** to operate on data.	Focuses on **objects** that combine data (attributes) and behavior (methods).
Data Handling	Data is separate from functions; global data is common.	Data is encapsulated within objects. Data and behavior are bound together.
Modularity	Program is divided into functions.	Program is divided into classes and objects.
Reusability	Limited reusability; functions can be reused, but data management is harder to extend.	High reusability via inheritance and polymorphism.
Data Security	Data is exposed to the whole program (unless special techniques are used).	Data is hidden and accessible only through methods (Encapsulation).
Example Languages	C, Pascal, Fortran	C++, Java, Python, C#
Approach	Top-down approach.	Bottom-up approach.
Ease of Maintenance	Less maintainable for large projects because of scattered global data and lack of structure.	Easier to maintain and scale due to modularity and encapsulation.
Polymorphism and Inheritance	Not supported directly.	Supported: enables code reuse and dynamic method binding.
State Management	State is managed by global variables and parameters passed to functions.	Each object maintains its own state.

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

➢ **Ans:-**

Advantage	Description
Encapsulation	Combines data and methods; hides internal state.
Reusability	Inheritance allows reuse of code.
Polymorphism	Same interface, different implementations.
Modularity	Organized into classes & objects for easier management.
Maintainability	Easier to update, fix, and extend code.
Abstraction	Hide complexity from users, exposing only necessary features.
Real-World Modeling	Objects map naturally to real-world entities.

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

➢ **Ans:-**

Step 1: Download and Install MinGW.
Step 2: Add MinGW to System Path.
Step 3: Install Visual Studio Code.
Step 4: Install C/C++ Extension.
Step 5: Create a New C++ File.
Step 6: Build and Run

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

➢ **Ans:-**

The main input/output (I/O) operations are handled by the **(<iostream>)** library, The two primary objects for I/O are **(cin)** for input and **(cout)** for output.

Standard Output (cout):-

Cout is a stream object used to send data to the standard output device, which is typically the console. You use the insertion operator **(<<)** to direct data into the stream.

Standard Input (cin):-

Cin is a stream object used to read data from the standard input device, usually the keyboard. You use the extraction operator (>>) to pull data from the stream and store it in a variable.

Main Components of <iostream>:-

The library is built around the concept of **streams**, which are sequences of bytes. Data flows from a source (like a keyboard or file) into a stream and from a stream to a destination (like the screen or another file).

```cpp
#include<iostream> //Header File
using namespace std;
int main()
{
   int a;
   cout<<"\nEnter Your A value::";    //cout
   cin>>a;    //cin
   cout<<"\n Your A value is::"<<a;
   return 0;
}
```

Output:-

```
PS C:\4Aug_Java_Raj\Cpp> g++ .\1st.cpp

PS C:\4Aug_Java_Raj\Cpp> .\a.exe

Enter Your A value::Raj

 Your A value is::0
```

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

- **Ans:-**

Basic Data Types:-

int → Integer numbers (e.g., int age = 25;)
float → Floating-point numbers (e.g., float pi = 3.14;)
double → Double precision floating-point (e.g., double price = 99.99;)
char → Single character (e.g., char grade = 'A';)
bool → Boolean value (true or false)

Derived Data Types:-

Arrays (e.g., int arr[5];)
Pointers (e.g., int* ptr;)

User-Defined Types:-

struct, class, enum

Ex:-

```cpp
#include <iostream>
using namespace std;
int main() {
    // Basic Data Types
    int age = 25;
    float pi = 3.14;
    double price = 99.99;
    char grade = 'A';
    bool isStudent = true;
    // Array
    int numbers[3] = {10, 20, 30};
    // Pointer
    int* ptr = &age;
    cout << "Age: " << age << endl; // int
    cout << "Pi: " << pi << endl;  //float
    cout << "Price: $" << price << endl; // double
    cout << "Grade: " << grade << endl; //char
    cout << "Is Student: " << isStudent << endl;  //boolean
    cout << "Array numbers: ";
    for (int i = 0; i < 3; i++)
        cout << numbers[i] << " "; // array
    cout << endl;
    cout << "Pointer points to age: " << *ptr << endl; // pointer
    return 0;
}
```

Output:-

```
Age: 25
Pi: 3.14
Price: $99.99
Grade: A
Is Student: 1
Array numbers: 10 20 30
Pointer points to age: 25
```

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

➢ **Ans:-**

✓ Feature	Implicit Type Conversion	Explicit Type Conversion
Definition	Automatic conversion by the compiler	Manual conversion by the programmer
Also called	Type Coercion	Type Casting
Example	int a = 10; float b = a; (int → float automatically)	float pi = 3.14; int x = (int)pi; (explicit cast)
Purpose	To prevent data loss when converting to a larger type	When programmer wants to force conversion despite potential data loss
Programmer control	No control (done automatically)	Programmer controls the conversion
Risk of Data Loss	Low (because compiler handles safe conversions)	Higher (may lose data if not careful)

✓ Feature	Implicit Type Conversion	Explicit Type Conversion
Syntax Example	float b = a;	int x = (int)3.14; or int x = static_cast<int>(3.14);

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

➢ **Ans:-**

Arithmetic Operators:- These are used for mathematical calculations.

Operator	Name	Example
+	Addition	int sum = 5 + 3; (8)
-	Subtraction	int diff = 10 - 4; (6)
*	Multiplication	int prod = 2 * 6; (12)
/	Division	int div = 15 / 5; (3)
%	Modulo	int rem = 10 % 3; (1)
++	Increment	int x = 5; x++; (6)
--	Decrement	int y = 8; y--; (7)

Relational Operators:- These compare two values and return a bool (true or false) result.

Operator	Name	Example
==	Equal to	check = (7 == 7); (true)
!=	Not equal to	check = (5 != 3); (true)
>	Greater than	check = (10 > 5); (true)
<	Less than	check = (4 < 9); (true)
>=	Greater than or equal to	check = (6 >= 6); (true)
<=	Less than or equal to	check = (2 <= 5); (true)

Logical Operators:- These combine or modify boolean expressions.

Operator	Name	Example
&&	Logical AND	if (c = 5 && d = 2)
!!	Logical OR	if ((c==5) || (d>5))
!	Logical NOT	if (!(c==5))

Assignment Operators:- These are used to assign a value to a variable. The simple assignment operator is =, but there are also compound assignment operators.

Operator	Name	Example	Equivalent to
=	Simple Assignment	int x = 10;	-
+=	Add and Assign	x += 5;	x = x + 5;
-=	Subtract and Assign	x -= 3;	x = x - 3;
*=	Multiply and Assign	x *= 2;	x = x * 2;
/=	Divide and Assign	x /= 4;	x = x / 4;

Bitwise Operators:- These perform operations on the individual bits of integer data.

Operators	Name
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

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

➢ **Ans:-**

Purpose of Constants in C++

- Constants are fixed values that do not change during program execution.
- Declared using the keyword (**const**).
- Used to prevent accidental modification of important values.

Ex:-

```
#include <iostream>
using namespace std;
int main() {
   const double Pie= 3.14; // Constant value
   double radius = 5.0;
   double area = Pie * radius * radius;
   cout << "The area is: " << area << std::endl;
   return 0;
}
```

Output:-

```
The area is: 78.5
```

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

➢ **Ans:-**

- Used to perform different actions based on conditions.

Common types: if, if-else, nested-if, switch.

if Statement:-
An if statement executes a block of code only if a specified condition is true. It's the simplest form of a conditional statement.

if-else Statement:-
The if-else statement allows you to execute one block of code if the condition is true and a different block of code if it's false.
Nested if Statement:-
A nested if statement is an if or if-else statement placed inside another if or else block. This is used when you need to check for a condition only after another condition has already been met.

switch Statement:-
A switch statement is an efficient alternative to a long chain of if-else if statements. It tests the value of a variable against a list of specific cases.

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

➢ **Ans:-**

Feature	for Loop	while Loop	do-while Loop
Syntax Example	for (initialization; condition; update) {}	while (condition) {}	do { } while (condition);
Use Case	When the number of iterations is known beforehand	When the number of iterations is not known	When the loop must run at least once
Condition Checked	Before each iteration	Before each iteration	After each iteration
Minimum Execution	May execute zero times if condition is false	May execute zero times if condition is false	Executes at least once, even if condition is false
Example	for (int i = 0; i < 5; i++) { /* code */ }	int i = 0; while (i < 5) { i++; /* code */ }	int i = 0; do { i++; /* code */ } while (i < 5);
When to Use	Best for fixed number of iterations	Best when condition depends on dynamic input	Best when you need at least one iteration always

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

➢ **Ans:-**

Break statement:-
It is used to immediately exit the loop when a certain condition is met.

Continue statement:-
It is used to skip the current iteration and proceed to the next iteration of the loop.

Goto statement:-
It is used to jump to a labeled statement in the program.
It provides unconditional jump to a specified label, which is not recommended in modern programming because it makes the code harder to read and maintain.

Ex:-

```
#include<iostream>
using namespace std;
int main()
{
    int i;
    i=1;
    while(1)
    {
        if(i==10)
        {
        break;
        }
        cout<<"Hello "<<i<<"\n";
        i++;
    }
    int i;
    i=0;
    while(i<10)
    {
        i++;
        if (i==5)
        {
            continue;
        }
        cout<<"Hello "<<i<<"\n";
    }
    int i;
    i=1;
    label:
    if(i<=5)
    {
        cout<<"Hello "<<i<<"\n";
        i++;
        goto label;
    }
}
```

12. Explain nested control structures with an example.

➢ **Ans:-**

Nested control structures mean putting one control structure (like if, for, or while) inside another.
This allows more complex decision-making and repeated operations.

Ex:-

```
#include<iostream>
using namespace std;
int main()
{
    for (int i = 1; i <= 2; i++)
    {
        for(int j=1;j<=10;j++)
        {
            cout<<i<<"*"<<j<<"="<<i * j<<endl;
        }
        cout<<"\n";
    }
}
```

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

➢ **Ans:-**

A function is a block of code that performs a specific task.

Declaration: Tells function name, return type, and parameters.
Example: int add(int, int);

Definition: Provides the function body.
Example: int add(int a, int b) { return a + b; }

Calling: Executes the function.
Example: add(5, 3);

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

➢ **Ans:-**

The scope of a variable refers to the region in the code where the variable is accessible (can be used).

Feature	Local Variable	Global Variable
Where Declared	Inside a function or block {}	Outside all functions (usually at the top of the file)
Accessibility	Accessible only within the function/block it is declared	Accessible throughout the entire program
Lifetime	Exists only during function execution	Exists throughout program execution
Example	cpp int main() { int x = 5; /* x is local */ }	cpp int x = 10; int main() { cout << x; }
Use Case	Temporary data for functions	Shared data among functions (used sparingly)

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

➢ **Ans:-**

Recursion is a programming technique where a function calls itself to solve a smaller part of the problem.

It must have a base case to stop the recursion, otherwise it will go into infinite calls and cause a crash (stack overflow).

Ex:-

```
#include <iostream>
using namespace std;
// Recursive function to calculate factorial
int factorial(int n) {
    if (n == 0)  // Base case: factorial of 0 is 1
        return 1;
    else
        return n * factorial(n - 1);  // Recursive call
}
int main() {
    int number = 5;
    cout << "Factorial of " << number << " is " << factorial(number) << endl;
    return 0;
}
```

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

➢ **Ans:-**

Function Prototype:-

A function prototype is a declaration of a function that tells the compiler:

- The function's name
- Return type
- Parameters (number and types)

It appears before main() or function definition.

Why are the used:-

- To inform the compiler about the function before it is used (called).
- Allows functions to be called before their actual definition in the code.
- Helps the compiler check if the function is used correctly (correct arguments & return type).
- Prevents compilation errors in case of out-of-order function definitions.

17. 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.
It allows us to store multiple values of the same type using a single name and access them using an index.

Syntax:-

data_type arrayName[arraySize];

Difference between single-dimensional and multi-dimensional arrays:-

Feature	Single-Dimensional Array	Multi-Dimensional Array
Structure	Linear list of elements	Grid or table (rows & columns)
Access	array[index]	array[row][column]
Example	int arr[5];	int matrix[2][3];
Use Case	Store list of values (e.g., marks of 5 students)	Store matrix or table data (e.g., 2D game board)

18. Explain string handling in C++ with examples.
Ans:-

We use string from the standard library.
Strings in C++ are objects of class std::string (from <string>).
Much easier to handle.

Ex:-

```
#include <iostream>
#include <string>
using namespace std;
int main() {
   string str1 = "Hello";
   string str2 = "World";
   // Combine strings
   string result = str1 + " " + str2;

   cout << result << endl;         // Output: Hello World
   cout << "Length: " << result.length() << endl;   // Output: 11

   return 0;
}
```

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

An array in C++ is a collection of elements of the same type stored in contiguous memory.

Initialization of 1D Array (Single-Dimensional Array):-

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

```
int main() {
   int numbers[5] = {10, 20, 30, 40, 50};
   // Output the elements
   for (int i = 0; i < 5; i++) {
      cout << numbers[i] << " ";
   }
   // Output: 10 20 30 40 50
   return 0;
}
```

Initialization of 2D Array (Two-Dimensional Array):-

```
#include <iostream>
using namespace std;
int main() {
   int matrix[2][3] = { {1, 2, 3}, {4, 5, 6} };
   // Output elements
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 3; j++) {
         cout << matrix[i][j] << " ";
      }
      cout << endl;
   }

   // Output:
   // 1 2 3
   // 4 5 6
   return 0;
}
```

20. **Explain string operations and functions in C++.**
Ans:-

Concatenation:-

Purpose: Join two strings together
Syntax:-
```
string str3 = str1 + str2;
str1.append(str2);
```

Length of String:-

Purpose: Get the number of characters in a string
Syntax:-
```
size_t len = str.length();
```

Access Character by Index:-

Purpose: Access individual characters in a string
Syntax:-

char ch = str[index]; // Example: str[0]

Compare Strings:-

Purpose: Compare two strings
Syntax:-
if (str1 == str2) { /* Equal */ }
int result = str1.compare(str2); // Returns 0 if equal

Append String:-

Purpose: Add text at the end of the string
Syntax:-
str.append(" More Text");

Substring Extraction:-

Purpose: Get part of a string
Syntax:-
string sub = str.substr(start_index, length);

Clear String:-

Purpose: Remove all contents of a string
Syntax:-
str.clear();

Check if String is Empty:-

Purpose: Check if the string has no characters
Syntax:-
bool isEmpty = str.empty();

21. Explain the key concepts of Object-Oriented Programming (OOP).
Ans:-

Key Concepts:-

Encapsulation:-

Combines data and functions in a class.
Keeps data private and accessible via public methods.

Abstraction:-

Hides internal details.
Shows only essential features to the user.

Inheritance:-

A class (derived) gets properties and methods from another class (base).
Helps reuse code.

Polymorphism:-

Same function name behaves differently (Overloading and Overriding).

22. **What are classes and objects in C++? Provide an example.**
Ans:-

Classes and Objects:-
Class:-

A class is a blueprint or template for creating objects.
It contains data members (variables) and member functions (methods) that define the behavior of objects.

Object:-

An object is an instance of a class.
It holds actual values and can perform actions using its functions.

Ex:-
```
#include <iostream>
using namespace std;

// Class definition
class Car {
public:
    string brand;
    int year;
    void display() {
        cout << "Brand: " << brand << ", Year: " << year << endl;
    }
};
int main() {
    // Create an object of class Car
    Car car1;
    // Set values
    car1.brand = "Toyota";
    car1.year = 2020;
    // Call member function
    car1.display();
    return 0;
}
```

23. **What is inheritance in C++? Explain with an example.**
Ans:-

Inheritance is an OOP concept where a class (called Derived Class) inherits properties (data members) and behaviors (functions) from another class (called Base Class).
It helps in code reusability and creating a relationship between classes.

Ex:-

```
#include <iostream>
using namespace std;
// Base Class
class Animal {
public:
    void eat() {
        cout << "Animal eats food" << endl;
    }
};
// Derived Class
class Dog : public Animal {  // Inherits from Animal
public:
    void bark() {
        cout << "Dog barks" << endl;
    }
};

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

24. What is encapsulation in C++? How is it achieved in classes?
Ans:-

Encapsulation:-

Encapsulation is an OOP concept where data (variables) and functions (methods) are bundled together in a class.
It restricts direct access to some of the object's components, protecting the internal state of the object.
Access is controlled using access specifiers: private, protected, and public.

Achieved in Classes:-

Data members are usually made private so they cannot be accessed directly from outside the class.

Public setter and getter methods are provided to control reading and modifying the private data safely.

Ex:-

```
#include <iostream>
using namespace std;
class Person {
private:
   int age;  // Private variable
public:
   void setAge(int a) {  // Setter function
      age = a;
   }
   int getAge() {       // Getter function
      return age;
   }
};

int main() {
   Person p;
   p.setAge(30);              // Set age
   cout << "Age is: " << p.getAge() << endl;  // Output: Age is: 30
   return 0;
}
```