MODULE: 3

SE – Introduction to OOPS Programming

Introduction to C++:-

1. What is OOP? List OOP concepts.

➢ OOP stands for **Object-Oriented Programming**. As the name suggests uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

~ There are some basic concepts that act as the building blocks of OOPs i.e.: Class, Object, Encapsulation, Abstraction, Polymorphism, Inheritance, Dynamic Binding, Message Passing.

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

POP	OOP
The program is divided into small parts called functions. (small tasks)	The program is divided into small parts called objects. (real-world entities)
Follows a top-down approach (solves main task first, then details).	Follows a bottom-up approach (builds objects first, then combines).
There is no access specifier in procedural programming.	Has access specifiers like private, public, protected.
Adding new data and functions is not easy.	Adding new data and function is easy.
It does not have any proper way of hiding data so it is less secure.	It provides data hiding so it is more secure.
Not possible to overload functions.	Supports overloading (same function name, different tasks).
There is no concept of data hiding and inheritance.	The concept of data hiding and inheritance is used. (reuse properties of other classes)
The function is more important than the data.	Data is more important than function.
It is based on the unreal world.	It is based on the real world. (objects represent entities like car, person)
It is used for small to medium-sized programs.	It is used for designing large and complex programs.
Uses procedure abstraction (hides implementation details in functions).	Uses data abstraction (hides data and shows necessary details).
No code reusability (every function is written independently).	Code reusability through inheritance and classes.
Ex:- C, FORTRAN, Pascal, Basic, etc.	Ex:- C++, Java, Python, C#, etc.

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

➢ **Advantages of OOP:-**

- Improved software-development productivity:- OOP is modular, as it provides separation of duties in object-based program development. It is also extensible, as objects can be extended to include new attributes and behaviors. Objects can also be reused within an across applications. Because of these three factors – modularity, extensibility, and reusability – object-oriented programming provides improved software-development productivity over traditional procedure-based programming techniques.
- Improved software maintainability:- Since the design is modular, part of the system can be updated in case of issues without a need to make large-scale changes.
- Faster development:- Reuse enables faster development. Object-oriented programming languages come with rich libraries of objects, and code developed during projects is also reusable in future projects.
- Lower cost of development:- The reuse of software also lowers the cost of development. Typically, more effort is put into the object-oriented analysis and design, which lowers the overall cost of development.
- Higher-quality software:- Faster development of software and lower cost of development allows more time and resources to be used in the verification of the software. Although quality is dependent upon the experience of the teams, object oriented programming tends to result in higher-quality software.

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

➢ **Steps to Set Up C++ Development Environment**

- Install Visual Studio Code (VS):- Download and install VS Code from the official website.
~ Install extensions like C/C++ (Microsoft) and optionally Code Runner for easier coding.
- Download MinGW:- Go to the MinGW website and download the MinGW installer.
~ Install MinGW, selecting the necessary packages (e.g., mingw32-gcc-g++ for C++).
- Set Environment Variables:- Locate the MinGW installation directory (e.g., C:\MinGW\bin).
~ Add the bin folder to the PATH environment variable:- Right-click "This PC" → Properties → Advanced system settings → Environment Variables.
Edit the PATH variable and add C:\MinGW\bin.
- Verify GCC Installation:- Open a terminal or command prompt and run:
gcc --version
g++ --version
~ If installed correctly, it will display the GCC version.
- Configure VS Code for C++:- Open VS Code and create a new C++ file (.cpp).
~ Configure tasks:- Open Command Palette (Ctrl+Shift+P), search for Tasks: Configure Task, and select Create tasks.json file from template → GCC.
~ Configure launch settings:- Open Run and Debug (Ctrl+Shift+D), click create a launch.json file, and select C++ (GDB/LLDB).
- Test the Setup:- Write a simple program, e.g., hello.cpp:

```
#include <iostream>
int main() {
   std::cout << "Hello, World!" << std::endl;
   return 0;
}
```

~ Compile and run using the terminal:

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

- Optional: Use Code Runner:- If Code Runner is installed, press Ctrl+Alt+N to compile and run the program automatically.

5. Help students understand how to install, configure, and run programs in an IDE.

➢ An Integrated Development Environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. It typically includes a code editor, compiler, debugger, and build automation tools.

- Installing an IDE:- Example: Installing Code::Blocks for C++

~ Go to Code::Blocks website.
~ Download the version with MinGW (compiler included).
~ Install it by following the setup instructions.
~ Open Code::Blocks after installation.

- Configuring the IDE:- Basic Steps (Code::Blocks as an Example):

~ Open Code::Blocks
~ Go to Settings → Compiler
~ Ensure the correct compiler (e.g., GNU GCC Compiler) is selected.
~ Set the path of the compiler if necessary.

- Writing and Running a Program:- Example: Running a C++ Program in Code::Blocks

~ Open Code::Blocks.
~ Click on File → New → Project → Select Console Application.
~ Choose C++ and provide a project name and location.
~ Click Finish to create the project.
~ Open main.cpp file and enter the following code:
~ cpp

```
#include <iostream>
using namespace std;
int main() {
   cout << "Hello, World!";
   return 0;
}
```

~ Click Build and Run (F9).
~ The output will appear in the console window.

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

➢ In C++, input and output operations are handled using the iostream library, which provides two main objects:

~ **cin →** Standard input (used for reading input from the keyboard).

~ **cout →** Standard output (used for displaying output on the screen).

~ Additionally, C++ provides cerr for error messages and clog for logging.

- Standard Output Stream – cout:- The C++ **cout** is the instance of the **ostream** class used to produce output on the standard output device which is usually the display screen. The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(<<).

~ Syntax:- cout << value/variable;

~ For example, if we want to print text "Welcome " on the display, we can use the cout as shown:

```
#include <iostream>
using namespace std;
int main() {
// Printing the given text using cout
cout << "Welcome";
}
```

Output:- Welcome

- Standard Input Stream – cin:- The C++ **cin** statement is the instance of the class **istream** and is used to read input from the standard input device which is usually a keyboard. The extraction operator (>>) is used along with the object cin for extracting the data from the input stream and store it in some variable in the program.

~ Syntax:- cin >> variable;

~ For example, if we want to ask user for his/her age, then we can use cin as shown:

```
#include <iostream>
using namespace std;
int main() {
int age;
// Output a label
cout << "Enter your age:";
// Taking input from user and store it in variable
cin >> age;
// Output the entered age
cout << "Age entered: " << age;
return 0;
}
```

Input:- Enter your age: 18 *(18 entered by the user)*

Output:- Your age is: 18

Variables, Data Types, and Operators:-

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

➢ In C++, data types specify the type of data that a variable can store. There are several built-in data types categorized into different groups.

- **Primary (Fundamental) Data Types:-** These are the most basic data types in C++.

DATA TYPE	DESCRIPTION	EXAMPLE
Int	Stores whole numbers	int x = 10;
Float	Stores decimal numbers (single precision)	float y = 3.14f;
Double	Stores large decimal numbers (double precision)	double z = 3.14159;
Char	Stores single characters	char letter = 'A';
Bool	Stores true or false values	bool isValid = true;
Void	Represents "nothing" (used for functions that return nothing)	void myFunction();

- **Derived Data Types:-** These data types are derived from fundamental data types.

DATA TYPE	DESCRIPTION	EXAMPLE
Array	Stores multiple values of the same type	int arr[5] = {1, 2, 3, 4, 5};
Pointer	Stores the memory address of another variable	int *ptr = &age;
Reference	Acts as an alias for another variable	int &ref = age;

- **User-Defined Data Types:-** These are created by the programmer using fundamental data types.

DATA TYPE	DESCRIPTION	EXAMPLE
struct	Groups different data types together	struct Person { string name; int age; };
class	Defines a blueprint for objects	class Car { string model; int year; };
union	Similar to a struct but shares memory among members	union Data { int x; float y; };
enum	Define set of named integer constants	enum Color {RED, GREEN, BLUE};

- **Modifiers (Type Modifiers):-** Modifiers alter the size and range of fundamental data types.

MODIFIER	USED WITH	EXAMPLE
signed	int, char	signed int x = -10;
unsigned	int, char	unsigned int x = 10;
short	int	short int y = 32767;
long	int, double	long int z = 100000;

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

IMPLICIT	EXPLICIT
The compiler automatically converts one data type to another without programmer intervention.	The programmer manually specifies the conversion between data types using casting.
Also called "automatic type conversion."	Also called "manual type conversion."
Performed automatically by the compiler.	Performed by the programmer using type casting. Uses (type) or static_cast<type>().
For widening conversions (e.g., int to double)	For narrowing conversions or overriding compiler behavior
Prevents data loss. No data loss (except when converting float → int).	Possible data loss (e.g., double → int).
Follows the type hierarchy: char → short → int → long → float → double → long double	Requires using casting operators such as: C-style cast: (type) C++ static cast: static_cast<type>()

#include <iostream>
using namespace std;
int main() {
int a = 10;
double b = a;
// Implicit conversion from int to double
cout << "Integer value: " << a << endl;
cout << "Converted to double: " << b << endl;
// Outputs 10.0
return 0;
}
	#include <iostream>
	using namespace std;
	int main() {
	double num = 10.75;
	int intNum = (int)num;
	// Explicit conversion from double to int
	cout << "Original double: " << num << endl;
	cout << "Converted to int: " << intNum << endl;
	// Outputs 10 (fractional part removed)
	return 0;
	}

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

- **Arithmetic operators** are used to perform arithmetic or mathematical operations on the operands. For example, '+' is used for addition.

OPERATOR	DESCRIPTION	EXAMPLE
+ (Addition)	Adds two operands.	a + b
- (Subtraction)	Subtracts second operand from the first.	a – b
* (Multiplication)	Multiplies two operands.	a * b
/ (Division)	Divides first operand by the second operand.	a / b
% (Modulus- remainder)	Returns the remainder an integer division.	a % b

++ (Increment)	Increase the value of operand by 1.	a++, ++a
--(Decrement)	Decrease the value of operand by 1.	b--, --b

- **Relational operators** are used for the comparison of the values of two operands. For example, '>' check right operand is greater.

OPERATOR	DESCRIPTION	EXAMPLE
== (Is Equal To)	Checks both operands are equal	a == b
> (Greater Than)	Checks first operand is greater than the second operand	a > b
>= (Greater Than or Equal To)	Checks first operand is greater than equal to the second operand	a >= b
< (Less Than)	Checks first operand is lesser than the second operand	a < b
<= (Less Than or Equal To)	Checks first operand is lesser than equal to the second operand	a <= b
!= (Not Equal To)	Checks both operands are not equal	a != b

- **Logical operators** are used to combine two or more conditions or constraints or to complement the evaluation of the original condition in consideration. The result returns a Boolean value, i.e., true or false.

OPERATOR	DESCRIPTION	EXAMPLE
&& (Logical AND)	Returns true only if all the operands are true or non-zero.	(a > 5 && b < 10)
|| (Logical OR)	Returns true if either of the operands is true or non-zero.	(a > 15 || b < 25)
! (Logical NOT)	Returns true if the operand is false or zero.	!(a > b)

- **Bitwise operators** are works on bit-level. So, compiler first converted to bit-level and then the calculation is performed on the operands. Only char and int data types can be used with Bitwise Operators.

OPERATOR	DESCRIPTION	EXAMPLE
& (Binary AND)	Copies a bit to the evaluated result if it exists in both operands	a & b
| (Binary OR)	Copies a bit to the evaluated result if it exists in any of the operand	a | b
^ (Binary XOR)	Copies the bit to the evaluated result if it is present in either of the operands but not both	a ^ b
<< (Left Shift)	Shifts the value to left by the number of bits specified by the right operand.	~a
>> (Right Shift)	Shifts the value to right by the number of bits specified by the right operand.	a << 2
~ (One's Complement)	Changes binary digits 1 to 0 and 0 to 1	a >> 2

- **Assignment operators** are used to assign value to a variable. We assign the value of right operand into left operand according to which assignment operator we use.

OPERATOR	DESCRIPTION	EXAMPLE
= (Assignment)	Assigns the value on the right to the variable on the left.	a = b
+= (Add and Assignment)	First add right operand value into left operand then assign that value into left operand.	a += b (a = a + b)
-= (Subtract and Assignment)	First subtract right operand value into left operand then assign that value into left operand.	a -= b (a = a - b)
*= (Multiply and Assignment)	First multiply right operand value into left operand then assign that value into left operand.	a *= b (a = a * b)
/= (Divide and Assignment)	First divide right operand value into left operand then assign that value into left operand.	a /= b (a = a / b)
%= (Modulus & Assignment)		a %= b (a = a % b)

- **Ternary or conditional operator** returns the value, based on the condition.

OPERATOR	DESCRIPTION	EXAMPLE
? : (Condition)	The ternary operator ? determines the answer on the basis of the evaluation of Expression1. If it is true, then Expression2 gets evaluated and is used as the answer for the expression. If Expression1 is false, then Expression3 gets evaluated and is used as the answer for the expression.	Expression1 ? Expression2 : Expression3 (a > b ? a : b)

- Apart from these operators, there are a few operators that do not fit in any of the above categories. These are:

OPERATOR	DESCRIPTION	EXAMPLE
sizeof	sizeof operator is a unary operator used to compute the size of its operand or variable in bytes.	sizeof (char); sizeof(var_name);
Addressof	It is used to find the memory address in which a particular variable is stored. In C++, it is also used to create a reference.	&var_name;
, (Comma operator)	It is a binary operator that is used for multiple purposes. It is used as a separator or used to evaluate its first operand and discards the result; it then evaluates the second operand and returns this value (and type).	int n = (m+1, m-2, m+5); int a, b, c;
. (dot) and -> (arrow)	Member operators are used to reference individual members of classes, structures, and unions.	
Cast	Casting operators convert one data type to another.	int(2.2000) would return 2.

& (Pointer)	It returns the address of a variable. Look at the example it will give actual address of the variable.	&a
*(Pointer)	It is pointer to a variable. Look at the example it will pointer to a variable var.	*var

- **Scope Resolution Operator (::)** is used to access global variables, class members, namespaces, and function definitions that might otherwise be hidden due to local variables or inheritance.

USE CASE	EXAMPLE
Access global variables when local variable has the same name	::x
Define class member functions outside the class	ClassName::FunctionName()
Access static class members	ClassName::StaticVariable
Access namespace members	NamespaceName::VariableName
Call base class method in derived class (method overriding)	BaseClass::Function()
Resolve ambiguity in multiple inheritance	Base1::Function(), Base2::Function()

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

➢ **Constants:-** Constants are used to define values that remain fixed and cannot be changed during the execution of a program.

~ In C++, they are typically declared using the const keyword. When you declare a constant, you specify its data type and give it a name.

~ Constants are often used to make code more readable, and self-documenting, and to prevent accidental changes to important values.

~ You can define constants of various data types, including integers, floating-point numbers, characters, and more.

~ Constants are evaluated at compile-time, and their values are replaced directly in the code.

~ In the below code, maxCount, pi, and newline are constants, and their values cannot be modified after declaration.

```
const int maxCount = 100;
const double pi = 3.14159;
const char newline = '\n';
```

- Using const Keyword:- The const keyword is used to define constant variables.
- Using #define Preprocessor Directive:- The #define directive is used to define constants before compilation.
- Using enum (Enumerations):- Enums allow defining a set of integer constants.

➢ **Literals:-** Literals are the actual values that are directly written into your code to represent specific data. They are used to provide initial values for variables, as operands in expressions, or as direct values in statements.

C++ supports different types of literals:

- Integer Literals: These represent whole numbers, and they can be written in decimal, octal, or hexadecimal formats.
  int decimal = 42; // Decimal integer literal
  int octal = 052; // Octal integer literal (0 prefix)
  int hex = 0x2A; // Hexadecimal integer literal (0x prefix)
- Floating-Point Literals: These represent real numbers and can be written in decimal or exponential notation.
  double decimalNum = 3.14159; // Decimal floating-point literal
  double exponentNum = 6.02e23; // Exponential floating-point literal
- Character Literals: These represent single characters and are enclosed in single quotes.
  char ch = 'A'; // Character literal 'A'
  char newline = '\n'; // Character literal for newline
- String Literals: These represent sequences of characters and are enclosed in double quotes.
  const char* greeting = "Hello, World!"; // String literal
- Boolean Literals: C++ has two boolean literals, true and false, which represent the Boolean values.
  bool isTrue = true;
  bool isFalse = false;

11. Understand the difference between variables and constants.

VARIABLES	CONSTANTS
It is a named storage location in memory that can hold a value, which can be changed during program execution.	It is a named storage location that holds a fixed value, which cannot be changed after initialization.
Declared using a data type (e.g., int, float, char).	Declared using const, #define, or enum.
The value can be updated anytime.	The value remains fixed throughout execution.
int age = 20; age = 25;	const int age = 20; (Cannot be changed)
Used when values need to change (e.g., loop counters, user inputs).	Used for fixed values (e.g., Pi, gravity, tax rate).
Storing user input, Loop counters	Pi value (3.14159), Configuration values (e.g., tax rate, max size)

Control Flow Statements:-

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

➢ Conditional statements in C++ allow a program to make decisions based on conditions. These statements execute different code blocks depending on whether a specified condition is true or false.

- if Statement:- The if statement executes a block of code only if the condition is true**.** (Checking a single condition)

~ Syntax : if(condition)
{ block of code }

- if-else Statement:- The if-else statement provides an alternative block to execute when the condition is false. (Executing code for true/false condition)

~ Syntax : if(condition)
{ block of code }
Else
{ block of code }

- if-else-if Ladder:- The if-else-if ladder allows checking multiple conditions. (Checking multiple conditions sequentially)

~ Syntax:- if (condition1)
{ // Executes if condition1 is true }
else if (condition2)
{ // Executes if condition2 is true }
else
{ // Executes if no conditions are true }

- Nested if Statement:- An if statement inside another if statement. (Checking multiple conditions inside another condition)

~ Syntax:- if (condition1)
{
if (condition2)
{ // Executes if both conditions are true }
}

- Switch Statement:- The switch statement is used when there are multiple cases to handle a single variable. (Selecting from multiple fixed choices (menu, options, etc.))

~ Syntax:- switch (expression)
{ case value1: // Code block
break;
case value2: // Code block
break;
default: // Default block (if no cases match)
}

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

➢ for Loop:- A for loop is a repetition control structure that allows us to write a loop that is executed a specific number of times. It is an entry-controlled loop that enables us to perform n number of steps together in one line.

~ Syntax:- for (initialization; condition; updation)
{ // body of for loop }

- while Loop:- While studying for loop, we have seen that the number of iterations is known beforehand, i.e. the number of times the loop body is needed to be executed is known to us. while loop is used in situations where we do not know the exact number of iterations of the loop beforehand. It is an entry-controlled loop whose execution is terminated on the basis of the test conditions.

~ Syntax:- while (condition)
{ // Body of the loop
update expression }

~ Only the condition is the part of while loop syntax, we have to do the initialization and updation manually outside and inside the loop respectively.

- do while Loop:- The do-while loop is also a loop whose execution is terminated on the basis of test conditions. The main difference between a do-while loop and the while loop is in the do-while loop the condition is tested at the end of the loop body, i.e. do-while loop is exit controlled whereas the other two loops are entry-controlled loops. So, in a do-while loop, the loop body will *execute at least once* irrespective of the test condition.

~ Syntax:- do
{ // Body of the loop
// Update expression }
while (condition);

~ Like while loop, only the condition is the part of do while loop syntax, we have to do the initialization and updation manually outside and inside the loop respectively.

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

➢ In C++, break and continue are control flow statements used in loops (for, while, and do-while) to modify the normal execution of the loop.

- break Statement:- The break statement is used to immediately exit the loop, regardless of the loop condition. The break statement can also be used to jump out of a loop.

~ This example jumps out of the loop when i is equal to 4:

```
for (int i = 0; i < 10; i++) {
 if (i == 4) {
  break;  // Exit the loop when i equals 4
 }
 cout << i << "\n";
}
```

output : 0 1 2 3

- Continue:- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

~ This example skips the value of 4:

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;       // Skip printing 4
  }
  cout << i << "\n";
}
```

Output: 0 1 2 3 5 6 7 8 9

15.Explain nested control structures with an example.

➢ Nested control structures refer to the usage of control structures (such as an if, for, while, or switch) within each other. This allows us to handle more complex logic in a structured way. This allows us to create complex decision-making processes and iterate over multiple levels of data. By nesting control structures, we can perform different actions based on multiple conditions. This provides us with more flexibility and allows us to create powerful algorithms.

- Example:-

```
#include <iostream>
using namespace std;
int main() {
   int age = 25;
   bool isStudent = true;
   if (age >= 18) {
      cout << "You are an adult." << endl;
      if (isStudent) {
         cout << "You are a student." << endl;
      } else {
         cout << "You are not a student." << endl;
      }
   } else {
      cout << "You are not an adult." << endl;
   }
}
```

Output:- You are an adult.
You are a student.

- Explanation:

~ The outer if checks whether age >= 18. Since age is 25, it prints "You are an adult."

~ Inside the outer if, there is another if statement:

- If isStudent is true, it prints "You are a student."
- Otherwise, it prints "You are not a student."

~ Since isStudent = true, the second message is "You are a student."

Functions and Scope:-

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

➢ A **function** in C++ is a block of code designed to perform a specific task. It helps in code reusability, modularity, and maintainability. A function is a set of statements that take inputs, perform a specific computation, and produce an output. Functions help in modular programming by dividing a program into smaller, reusable blocks.

~ Examples of Functions:
main(): The entry point of a C++ program.
sum(): A function that calculates the sum of two numbers.
swap(): A function that swaps two values.

➢ Parts of Functions:-

- Function Definition:- A function definition specifies the actual implementation of a function.
  Syntax:
  return_type function_name(parameter list)
  { // body of the function }

~ Components:-
~ Return_type : It is data type of value which function will return. If function does not return any value data type will be void().
~ Function_name : It is the name given to the function by the programmer.
~ Parameter list : This list refers to type, order and number of parameters of the function. A function can have no parameters also.
~ Body of function : It is collection of statements that define the working of the function.

- Function Declaration:- A function declaration tells the compiler about the function name, return type, and parameters. This must be done before calling the function if its definition appears after main().
  Syntax: return_type function_name (parameter_list);

~ Only the data types of parameters are required in the declaration. The parameter names can be omitted. Ex: int add(int, int); // Function declaration

- Function Call:- To use a function, we have to call that function to perform the given task.

~ When a program calls a function, the compiler gets redirected towards the function definition. Function Call simply pass the required parameters along with the function name.
~ To use a function, we need to call it inside main() or another function. When a program calls a function, the compiler redirects execution to the function definition.
Syntax: function_name(arguments);

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

- In C++, the **scope** of a variable is the extent in the code upto which the variable can be accessed or worked with. It is the region of the program where the variable is accessible using the name it was declared with.
- **Scope** is the part of the program where a defined variable can have its existence and beyond that it cannot exist.

LOCAL VARIABLES	GLOBAL VARIABLES
Limited to the block of code	Accessible throughout the program
Typically within functions or specific blocks	Outside of any function or block
Accessible only within the block where they are declared	Accessible from any part of the program
Created when the block is entered and destroyed when it exits	Retain their value throughout the lifetime of the program
Can have the same name as variables in other blocks	Should be used carefully to avoid unintended side effects
Temporary storage, specific to a block of code	Values that need to be accessed and modified by multiple parts of the program

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

- It is the process by which a function calls itself repeatedly, until some specific condition has been satisfied.
- This process is used in place of repetitive computations in which each action is stated in terms of a previous result
- In C++, recursion is implemented by defining a function that calls itself with modified parameters and includes a base condition to terminate the recursion
- Syntax : data_type function_name (argument list)
  { function_name(. . .) ; }
- Example of Factorial Number:

```
#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) << endl;
}
```
Output: Enter a number: 5
Factorial of 5 is: 120

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

- The function prototype is the template of the function which tells the details of the function e.g(name, parameters) to the compiler. Function prototypes help us to define a function after the function call.
- A function prototype is a declaration of a function that tells the compiler: Function name, Return type, Parameter types and order
- It allows us to define the function after calling it in main() or another function.
- Syntax: return_type function_name(parameter_list);
- Example of Function Prototype

```
#include <iostream>
using namespace std;
// Function Prototype
int sum(int, int);
int main() {
int result = sum(10, 5); // Function Call
cout << "Sum: " << result << endl;
return 0;
}
// Function Definition
int sum(int a, int b) {
return a + b;
}
```
Output: Sum: 15

- Function Prototypes Used:-
- Allows Function Definition After main()- The function can be written later in the program.
- Improves Code Readability & Organization- Helps in modular programming by separating declaration and definition.
- Prevents Compilation Errors- Compiler knows the function signature before calling it.
- Mandatory in Some Cases- When defining functions after main() or in a different file.

Arrays and Strings:-

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

- An **array** in C++ is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow us to store multiple values under one variable name.
- ~ Syntax of an Array:- data_type array_name[size];
- ~ Example: int arr[5];

SINGLE-DIMENSIONAL	MULTI- DIMENSIONAL
Store a single list of the element of a similar data type.	Store a 'list of lists' of the element of a similar data type.
Represent multiple data items as a list.	Represent multiple data items as a table consisting of rows and columns.
The declaration For C++, datatype variable_name[row]	The declaration For C++, datatype variable_name[row][column]
One Dimension	Two Dimension
size of(datatype of the variable of the array) * size of the array	size of(datatype of the variable of the array)* the number of rows* the number of columns.
Address of a[index] is equal to (base Address+ Size of each element of array * index).	Address of a[i][j] can be calculated in two ways row-major and column-major **Column Major:** Base Address + Size of each element (number of rows(j-lower bound of the column)+(i-lower bound of the rows)) **Row Major:** Base Address + Size of each element (number of columns(i-lower bound of the row)+(j-lower bound of the column))

int arr[5]; //an array with one row and five columns will be created.
{a , b , c , d , e}
	int arr[2][5]; //an array with two rows and five columns will be created.
	a b c d e
	f g h i j

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

- An **array** is a collection of elements of the same data type stored in contiguous memory locations. It allows storing multiple values under a single variable name.
- ~ To **initialize** an array in C++, we can use the assignment operator = to assign values to the array at the time of declaration. The values are provided in a comma-separated list enclosed in curly braces {}.
- ~ Syntax:- We can use the below syntax to initialize an array at the time of declaration.
  datatype arrayName[arraySize] = {element, element2, ..., elementN};
- 1D Arrays (One-Dimensional Arrays):- A 1D array is a list of elements stored in a single row.

~ Syntax: data_type array_name[size] = {value1, value2, ..., valueN};

~ Example:

```
#include <iostream>
using namespace std;
int main() {
int arr[5] = {10, 20, 30, 40, 50}; // Initializing 1D array
cout << "1D Array Elements: ";
for (int i = 0; i < 5; i++) {
cout << arr[i] << " ";  // Printing array elements
}
return 0;
}
```

Output:- 1D Array Elements: 10 20 30 40 50

- 2D Arrays (Two-Dimensional Arrays):- A 2D array is a table-like structure with rows and columns.

~ Syntax: data_type array_name[rows][columns] = {

```
   {row1_values},
   {row2_values}, ...
};
```

~ Example:-

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

Output:- 2D Array Elements:

1 2 3

4 5 6

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

➢ C++ strings are sequences of characters stored in a char array. Strings are used to store words and text. They are also used to store data, such as numbers and other types of information. Strings in C++ can be defined either using the std::string class or the C-style character arrays.

- **C Style Strings:-** These strings are stored as the plain old array of characters terminated by a null character '\0'. They are the type of strings that C++ inherited from C language. They require functions from the <cstring> library for manipulation.

~ Syntax: char str[] = "GeeksforGeeks";

~ Common C-Style String Functions (#include <cstring>)

FUNCTION	DESCRIPTION
strlen(str)	Returns length of string
strcpy(dest, src)	Copies src to dest
strcat(dest, src)	Appends src to dest
strcmp(str1, str2)	Compares two strings

~ Example:-

```
#include <iostream>
using namespace std;
int main()
{
char s[] = "GeeksforGeeks";
cout << s << endl;
}
```

Output:- GeeksforGeeks

- **std::string Class:-** std::string provides a more flexible and safer way to handle strings. These are the new types of strings that are introduced in C++ as std::string class defined inside <string> header file. This provides many advantages over conventional C-style strings such as dynamic size, member functions, etc. No need for \0 termination. Supports built-in operators (+, ==, etc.).

~ Syntax: std::string str("GeeksforGeeks");

~ Common std::string Methods (#include <string>)

FUNCTION	DESCRIPTION
length()	Returns string length
append(str)	Appends another string
substr(start, len)	Extracts substring
find(str)	Finds the first occurrence of a substring
erase(start, len)	Removes characters from a string
insert(pos, str)	Inserts a substring

~ Example:-

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

```
int main()
{
string str("GeeksforGeeks");
cout << str;
}
```

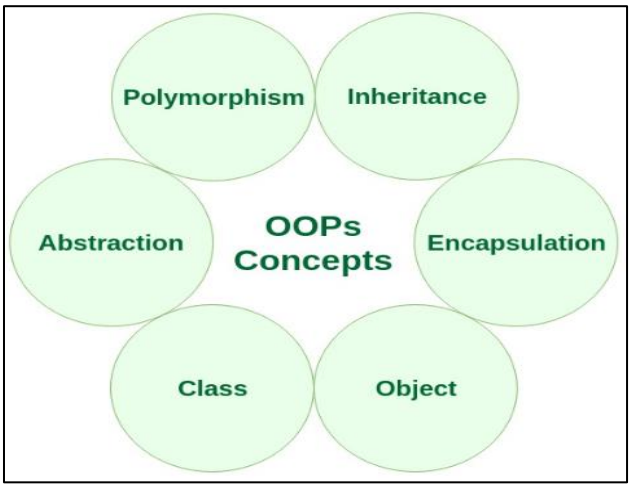
23.Explain string operations and functions in C++. (Answer:- Q-22)

- String Operations in C++

CATEGORY	FUNCTION/OPERATOR	FUNCTIONALITY	EXAMPLE
String Length	length() or size()	Returns the length of the string.	str.length();
Accessing Characters	Indexing (str[index])	Access individual characters using indexing.	str[0];
	at(index)	Access a character at a specified index.	str.at(2);
Appending & Concatenation	+ Operator	Used to concatenate two strings.	str1 + str2;
	append()	Adds one string to the end of another.	str1.append(str2);
String Comparison	== Operator	Compares two strings.	if (str1 == str2) {...}
	compare()	Returns an integer indicating comparison result.	str1.compare(str2);
Substrings	substr(pos, len)	Extracts a substring from a string.	str.substr(1, 3);
Searching	find("word")	Finds the position of a substring.	str.find("C++");
Modifying Strings	replace(pos, len, "new")	Modifies part of a string.	str.replace(0, 5, "Hi");
	insert(pos, "text")	Inserts a substring at a specified position.	str.insert(5, "Code");
	erase(pos, len)	Removes part of a string.	str.erase(0, 3);
Conversion	c_str()	Converts std::string to a C-style string.	const char* c = str.c_str();

Introduction to Object-Oriented Programming:-

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



- **Object-oriented programming** – As the name suggests uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.
- There are some basic concepts that act as the building blocks of OOPs i.e. Fig→

- **Class:-** It is specified as a blueprint that describe the behaviors/states for an object. Its is user defined datatype. Class consist of data members and members function, which can be accessed depending on the access specifier. The object of the class hold separate copies of data members. Its is possible to create as many objects of a class as required.
- Syntax: class class_Name{
  access modifier:
  data member;
  access modifier:
  member function(){
  body of function.
  }
  };
- **Object:-** An Object can be defined as an entity that has a state and behavior, or in other words, anything that exists physically in the world is called an object. It can represent a dog, a person, a table, etc. An object means a combination of data and programs, which further represent an entity.
- Syntax: class_Name object _name;
  in this we can protect our data by using access specifier/modifier.
- They are Three types:-
  - public: you can used public data any where in the program.
  - private: you can't the data directly in other class or function for that you need get and set function to use it
  - protected :it can be used in the child class only.
- **Encapsulation:-** It is the process of binding together the data member and member functions that manipulate the data, thereby keeping the members safe from outside misuse. It is the process of hiding the data and functions into a single unit and protecting it from the outside world. It provides the concepts of data hiding.

~ Features:- Whenever the implementations of the class changes, the interface remains the same.
- It reduces human errors.
- It provides the security to data.

~ The main benefit of using Encapsulation is to keep data secure from other methods. if we make the data private, then the data can be used within a class ,but not accessible outside the class.

- **Inheritance:-** It is used to define a new class from an existing class thereby making it easier to create and maintain a software application. It provides an opportunity to reuse the code functionality (ie. a way to reuse once written code again and again) and help in reducing the code size and ensuring faster implementation.

~ Types of Inheritance:-
- Single Inheritance.
- Multiple Inheritance.
- Multilevel Inheritance.
- Hierarchical Inheritance.
- Hybrid Inheritance.(virtual Inheritance).
- → virtual base class and diamond problem.

- **Polymorphism:-** It is the combination of the words poly which means many and morph which means forms .in short many forms. The ability to take more then one form is referred to as Polymorphism. It is the capability of using an operator or a function in different ways. A single function or an operator can function in many ways on different instances depending upon the usage.

~ The operator which can't be overloaded:
1: typeid,
2: ::(scope resolution operator),
3: . (dot) ,*
4: sizeof() ,
5: ?:

~ Function overloading: Function names are same but parameter are different, or datatype can be different

~ Operator overloading: It gives special meaning to an existing operator without changing its original meaning. +,++,--,<,>,

~ syntax:- return_type operator (operator symbol)(classname & objectname)

- **Abstraction:-** It refers to representing the required information to the outside world without presenting the background details, i e.: hiding the details of an object and showing only the essential information that a user can understand.
  Data abstraction is a programming technique that relies on the separation of interface and implementation.

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

- **Class:-** A class is a blueprint or template for creating objects. It defines data members (variables) and member functions (methods) that operate on the data. A class is a user-defined data type that encapsulates data and behavior.

~ It is specified as a blueprint that describe the behaviors/states for an object. It's is user defined datatype. Class consist of data members and members function, which can be accessed depending on the access specifier. The object of the class hold separate copies of data members. It's is possible to create as many objects of a class as required.

~ Syntax: class ClassName {

```
access_specifier:
    // Data members (variables)
    // Member functions (methods)
};
```

- **Object:-** An object is an instance of a class. It represents a real-world entity with state (data members) and behavior (functions/methods). Each object has its own copy of the data members of the class.

~ An Object can be defined as an entity that has a state and behavior, or in other words, anything that exists physically in the world is called an object. It can represent a dog, a person, a table, etc. An object means a combination of data and programs, which further represent an entity.

~ Syntax: class_Name object _name; // Object creation
in this we can protect our data by using access specifier/modifier.

~ They are Three types:-

o public: you can used public data anywhere in the program.

o private: you can't the data directly in other class or function for that you need get and set function to use it

o protected :it can be used in the child class only.

- <u>Example of class and object:</u>

```
#include <iostream>
using namespace std;
class Car {
private: string brand; // Private data member
public:
void setBrand(string b) { // Public function to set brand
brand = b;
}
void display() { // Public function to display brand
cout << "Car brand: " << brand << endl;
} };
int main() {
```

```
Car myCar; // Creating an object of the Car class
myCar.setBrand("Toyota");
myCar.display();
return 0;
}
```
Output:- Car brand: Toyota

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

- It is used to define a new class from an existing class thereby making it easier to create and maintain a software application. It provides an opportunity to reuse the code functionality (ie. a way to reuse once written code again and again) and help in reducing the code size and ensuring faster implementation.

- <u>Advantages of Inheritance:</u>

~ Code Reusability – Avoids code duplication by reusing existing code.
~ Maintainability – Modifications in the base class automatically reflect in the derived class.
~ Faster Development – Helps in rapid application development.

- <u>Types of Inheritance:-</u>

~ Single Inheritance: One class inherits from another.
~ Multiple Inheritance: A class inherits from multiple base classes.
~ Multilevel Inheritance: A class inherits from another derived class.
~ Hierarchical Inheritance: Multiple classes inherit from a single base class.
~ Hybrid Inheritance.(virtual Inheritance): A combination of multiple inheritance types.
~ virtual base class and diamond problem.

- <u>Example of single Inheritance:-</u>

```
#include <iostream>
using namespace std;
// Base class (Parent)
class Animal {
public:
void eat() {
cout << "This animal eats food." << endl;
}};
// Derived class (Child)
class Dog : public Animal {
public:
void bark() {
cout << "The dog barks." << endl;
} };
int main() {
Dog myDog;
```

```
myDog.eat();  // Inherited function from Animal
myDog.bark(); // Function of Dog class
return 0;
}
```

Output:- This animal eats food.
The dog barks.

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

- Encapsulation is an Object-Oriented Programming (OOP) concept that binds data (variables) and functions (methods) together into a single unit (class) and prevents direct access to the data from outside the class.
- ~ It is the process of binding together the data member and member functions that manipulate the data, thereby keeping the members safe from outside misuse.
- ~ It is the process of hiding the data and functions into a single unit and protecting it from the outside world.
- ~ It provides the concepts of data hiding.
- ~ The main benefit of using Encapsulation is to keep data secure from other methods. if we make the data private, then the data can be used within a class ,but not accessible outside the class.

- Features:

- ~ Data Hiding – Prevents direct access to class members from outside.
- ~ Increases Security – Protects data from unintended modifications.
- ~ Reduces Complexity – Implementation details are hidden from the user.
- ~ Improves Maintainability – Changes to internal implementation do not affect the external code.

- **How is Encapsulation Achieved in C++?**

- ~ Encapsulation is implemented using Access Specifiers**:**
  private – Members can be accessed only within the class.
  protected – Members can be accessed within the class and derived classes.
  public – Members can be accessed from anywhere.