Module #3 Theory

1. Introduction to C++ :

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

Ans:

Procedural Programming: Procedural programming executes instructions in a step-by-step sequence in which they are written. Large programs are divided into functions, and the program works on data passed through arguments to them. Code reusability is limited to functions only. The data provided to functions are outside of them, so it can be easily manipulated and lacks data security.

Object-Oriented Programming (OOP): OOP uses classes and objects to bundle data and functions together. It keeps data separate and secure from the rest of the program to prevent accidental modification and provides controlled access (public, protected, private). Data can only be accessed through objects, which are instances of classes. In OOP, code reusability is achieved easily through inheritance and polymorphism. Objects can communicate with each other, and the functions of one object can be accessed by another object's function. It can be easily scaled, and new functions and data can be added anytime.

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

Ans:
Code organization: Code is organized into classes and objects.
Data encapsulation: Objects keep data inside, making it safe from accidental changes.
Controlled access: Data can only be accessed through member functions and access specifiers (private, protected, public).
Code reusability: Code can be easily reused with polymorphism and inheritance.
Real-world modeling: Objects and classes allow using real-world entities in programming (e.g., dog, car, human).
Dynamic behavior: Objects and functions can behave differently based on the situation (e.g., **function overriding**, polymorphism, inheritance).

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

Ans:
Install a compiler: A compiler translates C++ code into machine code. Examples include GCC (Linux, macOS), MinGW (Windows), or MSYS2. Compilers can be downloaded as executables or installed via package managers.

Install an IDE or text editor: A text editor or IDE is used to write code. Examples include Visual Studio Code (text editor), Dev-C++, or CLion (IDE).

Configure environment variables: Ensure that the compiler and editor paths are set in environment variables.

Verify and test execution: Write the following code in a `.cpp` file and compile it through the terminal.

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

Compile with `gcc filename.cpp`, run the executable, and check the output in the terminal.

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

Ans:
Main input/output operations in C++:

- **`std::cin` – Used for user input.**
- **`std::cout` – Used to print values on the console.**

Example:

```
#include <iostream>
using namespace std;
int main() {
   int a;

   cout << "Hello";
   cin >> a; // If the user inputs 10, it will be stored in variable 'a'.
   cout << a; // Prints 10 on the console.

   return 0;
}
```

2.Variables, Data Types, and Operators:

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

Ans:
C++ has two main types of data types:

1. Primitive: These are fundamental data types available in C++.

Int: Used to store whole numbers.
Example: `int height = 10;`

Float/Double: Used to store numbers with fractional parts.
Example: `double gpa = 3.3;`
Example: `float percentage = 93.5;`

Char: Used to store a single character value.
Example: `char ch = 'A';`

Bool: Used to store logical values (true or false).
Example: `bool isStudent = true;`
Example: `bool isAdult = false;`

2. Non-Primitive: These include data types derived from primitive types.

Array: Stores multiple values of the same data type in a continuous sequence.
Example: `int roll[4] = {101, 102, 103, 104};`

Functions: Can take multiple data type values as parameters and return a specific data type value.
Example:
```
 int add(int a, int b) {
   return a + b;
}
```

Pointers: Stores the memory address of a variable.
Example:
```
 int a = 10;
int *ptr = &a;
```

Structures: Stores multiple values of multiple data types.
Unions: Similar to structures but can store a single value across all members at a time.

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

Ans:

Implicit Conversion: Automatically performed by the compiler when converting one data type to another during operations.

Explicit Conversion: Performed manually by the user by specifying the target type.
Example:

```
 int a = 10;
float b = (float)a; // 'a' becomes a fractional value
```

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

Ans:

Arithmetic Operators:

Addition (+)	: `int c = a + b;`
Subtraction (−)	: `int c = a - b;`
Multiplication (*)	: `int c = a * b;`
Division (/)	: `int c = a / b;`
Modulus (%)	: `int c = a % b;`
Increment (++)	: `a++;`
Decrement (−−)	: `b--;`

Relational Operators:

Equality (==)	: `a == b`
Less than (<)	: `a < b`
Greater than (>)	: `a > b`
Not equal (!=)	: `a != b`
Less than or equal to (<=)	: `a <= b`
Greater than or equal to (>=)	: `a >= b`

Logical Operator

Logical AND (&&)	: `a && b`
Logical OR (||)	: `a || b`
Logical NOT (!)	: `!a`

Bitwise Operators: Operate on binary representations.

AND (&), OR (|), NOT (~), XOR (^), Right Shift (>>), Left Shift (<<)

Ternary Operator:

Syntax: `condition ? expr1 : expr2;`
Example: `string c = (a < b) ? "true" : "false";`

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

Ans:

Constants: Fixed values stored in variables that cannot be changed throughout the program.

Example: `const int pi = 3.14;`

Literals: Fixed constant values written directly in code without any identifier.

Integer literals: `12`, `0x43`, `043`
Float literals: `34.34`
String literals: `"kapil"`
Char literals: `'S'`

Literals are tokens in C++ and are part of the language to display values.

4.Control Flow Statements:

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

Ans:

Conditional Statements: These control the flow of a C++ program based on specific conditions evaluated as true or false. A specific block of code executes if the condition is true; otherwise, another block executes.

1. If-Else: The if-else statement can be simple, ladder, or nested.
Simple If-Else:

```
if (true) {
   cout << "Block of code when true";
} else {
   cout << "Block when false";
}
```

Ladder If-Else:

```
if (true) {
   cout << "True at first";
} else if (true) {
   cout << "True at second";
} else {
   cout << "All conditions false";
}
```

Nested If-Else:

```
if (true) {
   if (true) {
      cout << "True at both levels";
   }
} else {
   cout << "All conditions false";
}
```

2. Switch Statement: The switch statement compares a condition with multiple cases and executes the matching block.

```
switch (value) {
   case 1:
      cout << "Case 1";
      break;
   case 2:
      cout << "Case 2";
      break;
   default:
      cout << "No condition matched";
      break;
}
```

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

Ans:

For Loop: Used when the number of iterations is known beforehand.

```
for (int i = 0; i < 10; i++) {
   cout << "Iteration " << i << endl;
}
```

While Loop: Executes while the condition is true. Used when the number of iterations is not known.

```
int i = 0;
while (i < 10) {
   cout << "Iteration " << i << endl;
   i++;
}
```

Do-While Loop: Executes at least once, as the condition is checked after execution.

```
int i = 0;
do {
   cout << "Iteration " << i << endl;
   i++;
} while (i < 10);
```

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

Ans:

Break Statement: Exits the loop immediately when encountered.

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break; // Exit loop
    }
    cout << i << endl;
}
```

Continue Statement: Skips the current iteration and proceeds to the next.

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        continue; // Skip iteration
    }
    cout << i << endl;
}
```

4. Explain nested control structures with an example.

Ans:

Nested control structures refer to placing a loop or conditional statement inside another loop or conditional statement. For example, an `if` condition inside a `for` loop or a `for` loop inside another `for` loop (nested loop).

Example :

```
for (int i = 1; i <= 3; i++) { // Outer loop
    cout << "Outer loop: " << i << endl;
    for (int j = 1; j <= 2; j++) { // Inner loop
        cout << "  Inner loop: " << j << endl;
    }
}
```

4.Functions and Scope

1. 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 when called. It executes the statements written within it. Functions are used to divide a program into smaller, manageable code blocks and organize the code.

C++ has four types of functions:

- With return type and with arguments
- With return type and without arguments
- Without return type and with arguments
- Without return type and without arguments

A function involves three steps:

1. **Declaration:** Determines the data type, function name, and arguments**.**
2. **Calling:** Executes the function in the main program.
3. **Definition:** Contains the code to be executed when the function is called**.**

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

Ans:
The scope of a variable in C++ refers to the part of the program where the variable is accessible.
Local Scope: A variable declared inside a function or block of code has local scope and is accessible only within that block. Example:

```
{
   int a = 44; // Local scope
}
```

Global Scope: A variable declared outside any function or class has global scope and is accessible throughout the program. Example:

```
int a = 55; // Global scope
{
   cout << a; // Access global variable
   a = 66;
   cout << a;
}
```

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

Ans:
Recursion is a programming concept where a function calls itself to solve smaller instances of the same problem. It is typically used to break down complex problems into simpler sub-problems. A recursive function has a base case to stop the recursion and a recursive case where the function calls itself.

Example: Finding the sum of array elements.
```
int sum(int arr[], int size) {
   // Base case
   if (size == 0) {
      return 0;
   }
   // Recursive case
   return arr[size - 1] + sum(arr, size - 1);
}
```

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

Ans:
A function prototype is a declaration of a function that specifies the function name, return type, and parameters without providing its implementation. It acts as a blueprint or contract, informing the compiler about the function's existence and how it should be called.

5.Arrays and Strings

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

Ans:
An array is a collection of similar data type values stored in a contiguous sequence.
Single-dimensional array: A linear collection of elements of the same type.
Example:
```
int a[5] = {1, 2, 3, 4, 5};
```

Multi-dimensional array: An array where each element itself is an array, allowing for multiple dimensions.
Example:
```
int a[3][2] = {
   {1, 2},
   {3, 4},
   {5, 6}
}
```

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

Ans:
The C++ standard library provides powerful functionality for handling strings, offering features such as dynamic sizing, easy concatenation, and various string methods.
To declare a string:
```
string name = "Kapil"; // 'name' holds "Kapil" as a value.
```

Examples of string functions:
- `str.length()` - Finds the length of a string.
- `str.substr()` - Extracts a substring.
- `str.find()` - Finds a substring.
- `str.replace()` - Replaces part of the string.

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

Ans:
Arrays in C++ are initialized sequentially, starting with index 0.
1D Array: A linear collection of elements of the same type. Example:
```
int arr[3] = {1, 2, 3};
```

2D Array: A matrix-like collection of elements arranged in rows and columns. Example:
```
int arr[2][2] = {
  {1, 4},
  {5, 6}
};
```

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

Ans:
In C++, a string is a sequence of characters stored in a `char` array or using the `string` class.
Examples:
```
string str = "Text";
char str2[] = {'T', 'e', 'x', 't', '\0'};
```

Common string functions:
`strlen(str1)` - Finds the length of a string.
`strcpy(str1, str2)` - Copies the content of one string to another.
`strcat(str1, str2)` - Appends two strings.
`strcmp(str1, str2)` - Compares two strings.

6.Introduction to Object-Oriented Programming

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

Ans:
Four key concepts of object-oriented programming:

1. **Encapsulation**: Encapsulates important information within an object, exposing only selected details. Protects data from external interference.
2. **Abstraction:** Hides unnecessary implementation details and reveals only relevant internal mechanisms.
3. **Inheritance:** Allows classes to reuse code and properties from other classes.
4. **Polymorphism:** Enables objects to share behaviors and take on more than one form.

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

Ans:
Classes and objects are fundamental concepts in object-oriented programming.
Class: A user-defined data type that serves as a blueprint for creating objects.
Example:

```
class Student {
  /* code */
};
```

Object: An instance of a class containing data and methods. Example:

```
Student s1;
```

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

Ans:
Inheritance in C++ allows a class to inherit properties and methods from another class.

- **Base class:** The class being inherited from.
- **Derived class:** The class inheriting from another class.

Example:

```
class Base {
public:
  void display() {
    cout << "Base class display" << endl;
  }
};
```

```
class Derived : public Base {
   // Inherits Base class
};
```

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

Ans:
Encapsulation bundles data and methods into a class while restricting access using access specifiers:

- **Private:** Accessible only within the class.
- **Public:** Accessible from outside the class.

Example:
```
class Employee {
private:
   int age; // Hidden
public:
   void setAge(int a) { age = a; } // Access via methods
   int getAge() { return age; }
};
```