🚀 3-Month Intensive Plan for an Aspiring Software Engineer

Mastering C++ Foundations (Start from Scratch, No OOP Yet)

- ❖ features of C++
- ❖ Structure of a C++ program
- ❖ Compilation and Execution process

Setup: Install & configure modern IDEs (VS Code + C++ extensions, or CLion)

Core Language Skills:

- Variables, Data Types, Type modifiers (signed, unsigned, long, short)
- Constants, and Literals
- Input/Output streams (`cin`, `cout`) with Formatting output (endl, \n, setw, setprecision from <iomanip>)
- Operators (arithmetic, logical, bitwise) & Expressions
- Conditional (Ternary) operator
- Operator precedence and associativity

□ Flow Control:

□ **Decision Making:**

if, if-else, else-if ladder

switch statement

□ **Loops:**

while, do-while, for loop

- Nested loops
- Range based for loop
- Loop control statements (break, continue, goto)

Strings in C++

- C-style strings vs string class (c vs cpp string)
- String operations using string class:

- **Functions:**

- Function definition, declaration, and call
- Function parameters and return values
- Call by value vs Call by reference
- Default arguments
- Function overloading (Intro)
- Recursion
- Inline functions

□ Data Collections:

Arrays

- One-dimensional arrays
- Declaration, Initialization, Access
- Traversal, Search, Sort
- Two-dimensional arrays (Matrix)
- Input/output
- Memory representation and accessing of simple and 2D array.
- Matrix operations (addition, transpose, etc.)
- Array of characters (C-style strings)
- Common string functions (strlen, strcpy, strcat, strcmp)
- Arrays In functions passing as arguments.
- Strings in function as arguments and return array and strings from a function as a return type.

Pointers & Memory:

- Introduction to pointers
  - Pointer variables and addresses
  - Pointer arithmetic
  - Pointers and arrays (basic relationship)
  - Pointers and functions (pass by address)
  - NULL and nullptr
- Types of pointers in cpp

 Dynamic memory allocation: new, delete

User-Defined Data Types

- typedef
- enum
- struct (structure)
  - Declaring and using structures
  - Nested structures
  - Array of structures
- Macros and Preprocessor Directives

Namespaces

- What is a namespace
- Using the `std` namespace

Type Conversion & typeCasting

. File Handling (Basics)

- Streams (ifstream, ofstream, fstream)
- Opening and closing files
- Reading from and writing to files
- File modes (ios::in, ios::out, ios::app, etc.)

Basic Error Handling

- Syntax errors vs logical errors vs runtime errors

- Very basic intro to try-catch block (without diving into exception classes)

Practice & Projects:

- Build several mini-projects: calculator, array utilities, pattern printing, text-based games (guess the number)
- Daily coding challenges to solidify concepts

C++ OOP Deep Dive

- **STEP 1: Introduction to OOP Concepts**
- Understand the *conceptual foundations* before writing code.
- **Topics:**
  - What is Object-Oriented Programming?
  - Differences: Procedural vs Object-Oriented
  - Principles of OOP:
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism
  - Real-world examples of OOP

- **STEP 2: Classes and Objects**
- Core building blocks of OOP in C++
- **Subtopics:**
  - Defining a class
  - Creating objects
  - Access modifiers: public, private, protected
  - Accessing data members and member functions
  - scope resolution (::) operator
  - Dot operator with objects

- **STEP 3: Constructors and Destructors**
- Automatic object creation and cleanup
- **Subtopics:**
  - Default constructor
  - Parameterized constructor
  - Constructor overloading
  - Copy constructor
  - Constructor vs. function
  - Destructor and its use
  - Constructor/Destructor visibility (public/private)

- **STEP 4: Encapsulation and Abstraction**

- Hide details and protect data
- 🔨 **Subtopics:**
  - Getters and setters (accessor and mutator methods)
  - Making data private
  - Advantages of encapsulation
  - Friend functions and classes (when and why to break encapsulation — carefully)
  - Abstraction using interfaces (via pure virtual functions — briefly introduced)

- 🧱 **STEP 5: Static Members**
- Shared data/functions among all objects of a class
- 🔨 **Subtopics:**
  - Static data members
  - Static member functions
  - Accessing static members
  - Use cases: object count, shared settings
- ---
- 🧱 **STEP 6: Constant Members**
- Immutable functions and objects
- 🔨 **Subtopics:**
  - const data members
  - const member functions
  - const objects
  - Why use const in classes?
  - Rule: const functions can only call other const functions
- ---
- 🧱 **STEP 7: this Pointer**
- Pointer to the current object
- 🔨 **Subtopics:**
  - What is this pointer?
  - Using this-> to access members
  - Use in setters and constructor chaining
- ---
- 🧱 **STEP 8: Dynamic Memory Allocation in Classes**
- Use of new and delete with objects
- 🔨 **Subtopics:**
  - Creating objects with new
  - Array of objects with new[]
  - Destructor with dynamic memory
  - Deep vs shallow copy
  - Writing user-defined copy constructor and assignment operator
- ---
- 🧱 **STEP 9: Arrays of Objects**
- Managing multiple objects using arrays
- 🔨 **Subtopics:**
  - Declaring arrays of class objects
  - Initializing and accessing members

- Using constructors in arrays
- Input/output using loops

- 🧱 **STEP 10: Pointers to Objects**
- Access objects via pointers
- 📌 **Subtopics:**
  - Pointer to object syntax
  - Arrow operator ->
  - Accessing members using pointers
  - Array of pointers to objects

- 🧱 **STEP 11: Inheritance**
- Code reusability and hierarchy
- 📌 **Subtopics:**
  - Base and derived classes
  - Single inheritance
  - Multilevel inheritance
  - Hierarchical inheritance
  - Access specifiers in inheritance (public, private, protected)
  - Constructor and destructor behavior in inheritance
  - Overriding functions

Function Hiding in Inheritance

Happens when base and derived have same-named functions.
Add under Inheritance or Overriding:

- How function hiding differs from overriding

- Object Slicing

  A subtle but critical runtime behavior in inheritance.

  Add under Inheritance or Pointers to Objects:

  What is object slicing?

  How to avoid it (use pointers/references to base)

- 🧱 **STEP 12: Polymorphism**
- Many forms of behavior (compile-time and runtime)
- 📌 **Subtopics:**
- **a. Compile-Time Polymorphism**
  - Function overloading
  - Operator overloading (+, -, ==, etc.)
- **b. Run-Time Polymorphism**
  - Virtual functions
  - Function overriding
  - Base class pointer to derived class object
  - Dynamic binding
  - virtual destructor

. Virtual Table (vtable) Basics

Not for coding, but understanding polymorphism

- ---
- 🧱 **STEP 13: Abstract Classes and Interfaces**
- Provide base structure, enforce behavior
- 📌 **Subtopics:**
  - Pure virtual functions
  - Abstract class definition
  - Using abstract classes as interfaces
  - Cannot instantiate abstract classes
  - Derived classes must override all pure virtual functions
- ---
- 🧱 **STEP 14: Virtual Base Classes and Diamond Problem**
- Fix multiple inheritance issues
- 📌 **Subtopics:**
  - Diamond problem explained
  - Virtual inheritance
  - Resolving ambiguity with virtual keyword
  - Constructor order in multiple inheritance
- ---
- 🧱 **STEP 15: Function Overloading vs Overriding**
- Understand clear differences
- 📌 **Subtopics:**
  - Compile-time vs runtime resolution
  - Rules of overloading (same function name, different parameters)
  - Rules of overriding (virtual base function, same signature)
  - Use of override keyword (from C++11)
- ---

STL:

****Vectors in C++:**

What is a vector?

Vector vs array

Why vectors are useful (dynamic sizing, memory management)

Vector Declaration and Initialization

Different ways to initialize

Vector Input and Output

Vector Member Functions (Operations)

Traversing a Vector

- **Index-based loop**
- **Range-based for loop**
- **Using iterators: begin(), end()**

2D Vectors

** list

** map

** sets

Templates in C++ (basics)

- **Introduction to Templates**
- **What are templates?**
- **Advantages of generic programming**
- **Use in STL (vector, list, map)**

- 🧱 **STEP 16: Real-World OOP Mini Projects**
- Apply everything practically
- 🔨 **Mini Projects Ideas:**
  - Bank account system (classes, inheritance, file handling)
  - Library management (OOP + pointers to objects)
  - Student report system (arrays of objects + file handling)
  - Simple HR or employee system

Sample Daily Time Allocation (Month 3)

Task	Hours/Day
Basics C++ Practice	5-6 hours
Git & GitHub Workflow	20 minutes