📝 Notes | September 3

- Learning **C++ from scratch** with **daily practice and mini projects** is a smart and effective plan. Since you're starting fresh, we'll structure your learning step-by-step with:
-

- ### ☑ Daily Plan Structure:

- **Concept of the Day** – Easy explanation with examples.
- **Mini Task** – Small coding practice to reinforce the concept.
- **Mini Project Idea** – Based on today's concept.
- **Bonus Tip** – Good habits, shortcuts, or extra tricks.
-

- ## 🚀 Day 1: Introduction to C++ & Basic Input/Output

- ### 🧠 Concept of the Day:

- **1. Structure of a C++ program**
- **2. Header files, `main()` function**
- **3. Basic Input (`cin`) and Output (`cout`)**

- ### 🧾 *Syntax Example:*

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

  int main() {
      cout << "Welcome to C++!\n";
      int age;
      cout << "Enter your age: ";
      cin >> age;
      cout << "You are " << age << " years old.\n";
      return 0;
  }
  ```
  ```

-

- ☑️ **Mini Task:**

- Take user input:
- Name
- Age
- Favourite number
- Then output them in one sentence.
- 💡 *Expected Output:*
- `Hello Anindita, you are 21 years old and your favorite number is 7.`
-

- 🛠️ **Mini Project Idea:**

- 📋 **C++ Greeting Card Generator**
- 📝 Description:
- Create a simple greeting message app where the user enters:
- Their name
- Occasion (Birthday, New Year, etc.)
- A short message
- 🎯 Output: Nicely formatted greeting.
- ```
  // Sample output
  --------------------------
  🎉 Happy Birthday, Anindita! 🎉
  Wishing you all the success!
  --------------------------
  ```

-

- 🧽 **Bonus Tip:**

- Always write `using namespace std;` after `#include <iostream>` to avoid prefixing `std::` every time.
-

- **Q. using namespace std; It's important to understand why I write.**
  ```

-

- 🔍 Because:

- All functions, classes, objects, etc. in the standard library of C++ are inside a namespace called std.
- Such as:
- The actual name of cout is: std::cout
- Cin ‘s real name is: std::cin
- string → std::string
-

🔧 if using namespace std; Do not write:

Then you have to write std:: every time.

Example:

- ```
  #include <iostream>

  int main() {
      std::cout << "Hello, World!\n";
      int age;
      std::cout << "Enter your age: ";
      std::cin >> age;
      std::cout << "You are " << age << " years old.";
      return 0;
  }
  ```

-

✨ And if you write:

- `using namespace std;`

- You don't have to write std::. The code is short and easy to read.
- ```
  #include <iostream>
  using namespace std;
  ```

```
int main() {
    cout << "Hello, World!\n";
    int age;
    cout << "Enter your age: ";
    cin >> age;
    cout << "You are " << age << " years old.";
    return 0;
}
```

-
- ⚠ Extra information (required when project):
- Large projects can have many namespaces (custom namespaces too). then using namespace std; Instead of typing, it is better to use std::cout, so that there is no confusion.
-

📌 Conclusion:

- → using Namespace std in small programs; ✅
- → std on large projects: Better ⚠ to use

Yes! If you ask the **user** to input a short message (like: “Stay happy always!”), and then **store it properly**, your program can **use that message** in the final greeting.

☑ Here's how you can handle it:

◇ *Problem:*

If you use `cin >> message;`, it will **only take the first word** (e.g., just "Stay").

◇ ☑ *Solution:*

Use `getline(cin, message);` instead, which can read the **entire sentence** (including spaces).

💡 Sample Code:

```
#include <iostream>
#include <string> // Required for string and getline
using namespace std;

int main() {
    string name, occasion, message;

    cout << "Enter your name: ";
    getline(cin, name);

    cout << "Enter the occasion (e.g. Birthday, New Year): ";
    getline(cin, occasion);

    cout << "Enter a short message: ";
    getline(cin, message);

    cout << "\n--------------------------\n";
    cout << "🎉 Happy " << occasion << ", " << name << "! 🎉\n";
    cout << message << "\n";
    cout << "--------------------------\n";

    return 0;
}
```

🧪 Sample Input:

```
Name: Anindita
Occasion: Birthday
Message: Wishing you endless joy and success!
```

💻 Output:

```
--------------------------
🎉 Happy Birthday, Anindita! 🎉
Wishing you endless joy and success!
--------------------------
```

🌟 **Excellent Work!** You're already ahead of Day 1 – you've added:

- 🎯 `getline()` for full input
- 🎁 Random quote generation with `rand()` and `srand()`
- 🧠 Used `sizeof()` smartly to get array size
- ✨ Nice formatting with lines and tabs

☑️ Output Review:

Your output will look something like:

```
Enter your Name : Anindita
Tell me What's there in today?Name of occasion or event: Friendship

------------------------------------------------------------------------
------
Hello    Anindita Happy    Friendship    Day
------------------------------------------------------------------------
------

Enjoy through the day
------------------------------------------------------------------------
```

This is a beautiful **Day 1 Mini Project**:

✅ Personalized

✅ Encouraging

✅ Dynamic

🔍 A Few Suggestions (Optional Enhancements):

1. **Add emojis (optional):**

```
cout << "🎉 Hello " << name << "! Happy " << occasion << " Day! 🎉\n";
```

2. **Use `\n` instead of multiple `cout`s:**

```
cout << "Hello " << name << "! Happy " << occasion << " Day!\n";
```

3. **Capitalize First Letter of Occasion** (Advanced: can be tried later with functions)

🥳 Day 1 = ☑ SUCCESS

Awesome! Let's begin **Day 2: Variables & Data Types** with a fun and practical **Mini Calculator Project** 💻

🚀 Day 2: C++ Variables & Data Types

🧠 Concept of the Day:

☑️ *Variables:*

A **variable** is a container to store data.

```
int age = 20;
float height = 5.6;
char grade = 'A';
string name = "Anindita";
```

☑️ *Common Data Types:*

Type	Description	Example
`int`	Integer numbers	`int a = 10;`
`float`	Decimal numbers	`float f = 3.14;`
`double`	More precise decimals	`double d = 3.141592;`
`char`	Single character	`char c = 'A';`
`string`	Text/word (needs `<string>`)	`string s = "Hi";`
`bool`	true/false values	`bool ok = true;`

🛠️ Mini Project: Simple Calculator

🔧 *What it does:*

- Takes **two numbers**
- Takes an **operator** (+, -, *, /)
- Performs calculation and prints the result

💻 Sample Code Skeleton:

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

int main() {
    double num1, num2;
    char op;

    cout << "Enter first number: ";
    cin >> num1;

    cout << "Enter an operator (+, -, *, /): ";
    cin >> op;

    cout << "Enter second number: ";
    cin >> num2;

    double result;

    if (op == '+') {
        result = num1 + num2;
    } else if (op == '-') {
        result = num1 - num2;
    } else if (op == '*') {
        result = num1 * num2;
    } else if (op == '/') {
        if(num2 != 0)
            result = num1 / num2;
        else {
            cout << "Error: Division by zero!\n";
            return 1;
        }
    } else {
        cout << "Invalid operator!\n";
        return 1;
    }

    cout << "Result: " << result << endl;
```

```
    return 0;
}
```

✍️ Your Task:

- Write your own version of this mini calculator.
- Add your **name input** and a **welcome message**.

Perfect! 🎯

CPP_BEGINNER/minicalculator.cpp at main · Anindita531/CPP_BEGINNER

🚀 Day 3: Conditional Statements & Comparison Operators

🧠 Concept of the Day:

☑️ *if, else if, else Structure:*

```
if (condition) {
    // executes if condition is true
} else if (another_condition) {
    // executes if second condition is true
} else {
    // executes if none are true
}
```

☑️ *Comparison Operators:*

Operator	Meaning	Example (a=5, b=3)	Result
==	Equal to	a == b	false
!=	Not equal to	a != b	true
>	Greater than	a > b	true
<	Less than	a < b	false
>=	Greater or equal	a >= b	true
<=	Less or equal	a <= b	false

🛠️ Mini Project: Smart Grader + Personality Checker

◇ What it does:

1. Takes marks as input from the user (0–100)
2. Gives grade based on range:
    a. 90–100 → A+
    b. 80–89 → A
    c. 70–79 → B
    d. 60–69 → C
    e. 50–59 → D
    f. < 50 → F
3. Gives a personality quote/message based on grade

💻 Sample Output:

```
Enter your name: Anindita
Enter your marks (0-100): 86

Hi Anindita!
🎓 Your Grade: A
```

💬 Message: Great job! Keep pushing toward excellence.

🔧 Starter Code (Fill the logic):

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

int main() {
    string name;
    int marks;

    cout << "Enter your name: ";
    getline(cin, name);

    cout << "Enter your marks (0–100): ";
    cin >> marks;

    cout << "\nHi " << name << "!\n";

    // Grading System
    if (marks >= 90 && marks <= 100) {
        cout << "🎓 Your Grade: A+\n";
        cout << "💬 Message: You're a rockstar! Keep it up!\n";
    } else if (marks >= 80) {
        cout << "🎓 Your Grade: A\n";
        cout << "💬 Message: Great job! Keep pushing toward
excellence.\n";
    } else if (marks >= 70) {
        cout << "🎓 Your Grade: B\n";
        cout << "💬 Message: You're doing well, aim even higher.\n";
    } else if (marks >= 60) {
        cout << "🎓 Your Grade: C\n";
        cout << "💬 Message: You passed! Let’s improve more.\n";
    } else if (marks >= 50) {
```

```
        cout << "🎓 Your Grade: D\n";
        cout << "💬 Message: Barely made it, don’t give up!\n";
    } else if (marks < 50 && marks >= 0) {
        cout << "🎓 Your Grade: F\n";
        cout << "💬 Message: Failure is a step to success. Try
again!\n";
    } else {
        cout << "❗ Invalid Marks Entered. Please enter 0–100.\n";
    }

    return 0;
}
```

🎯 Your Task:

1. Copy or re-write the project in your style.
2. Add extra fun if you want: emojis, quotes, advice, etc.

Got it, Anindita! 🌟

Let's jump into **Day 4: Loops in C++** with **two fun mini-projects**:

→ 📊 **Table Generator**

→ 🎯 **Guess the Number Game**

🧠 Concept of the Day: Loops in C++

🔁 1. for loop

Used when the number of iterations is known.

```
for(int i = 1; i <= 5; i++) {
    cout << i << " ";
```

```
}
```

🔁 2. `while` loop

Used when the condition needs to be checked before entering the loop.

```
int i = 1;
while(i <= 5) {
    cout << i << " ";
    i++;
}
```

🔁 3. `do-while` loop

Executes the block **at least once**, even if the condition is false.

```
int i = 1;
do {
    cout << i << " ";
    i++;
} while(i <= 5);
```

🛠️ Mini Project 1: 📊 Table Generator

🎯 Goal:

Ask user to input a number.

Print its multiplication table from 1 to 10.

☑️ Sample Output:

```
Enter a number: 7
```

```
Multiplication Table of 7:
7 x 1 = 7
7 x 2 = 14
...
7 x 10 = 70
```

🛠️ Mini Project 2: 🎯 Guess the Number Game

🎯 Goal:

- Generate a random number between 1 to 100.
- Let the user keep guessing until they get it right.
- After each guess, tell if it's **too high** or **too low**.

☑️ Sample Output:

```
Guess the number (1–100): 50
Too high! Try again.
Guess the number (1–100): 30
Too low! Try again.
Guess the number (1–100): 42
Correct! You guessed it in 3 tries.
```

✨ Your Task:

You can:

1. Do both projects together ✅
2. Do only one today, and the other tomorrow.

Once you're done, send me your code. I'll review and move you to Day 5.

Let me know:

“Starting both” or **“Doing Table Generator only”** or **“Doing Guess Game only”**

CPP_BEGINNER/table_generator.cpp at main · Anindita531/CPP_BEGINNER

You're doing **really great**! You're actively polishing the project step-by-step — this is how you master coding through projects. 🔥

Now here's a **review and polish** of your current code, pointing out both **what's good** and **what can be improved**.

☑ WHAT'S GREAT IN YOUR CODE:

✅ Feature	💬 Comment
Difficulty Levels	Nicely implemented using `generateLevel()`
`srand(time(0))`	Good use of randomization
Clear UI Prompts	Friendly game instructions
Use of functions	Good modular structure (reusable, clean)
`do-while` loop	Perfect for "Play Again" feature

❌ WHAT NEEDS FIXING OR IMPROVEMENT:

1. 🔧 Input Validation Mismatch

You allow difficulty levels up to 500, but still check if guess is between 1–100 only:

```
if (userGuess < 1 || userGuess > 100)  // ❌ Not correct for all levels
```

✅ **Fix:**

Use a variable `maxLimit` to check correct range dynamically.

2. ⚠️ Missing `return` in `generateLevel()` if invalid choice

If user enters something like 5, there's no return value → leads to undefined behavior.

✅ **Fix:**

Handle invalid choices with error + loop.

3. 🎨 Add Clearer UI with Emojis or Text Symbols

Let's clean the look slightly and make it universal (with or without emojis).

☑ POLISHED FINAL VERSION (WITH LEVELS, VALIDATION)

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

// Generate a random number within range
int generateRandomNumber(int range) {
    return rand() % range + 1;
}

// Give user a hint based on their guess
void giveHint(int actual, int guess) {
    if (guess == actual) {cout << "\n🎯 Correct Guess!
Congratulations!!\n";
        cout << "Your Guess: " << guess << "\n";
        cout << "Actual Number: " << actual << "\n";
```

```
    } else if (guess < actual) {
        cout << "🔼 Too Low! Try a higher number.\n";
    } else {
        cout << "🔽 Too High! Try a lower number.\n";
    }
}

// Display farewell message
string farewell() {
    return "\n🎮 Thanks for playing the Guess Game! 🎉\n";
}

// Generate number based on selected level
int generateLevel(int ch, int &maxLimit) {
    if (ch == 1) {
        maxLimit = 50;
        cout << "\n🟢 EASY MODE: Guess a number between 1 and 50.\n";
    } else if (ch == 2) {
        maxLimit = 100;
        cout << "\n🟡 MEDIUM MODE: Guess a number between 1 and
100.\n";
    } else if (ch == 3) {
        maxLimit = 500;
        cout << "\n🔴 HARD MODE: Guess a number between 1 and 500.\n";
    } else {
        maxLimit = 100; // Default fallback
        cout << "\n⚠️ Invalid choice. Defaulting to MEDIUM mode (1-
100).\n";
    }
    return generateRandomNumber(maxLimit);
}

int main() {
    srand(time(0)); // Seed for random numbers
    char playAgain;

    cout << "🎉 Welcome to the 'Guess the Number' Game! 🎉\n";
```

```
    do {
        int userGuess, attempts = 0, levelChoice, maxLimit;

        // Ask for level choice
        cout << "\nChoose Difficulty Level:\n";
        cout << "1. Easy (1–50)\n2. Medium (1–100)\n3. Hard (1–
500)\n";
        cout << "Enter your choice: ";
        cin >> levelChoice;

        // Generate number based on level
        int actualNumber = generateLevel(levelChoice, maxLimit);

        // Start guessing loop
        while (true) {
            cout << "\nEnter your guess: ";
            cin >> userGuess;
            attempts++;

            if (userGuess < 1 || userGuess > maxLimit) {
                cout << "🚫 Invalid guess! Please enter a number
between 1 and " << maxLimit << ".\n";
                continue;
            }

            giveHint(actualNumber, userGuess);

            if (userGuess == actualNumber) {
                cout << "✅ You guessed it in " << attempts << "
attempt(s)!\n";
                break;
            }
        }

        // Ask to play again
        cout << "\n🔁 Do you want to play again? (y/n): ";
        cin >> playAgain;

    } while (playAgain == 'y' || playAgain == 'Y');
```

```
    cout << farewell();
    return 0;
}
```

✨ OUTPUT EXAMPLE

```
🎉 Welcome to the 'Guess the Number' Game! 🎉

Choose Difficulty Level:
1. Easy (1–50)
2. Medium (1–100)
3. Hard (1–500)
Enter your choice: 3

🔴 HARD MODE: Guess a number between 1 and 500.

Enter your guess: 230
🔼 Too Low! Try a higher number.

Enter your guess: 350
🔽 Too High! Try a lower number.

Enter your guess: 301
🎯 Correct Guess! Congratulations!!
Your Guess: 301
Actual Number: 301
✅ You guessed it in 3 attempt(s)!

🔁 Do you want to play again? (y/n): n

🎮 Thanks for playing the Guess Game! 🎉
```

Want Next Project? 🔧

Let me know if you want:

1. 🧮 **Mini Calculator (with operations)**
2. 🎲 **Dice Roll Simulator**
3. 🧠 **Number Memory Challenge**
4. 📊 **Basic Stats Analyzer**

Alright Anindita! 🙂

You've done **amazing** so far:

- Day 1 → **Personalized Quotes App** ✅
- Day 2 → **Calculator Project** ✅
- Day 3 → **Guess the Number Game** ✅

Now it's time for **Day 4** — where we'll **level up your C++ knowledge** and prepare for **bigger projects**. 🚀

📘 Day 5 – Functions, Scope & Modular Programming

Today, we'll focus on **how C++ programs are structured** so you can write **cleaner**, **reusable**, and **bug-free** code.

We'll also build a **mini-project** at the end. 🎯

1. Theory: What You'll Learn Today

(a) Types of Functions

There are **4 types** based on **input** & **output**:

Function Type	Example	Input	Output
1️⃣ No Input, No Output	`void greet()`	❌	❌
2️⃣ Input, No Output	`void printSquare(int n)`	✅	❌
3️⃣ No Input, Output	`int getRandom()`	❌	✅
4️⃣ Input & Output	`int add(int a, int b)`	✅	✅

(b) Function Declaration & Definition

```
// Declaration
int add(int, int);

// Definition
int add(int x, int y) {
    return x + y;
}

// Usage
int sum = add(5, 10);
```

(c) Scope of Variables

- **Local Variables** → Declared inside functions → accessible **only there**
- **Global Variables** → Declared outside → accessible **anywhere**
- **Static Variables** → Retains value **even after function ends**

Example:

```
int g = 10;  // Global variable

void demo() {
    static int count = 0;  // Retains value across calls
    count++;
    cout << "Called " << count << " times\n";
}
```

(d) Header Files & Modular Code

- Use `#include` to organize your project into **separate files**.
- Example:
    - `math_utils.cpp` → all math functions
    - `main.cpp` → game logic

This keeps your code **cleaner** and **scalable**.

2. Mini Project: Math Trainer 🧠

A **Math Quiz App** to test user's skills.

Features:

✅ Randomly generates math problems

✅ Supports addition, subtraction, multiplication, division

✅ Counts score

✅ Uses **functions** for modularity

Sample Output

```
🎯 Welcome to Math Trainer 🎯

Question 1: 7 + 3 = ?
Your Answer: 10
✅ Correct! Score: 1

Question 2: 5 × 6 = ?
Your Answer: 31
❌ Wrong! Correct Answer: 30

Final Score: 7/10
```

3. Today's Plan

Time	Task
30 mins	Learn about functions & scope
45 mins	Practice 5 function-based problems
45 mins	Build **Math Trainer Project**
15 mins	Revision + Q&A

4. Homework

- Create a program with **5 functions**:
  - `add()`
  - `subtract()`
  - `multiply()`
  - `divide()`
  - `power(base, exponent)`
- Test them **individually** and **inside a calculator app**.

Alright 😊 Since you're on **Day 5** of your C++ learning journey, the next step is to **build a small project**: **Math Trainer** 🧠✨

This project will help you **practice loops, functions, conditions, random number generation, and user interaction**. Before writing code, let me explain the **full project structure** in detail so you understand how everything will work.

📌 Math Trainer Project – Structure & Flow

🎯 Project Goal

A console-based C++ program where:

- The program **generates random math problems**.
- The user must **solve them**.
- The program **evaluates answers**, **gives hints**, **tracks score**, and **shows final performance**.

🔷 Project Features

We'll break it into **modules**:

1. Welcome & Instructions Module

- Display a **welcome message**.
- Show how the game works.
- Example:

```
Welcome to Math Trainer! 🧠
Solve as many math problems as you can.
Enter your answer and I'll tell you if it's correct.
Type 0 to quit anytime.
```

2. Difficulty Selection Module

- User selects difficulty:
  - **Easy** → numbers 1–10
  - **Medium** → numbers 1–50
  - **Hard** → numbers 1–100

We'll store difficulty in a variable `maxRange`.

3. Question Generator Module

- Uses `rand()` + `srand(time(0))` to generate random problems.
- Types of problems:
  - Addition → `a + b`
  - Subtraction → `a - b`
  - Multiplication → `a * b`
  - Division → `a / b`
- Randomly pick one operator for variety.

Example:

```
int num1 = rand() % maxRange + 1;
int num2 = rand() % maxRange + 1;
char op = operators[rand() % 4];  // '+', '-', '*', '/'
```

4. User Input & Answer Checking Module

- Ask the user for an answer.
- Compare it with the **correct result**.
- If correct → increase score.
- If wrong → show correct answer.

Example interaction:

```
Q1) 7 * 6 = ?
Your Answer: 42 ✅ Correct!
```

5. Scoring & Performance Module

- Keep track of:
  - **Total questions attempted**
  - **Correct answers**
  - **Accuracy percentage**
- At the end:

```
You attempted 10 questions.
Correct Answers: 7 ✅
Wrong Answers: 3 ❌
Accuracy: 70%
```

6. Replay Module

- After finishing, ask:

```
Do you want to play again? (y/n)
```

- If y → restart game.
- If n → exit with a goodbye message.

◇ Project Architecture

```
MathTrainer/
│
├── main.cpp            // Main file controlling game flow
├── functions.cpp       // All helper functions
```

```
├── functions.h          // Function declarations
└── README.txt           // Instructions (optional)
```

◇ Functions We Will Create

Function Name	Purpose
`void welcome()`	Displays welcome screen & instructions
`int selectLevel()`	Lets user choose difficulty
`void askQuestion()`	Generates random question
`bool checkAnswer()`	Verifies user's answer
`void showScore()`	Shows final performance
`bool playAgain()`	Asks if user wants to restart

◇ Concepts You'll Practice

- **Loops** → for generating multiple questions
- **Functions** → modularize code
- **Switch / If-Else** → handle operators
- **Random Numbers** → generate problems
- **Strings** → messages & feedback
- **Score Tracking** → learn basic data handling

CPP_BEGINNER/mathtrainer.cpp at main · Anindita531/CPP_BEGINNER

Let's start **Day 6** of your **C++ from Scratch + Mini Projects** learning plan.

Today's focus will be on **Arrays** and **Strings** — two very important topics for both problem-solving and projects.

We'll also build a **mini project** at the end of the lesson. 🚀

🧮 Day 6 – Arrays & Strings in C++

🔷 Part 1: Arrays (Basics)

An **array** is a **collection of elements** of the **same data type** stored in **contiguous memory locations**.

1. Array Declaration

```
int marks[5]; // Declare an integer array of size 5
```

2. Array Initialization

```
int marks[5] = {90, 85, 78, 92, 88};
```

3. Accessing Array Elements

```
cout << marks[0];  // First element → 90
cout << marks[3];  // Fourth element → 92
```

4. Input & Output of Arrays

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

int main() {
    int n;
    cout << "Enter number of students: ";
```

```
    cin >> n;

    int marks[n];

    // Input marks
    cout << "Enter marks:\n";
    for (int i = 0; i < n; i++) {
        cin >> marks[i];
    }

    // Output marks
    cout << "\nMarks are:\n";
    for (int i = 0; i < n; i++) {
        cout << "Student " << i + 1 << ": " << marks[i] << "\n";
    }
}
```

◇ Part 2: Strings (Basics)

Strings in C++ are used to **store and manipulate text**.

1. Declaration

```
string name = "Anindita";
```

2. Input & Output

```
string name;
cout << "Enter your name: ";
getline(cin, name);
cout << "Hello, " << name << "!";
```

3. Common String Functions

Function	Example	Output
`.length()`	`name.length()`	Returns length
`.substr(pos, len)`	`name.substr(0, 3)`	First 3 chars
`.find("text")`	`name.find("di")`	Index position
`.append()`	`name.append("Ghosh")`	Adds text
`.compare()`	`s1.compare(s2)`	0 if equal

◇ Part 3: Mini Project – Student Grade Analyzer 🏆

🎯 Project Goal

- Input **names** and **marks** of students.
- Calculate:
  - **Average marks**
  - **Highest marks**
  - **Lowest marks**
- Display results in a **clean format**.

📌 Code: Student Grade Analyzer

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

int main() {
    int n; cout << "Enter total number of students: ";
    cin >> n;

    string names[n];
    int marks[n];
```

```
    int total = 0;

    // Input student details
    cout << "\nEnter names and marks:\n";
    for (int i = 0; i < n; i++) {
        cout << "Student " << i + 1 << " name: ";
        cin >> names[i];
        cout << "Marks: ";
        cin >> marks[i];
        total += marks[i];
    }

    // Calculate average, highest, lowest
    double avg = (double)total / n;
    int highest = marks[0], lowest = marks[0];
    string topStudent = names[0], lowStudent = names[0];

    for (int i = 1; i < n; i++) {
        if (marks[i] > highest) {
            highest = marks[i];
            topStudent = names[i];
        }
        if (marks[i] < lowest) {
            lowest = marks[i];
            lowStudent = names[i];
        }
    }

    // Display Results
    cout << "\n================ STUDENT GRADE ANALYZER
================\n";
    cout << "Average Marks : " << avg << "\n";
    cout << "Highest Marks : " << highest << " (" << topStudent <<
")\n";
    cout << "Lowest Marks  : " << lowest << " (" << lowStudent <<
")\n";
    cout <<
"=========================================================\n";
```

```
    return 0;
}
```

◇ Output Example

```
Enter total number of students: 3
Enter names and marks:
Student 1 name: Riya
Marks: 85
Student 2 name: Anindita
Marks: 92
Student 3 name: Priya
Marks: 78

================ STUDENT GRADE ANALYZER ================
Average Marks : 85
Highest Marks : 92 (Anindita)
Lowest Marks  : 78 (Priya)
========================================================
```

◇ What You Learned Today (Day 6)

✅ Arrays → Declaration, Initialization, Traversal

✅ Strings → Input, Output, Functions

✅ Mini Project → **Student Grade Analyzer**

Alright 🙂 Let's move to **Day 7** of your **C++ from Scratch + Mini Projects** journey 🚀

Today's focus will be on **Functions in C++** — both **normal functions** and **recursion** — and we'll build a **Function-based Scientific Calculator** mini project.

🧮 Day 7 – Functions & Recursion in C++

🔷 Part 1: What is a Function?

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

We **call** a function whenever we need it, which makes code **reusable** and **organized**.

1. Function Syntax

```
return_type function_name(parameters) {
    // code
    return value;  // optional
}
```

2. Types of Functions

Type	Example	Use Case
Predefined	`sqrt(16)`	Already built-in
User-defined	`int add(int a, int b)`	We create them
Void functions	`void greet()`	No return value
Parameterized	`int sum(int x, int y)`	Take inputs
Recursive	`factorial(n)`	Function calls itself

🔷 Part 2: Normal Functions Example

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

int add(int a, int b) {
```

```
    return a + b;
}

int main() {
    int x = 5, y = 10;
    cout << "Sum = " << add(x, y);
    return 0;
}
```

Output:

```
Sum = 15
```

◇ Part 3: Recursion Basics

A function **calls itself** until a **base condition** is met.

Example: Factorial

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

long long factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}

int main() {
    int num;
    cout << "Enter number: ";
    cin >> num;
    cout << "Factorial = " << factorial(num);
    return 0;
}
```

Input: 5

Output: 120

◇ Part 4: Mini Project – Scientific Calculator 🧮

🎯 Project Goal

- Build a **function-based calculator**.
- Support operations:

✅ Addition

✅ Subtraction

✅ Multiplication

✅ Division

✅ Power

✅ Factorial

✅ Square root

📌 Code: Function-based Scientific Calculator

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

// Function declarations
float add(float a, float b) { return a + b; }
float subtract(float a, float b) { return a - b; }
float multiply(float a, float b) { return a * b; }
float divide(float a, float b) { return (b != 0) ? a / b : NAN; }
```

```
double power(double base, double exp) { return pow(base, exp); }
double squareRoot(double num) { return sqrt(num); }

long long factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}

int main() {
    int choice;
    float num1, num2;

    cout << "\n========== SCIENTIFIC CALCULATOR ==========\n";
    cout << "1. Addition\n2. Subtraction\n3. Multiplication\n4.
Division\n";
    cout << "5. Power\n6. Square Root\n7. Factorial\n8. Exit\n";
    cout << "==========================================\n";

    while (true) {
        cout << "\nEnter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter two numbers: ";
                cin >> num1 >> num2;
                cout << "Result = " << add(num1, num2) << "\n";
                break;

            case 2:
                cout << "Enter two numbers: ";
                cin >> num1 >> num2;
                cout << "Result = " << subtract(num1, num2) << "\n";
                break;

            case 3:
                cout << "Enter two numbers: ";
                cin >> num1 >> num2;
                cout << "Result = " << multiply(num1, num2) << "\n";
```

```
            break;

        case 4:
            cout << "Enter two numbers: ";
            cin >> num1 >> num2;
            if (num2 == 0)
                cout << "Error: Division by zero!\n";
            else
                cout << "Result = " << divide(num1, num2) << "\n";
            break;

        case 5:
            cout << "Enter base and exponent: ";
            cin >> num1 >> num2;
            cout << "Result = " << power(num1, num2) << "\n";
            break;

        case 6:
            cout << "Enter a number: ";
            cin >> num1;
            cout << "Square Root = " << squareRoot(num1) << "\n";
            break;

        case 7:
            int n;
            cout << "Enter an integer: ";
            cin >> n;
            cout << "Factorial = " << factorial(n) << "\n";
            break;

        case 8:
            cout << "Exiting... Thank you!\n";
            return 0;

        default:
            cout << "Invalid choice! Please try again.\n";
    }
}
```

```
}
```

◇ Output Example

```
========== SCIENTIFIC CALCULATOR ==========
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Power
6. Square Root
7. Factorial
8. Exit
==========================================

Enter your choice: 1
Enter two numbers: 5 8
Result = 13
```

◇ What You Learned Today (Day 7)

✅ Functions — declaration, calling, and returning values

✅ Recursion — factorial example

✅ Built a **Scientific Calculator** mini project

CPP_BEGINNER/minicalculator.cpp at main · Anindita531/CPP_BEGINNER

Perfect 😊

Since you've confirmed, we'll move to **Day 7** of your **C++ learning plan**.

Today's focus is on **Functions – Intermediate & Advanced Level**.

We'll make this session **hands-on + project-based** so you understand **how functions work in real-world C++ programs**.

Day 7 – Functions (Intermediate to Advanced)

1. Topics to Cover

(A) Function Overloading

- **Definition:** Two or more functions having the same name but different parameters.
- **Use Case:** It makes code **cleaner** and **easier to maintain**.

Example:

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

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

double add(double a, double b) {
    return a + b;
}

int main() {
    cout << add(5, 10) << endl;       // Integer version
    cout << add(2.5, 3.7) << endl;    // Double version
    return 0;
}
```

(B) Default Arguments

- Default values are assigned to function parameters when the caller doesn't pass them.

Example:

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

void greet(string name = "Guest") {
    cout << "Hello, " << name << "!" << endl;
}

int main() {
    greet("Anindita");  // Output: Hello, Anindita!
    greet();            // Output: Hello, Guest!
    return 0;
}
```

(C) Pass by Value vs Pass by Reference

- **Pass by Value:** Copies the value → changes **do not** affect the original.
- **Pass by Reference:** Uses memory reference → changes **affect** the original.

Example:

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

void byValue(int x) {
    x += 10;
}

void byReference(int &x) {
    x += 10;
}
```

```
int main() {
    int n = 20;
    byValue(n);
    cout << "After byValue: " << n << endl;  // Still 20

    byReference(n);
    cout << "After byReference: " << n << endl;  // Now 30
    return 0;
}
```

(D) Inline Functions

- Used for **small functions** to improve performance.
- The compiler **replaces** the function call with its body.

Example:

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

inline int square(int x) {
    return x * x;
}

int main() {
    cout << square(5) << endl; // 25
    cout << square(10) << endl; // 100
    return 0;
}
```

2. Mini Project – C++ Math Trainer (V2) 🎯

We'll create an **upgraded math trainer** using **functions** effectively.

Features:

✅ Addition, subtraction, multiplication, division

✅ Random question generation

✅ Score tracking

✅ Pass by reference for score update

✅ Function overloading for easy handling

✅ User-friendly UI

Great 🙂 So for **Day 8** in your C++ learning journey, we'll focus on **Recursion**.

I'll explain everything step by step — **concept → examples → mini-project**.

Day 8: Recursion in C++ 🧩

1. What is Recursion?

Recursion is a process where a **function calls itself** until a specific **base condition** is met.

Structure:

```
returnType functionName(parameters) {
    if(base_condition) {
        return some_value;  // Stop recursion
    }
    // Recursive call
    return functionName(modified_parameters);
}
```

2. Key Concepts

- **Base Case** → When to stop recursion.
- **Recursive Case** → When the function calls itself.
- **Stack Behavior** → Recursion uses the **call stack**.

3. Examples

Example 1: Factorial

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

int factorial(int n) {
    if(n == 0 || n == 1)  // Base Case
        return 1;
    return n * factorial(n - 1);  // Recursive Call
}

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

Example 2: Fibonacci Series

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

int fibonacci(int n) {
    if(n == 0) return 0;
    if(n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

```
int main() {
    for(int i = 0; i < 10; i++) {
        cout << fibonacci(i) << " ";
    }
    return 0;
}
```

4. Mini Project — "Recursion Math Trainer" 🎯

We'll build a **menu-driven program**:

Features:

✅ Factorial Calculator

✅ Fibonacci Series

✅ Sum of N natural numbers

✅ Power of a number

Workflow:

1. Show a menu to the user.
2. Take the choice.
3. Call the respective recursive function.
4. Ask if the user wants to continue.

https://github.com/Anindita531/CPP_BEGINNER

Day 9 will focus on **2D Arrays in C++**

Since you're moving from **Day 8 (Recursion)** → **Day 9 (2D Arrays)**, I'll explain everything step by step:

📌 Day 9 – 2D Arrays in C++ 🧩

1. What is a 2D Array?

A **2D array** is basically an **array of arrays**.

It stores **rows** and **columns** of data, like a **matrix**.

Syntax:

```
data_type arrayName[rows][cols];
```

Example:

```
int arr[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

Here:

- `arr` → 2 rows × 3 columns.
- `arr[0][0] = 1`
- `arr[1][2] = 6`

2. Input and Output of 2D Arrays

Example:

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

int main() {
    int rows, cols;
```

```
    cout << "Enter rows and columns: ";
    cin >> rows >> cols;

    int arr[rows][cols];

    cout << "Enter elements:\n";
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> arr[i][j];
        }
    }

    cout << "\nMatrix:\n";
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

3. Important Operations on 2D Arrays

We'll cover **Day 9** in three levels:

(A) Basics

- Take input and display.
- Sum of all elements.
- Transpose of a matrix.

(B) Intermediate

- Sum of each row & column.

- Find maximum and minimum.
- Diagonal sum.

(C) Advanced (Mini Project)

Project: “2D Matrix Calculator”

Features:

- Matrix addition
- Matrix subtraction
- Matrix multiplication
- Transpose of a matrix

4. Mini Project: 2D Matrix Calculator 🎯

Features:

- Input two matrices.
- Choose operation:
  - Addition
  - Subtraction
  - Multiplication
  - Transpose
- Show the result.

Menu Example:

```
========== MATRIX CALCULATOR ==========
1. Matrix Addition
2. Matrix Subtraction
3. Matrix Multiplication
4. Transpose
5. Exit
=======================================
```

```
Enter your choice:
```

5. Day 9 Plan

Step	Topic	Practice
1	2D Array Basics	Create, input, display
2	Row & Column Operations	Sum, max, min
3	Diagonals	Primary & secondary diagonal sums
4	Matrix Calculator	Mini project

CPP_BEGINNER/2DArray.cpp at main · Anindita531/CPP_BEGINNER

Alright 🙂 Since you've completed **Day 9: 2D Arrays (Matrix Operations)**,

let's move to **Day 10**.

Day 10 — Strings in C++ 🧩

Strings are one of the most **important concepts** in C++ and widely used in **DSA, coding interviews, and projects**.

Today, we'll **start from scratch**, understand the **basics of strings**, and build a **String Utility Tool** as a mini-project.

Lesson Plan

1. What You'll Learn Today

- What is a **string** in C++

- Ways to declare and initialize strings
- Common string operations
- Important functions (`length`, `substr`, `find`, `erase`, `replace`, etc.)
- Practice mini-problems
- **Mini Project** → *String Utility Tool*

2. Understanding Strings

A. Using C-style strings (char arrays)

```
char name[20] = "Anindita";
cout << name; // Output: Anindita
```

- Fixed size
- Ends with `'\0'` (null character)
- Limited functionalities

B. Using `std::string` ☑ (Recommended)

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

int main() {
    string name = "Anindita";
    cout << name;  // Output: Anindita
}
```

- Dynamic size ✅
- Easy to manipulate ✅
- Comes with **powerful functions** ✅

3. Most Useful String Functions

Function	Description	Example	Output
`s.length()`	Get length	`"hello".length()`	5
`s.substr(pos, len)`	Substring	`"hello".substr(1,3)`	"ell"
`s.find("lo")`	Find position	`"hello".find("lo")`	3
`s.erase(pos, len)`	Remove part	`"hello".erase(2,2)`	"heo"
`s.insert(pos, str)`	Insert text	`"hello".insert(2,"yy")`	"heyyllo"
`s.replace(p,l,str)`	Replace part	`"hello".replace(1,2,"yy")`	"hyylo"
`s.compare(str)`	Compare two strings	`"abc".compare("abc")`	0 (equal)

4. Mini Practice Problems ☑

Problem 1 — Count vowels in a string

Input: `Anindita`

Output: 4

Problem 2 — Check if a string is palindrome

Input: `madam` → Output: `Palindrome`

Input: `hello` → Output: `Not Palindrome`

Problem 3 — Reverse a string

Input: `DataScience`

Output: `ecneicSataD`

5. Mini Project — String Utility Tool 🚀

Features:

- Take a string input from the user
- Display:
    - Length of string
    - Number of vowels & consonants
    - Reverse of the string
    - Palindrome check
    - Word count
- Use **functions** for each task ✅

Do you want me to directly give you the **full C++ code** for the **String Utility Tool** mini-project,

or first explain **step by step** with diagrams and flow before coding?