C++ Programming Course Outline (Covered Topics)

1. Introduction to Programming and C++

- ☐ **What is programming?**
- ☐ **Basics of C++ syntax**
- ☐ **Writing your first program (Hello World)**
- ☐ **Input and output (cin, cout)**
- ☐ **Data types and variables**

2. Control Structures

- ☐ **Conditional statements (if, else, switch)**
- ☐ **Loops (for, while, do-while)**

3. Functions

- ☐ **What are functions?**
- ☐ **Defining and calling functions**
- ☐ **Function parameters and return values**

4. Arrays

- ☐ **Declaring and initializing arrays**
- ☐ **Accessing and modifying elements**
- ☐ **Looping through arrays**
- ☐ **Taking user input in arrays**
- ☐ **Summing array elements**

5. Strings

- ☐ **String basics in C++**
- ☐ **Using `string` class**
- ☐ **Input and output of strings**
- ☐ **String manipulation basics**

6. Pointers

- ☐ **What is a pointer?**
- ☐ **Pointer declaration and usage**
- ☐ **Pointer arithmetic basics**
- ☐ **Pointer to pointer concept**

7. Structures (struct)

- ☐ **Defining structures**
- ☐ **Declaring structure variables**
- ☐ **Array of structures**

- ☐ **Nested structures**
- ☐ **Passing structures to functions**

8. File Handling

- ☐ **Introduction to file handling**
- ☐ **Writing to files using `ofstream`**
- ☐ **Reading from files using `ifstream`**
- ☐ **Reading files line by line**
- ☐ **Closing files properly**

🌱 1. What is Programming?

Programming means giving instructions to the computer so it can do a task for you — like adding numbers, showing messages, or saving data.

💻 2. C++ Programming (Basic)

C++ is a programming language used to tell the computer what to do.

✨ **Your First C++ Program**

```
#include<iostream>

using namespace std;


int main() {
    cout << "Hello, Usama!";
    return 0;
}
```

Line	Meaning
`#include<iostream>`	Tells the computer to use input/output functions (like cout).
`using namespace std;`	Lets you write `cout` instead of `std::cout`.
`int main()`	Starting point of the program.
`cout << "Hello, Usama!";`	Shows message on the screen.
`return 0;`	Ends the program successfully.

Some Important concepts in C++:

📍 1. cout

- Used to print something on the screen.
- Example: cout << "Welcome!";

📍 2. Semicolon (;)

- Every statement in C++ ends with a `;`.

📍 3. Comments

- Used to explain code (not run by computer).
- Single line: `// This is a comment`
- Multi-line:

```
/* This is a
  multi-line comment */
```

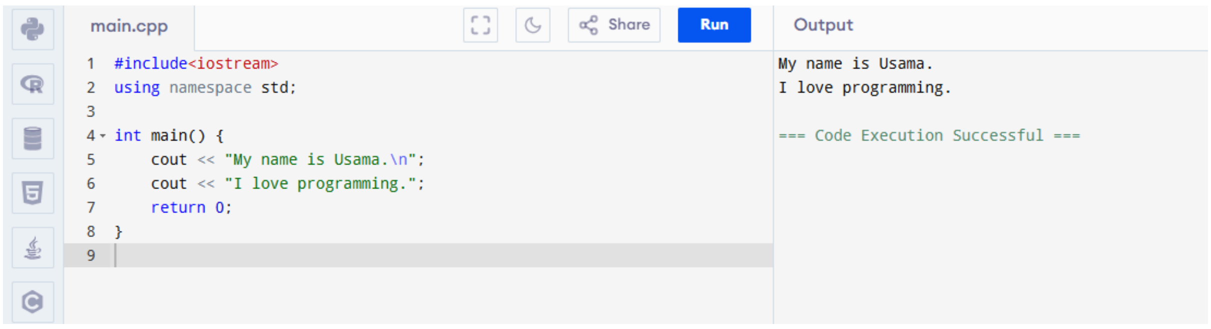
Task: Write a program that shows:

My name is Usama.

I love programming.

Program:

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

int main() {
   cout << "My name is Usama.\n";
   cout << "I love programming.";
   return 0;
}
```



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

int main() {
    cout << "My name is Usama.\n";
    cout << "I love programming.";
    return 0;
}

```

```
My name is Usama.
I love programming.

=== Code Execution Successful ===
```

Variables and Data Types in C++:

💡 What is a Variable?

A **variable** is like a **box** that stores information.

Example:
If you want to store your age, you can create a box (variable) called `age`.

🔤 What is a Data Type?

Data types tell the computer **what kind of data** you are storing in the variable.

Common Data Types in C++

Data Type	Meaning	Example
`int`	Integer (whole number)	10, 25, -5

`float`	Decimal number	3.14, 7.5
`char`	Single character	'A', 'z'
`string`	Text (words or sentence)	"Usama", "Hello"
`bool`	True or False	true, false

🧪 Examples:

1. Integer:

```
int age = 21;

cout << "Age: " << age;
```

2. Float:

```
float pi = 3.14;

cout << "Value of PI: " << pi;
```

3. Char:

```
char grade = 'A';

cout << "Your grade is: " << grade;
```

4. String:

```
string name = "Usama";

cout << "Hello " << name;
```

Note: To use `string`, add `#include<string>` at the top

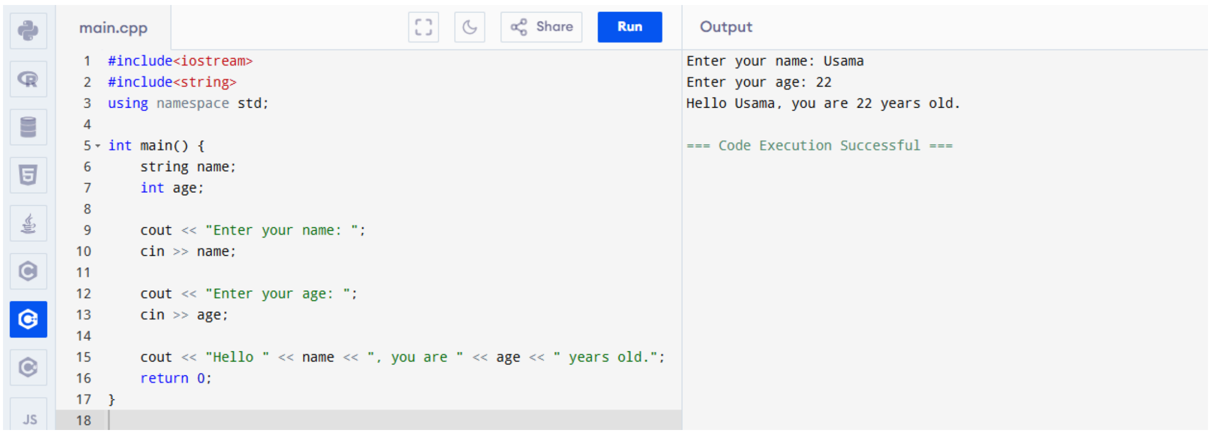
5. Boolean:

```
bool passed = true;

cout << "Passed: " << passed;
```

Input from User

Let the user type their own value.

```
#include<iostream>

#include<string>

using namespace std;


int main() {

    string name;

    int age;


    cout << "Enter your name: ";

    cin >> name;


    cout << "Enter your age: ";

    cin >> age;


    cout << "Hello " << name << ", you are " << age << " years old.";

    return 0;

}
```

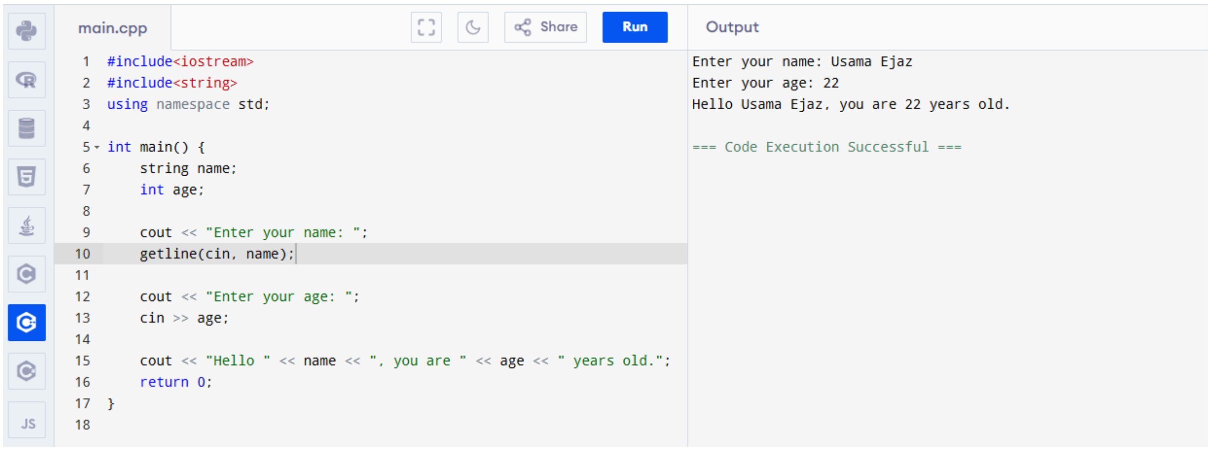

main.cpp

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

int main() {
    string name;
    int age;

    cout << "Enter your name: ";
    cin >> name;

    cout << "Enter your age: ";
    cin >> age;

    cout << "Hello " << name << ", you are " << age << " years old.";
    return 0;
}
```

Output

```
Enter your name: Usama
Enter your age: 22
Hello Usama, you are 22 years old.

=== Code Execution Successful ===
```

📌 If you want to input full name (with spaces), use `getline(cin, name);` instead of `cin >> name;`


main.cpp

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

int main() {
    string name;
    int age;

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

    cout << "Enter your age: ";
    cin >> age;

    cout << "Hello " << name << ", you are " << age << " years old.";
    return 0;
}
```

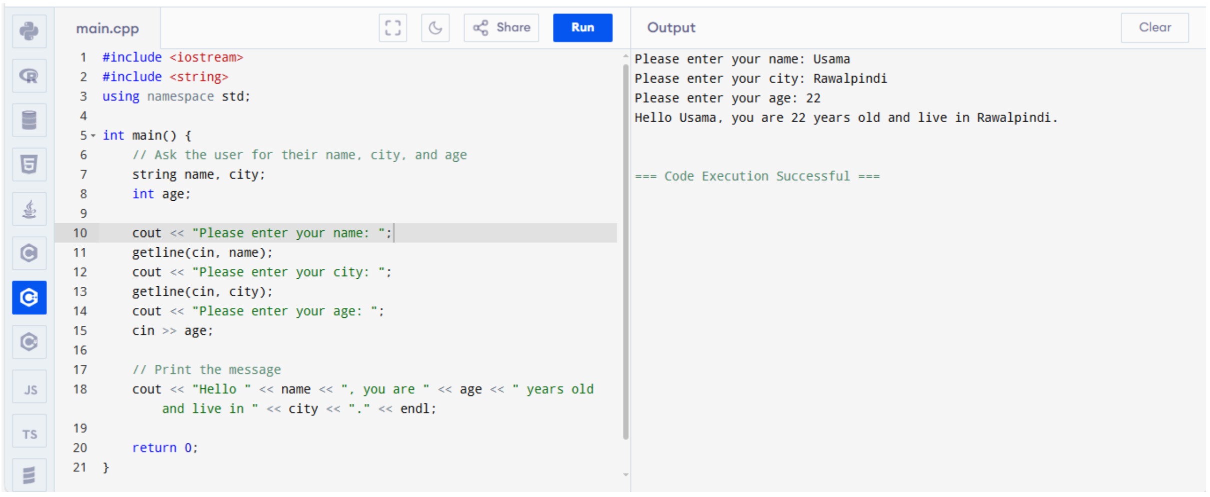
Output

```
Enter your name: Usama Ejaz
Enter your age: 22
Hello Usama Ejaz, you are 22 years old.

=== Code Execution Successful ===
```

Task

```cpp
#include <iostream>

#include <string>

using namespace std;
```

```
int main() {

    // Ask the user for their name, city, and age

    string name, city;

    int age;



    cout << "Please enter your name: ";

    getline(cin, name);

    cout << "Please enter your city: ";

    getline(cin, city);

    cout << "Please enter your age: ";

    cin >> age;



    // Print the message

    cout << "Hello " << name << ", you are " << age << " years old and live in " << city << "."
<< endl;



    return 0;

}
```



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

int main() {
    // Ask the user for their name, city, and age
    string name, city;
    int age;

    cout << "Please enter your name: ";
    getline(cin, name);
    cout << "Please enter your city: ";
    getline(cin, city);
    cout << "Please enter your age: ";
    cin >> age;

    // Print the message
    cout << "Hello " << name << ", you are " << age << " years old
        and live in " << city << "." << endl;

    return 0;
}
```

```
Please enter your name: Usama
Please enter your city: Rawalpindi
Please enter your age: 22
Hello Usama, you are 22 years old and live in Rawalpindi.


=== Code Execution Successful ===
```

Operators and Conditions (if/else):

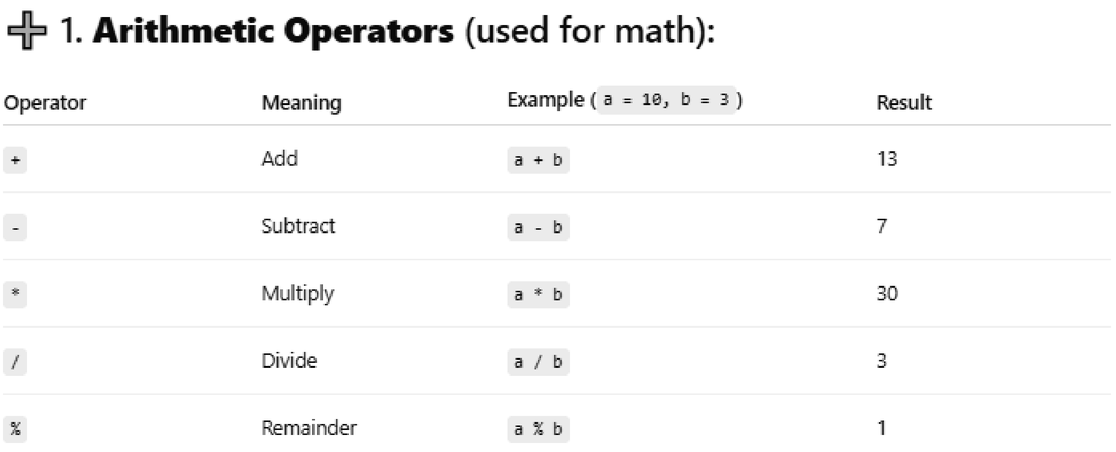
💡 What Are Operators?

Operators are symbols that perform actions like math, comparison, or logic.

➕ 1. Arithmetic Operators (used for math):

Operator	Meaning	Example (`a = 10, b = 3`)	Result
`+`	Add	`a + b`	13
`-`	Subtract	`a - b`	7

*	**Multiply**	a * b	**30**
/	**Divide**	a / b	**3**
%	**Remainder**	a % b	**1**



➕ 1. **Arithmetic Operators** (used for math):

Operator	Meaning	Example (`a = 10, b = 3`)	Result
`+`	Add	`a + b`	13
`-`	Subtract	`a - b`	7
`*`	Multiply	`a * b`	30
`/`	Divide	`a / b`	3
`%`	Remainder	`a % b`	1

🧪 Example:

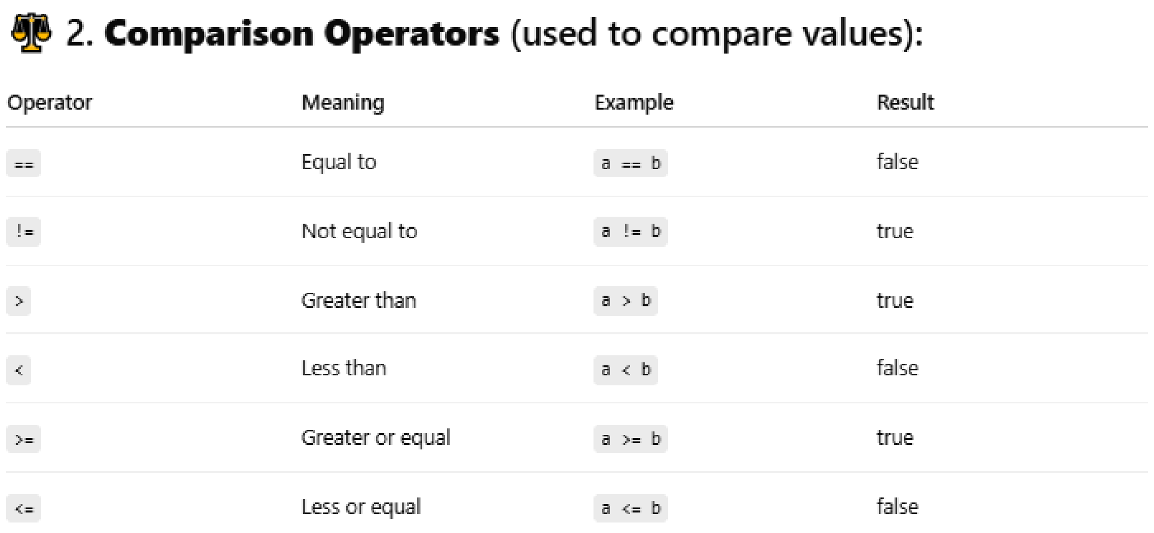
int a = 10, b = 3;

cout << "Sum: " << a + b;

⚖️ 2. Comparison Operators (used to compare values):

Operator	Meaning	Example	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



⚖️ 2. **Comparison Operators** (used to compare values):

Operator	Meaning	Example	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

🤔 What is if/else?

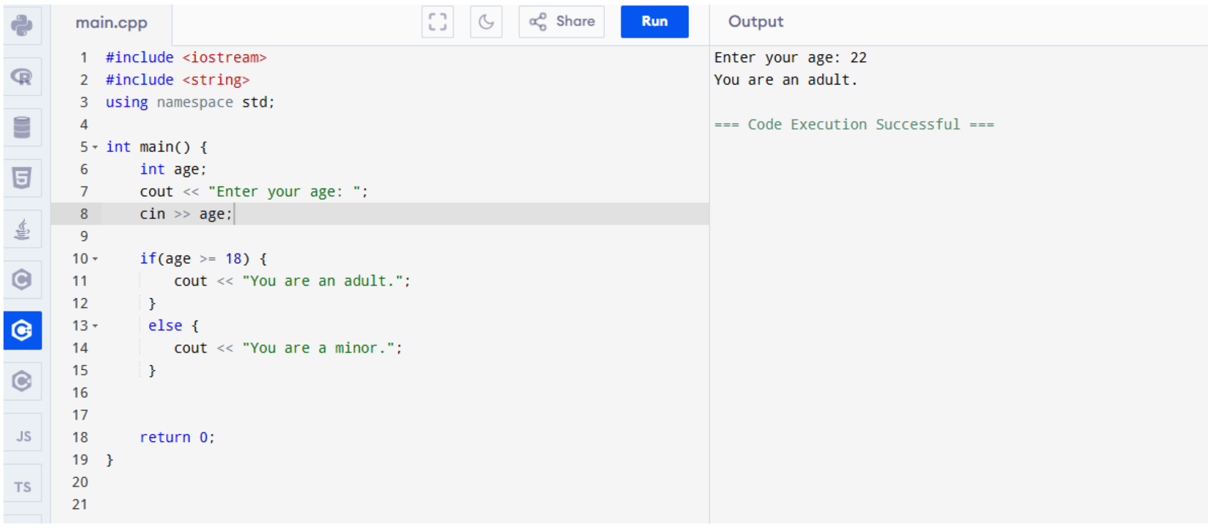
`if` and `else` are used to **make decisions** in a program.

3. if/else Condition Example:

```
#include <iostream>

#include <string>

using namespace std;


int main() {

    int age;

    cout << "Enter your age: ";

    cin >> age;


    if(age >= 18) {

        cout << "You are an adult.";

     }

     else {

        cout << "You are a minor.";

     }




    return 0;

}
```

💭 What it does:

- If `age` is 18 or more → says "adult"
- Otherwise → says "minor"

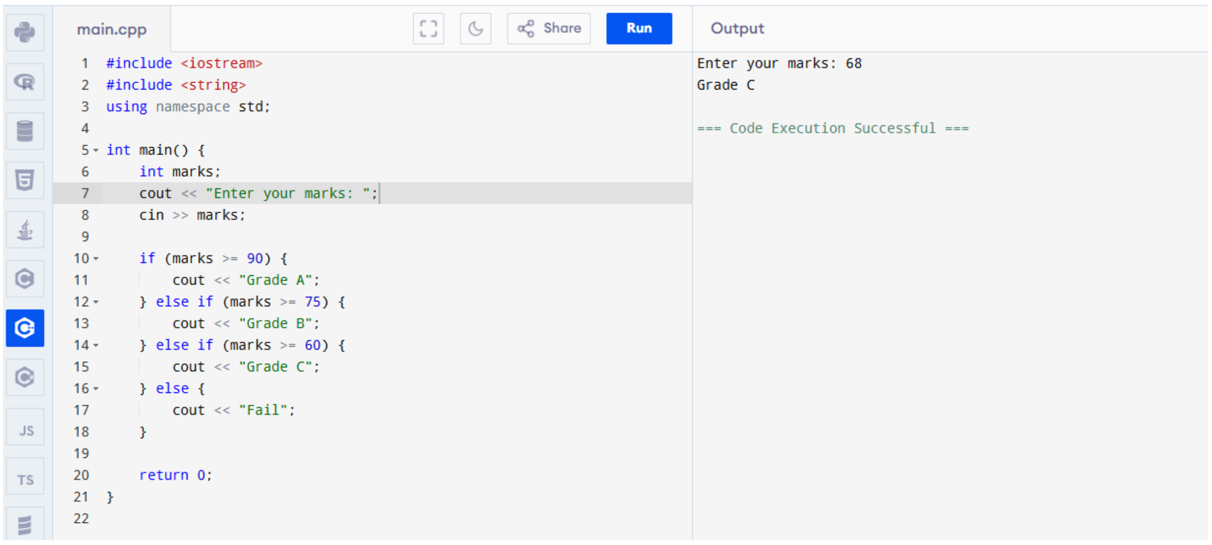


```
#include <iostream>
#include <string>
using namespace std;

int main() {
    int age;
    cout << "Enter your age: ";
    cin >> age;

    if(age >= 18) {
        cout << "You are an adult.";
     }
     else {
        cout << "You are a minor.";
     }


    return 0;
}
```

```
Enter your age: 22
You are an adult.

=== Code Execution Successful ===
```

4. else if Condition Example:

```
#include <iostream>

#include <string>

using namespace std;


int main() {

    int marks;

    cout << "Enter your marks: ";

    cin >> marks;


    if (marks >= 90) {

        cout << "Grade A";

    } else if (marks >= 75) {

        cout << "Grade B";

    } else if (marks >= 60) {

        cout << "Grade C";

    } else {
        cout << "Fail";
    }

    return 0;
}
```



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

int main() {
    int marks;
    cout << "Enter your marks: ";
    cin >> marks;

    if (marks >= 90) {
        cout << "Grade A";
    } else if (marks >= 75) {
        cout << "Grade B";
    } else if (marks >= 60) {
        cout << "Grade C";
    } else {
        cout << "Fail";
    }

    return 0;
}
```

```
Enter your marks: 68
Grade C

=== Code Execution Successful ===
```

Extra: Logical Operators

Symbol	Meaning	Example
`&&`	AND	`a > 5 && b < 10`
`		`
`!`	NOT	`!(a == b)`

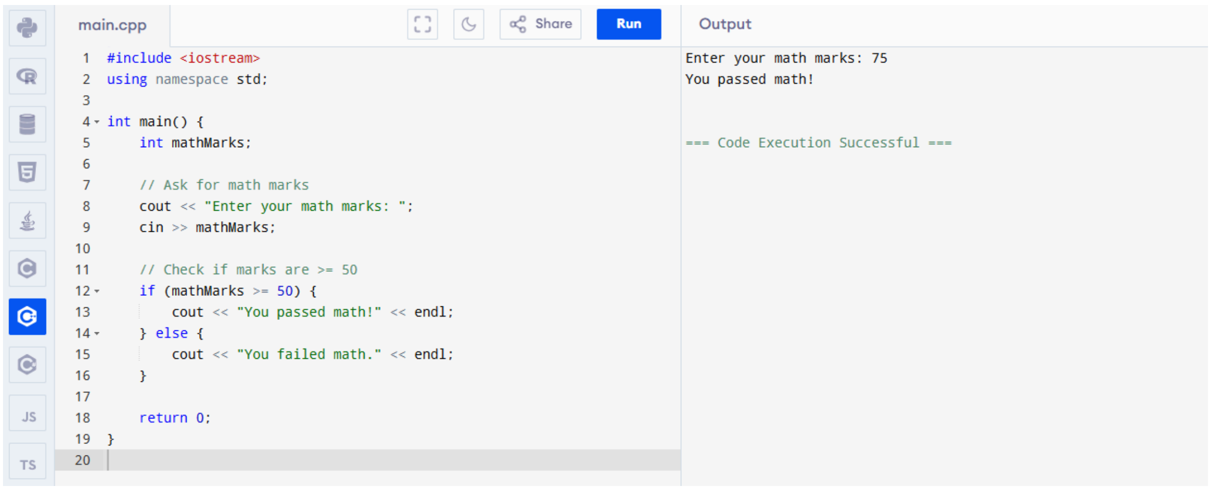
Symbol	Meaning	Example
`&&`	AND	`a > 5 && b < 10`
`		`
`!`	NOT	`!(a == b)`

Task:

Make a program that:

1. Asks for your **math marks**.
2. If marks are `>= 50` → print `"You passed math!"`
3. Otherwise → print `"You failed math."`

```
#include <iostream>

using namespace std;


int main() {

    int mathMarks;


    // Ask for math marks

    cout << "Enter your math marks: ";

    cin >> mathMarks;


    // Check if marks are >= 50

    if (mathMarks >= 50) {

        cout << "You passed math!" << endl;
    } else {
        cout << "You failed math." << endl;
    }

    return 0;
}
```



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

int main() {
    int mathMarks;

    // Ask for math marks
    cout << "Enter your math marks: ";
    cin >> mathMarks;

    // Check if marks are >= 50
    if (mathMarks >= 50) {
        cout << "You passed math!" << endl;
    } else {
        cout << "You failed math." << endl;
    }

    return 0;
}
```

```
Enter your math marks: 75
You passed math!

=== Code Execution Successful ===
```

Loops in C++:

💡 What is a Loop?

A **loop** means **repeat something again and again** until a condition is met.

Example:
If you want to print `"Hello"` 5 times, you don't have to write it 5 times. Just use a loop.

🔁 1. while loop

📘 Syntax:

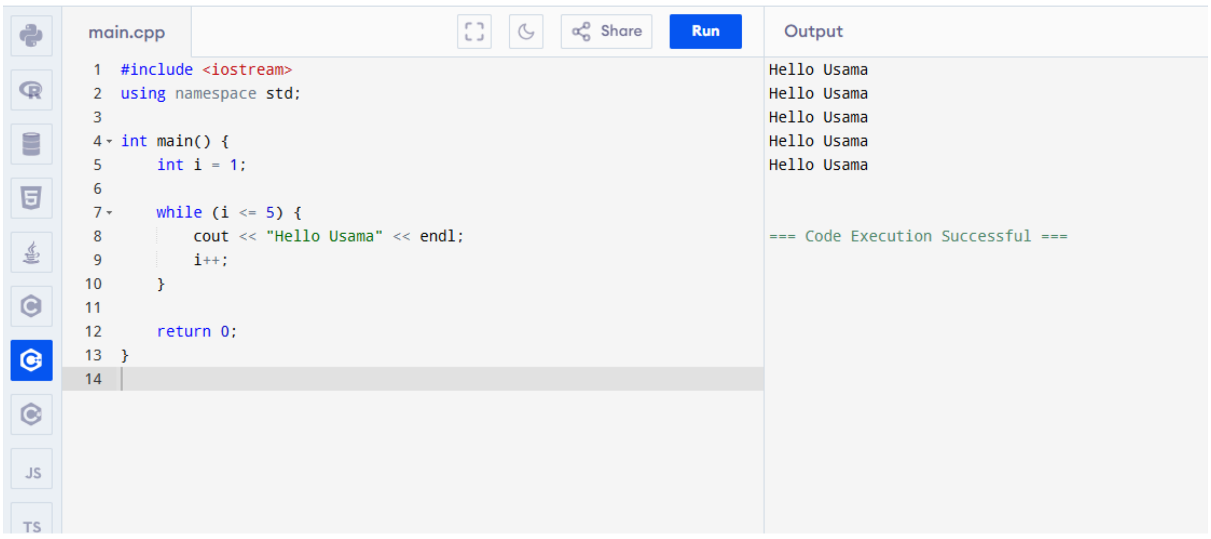
```
while(condition) {
    // code to repeat
}
```

Example:

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

int main() {
    int i = 1;

    while (i <= 5) {
        cout << "Hello Usama" << endl;
        i++;
    }

    return 0;
}
```

Program logic

- Start at `i = 1`
- Print "Hello Usama"
- Increase `i` by 1
- Stop when `i > 5`



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

int main() {
    int i = 1;

    while (i <= 5) {
        cout << "Hello Usama" << endl;
        i++;
    }

    return 0;
}

```

```
Hello Usama
Hello Usama
Hello Usama
Hello Usama
Hello Usama


=== Code Execution Successful ===
```

🔁 2. for loop

📘 Syntax:

```
for(start; condition; update) {
   // repeat this code
}
```

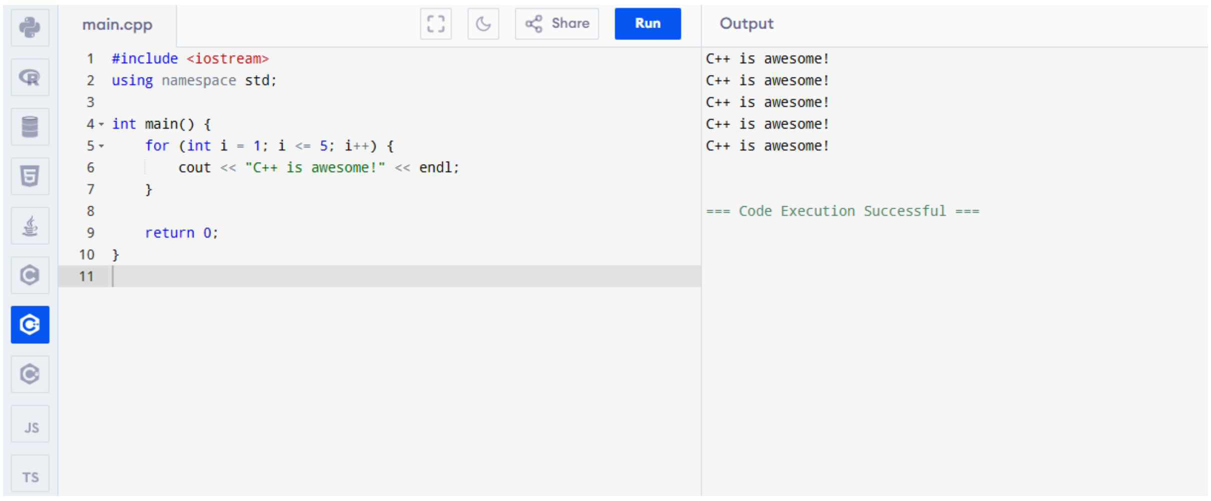
Example:

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

int main() {
   for (int i = 1; i <= 5; i++) {
       cout << "C++ is awesome!" << endl;
   }
```

    return 0;

}

This prints the line 5 times.



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

int main() {
    for (int i = 1; i <= 5; i++) {
        cout << "C++ is awesome!" << endl;
    }

    return 0;
}
```

```
C++ is awesome!
C++ is awesome!
C++ is awesome!
C++ is awesome!
C++ is awesome!


=== Code Execution Successful ===
```

.

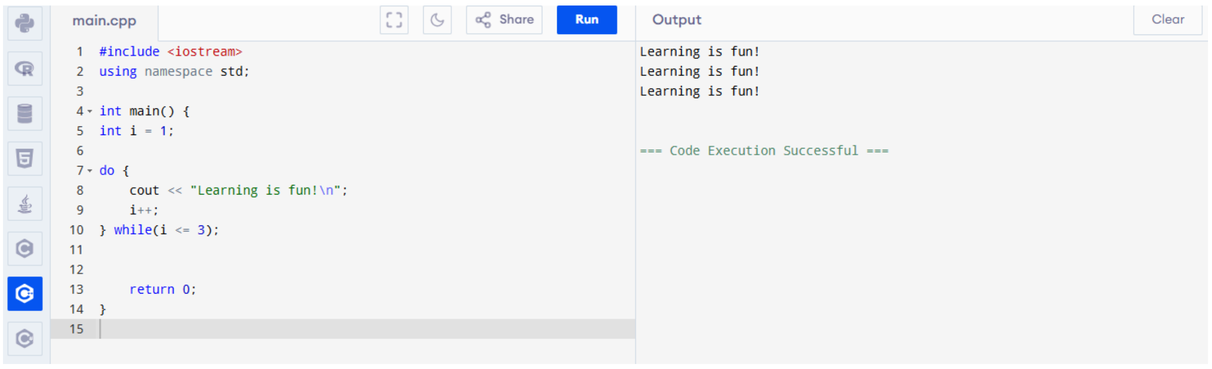
🔁 3. do-while loop

📘 Syntax:

```
do {
    // run this
} while(condition);
```

Example:

```
#include <iostream>

using namespace std;


int main() {

    int i = 1;
    do {
        cout << "Learning is fun!" << endl;
        i++;
    } while (i <= 3);

    return 0;
}
```

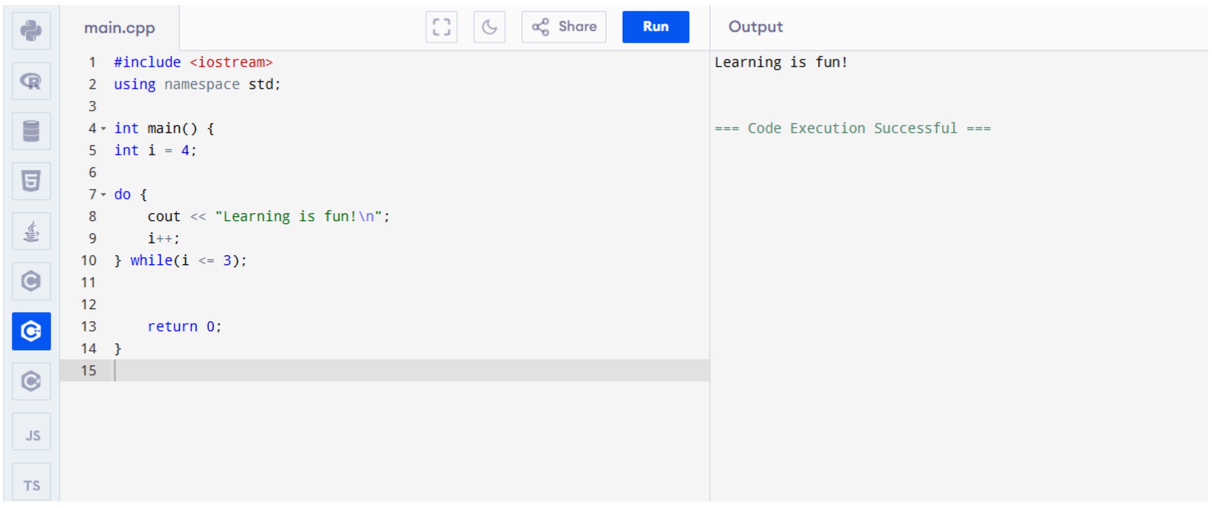


```
#include <iostream>
using namespace std;

int main() {
int i = 1;

do {
    cout << "Learning is fun!\n";
    i++;
} while(i <= 3);


    return 0;
}
```

```
Learning is fun!
Learning is fun!
Learning is fun!


=== Code Execution Successful ===
```

💡 Special Point:

`do-while` runs **at least one time**, even if the condition is false.

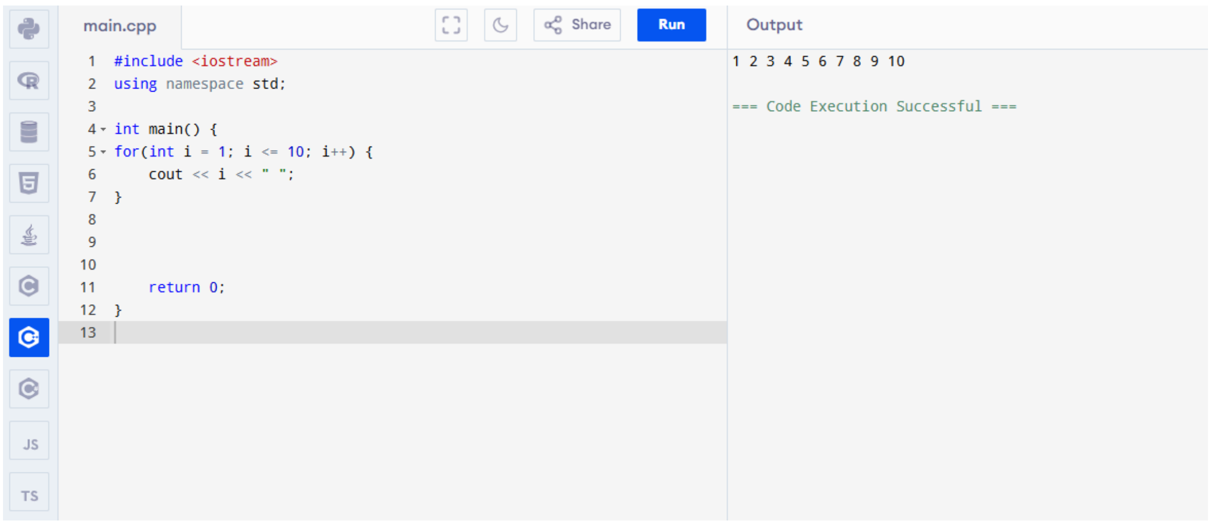


```
#include <iostream>
using namespace std;

int main() {
int i = 4;

do {
    cout << "Learning is fun!\n";
    i++;
} while(i <= 3);


    return 0;
}
```

```
Learning is fun!


=== Code Execution Successful ===
```

Example: Print 1 to 10:

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

int main() {
for(int i = 1; i <= 10; i++) {
    cout << i << " ";
}



    return 0;
}
```



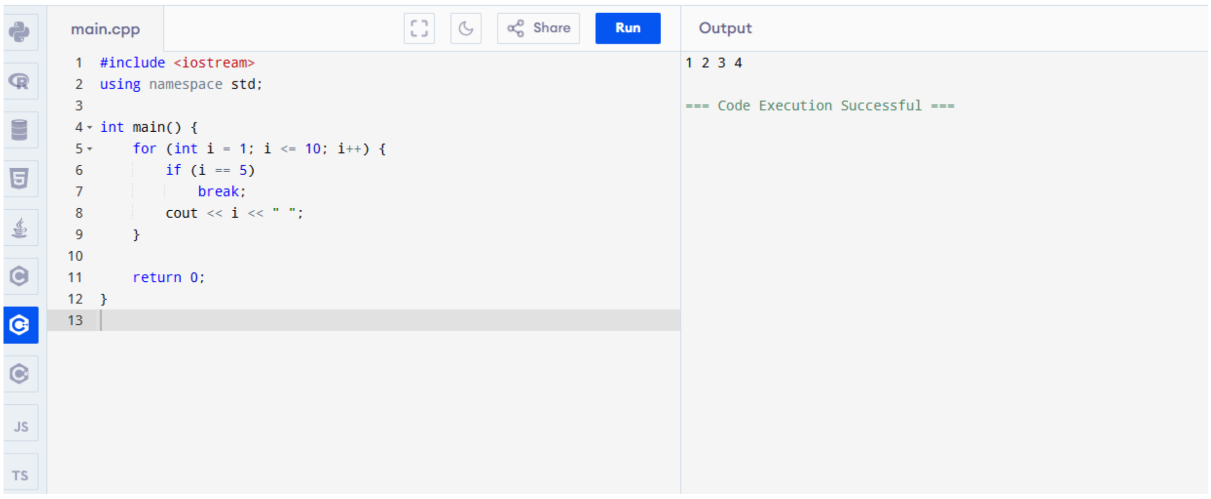
```
#include <iostream>
using namespace std;

int main() {
for(int i = 1; i <= 10; i++) {
    cout << i << " ";
}



    return 0;
}

```

```
1 2 3 4 5 6 7 8 9 10

=== Code Execution Successful ===
```

🚫 Break and Continue

🛑 `break`: Stops the loop

```
#include <iostream>

using namespace std;


int main() {

    for (int i = 1; i <= 10; i++) {

        if (i == 5)

            break;

        cout << i << " ";

    }


    return 0;

}
```



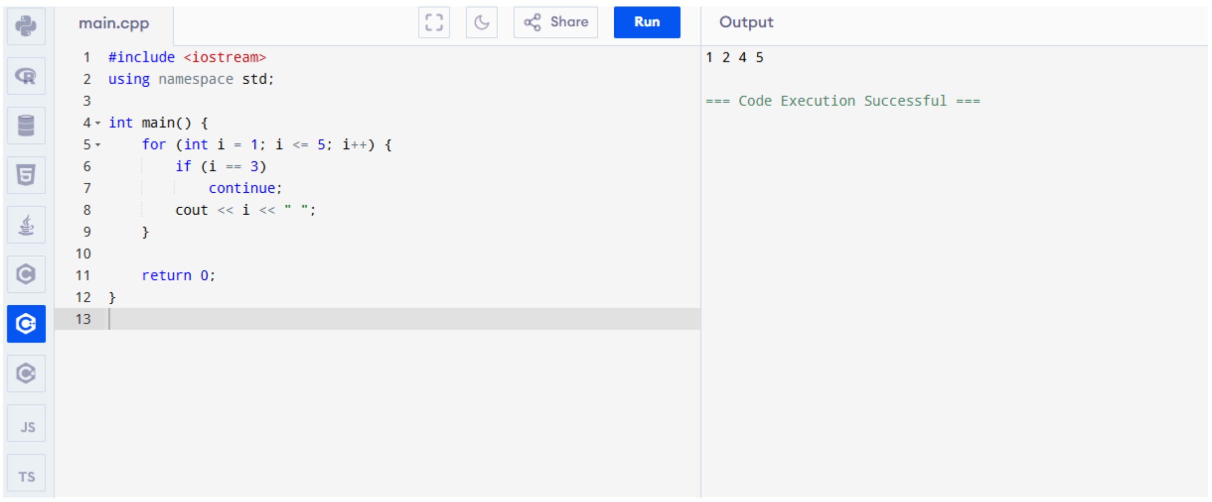
```
#include <iostream>
using namespace std;

int main() {
    for (int i = 1; i <= 10; i++) {
        if (i == 5)
            break;
        cout << i << " ";
    }

    return 0;
}
```

```
1 2 3 4

=== Code Execution Successful ===
```

🔁 `continue`: Skips current step

```
#include <iostream>

using namespace std;
```

```
int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3)
            continue;
        cout << i << " ";
    }

    return 0;
}
```



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

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3)
            continue;
        cout << i << " ";
    }

    return 0;
}
```

```
1 2 4 5

=== Code Execution Successful ===
```

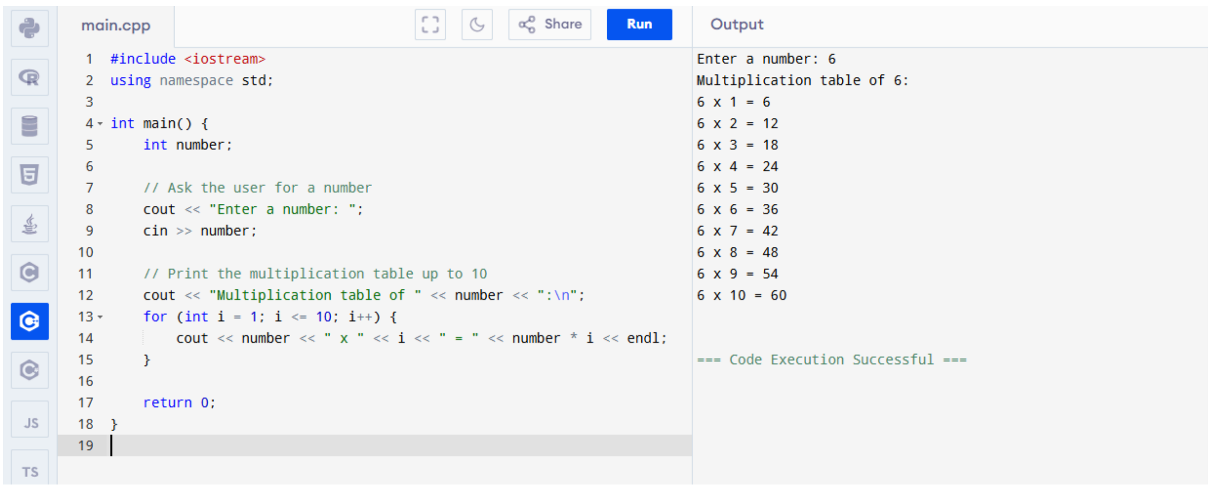
Task:

Write a program to:

- Ask the user for a number.
- Print the **table** of that number up to 10.

```
#include <iostream>
```

```
using namespace std;

int main() {
    int number;

    // Ask the user for a number
    cout << "Enter a number: ";
    cin >> number;

    // Print the multiplication table up to 10
    cout << "Multiplication table of " << number << ":\n";
    for (int i = 1; i <= 10; i++) {
        cout << number << " x " << i << " = " << number * i << endl;
    }

    return 0;
}
```



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

int main() {
    int number;

    // Ask the user for a number
    cout << "Enter a number: ";
    cin >> number;

    // Print the multiplication table up to 10
    cout << "Multiplication table of " << number << ":\n";
    for (int i = 1; i <= 10; i++) {
        cout << number << " x " << i << " = " << number * i << endl;
    }

    return 0;
}
```

```
Enter a number: 6
Multiplication table of 6:
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60


=== Code Execution Successful ===
```

Functions in C++

🔧 What is a Function?

A **function** is a **block of code** that does **one task**, and you can **reuse** it anywhere in your program.

👉 Think of it like a **machine**:
You give it something (input), it gives you something back (output).

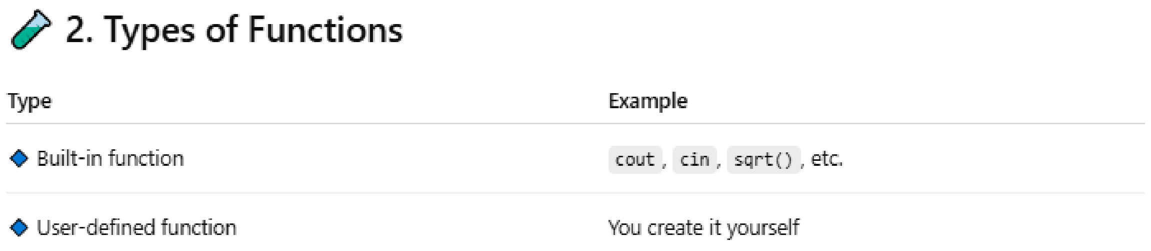
🧱 1. Why Use Functions?

- ☐ Makes code **clean**
- ☐ Makes code **reusable**
- ☐ Helps break big tasks into **small parts**

🧪 2. Types of Functions

Type	Example

🔹 **Built-in function**	`cout`, `cin`, `sqrt()`, **etc.**
🔹 **User-defined function**	**You create it yourself**



🧪 2. Types of Functions

Type	Example
🔹 Built-in function	`cout` , `cin` , `sqrt()` , etc.
🔹 User-defined function	You create it yourself

🛠️ 3. Structure of a Function

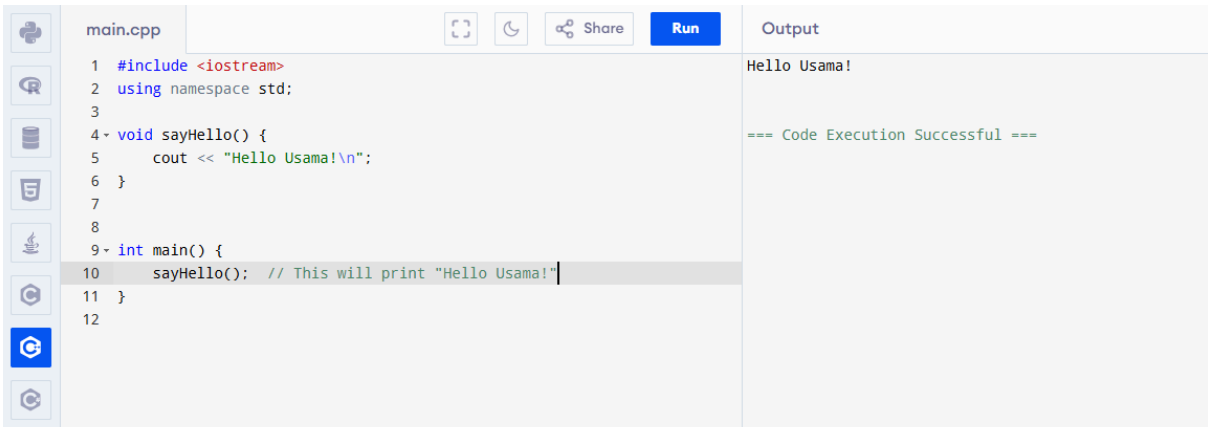
```
return_type function_name(parameters) {
    // code
    return something;
}
```

4. Simple Example (No Return, No Parameters)

```
void sayHello() {
    cout << "Hello Usama!\n";
}
```

To use it (call it):

```
sayHello();  // This will print "Hello Usama!"
```



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

void sayHello() {
    cout << "Hello Usama!\n";
}


int main() {
    sayHello();  // This will print "Hello Usama!"
}
```

```
Hello Usama!


=== Code Execution Successful ===
```

🧮 5. Function With Parameters

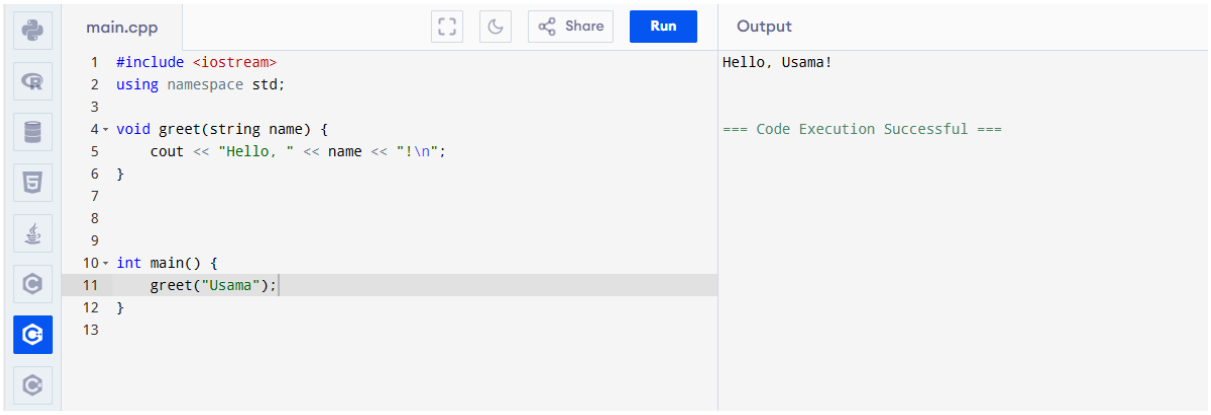
void greet(string name) {

    cout << "Hello, " << name << "!\n";

}

Call it like this:

greet("Usama");



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

void greet(string name) {
    cout << "Hello, " << name << "!\n";
}



int main() {
    greet("Usama");
}
```

```
Hello, Usama!


=== Code Execution Successful ===
```

6. Function With Return Value

int add(int a, int b) {

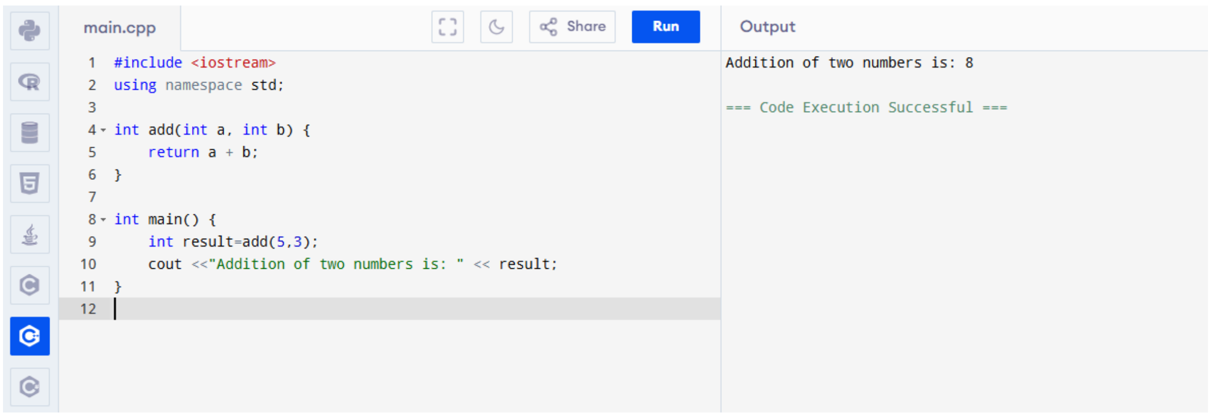
    return a + b;

}

Call it:

int result = add(5, 3);

cout << result; // Output: 8

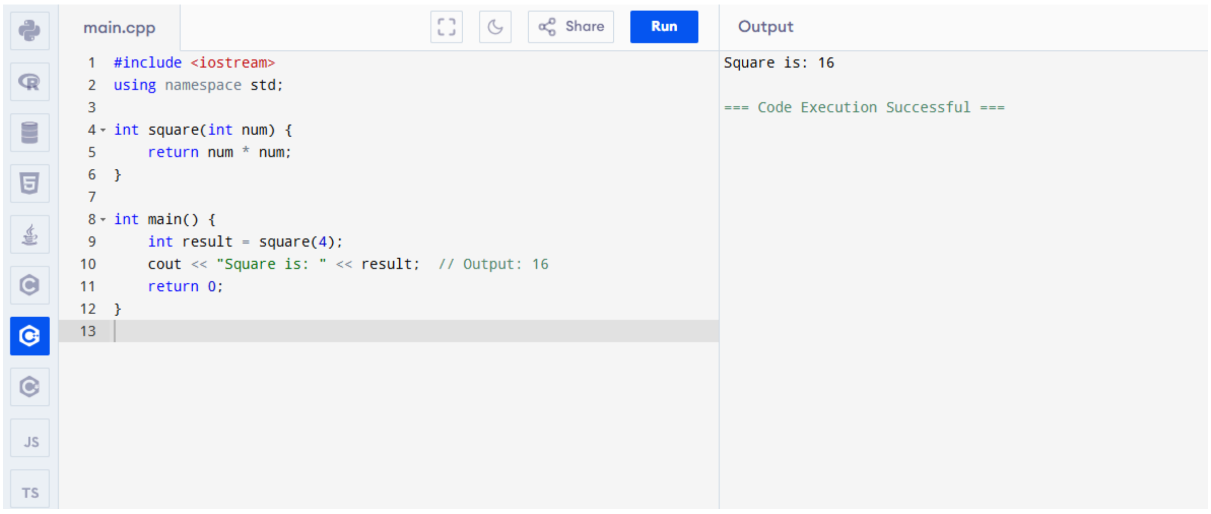


```cpp
#include <iostream>
using namespace std;

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

int main() {
    int result=add(5,3);
    cout <<"Addition of two numbers is: " << result;
}
```

```
Addition of two numbers is: 8

=== Code Execution Successful ===
```

Example:

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

int square(int num) {
    return num * num;
}

int main() {
    int result = square(4);
    cout << "Square is: " << result;  // Output: 16
    return 0;
}
```



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

int square(int num) {
    return num * num;
}

int main() {
    int result = square(4);
    cout << "Square is: " << result;  // Output: 16
    return 0;
}

```

```
Square is: 16

=== Code Execution Successful ===
```

🤔 Function Call Flow:

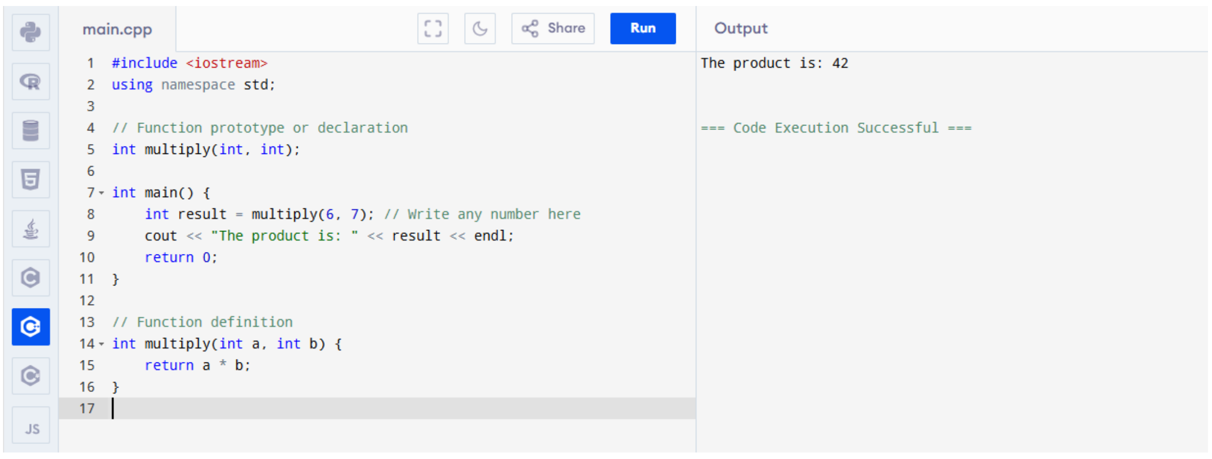
1. main() starts
2. main calls `square(4)`
3. `square()` does its job and sends result back
4. main uses the result

Note: If you define function below main, then you must declare it first:

```
int add(int, int);  // Function prototype


int main() {

    cout << add(3, 4);

}


int add(int a, int b) {

    return a + b;

}
```

Full Program:

```
#include <iostream>

using namespace std;


// Function prototype or declaration

int multiply(int, int);


int main() {

    int result = multiply(6, 7); // Aap yahan koi bhi numbers de sakte hain

    cout << "The product is: " << result << endl;

    return 0;

}


// Function definition

int multiply(int a, int b) {

    return a * b;

}
```



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

// Function prototype or declaration
int multiply(int, int);

int main() {
    int result = multiply(6, 7); // Write any number here
    cout << "The product is: " << result << endl;
    return 0;
}

// Function definition
int multiply(int a, int b) {
    return a * b;
}
```

```
The product is: 42


=== Code Execution Successful ===
```

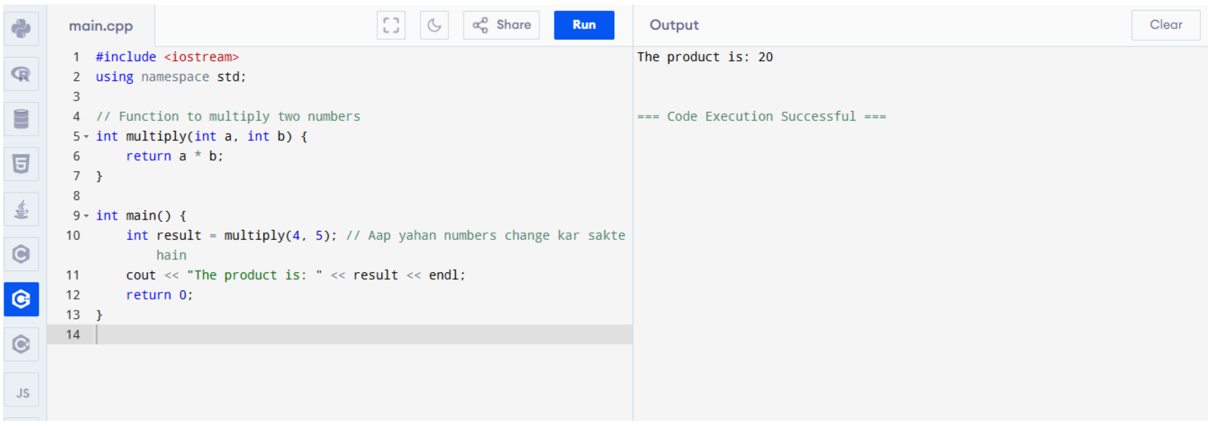
🧪 Task:

🔸 **Write a function called `multiply()` that:**

- **takes 2 numbers**
- **returns their product**

➡️ **Then call it in `main()` and show the result.**

```
#include <iostream>

using namespace std;


// Function to multiply two numbers

int multiply(int a, int b) {

    return a * b;

}


int main() {

    int result = multiply(4, 5); // Aap yahan numbers change kar sakte hain

    cout << "The product is: " << result << endl;

    return 0;

}
```



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

// Function to multiply two numbers
int multiply(int a, int b) {
    return a * b;
}

int main() {
    int result = multiply(4, 5); // Aap yahan numbers change kar sakte
        hain
    cout << "The product is: " << result << endl;
    return 0;
}

```

```
The product is: 20


=== Code Execution Successful ===
```

Arrays in C++

🧱 What is an Array?

An **array** is a **box** that stores **many values** of the **same type** using **one name**.

Example:

Imagine a **carton with 5 cups** inside.
You don't give each cup a different name — you say: "Cup[0], Cup[1], ... Cup[4]".

Why Use Arrays?

- ☐ Store lots of data in **one variable**
- ☐ Easy to **loop** through
- ☐ Helps with **sorting, searching**, and more

🧪 1. Declaring an Array

```
int numbers[5];  // an array of 5 integers
```

It looks like this in memory:

```
Index:   0   1    2    3    4
Value:  [ ]  [ ]  [ ]   [ ]   [ ]
```

Indexes always start from **0.**

2. Initializing an Array

```
int nums[3] = {10, 20, 30};
```

This means:

nums[0] = 10

nums[1] = 20

nums[2] = 30

3. Accessing and Printing Elements

```
cout << nums[1];  // Output: 20
```

4. Loop Through an Array

```
for(int i = 0; i < 3; i++) {
    cout << nums[i] << " ";
}
```

Output: 10 20 30

✍️ 5. Take Input from User in Array

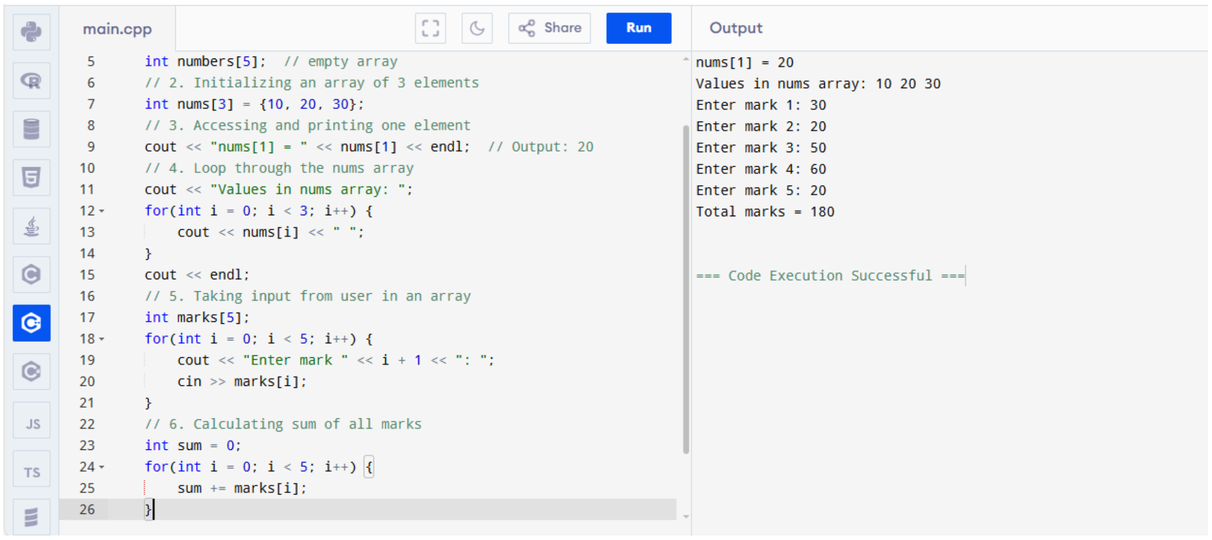
```
int marks[5];

for(int i = 0; i < 5; i++) {

    cout << "Enter mark " << i+1 << ": ";

    cin >> marks[i];

}
```

💡 6. Sum of All Array Elements

```
int sum = 0;

for(int i = 0; i < 5; i++) {

    sum += marks[i];

}

cout << "Total = " << sum;
```

Program:

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

int main() {
   // 1. Declaring an array of 5 integers
   int numbers[5];  // empty array

   // 2. Initializing an array of 3 elements
   int nums[3] = {10, 20, 30};

   // 3. Accessing and printing one element
   cout << "nums[1] = " << nums[1] << endl;  // Output: 20

   // 4. Loop through the nums array
   cout << "Values in nums array: ";
   for(int i = 0; i < 3; i++) {
       cout << nums[i] << " ";
   }
   cout << endl;

   // 5. Taking input from user in an array
   int marks[5];
   for(int i = 0; i < 5; i++) {
       cout << "Enter mark " << i + 1 << ": ";
       cin >> marks[i];
   }

   // 6. Calculating sum of all marks
   int sum = 0;
   for(int i = 0; i < 5; i++) {
       sum += marks[i];
   }

   cout << "Total marks = " << sum << endl;

   return 0;
}
```



```
int numbers[5];  // empty array
// 2. Initializing an array of 3 elements
int nums[3] = {10, 20, 30};
// 3. Accessing and printing one element
cout << "nums[1] = " << nums[1] << endl;  // Output: 20
// 4. Loop through the nums array
cout << "Values in nums array: ";
for(int i = 0; i < 3; i++) {
    cout << nums[i] << " ";
}
cout << endl;
// 5. Taking input from user in an array
int marks[5];
for(int i = 0; i < 5; i++) {
    cout << "Enter mark " << i + 1 << ": ";
    cin >> marks[i];
}
// 6. Calculating sum of all marks
int sum = 0;
for(int i = 0; i < 5; i++) {
    sum += marks[i];
}
```

```
nums[1] = 20
Values in nums array: 10 20 30
Enter mark 1: 30
Enter mark 2: 20
Enter mark 3: 50
Enter mark 4: 60
Enter mark 5: 20
Total marks = 180


=== Code Execution Successful ===
```

🧪 Task:

Write a program that:

- **Takes 10 numbers from user into an array**
- **Finds and prints the largest number**

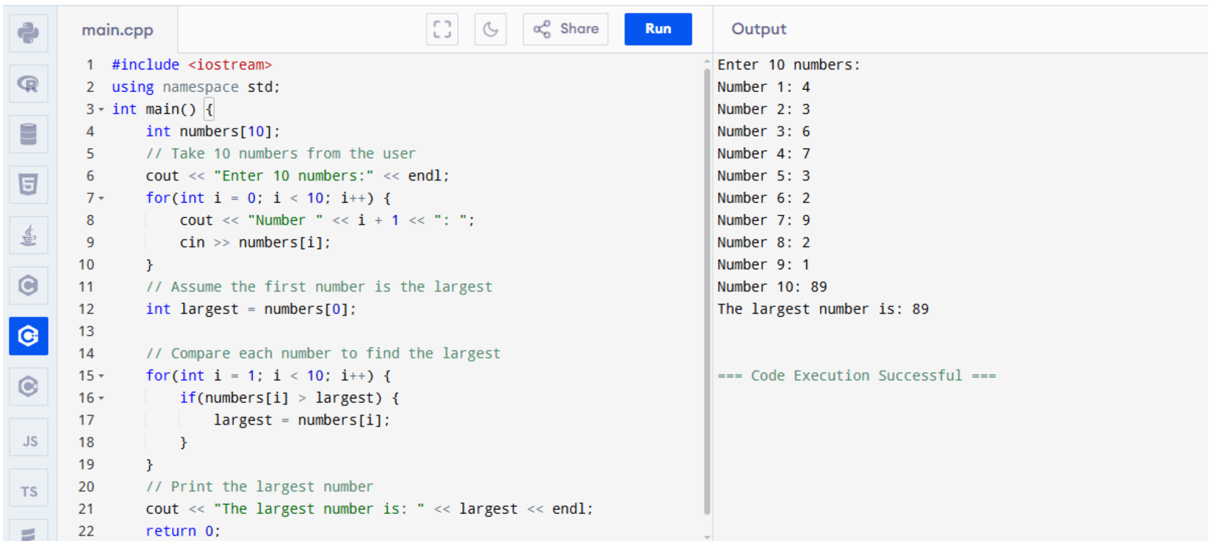
```
#include <iostream>
using namespace std;

int main() {
    int numbers[10];

    // Take 10 numbers from the user
    cout << "Enter 10 numbers:" << endl;
    for(int i = 0; i < 10; i++) {
        cout << "Number " << i + 1 << ": ";
        cin >> numbers[i];
    }

    // Assume the first number is the largest
    int largest = numbers[0];

    // Compare each number to find the largest
    for(int i = 1; i < 10; i++) {
        if(numbers[i] > largest) {
            largest = numbers[i];
        }
    }
```

// Print the largest number

cout << "The largest number is: " << largest << endl;

return 0;

}



```cpp
#include <iostream>
using namespace std;
int main() {
    int numbers[10];
    // Take 10 numbers from the user
    cout << "Enter 10 numbers:" << endl;
    for(int i = 0; i < 10; i++) {
        cout << "Number " << i + 1 << ": ";
        cin >> numbers[i];
    }
    // Assume the first number is the largest
    int largest = numbers[0];

    // Compare each number to find the largest
    for(int i = 1; i < 10; i++) {
        if(numbers[i] > largest) {
            largest = numbers[i];
        }
    }
    // Print the largest number
    cout << "The largest number is: " << largest << endl;
    return 0;
```

```
Enter 10 numbers:
Number 1: 4
Number 2: 3
Number 3: 6
Number 4: 7
Number 5: 3
Number 6: 2
Number 7: 9
Number 8: 2
Number 9: 1
Number 10: 89
The largest number is: 89


=== Code Execution Successful ===
```

Strings in C++

🧵 What is a String?

A string is a collection of characters (letters, numbers, symbols) that form words or sentences.

Example:

string name = "Usama";

🧪 1. Declaring a String

#include <iostream>

#include <string>

using namespace std;

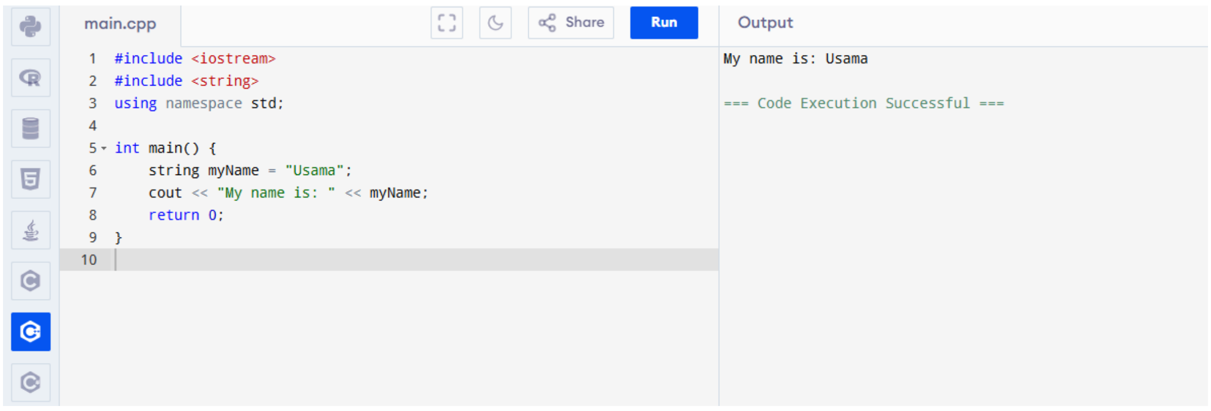
int main() {

    string myName = "Usama";

    cout << "My name is: " << myName;

    return 0;

}



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

int main() {
    string myName = "Usama";
    cout << "My name is: " << myName;
    return 0;
}
```

```
My name is: Usama

=== Code Execution Successful ===
```

🛠️ 2. Taking Input from User

🧍 Only one word:

string name;

cin >> name;

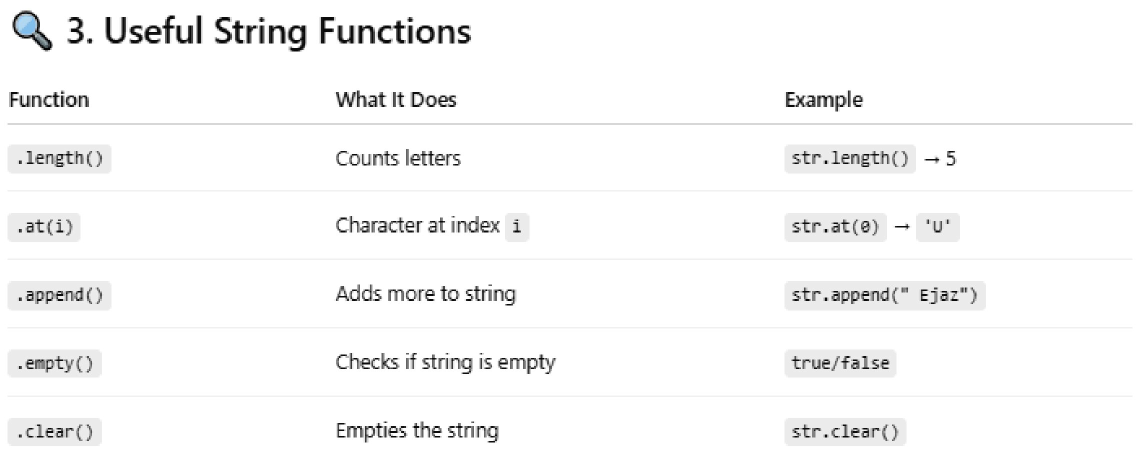
🧑‍🤝‍🧑 Full sentence (with spaces):

string sentence;

getline(cin, sentence);

🔍 3. Useful String Functions

Function	What It Does	Example
`.length()`	Counts letters	`str.length()` → 5
`.at(i)`	Character at index `i`	`str.at(0)` → `'U'`
`.append()`	Adds more to string	`str.append(" Ejaz")`
`.empty()`	Checks if string is empty	`true/false`
`.clear()`	Empties the string	`str.clear()`

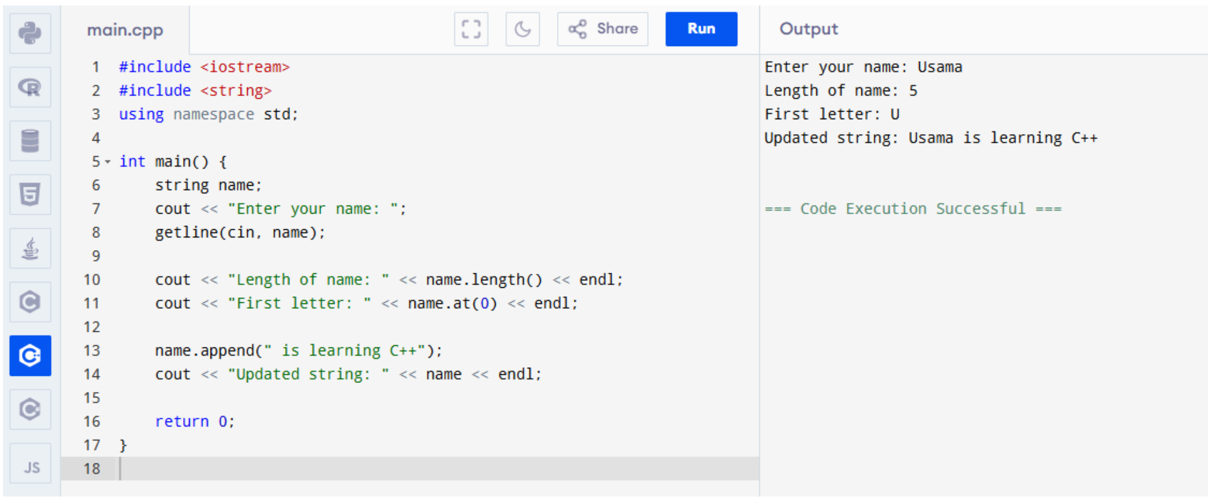


🔍 3. Useful String Functions

Function	What It Does	Example
`.length()`	Counts letters	`str.length()` → 5
`.at(i)`	Character at index `i`	`str.at(0)` → `'U'`
`.append()`	Adds more to string	`str.append(" Ejaz")`
`.empty()`	Checks if string is empty	`true/false`
`.clear()`	Empties the string	`str.clear()`

4. Example :

```
#include <iostream>

#include <string>

using namespace std;


int main() {

    string name;

    cout << "Enter your name: ";

    getline(cin, name);


    cout << "Length of name: " << name.length() << endl;

    cout << "First letter: " << name.at(0) << endl;


    name.append(" is learning C++");

    cout << "Updated string: " << name << endl;
    return 0;
}
```

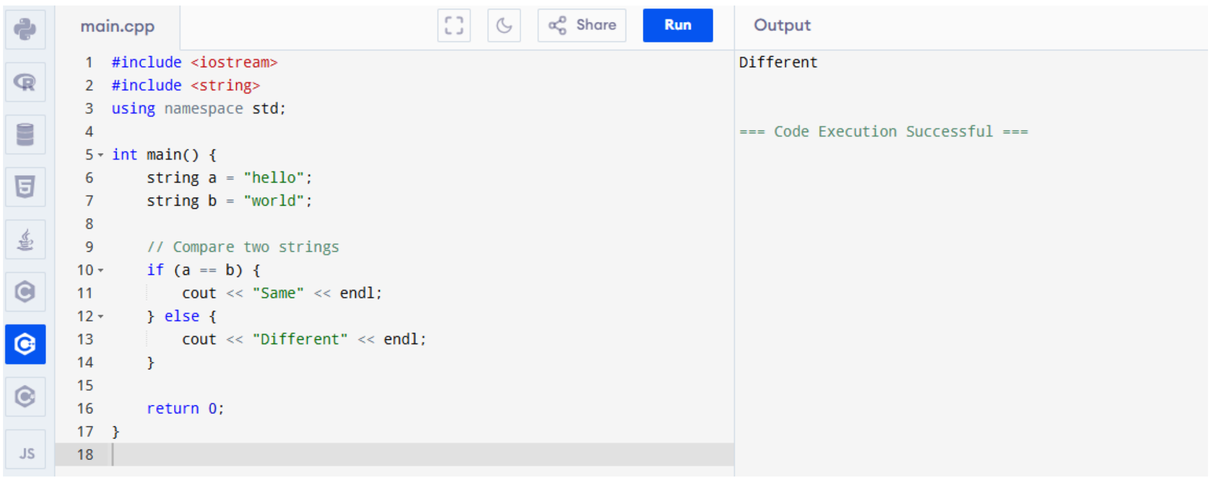


```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string name;
    cout << "Enter your name: ";
    getline(cin, name);

    cout << "Length of name: " << name.length() << endl;
    cout << "First letter: " << name.at(0) << endl;

    name.append(" is learning C++");
    cout << "Updated string: " << name << endl;

    return 0;
}
```

```
Enter your name: Usama
Length of name: 5
First letter: U
Updated string: Usama is learning C++


=== Code Execution Successful ===
```

5. Comparing Strings

```
string a = "hello";
string b = "world";

if (a == b) {
    cout << "Same";
} else {
    cout << "Different";
}
```



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

int main() {
    string a = "hello";
    string b = "world";

    // Compare two strings
    if (a == b) {
        cout << "Same" << endl;
    } else {
        cout << "Different" << endl;
    }

    return 0;
}
```

```
Different


=== Code Execution Successful ===
```

C++ Program (Char Array vs String)

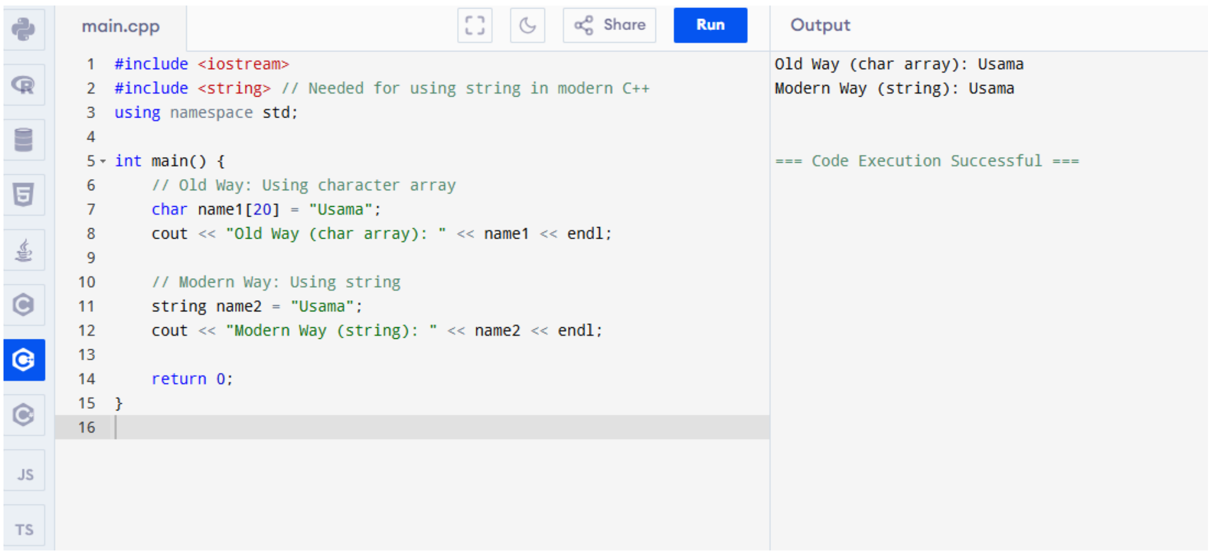
String as Char Array (Old Way)

char name[20] = "Usama";

cout << name;

But in modern C++, we prefer using `string` not `char array`.

```
#include <iostream>

#include <string> // Needed for using string in modern C++

using namespace std;


int main() {

    // Old Way: Using character array

    char name1[20] = "Usama";

    cout << "Old Way (char array): " << name1 << endl;


    // Modern Way: Using string

    string name2 = "Usama";

    cout << "Modern Way (string): " << name2 << endl;
    return 0;
}
```



```cpp
#include <iostream>
#include <string> // Needed for using string in modern C++
using namespace std;

int main() {
    // Old Way: Using character array
    char name1[20] = "Usama";
    cout << "Old Way (char array): " << name1 << endl;

    // Modern Way: Using string
    string name2 = "Usama";
    cout << "Modern Way (string): " << name2 << endl;

    return 0;
}
```

```
Old Way (char array): Usama
Modern Way (string): Usama


=== Code Execution Successful ===
```

📌 Difference Between Char Array and String:

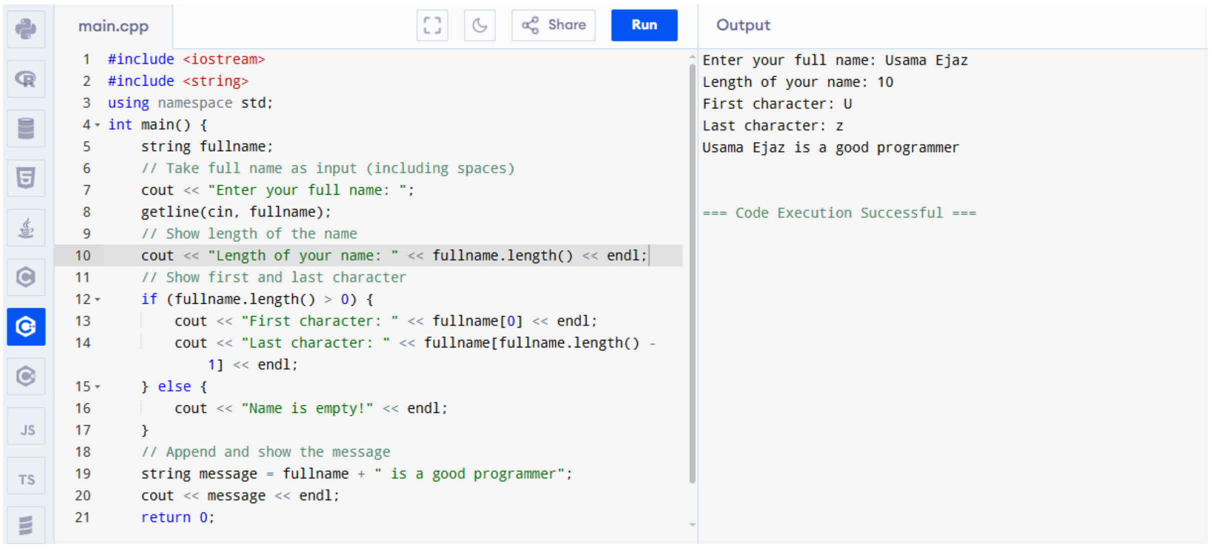
Feature	`char[]` (Old Way)	`string` (Modern C++ Way)
Header Needed	No special header needed	Requires `#include <string>`
Fixed Size	Must define size (e.g., `char name[20]`)	Automatically adjusts to content size
String Functions	Limited (`strcpy` , `strlen` , etc. via `<cstring>`)	Rich functions like `.length()` , `.substr()` , `.append()`
Safety	Risk of overflow	Safer and managed by C++ STL
Concatenation	Manual (`strcat`)	Easy using `+` operator
Readability & Ease	Verbose and manual memory handling	Clean and easy to use
Recommended for New Code	❌ No	✅ Yes

🧪 Task:

🔹 **Write a program that:**

- **Takes your full name as input**
- **Shows the length**
- **Shows the first and last character**
- **Appends `" is a good programmer"` to it**

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

int main() {
    string fullname;

    // Take full name as input (including spaces)
    cout << "Enter your full name: ";
    getline(cin, fullname);

    // Show length of the name
    cout << "Length of your name: " << fullname.length() << endl;

    // Show first and last character
    if (fullname.length() > 0) {
        cout << "First character: " << fullname[0] << endl;
        cout << "Last character: " << fullname[fullname.length() - 1] << endl;
    } else {
        cout << "Name is empty!" << endl;
    }

    // Append and show the message
    string message = fullname + " is a good programmer";
    cout << message << endl;

    return 0;
}
```



```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string fullname;
    // Take full name as input (including spaces)
    cout << "Enter your full name: ";
    getline(cin, fullname);
    // Show length of the name
    cout << "Length of your name: " << fullname.length() << endl;
    // Show first and last character
    if (fullname.length() > 0) {
        cout << "First character: " << fullname[0] << endl;
        cout << "Last character: " << fullname[fullname.length() - 1] << endl;
    } else {
        cout << "Name is empty!" << endl;
    }
    // Append and show the message
    string message = fullname + " is a good programmer";
    cout << message << endl;
    return 0;
```

```
Enter your full name: Usama Ejaz
Length of your name: 10
First character: U
Last character: z
Usama Ejaz is a good programmer


=== Code Execution Successful ===
```

🌟 Switch Statement in C++

🔘 What is Switch Statement?

Switch statement ek tarah ka decision-making tool hai jo multiple options mein se ek ko select karta hai.

🔎 Why use Switch?

Instead of writing multiple if-else, switch makes code cleaner and easier to read when there is a need to check multiple cases.

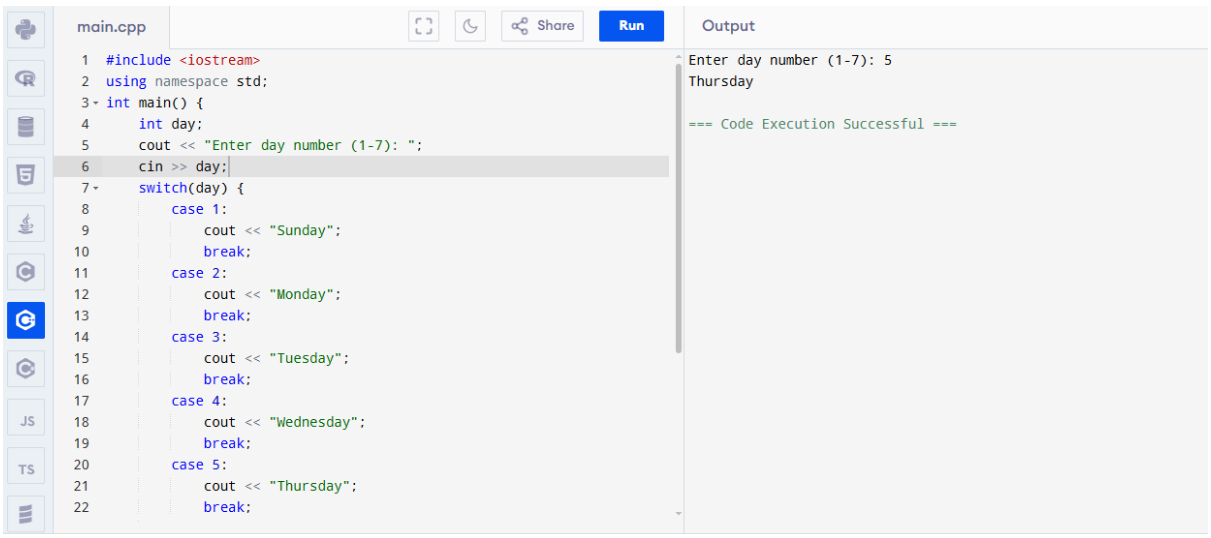
🧰 Syntax:

```
switch(expression) {
    case value1:
        // code if expression == value1
        break;
    case value2:
        // code if expression == value2
        break;
    ...
    default:
        // code if expression doesn't match any case
}
```

🧪 Example: Days of Week

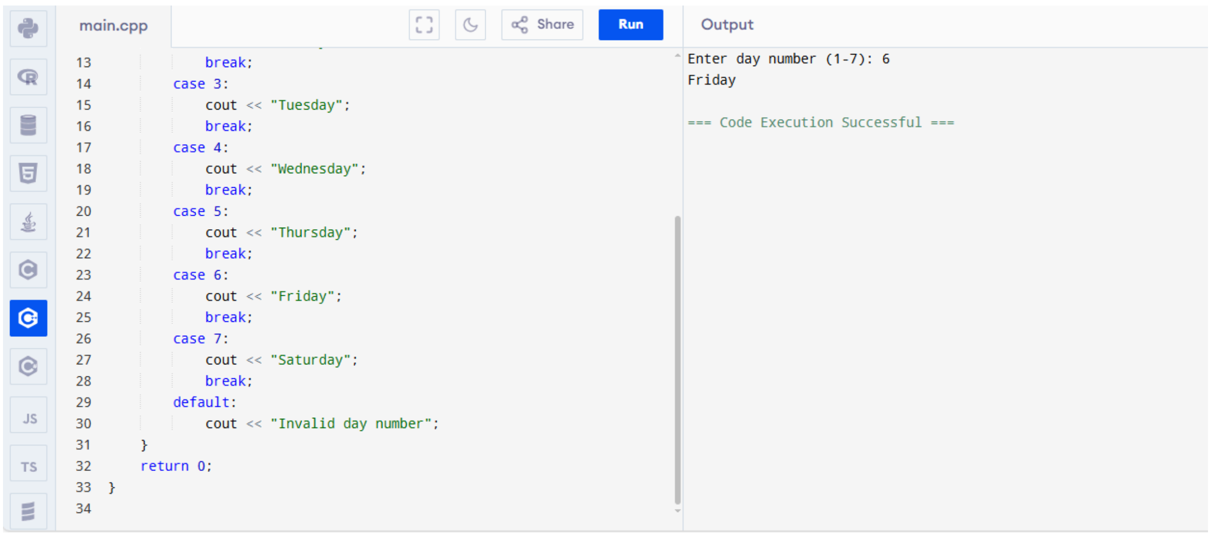
```
#include <iostream>
using namespace std;

int main() {
    int day;
    cout << "Enter day number (1-7): ";
    cin >> day;
```

```
switch(day) {
    case 1:
        cout << "Sunday";
        break;
    case 2:
        cout << "Monday";
        break;
    case 3:
        cout << "Tuesday";
        break;
    case 4:
        cout << "Wednesday";
        break;
    case 5:
        cout << "Thursday";
        break;
    case 6:
        cout << "Friday";
        break;
    case 7:
        cout << "Saturday";
        break;
    default:
        cout << "Invalid day number";
    }
    return 0;
}
```



```
#include <iostream>
using namespace std;
int main() {
    int day;
    cout << "Enter day number (1-7): ";
    cin >> day;
    switch(day) {
        case 1:
            cout << "Sunday";
            break;
        case 2:
            cout << "Monday";
            break;
        case 3:
            cout << "Tuesday";
            break;
        case 4:
            cout << "Wednesday";
            break;
        case 5:
            cout << "Thursday";
            break;
```

```
Enter day number (1-7): 5
Thursday

=== Code Execution Successful ===
```



```
            break;
        case 3:
            cout << "Tuesday";
            break;
        case 4:
            cout << "Wednesday";
            break;
        case 5:
            cout << "Thursday";
            break;
        case 6:
            cout << "Friday";
            break;
        case 7:
            cout << "Saturday";
            break;
        default:
            cout << "Invalid day number";
    }
    return 0;
}
```

```
Enter day number (1-7): 6
Friday

=== Code Execution Successful ===
```

🔑 Important Points:

- ☐ The `break` statement is written at the end of each case; otherwise, the program will continue to the next case (fall-through).

☐ The `default` case is optional, but it's useful when none of the cases match.

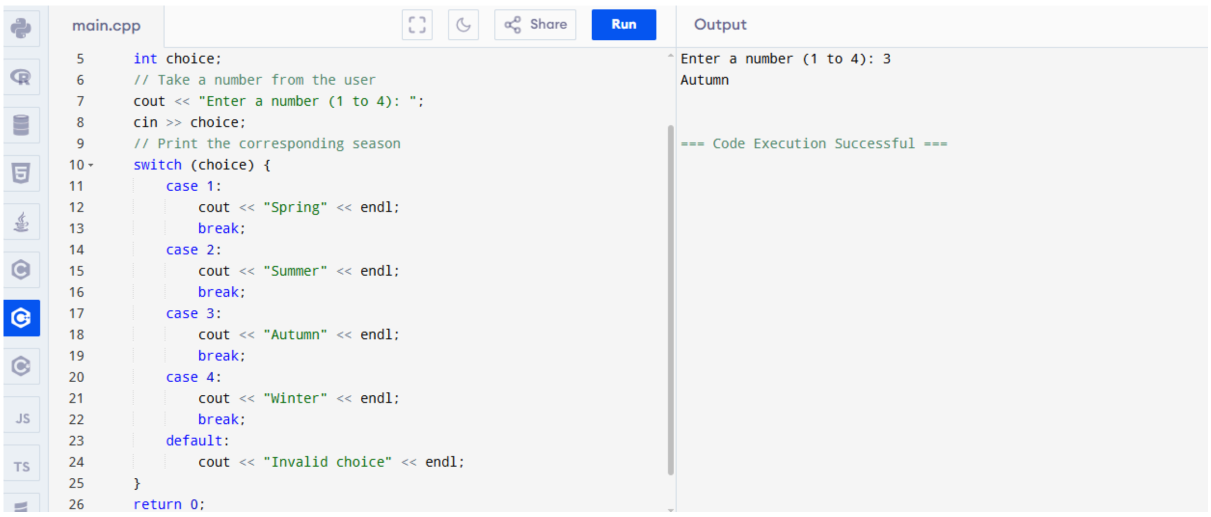
☐ The switch expression can only be of type int, char, or enum (not floating-point types like `float` or `double`).

🧪 Task:

Write a program that:

- **Takes a number from 1 to 4**
- **Prints the corresponding season:**
  1 = Spring
  2 = Summer
  3 = Autumn
  4 = Winter
- **Prints "Invalid choice" otherwise**

```
#include <iostream>

using namespace std;


int main() {

    int choice;

    // Take a number from the user

    cout << "Enter a number (1 to 4): ";

    cin >> choice;

    // Print the corresponding season

    switch (choice) {
        case 1:
            cout << "Spring" << endl;
            break;
        case 2:
            cout << "Summer" << endl;
            break;
        case 3:
            cout << "Autumn" << endl;
            break;
        case 4:
            cout << "Winter" << endl;
            break;
        default:
            cout << "Invalid choice" << endl;
    }
    return 0;
}
```



```cpp
    int choice;
    // Take a number from the user
    cout << "Enter a number (1 to 4): ";
    cin >> choice;
    // Print the corresponding season
    switch (choice) {
        case 1:
            cout << "Spring" << endl;
            break;
        case 2:
            cout << "Summer" << endl;
            break;
        case 3:
            cout << "Autumn" << endl;
            break;
        case 4:
            cout << "Winter" << endl;
            break;
        default:
            cout << "Invalid choice" << endl;
    }
    return 0;
```

```
Enter a number (1 to 4): 3
Autumn


=== Code Execution Successful ===
```

Pointers in C++

📌 What is a Pointer?

A pointer is a variable that stores the memory address of another variable.

Example:

Imagine a house.

- ☐ The house = variable
- ☐ The address of the house = pointer

💡 Syntax:

int a = 10;

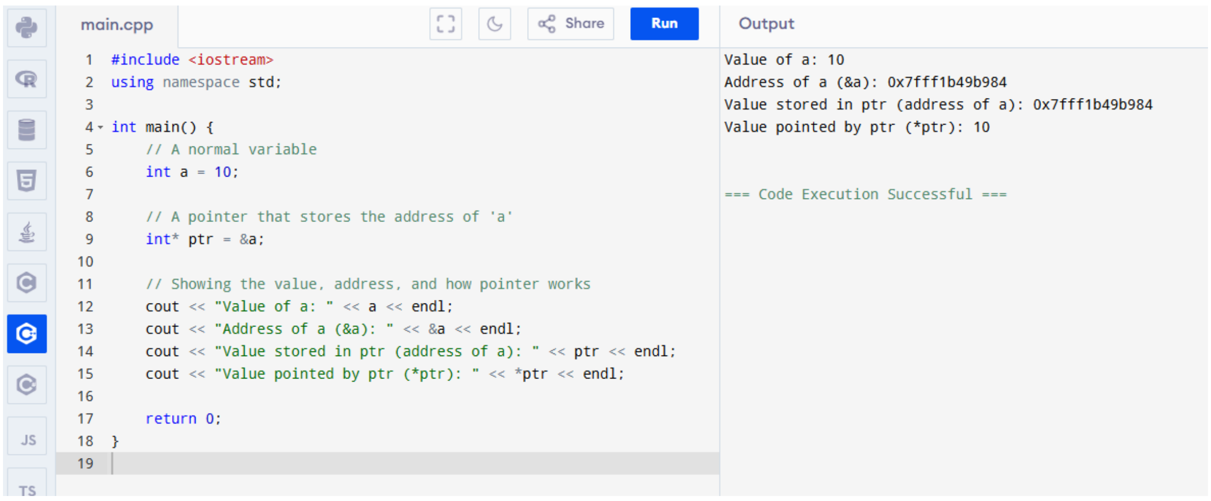
int* ptr = &a;

- ☐ a is a normal variable
- ☐ &a gives the address of a

- ☐ `ptr` stores that address

C++ Program: Understanding Pointers

```
#include <iostream>

using namespace std;


int main() {

    // A normal variable

    int a = 10;


    // A pointer that stores the address of 'a'

    int* ptr = &a;


    // Showing the value, address, and how pointer works

    cout << "Value of a: " << a << endl;

    cout << "Address of a (&a): " << &a << endl;

    cout << "Value stored in ptr (address of a): " << ptr << endl;

    cout << "Value pointed by ptr (*ptr): " << *ptr << endl;


    return 0;

}
```

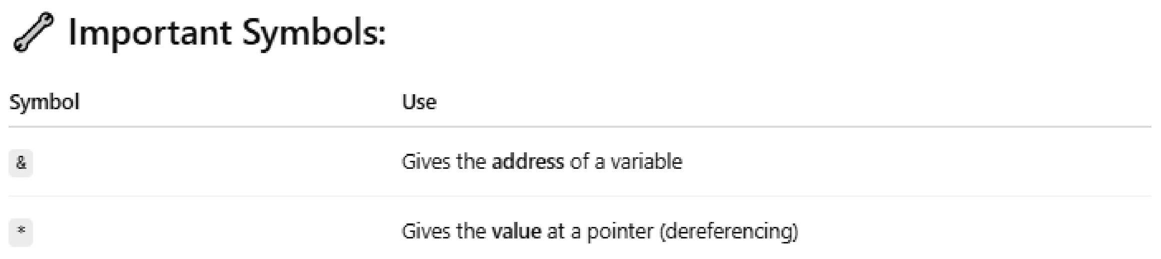


```cpp
#include <iostream>
using namespace std;

int main() {
    // A normal variable
    int a = 10;

    // A pointer that stores the address of 'a'
    int* ptr = &a;

    // Showing the value, address, and how pointer works
    cout << "Value of a: " << a << endl;
    cout << "Address of a (&a): " << &a << endl;
    cout << "Value stored in ptr (address of a): " << ptr << endl;
    cout << "Value pointed by ptr (*ptr): " << *ptr << endl;

    return 0;
}
```

```
Value of a: 10
Address of a (&a): 0x7fff1b49b984
Value stored in ptr (address of a): 0x7fff1b49b984
Value pointed by ptr (*ptr): 10


=== Code Execution Successful ===
```

🔧 Important Symbols:

Symbol	Use
&	Gives the address of a variable
*	Gives the value at a pointer (dereferencing)

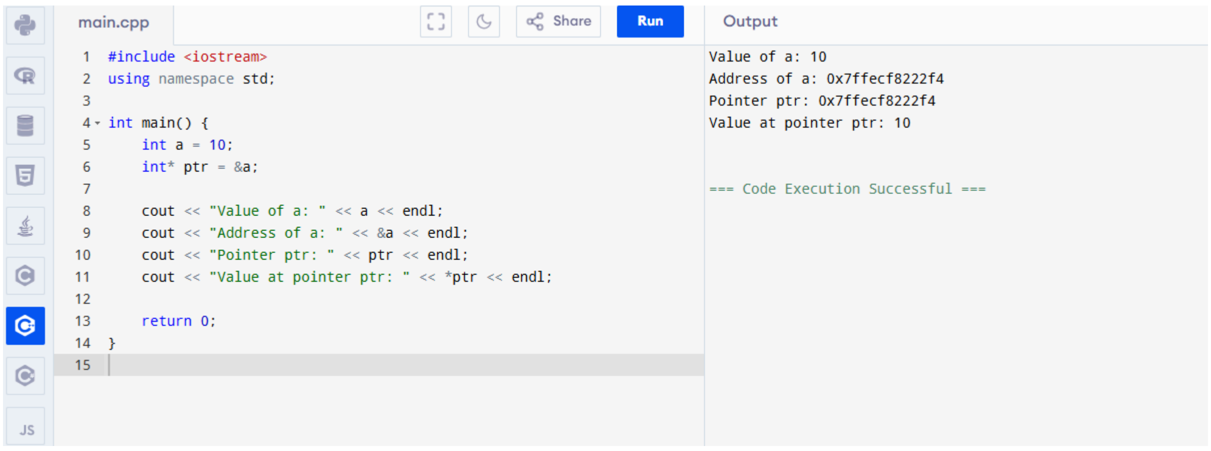


🔧 Important Symbols:

Symbol	Use
`&`	Gives the **address** of a variable
`*`	Gives the **value** at a pointer (dereferencing)

Example:

```
#include <iostream>

using namespace std;


int main() {

   int a = 10;

   int* ptr = &a;


   cout << "Value of a: " << a << endl;

   cout << "Address of a: " << &a << endl;

   cout << "Pointer ptr: " << ptr << endl;

   cout << "Value at pointer ptr: " << *ptr << endl;


   return 0;

}
```



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

int main() {
    int a = 10;
    int* ptr = &a;

    cout << "Value of a: " << a << endl;
    cout << "Address of a: " << &a << endl;
    cout << "Pointer ptr: " << ptr << endl;
    cout << "Value at pointer ptr: " << *ptr << endl;

    return 0;
}
```

```
Value of a: 10
Address of a: 0x7ffecf8222f4
Pointer ptr: 0x7ffecf8222f4
Value at pointer ptr: 10

=== Code Execution Successful ===
```

Why Do We Use Pointers?

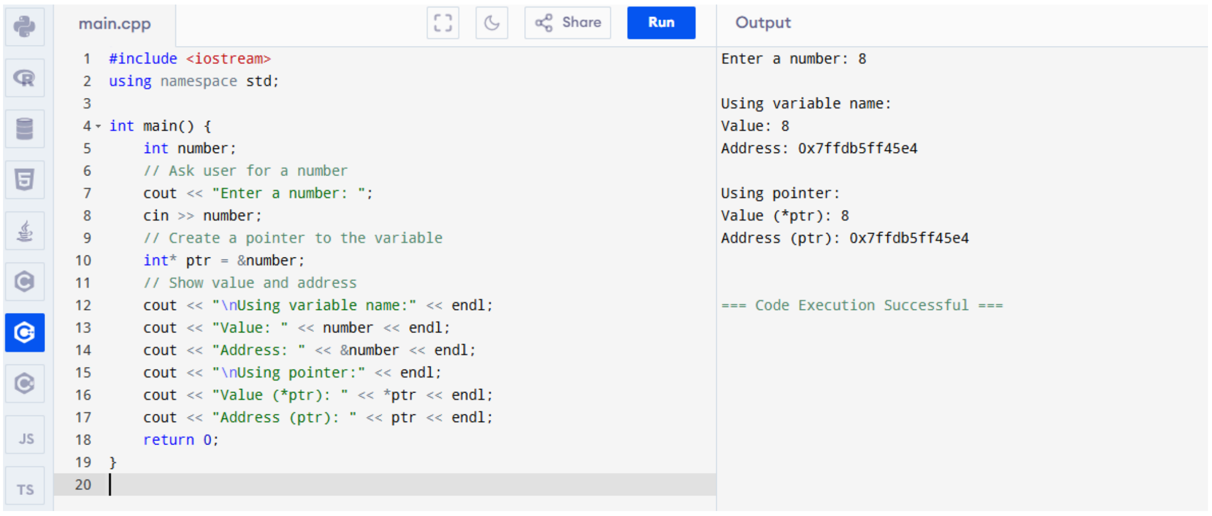
- ☐ To save memory
- ☐ To access and change values directly using address
- ☐ Used in arrays, functions, and data structures (like linked list)

🧪 Task :

1. Ask user for a number
2. Store the number in a variable
3. Create a pointer to that variable
4. Print the value and address using both & and *

C++ Program: Pointer Basics with User Input

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

int main() {
    int number;

    // Ask user for a number
    cout << "Enter a number: ";
    cin >> number;

    // Create a pointer to the variable
    int* ptr = &number;

    // Show value and address
    cout << "\nUsing variable name:" << endl;
    cout << "Value: " << number << endl;
    cout << "Address: " << &number << endl;

    cout << "\nUsing pointer:" << endl;
    cout << "Value (*ptr): " << *ptr << endl;
    cout << "Address (ptr): " << ptr << endl;

    return 0;
}
```



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

int main() {
    int number;
    // Ask user for a number
    cout << "Enter a number: ";
    cin >> number;
    // Create a pointer to the variable
    int* ptr = &number;
    // Show value and address
    cout << "\nUsing variable name:" << endl;
    cout << "Value: " << number << endl;
    cout << "Address: " << &number << endl;
    cout << "\nUsing pointer:" << endl;
    cout << "Value (*ptr): " << *ptr << endl;
    cout << "Address (ptr): " << ptr << endl;
    return 0;
}
```

```
Enter a number: 8

Using variable name:
Value: 8
Address: 0x7ffdb5ff45e4

Using pointer:
Value (*ptr): 8
Address (ptr): 0x7ffdb5ff45e4


=== Code Execution Successful ===
```

Pointer to Pointer concept in C++:

🌟 What is a Pointer to Pointer?

Just like a normal pointer stores the address of a variable,
a Pointer to Pointer stores the address of a pointer.

It's like this:

int a = 10;

int* p = &a;   // p stores address of a

int** pp = &p;   // pp stores address of p

Real-Life Analogy:

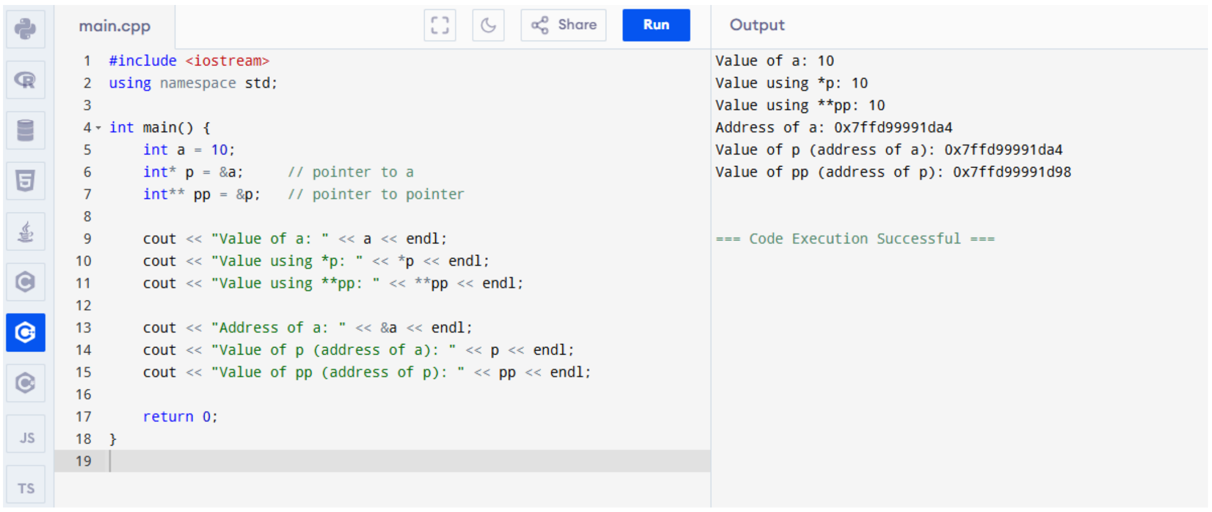
📦 a = a gift
🏠 p = address of the gift
🗺️ pp = map where that address is written

Example:

#include <iostream>

using namespace std;

```
int main() {

    int a = 10;

    int* p = &a;      // pointer to a

    int** pp = &p;   // pointer to pointer


    cout << "Value of a: " << a << endl;

    cout << "Value using *p: " << *p << endl;

    cout << "Value using **pp: " << **pp << endl;


    cout << "Address of a: " << &a << endl;

    cout << "Value of p (address of a): " << p << endl;

    cout << "Value of pp (address of p): " << pp << endl;


    return 0;

}
```



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

int main() {
    int a = 10;
    int* p = &a;      // pointer to a
    int** pp = &p;    // pointer to pointer

    cout << "Value of a: " << a << endl;
    cout << "Value using *p: " << *p << endl;
    cout << "Value using **pp: " << **pp << endl;

    cout << "Address of a: " << &a << endl;
    cout << "Value of p (address of a): " << p << endl;
    cout << "Value of pp (address of p): " << pp << endl;

    return 0;
}

```

```
Value of a: 10
Value using *p: 10
Value using **pp: 10
Address of a: 0x7ffd99991da4
Value of p (address of a): 0x7ffd99991da4
Value of pp (address of p): 0x7ffd99991d98


=== Code Execution Successful ===
```

📌 Why use Pointer to Pointer?

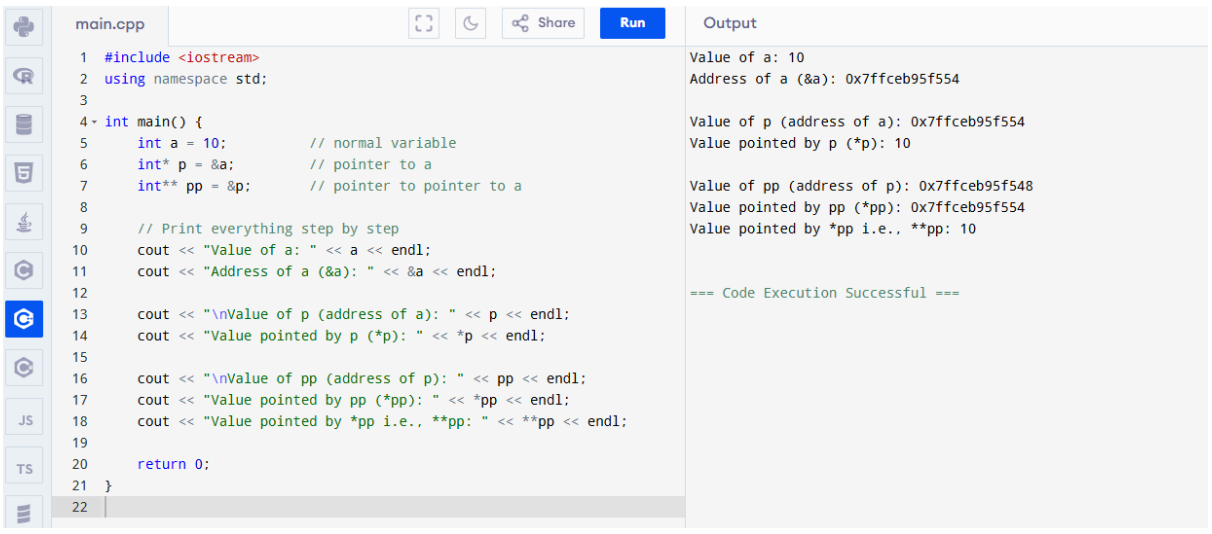
- ☐ Used in multidimensional arrays
- ☐ Needed when passing a pointer to a function (by reference)
- ☐ Used in dynamic memory allocation and advanced data structures

Pointer Understanding flow diagram:

```
a   = 10
p   = &a   → points to a
pp  = &p   → points to p

*pp   = p (address of a)
**pp  = a (actual value)
```

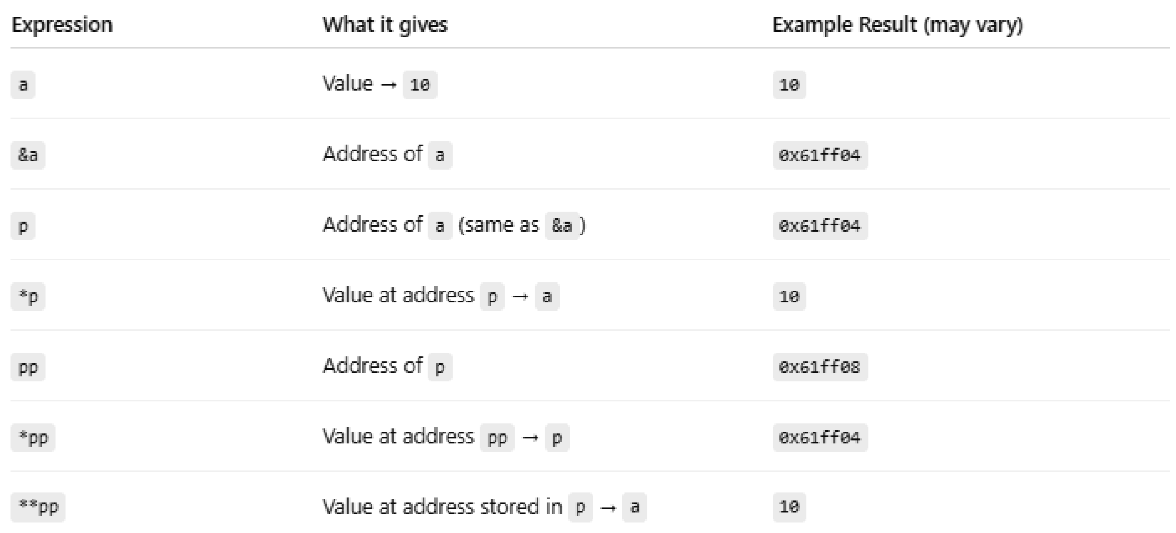
C++ Program: Pointer to Pointer Concept

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

int main() {
    int a = 10;          // normal variable
    int* p = &a;         // pointer to a
    int** pp = &p;       // pointer to pointer to a
    // Print everything step by step
    cout << "Value of a: " << a << endl;
    cout << "Address of a (&a): " << &a << endl;

    cout << "\nValue of p (address of a): " << p << endl;
    cout << "Value pointed by p (*p): " << *p << endl;

    cout << "\nValue of pp (address of p): " << pp << endl;
    cout << "Value pointed by pp (*pp): " << *pp << endl;
    cout << "Value pointed by *pp i.e., **pp: " << **pp << endl;

    return 0;
}
```



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

int main() {
    int a = 10;          // normal variable
    int* p = &a;         // pointer to a
    int** pp = &p;       // pointer to pointer to a

    // Print everything step by step
    cout << "Value of a: " << a << endl;
    cout << "Address of a (&a): " << &a << endl;

    cout << "\nValue of p (address of a): " << p << endl;
    cout << "Value pointed by p (*p): " << *p << endl;

    cout << "\nValue of pp (address of p): " << pp << endl;
    cout << "Value pointed by pp (*pp): " << *pp << endl;
    cout << "Value pointed by *pp i.e., **pp: " << **pp << endl;

    return 0;
}
```

```
Value of a: 10
Address of a (&a): 0x7ffceb95f554

Value of p (address of a): 0x7ffceb95f554
Value pointed by p (*p): 10

Value of pp (address of p): 0x7ffceb95f548
Value pointed by pp (*pp): 0x7ffceb95f554
Value pointed by *pp i.e., **pp: 10


=== Code Execution Successful ===
```



Expression	What it gives	Example Result (may vary)
`a`	Value → `10`	`10`
`&a`	Address of `a`	`0x61ff04`
`p`	Address of `a` (same as `&a`)	`0x61ff04`
`*p`	Value at address `p` → `a`	`10`
`pp`	Address of `p`	`0x61ff08`
`*pp`	Value at address `pp` → `p`	`0x61ff04`
`**pp`	Value at address stored in `p` → `a`	`10`

Structures (`struct`) in C++

📘 What is a Structure (struct)?

A structure is a user-defined data type.
 It lets you group different types of variables together under one name.

> Think of it like a custom box 📦 where you keep related items together.

Syntax of Structure:

struct Person {

   string name;

   int age;

   float height;

};

Now you can create **objects** of this structure:

Person p1;

Example:

You want to store data of a student:

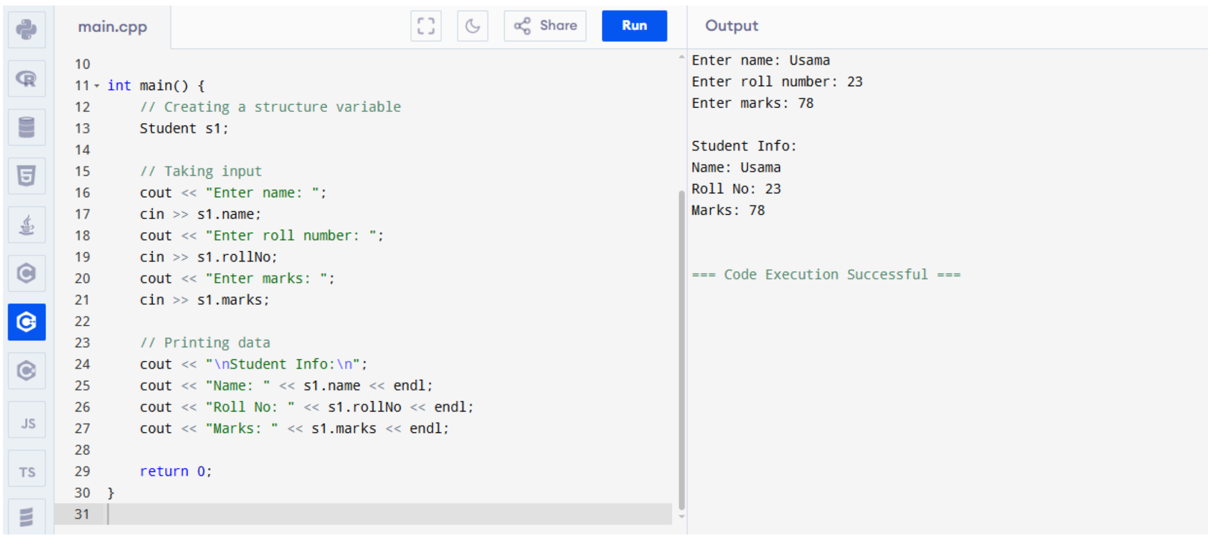
- Name (text)
- Roll number (number)
- Marks (decimal)

Instead of making 3 separate variables, make a structure:

```
struct Student {
    string name;
    int rollNo;
    float marks;
};
```

Example:

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

// Structure definition
struct Student {
    string name;
    int rollNo;
    float marks;
};

int main() {
    // Creating a structure variable
    Student s1;

    // Taking input
    cout << "Enter name: ";
    cin >> s1.name;
    cout << "Enter roll number: ";
    cin >> s1.rollNo;
    cout << "Enter marks: ";
    cin >> s1.marks;

    // Printing data
    cout << "\nStudent Info:\n";
    cout << "Name: " << s1.name << endl;
    cout << "Roll No: " << s1.rollNo << endl;
    cout << "Marks: " << s1.marks << endl;

    return 0;
}
```



```cpp

int main() {
    // Creating a structure variable
    Student s1;

    // Taking input
    cout << "Enter name: ";
    cin >> s1.name;
    cout << "Enter roll number: ";
    cin >> s1.rollNo;
    cout << "Enter marks: ";
    cin >> s1.marks;

    // Printing data
    cout << "\nStudent Info:\n";
    cout << "Name: " << s1.name << endl;
    cout << "Roll No: " << s1.rollNo << endl;
    cout << "Marks: " << s1.marks << endl;

    return 0;
}

```

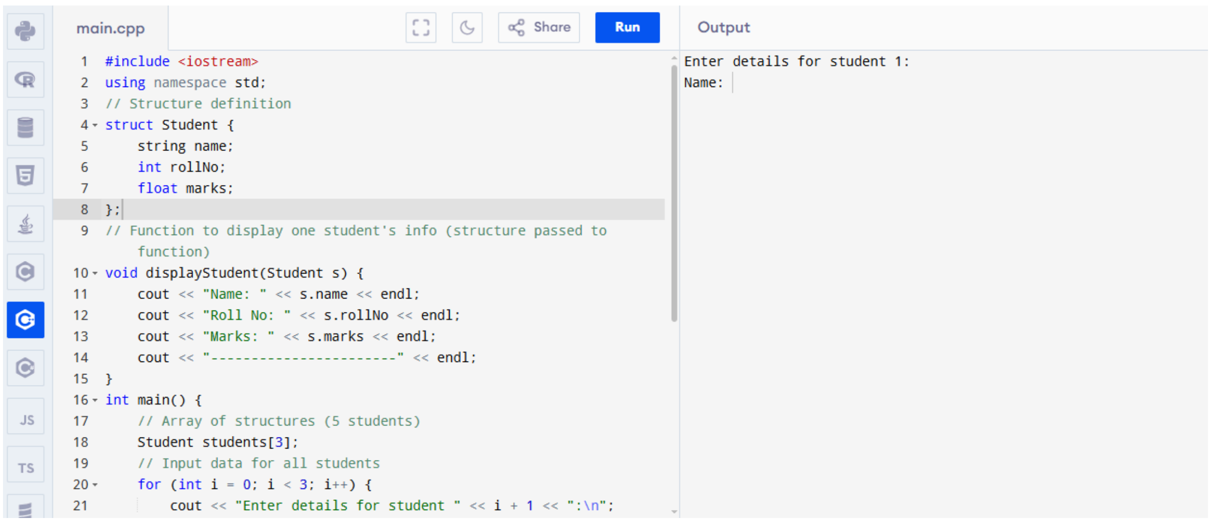
```
Enter name: Usama
Enter roll number: 23
Enter marks: 78

Student Info:
Name: Usama
Roll No: 23
Marks: 78


=== Code Execution Successful ===
```

🧪 Program: Handle Multiple Students Using `struct`

- ☐ Multiple students using structure
- ☐ Structure array
- ☐ Passing structure to function

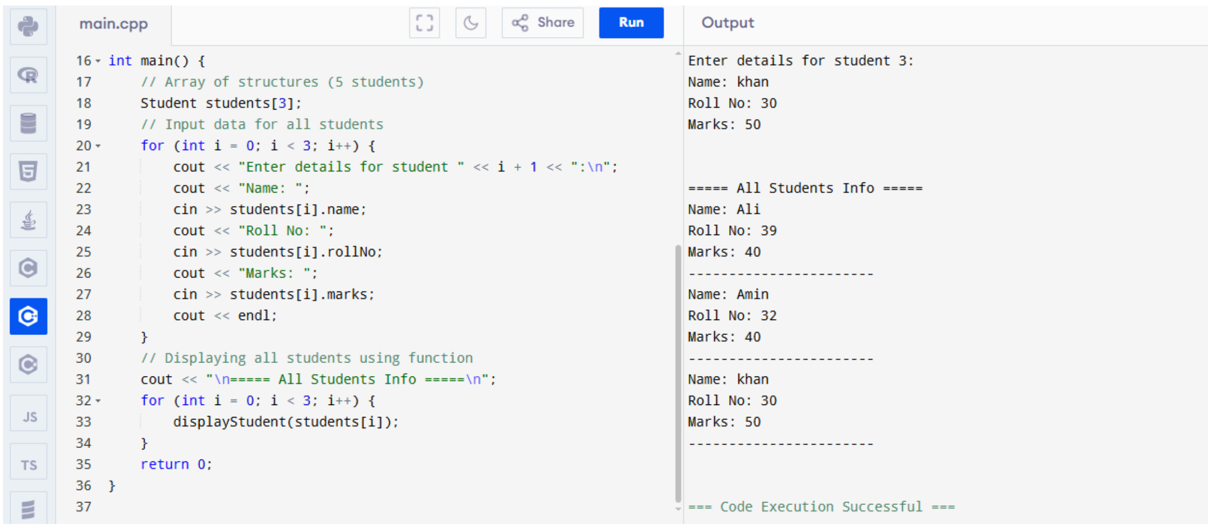
```cpp
#include <iostream>

using namespace std;


// Structure definition

struct Student {

    string name;

    int rollNo;

    float marks;

};
```

```
// Function to display one student's info (structure passed to function)
void displayStudent(Student s) {
    cout << "Name: " << s.name << endl;
    cout << "Roll No: " << s.rollNo << endl;
    cout << "Marks: " << s.marks << endl;
    cout << "-----------------------" << endl;
}

int main() {
    // Array of structures (5 students)
    Student students[3];

    // Input data for all students
    for (int i = 0; i < 3; i++) {
        cout << "Enter details for student " << i + 1 << ":\n";
        cout << "Name: ";
        cin >> students[i].name;
        cout << "Roll No: ";
        cin >> students[i].rollNo;
        cout << "Marks: ";
        cin >> students[i].marks;
        cout << endl;
    }
    // Displaying all students using function
    cout << "\n===== All Students Info =====\n";
    for (int i = 0; i < 3; i++) {
        displayStudent(students[i]);
    }

    return 0;
}
```



```
#include <iostream>
using namespace std;
// Structure definition
struct Student {
    string name;
    int rollNo;
    float marks;
};
// Function to display one student's info (structure passed to
    function)
void displayStudent(Student s) {
    cout << "Name: " << s.name << endl;
    cout << "Roll No: " << s.rollNo << endl;
    cout << "Marks: " << s.marks << endl;
    cout << "----------------------" << endl;
}
int main() {
    // Array of structures (5 students)
    Student students[3];
    // Input data for all students
    for (int i = 0; i < 3; i++) {
        cout << "Enter details for student " << i + 1 << ":\n";
```

```
Enter details for student 1:
Name:
```

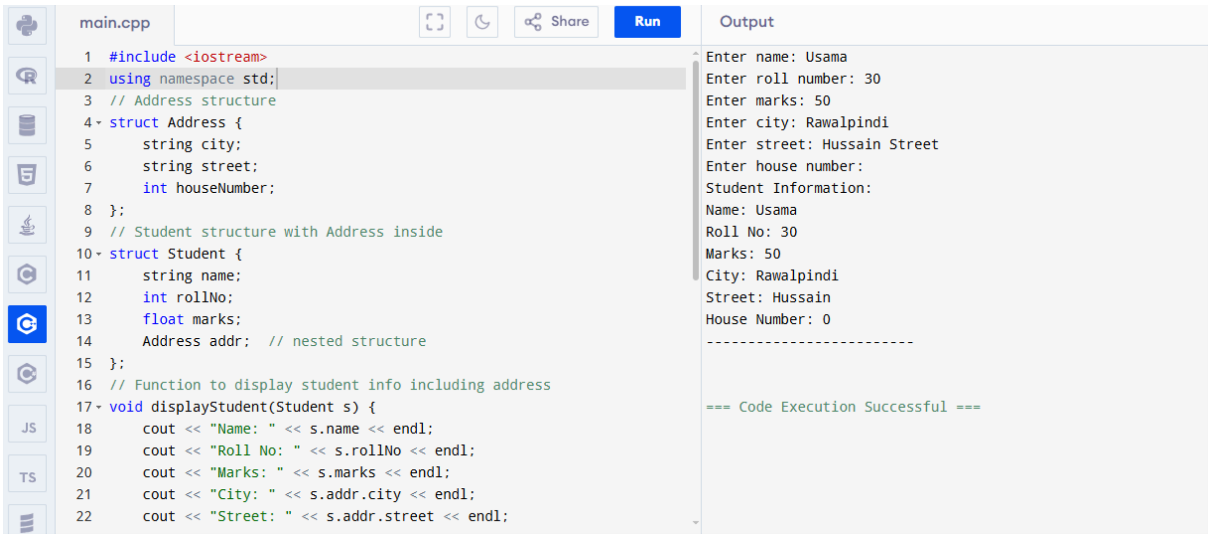


```
int main() {
    // Array of structures (5 students)
    Student students[3];
    // Input data for all students
    for (int i = 0; i < 3; i++) {
        cout << "Enter details for student " << i + 1 << ":\n";
        cout << "Name: ";
        cin >> students[i].name;
        cout << "Roll No: ";
        cin >> students[i].rollNo;
        cout << "Marks: ";
        cin >> students[i].marks;
        cout << endl;
    }
    // Displaying all students using function
    cout << "\n===== All Students Info =====\n";
    for (int i = 0; i < 3; i++) {
        displayStudent(students[i]);
    }
    return 0;
}
```

```
Enter details for student 3:
Name: khan
Roll No: 30
Marks: 50


===== All Students Info =====
Name: Ali
Roll No: 39
Marks: 40
----------------------
Name: Amin
Roll No: 32
Marks: 40
----------------------
Name: khan
Roll No: 30
Marks: 50
----------------------


=== Code Execution Successful ===
```

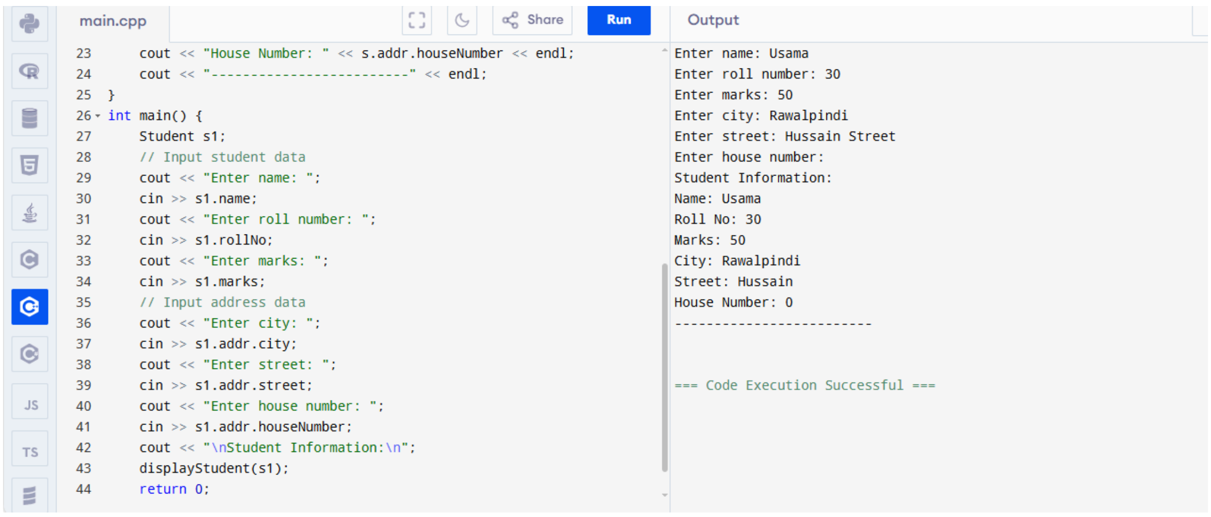
🧩 Nested Structures Example (Student with Address)

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

// Address structure
struct Address {
    string city;
    string street;
    int houseNumber;
};

// Student structure with Address inside
struct Student {
    string name;
    int rollNo;
    float marks;
    Address addr;  // nested structure
};

// Function to display student info including address
void displayStudent(Student s) {
    cout << "Name: " << s.name << endl;
    cout << "Roll No: " << s.rollNo << endl;
    cout << "Marks: " << s.marks << endl;
    cout << "City: " << s.addr.city << endl;

    cout << "Street: " << s.addr.street << endl;

    cout << "House Number: " << s.addr.houseNumber << endl;

    cout << "------------------------" << endl;

}


int main() {

    Student s1;


    // Input student data

    cout << "Enter name: ";

    cin >> s1.name;

    cout << "Enter roll number: ";

    cin >> s1.rollNo;

    cout << "Enter marks: ";

    cin >> s1.marks;


    // Input address data

    cout << "Enter city: ";

    cin >> s1.addr.city;

    cout << "Enter street: ";

    cin >> s1.addr.street;

    cout << "Enter house number: ";

    cin >> s1.addr.houseNumber;
    cout << "\nStudent Information:\n";

    displayStudent(s1);


    return 0;

}
```



```
#include <iostream>
using namespace std;
// Address structure
struct Address {
    string city;
    string street;
    int houseNumber;
};
// Student structure with Address inside
struct Student {
    string name;
    int rollNo;
    float marks;
    Address addr;  // nested structure
};
// Function to display student info including address
void displayStudent(Student s) {
    cout << "Name: " << s.name << endl;
    cout << "Roll No: " << s.rollNo << endl;
    cout << "Marks: " << s.marks << endl;
    cout << "City: " << s.addr.city << endl;
    cout << "Street: " << s.addr.street << endl;
```

```
Enter name: Usama
Enter roll number: 30
Enter marks: 50
Enter city: Rawalpindi
Enter street: Hussain Street
Enter house number:
Student Information:
Name: Usama
Roll No: 30
Marks: 50
City: Rawalpindi
Street: Hussain
House Number: 0
------------------------


=== Code Execution Successful ===
```



```
    cout << "House Number: " << s.addr.houseNumber << endl;
    cout << "------------------------" << endl;
}
int main() {
    Student s1;
    // Input student data
    cout << "Enter name: ";
    cin >> s1.name;
    cout << "Enter roll number: ";
    cin >> s1.rollNo;
    cout << "Enter marks: ";
    cin >> s1.marks;
    // Input address data
    cout << "Enter city: ";
    cin >> s1.addr.city;
    cout << "Enter street: ";
    cin >> s1.addr.street;
    cout << "Enter house number: ";
    cin >> s1.addr.houseNumber;
    cout << "\nStudent Information:\n";
    displayStudent(s1);
    return 0;
```

```
Enter name: Usama
Enter roll number: 30
Enter marks: 50
Enter city: Rawalpindi
Enter street: Hussain Street
Enter house number:
Student Information:
Name: Usama
Roll No: 30
Marks: 50
City: Rawalpindi
Street: Hussain
House Number: 0
------------------------


=== Code Execution Successful ===
```

File Handling in C++

📂 What is File Handling?

File Handling means reading data from files and writing data to files using your program.

This helps to save data permanently, even after the program ends.

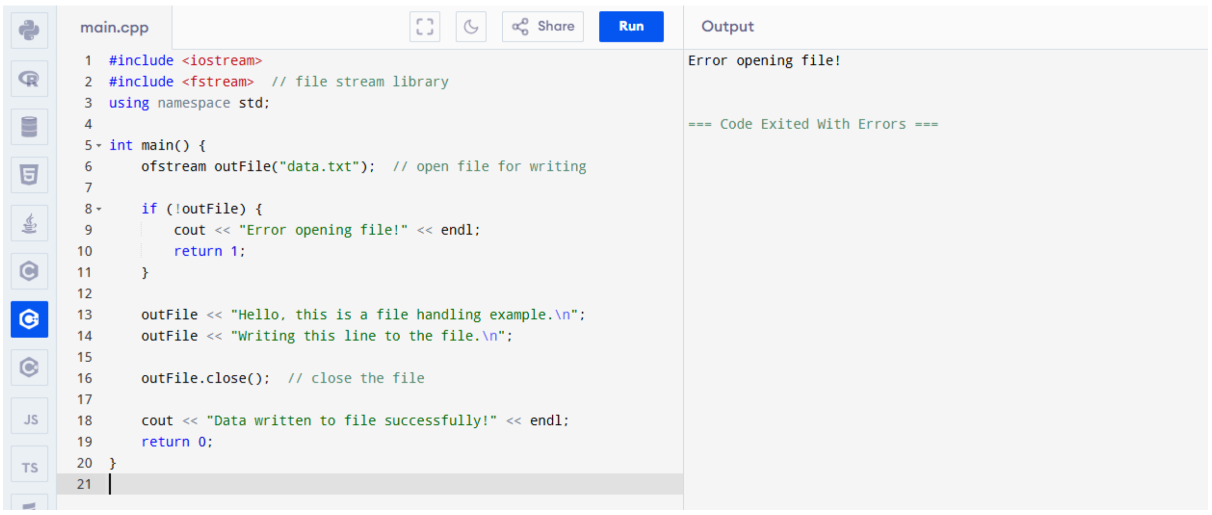
🛠️ Basics of File Handling in C++

C++ provides a library called `<fstream>` to work with files.
There are 3 main classes:

- ☐ ofstream — to write data to a file
- ☐ ifstream — to read data from a file
- ☐ fstream — for both reading and writing

Step 1: Writing to a File

```
#include <iostream>

#include <fstream>  // file stream library

using namespace std;


int main() {

    ofstream outFile("data.txt");  // open file for writing


    if (!outFile) {

        cout << "Error opening file!" << endl;
        return 1;
    }

    outFile << "Hello, this is a file handling example.\n";
    outFile << "Writing this line to the file.\n";

    outFile.close();  // close the file

    cout << "Data written to file successfully!" << endl;

    return 0;
}
```



```
#include <iostream>
#include <fstream>  // file stream library
using namespace std;

int main() {
    ofstream outFile("data.txt");  // open file for writing

    if (!outFile) {
        cout << "Error opening file!" << endl;
        return 1;
    }

    outFile << "Hello, this is a file handling example.\n";
    outFile << "Writing this line to the file.\n";

    outFile.close();  // close the file

    cout << "Data written to file successfully!" << endl;
    return 0;
}
```

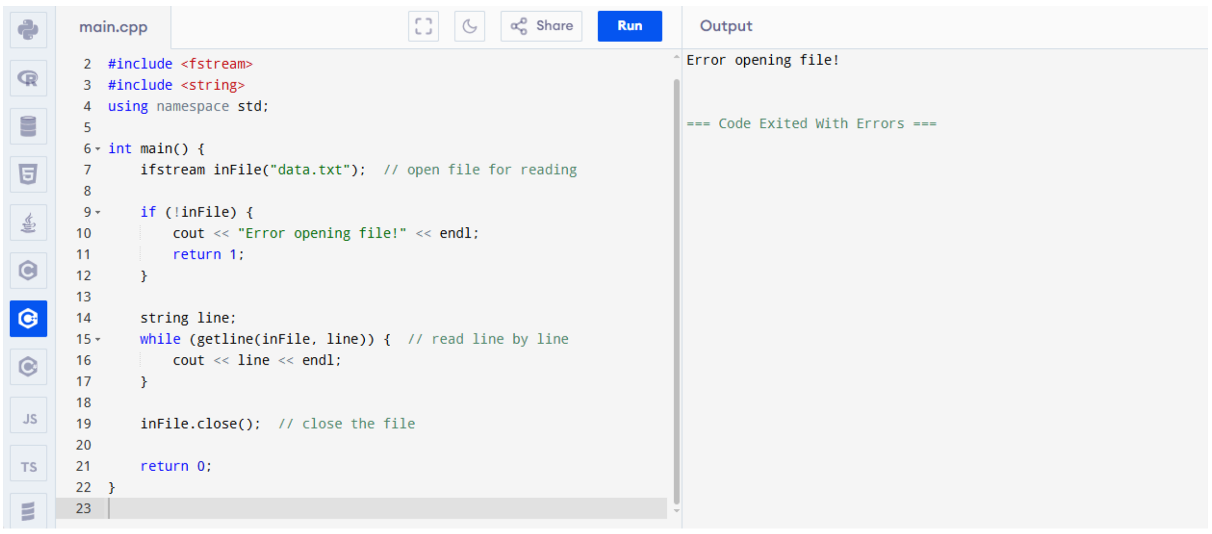
```
Error opening file!


=== Code Exited With Errors ===
```

🖥️ What happens here?

- ☐ `ofstream outFile("data.txt");` opens (or creates) a file named **data.txt** for writing
- ☐ `outFile << "text"` writes the text to the file
- ☐ `outFile.close();` closes the file to save it properly

Step 2: Reading from a File

```
#include <iostream>

#include <fstream>

#include <string>

using namespace std;


int main() {

    ifstream inFile("data.txt");  // open file for reading


    if (!inFile) {

        cout << "Error opening file!" << endl;

        return 1;

    }


    string line;

    while (getline(inFile, line)) {  // read line by line

        cout << line << endl;

    }
    inFile.close();  // close the file


    return 0;
}
```



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

int main() {
    ifstream inFile("data.txt");  // open file for reading

    if (!inFile) {
        cout << "Error opening file!" << endl;
        return 1;
    }

    string line;
    while (getline(inFile, line)) {  // read line by line
        cout << line << endl;
    }

    inFile.close();  // close the file

    return 0;
}
```

```
Error opening file!


=== Code Exited With Errors ===
```

🖥️ What happens here?

- ☐ `ifstream inFile("data.txt");` opens the file for reading
- ☐ `getline(inFile, line)` reads the file line by line until the end
- ☐ prints each line to the screen