C++

Video 6:

C++ Header files & Operators | C++ Tutorials for Beginners #6

What are Header files?

Header files in C++ contain function declarations, macros, and definitions that help in code reusability and modular programming. They are included using #include at the beginning of a program.

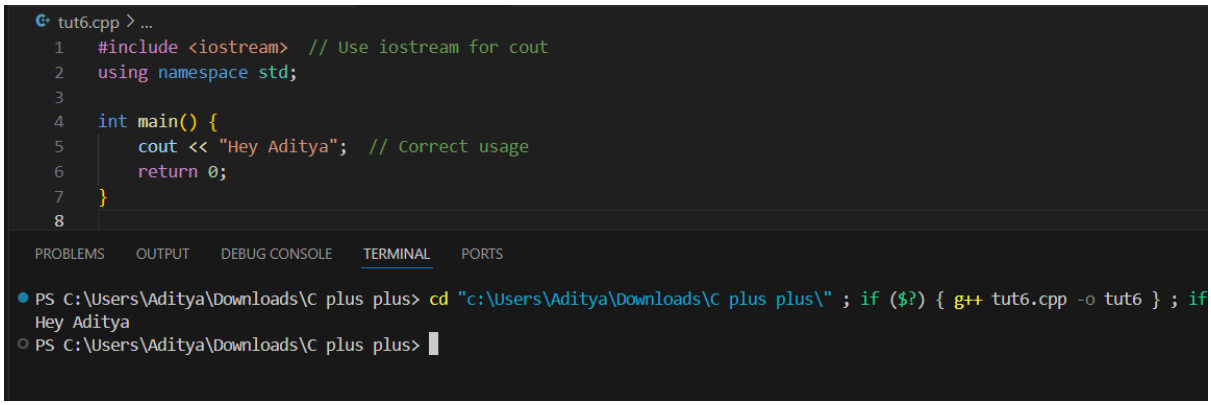
Types of Header Files

1. **Standard Header Files (Built-in) or system header files comes with compiler**

   Provided by C++ for common tasks.

   Examples:

   - #include <iostream> → Input/Output
   - #include <cmath> → Mathematical functions
   - #include <string> → String handling



2. **User-Defined Header Files**

   Created by programmers for modularity.

   Example: #include "myfile.h"

What are operators in C++?

Operators in C++ are symbols that perform operations on variables and values. They are categorized into different types.

Types:

Type	Operators
Arithmetic	+ - * / %
Relational	== != > < >= <=
Logical	`&&
Bitwise	`&
Assignment	= += -= *= /= %=
Increment/Decrement	++ --
Ternary	? :
Type Cast	(type)

1. **Arithmetic Operators**

Used for basic mathematical operations.

Operator	Description	Example (a = 10, b = 5)	Output
+	Addition	a + b	15
-	Subtraction	a - b	5
*	Multiplication	a * b	50
/	Division	a / b	2
%	Modulus (Remainder)	a % b	0

Example:

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

int main(){
    int a = 10, b = 8;
    cout << "Sum " << a+b  << endl;
    cout << "Difference " << a-b  << endl;
    cout << "Multiplication " << a*b  << endl;
    cout << "Divison " << a/b  << endl;
    cout << "Modulo " << a%b  << endl;

}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++ tut6.cpp -o tut6 }
Sum 18
Difference 2
Multiplication 80
Divison 1
Modulo 2
PS C:\Users\Aditya\Downloads\C plus plus>
```

2. Increment and Decrement Operators

Used to increase or decrease values.

Operator	Description	Example (a = 5)	Output
++	Increment	a++ or ++a	6
--	Decrement	a-- or --a	4

Example:

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

int main() {
    int a = 5, b = 33;
    cout << "a is: " << a << "  b is: " << b << endl;
    cout << "Increment: " << a++ << endl;  // Prints 5, then a becomes 6
    cout << a << endl;
    cout << "Decrement: " << b-- << endl;  // Prints 33, then b becomes 32
    cout << b << endl;
    cout << "Increment: " << ++a << endl;  // a is 7 now (6+1), prints 7
    cout << a << endl;
    cout << "Decrement: " << --b << endl;  // b is 31 now (32-1), prints 31
    cout << b << endl;
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++ tut6.cpp -o tut6 } ; if ($?) { .\tut6 }
a is: 5  b is: 33
Increment: 5
6
Decrement: 33
32
Increment: 7
7
Decrement: 31
31
PS C:\Users\Aditya\Downloads\C plus plus>
```

What is the basic difference between a++ and ++a?

Both a++ (post-increment) and ++a (pre-increment) increase the value of a by 1, but they behave differently in expressions.

Operator	Type	Behavior
a++	**Post-Increment**	Returns the current value of a, then increments it.
++a	**Pre-Increment**	Increments a first, then returns the new value.

3. Assignment Operators

Used to assign values.

Operator	Description	Example (a = 10)	Equivalent To
=	Assign value	a = 10	a = 10
+=	Add and assign	a += 5	a = a + 5
-=	Subtract and assign	a -= 5	a = a - 5
*=	Multiply and assign	a *= 5	a = a * 5
/=	Divide and assign	a /= 5	a = a / 5
%=	Modulus and assign	a %= 5	a = a % 5

A basic example:

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

int main() {
    int a = 5; //assigning value of a to be 5
    int b = 6; //assigning value of b to be 6
    cout << "a is " << a << " b is " << b << endl;
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++ tut6.cpp -o tut6 } ; if ($?
a is 5 b is 6
PS C:\Users\Aditya\Downloads\C plus plus>
```

Some more example:

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

int main() {
    int a = 10; // Assign value
    cout << "Initial value of a: " << a << endl;
    a += 5; // Add and assign
    cout << "After a += 5: " << a << endl; // a = 10 + 5 = 15
    a -= 5; // Subtract and assign
    cout << "After a -= 5: " << a << endl; // a = 15 - 5 = 10
    a *= 5; // Multiply and assign
    cout << "After a *= 5: " << a << endl; // a = 10 * 5 = 50
    a /= 5; // Divide and assign
    cout << "After a /= 5: " << a << endl; // a = 50 / 5 = 10
    a %= 5; // Modulus and assign
    cout << "After a %= 5: " << a << endl; // a = 10 % 5 = 0
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++ tut6.cpp -o tut6 } ; if ($?) { .\tu
Initial value of a: 10
After a += 5: 15
After a -= 5: 10
After a *= 5: 50
After a /= 5: 10
After a %= 5: 0
PS C:\Users\Aditya\Downloads\C plus plus>
```

4. Comparison Operators

Used to compare values.

Operator	Description	Example (a = 10, b = 5)	Output
==	Equal to	a == b	false
!=	Not equal to	a != b	true
>	Greater than	a > b	true
<	Less than	a < b	false
>=	Greater than or equal to	a >= b	true
<=	Less than or equal to	a <= b	false

```cpp
//4. Comparison operator
#include <iostream>
using namespace std;

int main() {
    int a = 5, b = 7;

    cout << "Checking if a == b: " << (a == b) << endl;  // false (0)
    cout << "Checking if a != b: " << (a != b) << endl;  // true (1)
    cout << "Checking if a > b: " << (a > b) << endl;    // false (0)
    cout << "Checking if a < b: " << (a < b) << endl;    // true (1)
    cout << "Checking if a >= b: " << (a >= b) << endl;  // false (0)
    cout << "Checking if a <= b: " << (a <= b) << endl;  // true (1)

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++ tut6.cpp -o tut
Checking if a == b: 0
Checking if a != b: 1
Checking if a > b: 0
Checking if a < b: 1
Checking if a >= b: 0
Checking if a <= b: 1
PS C:\Users\Aditya\Downloads\C plus plus>
```

5. Logical operators

Used to perform logical operations.

Operator	Description	Example (a = 10, b = 5)	Output
&&	Logical AND	(a > 5 && b < 10)	true
`	`	`	Logical OR
!	Logical NOT	!(a == b)	true

Truth Table for && (AND):

Condition 1	Condition 2	Result (&&)
true	true	true (1)
true	false	false (0)
false	true	false (0)
false	false	false (0)

Truth Table for ! (NOT):

Condition	Result (!)
true	false (0)
false	true (1)

Example:

```cpp
//5. Logical operator
#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 5;

    cout << "Logical AND (a > 5 && b < 10): " << (a > 5 && b < 10) << endl; // true (1)
    cout << "Logical OR (a < 5 || b < 10): " << (a < 5 || b < 10) << endl;  // true (1)
    cout << "Logical NOT (!(a == b)): " << !(a == b) << endl;               // true (1)

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++ tut6.cpp -o tut6 } ; if ($?) { .\tut6 }
Logical AND (a > 5 && b < 10): 1
Logical OR (a < 5 || b < 10): 1
Logical NOT (!(a == b)): 1
PS C:\Users\Aditya\Downloads\C plus plus>
```

Video 7:

C++ Reference Variables & Typecasting | C++ Tutorials for Beginners #7

What is scope resolution operator?

Scope Resolution Operator (::) in C++

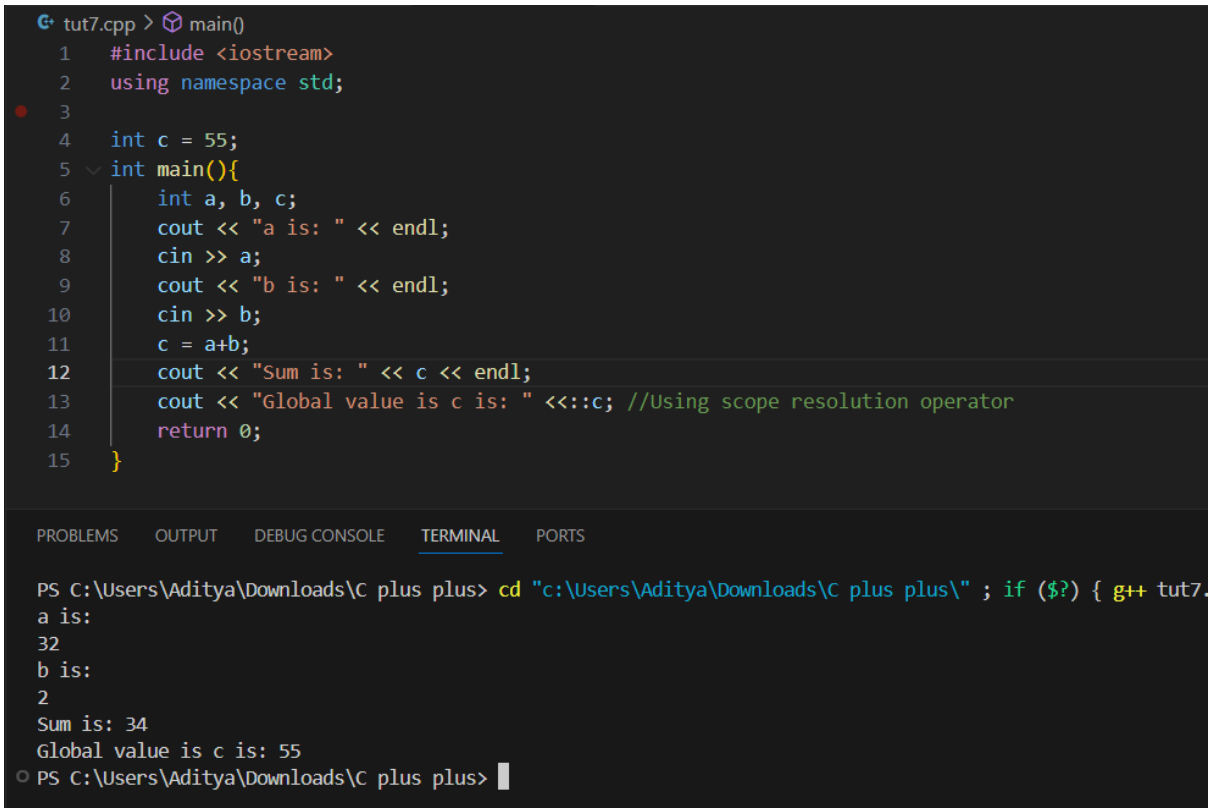
The **scope resolution operator (::)** in C++ is used to access variables, functions, or classes that are defined in a specific scope when there is a name conflict or ambiguity.

Use Cases of :: (Scope Resolution Operator)

Use Case	Description
Global Variable Access	Accesses a global variable when a local variable has the same name.
Class Member Access	Defines functions outside the class.
Namespace Access	Accesses elements inside a specific namespace.
Static Class Members	Accesses static variables or functions in a class.

A basic example:

Printing the value of global variable when the local value of it also exists.



How Are float and long double Passed to a Function in C++?

When a **float** or **long double** is passed to a function, it undergoes **type promotion** based on the function signature.

```
16
17  int main(){
18      int a = 45; //it is an integer
19      float b = 34.4; //Will be passed as double if called in function
20      long double c = 34.4; //will be passed as double if called in function
21      return 0;
22  }
```

How to let it continue its original form?

```
24  int main(){
25      int a = 45; //it is an integer
26      float b = 34.4f; //Will be passed as float as f or F is used
27      long double c = 34.4l; //will be passed as double as l or L is used
28      return 0;
29  }
```

How to get the size of the numbers?
We can use sizeof(value) to get the size of the number depending on the system architecture.

Example:

```
31  int main(){
32      int a = 34;
33      float b = 4.4f;
34      long double c = 3.4l;
35      cout << "Size of 34 is " << sizeof(a) << endl;
36      cout << "Size of 4.4f is " << sizeof(b) << endl;
37      cout << "Size of 3.4l is " << sizeof(c) << endl;
38      return 0;
39
40  }
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g+
Size of 34 is 4
Size of 4.4f is 4
Size of 3.4l is 12
PS C:\Users\Aditya\Downloads\C plus plus>
```

Floating-point numbers like 5.5, 3.14, and 2.71828 are called **floating-point literals** in C++. These literals can be explicitly specified as float, double, or long double using suffixes.

Types of Floating-Point Literals in C++

Literal	Type	Example
float	Ends with f or F	3.14f, 5.5F
double (Default)	No suffix (default type)	3.14, 5.5
long double	Ends with l or L	3.14L, 5.5l

What are Reference variables?

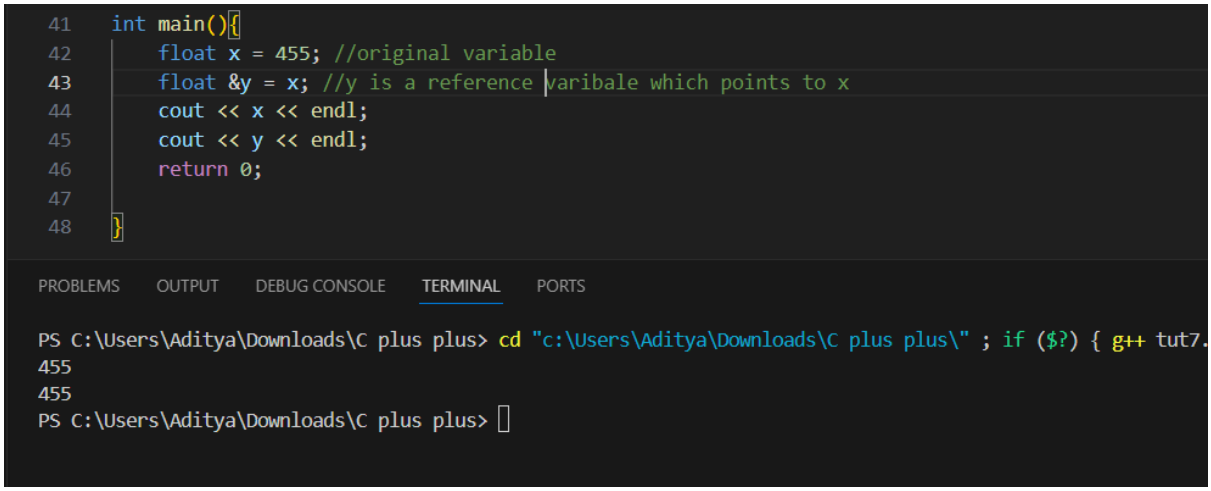
A reference variable in C++ is an alias for an existing variable. It does not store a separate value but rather refers to the original variable. Any changes made to the reference will affect the original variable**.**

Syntax:

data_type &reference_name = original_variable;

- & before the reference name means it is a **reference**.
- A reference **must be initialized** when declared.

Example a:



Example b:

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

int main() {
    int x = 10;
    int &y = x;  // y is a reference to x
    cout << "x: " << x << ", y: " << y << endl;  // Both print 10
    y = 20;  // Changing y also changes x
    cout << "x: " << x << ", y: " << y << endl;  // Both print 20
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($
x: 10, y: 10
x: 20, y: 20
PS C:\Users\Aditya\Downloads\C plus plus>
```

Key Properties of Reference Variables

Property	Description
Must be initialized	A reference must be assigned a value when declared.
Acts as an alias	Changes to the reference affect the original variable.
Cannot be re-assigned	A reference cannot refer to another variable after initialization.
Memory location is same	The reference and original variable share the same memory address.

What is Typecasting?

Typecasting in C++ is the process of converting a variable from one data type to another. It helps in handling different types of data efficiently, ensuring compatibility between variables of different types.

Types of Typecasting in C++

Type	Description	Example
Implicit Typecasting (Automatic Conversion)	Performed by the compiler when converting a smaller data type to a larger one (e.g., int to float).	int x = 10; float y = x;
Explicit Typecasting (Manual Conversion)	Done by the programmer using casting syntax (e.g., (type)value or static_cast<type>(value)).	double d = 3.14; int x = (int)d;
C++-Style Casting	Uses static_cast, dynamic_cast, const_cast, and reinterpret_cast for better control.	int x = static_cast<int>(3.99);

Implicit typecasting:

The compiler **automatically converts** a smaller data type into a larger one

Expression	Implicit Type Promotion	Result Type
int + float	int → float	float
int + double	int → double	double
float + double	float → double	double

```
64  #include <iostream>
65  using namespace std;
66  int main() {
67      float x = 10.43;
68      int y = x;  // Implicit conversion from float to int
69      cout << "y: " << y << endl;  // Output: 10
70      return 0;
71  }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++ tut7.cpp -o tut
y: 10
```

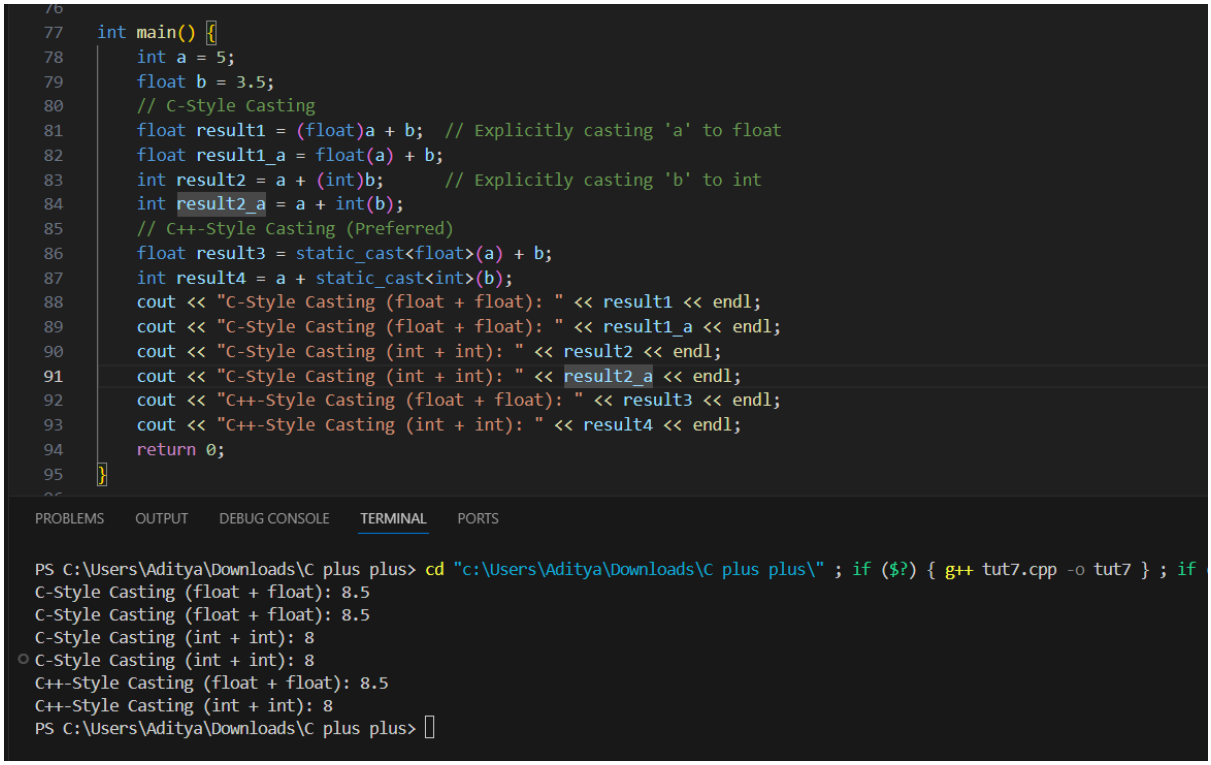
Explicit typecasting:

In **explicit typecasting**, we manually convert one data type to another instead of relying on implicit type promotion.

Methods of Explicit Typecasting in C++

1. **C-Style Casting** → (type)value
2. **C++-Style Casting** → static_cast<type>(value)

Casting Method	Syntax	Usage
C-Style Casting	**(type)value**	**Works but is less safe.**
C++-Style Casting	**static_cast<type>(value)**	**Safer and preferred in modern C++.**



```cpp
int main() {
    int a = 5;
    float b = 3.5;
    // C-Style Casting
    float result1 = (float)a + b;  // Explicitly casting 'a' to float
    float result1_a = float(a) + b;
    int result2 = a + (int)b;      // Explicitly casting 'b' to int
    int result2_a = a + int(b);
    // C++-Style Casting (Preferred)
    float result3 = static_cast<float>(a) + b;
    int result4 = a + static_cast<int>(b);
    cout << "C-Style Casting (float + float): " << result1 << endl;
    cout << "C-Style Casting (float + float): " << result1_a << endl;
    cout << "C-Style Casting (int + int): " << result2 << endl;
    cout << "C-Style Casting (int + int): " << result2_a << endl;
    cout << "C++-Style Casting (float + float): " << result3 << endl;
    cout << "C++-Style Casting (int + int): " << result4 << endl;
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++ tut7.cpp -o tut7 } ; if
C-Style Casting (float + float): 8.5
C-Style Casting (float + float): 8.5
C-Style Casting (int + int): 8
C-Style Casting (int + int): 8
C++-Style Casting (float + float): 8.5
C++-Style Casting (int + int): 8
PS C:\Users\Aditya\Downloads\C plus plus>
```

Is int(a) and (int)a same?

int(a) and (int)a are not exactly the same, even though they may seem similar in simple cases.

Syntax	Style	Usage	Safety
(int)a	C-style Cast	Works for basic types	❌ Unsafe for complex types
int(a)	Functional Cast	Works for basic types and constructors	⚠️ Can be ambiguous
static_cast<int>(a)	C++-Style Cast (Best Practice)	Works for both basic and complex types	✅ Safest option

C++-style casting:

C++ provides safer typecasting methods.

```cpp
#include <iostream>
using namespace std;
int main() {
    double num = 9.87;
    int val = static_cast<int>(num);  // Safe conversion to integer
    cout << "num: " << num << ", val: " << val << endl;
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++
num: 9.87, val: 9
PS C:\Users\Aditya\Downloads\C plus plus>
```

Video 8:

Constants, Manipulators & Operator Precedence | C++ Tutorials for Beginners #8

We know that we can change the value of a variable even after it is declared once, example:

tut8.cpp > main()

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

int main(){
    int a = 44;
    cout << "The value of a was:" << a << endl;
    a = 99;
    cout << "The value of a is:" << a << endl;
    return 0;

}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++ tut8.cpp -o tut8
The value of a was:44
The value of a is:99
PS C:\Users\Aditya\Downloads\C plus plus>
```

So, we can overwrite the declared variables, but how can we avoid it? By using the const keyword.

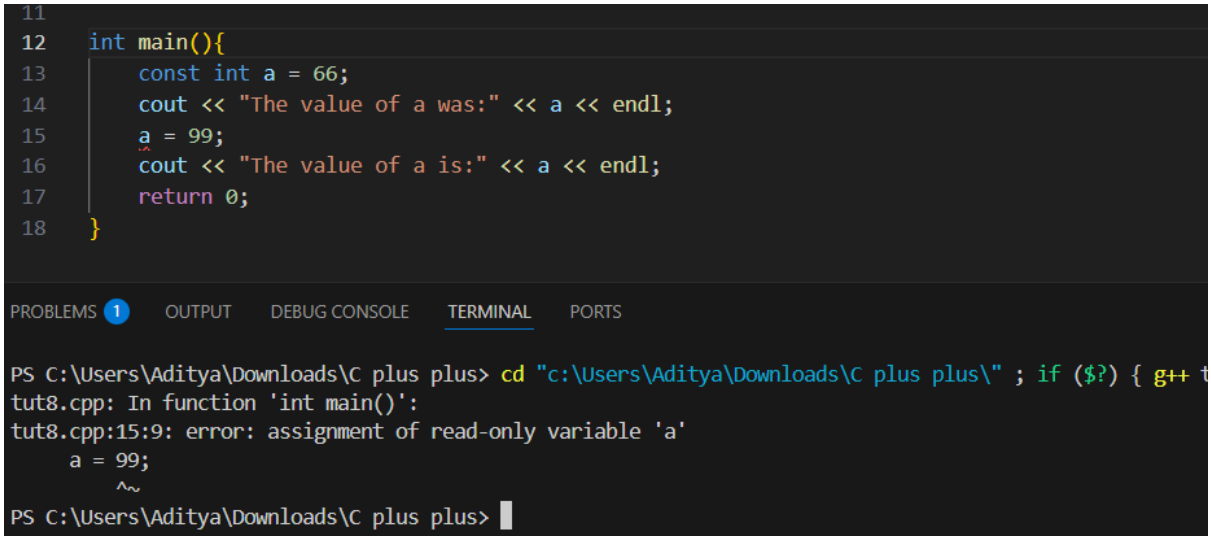
What Are Constants in C++?

A **constant** in C++ is a fixed value that **cannot be changed** after it is defined. Constants help in making code more readable, preventing accidental modifications of important values.

Types of Constants in C++

Type	Description	Example
Literal Constants	Hardcoded values in the program.	10, 3.14, 'A', "Hello"
const Keyword	Declares a variable as constant.	const int x = 5;
#define Macro	Preprocessor directive for defining constants.	#define PI 3.14
constexpr (C++11)	Compile-time constant evaluation.	constexpr int y = 10;
Enumerations (enum)	Defines named integer constants.	enum Color { RED=1, BLUE=2 };

A basic example:



What Are Manipulators in C++?

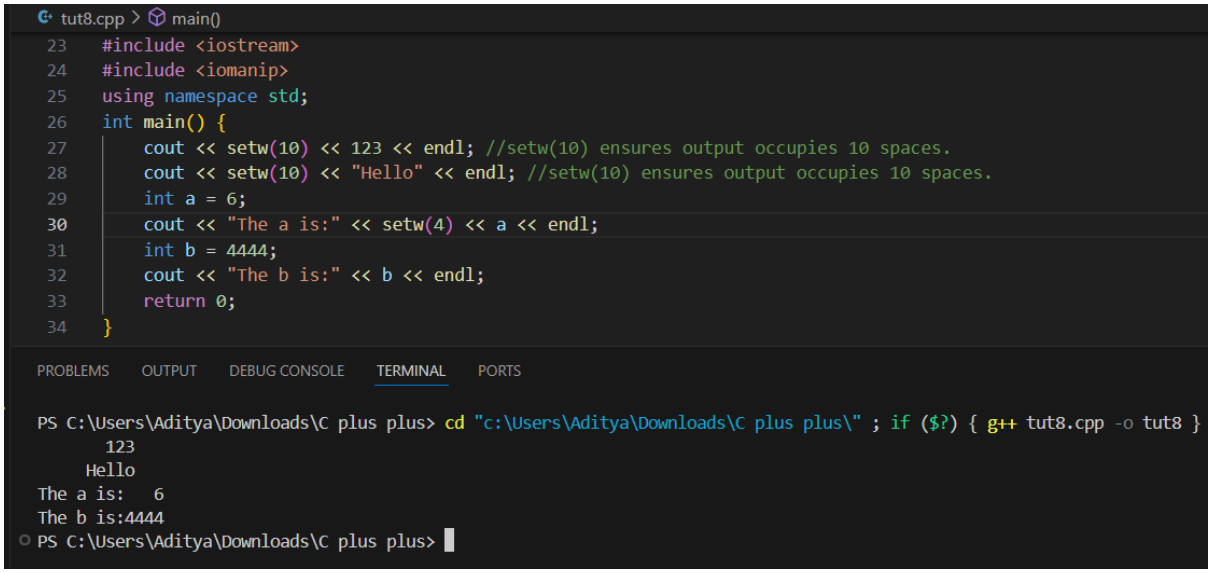
Manipulators in C++ are functions used to **format output** in cout or **modify input** in cin. They help in controlling **alignment, precision, width, and data formatting** without modifying the actual data.

Types of Manipulators in C++

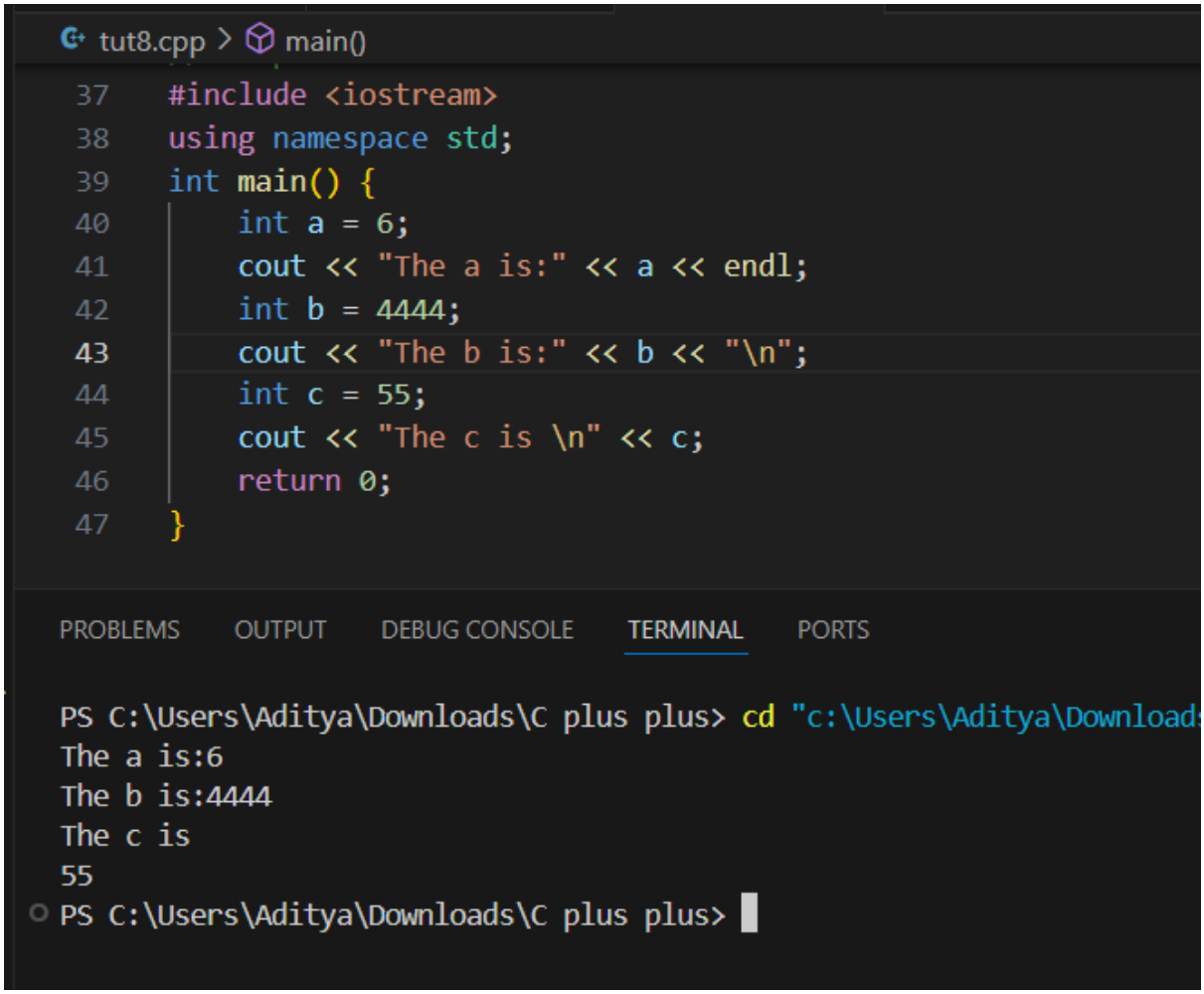
Manipulator	Description	Example
endl	Moves to the next line (same as \n).	cout << "Hello" << endl;
setw(n)	Sets width of output (requires <iomanip>).	cout << setw(10) << 123;
setprecision(n)	Sets decimal precision for floating numbers.	cout << setprecision(3) << 3.14159;
fixed	Displays floating-point numbers in fixed decimal notation.	cout << fixed << 3.14159;

Manipulator	Description	Example
scientific	Displays floating-point numbers in scientific notation.	cout << scientific << 3.14159;
setfill(c)	Fills empty spaces with character c (requires <iomanip>).	cout << setfill('*') << setw(5) << 42;
left / right	Aligns output left or right.	cout << left << setw(10) << "Hello";

Example of setw(n):



Example of endl:



What Is Precedence of Operators in C++?

Operator precedence in C++ determines **which operator is evaluated first** when multiple operators appear in an expression. Operators with **higher precedence** are executed **before** those with **lower precedence**.

Precedence	Operator	Associativity	Category
1 (Highest)	() [] -> .	**Left to Right**	Parentheses, Member Access
2	++ -- + - ! ~ * & sizeof	**Right to Left**	Unary Operators
3	* / %	**Left to Right**	Multiplication, Division, Modulo
4	+ -	**Left to Right**	Addition, Subtraction
5	<< >>	**Left to Right**	Bitwise Shift
6	< <= > >=	**Left to Right**	Relational Operators
7	== !=	**Left to Right**	Equality Operators

Precedence	Operator	Associativity	Category
8	&	**Left to Right**	Bitwise AND
9	^	**Left to Right**	Bitwise XOR
10	`	`	**Left to Right**
11	&&	**Left to Right**	Logical AND
12	`		`
13	?:	**Right to Left**	Ternary Conditional
14	= += -= *= /= %=	**Right to Left**	Assignment Operators
15 (Lowest)	,	**Left to Right**	Comma Operator

Evaluation of a * 5 + b - 45 + 87 in C++

Evaluation Steps:

1. a * 5 → 2 * 5 = 10
2. 10 + b → 10 + 3 = 13
3. 13 - 45 → -32
4. -32 + 87 → 55

A basic code example:

```
59  int main(){
60      int answer = 10 * 5 + 2;
61      cout << "Result: " << answer << endl;
62      int result  = 10 + 5 * 2; // Multiplication (*) has higher precedence than addition (+)
63      cout << "Result: " << result;
64      return 0;
65  }
66
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++ tut8.cpp -o
Result: 52
Result: 20
PS C:\Users\Aditya\Downloads\C plus plus>
```

Video 9:

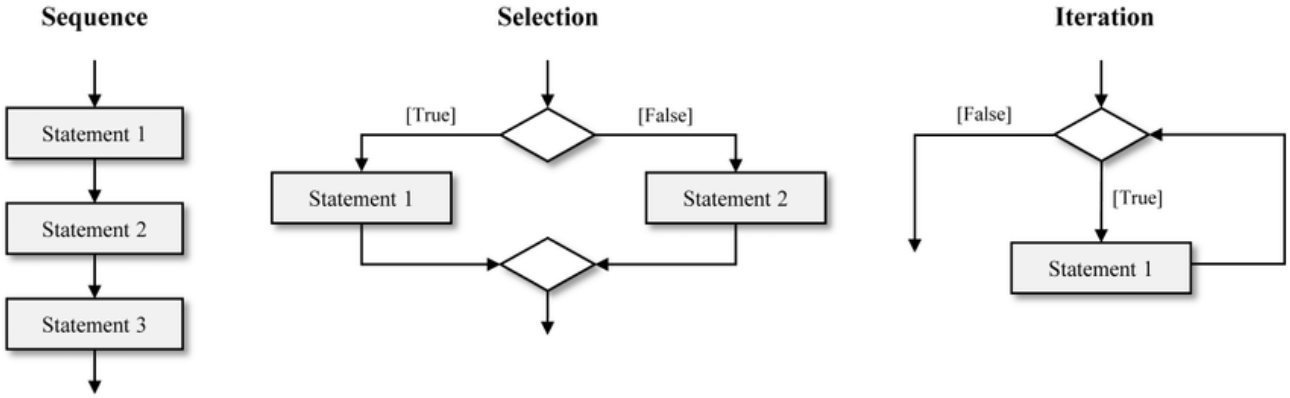
C++ Control Structures, If Else and Switch-Case Statement | C++ Tutorials for Beginners #9

What Are Control Structures in C++?

Control structures in C++ are the fundamental building blocks that **control the flow of execution** in a program. They allow decisions, loops, and jumps, ensuring dynamic and logical behavior in programs.

Types of Control Structures in C++

1. **Sequential Control Structure** – Executes statements **line by line** (default).
2. **Decision/Selection Control Structure** – Executes **specific blocks of code** based on conditions.
3. **Looping/Iteration Control Structure** – Repeats a block of code **multiple times**.
4. **Jump Control Structure** – **Alters normal execution** (e.g., break, continue, goto).



Example of Sequential:

```
tut9.cpp > main()
1   #include <iostream>
2   using namespace std;
3   int main() {
4       cout << "Step 1" << endl; //first
5       cout << "Step 2" << endl; //second
6       cout << "Step 3" << endl; //third
7       return 0;
8   }

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?
if ($?) { .\tut9 }
Step 1
Step 2
Step 3
PS C:\Users\Aditya\Downloads\C plus plus>
```

Example of Selection: if else

```cpp
//Selection
int main(){
    int a = 9;
    if (a>10)
    {
        cout << "a is greater than 10" << endl;
    }
    else{
        cout << "a is less than 10";
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if (
if ($?) { .\tut9 }
a is less than 10
PS C:\Users\Aditya\Downloads\C plus plus>
```

Example of selection: switch case

```cpp
//Jump (under sequential)
#include <iostream>
using namespace std;
int main() {
    int choice = 2;
    switch (choice) {
        case 1: cout << "Choice is 1"; break;
        case 2: cout << "Choice is 2"; break;
        default: cout << "Invalid choice";
    }
    return 0;
}

```

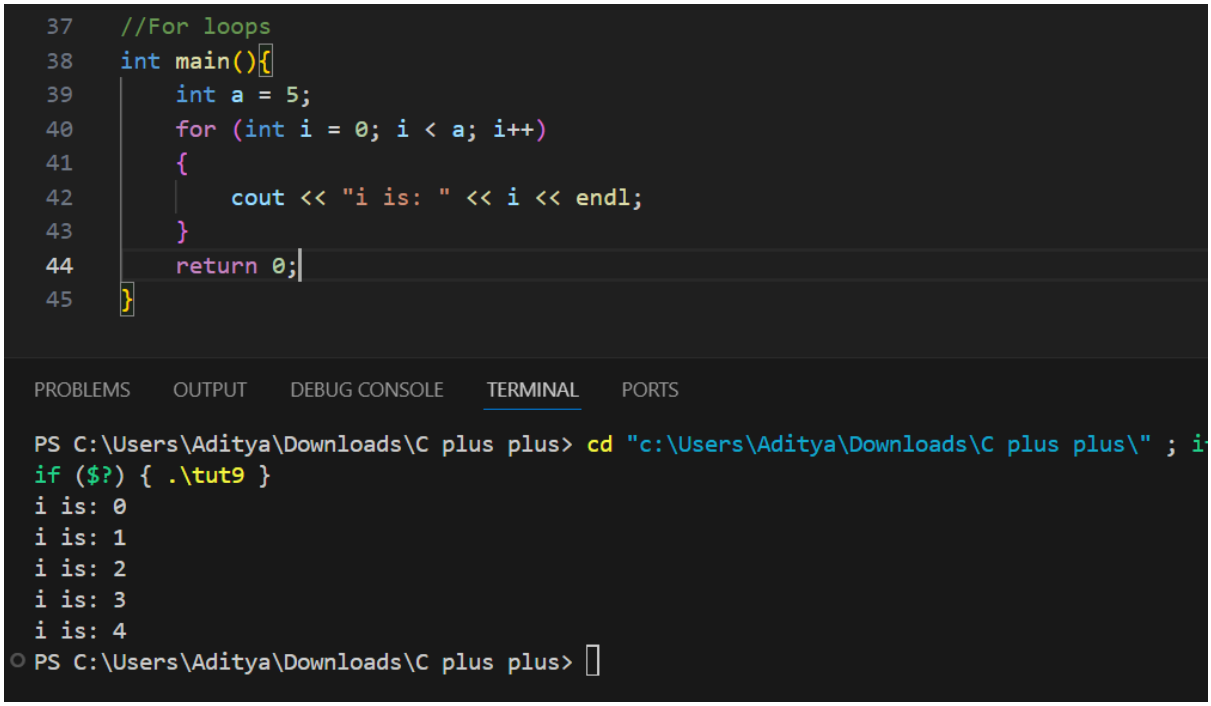
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus
if ($?) { .\tut9 }
Choice is 2
PS C:\Users\Aditya\Downloads\C plus plus>
```

Video 10:

For, While and do-while loops in C++ | C++ Tutorials for Beginners #10

Example of for loops:

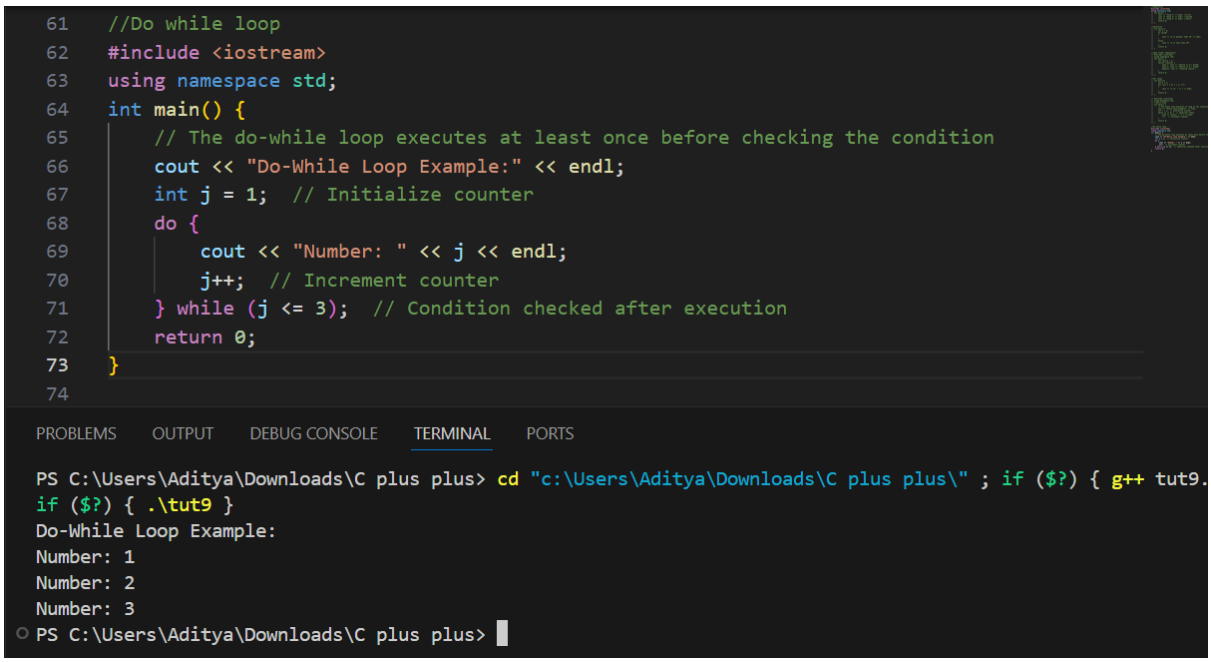


```
37  //For loops
38  int main(){
39      int a = 5;
40      for (int i = 0; i < a; i++)
41      {
42          cout << "i is: " << i << endl;
43      }
44      return 0;
45  }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; i
if ($?) { .\tut9 }
i is: 0
i is: 1
i is: 2
i is: 3
i is: 4
PS C:\Users\Aditya\Downloads\C plus plus>
```

Example of do while loop:



```
61  //Do while loop
62  #include <iostream>
63  using namespace std;
64  int main() {
65      // The do-while loop executes at least once before checking the condition
66      cout << "Do-While Loop Example:" << endl;
67      int j = 1;  // Initialize counter
68      do {
69          cout << "Number: " << j << endl;
70          j++;  // Increment counter
71      } while (j <= 3);  // Condition checked after execution
72      return 0;
73  }
74
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++ tut9.
if ($?) { .\tut9 }
Do-While Loop Example:
Number: 1
Number: 2
Number: 3
PS C:\Users\Aditya\Downloads\C plus plus>
```

Example of while loops:

```cpp
#include <iostream>
using namespace std;
//while loop
int main() {
    // The while loop executes as long as the condition is true
    cout << "While Loop Example:" << endl;
    int i = 1;  // Initialize counter
    while (i <= 5) {  // Condition check
        cout << "Count: " << i << endl;
        i++;  // Increment counter
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g++
if ($?) { .\tut9 }
While Loop Example:
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
PS C:\Users\Aditya\Downloads\C plus plus>
```

Example of break statements:

```cpp
#include <iostream>
using namespace std;
int main() {
    // The break statement terminates the loop prematurely
    cout << "Break Statement Example:" << endl;
    for (int k = 1; k <= 5; k++) {
        if (k == 3) break;  // Stop loop when k is 3
        cout << "k = " << k << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; i
if ($?) { .\tut9 }
Break Statement Example:
k = 1
k = 2
PS C:\Users\Aditya\Downloads\C plus plus>
```

Example of continue statement:

```cpp
#include <iostream>
using namespace std;
int main() {
    // The continue statement skips the current iteration and moves to the next
    cout << "Continue Statement Example:" << endl;
    for (int m = 1; m <= 5; m++) {
        if (m == 3) continue;  // Skip when m is 3
        cout << "m = " << m << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g+
if ($?) { .\tut9 }
Continue Statement Example:
m = 1
m = 2
m = 4
m = 5
PS C:\Users\Aditya\Downloads\C plus plus>
```

Control Statement	Importance in Competitive Programming	Usage
For Loop	🔥🔥🔥🔥🔥 (Most important)	Array traversal, brute force, DP, Graphs
While Loop	🔥🔥🔥🔥 (Very important)	Searching, GCD, Number Theory
Break Statement	🔥🔥🔥🔥 (Optimization)	Early exits, Prime check
Continue Statement	🔥🔥🔥 (Helpful)	Skipping cases, filtering
Do-While Loop	🔥 (Rare)	Menu-driven problems
Goto Statement	❌ (Avoid)	Almost never used