C++

Video 11:

Break and Continue Statements in C++ | C++ Tutorials for Beginners #11

Break Statement in C++

The break statement in C++ is used to **immediately exit a loop** or a switch statement when a certain condition is met. It **stops further execution** inside the loop and moves to the next statement after the loop.

Usage of break in C++

1. I**nside Loops** (for, while, do-while) – Terminates the loop early.
2. I**nside switch Statements** – Prevents fall-through to the next case.

A basic example: where the loop ends when i == 2:

```
4   //example of a break keyword
5   //Leaving the loop when i =2
6   int main(){
7       for (int i = 0; i < 4; i++)
8       {
9           cout << "i is: " << i << endl;
10          if (i==2)
11          {
12              break; //it will print as 0,1,2 as if condition came after cout
13          }
14      }
15      return 0;
16  }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if
i is: 0
i is: 1
i is: 2
PS C:\Users\Aditya\Downloads\C plus plus>
```

Another example:

```
17
18  //Using the break statement before the cout
19  int main(){
20      for (int i = 0; i < 4; i++)
21      {
22          if (i==2)
23          {
24              break; //it will print as 0,1 as if condition came before cout
25          }
26          cout << "i is: " << i << endl;
27      }
28      return 0;
29  }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

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

When to Use break?

✓ When you want to **exit a loop** immediately.
✓ When handling **menu-based programs** using switch.
✓ When optimizing search problems (**exit early when found**).

continue Statement in C++

The continue statement in C++ is used to **skip the current iteration** of a loop and jump to the next iteration. Unlike break, it does **not terminate the loop**; it just skips the remaining code for the current loop iteration and proceeds to the next cycle.

Usage of continue in C++

- Works inside for, while, and do-while loops.
- Skips certain iterations without stopping the entire loop.

Example 1:

```cpp
//Example of continue keyword
//Using continue keyword after cout
int main(){
    for (int i = 0; i <=4; i++)
    {
        cout << "i is: " << i << endl;
        if (i==0)
        {
            continue;
        }
    }
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\"
i is: 0
i is: 1
i is: 2
i is: 3
i is: 4
PS C:\Users\Aditya\Downloads\C plus plus>
```

Example 2:

```cpp
//Example of continue keyword
//Using continue keyword before cout
int main(){
    for (int i = 0; i <=4; i++)
    {
        if (i==2)
        {
            continue;
        }
        cout << "i is: " << i << endl;
    }
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; i
i is: 0
i is: 1
i is: 3
i is: 4
PS C:\Users\Aditya\Downloads\C plus plus>
```

When to Use continue?

✓ When you want to **skip certain iterations** without breaking the loop.
✓ Useful for **filtering elements** in arrays or conditions.
✓ Efficient in **search problems** to skip unnecessary checks.

Video 12:

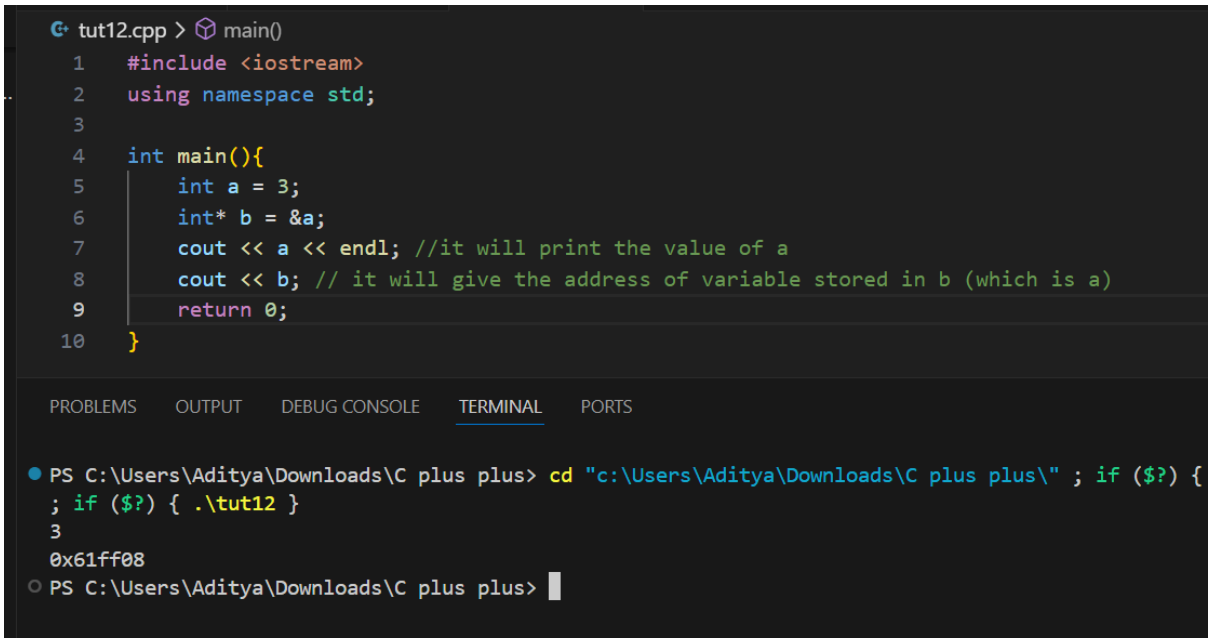
Pointers in C++ | C++ Tutorials for Beginners #12

What are pointers in C++?

A pointer in C++ is a variable that stores the memory address of another variable. It allows direct memory access and manipulation, improving efficiency. Pointers are useful in dynamic memory allocation, arrays, and functions.

It is a type of data type which stores the address of other data types.

A basic example:



We can get the address of the variable by two ways:

Way 1: Above one

Way 2:

```cpp
int main(){
    int a = 3;
    int* b = &a;
    cout << a << endl; //it will print the value of a
    cout << &a;
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plu
; if ($?) { .\tut12 }
3
0x61ff08
PS C:\Users\Aditya\Downloads\C plus plus>
```

Thus, using the pointer name(b) or &a is same.

One more example:

```cpp
int main() {
    int x = 10;       // Normal variable
    int* ptr = &x;    // Pointer storing the address of x
    cout << "Value of x: " << x << endl;
    cout << "Address of x: " << &x << endl;
    cout << "Pointer value (address stored): " << ptr << endl;
    cout << "Value at pointer (dereferencing): " << *ptr << endl;
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plu
; if ($?) { .\tut12 }
Value of x: 10
Address of x: 0x61ff08
Pointer value (address stored): 0x61ff08
Value at pointer (dereferencing): 10
PS C:\Users\Aditya\Downloads\C plus plus>
```

ptr stores the **address** of x.

*ptr (dereferencing) gives the **value of x**.

Pointers to pointer:

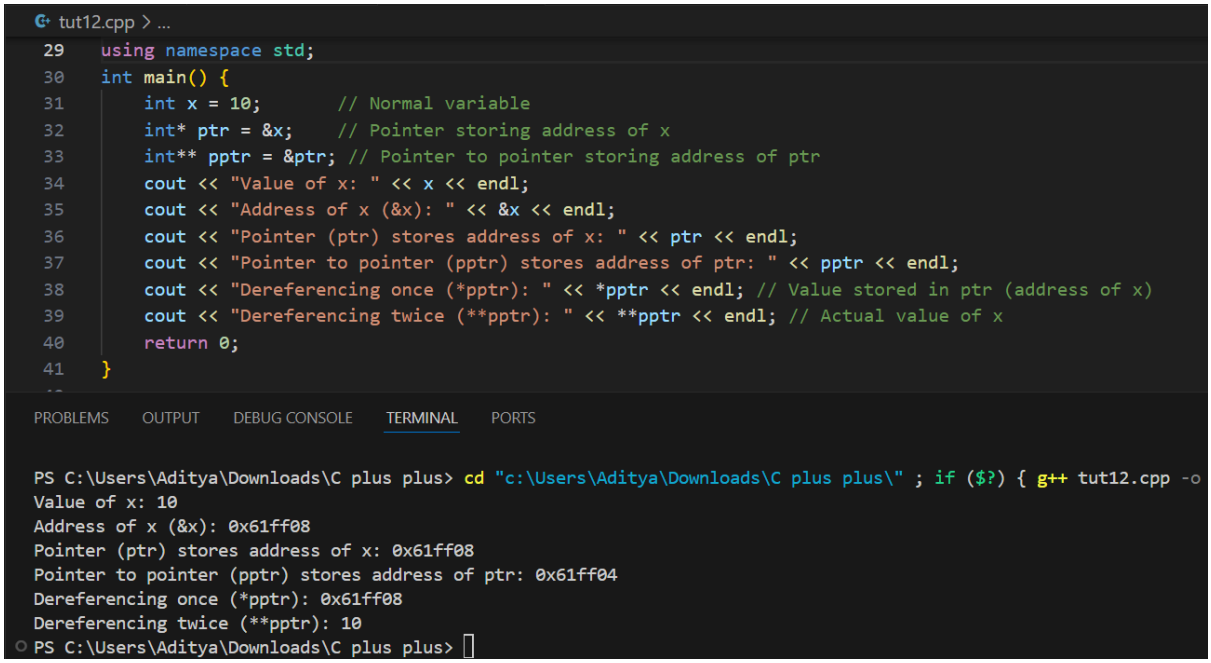
A pointer to a pointer in C++ is a pointer that stores the address of another pointer, which in turn stores the address of a variable. This allows multi-level indirection and is useful in dynamic memory allocation, arrays, and advanced data structures like linked lists.

Syntax:
datatype **pointer_name;

When to Use Pointer to Pointer?

✓ Passing pointers by reference in functions.
✓ Managing dynamic memory allocation (especially in 2D arrays).
✓ Handling linked lists and complex data structure

A basic example:



Video 13:

Arrays & Pointers Arithmetic in C++ | C++ Tutorials for Beginners #13

Arrays:

An **array** in C++ is a **fixed-size collection of elements** of the same data type, stored in **contiguous memory locations**. Arrays allow easy access to elements using an **index**. They can be **one-dimensional, multi-dimensional (2D, 3D), or dynamic**.

Syntax:

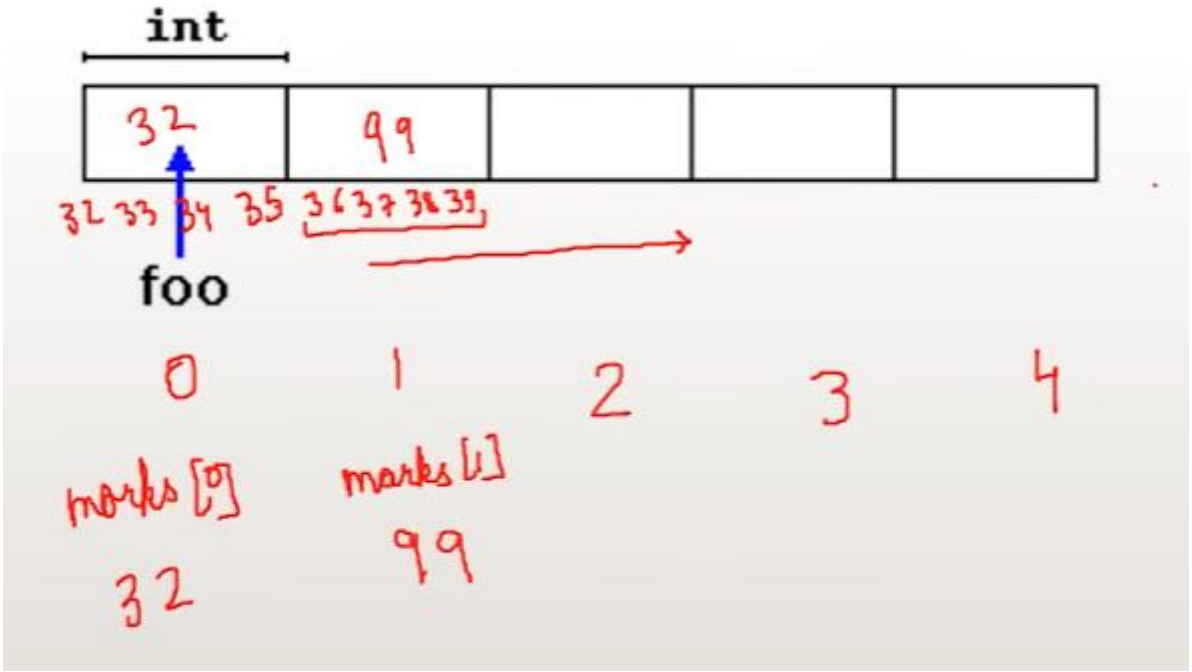
datatype array_name[size];

Types of Arrays in C++

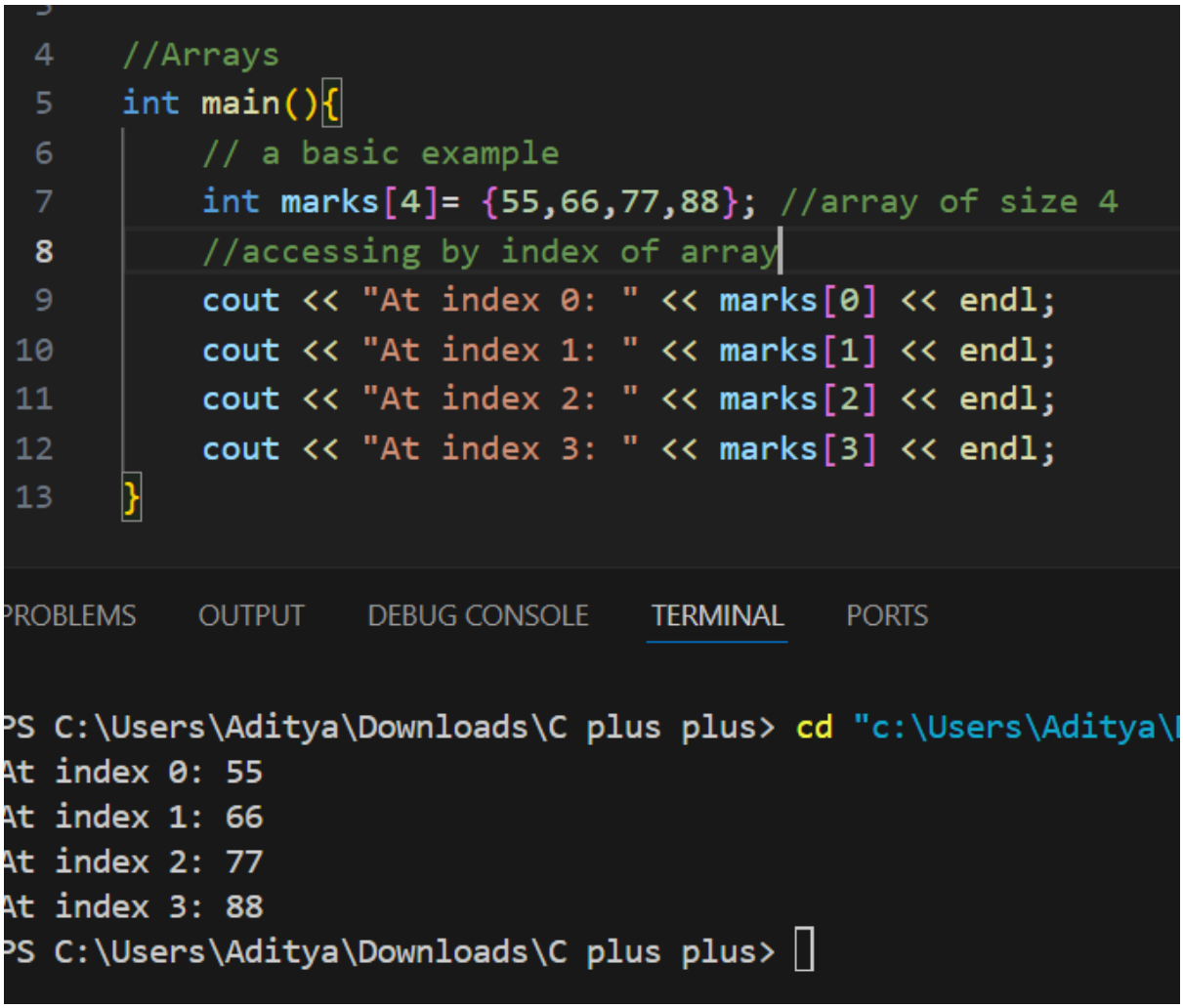
Type	Description
1D Array	Stores data in a single row (e.g., int arr[5]).
2D Array	Stores data in rows & columns (e.g., int matrix[3][3]).
3D Array	Extends to three dimensions (e.g., int cube[3][3][3]).
Dynamic Array	Size is allocated at runtime using new (Heap Memory).

Key Features of Arrays

✓ **Fast Access** – Elements are accessed using an index.
✓ **Fixed Size** – The size of the array is determined at declaration.
✓ **Efficient Memory Usage** – Uses contiguous memory locations.
✓ **Supports Multiple Dimensions** – 1D, 2D, 3D arrays.



A basic example:



We can also leave the marks[4] as marks[], C++ will automatically get the size as per the values entered.

```cpp
//Arrays
int main(){
    // a basic example
    int marks[]= {55,66,77,88}; //array of size 4
    //accessing by index of array
    cout << "At index 0: " << marks[0] << endl;
    cout << "At index 1: " << marks[1] << endl;
    cout << "At index 2: " << marks[2] << endl;
    cout << "At index 3: " << marks[3] << endl;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if
At index 0: 55
At index 1: 66
At index 2: 77
At index 3: 88
PS C:\Users\Aditya\Downloads\C plus plus>
```

If we declared size of array as n but added m elements (m>n) then error will come.

Example:

```cpp
//What if we declared size as 4 but given value more than 4
int main(){
    // a basic example
    int marks[4]= {55,66,77,88,70}; //array of size 4
    //accessing by index of array
    cout << "At index 0: " << marks[0] << endl;
    cout << "At index 1: " << marks[1] << endl;
    cout << "At index 2: " << marks[2] << endl;
    cout << "At index 3: " << marks[3] << endl;
    return 0;
}
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?
tut13.cpp: In function 'int main()':
tut13.cpp:30:34: error: too many initializers for 'int [4]'
     int marks[4]= {55,66,77,88,70}; //array of size 4
                                  ^
PS C:\Users\Aditya\Downloads\C plus plus>
```

Assigning value in array using the indexes:

Syntax:

Int Array_name[size_of_array];

Array_name[index_value] = value;

A basic example:

```cpp
int main(){
    int marks[4];
    marks[0]=55;
    marks[1]=95;
    marks[2]=75;
    marks[3]=45;
    cout << "At index 0: " << marks[0] << endl;
    cout << "At index 1: " << marks[1] << endl;
    cout << "At index 2: " << marks[2] << endl;
    cout << "At index 3: " << marks[3] << 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 ($?
At index 0: 55
At index 1: 95
At index 2: 75
At index 3: 45
PS C:\Users\Aditya\Downloads\C plus plus>
```

Can we change the value of array? Yes, we can do.

```cpp
//Changing the value of  array
int main(){
    int marks[4] = {5,6,7,8};
    cout << "At index 0: " << marks[0] << endl;
    cout << "At index 1: " << marks[1] << endl;
    cout << "At index 2: " << marks[2] << endl;
    cout << "At index 3: " << marks[3] << endl;
    marks[2] = 44; //new value at index 2
    cout << "At index 2 new value: " << marks[2] << endl;
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads
At index 0: 5
At index 1: 6
At index 2: 7
At index 3: 8
At index 2 new value: 44
PS C:\Users\Aditya\Downloads\C plus plus>
```

Printing the value of array using the for loops:

```
66  int main(){
67      int marks[4] ={4,5,6,7};
68      for (int i = 0; i < 4; i++)
69      {
70          cout << "The value at index " << i << " is " << marks[i] << endl;
71      }
72      return 0;
73  }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ;
The value at index 0 is 4
The value at index 1 is 5
The value at index 2 is 6
The value at index 3 is 7
PS C:\Users\Aditya\Downloads\C plus plus>
```

Using the do while loop:

```
75  //using the do while loop
76  int main(){
77      int marks[4] ={4,5,6,7};
78      int i=0;
79      do
80      {
81          cout << "The value at index " << i << " is " << marks[i] << endl;
82          i++;
83      } while (i<=3);
84      return 0;
85  }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?)
The value at index 0 is 4
The value at index 1 is 5
The value at index 2 is 6
The value at index 3 is 7
PS C:\Users\Aditya\Downloads\C plus plus>
```

Using the while loop:

```cpp
//Using the while loop
int main(){
    int marks[4] ={4,8,9,7};
    int i=0;
    while (i<=3)
    {
        cout << "The value at index " << i << " is " << marks[i] << endl;
        i++;
    }
    return 0;
}
```

```
PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\"
The value at index 0 is 4
The value at index 1 is 8
The value at index 2 is 9
The value at index 3 is 7
PS C:\Users\Aditya\Downloads\C plus plus>
```

How to use pointers with array?

In C++, arrays and pointers are closely related. The **name of an array acts like a pointer** to its first element. You can use pointers to **access and manipulate array elements** efficiently.

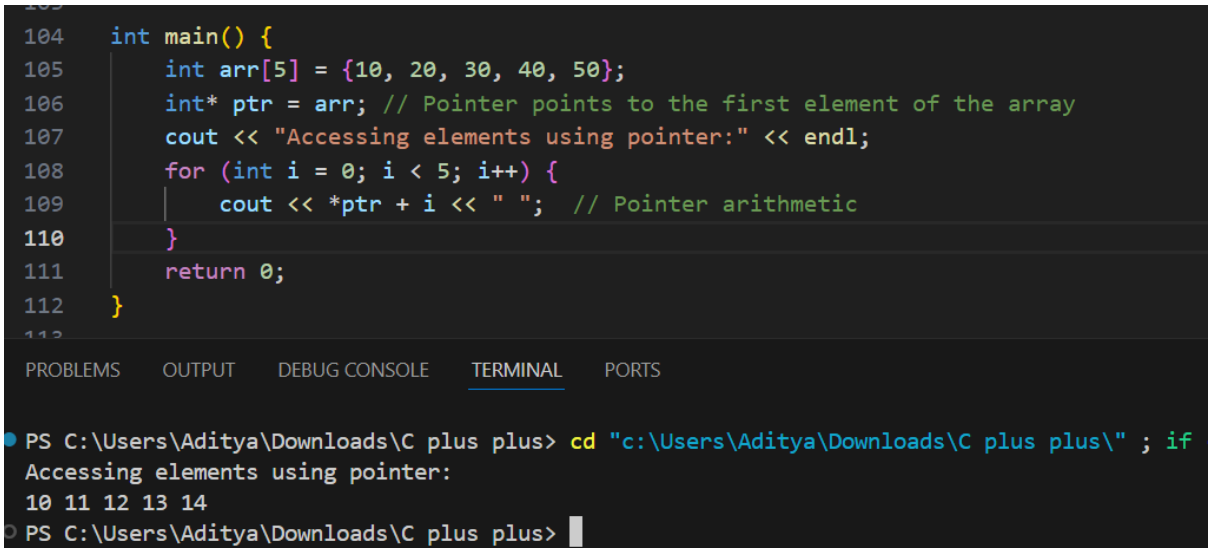
A basic example:

```cpp
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int* ptr = arr; // Pointer points to the first element of the array
    cout << "Accessing elements using pointer:" << endl;
    for (int i = 0; i < 5; i++) {
        cout << *(ptr + i) << " ";  // Pointer arithmetic
    }
    return 0;
}
```

```
PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) {
Accessing elements using pointer:
10 20 30 40 50
PS C:\Users\Aditya\Downloads\C plus plus>
```

Code	Meaning
arr	Points to &arr[0] (first element)
ptr + i	Moves i steps from the first element
*(ptr + i)	Dereferences to get value at index i

What if we remove those brackets?



What is Pointer Arithmetic in C++?

Pointer arithmetic refers to performing **arithmetic operations on pointers** to navigate through memory addresses—typically used with arrays. Since a pointer holds an address, you can move it forward/backward using operations like:

✅ **Valid Pointer Operations:**

Operation	Description
ptr + n	Moves pointer forward by n elements
ptr - n	Moves pointer backward by n elements
ptr1 - ptr2	Gives number of elements between two pointers
++ptr / --ptr	Increments/decrements pointer by one element

🚫 **Multiplication/Division** of pointers is **not allowed**.

A best example:

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

int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int* ptr = arr;
    cout << "First element: " << *ptr << endl;
    cout << "Second element: " << *(ptr + 1) << endl;
    cout << "Third element using increment: " << *(++ptr + 1) << 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\"
First element: 10
Second element: 20
Third element using increment: 30
PS C:\Users\Aditya\Downloads\C plus plus>
```

How It Works:

- Pointers move in steps of the **data type size** (e.g., int = 4 bytes).
- So ptr + 1 moves to the next **integer**, not just the next byte.
- *(ptr + i) gives value at index i

Above approach is same as below:

```cpp
int main() {
    // Declare an array 'arr' of 5 elements with values 10, 20, 30, 40, and 50
    int arr[5] = {10, 20, 30, 40, 50};
    // Declare a pointer 'ptr' and initialize it to point to the first element of 'arr'
    int* ptr = arr;
    // Dereference the pointer to access the first element (arr[0])
    cout << "First element: " << *ptr << endl;  // Output: 10
    // Dereference (ptr + 1) to access the second element (arr[1])
    cout << "Second element: " << *(ptr + 1) << endl;  // Output: 20
    // Dereference (ptr + 2) to access the third element (arr[2])
    cout << "Third element using (ptr + 2): " << *(ptr + 2) << endl;  // Output: 30
    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++ tut
First element: 10
Second element: 20
Third element using (ptr + 2): 30
PS C:\Users\Aditya\Downloads\C plus plus>
```

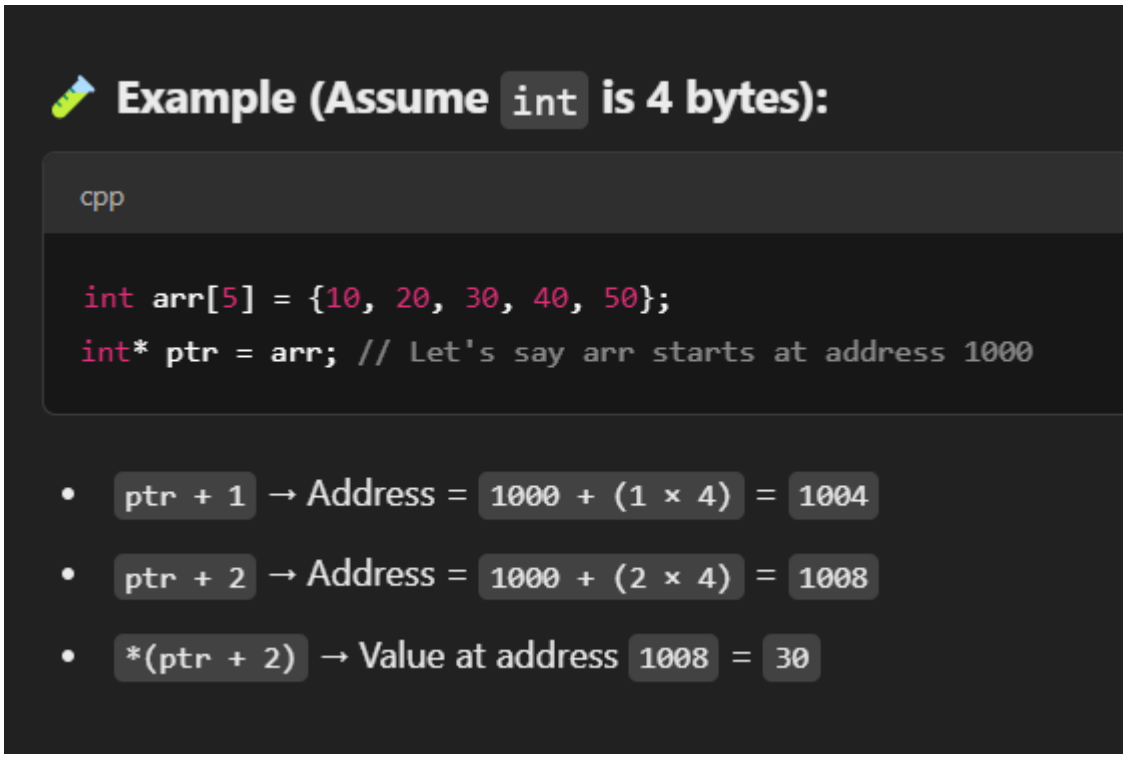
Formula of Pointer Arithmetic in C++

When performing arithmetic on pointers, the **actual memory address change** depends on the **size of the data type** the pointer refers to

New_Address = Current_Address + (n × Size_of_DataType)

Explanation:

- Current_Address: The current memory address held by the pointer.
- n: Number of elements to move forward (can be negative to move backward).
- Size_of_DataType: Size (in bytes) of the data type the pointer points to.



One more basic example:

```cpp
int main(){
    int arr[4] = {5,6,7,8};
    int* ptr = arr;
    cout << "At index 0" << *ptr << endl;
    cout << "At index 1" << *(ptr+1) << endl;
    cout << "At index 2" << *(ptr+2) << endl;
    cout << "At index 3" << *(ptr+3) << endl;
    cout << "Adding 1 to the value at 0th index " << *ptr+1 << endl;
    cout << "Adding 1 to the value at 3rd index " << *(ptr+3)+1 << endl;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ;
At index 05
At index 16
At index 27
At index 38
Adding 1 to the value at 0th index 6
Adding 1 to the value at 3rd index 9
PS C:\Users\Aditya\Downloads\C plus plus>
```

One important example:

```cpp
int main() {
    int arr[4] = {44, 55, 66, 77};  // An array with 4 elements: [44, 55, 66, 77]
    int* ptr = arr;  // Pointer 'ptr' points to the first element of 'arr' (44)
    cout << *ptr << endl;  // Dereference 'ptr' and print the value it points to (44)
    cout << *ptr + 1 << endl;  // Dereference 'ptr' to get 44, then add 1, prints 45
    cout << *(ptr + 1) << endl;  // Move the pointer to the next element (55), then print the value at that location (55)
    cout << *(ptr++) << endl;  // Print the current value 'ptr' is pointing to (44), then move 'ptr' to the next element
    cout << *ptr << endl;  // After the post-increment, 'ptr' now points to the second element (55), print that value
    cout << *(++ptr) << endl;  // Move 'ptr' to the next element (66) first, then dereference and print that value
    cout << *ptr << endl;  // Now 'ptr' points to the third element (66), print it
    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 ($?) { g++ tut13.cpp -o tut13 } ; if ($?) { .\tut1
44
45
55
44
55
66
66
PS C:\Users\Aditya\Downloads\C plus plus>
```

Video 14:

Structures, Unions & Enums in C++ | C++ Tutorials for Beginners #14

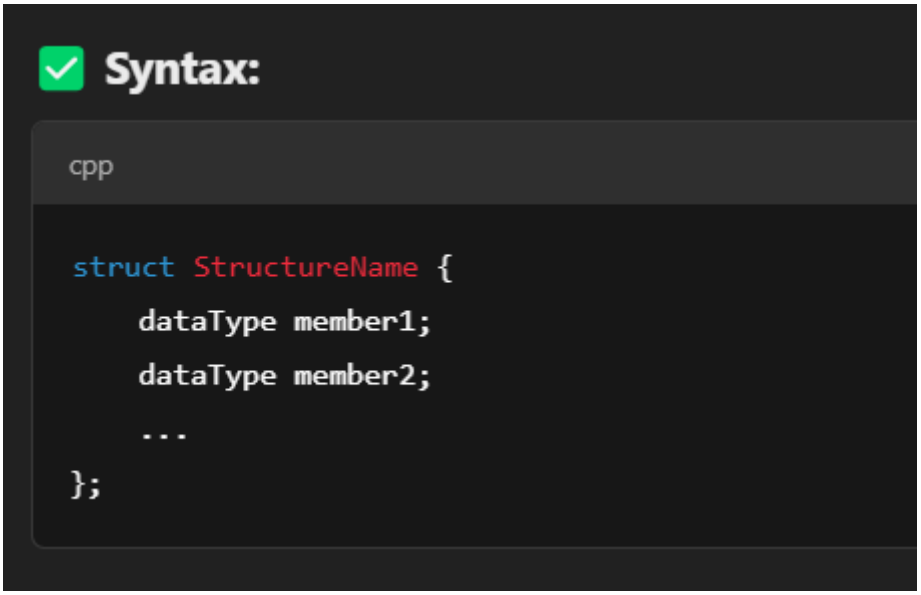
What are structures?

It is user defined datatype which is used to collect different data types.

A structure in C++ is a user-defined data type that groups variables of different data types under a single name. It is used to represent a record or composite object (e.g., a student having name, roll number, and marks).

Why Use Structures?

- Combine different data types (e.g., int, char[], float)
- Represent real-world entities (like employee, car, book)
- Useful in competitive programming and system-level design



A basic example:

```cpp

// Defining a structure named 'employee' to represent an employee's details
struct employee {
    int eId;       // Variable to store the employee's ID
    char favChar;  // Variable to store the employee's favorite character
    float salary;  // Variable to store the employee's salary
};

int main() {
    // Creating an instance of the 'employee' structure and naming it 'aditya'
    struct employee aditya;
    // Assigning values to the members of the 'aditya' structure
    aditya.eId = 1;           // Setting employee ID to 1
    aditya.favChar = 'c';     // Setting employee's favorite character to 'c'
    aditya.salary = 120000;   // Setting employee's salary to 120000
    // Printing the value of the 'favChar' member of 'aditya'
    cout << aditya.favChar << endl;  // Output will be 'c'
    return 0;  // Returning 0 to indicate successful execution of the program
}

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) {
c
PS C:\Users\Aditya\Downloads\C plus plus>
```

Using typedef with Structures in C++

The typedef keyword in C++ is used to create an alias (shorter name) for a data type—including structures. This simplifies syntax and improves code readability, especially when dealing with complex structure names.

✅ **Why Use typedef with struct?**

Without typedef, you must write struct StructName every time you declare a variable. With typedef, you can **skip struct** and just use the alias.

A basic example:

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

typedef struct {
    int rollNo;
    float marks;
} Student;  // 'Student' becomes an alias for this structure

int main() {
    Student s1;  // No need to write 'struct'
    s1.rollNo = 101;
    s1.marks = 92.5;

    cout << "Roll: " << s1.rollNo << ", Marks: " << s1.marks;
    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 ($?)
Roll: 101, Marks: 92.5
PS C:\Users\Aditya\Downloads\C plus plus>
```

Also,

✅ Alternate Style (Preferred in Modern C++):

cpp

```cpp
struct Student {
    int rollNo;
    float marks;
};
typedef Student Stud;  // Creating alias later

Stud s2;  // You can now use 'Stud' instead of 'struct Student'
```

What are unions?

A **union** in C++ is a **user-defined data type** similar to a structure, but with a key difference: **all members share the same memory location.** Only **one member can store a value at a time.** It's useful when you want to store different data types in the same memory location.

✅ Syntax:

```cpp
union UnionName {
    dataType member1;
    dataType member2;
    ...
};
```

🧠 **Key Concepts:**

- Only **one member holds a valid value** at any time.
- Memory size = size of the **largest member**.
- Useful in **memory optimization** or **hardware-related programming**.

Aspect	Structure (struct)	Union (union)
Memory Allocation	Separate memory for each member	Shared memory for all members
Total Size	Sum of sizes of all members	Size of the largest member only
Member Access	All members can hold values simultaneously	Only one member holds a valid value at a time
Use Case	When multiple data fields are needed at once	When one field is used at a time (e.g., memory saving)
Data Overlap	No data overlap; values stored independently	Data overlap; one member's change affects others

✅ **Example Use Case:**

- **Structure**: Representing a student with name, age, marks (all needed together).
- **Union**: Representing a value that can either be int, float, or char, but only one at a time (like variant data).

A basic example:

```cpp
//Union creation
union money
{
    int eId;
    char favChar;
    float salary;
};

int main(){
    union money m1;
    m1.eId = 22;
    cout << m1.eId;
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads
22
PS C:\Users\Aditya\Downloads\C plus plus>
```

What if we try to assign the salary? It will give garbage value if tried to be printed the eId.

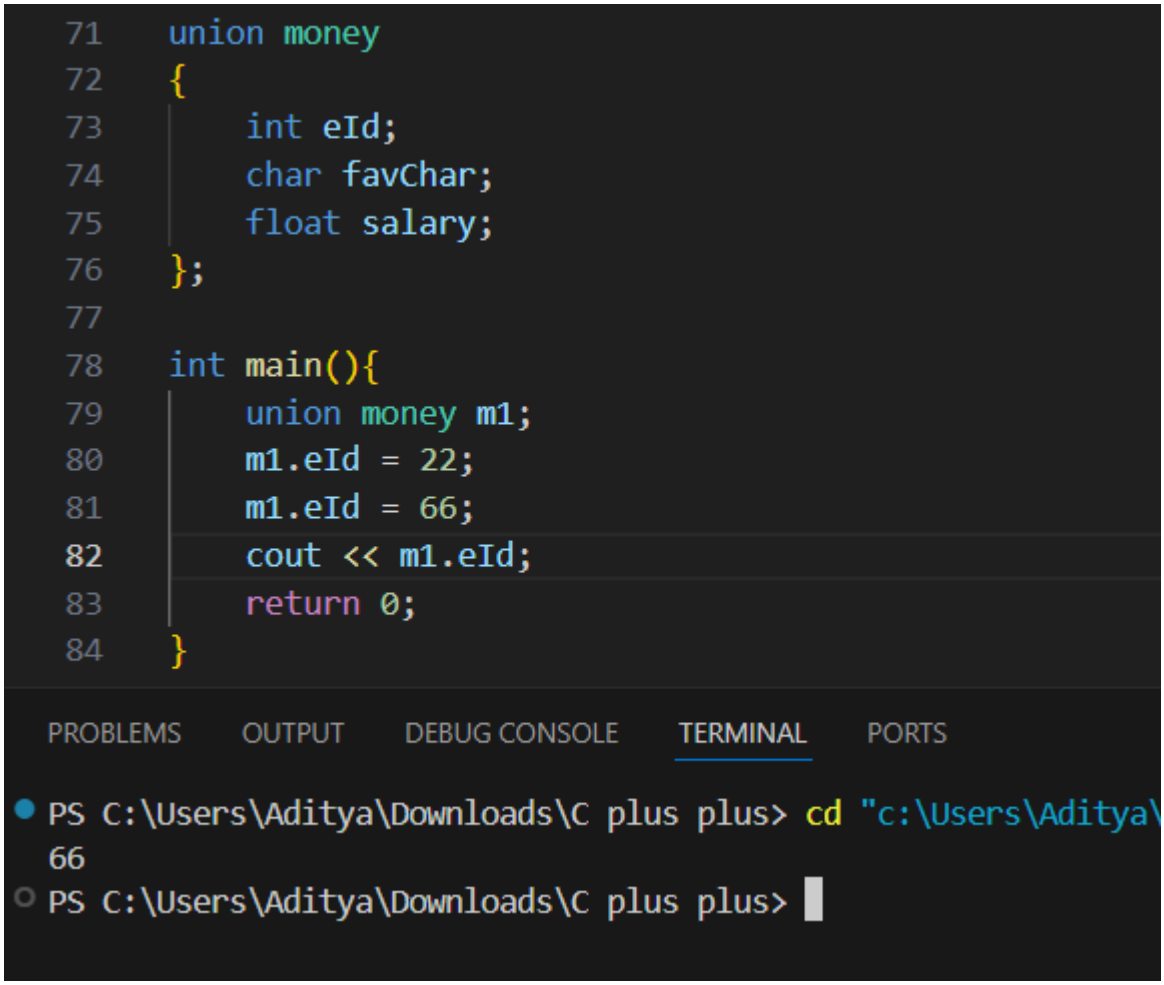
```cpp
union money
{
    int eId;
    char favChar;
    float salary;
};

int main(){
    union money m1;
    m1.eId = 22;
    m1.salary = 4545;
    cout << m1.eId;
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus pl
22
PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus pl
1166936064
```

```cpp
union money
{
    int eId;
    char favChar;
    float salary;
};

int main(){
    union money m1;
    m1.eId = 22;
    m1.salary = 4545;
    //cout << m1.eId; //will give garbage value
    cout << m1.salary;
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downlo
4545
PS C:\Users\Aditya\Downloads\C plus plus>
```


71 union money
72 {
73 int eId;
74 char favChar;
75 float salary;
76 };
77
78 int main(){
79 union money m1;
80 m1.eId = 22;
81 m1.eId = 66;
82 cout << m1.eId;
83 return 0;
84 }
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\
66
PS C:\Users\Aditya\Downloads\C plus plus>

One example more:

```cpp
union Data {
    int i;
    float f;
    char ch;
};

int main() {
    Data d;
    d.i = 10;
    cout << "Integer: " << d.i << endl;
    d.f = 5.5;
    cout << "Float: " << d.f << endl;
    d.ch = 'A';
    cout << "Char: " << d.ch << endl;
    // Now d.i and d.f are likely overwritten
    cout << "Integer after char: " << d.i << 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
Integer: 10
Float: 5.5
Char: A
Integer after char: 1085276225
PS C:\Users\Aditya\Downloads\C plus plus>
```

What is enum in C++?

An **enum (enumeration)** is a **user-defined data type** that assigns names to a set of integral constants, starting from 0 by default. It improves **code readability**, avoids **magic numbers**, and is commonly used to represent fixed sets of related values like days, directions, states, etc.

🎯 **Importance & Use of enum:**

Feature	Benefit
Improves clarity	Named values make code easier to understand.
Reduces bugs	Avoids hard-coded numbers ("magic numbers").
Easier debugging	You know what 2 or 3 stands for via names.
Scoped enums	In modern C++ (enum class), avoids name clashes.

Use in Competitive Programming:

- Use enum when your code has **multiple fixed states or options**, like:
  - Directions: UP, DOWN, LEFT, RIGHT
  - Status: ALIVE, DEAD
  - Modes: EASY, MEDIUM, HARD
- Replaces multiple #define or const int values.
- Helps in **state machines**, **switch cases**, or **directional arrays**.

✅ **Where to Use enum:**

- Game logic (player states, moves)
- Grid-based problems (directions)
- Status flags (input/output states, parsing modes)
- Finite State Machines (FSM)

A basic example:

```
112  enum meal {breakfast, lunch, dinner};
113
114  int main(){
115      cout << breakfast <<endl; //first of it be indexed 0
116      cout << lunch <<endl; //second of it be indexed 0
117      cout << dinner <<endl; //third of it be indexed 0
118      return 0;
119  }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\"
0
1
2
PS C:\Users\Aditya\Downloads\C plus plus>
```

```
121  //Writing Enum Inside ide the main() function
122  int main(){
123      enum meal {breakfast, lunch, dinner};
124      cout << breakfast <<endl; //first of it be indexed 0
125      cout << lunch <<endl; //second of it be indexed 1
126      cout << dinner <<endl; //third of it be indexed 2
127      return 0;
128  }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($
0
1
2
PS C:\Users\Aditya\Downloads\C plus plus>
```

One standard example:

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

enum Direction { NORTH, EAST, SOUTH, WEST };

int main() {
    Direction dir = SOUTH;
    cout << "Direction: " << dir << endl;  // Output: 2 (since SOUTH is at index 2)
    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++ t
Direction: 2
PS C:\Users\Aditya\Downloads\C plus plus>
```

Video 15:

Functions & Function Prototypes in C++ | C++ Tutorials for Beginners #15

What are functions?

Functions in C++ are **blocks of reusable code** designed to perform a specific task. They allow **modular programming**, making code **cleaner**, **easier to debug**, and **reusable**. Functions can take inputs (called parameters), perform operations, and return results.

✅ **Why Use Functions?**

- Break complex problems into smaller parts
- Avoid code repetition
- Improve readability and maintainability
- Easier to test/debug

✅ **Basic Syntax of a Function:**

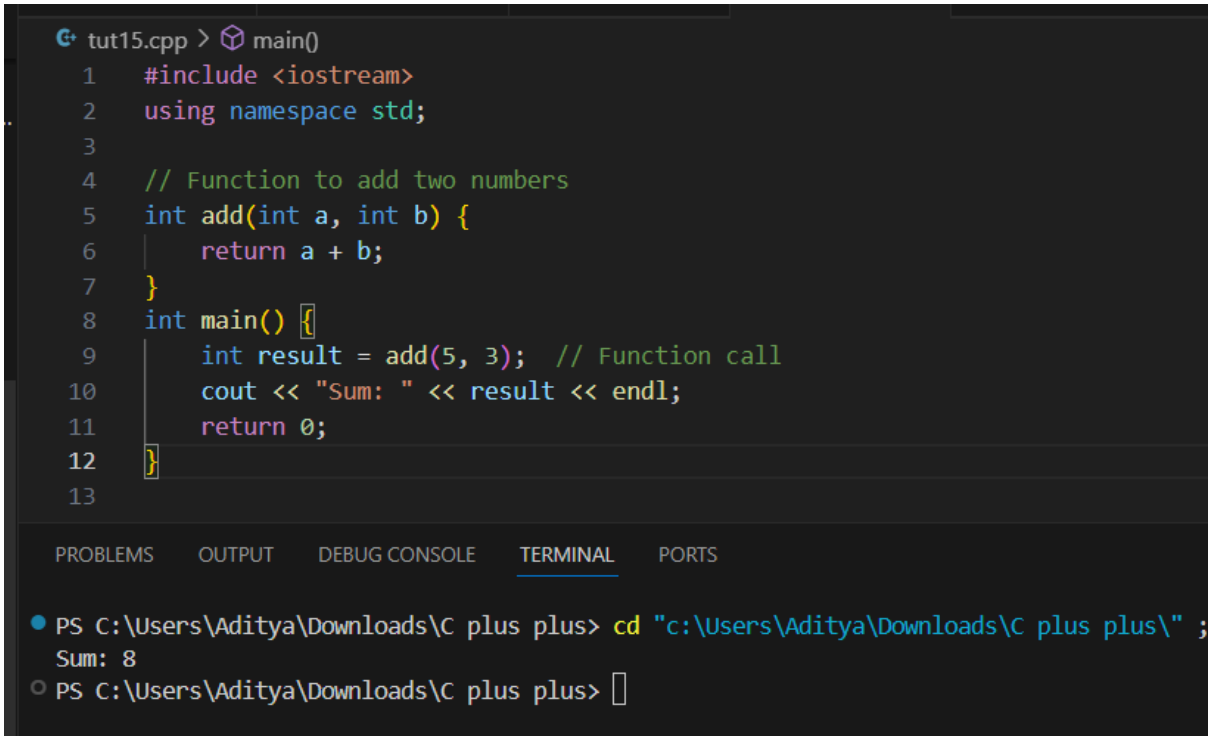
cpp

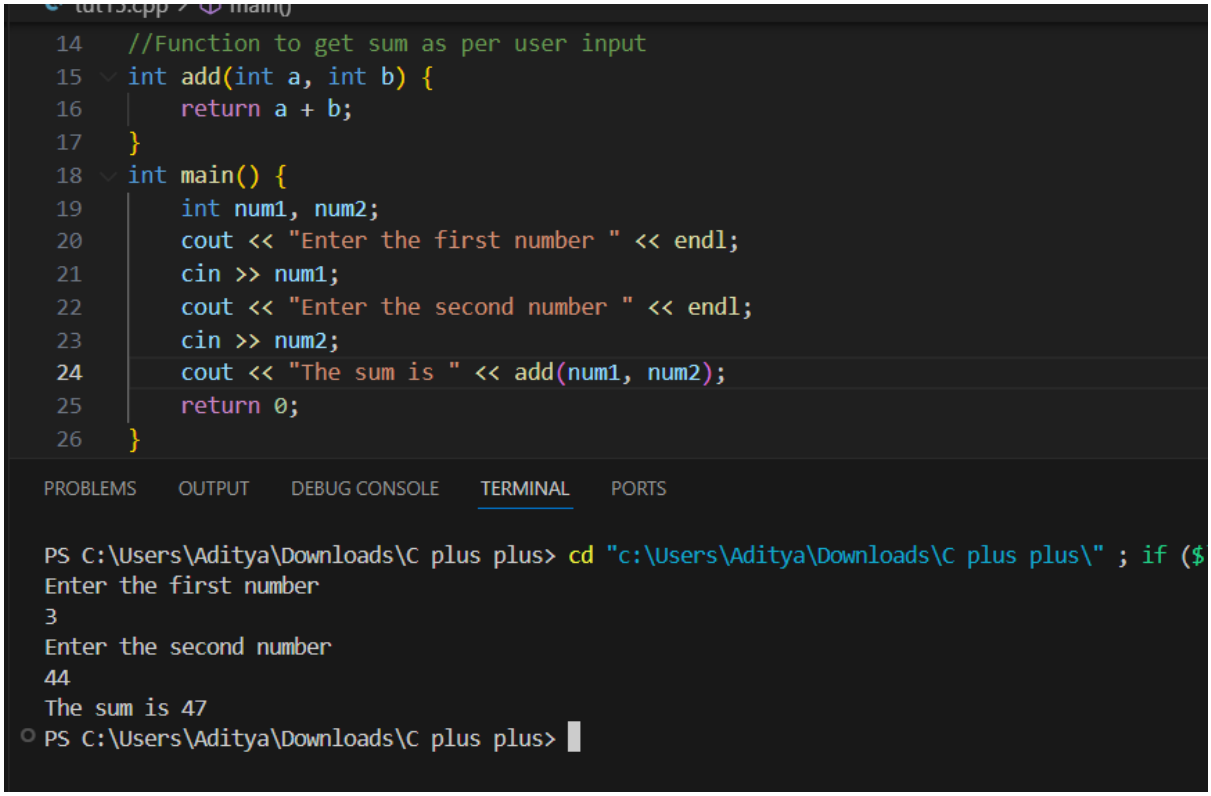
```cpp
return_type function_name(parameter_list) {
    // code to execute
    return value; // if return type is not void
}
```

🔍 **Types of Functions:**

Type	Example
Built-in Functions	cout, sqrt(), etc.
User-defined Functions	int add(int a, int b)
Void Functions	Functions with no return
Parameterised/Non-parameterised	Accept arguments or not

A basic example:





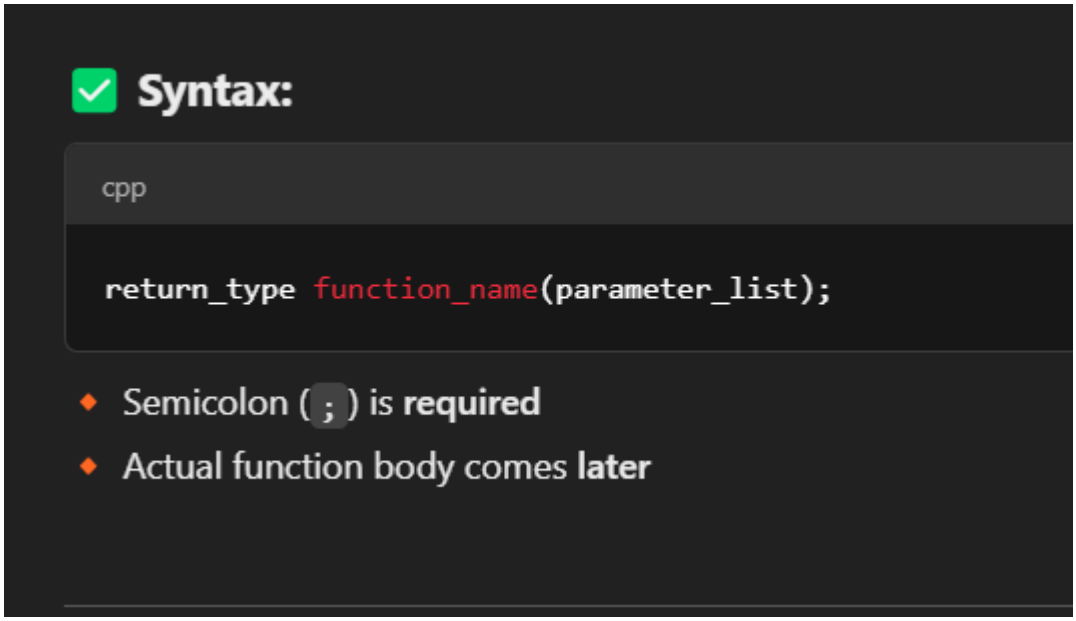
What if add() is written after main() in C++?

If you write the function **after main()**, the compiler **won't know** what add() is when it encounters it during the first pass—**unless** you provide a **function declaration (prototype) before** main().

How to tackle this issue? Use function prototype.

What Are Function Prototypes in C++?

A function prototype is a declaration of a function that tells the compiler the function's name, return type, and parameters before its actual definition. It allows the function to be called before it is defined in the code.



Why Use Function Prototypes?

- Allows calling functions before their definition
- Helps catch errors early (wrong number or type of arguments)
- Supports better code organization (like defining functions at the end)

A basic example:

```cpp
//Function prototypes
int add(int,int); //Acceptable
//int sum(int a,b); //Not acceptable
//int sum(a,b) // not acceptable
//int add(int a, int b); //acceptable
int main() {
    int num1, num2;
    cout << "Enter the first number " << endl;
    cin >> num1;
    cout << "Enter the second number " << endl;
    cin >> num2;
    cout << "The sum is " << add(num1, num2);
    return 0;
}
int add(int a, int b) {
    return a + b;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ;
Enter the first number
44
Enter the second number
5
The sum is 49
PS C:\Users\Aditya\Downloads\C plus plus>
```

What are Formal and Actual Parameters in C++?

In C++, when a function is called, values are passed to it using **parameters**. These are classified into:

Type	Definition
Formal Parameters	The variables defined in the function **definition** that receive values.
Actual Parameters	The values or variables passed in the **function call**.

```cpp
int add(int,int); //Acceptable
//int sum(int a,b); //Not acceptable
//int sum(a,b) // not acceptable
//int add(int a, int b); //acceptable

int main() {
    int num1, num2; //num1 and num2 are actual parameters
    cout << "Enter the first number " << endl;
    cin >> num1;
    cout << "Enter the second number " << endl;
    cin >> num2;
    cout << "The sum is " << add(num1, num2);
    return 0;
}
int add(int a, int b) {
    return a + b; // a and b are formal parameters
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; if ($?) { g
Enter the first number
4
Enter the second number
2
The sum is 6
PS C:\Users\Aditya\Downloads\C plus plus>
```

Can Actual and Formal Parameter Names Be the Same in C++?

✅ **Yes**, the names of actual and formal parameters **can be the same** in C++.
However, they are stored in **different memory locations**, so there's **no conflict**.

What is the Significance of return in a Function (excluding main())?

The return statement in a function is used to:

1. **Send back a result** from the function to the calling code.
2. **Exit** the function immediately, ending its execution.
3. **Pass control and value** (if any) back to the caller.

🧹 Example with `return`:

cpp Copy Edit

```cpp
int square(int n) {
    return n * n;  // returns the square to caller
}
```

Can a Function Work Without return?

✅ **Yes**, if the function's **return type is void**, it does **not need a return statement with a value**.

A basic example;

```
47  void greet(void); //--> acceptable
48
49  int main(){
50      greet();
51      return 0;
52  }
53
54  void greet(){
55      cout << "Hello Aditya";
56  }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus>
```

```
47  //void greet(void); //--> acceptable
48
49  void greet(); // --> Acceptable
50  int main(){
51      greet();
52      return 0;
53  }
54
55  void greet(){
56      cout << "Hello Aditya";
57  }
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C
Hello Aditya
PS C:\Users\Aditya\Downloads\C plus plus>
```

```
50  void show(); // --> Acceptable
51  int main(){
52      show();
53      return 0;
54  }
55
56  void show() {
57      cout << "Before return" << endl;
58      return;  // just exits the function, no value returned
59  }
60
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Aditya\Downloads\C plus plus> cd "c:\Users\Aditya\Downloads\C plus plus\" ; it
Before return
PS C:\Users\Aditya\Downloads\C plus plus>
```

✅ **Explanation:**

- The function **show() doesn't return a value**.
- It just **prints something to the console** using cout.
- The return; statement **just exits the function**, it **does NOT send data back** to the caller.
- The output seen (Before return) is from cout, not from a return.