# What is an Separators?

**Separators** are **symbols** used to **separate different parts** of a C++ program, like statements, parameters, and blocks of code. They help **structure** and **organize** the code properly.

## Types of Separators :

**1.Semicolon (;)**

- Ends a statement
- **Example**: int a = 10;

**2.Comma (,)**

- Separates multiple variables or arguments
- **Example**: int x = 5, y = 10;

**3.Parentheses (( ))**

- Used in function calls, condition checks, loops
- **Example**: if (x > 0), sum(a, b)

**4.Braces ({ })**

- Define the beginning and end of a block of code
- **Example**:{

  }

**5.Brackets ([ ])**

- Used for array declarations and indexing
- **Example**: arr[0] = 5;

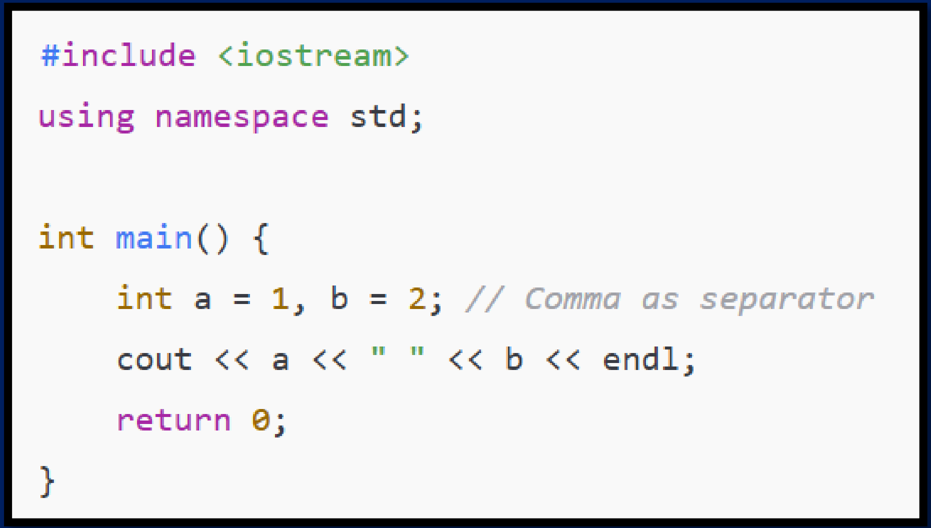
**6.Colon (:)**

- Used in labels (for case in switch or inheritance)
- **Example**: case 1:, class B : public A

**7.Hash (#)**

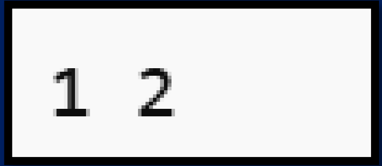
- Used for preprocessor directives
- **Example**: #include <iostream>

## Example 1:



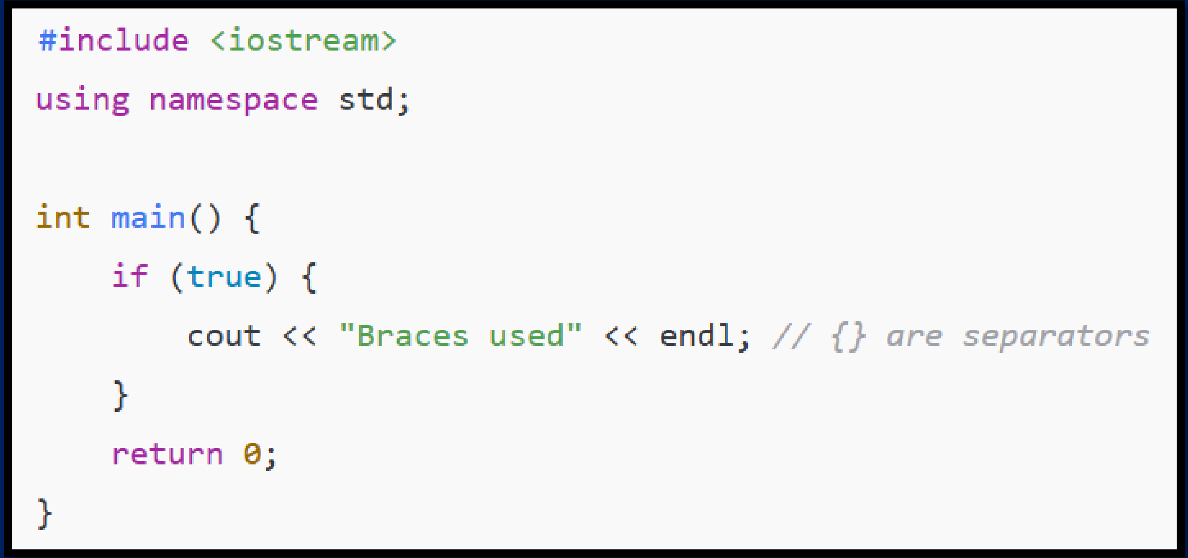
```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 1, b = 2; // Comma as separator
    cout << a << " " << b << endl;
    return 0;
}
```



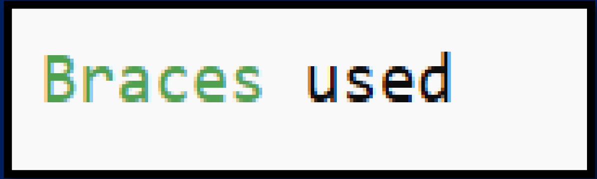
```
1 2
```

## Example 2 :



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

int main() {
    if (true) {
        cout << "Braces used" << endl; // {} are separators
    }
    return 0;
}
```



```
Braces used
```

## Data Types

A **data type** tells the **compiler** what kind of data a variable can hold — such as **integers, characters, floating-point numbers**, etc.

### Why important?

Because it defines:

- ✓ The **type of data** stored
- ✓ The **memory size**
- ✓ The **operations** that can be performed

### Classification of Data Types

- ❖ **Primary** : int, float, double, char, bool, void
- ❖ **Derived** : Arrays, Functions, Pointers, References.
- ❖ **User-defined** : Structures (struct), Unions (union), Classes

## Primary(Built-in)

| Data Type | Size (Bytes) | Range | Default Value | Example |
|---|---|---|---|---|
| int | 4 | –2,147,483,648 to 2,147,483,647 | 0 | int age = 25; |
| short | 2 | –32,768 to 32,767 | 0 | short temp = 100; |
| long | 4 or 8 | Larger than int | 0 | long distance = 100000; |
| unsigned int | 4 | 0 to 4,294,967,295 | 0 | unsigned int u = 50; |
| float | 4 | 1.2E–38 to 3.4E+38 (6–7 digits precision) | 0.0 | float pi = 3.14f; |
| double | 8 | 2.3E–308 to 1.7E+308 (15–16 digits precision) | 0.0 | double g = 9.81; |
| char | 1 | ASCII 0 to 127 | '\0' (null) | char grade = 'A'; |
| bool | 1 | true or false | false | bool isPass = true; |
| void | 0 | No data | N/A | void display(); |

## Derived

Derived data types are **based on primary (built-in)** types and provide **more complex ways to work with data**, such as collections, addresses, or functions.

| Data Type | Description | Syntax Example | Use Case |
|---|---|---|---|
| Array | Collection of fixed-size elements of same type | int marks[5]; | Store multiple values under one name |
| Pointer | Stores the memory address of another variable | int *ptr = &num; | Dynamic memory, passing by address |
| Function | Group of code that performs a task | int add(int a, int b); | Code reuse, modular programming |
| Reference | An alias for an existing variable | int &ref = original; | Modify original variable via alias |

## User-Defined

User-defined data types allow programmers to **create their own types** by combining existing data types to represent **real-world entities**.

| **Type** | **Keyword** | **Description** | **Example Usage** |
|---|---|---|---|
| Structure | struct | Combines variables of different types under one name | struct Student { ... }; |
| Union | union | Similar to struct, but shares same memory | union Data { ... }; |
| Enum | enum | Used to define named integer constants | enum Color { RED, GREEN }; |
| Class | class | Defines objects and behaviors using OOP | class Car { ... }; |
| Typedef | typedef | Creates alias/nickname for data types | typedef int Marks; |
| Using | using | Modern version of typedef (C++11+) | using Age = int; |

## What is a Variable?

A **variable** is a **named storage location** in memory that holds a **value** which can **change** during program execution.

### Why Do We Use Variables?

- ❖ To **store input** from users
- ❖ To **perform calculations**
- ❖ To **track changing values** during a program

### Rules for Naming Variables (Identifiers):

- ❖ Must start with a **letter** (A–Z or a–z) or underscore (_)
- ❖ Can include **letters**, **digits**, and **underscores**
- ❖ Cannot use **C++ keywords** (like int, if, return)
- ❖ Cannot contain **spaces** or **special characters** (@, #, $)
- ❖ Are **case-sensitive** (Age ≠ age)

## Syntax :

```
data_type    variable_name    =    value;
```

## Example :

```
int age = 25;
float salary = 45000.50;
```

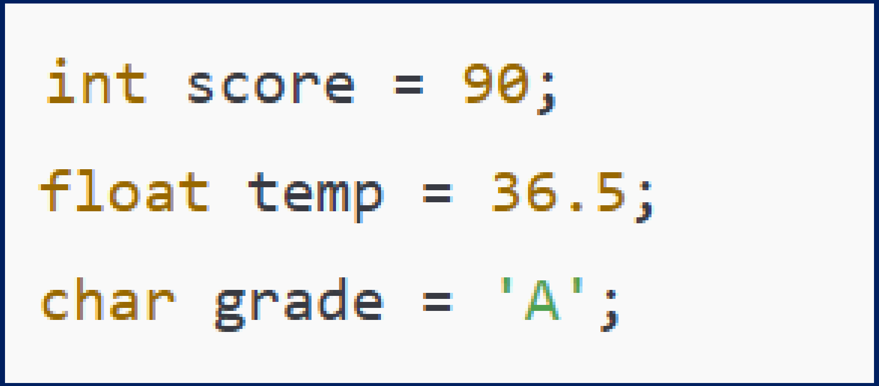
| Type | Syntax | Example | Description |
|---|---|---|---|
| Declaration Only | data_type variable_name; | int age; | Declares a variable without assigning a value |
| Declaration with Initialization | data_type variable_name = value; | float pi = 3.14; | Declares and assigns a value at the same time |
| Multiple Declarations | data_type var1 = val1, var2 = val2; | int x = 5, y = 10; | Declare and initialize multiple variables in one line |
| Initialization Later | First declare, then assign value later | int num; num = 100; | Useful when value is unknown at the time of declaration |
| Constant Initialization | const data_type var = value; | const int max = 50; | Value cannot be changed once assigned |

## Types of Variable Storage

- **Single Variable :** Holds one value
- **Multiple Variable :** Holds Many values

### Single Value Variables

- **Store only one piece of data per variable**
- **Simple and useful for basic operations**



```
int score = 90;
float temp = 36.5;
char grade = 'A';
```

## Multiple Value Variables

| Technique | Description | Example |
|---|---|---|
| Array | Group of similar values | int marks[3] = {90, 85, 78}; |
| Structure | Group of different types under one name | struct Student {<br>    int roll;<br>    float marks;<br>};<br><br>Student s1 = {101, 88.5}; |
| Class | Object with variables and functions | class Product {<br>public:<br>    int id;<br>    string name;<br>    float price;<br>}; |
| Vector | Dynamic version of array (C++ STL) | #include &lt;vector&gt;<br>vector&lt;int&gt; scores = {90, 85, 95}; |
| Array of Struct | Multiple structured records | Student s[10]; |

OTZ

OPENTECHZ

A dream to realize your dream

# Thank You