C++ Intermediate

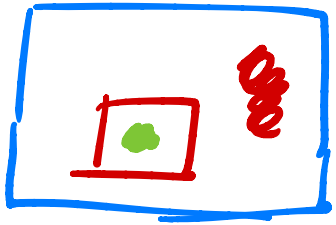
- Viraj Chandra

Goal

To understand:

- Scope
- Loops
- Nested Loops
- Arrays / Strings

Scope

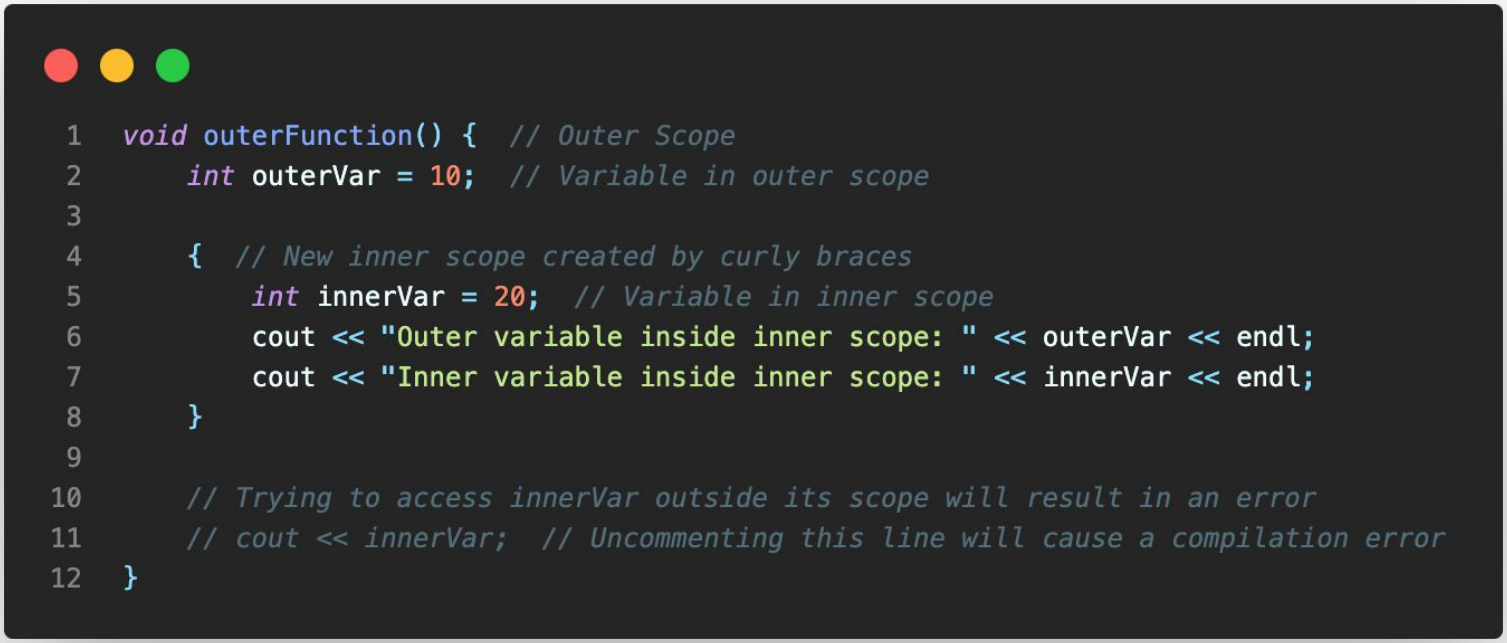


A scope is a **region** of the program.

Every pair of curly braces creates a new scope.

The variables inside the scope cannot be used outside the scope, but the variables outside the scope can be used inside.

Scope



```cpp
void outerFunction() {  // Outer Scope
    int outerVar = 10;  // Variable in outer scope

    {  // New inner scope created by curly braces
        int innerVar = 20;  // Variable in inner scope
        cout << "Outer variable inside inner scope: " << outerVar << endl;
        cout << "Inner variable inside inner scope: " << innerVar << endl;
    }

    // Trying to access innerVar outside its scope will result in an error
    // cout << innerVar;  // Uncommenting this line will cause a compilation error
}
```

Loop

{ if (n == 10) continue;
}

Loops are used to **repeat a block of code** until some condition is satisfied.

- An **iteration** in loop is defined as one time the loop gets executed. For example, 3rd iteration is the 3rd time the loop is run.
- **“break”** statement exits the current/innermost loop when executed.
- **“continue”** statement skips to the next iteration of the current/innermost loop when executed.

Types of Loops

There are three types of loops in C++:

- for loop
- while loop
- do-while loop

for loop, while loop → entry controlled

do-while loop → exit controlled

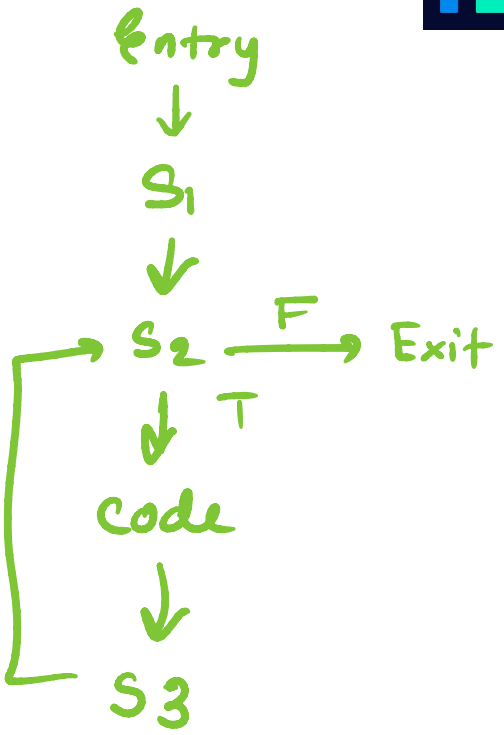
“for” Loop

- s1: Initialization of the loop.
- s2: Condition of the loop. Loop exits if false.
- s3: Executed after each iteration.

Syntax:-

```
for (s1; s2; s3) {
    // Code here
}
```

int i=1 ; i<=5 ; i++



“while” Loop

- Check if the condition is true and then execute the block of code.
- Repeat till condition becomes false, and exit.

→ S1

Syntax:-

```
while (condition) {
  // Code here
}
```

S3

→ S2

Use while over for – dynamic power over the control flow logic

“do while” Loop

- Execute the block of code first.
- Then check if the condition is true.
- Repeat the process as long as the condition remains true.

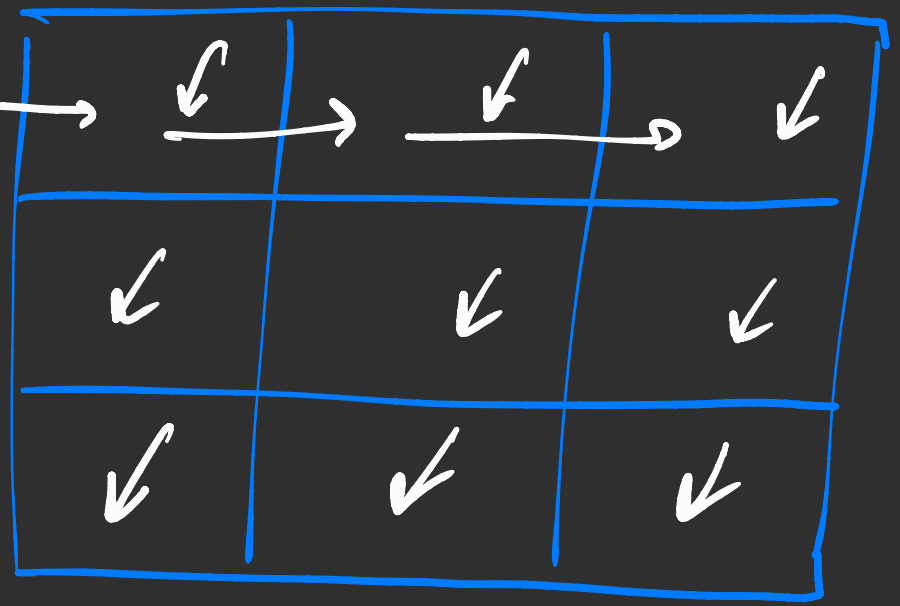
Syntax:-

```
do {
    // Code here
} while (condition);
```

Nested Loops

- Nested loops are **loops placed inside another loop**.
- **The inner loop** executes completely for **each iteration of the outer loop.**
- They are commonly used for tasks involving multi-dimensional data, such as **matrices or grids.**

```
for(   )
{
  for(   )
    row
}
```



```
for( int i=1 ; i<=3 ; i++)   → outer loop
{
    for( int j=1 ; j<=3 ; j++)   → inner loop
    {
        for( int l=1 ; l<=3 ; l++)
    }
}
```

i	j	
1	1	3 1
1	2	3 2
1	3	3 3
2	1	
2	2	
2	3	

Nested Loops

1×1 = 1 1×2 = 2
1×3 = 3 1×4 = 4 1×5 = 5

Example:
Multiplication Table

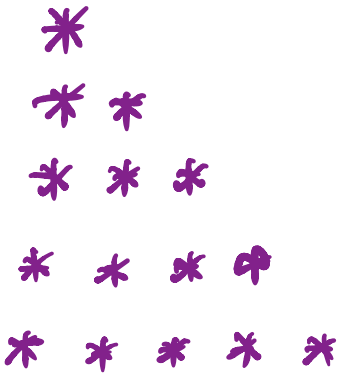
i=1 j=1
i=1 j=2

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

int main() {
    for (int i = 1; i <= 5; i++) {
        for (int j = 1; j <= 5; j++) {
            cout << i << " x " << j << " = " << i * j << "\t";
        }
        cout << endl; // Move to the next line after inner loop
    }
    return 0;
}
```

Nested Loops

Example:
Printing Pattern



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

int main() {
    int n = 5; // Number of rows
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            cout << "* "; // Print a star
        }
        cout << endl; // Move to the next row
    }
    return 0;
}
```

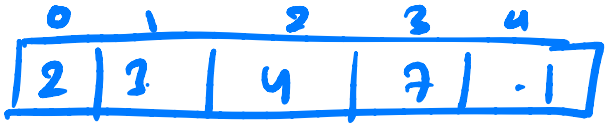
```
for( int i=1 ; i<=5 ; i++)
{
    for(int j=1 ; j<=i ; j--)
    { cout << "*  " ; }
    cout<<endl;
}
```

i	j
1	1
2	1
2	2
3	1
3	2
3	3

Nested Loops

- Ensure the **termination conditions** for both loops are well-defined to avoid infinite loops.
- Try to minimize the **depth of nested loops** when possible for better performance and readability.

0 to n-1 index —



Arrays

1D Category

An array is a collection of **multiple items of the same datatype.**

- Arrays are ordered, meaning they are continuous in memory.
- The size of an array cannot be changed, once declared.

Syntax:-

```
datatype name[size]
```

```
int a[5];
```

Input an Array

To input an array in C++, we need to:

- Declare the array with a specified size.
- Use a loop to take input for each element, using **cin**

Syntax:-

```
int a[5];
for (int i = 0; i < 5; i++) {
    cin >> a[i];
}
```

Output an Array

To output an array in C++, you need to:

- Use a loop to traverse through the elements.
- Print each element, using **cout**

Segmentation fault

Out of bounds

Syntax:-

```
int a[5] = {1, 2, 3, 4, 5};
for (int i = 0; i < 5; i++)
{
    cout << a[i] << " ";
}
cout << endl;
```

Challenge Yourself

- **Factorial** of a Number

  Example: N = 5

  Result: 1*2*3*4*5 = 120

- Find the **largest number** in an array.

- Print all **possible pairs** of integers in an array with **distinct integers.**

  Example: Array = [1, 2, 3]

  Result: (1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)