Programming Fundamental

Lab#12

Table of Contents

Scope of Variable:

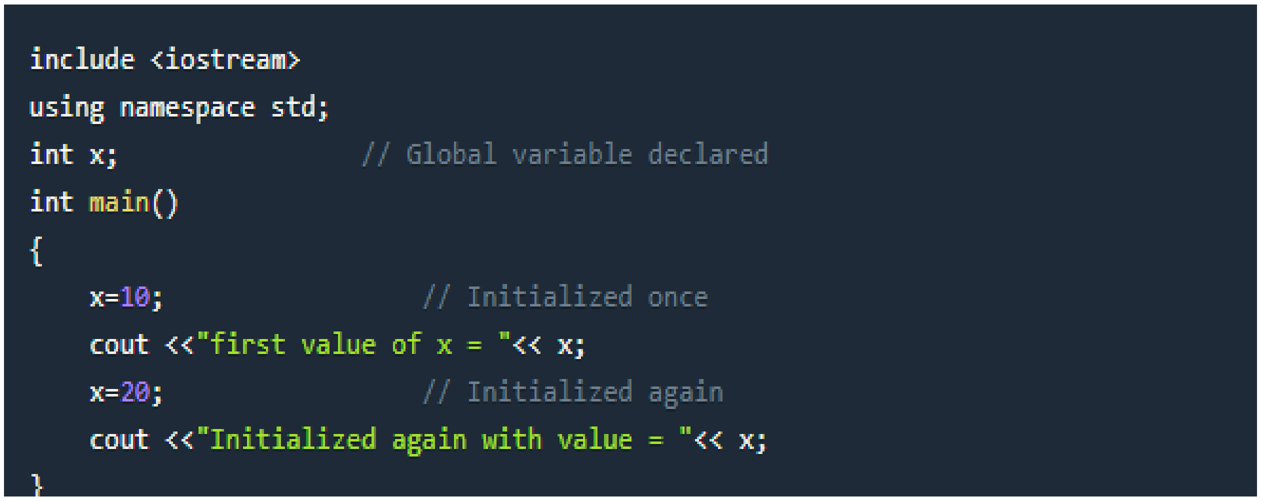
All the variables have their area of functioning, and out of that boundary they don't hold their value, this boundary is called scope of the variable

- Global Variables
- Local variables

Global Variables

Global variables are those, which are once declared and can be used throughout the lifetime of the program by any class or any function. They must be declared outside the main() function. If only declared, they can be assigned different values at different time in program lifetime. But even if they are declared and initialized at the same time outside the main() function, then also they can be assigned any value at any point in the program.

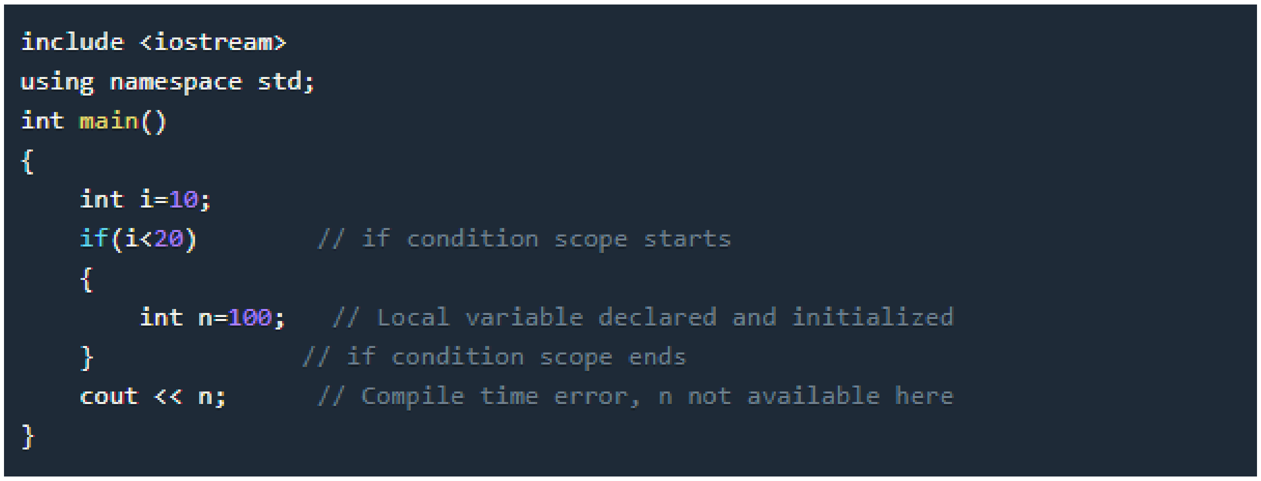
For example: Only declared, not initialized



```
include <iostream>
using namespace std;
int x;                  // Global variable declared
int main()
{
    x=10;                   // Initialized once
    cout <<"first value of x = "<< x;
    x=20;                   // Initialized again
    cout <<"Initialized again with value = "<< x;
}
```

Local variables

Local variables are the variables which exist only between the curly braces, in which its declared. Outside that they are unavailable and leads to compile time error.



```
include <iostream>
using namespace std;
int main()
{
    int i=10;
    if(i<20)        // if condition scope starts
    {
        int n=100;    // Local variable declared and initialized
    }               // if condition scope ends
    cout << n;      // Compile time error, n not available here
}
```

Pass by value Vs Pass by reference

Pass by value means the actual **value** is passed on. **Pass by reference** means that a number (called a memory address) is passed on, this address defines where the **value** is stored.

```
void swapNums(int &x, int &y) // & mean pass by reference

{
  int z = x;
  x = y;
  y = z;
}

int main() {
  int firstNum = 10;
  int secondNum = 20;

  cout << "Before swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  // Call the function, which will change the values of firstNum and
secondNum
  swapNums(firstNum, secondNum);

  cout << "After swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  return 0;
}
```

Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

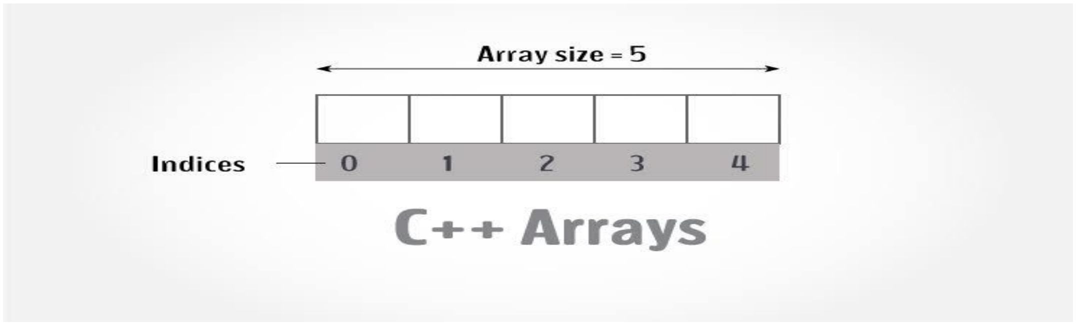
To declare an array, define the variable type, specify the name of the array followed by **square brackets** and specify the number of elements it should store:

To create an array of five integers, you could write:

```
int myNum[5] = {10, 20, 30, 40, 50};
```

Access the Elements of an Array

To access the element you have to provide the index number along with the name.



```
cout << myNum [0];
```

Change an Array Element

To change the value of a specific element, refer to the index number:

Example

```
myNum [2] = 4000
```

Loop through an Array

You can loop through the array elements with the for loop.

The following example outputs all elements in the **cars** array:

```
for(int i = 0; i < 4; i++)

{
  cout <<myNum[i] <<endl;
}
```

Advantages of an Array in C/C++:

1. Random access of elements using array index.
2. Use of less line of code as it creates a single array of multiple elements.
3. Easy access to all the elements.
4. Traversal through the array becomes easy using a single loop.
5. Sorting becomes easy as it can be accomplished by writing less line of code.

Disadvantages of an Array in C/C++:

1. Allows a fixed number of elements to be entered which is decided at the time of declaration. Unlike a linked list, an array in C is not dynamic.
2. Insertion and deletion of elements can be costly since the elements are needed to be managed in accordance with the new memory allocation.