

## مدخل إلى الخوارزميات والبرمجة هندسة الميكاترونكس سنة أولى

2023-2024

مدرس المقرر: د. عيسى الغنام

### Lecture No.6

### C++ For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

Syntax

```
for (statement 1; statement 2; statement 3) {  
    // code block to be executed  
}
```

**Statement 1** is executed (one time) before the execution of the code block.

**Statement 2** defines the condition for executing the code block.

**Statement 3** is executed (every time) after the code block has been executed.

The example below will print the numbers 0 to 4:

#### Example

```
for (int i = 0; i < 5; i++) {  
    cout << i << "\n";  
}
```

Example explained

Statement 1 sets a variable before the loop starts (int i = 0).

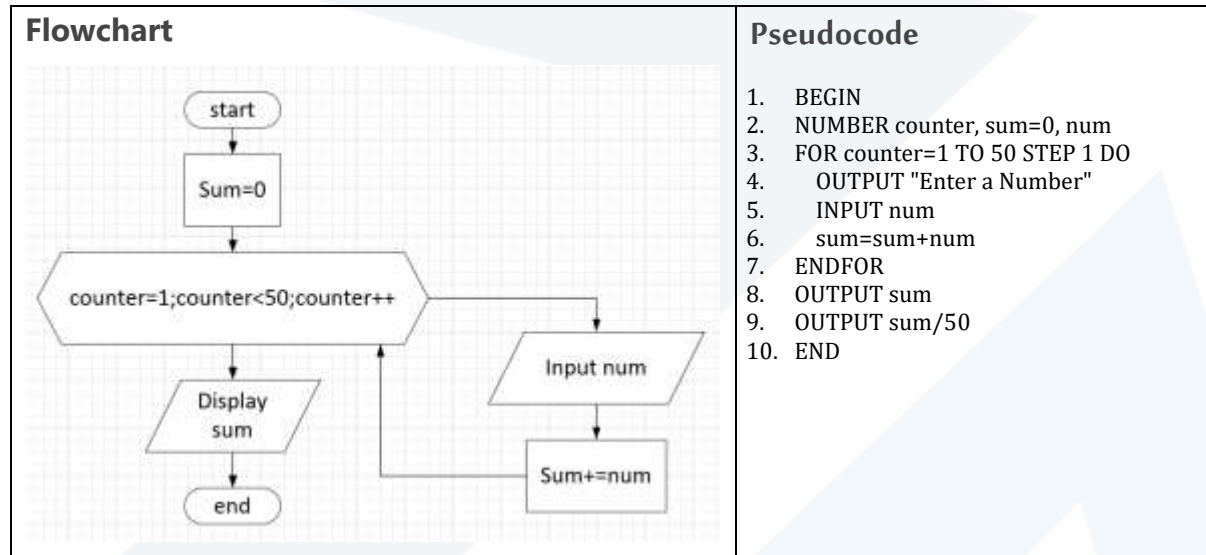
Statement 2 defines the condition for the loop to run (i must be less than 5). If the condition is true, the loop will start over again, if it is false, the loop will end.

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

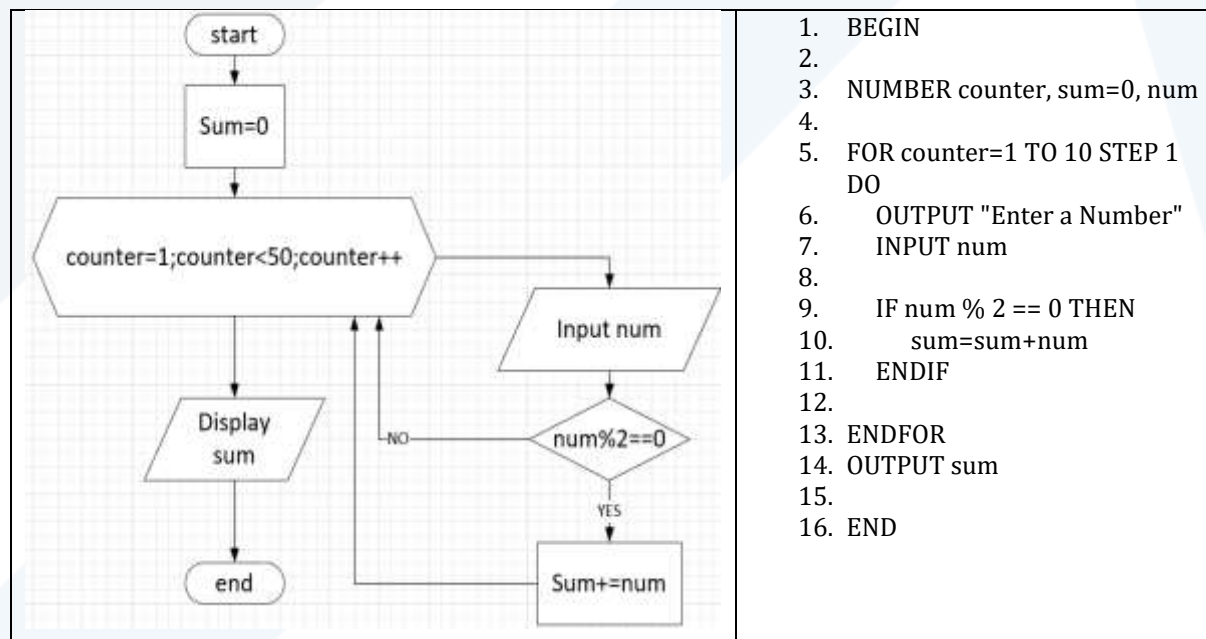
**Another Example:** This example will only print even values between 0 and 10:

```
for (int i = 0; i <= 10; i = i + 2) {  
    cout << i << "\n";  
}
```

Pseudocode Example: Read 50 numbers and find their sum and average. (Pseudocode For Loop Example)



EXAMPLE: Read 10 numbers and find sum of even numbers



## Nested Loops

It is also possible to place a loop inside another loop. This is called a **nested loop**.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example

```
// Outer loop
for (int i = 1; i <= 2; ++i) {
    cout << "Outer: " << i << "\n"; // Executes 2 times

    // Inner loop
    for (int j = 1; j <= 3; ++j) {
        cout << " Inner: " << j << "\n"; // Executes 6 times (2 * 3)
    }
}
```

## C++ Break and Continue

### C++ Break

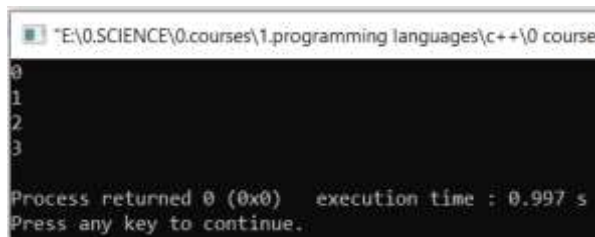
You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a **loop**.

This example jumps out of the loop when i is equal to 4:

Example

```
for (int i = 0; i < 10; i++) {
    if (i == 4) {
        break;
    }
    cout << i << "\n";
}
```



```
"E:\0.SCIENCE\0.courses\1.programming languages\c++\0 course\
0
1
2
3
Process returned 0 (0x0)   execution time : 0.997 s
Press any key to continue.
```

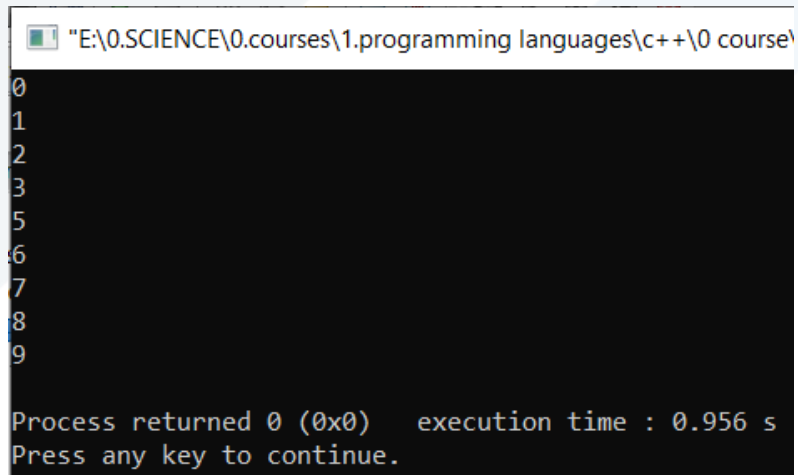
## C++ Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

Example

```
for (int i = 0; i < 10; i++) {  
    if (i == 4) {  
        continue;  
    }  
    cout << i << "\n";  
}
```



```
"E:\0.SCIENCE\0.courses\1.programming languages\c++\0 course"  
0  
1  
2  
3  
5  
6  
7  
8  
9  
  
Process returned 0 (0x0)   execution time : 0.956 s  
Press any key to continue.
```

## C++ 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:

```
string cars[4];
```

We have now declared a variable that holds an array of four strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
```

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

```
int myNum[3] = {10, 20, 30};
```

### Access the Elements of an Array

You access an array element by referring to the index number inside square brackets [].

This statement accesses the value of the **first element** in cars:

Example

```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
cout << cars[0];
// Outputs Volvo
```

**Note:** Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

### Change an Array Element

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

```
cars[0] = "Opel";
```

Example

```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = "Opel";
cout << cars[0];
// Now outputs Opel instead of Volvo
```

## C++ Arrays and Loops

### 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:

Example

```
string cars[5] = {"Volvo", "BMW", "Ford", "Mazda", "Tesla"};
for (int i = 0; i < 5; i++) {
    cout << cars[i] << "\n";
}
```

This example outputs the index of each element together with its value:

Example

```
string cars[5] = {"Volvo", "BMW", "Ford", "Mazda", "Tesla"};
for (int i = 0; i < 5; i++) {
    cout << i << " = " << cars[i] << "\n";
}
```

And this example shows how to loop through an array of integers:

Example

```
int myNumbers[5] = {10, 20, 30, 40, 50};
for (int i = 0; i < 5; i++) {
    cout << myNumbers[i] << "\n";
}
```

### The foreach Loop

There is also a "**for-each** loop" (introduced in C++ version 11 (2011), which is used exclusively to loop through elements in an array (or other data sets):

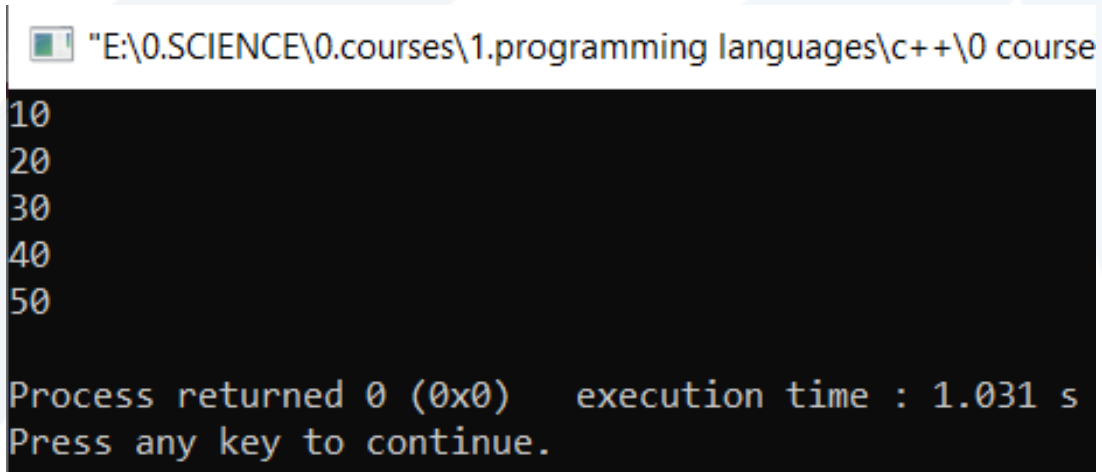
Syntax

```
for (type variableName : arrayName) {
    // code block to be executed
}
```

The following example outputs all elements in an array, using a "for-each loop":

Example:

```
# include <iostream>
using namespace std;
const int size = 50;
int main()
{
int myNumbers[5] = {10, 20, 30, 40, 50};
for (int i : myNumbers) {
    cout << i << "\n";
}
}
```



```
"E:\0.SCIENCE\0.courses\1.programming languages\c++\0 course
10
20
30
40
50
Process returned 0 (0x0) execution time : 1.031 s
Press any key to continue.
```

### C++ Omit Array Size

In C++, you don't have to specify the size of the array. The compiler is smart enough to determine the size of the array based on the number of inserted values:

```
string cars[] = {"Volvo", "BMW", "Ford"}; // Three arrays
```

The example above is equal to:

```
string cars[3] = {"Volvo", "BMW", "Ford"}; // Also three arrays
```

However, the last approach is considered as "good practice", because it will reduce the chance of errors in your program.

## Omit Elements on Declaration

It is also possible to declare an array without specifying the elements on declaration, and add them later:

Example

```
string cars[5];  
cars[0] = "Volvo";  
cars[1] = "BMW";  
...
```

## Get the Size of an Array

To get the size of an array, you can use the `sizeof()` operator:

Example

```
int myNumbers[5] = {10, 20, 30, 40, 50};  
cout << sizeof(myNumbers);
```

Result:

20

Why did the result show 20 instead of 5, when the array contains 5 elements?

It is because the `sizeof()` operator returns the size of a type in **bytes**.

You learned from the Data Types chapter that an `int` type is usually 4 bytes, so from the example above,  $4 \times 5$  (4 bytes x 5 elements) = **20 bytes**.

**To find out how many elements an array has**, you have to divide the size of the array by the size of the data type it contains:

Example

```
int myNumbers[5] = {10, 20, 30, 40, 50};  
int getArrayLength = sizeof(myNumbers) / sizeof(int);  
cout << getArrayLength;
```

Result:

5



## Loop Through an Array with sizeof()

In the Arrays and Loops Chapter, we wrote the size of the array in the loop condition ( $i < 5$ ). This is not ideal, since it will only work for arrays of a specified size.

However, by using the `sizeof()` approach from the example above, we can now make loops that work for arrays of any size, which is more sustainable.

Instead of writing:

```
int myNumbers[5] = {10, 20, 30, 40, 50};
for (int i = 0; i < 5; i++) {
    cout << myNumbers[i] << "\n";
}
```

It is better to write:

Example

```
int myNumbers[5] = {10, 20, 30, 40, 50};
for (int i = 0; i < sizeof(myNumbers) / sizeof(int); i++) {
    cout << myNumbers[i] << "\n";
}
```

Note that, in C++ version 11 (2011), you can also use the "for-each" loop:

Example

```
int myNumbers[5] = {10, 20, 30, 40, 50};
for (int i : myNumbers) {
    cout << i << "\n";
}
```

## C++ Multi-Dimensional Arrays

A multi-dimensional array is an array of arrays. To declare a multi-dimensional array, define the variable type, specify the name of the array followed by square brackets which specify how many elements the main array has, followed by another set of square brackets which indicates how many elements the sub-arrays have:

```
string letters[2][4];
```

As with ordinary arrays, you can insert values with an array literal - a comma-separated list inside curly braces. In a multi-dimensional array, each element in an array literal is another array literal.

```
string letters[2][4] = {  
    {"A", "B", "C", "D"},  
    {"E", "F", "G", "H"}  
};
```

Each set of square brackets in an array declaration adds another **dimension** to an array. An array like the one above is said to have two dimensions.

Arrays can have any number of dimensions. The more dimensions an array has, the more complex the code becomes. The following array has three dimensions:

```
string letters[2][2][2] = {  
    {  
        {"A", "B"},  
        {"C", "D"}  
    },  
    {  
        {"E", "F"},  
        {"G", "H"}  
    }  
};
```

### Access the Elements of a Multi-Dimensional Array

To access an element of a multi-dimensional array, specify an index number in each of the array's dimensions. This statement accesses the value of the element in the **first row (0)** and **third column (2)** of the **letters** array.

Example

```
string letters[2][4] = {  
    {"A", "B", "C", "D"},  
    {"E", "F", "G", "H"}  
};  
  
cout << letters[0][2]; // Outputs "C"
```

**Remember that:** Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

## Change Elements in a Multi-Dimensional Array

To change the value of an element, refer to the index number of the element in each of the dimensions:

Example

```
string letters[2][4] = {
    {"A", "B", "C", "D"},
    {"E", "F", "G", "H"}
};
letters[0][0] = "Z";

cout << letters[0][0]; // Now outputs "Z" instead of "A"
```

Loop Through a Multi-Dimensional Array

To loop through a multi-dimensional array, you need one loop for each of the array's dimensions.

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

Example

```
string letters[2][4] = {
    {"A", "B", "C", "D"},
    {"E", "F", "G", "H"}
};

for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 4; j++) {
        cout << letters[i][j] << "\n";
    }
}
```

This example shows how to loop through a three-dimensional array:

Example

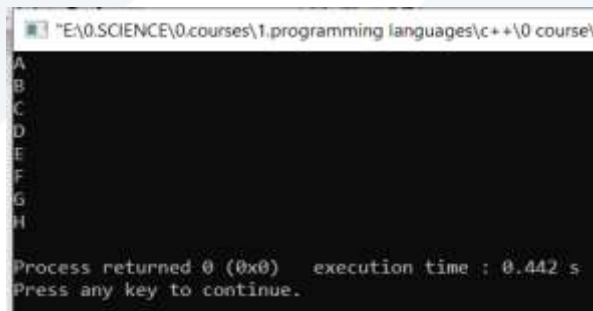
```
string letters[2][2][2] = {
    {
        {"A", "B"},
        {"C", "D"}
    },
    {
```

```

{"E", "F"},
{"G", "H"}
}
};

for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        for (int k = 0; k < 2; k++) {
            cout << letters[i][j][k] << "\n";
        }
    }
}

```



The screenshot shows a terminal window with the following output:

```

A
B
C
D
E
F
G
H
Process returned 0 (0x0)   execution time : 0.442 s
Press any key to continue.

```

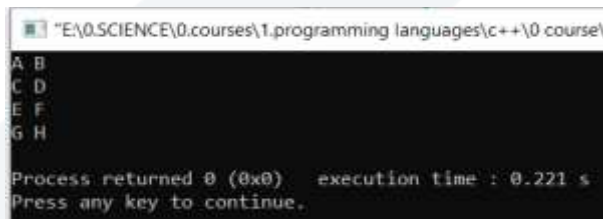
```

string letters[2][2][2] = {
{
    {"A", "B"},
    {"C", "D"}
},
{
    {"E", "F"},
    {"G", "H"}
}
};

for (int i = 0; i < 2; i++) {

```

```
for (int j = 0; j < 2; j++) {
    for (int k = 0; k < 2; k++) {
        cout << letters[i][j][k] << " ";
    }
    cout << endl;
}
```



```
"E:\0.SCIENCE\0.courses\1.programming languages\c++\0 course\
A B
C D
E F
G H
Process returned 0 (0x0) execution time : 0.221 s
Press any key to continue.
```

## Why Multi-Dimensional Arrays?

Multi-dimensional arrays are great at representing grids. This example shows a practical use for them. In the following example we use a multi-dimensional array to represent a small game of Battleship:

- Example

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

```
const int size = 50;
```

```
int main()
```

```
{// We put "1" to indicate there is a ship.
```

```
bool ships[4][4] = {
```

```
{ 0, 1, 1, 0 },
```

```
{ 0, 0, 0, 0 },
```

```
{ 0, 0, 1, 0 },
```

```
{ 0, 0, 1, 0 }
```

```
};
```

```
int hits = 0; // Keep track of how many hits the player has and how many turns they have played in these variables
```

```
int numberOfTurns = 0;
```

```
int row, column;

while (hits < 4) { // Allow the player to keep going until they have hit all four ships
    cout << "Selecting coordinates\n";

    do{// Ask the player for a row

        cout << "Choose a row number between 0 and 3: ";

        cin >> row;

    } while (row>3 || row<0);

    do{// Ask the player for a column

        cout << "Choose a column number between 0 and 3: ";

        cin >> column;

    } while (column>3 || column<0);

    if (ships[row][column]) { // Check if a ship exists in those coordinates

        ships[row][column] = 0; // If the player hit a ship, remove it by setting the value to zero.

        hits++; // Increase the hit counter

        cout << "Hit! " << (4-hits) << " left.\n\n"; // Tell the player that they have hit a ship and how many ships are left

    }

    else{

        cout << "Miss\n\n"; // Tell the player that they missed

    }

    numberOfTurns++; // Count how many turns the player has taken

}

cout << "Victory!\n";

cout << "You won in " << numberOfTurns << " turns";

return 0;}
```

انتهت المحاضرة