Arrays, Functions, Loops Calculation, Sort

Moien Makkiyan

Basic Programming
Session 2

December 1, 2024

Overview

Review

Your First Program:

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

int main() {
    cout << "Hello World!";
    return 0;
}
```

Conditions and If Statement

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is
  // false and condition2 is true
} else {
  // block of code to be executed if the condition1 is
  // false and condition2 is false
}
```

Switch Statements

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

While Loop

The while loop loops through a block of code as long as a specified condition is true.
Syntax:

```
while (condition) {
   // code block to be executed
 }
```

For 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.

Arrays

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:

```
int myNum[3];
```

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

```
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 [].

Change an Array Element

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

```
int myNum[3] = {10, 20, 30};
myNum[0] = "11";
cout << myNum[0];
// Now outputs 11 instead of 10
```

Note

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

Another Way To Declare an Array

Another Way To Declare an Array:

```
int myNum[] = {10, 20, 30, 40, 50};
cout << myNum[4];
// Now outputs 50
```

If you don't specify the array size, an error occurs:

```
int myNum[];
myNum[0] = 10;
        myNum[1] = 20;
        myNum[2] = 30;
        myNum[3] = 40;
```

C++ Arrays and Loops

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

```
int n = 10;
for(int i=0 ; i<n ; i++)
{
        cin>>n[i];
}
for(int i=0 ; i<n ; i++)
{
      cout<<n[i]*n[i]<<endl;
}
```

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 (and other data structures).

Syntax:

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

Example:

```
// Create an array of integers
int myNumbers[5] = {10, 20, 30, 40, 50};

// Loop through integers
for (int i : myNumbers) {
        cout << i << "\n";
}
```

C++ Array Size

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

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

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.
To find out how many elements an array has, you have to divide the size of the array by the size of the first element in the array:

```
int myNumbers[5] = {10, 20, 30, 40, 50};
int getArrayLength = sizeof(myNumbers) / sizeof(myNumbers[0]);
cout << getArrayLength;
```

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:

```
int table[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.

```
int table[2][4] = {
        { 1, 2, 3, 4 },
        { 5, 6, 7, 8 }
};
```

Functions

Create a Function

A function is a block of code which only runs when it is called.
You can pass data, known as parameters, into a function.
Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times. Syntax:

```
void myFunction() {
        // code to be executed
}
```

Example Explained:

myFunction() is the name of the function

void means that the function does not have a return value.

inside the function (the body), add code that defines what the function should do.

Call a Function

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are called. To call a function, write the function's name followed by two parentheses () and a semicolon ;
In the following example, myFunction() is used to print a text (the action), when it is called:

```
void myFunction() {
        cout << "I just got executed!";
}
int main() {
        myFunction(); // Outputs "I just got executed!"
        return 0;
}
```

A function can be called multiple times.

Function Declaration and Definition

A C++ function consist of two parts:

- Declaration: the return type, the name of the function, and parameters (if any)
- Definition: the body of the function (code to be executed)

Note

If a user-defined function, such as myFunction() is declared after the main() function, an error will occur.

Example with Error!

```
int main() {
        myFunction();
        return 0;
}

void myFunction() {
        cout << "I just got executed!";
}

// Error
```

Correct Example

```
// Function declaration
void myFunction();

// The main method
int main() {
        myFunction();  // call the function
        return 0;
}
// Function definition
void myFunction() {
        cout << "I just got executed!";
}
```

Parameters and Arguments

Information can be passed to functions as a parameter. Parameters act as variables inside the function.
Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma. Syntax:

```
void functionName(parameter1, parameter2, parameter3) {
        // code to be executed
}
```

Note

You can also use a default parameter value, by using the equals sign (=).

Return Values

The void keyword, used in the previous examples, indicates that the function should not return a value. If you want the function to return a value, you can use a data type (such as int, double, etc.) instead of void, and use the return keyword inside the function:

```
int myFunction(int x, int y) {
        return x + y;
}

int main() {
        cout << myFunction(5, 3);
        return 0;
}

// Outputs 8 (5 + 3)
```

Return Values

The void keyword, used in the previous examples, indicates that the function should not return a value. If you want the function to return a value, you can use a data type (such as int, double, etc.) instead of void, and use the return keyword inside the function:

```
int myFunction(int x, int y) {
        return x + y;
}

int main() {
        cout << myFunction(5, 3);
        return 0;
}

// Outputs 8 (5 + 3)
```

Return Values

You can also store the result in a variable:

```
int myFunction(int x, int y) {
        return x + y;
}

int main() {
        int z = myFunction(5, 3);
        cout << z;
        return 0;
}
// Outputs 8 (5 + 3)
```

Pass Arrays as Function Parameters

You can also pass arrays to a function:

```
void myFunction(int myNumbers[5]) {
        for (int i = 0; i < 5; i++) {
                cout << myNumbers[i] << "\n";
        }
}

int main() {
        int myNumbers[5] = {10, 20, 30, 40, 50};
        myFunction(myNumbers);
        return 0;
}
```

String