6.3 — Arrays and loops

BY ALEX ON JULY 2ND, 2007 | LAST MODIFIED BY ALEX ON JANUARY 23RD, 2020

Consider the case where we want to find the average test score of a class of students. Using individual variables:

```
int numStudents{ 5 };
int score0{ 84 };
int score1{ 92 };
int score2{ 76 };
int score3{ 81 };
int score4{ 56 };

int totalScore{ score0 + score1 + score2 + score3 + score4 };
auto averageScore{ static_cast<double>(totalScore) / numStudents };
```

That's a lot of variables and a lot of typing -- and this is just 5 students! Imagine how much work we'd have to do for 30 students, or 150.

Plus, if a new student is added, a new variable has to be declared, initialized, and added to the totalScore calculation. Any time you have to modify old code, you run the risk of introducing errors.

Using arrays offers a slightly better solution:

```
int scores[]{ 84, 92, 76, 81, 56 };
int numStudents{ static_cast<int>(std::size(scores)) }; // requires C++17 and <iterator> header
int totalScore{ scores[0] + scores[1] + scores[2] + scores[3] + scores[4] };
auto averageScore{ static_cast<double>(totalScore) / numStudents };
```

This cuts down on the number of variables declared significantly, but totalScore still requires each array element be listed individually. And as above, changing the number of students means the totalScore formula needs to be manually adjusted.

If only there were a way to loop through our array and calculate totalScore directly.

Loops and arrays

In a previous lesson, you learned that the array subscript doesn't need to be a constant value -- it can be a variable. This means we can use a loop variable as an array index to loop through all of the elements of our array and perform some calculation on them. This is such a common thing to do that wherever you find arrays, you will almost certainly find loops! When a loop is used to access each array element in turn, this is often called **iterating** through the array.

Here's our example above using a *for loop*:

```
int scores[]{ 84, 92, 76, 81, 56 };
int numStudents{ static_cast<int>(std::size(scores)) };
// const int numStudents{ sizeof(scores) / sizeof(scores[0]) }; // use this instead if not C++
int totalScore{ 0 };

// use a loop to calculate totalScore
for (int student{ 0 }; student < numStudents; ++student)
    totalScore += scores[student];

auto averageScore{ static_cast<double>(totalScore) / numStudents };
```

This solution is ideal in terms of both readability and maintenance. Because the loop does all of our array element accesses, the formulas adjust automatically to account for the number of elements in the array. This means the calculations do not have to be manually altered to account for new students, and we do not have to manually add the name of new array elements!

Here's an example of using a loop to search an array in order to determine the best score in the class:

```
#include <iostream>
#include <iterator> // for std::size

int main()
{
    int scores[]{ 84, 92, 76, 81, 56 };
    int numStudents{ static_cast<int>(std::size(scores)) };

    int maxScore{ 0 }; // keep track of our largest score
    for (int student{ 0 }; student < numStudents; ++student)
        if (scores[student] > maxScore)
            maxScore = scores[student];

    std::cout << "The best score was " << maxScore << '\n';

    return 0;
}
```

In this example, we use a non-loop variable called maxScore to keep track of the highest score we've seen. maxScore is initialized to 0 to represent that we have not seen any scores yet. We then iterate through each element of the array, and if we find a score that is higher than any we've seen before, we set maxScore to that value. Thus, maxScore always represents the highest score out of all the elements we've searched so far. By the time we reach the end of the array, maxScore holds the highest score in the entire array.

Mixing loops and arrays

Loops are typically used with arrays to do one of three things:
1) Calculate a value (e.g. average value, total value)
2) Search for a value (e.g. highest value, lowest value).
3) Reorganize the array (e.g. ascending order, descending order)

When calculating a value, a variable is typically used to hold an intermediate result that is used to calculate the final value. In the above example where we are calculating an average score, totalScore holds the total score for all the elements examined so far.

When searching for a value, a variable is typically used to hold the best candidate value seen so far (or the array index of the best candidate). In the above example where we use a loop to find the best score, maxScore is used to hold the highest score encountered so far.

Sorting an array is a bit more tricky, as it typically involves nested loops. We will cover sorting an array in the next lesson.

Arrays and off-by-one errors

One of the trickiest parts of using loops with arrays is making sure the loop iterates the proper number of times. Off-by-one errors are easy to make, and trying to access an element that is larger than the length of the array can have dire consequences. Consider the following program:

```
    int scores[]{ 84, 92, 76, 81, 56 };
    int numStudents{ static_cast<int>(std::size(scores)) };

    int maxScore{ 0 }; // keep track of our largest score
    for (int student{ 0 }; student <= numStudents; ++student)
        if (scores[student] > maxScore)
            maxScore = scores[student];

    std::cout << "The best score was " << maxScore << '\n';
```

The problem with this program is that the condition in the for loop is wrong! The array declared has 5 elements, indexed from 0 to 4. However, this array loops from 0 to 5. Consequently, on the last iteration, the array will execute this:

```
        if (scores[5] > maxScore)
            maxScore = scores[5];
```

But scores[5] is undefined! This can cause all sorts of issues, with the most likely being that scores[5] results in a garbage value. In this case, the probable result is that maxScore will be wrong.

However, imagine what would happen if we inadvertently assigned a value to array[5]! We might overwrite another variable (or part of it), or perhaps corrupt something -- these types of bugs can be very hard to track down!

Consequently, when using loops with arrays, always double-check your loop conditions to make sure you do not introduce off-by-one errors.

Quiz

1) Print the following array to the screen using a loop:

```
    int array[]{ 4, 6, 7, 3, 8, 2, 1, 9, 5 };
```

Hint: You can use std::size (as of C++17) or the sizeof() trick (prior to C++17) to determine the array length.

2) Given the array in question 1:

Ask the user for a number between 1 and 9. If the user does not enter a number between 1 and 9, repeatedly ask for an integer value until they do. Once they have entered a number between 1 and 9, print the array. Then search the array for the value that the user entered and print the index of that element.

You can test std::cin for invalid input by using the following code:

```
// if the user entered something invalid
if (std::cin.fail())
{
    std::cin.clear(); // reset any error flags
    std::cin.ignore(32767, '\n'); // ignore any characters in the input buffer
}
```

3) Modify the following program so that instead of having maxScore hold the largest score directly, a variable named maxIndex holds the index of the largest score.

```
#include <iostream>
#include <iterator> // for std::size

int main()
{
    int scores[]{ 84, 92, 76, 81, 56 };
    int numStudents{ static_cast<int>(std::size(scores)) };

    int maxScore{ 0 }; // keep track of our largest score

    // now look for a larger score
    for (int student{ 0 }; student < numStudents; ++student)
        if (scores[student] > maxScore)
            maxScore = scores[student];

    std::cout << "The best score was " << maxScore << '\n';

    return 0;
}
```

Quiz solutions

1) **Show Solution**

2) **Show Solution**

3) **Show Solution**

6.4 -- Sorting an array using selection sort

Index

6.2 -- Arrays (Part II)

C++ TUTORIAL | PRINT THIS POST

321 comments to 6.3 — Arrays and loops

« Older Comments 1 ... 3 4 5

moji
February 5, 2020 at 3:57 am · Reply

i'm using VSCode 2019 v6.1; my default cpp standard is set to C++20 & c standard to c11.
but i cant use std::size(array) and it give me the following error:

error: 'size' is not a member of 'std'
    int studentNum {static_cast<int>(std::size(score))};

is there any config i missed??

nascardriver
February 6, 2020 at 9:12 am · Reply

It doesn't matter which text editor you're using. You need to check which compiler you're using and look up if it supports C++17 (or C++20).

Matthias
December 27, 2019 at 10:22 am · Reply

The solution to Quiz 1 is "wrong" in terms of it doesn't correspond to the array asked in the question.

nascardriver
December 28, 2019 at 1:57 am · Reply

I broke it during an update and didn't notice, it's fixed now. Thanks!

BooGDaaN
November 29, 2019 at 11:58 am · Reply

This is my solution for quiz 2. I hope this is ok.

```cpp
#include <iostream>
#include <iterator>

int userInput();
void printArray(const int array[], unsigned int arraySize);
int searchArray(const int array[], unsigned int arraySize, int number);

int main()
{
    int userNumber{ userInput() };

    const int array[] = { 4, 6, 7, 3, 8, 2, 1, 9, 5 };
    unsigned int arraySize{ std::size(array) };

    printArray(array, arraySize);

    int indexArray{ searchArray(array,arraySize,userNumber) };
    std::cout << "The number " << userNumber << " was found at index: " << indexArray << '\

    return 0;
}


//Return one number between 1 and 9
int userInput()
{
    int number{ 0 };
    do
    {
        std::cout << "Enter a number between 1 and 9: ";
        std::cin >> number;

        // if the user entered something invalid
        if (std::cin.fail())
        {
            std::cin.clear(); // reset any error flags
            std::cin.ignore(32767, '\n'); // ignore any characters in the input buffer
        }
    } while (number < 1 || number > 9);

    return number;
}

void printArray(const int array[], unsigned int arraySize)
{
    std::cout << "The array is: ";
    for (unsigned int iii{ 0 }; iii < arraySize; ++iii)
        std::cout << array[iii] << " ";
    std::cout << '\n';
}

// Return the index where number is found in the array
int searchArray(const int array[], unsigned int arraySize, int number)
{
    int returnNumber{ 0 };
    for (unsigned int iii{ 0 }; iii < arraySize; ++iii)
        if (array[iii] == number)
            returnNumber = iii;
    return returnNumber;
}
```

nascardriver
December 2, 2019 at 4:19 am · Reply

- `std::size` returns a `std::size_t`, which is an unsigned integer type, but not necessarily `unsigned int`.
- Use single quotation marks for characters (' ' instead of " ").

Rest looks good :)

BooGDaaN
December 3, 2019 at 2:15 am · Reply

I updated the code. Thank you very much!

gacrux
October 28, 2019 at 12:42 pm · Reply

On code blocks 17.12 (also tried the 20191006 nightly):

```
#include <iostream>
#include <iterator>

int main()
{
    int scores[] = { 84, 92, 76, 81, 56 };
    const int numStudents = std::size(scores);
    std::cout << numStudents;
    return 0;
}
```

```
||=== Build: Debug in 63 lesson (compiler: GNU GCC Compiler) ===|
C:\Users\admin\Documents\cbprojects\63 lesson\main.cpp||In function 'int main()':|
C:\Users\admin\Documents\cbprojects\63 lesson\main.cpp|7|error: 'size' is not a member of 'std'|
||=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===|
```

https://en.cppreference.com/w/cpp/iterator/size
Defined in header <iterator> (since C++17)

Pretty sure the compiler is configured as shown in the very first lessons, already checked twice.

Even the example in cppreference shows the same error :(

nascardriver
October 29, 2019 at 2:32 am · Reply

https://www.learncpp.com/images/CppTutorial/Chapter0/CB-C++11-min.png
This is the setting you need to look for, but you need at least C++17. If C++17 isn't available, you need to update GCC.

Gacrux
October 29, 2019 at 9:11 am · Reply

The compiler is configured exactly as shown, but the error was still occurring.

Updating to MinGW-W64 GCC-8.1.0 (x86_64-posix-seh) fixed the issue.

For those interested: https://sourceforge.net/projects/mingw-w64/files/mingw-w64/

The toolchain have to be configured manually in the compiler settings for the new binaries.

nascardriver
October 29, 2019 at 9:14 am · Reply

Thanks for posting how you fixed it. I suppose you were using an old version of gcc and codeblocks didn't tell you that -std=c++17 was unsupported.

Chandler
December 8, 2019 at 9:27 pm · Reply

For some reason, C::B is still using an old compiler (v5.1.x) with their IDE. You'll definitely have to upgrade your compiler to have C++17 capabilities. I'm using Qt 5.13.2 and it ships with MinGW 7.3.0 and is 100% C++17 compliant.

Kim Stewart
September 22, 2019 at 12:52 pm · Reply

Write a program that reads 10 scores of a student, then find a average 10 scores, a number of passing score and a number of failing score.

Assume that possible score is 100
Passing score is 50 and above
Failing score is 49 and below

nascardriver

September 23, 2019 at 8:23 am · Reply

Where are you stuck? What have you tried? Why didn't it work?

Kim Stewart
September 23, 2019 at 2:35 pm · Reply

I tried to put numbers all together with int but, get did not work, can you show me how I should it be done, and thanks in advance.

nascardriver
September 23, 2019 at 11:02 pm · Reply

Show me your current code and mark what exactly you need help with. I won't do your homework for you.

Sam
August 5, 2019 at 6:00 pm · Reply

Here is my solution for Question #3

```
1   #include <iostream>
2   #include <iterator> // for std::size
3
4   int main()
5   {
6       int scores[] = { 84, 92, 76, 81, 56 };
7
8       const int numStudents = sizeof(scores) / sizeof(scores[0]);
9
10      int maxIndex = 0; // keep track of our largest score
11
12      // now look for a larger score
13      for (int student = 0; student < numStudents; ++student)
14          if (scores[student] > scores[maxIndex])
15              maxIndex = student;
16
17      std::cout << "The best score was " << scores[maxIndex] << '\n';
18
19      return 0;
20  }
```

nascardriver
August 6, 2019 at 3:04 am · Reply

- Initialize your variables with brace initializers.
- Compile-time constants should be `constexpr`.
- Line 2: "C++14" :)

Sam
August 6, 2019 at 2:52 pm · Reply

fixed :)

```cpp
#include <iostream>

int main()
{
    int scores[] { 84, 92, 76, 81, 56 };

    constexpr int numStudents { sizeof(scores) / sizeof(scores[0]) };

    int maxIndex { 0 };

    for (int student { 0 }; student < numStudents; ++student)
    {
      if (scores[student] > scores[maxIndex])
      {
        maxIndex = student;
      }
    }

    std::cout << "The best score was " << scores[maxIndex] << '\n';

    return 0;
}
```

« Older Comments 1 … 3 4 5