Learning Objectives: Array Basics

- **Create and initialize an array**
- **Access and modify array elements**
- **Iterate through arrays using both a regular `for` loop and an *enhanced* `for` loop**
- **Determine array output**

Creating an Array

What Is an Array?

Before we discuss **vectors**, we will first learn about **arrays**, a simpler form of a vector. An **array** is a data structure that stores a collection of data such as ints, doubles, strings, etc. This data is often referred to as the array's **elements**. Being able to store elements into an array helps reduce the amount of time needed to declare and initialize variables. For example, if you wanted to store the ages of all family members in your household, you would typically have to declare and initialize integer variables and values for each family member. Copy the code below into the text editor on the left and then click the `TRY IT` button to see the output. You can also click on the ++Code Visualizer++ link underneath to see how the program runs behind the scenes.

```
int Allan = 71;
int Bob = 42;
int Carol = 37;
int David = 5;
int Ellen = 18;

cout << Allan << endl;
```

Code Visualizer

challenge

What happens if you:

- Change `Allan` in `cout << Allan << endl` to `Bob`, `Carol`, `David`, or `Ellen`?

Code Visualizer

Array Creation

To avoid the repetitive task of declaring and initializing multiple variables, you can declare an array and directly assign values or elements into that array like below.

```
int ages[] = {71, 42, 37, 5, 18};
```

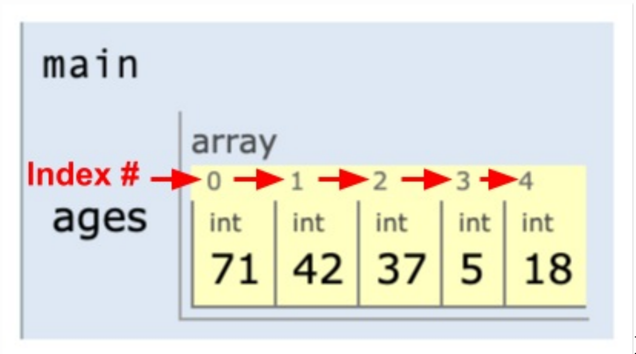
Code Visualizer

Method Syntax with Elements:

- Specify the data type that the array will store (i.e. `int`).
- Declare the variable name for the array (i.e. `ages`) followed by empty brackets `[]` followed by the assignment symbol `=`.
- Elements assigned to the array are separated by commas `,` and enclosed within curly braces `{}`.

▼ Additional information

If you used the Code Visualizer, you'll notice that the array variable `ages` refers to all of the elements as a collection. An array is considered to be a **collection** that bundles all of the data that it holds.



Note that the first array slot, or **index**, is always `0` so `71` is located at index `0` instead of `1`.

Alternatively, you can create an array without any elements in which you will need to declare and specify the array variable name and size before you can assign elements to the array.

```
int ages[5];
```

Method Syntax without Elements

- Specify the data type that the array will store (i.e. `int`).
- Declare the variable name for the array (i.e. `ages`) followed by the number of elements you want the array to hold within brackets (i.e. `[5]`).

▼ **Additional information**

Note that when you declare an array without initializing any elements, the system will still reserve enough memory for the array to hold the specified number of elements. This means that you can initialize elements within the array later on.

Array Details

If an element within an array has not been initialized yet, printing it will cause the system to output **random memory data**. Random memory data is often generated when array elements are not initialized.

```
int ages[5];
cout << ages[0] << endl;
```

Note that `ages[0]` in the example above refers the element at **index** `0`, also known as the first **position** within the array. Currently, the element at the first position is not initialized so printing the first element will only output random memory data. In fact, the same will happen if you try to print any other elements within the array. Additionally, all elements within the array must be of the same type. If you try to store a string within an integer array, or a double within a boolean array, you will get an error message.

```
int ages[] = {71, 42, 37, 5, "eighteen"};

cout << ages[4] << endl;
```

challenge

What happens if you:

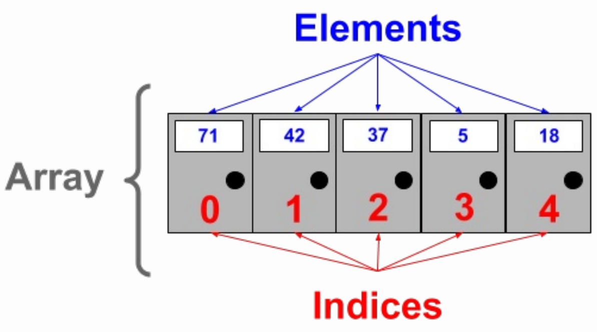
- Change `"eighteen"` in the code above to the integer `18`?
- Replace all your code with just `int ages[];`

important

IMPORTANT

When you create an array in C++, you *must* specify the number of elements that you expect the array to hold. Otherwise, you will get an error.

P.O. Boxes at the postal office are symbolically similar to arrays. Each row of P.O. Boxes is like an array, except each box can only store *one* item (element) *and* each item within that row must be of the same *type* (i.e. integers).



.guides/img/ArrayElementsIndices

Accessing an Array

Array Access

To access and print array elements, you need to know their position. The position at which an element is stored is called its **index**. For example, `names[0]` refers to the first element in the array called `names`. Array indices always start at `0` and increment by 1 with each element that comes next. Due to this, `numbers[4]` refers to the *fifth* element in the array, *not* the fourth.

```
string names[] = {"Alan", "Bob", "Carol", "David", "Ellen"};

cout << names[0] << endl;
```

challenge

What happens if you:

- Change `names[0]` in the code above to `names[2]`?
- Change `names[0]` in the code above to `names[3]`?
- Change `names[0]` in the code above to `names`?

important

IMPORTANT

You may have noticed that printing the `names` array without specifying an index resulted in an output that included a mixture of numbers and letters. This occurs because printing an array actually prints its memory location, not its elements. You'll learn how to print all elements in an array without having to specify all of their indices on a later page.

Array Key Points

```
bool bools[] = {true, false, true};
double decimals[] = {2.3, 4};
int integers[1];

cout << bools[0] << endl;
```

challenge

What happens if you:

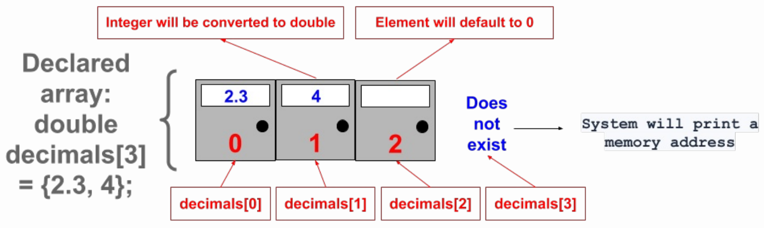
- Change `bools[0]` in the original code to `bools[1]`?
  ▼ Hint
  The system will print `0` because `bools[1]` refers to the second element in the `bools` array. Remember that `0` is the integer equivalent to `false` and `1` is the integer equivalent to `true`. To print their respective boolean equivalent, use `cout << boolalpha << bools[1] << endl;` instead.
- Change `bools[0]` in the original code to `decimals[1]`?
  ▼ Hint
  The system will print `4` because `decimals[1]` refers to the second element in the `decimals` array.
- Change `bools[0]` in the original code to `integers[1]`?
  ▼ Hint
  The system will print random memory data because `integers[1]` is not valid within the `integers` array. Currently there is only one element at the first index which has not been formally initialized yet.
- Change `int integers[1];` in the original code to `int integers[1] = {1.2};` and `cout << bools[0] << endl;` to `cout << integers[0] << endl;`?
  ▼ Hint
  The system will return an error message. Though you have tried to initialize the first element within the `integers` array to `1.2`, the system will not allow you to put a double element into an integer array. However, it is possible to put an integer element into a double array because integers can be expressed as doubles but not vice versa.

important

IMPORTANT

Here are some key points to keep in mind when working with arrays:

- When you declare an array, you must specify the number of elements you wish the array to hold before you can initialize any elements. One exception is if you declare and initialize the array and elements at the same time. Doing so means you do not need to specify the number of elements in brackets `[]` since the system will determine that based on the number of elements you have initialized.
- If you do not initialize any elements, printing the elements will only result in random memory data.
- If you try to access an element position that is not valid (i.e. the second element in the `integers` array), the system will also output random memory data.
- Elements must be of the same type as the array. The only exception is that integers can be expressed as doubles and can therefore be put into a double array.


Integer will be converted to double
Element will default to 0
Declared array: double decimals[3] = {2.3, 4};
2.3
4
0
1
2
Does not exist
System will print a memory address
decimals[0]
decimals[1]
decimals[2]
decimals[3]

.guides/img/ArrayExceptions

Modifying an Array

Array Modification

To modify an element within an array, simply find the index at which that element is stored and assign a new value to it.

```
int grades[] = {85, 95, 48, 100, 92};
cout << grades[2] << endl;

grades[2] = 88; //88 will replace 48 at index 2
cout << grades[2] << endl;
```

Code Visualizer

challenge

What happens if you:

- Change `int grades[] = {85, 95, 48, 100, 92};` in the code above to `int grades[5];`?
- Copy the original code but change all `cout << grades[2] << endl;` to `cout << grades[3] << endl;`?
- Change `grades[2] = 88;` in your current code to `grades[3] = 100;`?

Code Visualizer

Modifying Multiple Arrays

You can create and modify as many arrays as you'd like. For example, you can create an array to store your family members and another array to store their age.

```
string family[] = {"Dad", "Mom", "Brother", "Sister"};
int age[4];

cout << family[0] << " " << age[0] << endl;
cout << family[1] << " " << age[1] << endl;
cout << family[2] << " " << age[2] << endl;
cout << family[3] << " " << age[3] << endl;
```

Code Visualizer

challenge

What happens if you:

- Add `age[0] = 50;` directly below the line `int age[4];`?
- Add `age[1] = 45;` below the line `int age[4];` but before the cout statements?
- Add `age[2] = 25;` below the line `int age[4];` but before the cout statements?
- Add `age[3] = 20;` below the line `int age[4];` but before the cout statements?
- Change `"Sister"` within the string array to `"Brother2"`?

Code Visualizer

important

IMPORTANT

Since the integer array above was created without any initialization, random memory data were populated as elements within the array at first. Then by setting the array indices to specific values, you were able to modify the array to include the appropriate age for each family member.

Iterating an Array

Array Iteration

Though we can add many elements to our array, printing each of them can get quite tedious. For example, if we have 10 names of friends in our array, we would need to specify each of their array index to print them.

```
string friends[] = {"Alan", "Bob", "Carol", "David", "Ellen",
                    "Fred", "Grace", "Henry", "Ian", "Jen"};

cout << friends[0] << endl;
cout << friends[1] << endl;
cout << friends[2] << endl;
cout << friends[3] << endl;
cout << friends[4] << endl;
cout << friends[5] << endl;
cout << friends[6] << endl;
cout << friends[7] << endl;
cout << friends[8] << endl;
cout << friends[9] << endl;
```

Luckily, we can use loops which we had learned previously to help us with this process. To print out all of our friends' names without repeating the print statement ten times, we can use a `for` loop to iterate `10` times.

```
string friends[] = {"Alan", "Bob", "Carol", "David", "Ellen",
                    "Fred", "Grace", "Henry", "Ian", "Jen"};

for (int i = 0; i < 10; i++) {
  cout << friends[i] << endl;
}
```

challenge

What happens if you:

- Change `cout << friends[i] << endl;` in the code above to `cout << friends[0] << endl;`?
- Change the cout statement to `cout << friends[10] << endl;`?

important

IMPORTANT

Did you notice that the print statement above includes `i` as the index for `friends`? We do this because `i` will take on the values specified by the `for` loop. The loop starts at `0` and increments by 1 until it reaches `9` (not including 10). Thus, `friends[0]` will print, then `friends[1]`, so on and so forth until `friends[9]` is printed. Then the loop ends.

Array Size

To make the iteration process easier, we can use the `sizeof()` operator to determine how many elements are in our array. To use `sizeof()`, just call it by using the keyword `sizeof` followed by the array name within parentheses `()`.

```
string friends[] = {"Alan", "Bob", "Carol", "David", "Ellen",
                    "Fred", "Grace", "Henry", "Ian", "Jen"};

cout << sizeof(friends) << endl;
```

Why Does `sizeof(friends)` output `320`?

Unfortunately, the `sizeof()` operator does not determine the *number* of the elements within an array. Instead, `sizeof()` calculates the size of the array in *bytes*. In C++, a string takes up 32 bytes and since there are 10 string elements in the array, the *size* of the array in bytes is *320*.

To calculate the *number of elements* within an array, we will need to use `sizeof()` twice.

```
string friends[] = {"Alan", "Bob", "Carol", "David", "Ellen",
                    "Fred", "Grace", "Henry", "Ian", "Jen"};

cout << sizeof(friends) / sizeof(friends[0]) << endl;
```

`sizeof(friends)` calculates the array's size in bytes and `sizeof(friends[0])` calculates the first element's size in bytes. By dividing the array size by the element's size, we were able to determine the number of elements that exist within the array. Note that it doesn't matter whether we calculate the first element's size or the second's since all of the elements are of the same size (32 bytes each).

Looping Through the Elements

Now that you can determine number of elements within an array, you can loop through the array to output each element individually.

```
string friends[] = {"Alan", "Bob", "Carol", "David", "Ellen",
                    "Fred", "Grace", "Henry", "Ian", "Jen"};

for (int i = 0; i < sizeof(friends) / sizeof(friends[0]); i++) {
  cout << friends[i] << endl;
}
```

challenge

What happens if you:

- add `"Kate"` as an element to the array right after `"Jen"`?
- remove `"Alan"` and `"Bob"` from the array?

Notice how `sizeof(friends) / sizeof(friends[0])` continues to keep track of how many elements are in our array even though we've made several changes.

Enhanced For Loop

Using an Enhanced For-Loop

There is a special type of `for` loop that can be used with arrays called an **enhanced for loop**. An enhanced `for` loop, also known as a **range-based for loop**, can be used to iterate through array elements without having to refer to any array indices. To use an enhanced `for` loop, you need the following:
* The keyword `for` followed by parentheses `()`.
* A **typed** iterating variable followed by colon `:` followed by the array name.
* **Note** that the iterating variable must be of the *same* type as the array.
* Any commands that repeat within curly braces `{}`.
* **Note** that when using an enhanced `for` loop, you can print the iterating variable itself without using brackets `[]`.

```
string friends[] = {"Alan", "Bob", "Carol", "David", "Ellen",
                    "Fred", "Grace", "Henry", "Ian", "Jen"};

for (string i : friends) {
  cout << i << endl;
}
```

challenge

What happens if you:

- change `cout << i << endl;` in the code above to `cout << friends[i] << endl;`?
- copy the original code but change `string i` to `int i`?

important

IMPORTANT

One of the main differences between a regular `for` loop and an enhanced `for` loop is that an enhanced `for` loop does not refer to any index or position of the elements in the array. Thus, if you need to access or modify array elements, you **cannot** use an enhanced `for` loop. In addition, you **cannot** use an enhanced `for` loop to iterate through a *part* of the array. Think of an enhanced `for` loop as an *all-or-nothing* loop that just prints all of the array elements or nothing at all. Also note that the iterating variable type **must match** the array type. For example, you cannot use `for (int i : friends)` since `friends` is a string array and `i` is an integer variable. Use `for (string i : friends)` instead.

Helpful Array Algorithms

Array Algorithms

In addition to being used with loops, arrays can also be used with conditionals to help with tasks such as searching for a particular element, finding a minimum or maximum element, or printing elements in reverse order.

Searching for a Particular Element

```
string cars[] = {"Corolla", "Camry", "Prius", "RAV4",
        "Highlander"};
string Camry = "A Camry is not available."; //default string
        value

for (string s : cars) { //enhanced for loop
  if (s == "Camry") { //if "Camry" is in array
    Camry = "A Camry is available."; //variable changes if
        "Camry" exists
  }
}

cout << Camry << endl; //print whether Camry exists or not
```

challenge

What happens if you:

- delete `"Camry"` from the `cars` array?
- try to modify the code above so that the algorithm will look for `Prius` in the array and will print `A Prius is available.` if `Prius` is an element and `A Prius is not available.` if it is not an element.

▼ **Sample Solution**

```
string cars[] = {"Corolla", "Camry", "Prius", "RAV4",
        "Highlander"};
string Prius = "A Prius is not available.";

for (string s : cars) {
  if (s == "Prius") {
    Prius = "A Prius is available.";
  }
}

cout << Prius << endl;
```

Finding a Minimum or Maximum Value

```
int grades[] = {72, 84, 63, 55, 98};
int min = grades[0]; //set min to the first element in the array

for (int i : grades) { //enhanced for loop
  if (i < min) { //if element is less than min
    min = i; //set min to element that is less
  }
}
//elements are not modified so enhanced for loop can be used

cout << "The lowest grade is " << min << endl; //print lowest
        element
```

challenge

What happens if you:

- replace `72` in the int array with `42`?
- try to modify the code so that the algorithm will look for the **maximum** element instead?

▼ **Sample Solution**

```
int grades[] = {72, 84, 63, 55, 98};
int max = grades[0];

for (int i : grades) {
  if (i > max) {
    max = i;
  }
}

cout << "The highest grade is " << max << endl;
```

Printing Elements in Reverse Order

```
string letters[] = {"A", "B", "C", "D", "E"};
int elements = sizeof(letters) / sizeof(letters[0]); //number of
        elements

//start at index 4, then decrement by 1 until i < 0, then stop
for (int i = elements - 1; i >= 0; i--) {
  cout << letters[i] << endl;
}

//regular for loop needed to access each element index
```