CS 162C++
Defining, Initializing, and Using 1D Arrays

Thus far, we have used variables that are able to store a single value at a time. For example,

```
int x = 6;
```

creates a location in memory, gives it the name x, and stores the integer 6 in it. Sometimes, we want to be able to use more than one related number at a time. For example, consider keeping track of the names of the students in a class. We could do something like:

```
string name1 = "Sarah", name2 = "Bob", name3 = "Frodo", name4 = "Ann";
```

but this does not work very well when we start having classes with 20 or 30 students in them. This is when we use arrays or vectors. In this document, we will talk about arrays; vectors are considered separately.

Defining an Array

When we want to create an array, we have the same requirements as when we define any other variable. We need to specify what type of things will be stored in it and we need to give it a name. In addition, we need to specify how large the array is going to be. Some examples are

```
  // define an array named grades that will hold 10 integers
int grades[10];

  // using a constant instead of a literal to define the capacity
  // define an array named students that will hold 100 strings;
const int SIZE = 100;
string students[SIZE];
```

You use a similar approach anytime you want to create an array. Simply indicate what sort of an array it is, give it a legal name, and then using [ ] give it a desired size.

Initializing an Array

Like when using variables, it is possible to provide initial values for the **elements** of an array. When we are initializing a simple variable, we use an assignment statement as part of the definition like:

```
int value = 6;
```

For arrays we do something similar, with the caveat that we have to specify multiple values. For this, we enclose the values we want to store in the array in { }, like this:

```
int primes[7] = {2, 3, 5, 7, 11, 13, 17};
```

This will create something in memory that looks like this

primes	
	2
	3
	5
	7
	11
	13
	17

In other words, the **name** of the array is associated with seven locations in memory that sequentially hold the values that were provided in the **initialization list.** Note that the **elements** of the array are located **sequentially** in memory in **adjacent** locations. There are no spaces in memory between the elements.

If you want to only initialize part of the array, you can do that also. In that case, any elements you did not specify an initial value for are set to zero (or spaces for chars or the empty string for strings).

```
int primes[10] = {2, 3, 5, 7, 11, 13, 17};
```

This will create something in memory that looks like this

primes	
	2
	3
	5
	7
	11
	13
	17
	0
	0
	0

As a special case of this, you can initialize an entire array to zero by using empty braces {}

```
int values[6] = {};
```

This will create something in memory that looks like this:

primes	
	0
	0
	0
	0
	0
	0

Accessing Array Elements

Since an array has multiple values, you need to specify which one you want to use in a given situation. You do this by referencing the **elements index** as follows:

```
int primes[10] = {2, 3, 5, 7, 11, 13, 17};
cout << primes[3] << endl;
```

You might thing that this would display 5, since that is the third element of the array, but it actually displays 7. The **indexes start** with **0** as shown here.

primes	Element	Index
	2	0
	3	1
	5	2
	7	3
	11	4
	13	5
	17	6
	0	7
	0	8
	0	9

You can use the elements of an array in the same way you would use any other variable of that type, for example

```
cout << primes[4];
```

would display the number 11 on the console.

The instruction

```
cin >> primes[7];
```

would input an integer and store it replacing the zero after 17.

The instruction

```
primes[6] = 12;
```

would change the value 17 which is at index 6 to the number 12.

primes

Element	Index
2	0
3	1
5	2
7	3
11	4
13	5
12	6
0	7
0	8
0	9

Then the instruction

```
primes[primes[2]] = 22;
```

would change the value 13, which is at index 5 (the value of the element at index 2) to 22.

primes

Element	Index
2	0
3	1
5	2
7	3
11	4
22	5
12	6
0	7
0	8
0	9

Just as a comment, you normally do not do things like the above, I am only including that here to let you know that it is possible if you want.

Using For Loops with Arrays

Since arrays occupy sequential locations, the for loop is a natural way to access the elements of one. For example, to output the primes above, you would use the following bit of code:

```
const int SIZE = 10;
int primes[SIZE] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};

for(int i = 0; i < SIZE; i++)
    cout << primes[i] << endl;
```

For another example, to initialize an array by reading numbers from the console, you could do this:

```
const int SIZE = 10;
int values[SIZE];

for(int i = 0; i < SIZE; i++)
{
      cout << "Enter the next number" << endl;
      cin >> values[i];

}
```