Arrays

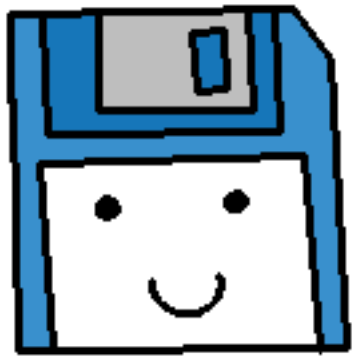
Written by Rachel J. Morris, last updated 2015-01-14

What are arrays?

In C++, arrays are the simplest data-type for storing a series (list, array) of values.

Some characteristics of simple arrays:

- All elements stored in the list must be of the same data-type
- There are no functions for our arrays – for example, we must keep track of the array's size manually
- An array cannot be resized; you set a specific size during its declaration and it cannot change.



What are arrays?

Arrays can store any data-type that you could use for a variable:

`int`

`string`

`float`

`bool`

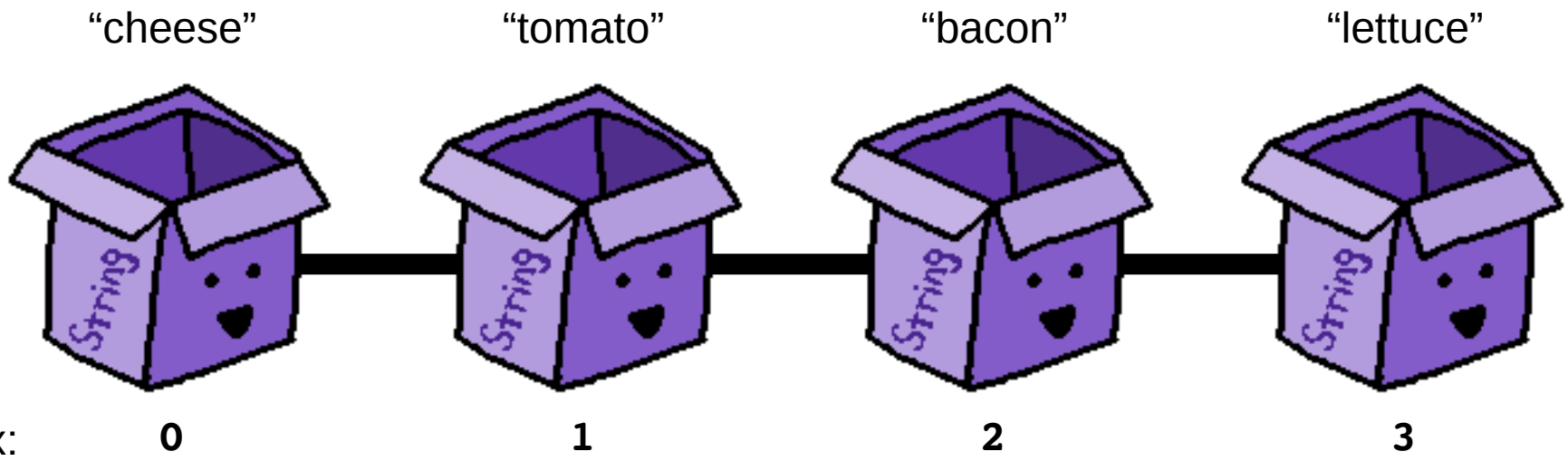
Even user-made objects

What are arrays?

Each element in an array has its own index, which is its position in the array.

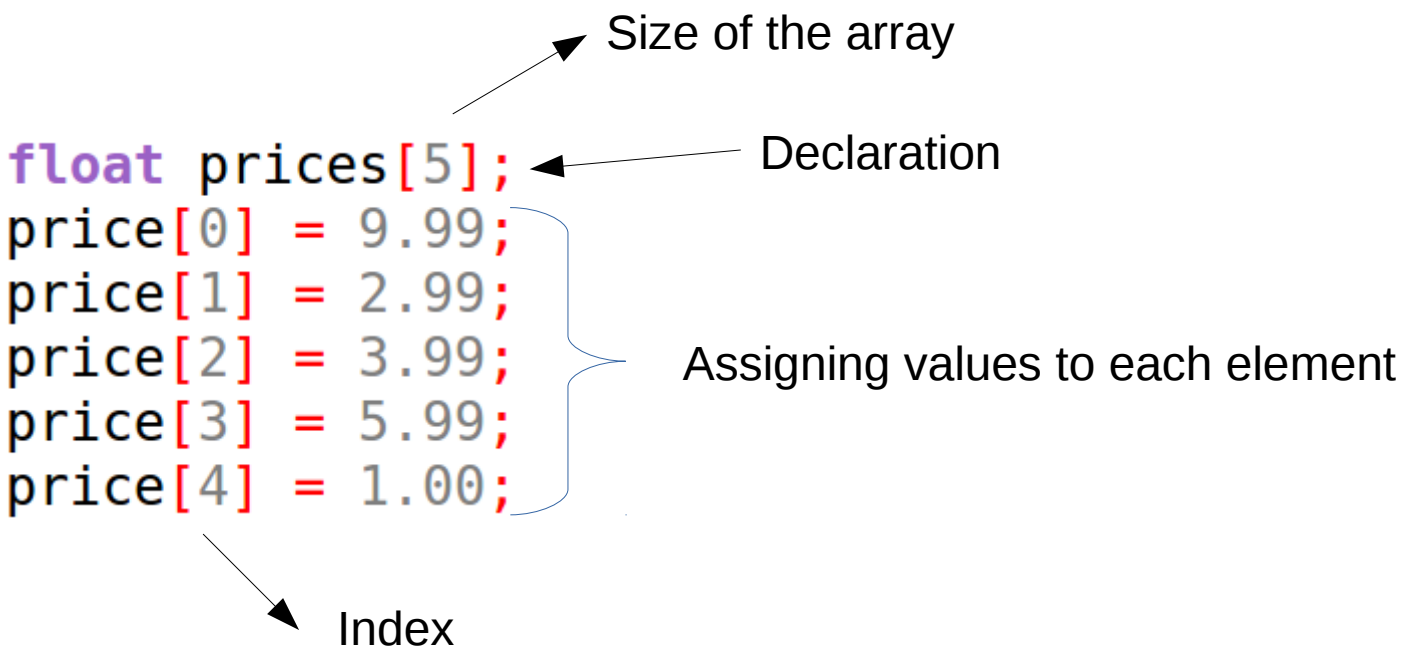
The first index is 0

In an array of size 4, the first element is at index 0 and the last one is at index 3.



Array declaration

Like a normal non-array variable, an array declaration requires the **data-type** and the **name** of the variable, but we must also specify the array's size when creating it.



Array declaration

We can declare an array without the size specified **only** if we are initializing the array with a list of data.

```
float prices[] = { 9.99, 2.99, 3.99, 5.99, 1.00 };
```

0 1 2 3 4

We still need the square brackets `[ ]` to specify that this is an array.

For Loops and Arrays

For loops are really handy once we're using arrays.

We can use a for loop to iterate through all the elements of an array

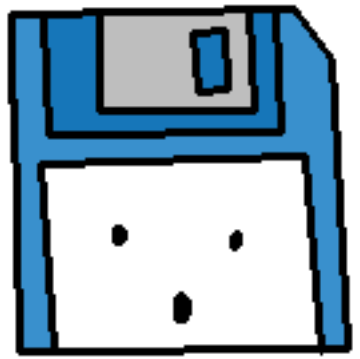
```
float prices[5] = { 9.99, 2.99, 3.99, 5.99, 1.00 };

for ( int i = 0; i < 5; i++ )
{
    cout << "Item " << i << ": $" << prices[i] << endl;
}
```

```
Item 0: $9.99
Item 1: $2.99
Item 2: $3.99
Item 3: $5.99
Item 4: $1
```

Why no resizing?

When we get into **memory management**, we will learn how to make **dynamic arrays** – arrays that can be assigned a size at run-time, instead of hard-coded into the program code.

The reason the C++ array does not have the ability to resize deals with memory allocation.

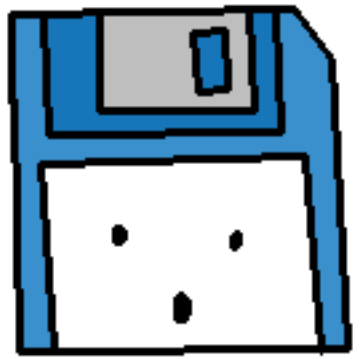


Why no resizing?

When an array is declared, as with any variable, memory is set aside for this item.

The amount of memory taken up is dependent on the data-type.

int and float = 4 bytes, bool = 1 byte.



Why no resizing?

An array allocates enough memory for ***n*** amount of that data-type, where ***n*** is the size of the array.


```
int number;
```

= 4 bytes

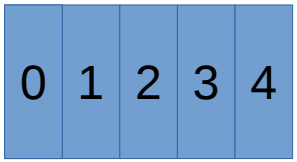

```
int numbers[5];
```

5 elements, 4 bytes each
= 20 bytes

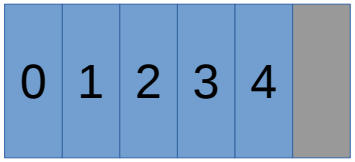
Y U NO RESIZE?

Additionally, the **elements** in the array are stored sequentially in memory – one after another.

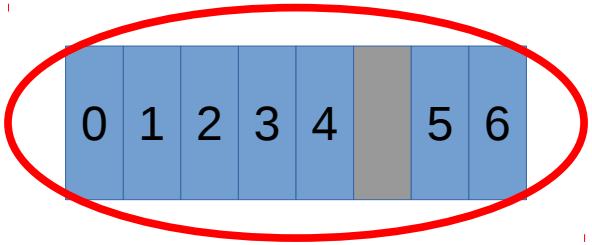
- Let's say we create an array of size 5, and it takes up 5 memory “blocks”.



- But then we create another variable, which takes up the memory “block” after our array.



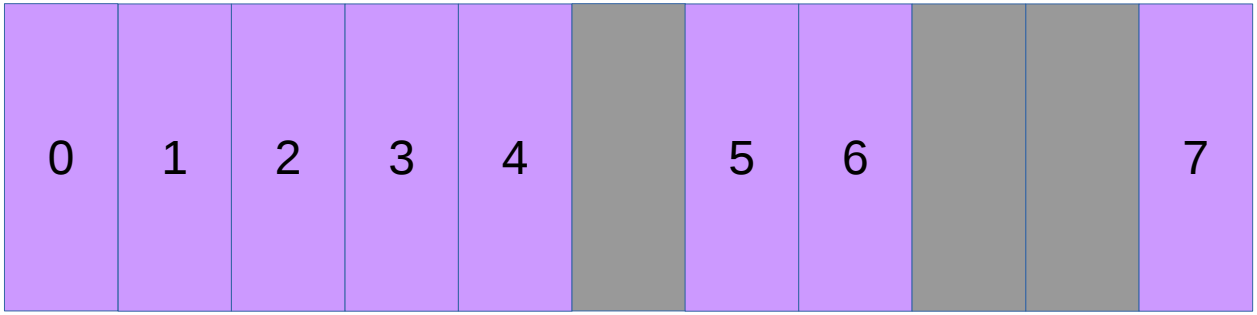
- Elements in an array must be sequential, but we cannot add to the array because that memory is taken up by something else.



NOPE

Y U NO RESIZE?

Well, why ***can't*** we store different elements in an array at different places in memory?

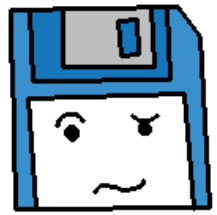


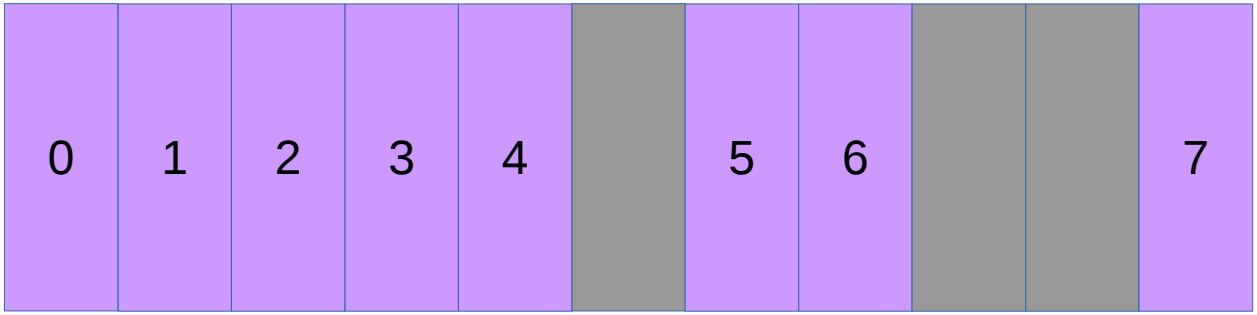
Arrays in C++ aren't “smart” enough to handle keeping track of its elements like this.

It knows enough to “walk” through its elements, and how big each element is (because they're all the same data-type!)

Y U NO RESIZE?

Why can't C++ do it? ***I*** can do that!



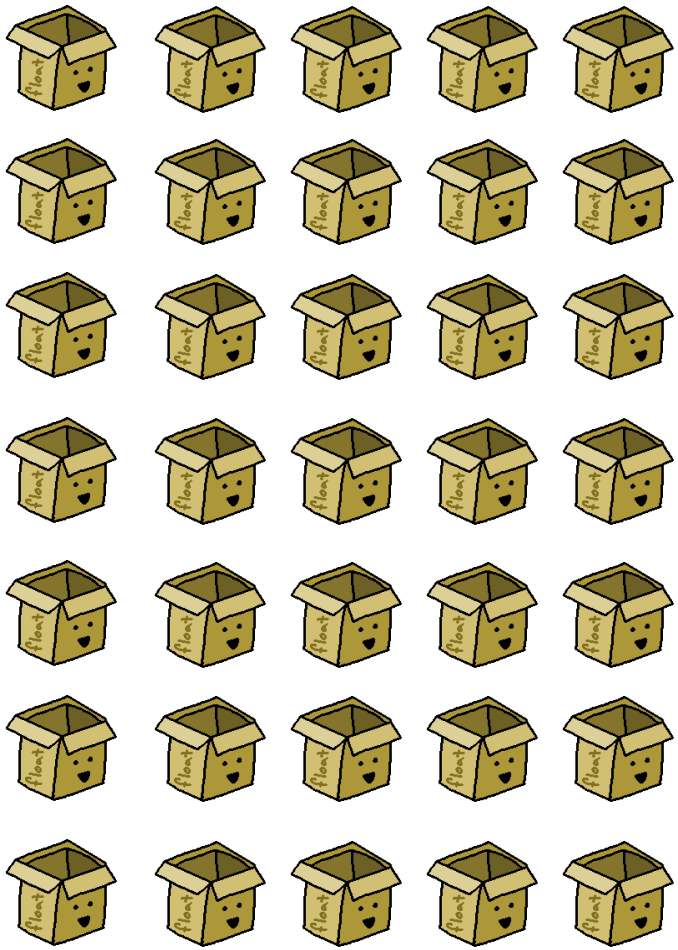


We can do that – we can write a **class** (object) that intelligently handles elements in an array, so we can **resize** the array whenever we want!

We will do this later on, after we've covered **pointers** and **classes**.

Arrays as Arguments

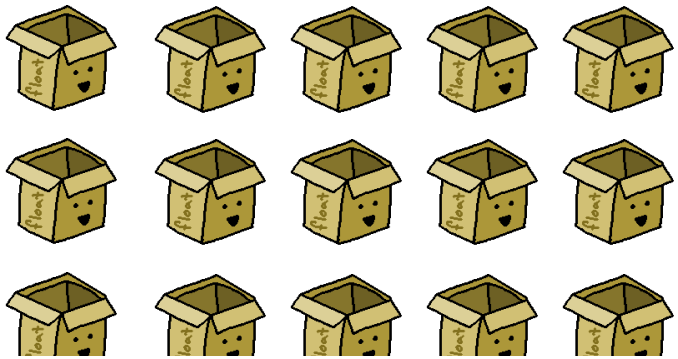
So, arrays can potentially take up a lot of memory.



Because of this, when arrays are passed into a function call as an argument, they are automatically passed by **reference**, not passed by **value**.

If they were passed by **value** by default, like non-array variables, your program would be really inefficient, and could potentially slow down. A lot.

Arrays as Arguments



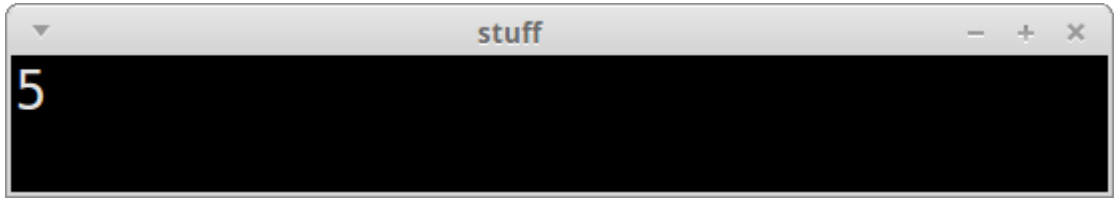
```
stuff
0
10
20
30
40

Process returned 0 (0x0)   ex
Press ENTER to continue.
```

```
void AddValues( int numbers[5] )
{
    for ( int i = 0; i < 5; i++ )
    {
        numbers[i] = i * 10;
    }
}

int main()
{
    int mylist[5];
    AddValues( mylist );
    for ( int i = 0; i < 5; i++ )
    {
        cout << mylist[i] << endl;
    }

    return 0;
}
```

Arrays as Arguments


```
void Modify( int value )
{
    value = 2;
}

int main()
{
    int number[5];

    number[0] = 5;
    Modify( number[0] );
    cout << number[0] << endl;

    return 0;
}
```

However, if we just pass one element of an array to a function, it **is** copied over, like any other non-array variable.



Arrays as Arguments

But what if we want to pass an array to a function,
But we want to make sure the data in the array doesn't change?

Time for **const!**

```
void Print( const int numbers[5] )
{
    for ( int i = 0; i < 5; i++ )
    {
        cout << numbers[i] << endl;
    }
}
```

Like with other pass by reference parameters, you can use **const** so that you can get the benefit of the speed of passing by reference, with the benefit of not being able to change a value, like with pass by value.

Multidimensional Arrays

Our arrays don't need to be one dimensional. We can have arrays that are 2D, 3D, and more!

```
float vertices[4][4][4];
```

```
float pixels[32][32];
```

Though if you find that your arrays are getting really complex, you might want to create a **class** or **struct** instead...

(More on that in another lecture)

Problems with Arrays

Arrays are pretty basic, so there are some problems you will need to look out for when writing programs.

Problems with Arrays

Arrays are pretty basic, so there are some problems you will need to look out for when writing programs.

For one, arrays don't keep track of what their size is. You have to manually keep track of this.

```
int numbers[3] = { 2, 4, 6 };
cout << numbers[5] << endl;
```

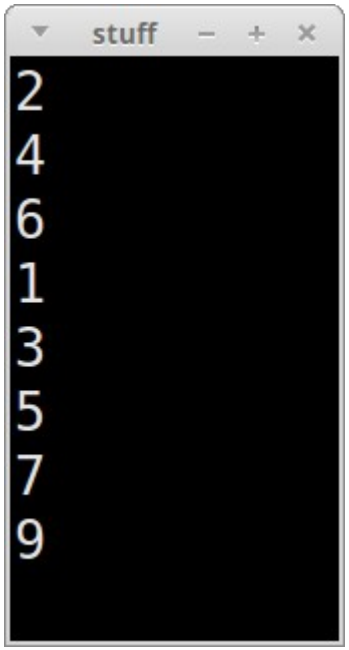
Keep in mind the size!

Because of that, you will need to be careful to make sure the program does not try to access an index outside of the array – causing an ***index out of range*** error.

Problems with Arrays

When passing an array, the size isn't required to be specified.

```
void Display( const int numbers[], int length )
{
    for ( int i = 0; i < length; i++ )
    {
        cout << numbers[i] << endl;
    }
}

int main()
{
    int even[3] = { 2, 4, 6 };
    int odd[5] = { 1, 3, 5, 7, 9 };

    Display( even, 3 );
    Display( odd, 5 );

    return 0;
}
```



Problems with Arrays

Arrays are pretty basic, so there are some problems you will need to look out for when writing programs.

For one, arrays don't keep track of what their size is. You have to manually keep track of this.

```
int numbers[3] = { 2, 4, 6 };
cout << numbers[5] << endl;
```

Keep in mind the size!

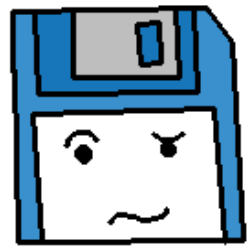
Because of that, you will need to be careful to make sure the program does not try to access an index outside of the array – causing an ***index out of range*** error.

Problems with Arrays

Another problem is that, even if you created an array of some size, that doesn't mean all the elements have been filled.

Beyond the maximum array size, you might also need to store how many elements are currently in the array.

If you access an element that has not been initialized, you will get **garbage**, just like if you declare a non-array variable without initializing it.



Problems with Arrays

```
void AddStudent( string students[30], string name, int& total )
{
    students[ total ] = name;
    total++;
}

int main()
{
    string students[30];
    int totalStudents = 0;

    AddStudent( students, "Usagi", totalStudents );
    AddStudent( students, "Ami", totalStudents );
    AddStudent( students, "Rei", totalStudents );
    AddStudent( students, "Makoto", totalStudents );
    AddStudent( students, "Minako", totalStudents );

    cout << "Total students: " << totalStudents << endl;

    return 0;
}
```

```
Total students: 5
```

When working with arrays and adding elements in the program, make sure to keep track of what the size of the array is (or, the last index used).

Being able to store a list of like-data is super useful, but it can also be tricky.

Let's work on some programs that will illustrate how we can use arrays