



Vectors

C++ Notes

This document contains a brief description of vectors, a comparison of vectors to raw arrays, and a brief description of how to use some vector library functions in code.

What is a Vector?

A **vector** is a type of data structure that is VERY SIMILAR to an **ARRAY**, but with some major improvements. It is used to store a list of data items. This is exactly what you have been using an array to do.

How is a Vector DIFFERENT from an Array?

- A vector will NOT allow you to cause **range-bound-errors**. When you use arrays, and you accidentally go out of bounds, there is a chance that your program is actually destroying something else (and potentially totally unrelated to your program) in memory. For example, if you have declared an array of integers like this:

```
int myArray[3];
```

And then you attempt to do the following:

```
myArray[8]=10;
```

Your computer will not stop you. Your program might crash...or it might run and cause some unknown effects in some other area of memory.

A vector, on the other hand, will NOT allow you to make this mistake. It will, when the program runs, throw an error and stop you from overwriting something important.

- The vector library includes a function that allows a vector to tell you its **size** (this will be shown in the section on coding). The array has no associated functions.
- It is possible to **resize** (both grow and shrink) a vector as needed while the program is running. It was not possible to resize an array, you were simply stuck with the size you determined at declaration.

How do I Program a Vector?

First you need a **preprocessor directive** to utilize the vector library. Add this to your program along with your usual preprocessor directives:

```
#include<vector>
```

Next you'll need to **declare** a specific vector. This looks a little different than with arrays. The format is:

```
vector<Type> variableName(size);
```

For example, if you want a vector that initially has room for 5 integers, your declaration would look like this:

```
vector<int>myList(5);
```

This will build a vector with 5 cells.

To **access individual cells**, the notation looks identical to that used with arrays. For example if I want to iterate through the vector, I can use a loop:

```
for(int i=0; i<5; i++)  
{  
    cout<<myList[i];  
}
```

To access a particular cell directly, the format is the same. If I want to place a value in the third cell, I can write:

```
cin>>myList[2];
```

If I want to **load EVERY cell** with the SAME value when the vector is declared, I can use the notation:

```
vector<Type>variableName(numCells, valueToFill);
```

So, for example, the following code will create my list initialized with all 10's inside:

```
vector<int>myList(5,10);
```

If we want to ask the vector for its current **size**, we can call its function using a **"dot operator"**:

```
cout<<"The vector's size:"<<myList.size();
```

This is particularly useful if we don't know in advance the number of cells the vector will end up having before we write a loop to iterate through it. We can write a more generic loop that's based on the `size()` function, and avoid having to go back and change our loop later – because our code will work no matter what size the vector is:

```
for(int i=0; i<myList.size(); i++)
{
    cout<<myList[i]<<endl;
}
```

If we want to **resize** a vector, we can call its function. The format for this is:

```
myVectorName.resize(newSizeToMakeTheVector);
```

If we want to resize our example vector to have 10 cells, the code would be:

```
myList.resize(10);
```

If we resize the vector to be **SMALLER**, we will lose any data contained in cells numbered past the end of the new size. If we resize the vector to be **LARGER**, we retain any data in cells numbered lower (from the original size of the vector).