Contents

[Introduction to Vectors 1](#_Toc57013197)

[Creating a Vector 1](#_Toc57013198)

[Initializing a Vector 2](#_Toc57013199)

[Vector Index 3](#_Toc57013200)

[Adding and Removing Elements 3](#_Toc57013201)

[.size() 4](#_Toc57013202)

[Operations 5](#_Toc57013203)

## Introduction to Vectors

To do just about anything of interest in a program, we need a group of data to work with.

For example, our program might need:

* A list of Twitter’s trending tags
* A set of payment options for a car
* A catalog of eBooks read over the last year

The need for storing a collection of data is endless.

We are familiar with data types like int and double, but how do we store a group of ints or a group of doubles?

In this lesson, we will start with one of the simplest, and arguably the most useful, ways of storing data in C++: a vector.

A *vector* is a sequence of elements that you can access by index.

## Creating a Vector

The std::vector lives in the <vector> header. So first, we need to add this line of code at the top of the program:

#include <vector>

For review, #include is a preprocessor directive that tells the compiler to include whatever library that follows. In our case that is the standard vector library.

And the syntax to create a vector looks like:

std::vector<type> name;

So to define a vector of ints called calories\_today:

std::vector<int> calories\_today;

Inside the angle brackets is the data type of the vector. After the angle brackets is the name of the vector.

**Note:** The type of the vector (i.e., what data type is stored inside) cannot be changed after the declaration.

## Initializing a Vector

Now we know how to create a vector, we can also initialize a vector, giving it values, as we are creating it in the same line.

For example, instead of just creating a double vector named location:

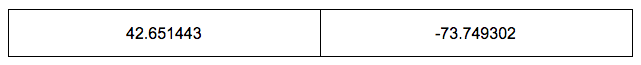
std::vector<double> location;

We can create and initialize location with specific values:

std::vector<double> location = {42.651443, -73.749302};

Here, we are storing [a latitude and a longitude](https://en.wikipedia.org/wiki/Geographic_coordinate_system).

So it would look something like this:



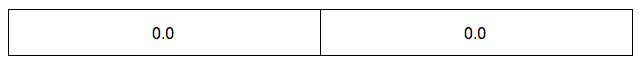
Another way we can initialize our vector is by presizing, or setting the size.

Suppose we want to create and initialize a vector with two elements. However, we don’t know what values we want to add yet:

std::vector<double> location(2);

Here, we are creating a double vector and setting the initial size to two using parentheses.

It would look something like this:



Because 0.0 is the default value for double.

## Vector Index

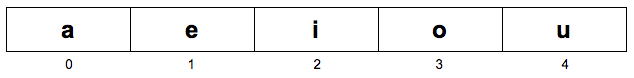
Now that we have a vector, how do we access an individual element? This is where index comes into play.

An index refers to an element’s position within an ordered list. Vectors are 0-indexed, meaning the first element has index 0, the second index 1, and so on.

For example, suppose we have a char vector with all the [vowels](https://en.wikipedia.org/wiki/Vowel):

std::vector<char> vowels = {'a', 'e', 'i', 'o', 'u'};

It should look something like this:



* The character at index 0 is 'a'.
* The character at index 1 is 'e'.
* The character at index 2 is 'i'.
* The character at index 3 is 'o'.
* The character at index 4 is 'u'.

To output each of the elements, we can do:

std::cout << vowels[0] << "\n";  
std::cout << vowels[1] << "\n";  
std::cout << vowels[2] << "\n";  
std::cout << vowels[3] << "\n";  
std::cout << vowels[4] << "\n";

Using the notation vector[index] with square brackets after the vector name and the element’s index number inside.

This will output:

a  
e  
i  
o  
u

## Adding and Removing Elements

Often, we start with a vector that’s either empty or a certain length. As we read or compute data we want, we can grow the vector as needed.

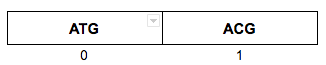
##### .push\_back()

To add a new element to the “back”, or end of the vector, we can use the .push\_back() function.

For example, suppose we have a vector called dna with three letter codes of [nucleotides](https://en.wikipedia.org/wiki/Nucleotide):

std::vector<std::string> dna = {"ATG", "ACG"};

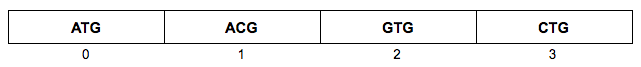
It would look like:



We can add elements using .push\_back():

dna.push\_back("GTG");

So now dna would look like:



##### .pop\_back()

You can also remove elements from the “back” of the vector using .pop\_back().

dna.pop\_back();

Notice how nothing goes inside the parentheses.

The vector would now look like:

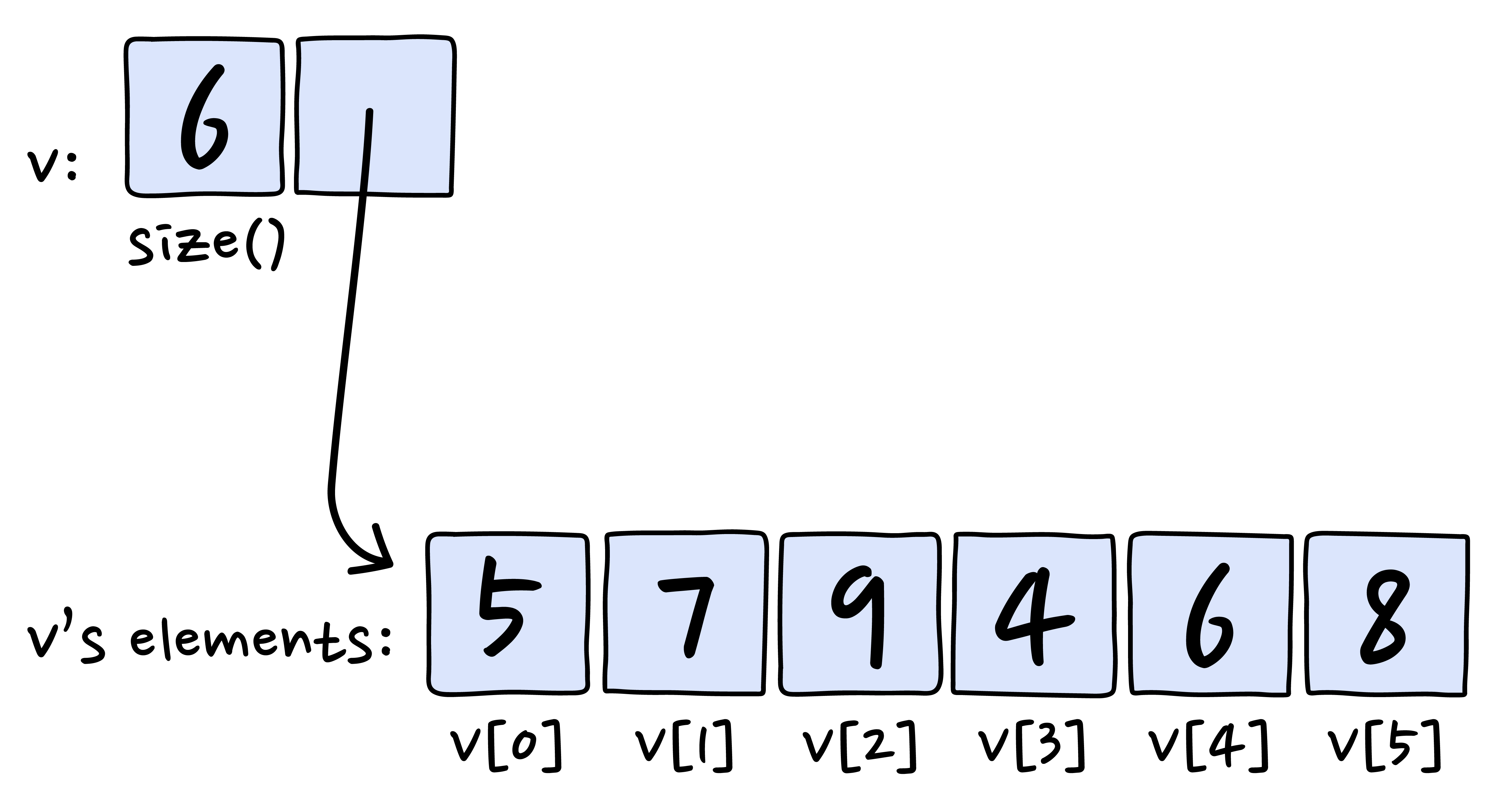
3

because CTG is removed!

**Note:** If you have programmed in other languages, be aware that .pop\_back() has no return value in C++.

## .size()

<std::vector> not only stores the elements; it also stores the size of the vector:

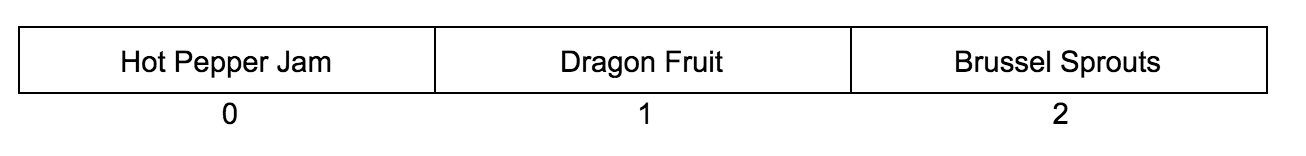


The .size() function returns the number of elements in the vector.

For example, suppose we have a std::string vector with Sonny’s grocery list:

std::vector<std::string> grocery = {"Hot Pepper Jam", "Dragon Fruit", "Brussel Sprouts"};

It should look something like this:



* The string at index 0 is "Hot Pepper Jam".
* The string at index 1 is "Dragon Fruit".
* The string at index 2 is "Brussel Sprouts".

std::cout << grocery.size() << "\n";

will return

3

Notice how nothing goes in the parentheses.

## Operations

So what happens when you want to change each of the values within a vector?

You can use a for loop!

For example, suppose we have an int vector that looks like this:

1

You can write a for loop that iterates from 0 to vector.size(). And here’s the cool part: you can use the counter of the for loop as the index! Woah.

for (int i = 0; i < vector.size(); i++) {  
   
  vector[i] = vector[i] + 10;  
   
}

This will change the vector to:

2

Here, we incremented i from 0 to vector.size(), which is 3. During each iteration, we are adding 10 to the element at position i:

* When i = 0, we added 10 to vector[0]
* When i = 1, we added 10 to vector[1]
* When i = 2, we added 10 to vector[2]