Learning Objectives: Vector Basics

- **Create an empty vector**
- **Add and remove vector elements**
- **Modify vector elements**
- **Iterate through vector using both a regular `for` loop and an *enhanced* `for` loop**
- **Determine vector output**
- **Determine key differences between vectors and arrays**

Creating a Vector

What Is a Vector?

Although arrays are very useful for data collection, they are considered **static**, meaning once they are created, you cannot add or remove elements from them without changing the way they are initialized. **Vectors**, on the other hand, are **dynamic**, meaning you can make changes to them while the program is running. Vectors are particularly helpful when you don't know how large your collection of elements will become. Since vectors are dynamic, you can add and remove elements later on if needed. In order to use vectors, you must include `#include <vector>` in the header of your program. For convenience, the program file to your left already contains the included statement.

Vector Creation

To create a vector, you need to include the following:
* The keyword `vector` followed by the data type in angle brackets `<>`.
* A variable name that refers to the vector.
* The number of elements the vector can hold within parentheses `()`.

```
vector<int> numbers(3);

cout << numbers << endl;
```

→ output memory space

important

IMPORTANT

When you try to print an array reference variable, you will get the array's memory address. However, this is **not** the case for vectors.

To print an element within the vector, use the `at()` function and specify the index of the position of the element you wish to print within the parentheses.

```
vector<int> numbers(3);

cout << numbers.at(0) << endl;
```

Similar to arrays, the first index or position of the vector also starts at index `0`. Thus, `numbers.at(0)` refers to the element at the first position in the vector, which currently does not contain any initialized elements. When a vector is declared without any initialized elements, the system will populate the vector with `0` as elements by default. This is common across vectors of many data types except strings. Additionally, if you try to output an element at an index that does not exist, you will get an `out_of_range` error message.

```
vector<int> numbers(3);

cout << numbers.at(3) << endl;
//index 3 refers to the fourth element, not third, which doesn't
        exist
```

Determining Vector Size

Vectors use the **function** `size()` to determine the number of elements that exist instead of the **operator** `sizeof()` which is used for arrays. Note their slight difference in syntax and usage.

```
vector<int> numbers(3);
int digits[3];

cout << numbers.size() << endl;
cout << sizeof(digits) / sizeof(digits[0]) << endl;
```

challenge

What happens if you:

- remove `3` within `vector<int> numbers(3);`?
- remove `3` within `int digits[3];`?

important

IMPORTANT

In both arrays and vectors, you must specify how many elements you expect them to hold. Otherwise, you will not be able to determine their size. However, if you initialize the array or vector upon declaration, then you don't have to specify the number of elements since the system can determine that automatically.

Adding and Removing Elements

Adding Vector Elements

To add elements to the vector, simply use the `push_back()` function. The `push_back()` function will add whatever element that is specified inside parentheses `()` to the **end** of the vector. If an element is added to an empty vector such as `vector<int> numbers(0)`, that element will be the first and only element in the vector.

```
vector<int> numbers(0); //vector with no elements
numbers.push_back(50); //add 50 as an element to end of vector

cout << numbers.at(0) << endl; //50 becomes first and only
        element
```

Note that the `push_back()` function does **not** replace elements.

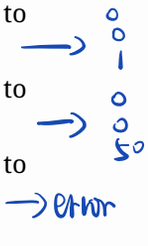
```
vector<int> numbers(2); //vector with 2 uninitialized elements
numbers.push_back(50); //add 50 as an element to end of vector

cout << numbers.at(0) << endl; //first uninitialized element
cout << numbers.at(1) << endl; //second uninitialized element
cout << numbers.at(2) << endl; //50 is the third element now
```

challenge

What happens if you:

- change `numbers.push_back(50);` in original code to `numbers.push_back(true);`?
- change `numbers.push_back(50);` in original code to `numbers.push_back(50.99);`?
- change `numbers.push_back(50);` in original code to `numbers.push_back("50.99");`? **Note:** “50.99”



important

IMPORTANT

Arrays are strict when it comes to data type compatibility, however, vectors are more flexible. Between the four common data types, string is the only type that cannot be associated with the other three in a vector. Integers, doubles, and booleans are all compatible with each other. Remember, in C++, `true` is `1` and `false` is `0`.

In addition, when doubles are converted into integers, their decimal value get eliminated. This is why pushing `50.99` into an `int` vector causes it to turn into `50` without the decimal value.

To add an element to a **specific index** in the vector, you can use the `insert()` along with the `begin()` functions like below.

```
vector<int> numbers(2);
numbers.insert(numbers.begin()+1, 50); //add 50 to index 1
cout << numbers.at(0) << endl;
cout << numbers.at(1) << endl;
cout << numbers.at(2) << endl;

numbers.insert(numbers.begin()+1, 100); //add 100 to index 1
cout << numbers.at(0) << endl;
cout << numbers.at(1) << endl;
cout << numbers.at(2) << endl; //50 now becomes index 2
```

challenge

What happens if you:

- change `numbers.insert(numbers.begin()+1, 50);` in the original code to `numbers.insert(numbers.begin()+2, 50);`?
- change `numbers.insert(numbers.begin()+1, 100);` in the original code to `numbers.insert(numbers.begin(), 100);`?

important

IMPORTANT

The `begin()` function always refer to the first position in the vector, which is also the 0th index. If you want to refer to the 1st index, use `begin()+1`. For the 2nd index, use `begin()+2`, so on and so forth.

Removing Vector Elements

To remove an element from the **end** of a vector, use the `pop_back()`. Note that using `pop_back` will remove the element and its index, thus decreasing the size of the vector by 1.

```
vector<int> numbers(0); //empty vector
numbers.push_back(50); //add 50 to vector
numbers.push_back(100); //add 100 to vector
cout << numbers.at(0) << endl;
cout << numbers.at(1) << endl << endl;

numbers.pop_back(); //remove last element vector
cout << numbers.at(0) << endl;
cout << numbers.at(1) << endl; //100 has been deleted completely
```

→ error

challenge

What happens if you:

- remove `cout << numbers.at(1) << endl;` from the original code?

→ 50
100
50

To remove an element from a **specific index** in the vector, use the `erase()` function and specify the index you want to erase with `begin()`. When an element and its index is removed from the vector, all of the elements to its right will be moved one place to the left.

```
vector<int> numbers(0); //empty vector
numbers.push_back(50); //add 50 to vector
numbers.push_back(100); //add 100 to vector
cout << numbers.at(0) << endl;
cout << numbers.at(1) << endl << endl;

numbers.erase(numbers.begin()); //removes 50
cout << numbers.at(0) << endl;
cout << numbers.at(1) << endl; //no longer exists
```

→ error

challenge

What happens if you:

- remove `cout << numbers.at(1) << endl;` from the original code?
- change `numbers.erase(numbers.begin());` from the original code to `numbers.erase(numbers.begin()+1);`?

→ 50
100

100

Modifying Elements

Modifying Vector Elements

To modify vector elements, use the `at()` method to specify the index number and then assign a new element to it.

```
vector<string> contact(0);
contact.push_back("First name");
contact.push_back("Last name");
contact.push_back("Phone number");
cout << contact.at(0) << " "
     << contact.at(1) << " "
     << contact.at(2) << endl;

contact.at(2) = "Email"; //change element at index 2 to "Email"
cout << contact.at(0) << " "
     << contact.at(1) << " "
     << contact.at(2) << endl;
```

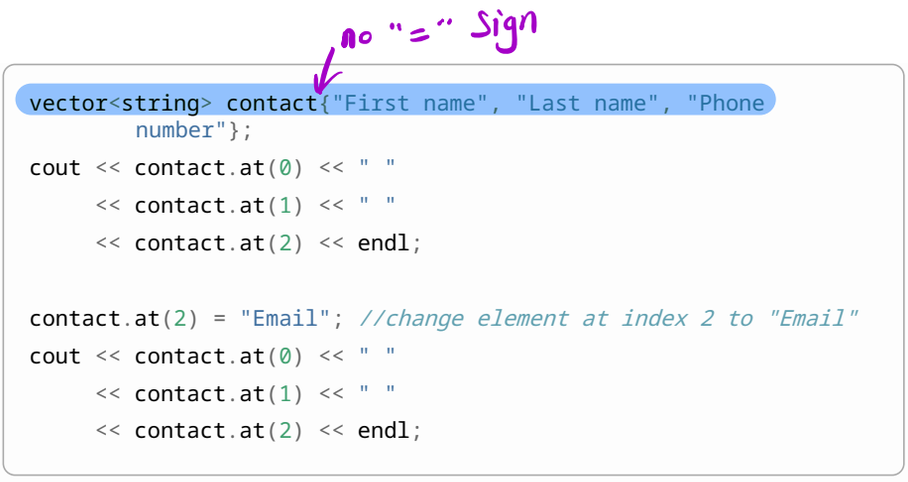
challenge

What happens if you:

- add `contact.at(1) = "Nick name";` to the line directly before `contact.at(2) = "Email";`?

Initializing Vector Elements

It is possible to initialize elements inside a vector without constantly using `push_back()`. The following code will produce the same result as the original code above.



When initializing elements within a vector, you do not specify the number of elements in parentheses. The system will automatically know how many elements are being added to the vector. The initialized elements should be in curly braces `{}` and separated by commas `,`.

Iterating a Vector

Iterating Vector Elements

Iterating through a vector is very similar to iterating through an array. The main difference is that in a vector, we use `at()` to access the elements instead of brackets `[]`. Both of the code blocks below use a regular `for` to produce the exact same results. The first code block contains an array and the second contains a vector.

```
//iterating through an array
int grades[] = {85, 95, 48, 100, 92};

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

```
//iterating through an vector
vector<int> grades{85, 95, 48, 100, 92};

for (int i = 0; i < grades.size(); i++) {
  cout << grades.at(i) << endl;
}
```

Enhanced For Loop in Vector

We can also use an **enhanced for loop**, or **range-based for loop**, to iterate through a vector.

```
//iterating a vector with Enhanced For Loop
vector<int> grades{85, 95, 48, 100, 92};

for (auto i : grades) { //can use int or auto for iterating
        variable!
  cout << i << endl;
}
```

important

IMPORTANT

When using an enhanced `for` loop for a vector, you must label the iterating variable accordingly. If your elements are of type `int` then your iterating variable must also be `int`. If the elements are `string`s then your variable must be typed as `string`. However, you can always use `auto` to force the variable to match your element type.

Vector vs. Array

Vector vs. Array

Which one is better: vector or array? The answer is, it really *depends*. If you know how many elements you need in your collection and you don't intend on changing the order of those elements, then it is better to use an **array**. On the other hand, if you don't know how many elements you need and want to modify the order of elements later on, then it is better to use a **vector**.

Although an array is shorter to write and arguably easier to use, it is **static**, meaning it is not possible to add additional elements to the array after it has already been initialized. In contrast, a vector is more **dyanamic**, meaning you can add, remove, and reorganize elements as needed later on.

Here is a table showing the differences between vectors and arrays. Note that `type` stands for data type. Also note that `var` stands for vector or array name, `num` stands for an integer number, `index` stands for index or position number, and `element` stands for a vector or array element.

Method/Types	Vector	Array
Create	vector<type> var(num) or vector<type> var{element, element...}	type var[num] or type var[] = {element, element...}
Find number of elements	var.size()	sizeof(var)/sizeof(var[0])
Access an element	var.at(index)	var[index]
Modify an element	var.at(index) = element	var[index] = element
Add an element	var.push_back(element) or var.insert(var.begin()+index, element)	n/a
Remove an element	var.pop_back() or var.erase(var.begin()+index)	n/a
for loop	for (int i = 0; i < var.size(); i++) {cout << var.at(i);}	for (int i = 0; i < sizeof(var)/sizeof(var[0]); i++) {cout << var[i];}
Enhanced for loop	for (type i : var) {cout << i}	for (type i : var) {cout << i}

Common compatible types	integer, double, boolean, strings	int, double, boolean, strings

Using Both a Vector and Array

Vectors and arrays can be used in tandem with each other. For example, the following code keeps track of the top five students in a class.

```
string top[] = {"First: ", "Second: ", "Third: ", "Fourth: ",
        "Fifth: "};
vector<string> names(0);

names.push_back("Alan");
names.push_back("Bob");
names.push_back("Carol");
names.push_back("David");
names.push_back("Ellen");

for (int i = 0; i < 5; i++) {
  cout << top[i] << names.at(i) << endl;
}
```

challenge

Without deleting any existing code, try to:

- switch `Alan` and `Carol`'s places.
- replace `David` with `Fred`.
- make `Grace` get "Fifth" place and remove `Ellen` from the list.

▼ **Sample Solution**

```
string top[] = {"First: ", "Second: ", "Third: ", "Fourth: ",
        "Fifth: "};
vector<string> names(0);

names.push_back("Alan");
names.push_back("Bob");
names.push_back("Carol");
names.push_back("David");
names.push_back("Ellen");

names.at(0) = "Carol"; //switch Alan with Carol
names.at(2) = "Alan";  //and vice versa

names.at(3) = "Fred"; //Fred replaces David

names.insert(names.begin()+4, "Grace"); //Grace takes Ellen's
        place
names.pop_back(); //Ellen's "Sixth" place gets removed

for (int i = 0; i < 5; i++) {
  cout << top[i] << names.at(i) << endl;
}
```

Helpful Vector Algorithms

Vector Algorithms

Like arrays, vectors can be used to search for a particular element and to find a minimum or maximum element. Additionally, vectors can reverse the order of elements rather than just simply printing the elements in reverse order.

Searching for a Particular Element

```
vector<string> cars(0);
string Camry = "A Camry is not available."; //default string
        value


cars.push_back("Corolla");
cars.push_back("Camry");
cars.push_back("Prius");
cars.push_back("RAV4");
cars.push_back("Highlander");

for (auto a : cars) { //enhanced for loop
  if (a == "Camry") { //if "Camry" is in vector
    Camry = "A Camry is available."; //variable changes if
        "Camry" exists
  }
}

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

challenge

What happens if you:

- add `cars.erase(cars.begin()+1);` to the line directly below `cars.push_back("Highlander");`?
- 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**

```
vector<string> cars(0);
string Prius = "A Prius is not available.";

cars.push_back("Corolla");
cars.push_back("Camry");
cars.push_back("Prius");
cars.push_back("RAV4");
cars.push_back("Highlander");

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

cout << Prius << endl;
```

Finding a Minimum or Maximum Value

```
vector<int> grades(0);
grades.push_back(72);
grades.push_back(84);
grades.push_back(63);
grades.push_back(55);
grades.push_back(98);

int min = grades.at(0); //set min to the first element in the
        array

for (auto a : grades) { //enhanced for loop
  if (a < min) { //if element is less than min
    min = a; //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:

- add `grades.at(0) = 42;` to the line directly below `grades.push_back(98);`?
- try to modify the code so that the algorithm will look for the **maximum** element instead?

▼ **Sample Solution**

```
vector<int> grades(0);
grades.push_back(72);
grades.push_back(84);
grades.push_back(63);
grades.push_back(55);
grades.push_back(98);

int max = grades.at(0);

for (auto a : grades) {
  if (a > max) {
    max = a;
  }
}

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

Reversing the Order of Elements

```
vector<string> letters(0);
letters.push_back("A");
letters.push_back("B");
letters.push_back("C");
letters.push_back("D");
letters.push_back("E");

int original = letters.size(); //original size

//regular for loops needed to access element indices

for (int i = letters.size() - 1; i >= 0; i--) {
  letters.push_back(letters.at(i));
} //add elements in reverse order to the vector

for (int j = 0; j < original; j++) {
  letters.erase(letters.begin());
} //remove all the original elements

//enhanced for loop can be used for just printing
for (auto a : letters) {
  cout << a << " "; //print all new vector elements
}
```

important

IMPORTANT

Note that we used `letters.erase(letters.begin())` which causes the system to delete both the **element** and the **index**. Thus, the next element in the vector becomes the *new* 0th index which we want to continue to delete.