C++ dynamic array topic.

C++ list topic. C++ container topic.
Called Vectors in C++. Not to be confused with the
dynamically-allocated array.
Also there are Lists in C++ (std::list),
which work exactly the same, except they are faster
when inserting or deleting elements (technically,
just as fast when doing it at the ends), but
you can't use the subscript operator with them.
Also see list loops topic for another potential difference.

Other containers.

vector	Allows random-access with the [subscript] operator
list	Facilitates efficient insertion into and deletion from the middle
stack	First-in, last-out container
queue	First-in, first-out container
deque	Efficient access to either end of the container
set	Stores a collection of unique elements
map	Associative collection of a key/value pair
priority_queue	Efficient retrieval of the highest-value element in the collection
unordered_set	Hash collection with extremely fast insertion, deletion, and retrieval

Taken from
file:///C:/Users/edhth/OneDrive/Documents/CPPLanguage/Week1/04%20Containers.htm

C++ dynamic array methods.

define()

```
vector<int> v = {1, 2, 3}; // #include <vector>
vector v = {1, 3, 3};
vector v {1, 2, 3};
```

lookup(index)

```
v[index]
```

append(value)

```
push_back(value);
emplace_back(value);
```

Push_back() vs emplace_back().

Emplace_back() avoids creating a temporary string.

More info.

https://www.geeksforgeeks.org/vectoremplace_back-c-stl/

assign(index, value)

```
v[index] = 4;
```

insert(index, value)

```
v.insert(v.begin(), value);
```

insert(start_index, stop_index, value)

Python example: my_list[0:0] = other_list //inserts to beginning of list

remove(index)

```
v.erase(v.begin());
v.erase(v.begin() + 5); // erases sixth element
v.erase(v.end());
```

length()

```
v.size();
```

Number of elements currently in array.

capacity()

Maximum amount of elements that can be stored.

empty()

Returns true if empty.

sublist(start_index, stop_index)

Gets a sublist, ie. a slice from the specified start index to the specified stop index. For strings, it gets a substring.

Sublist source and explanation.

https://stackoverflow.com/questions/61147045/is-there-a-c-function-similar-to-pythons-slice-notation

copy()

Perhaps best to do it manually with vectors.

Other copy ideas.

https://www.geeksforgeeks.org/ways-copy-vector-c/

Loops.

Vector loops topic. List loops topic.

```
for (int i = 0; i < v.size(); i++) {
        cout << " " << v[i];       // vector overloads the subscript operator
}
for (i = 0; i < v.size(); i++) {
}
for (vector<int>::iterator it = vec.begin(); it < vec.end(); it++) {
        //vector<int>... is long so auto allows compiler to detect
}
for(auto it=vec.begin(); it<vec.end(); it++) //does < actually work?
{
                                        //doesn't work for list...
        cout<<" "<<*it;// apparently printing an iterator prints element at its position
}
for (auto it = list.begin(); it != list.end(); it++) {
        //note that it != list.end() is used instead of it < list.end()
}
for (auto Number : MyVector) {
        std::cout << Number;
}
for (string s : v) {
}
```

Iterator topic.

The notation for iterators can get long (a good reason to use auto, for example). However, we can't use auto on everything. When you need a shorter type name, try a type alias.

Type alias topic.

```
using Grid = std::vector<std::vector<int>>;
Grid MyGrid {{
  { 1, 2, 3 },
  { 4, 5, 6 },
  { 7, 8, 9 }
}}; //from https://www.studyplan.dev/intro-to-programming/array
```

Using & in loops topic.

```
for (auto& Number : MyVector) {
        std::cout << Number;
}
```

The & operator causes it to be passed by reference, since the default behavior of just passing a copy to work with is rarely desired.

You can also mark it const if you won't modify the ref.

More info:

https://www.studyplan.dev/intro-to-programming/array