Standard Template Library (STL)

The C++ STL

- In 1990, Alex Stepanov and Meng Lee of Hewlett Packard Laboratories extended C++ with a library of class and function templates which has come to be known as the STL.
- In 1994, STL was adopted as part of ANSI/ISO Standard C++.

Components of the STL

- Program's main objective is to manipulate data and generate results
  - Requires ability to **store** data, **access** data, and **manipulate** data
- STL has three basic components:

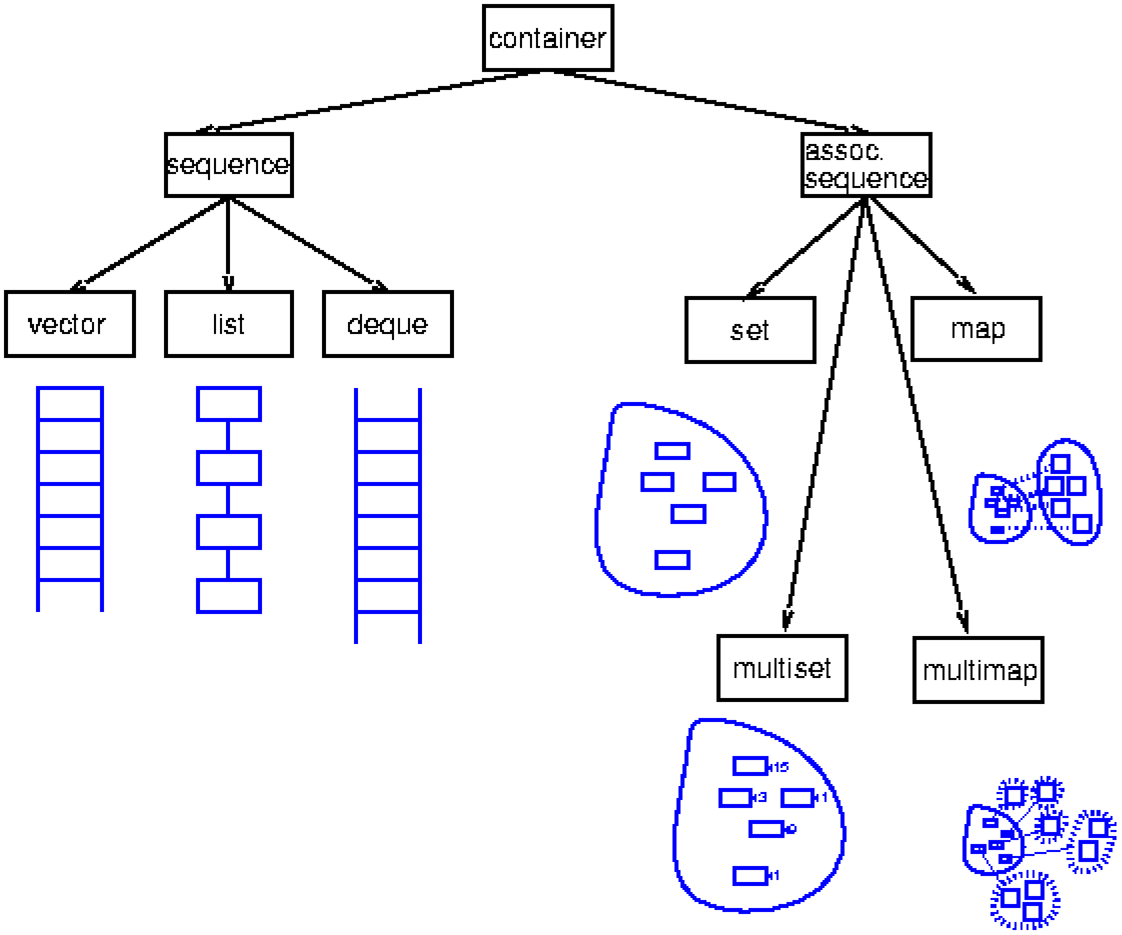
(1) **Containers**: generic class templates for **storing** collection of data (contain other objects).

(2) **Iterators:** generalized 'smart' **pointers** that provides operations for indirect access and facilitate use of containers. They provide an interface that is needed for STL algorithms to operate on STL containers.

(3) **Algorithms**: generic **function templates** for operating on containers.

Why use STL?

- STL offers an assortment of **containers**
- STL publicizes the time and storage **complexity** of its containers
- STL containers grow and shrink in **size** automatically
- STL provides built-in **algorithms** for processing containers
- STL provides **iterators** that make the containers and algorithms flexible and efficient.
- STL is **extendable** which means that users can add new containers and new algorithms**.**
- **Memory management**: no memory leaks or serious memory-access violations. (e.g., pointers)
- Reduce testing and debugging **time**.



container

sequence

assoc. sequence

vector

list

deque

set

map

multiset

multimap

Sequence Containers

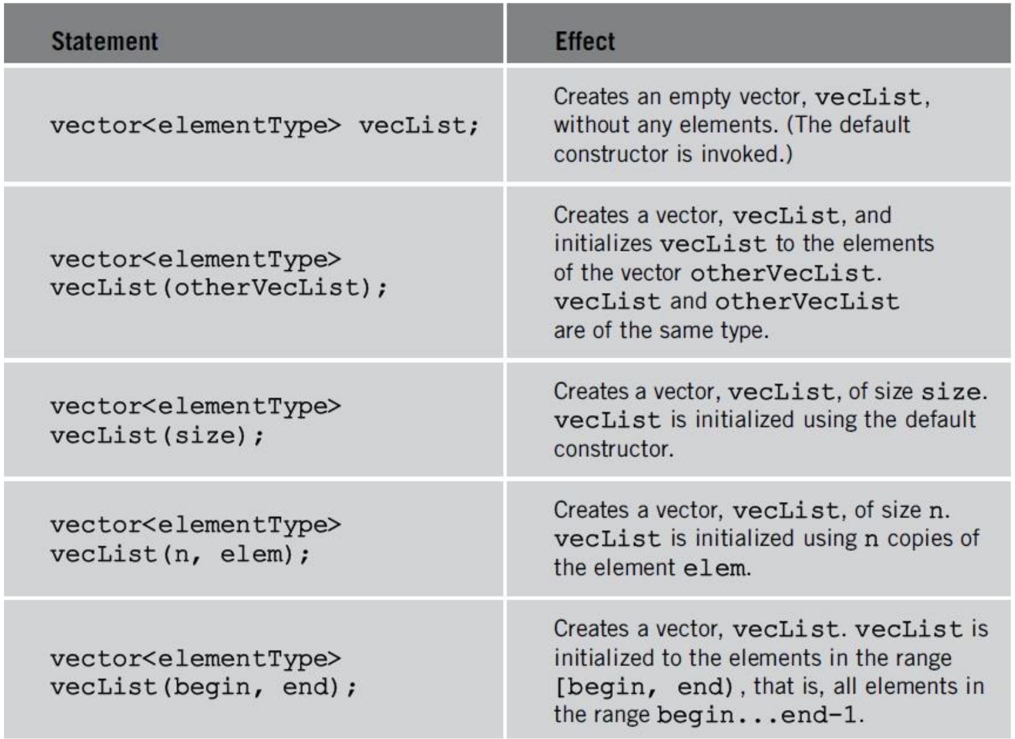
- Every object has a specific position
- Predefined sequence containers
  - `vector`, `deque`, `list`
- Sequence container `vector`
  - Logically: same as **arrays**
- All containers
  - Use same names for common operations
  - Have specific operations

Sequence Container: `vector`

- Vector container
  - Stores, manages objects in a **dynamic array**
  - Elements accessed **randomly**
  - Time-consuming item insertion: beginning and middle
  - Fast item insertion: end
- Class implementing vector container
  - `vector`
- Header file containing the `class vector`
  - `vector`
- Using a vector container in a program requires the following statement:
  - `#include <vector>`

- Declaring vector objects

Various ways to declare and initialize a vector container

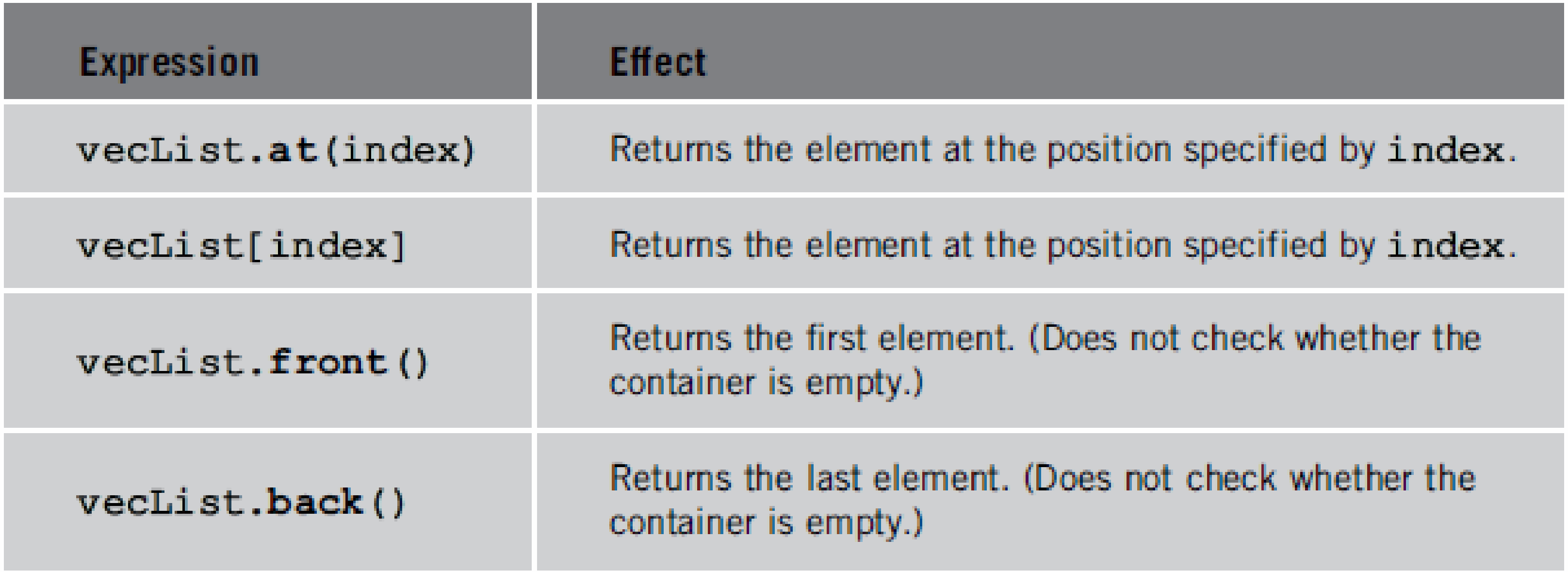


Statement	Effect
`vector<elementType> vecList;`	Creates an empty vector, `vecList`, without any elements. (The default constructor is invoked.)
`vector<elementType> vecList(otherVecList);`	Creates a vector, `vecList`, and initializes `vecList` to the elements of the vector `otherVecList`. `vecList` and `otherVecList` are of the same type.
`vector<elementType> vecList(size);`	Creates a vector, `vecList`, of size `size`. `vecList` is initialized using the default constructor.
`vector<elementType> vecList(n, elem);`	Creates a vector, `vecList`, of size `n`. `vecList` is initialized using `n` copies of the element `elem`.
`vector<elementType> vecList(begin, end);`	Creates a vector, `vecList`. `vecList` is initialized to the elements in the range `[begin, end)`, that is, all elements in the range `begin...end-1`.

– Examples:

  - `vector<int> intlist;`
  - `vector<string> stringList;`

Operations to **access** the elements of a vector container



Expression	Effect
`vecList.at(index)`	Returns the element at the position specified by `index`.
`vecList[index]`	Returns the element at the position specified by `index`.
`vecList.front()`	Returns the first element. (Does not check whether the container is empty.)
`vecList.back()`	Returns the last element. (Does not check whether the container is empty.)

```
myvector contains: 0 1 2 3 4 5 6 7 8 9
```

```cpp
#include <iostream>
#include <vector>

int main()
{
std::vector<int> myvector(10); // 10 zero-initialized ints

  // assign some values:
for (unsigned i = 0; i<myvector.size(); i++)
myvector.at(i) = i;

std::cout << "myvector contains:";
for (unsigned i = 0; i<myvector.size(); i++)
std::cout << ' ' << myvector.at(i);
std::cout << '\n';

return 0;}
```

Declaring an Iterator to a Vector Container

- Process vector container like an array
  - Using array subscripting operator
- Process vector container elements
  - Using an iterator
- `class vector`: function `insert`
  - Insert element at a specific vector container position
  - Uses an iterator
- `class vector`: function `erase`
  - Remove element
    - Uses an iterator

- `class vector` contains `typedef iterator`
  - Declared as a public member
  - Vector container iterator
  - Example

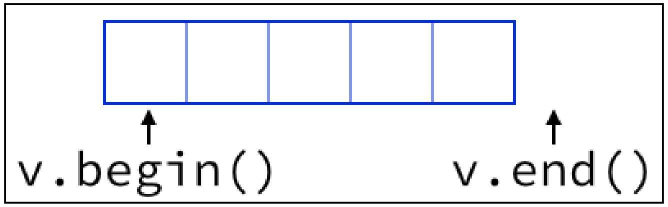
```
vector<int>::iterator intVecIter;
```

- Requirements for using `typedef iterator`
  1. Container name (vector)
  2. Container element type (<int>)
  3. Scope resolution operator (::)

- **`++intVecIter`**
  - Advances iterator `intVecIter` to next element into the container
- **`*intVecIter`**
  - Dereferencing
  - Returns element at current iterator position

Containers and the Functions `begin` and `end`

- A sequence is defined by a pair of iterators defining a half-open range **[begin:end)**
  - Includes first element but excludes last element.
- **`begin`**
  - Returns an iterator to the first element in the container
- **`end`**
  - Returns an iterator to the element past the end. It does not point to any element. Never read from or write to *end.



```
#include <iostream>
#include <vector>
using namespace std;
int main()
{ vector<int> v1;
v1.push_back(2);
v1.push_back(4);
v1.push_back(7);
vector<int> v2(v1);
vector<int> v3(3);
v3.at(0) = 4;
v3.at(1) = 6;
v3.at(2) = 4;
vector<int> v4(4, 2);
vector<int> v5(v2.begin(), v2.end());

for (unsigned i = 0; i < v1.size(); i++)
{cout << ' ' << v1.at(i) << "\t" << v2[i] << "\t" << v3.at(i) << "\t" <<
v4.at(i) << "\t"<< v5.at(i);
cout << '\n';}

return 0;}
```

```
 2	2	4	2	2
 4	4	6	2	4
 7	7	4	2	7
```

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
vector<int> v1;
v1.push_back(3);
v1.push_back(4);
v1.push_back(6);
vector<int>::iterator it;

cout << v1.front() << v1.back() << "\n";

for (it = v1.begin(); it != v1.end(); it++)
cout << *it;

return 0;}
```

```
36
346
```