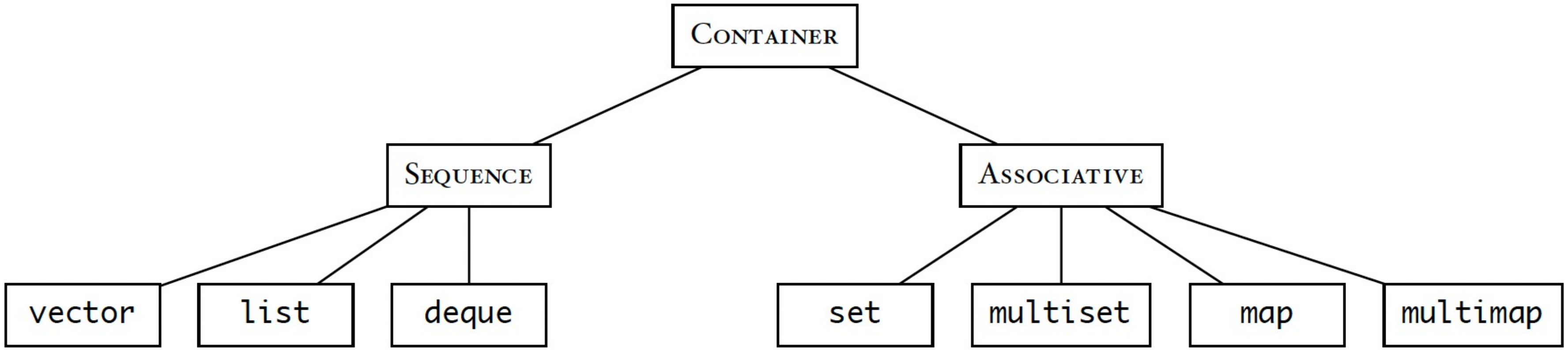
standard template library vector, iterator, list, map

- Standard Template Library
  - Extension to C++
  - Object-oriented
  - Based on Alex Stepanov and Meng Lee of Hewlett-Packard Laboratories（1990）
  - Generic entities: container, iterator, algorithm
    - Container: data structure that hold objects, vector, list, stack, queue …
    - Iterator: A generalization of a pointer, used to reference an element in a container
    - Algorithm: generic functions.

Why STL?

- Offers an assortment of containers.
- Publicizes the time and storage complexity of its containers
- Containers grow and shrink dynamically
- Built-in algorithms for common tasks
- Iterators that are flexible and efficient
- Good memory management (reduce memory leak or serious memory access violations)
- Reduce testing and debugging time

Containers


Container
Sequence
Associative
vector
list
deque
set
multiset
map
multimap

Sequence containers

- Every element (object) has a specific position
- The order of the elements inside is important
- STL common sequence containers: vector, list, deque
- In general, STL containers
  - Have efficient methods for the operations they support.
  - If not efficient, then the method is not provided for that container.
  - Same method name for same operations across different containers

vector container in STL

- Simplest container in STL
  - Probably not a good name since “vector” has a different meaning in math. (The designer of the STL is aware of this not-so-good choice).
- Stores and manages elements in a dynamic array.
- Support O(1) random access
- Other than insertion/deletion at the end (push_back(), pop_back()), time consuming insertion anywhere else.
- Header file <vector>; class: vector

Declare a vector

- Default   -- empty vector
- (vector v) -- start with copies of values in v
- (size_t n) -- start with n element of default value (if type int, then default value is 0)
- (size_t n, T x) – start with n elements with value x
- (Iterator a, Iterator b) – copy the range

Examples:

```
  vector <int>  numbers;
  vector <int>  fivezeros(5);
  vector <int>  fivefives(5,5);
   vector<vector<int> >   twod_vec;
```

Common Methods

- void push_back(const T& el) – insert an element *el* at the end of the vector.
- at(…); pop_back(..); resize(..)
- void clear();
- insert(…); //inefficient but at the end.
- iterator begin() //return an iterator that references the 1st element of the vector
- iterator end() //return an iterator that references the position beyond the last elment of the vector

Accessing element: at() versus [ ]

- v.at(index)
- v[index]
- v.front() first element
- v.back() last element

- Different between at() and []:
  - At() does bound checking and will throw an exception if out of bounds
  - [] will likely crash with segfault if out of bounds. Faster, possibly dangerous.

Example Code

```
…
#include <vector>

int main()
{
  std::vector<int> v1;  //empty vector
  for (int i=0; i<5; i++)
    v1.push_back(i); //v1 = (0 1 2 3 4)

  for(int i=0; i<v1.size(); i++)
     std::cout << " ' << v1.at(i);

  return 0;
}
```

If using the constructor this way:

- vector<int> x(5); //the vector x contains 5 0s.

resize()

- https://cplusplus.com/reference/vector/vector/resize/
- void resize(size_type n, value_type val = value_type())
- The parameter n can be bigger or smaller than current size. If also greater than the current container capacity, then automatic reallocation takes place.
- Capacity can be equal or greater to the vector size.

Example:

```
myvector.resize(5);
myvector.resize(8, 100);
```

For more methods of vector:

- https://cplusplus.com/reference/vector/vector/
  - More member functions
  - Capacity
  - Modifier
  - Iterators

Iterator: Introduction (advanced concepts later)

- Let us start from an array arr:
-    for (int i=0; i<; i++) cout <<arr[i];
- If linked list:
-     node *begin = list.head; *end = nullptr, *p = begin;
-     while(p!= end)
-       { cout << p->val; p = p->next; }

- If we rewrite the arr iteration using pointer:
-    int *begin = arr; *end = arr+N; *p = begin;
-    while(p!=end)
-       { cout << *p; ++p;}

Pattern to iterate over anything:

- Know where to begin and end
- Keep track of current position (p)
- Moving from current to next (++p for pointer, or p=p->next for linked list)

Iterator

- Work like pointers
  - An iterator object (say *it*) must
  - Indicate the position of a specific element in some sequence
  - Support deference operator (*it)
  - Support increment operator (it++, ++it) to point to next position
  - Support == (and !=) to know if two iterators are at the same position.

Declare pointers and auto keyword

std::vector <int>::iterator iterator1;

std::vector <int>::const_iterator iterator2;

- Modern C++ support keyword *auto*

auto iterator3 = somevector.begin();

Compiler knows the type!

Iterator loop

```
for(auto i=container.begin(); i!=container.end(); ++i)
   { cout << *i; }
```

- *Or*

```
auto p = container.begin();
while(p != container.end())
{ cout << *p; ++p; }
```

Work for any container!!!