Contents

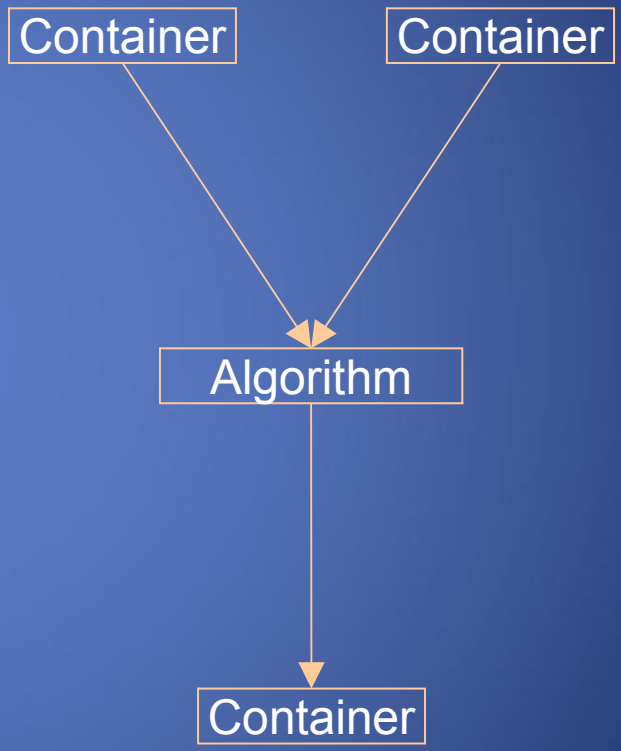
- Introduction To STL
- Containers
- Iterators
- Algorithms
- Function Objects

Introduction To STL

- STL is Standard Template Library
  - Powerful, template-based components
    - Containers: template data structures
    - Iterators: like pointers, access elements of containers
    - Algorithms: data manipulation, searching, sorting, etc.
  - Object- oriented programming: reuse, reuse, reuse
  - Only an introduction to STL, a huge class library

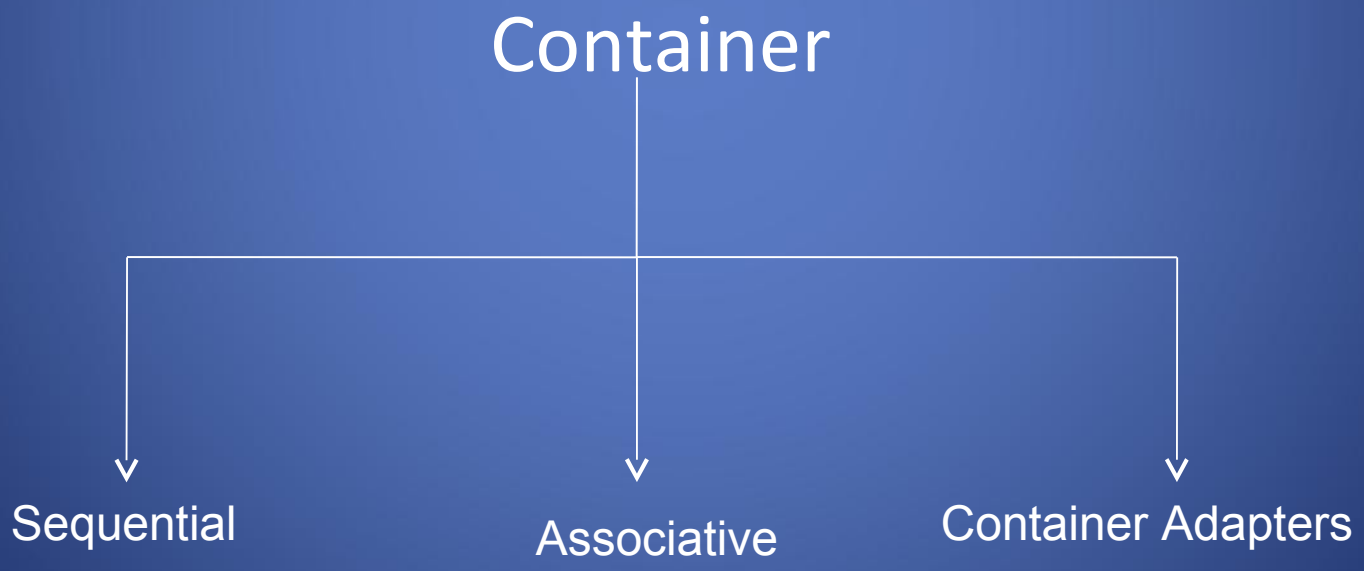
STL components overview

- Data storage, data access and algorithms are separated
  - *Containers* hold data
  - *Iterators* access data
  - *Algorithms*, *function objects* manipulate data
  - *Allocators*... allocate data (mostly, we ignore them)



Container

- A *container* is a way that stored data is organized in memory, for example an array of elements.



Container ctd-

- Sequence containers
  - **`vector`**
  - **`deque`**
  - **`list`**
- Associative containers
  - **`set`**
  - **`multiset`**
  - **`map`**
  - **`multimap`**
- Container adapters
  - **`stack`**
  - **`queue`**

Sequential Container

- `vector<T>` – dynamic array
  - Offers random access, back insertion
  - Should be your ***default choice, but choose wisely***
  - Backward compatible with C : `&v[0]` points to the first element
- `deque<T>` – double-ended queue (usually array of arrays)
  - Offers random access, back and front insertion
  - Slower than vectors, no C compatibility
- `list<T>` – 'traditional' doubly linked list
  - Don't expect random access, you can insert anywhere though

Some functions of vector class

-size()

    -provides the number of elements

-push_back()

    -appends an element to the end

-pop_back()

    -Erases the last element

-begin()

    -Provides reference to last element

-end()

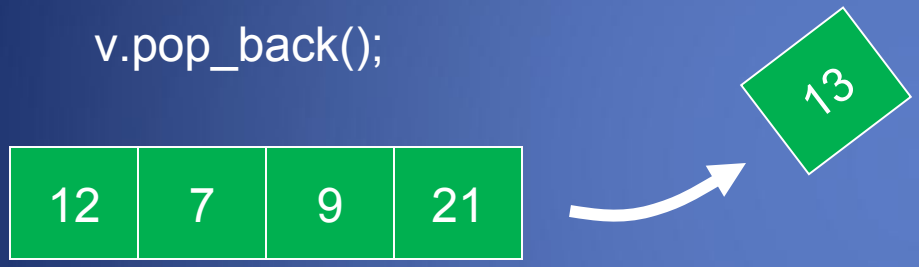
    -Provides reference to end of vector

Vector container

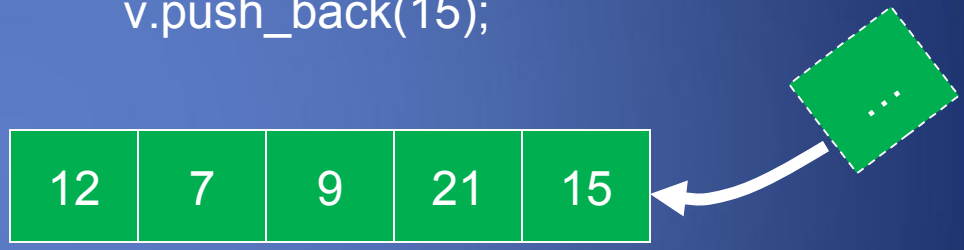
int array[5] = {12, 7, 9, 21, 13 };

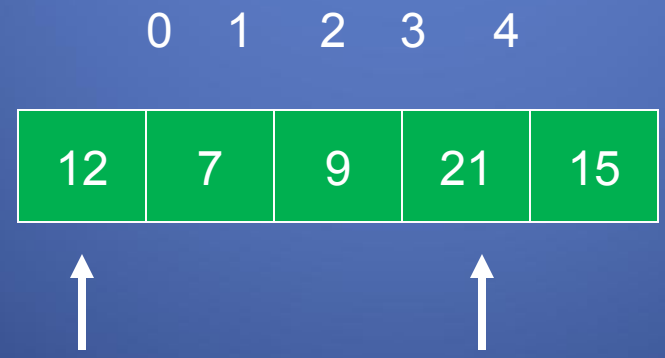
Vector<int> v(array,array+5);

v.pop_back();



v.push_back(15);





v.begin();

v[3]

Some function of list class

- **list** functions for object **t**
  - **t.sort()**
    - Sorts in ascending order
  - **t.splice(iterator, otherObject);**
    - Inserts values from **otherObject** before **iterator**
  - **t.merge(otherObject)**
    - Removes **otherObject** and inserts it into **t**, sorted
  - **t.unique()**
    - Removes duplicate elements

Functions of list class cntd-

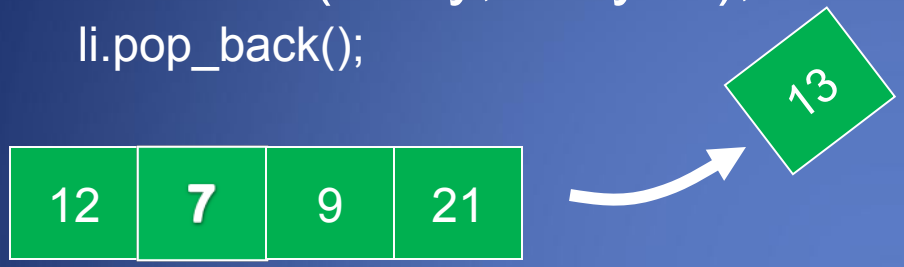
- **list** functions
  - **t.swap(otherObject);**
    - Exchange contents
  - **t.assign(iterator1, iterator2)**
    - Replaces contents with elements in range of iterators
  - **t.remove(value)**
    - Erases all instances of **value**

List container

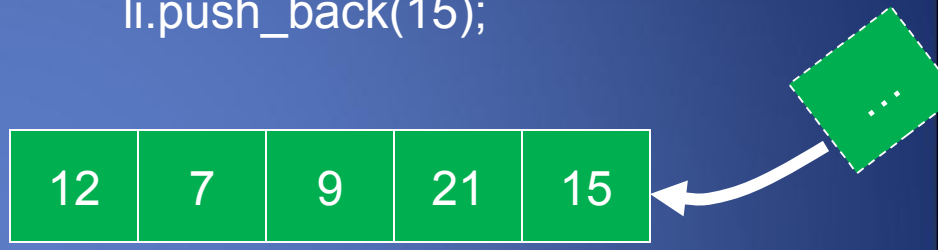
int array[5] = {12, 7, 9, 21, 13 };

list<int> li(array,array+5);

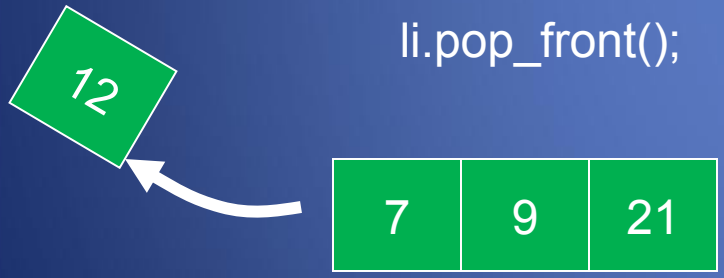
li.pop_back();



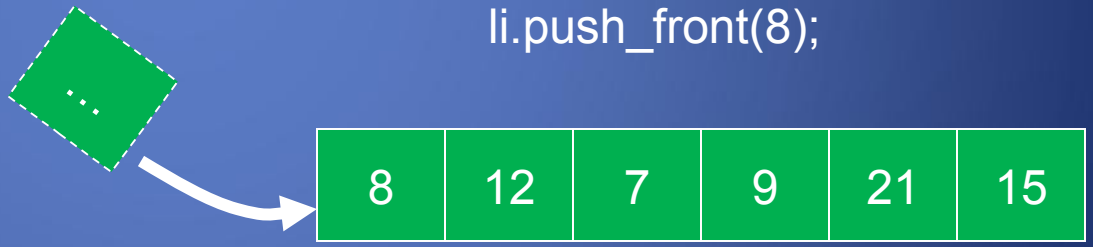
li.push_back(15);



li.pop_front();



li.push_front(8);



li.insert()

Functions of dequeue class

- dequeue functions for object d

  -**d.front()**

  -Return a reference (or const_reference) to the first component of d

  -**d.back()**

  -Return a reference (or const_reference) to the last component of d.

  -**d.size()**

  -Return a value of type size_type giving the number of values currently in d.

Functions of dequeue class contd-

-**d.push_back(val)**

-Add val to the end of d, increasing the size of d by one.

-**d.push_front(val)**

-Add val to the front of d, increasing the size of d by one.

-**d.pop_back()**

-Delete the last value of d. The size of d is reduced by one.

-**d.pop_front()**

-Delete the first value of d. The size of d is reduced by one.

Associative Containers

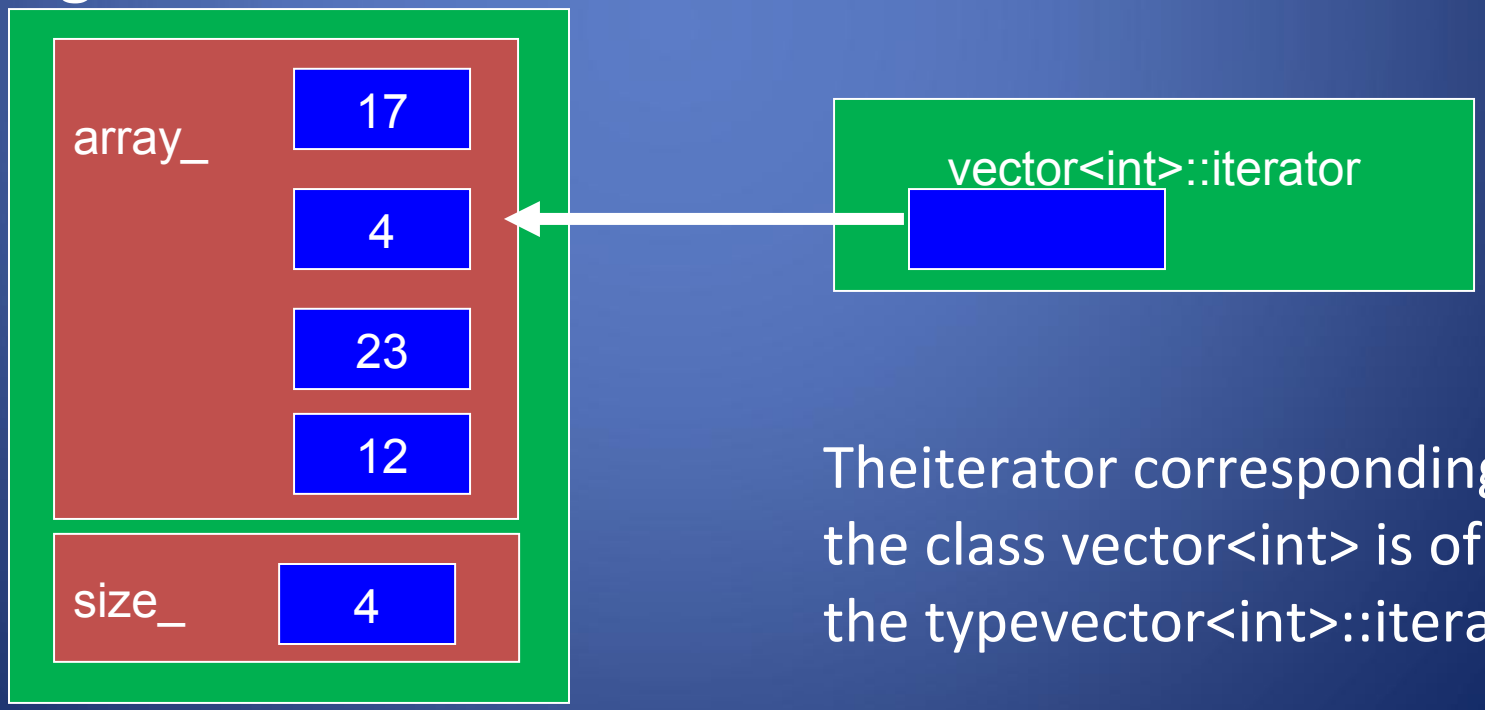
- Offer *O(log n)* insertion, suppression and access
- Store only weakly strict ordered types (eg. numeric types)
  - Must have `operator<()` and `operator==()` defined and !(a<b) && !(b<a) ≡ (a==b)
- The sorting criterion is also a template parameter
- `set<T>` – the item stored act as key, no duplicates
- `multiset<T>` – set allowing duplicate items
- `map<K,V>` – separate key and value, no duplicates
- `multimap<K,V>` – map allowing duplicate keys
- hashed associative containers *may* be available

Container adaptors

- Container adapters
  - **stack**, **queue** and **priority_queue**
  - Not first class containers
    - Do not support iterators
    - Do not provide actual data structure
  - Programmer can select implementation
  - Member functions **push** and **pop**

Iterators

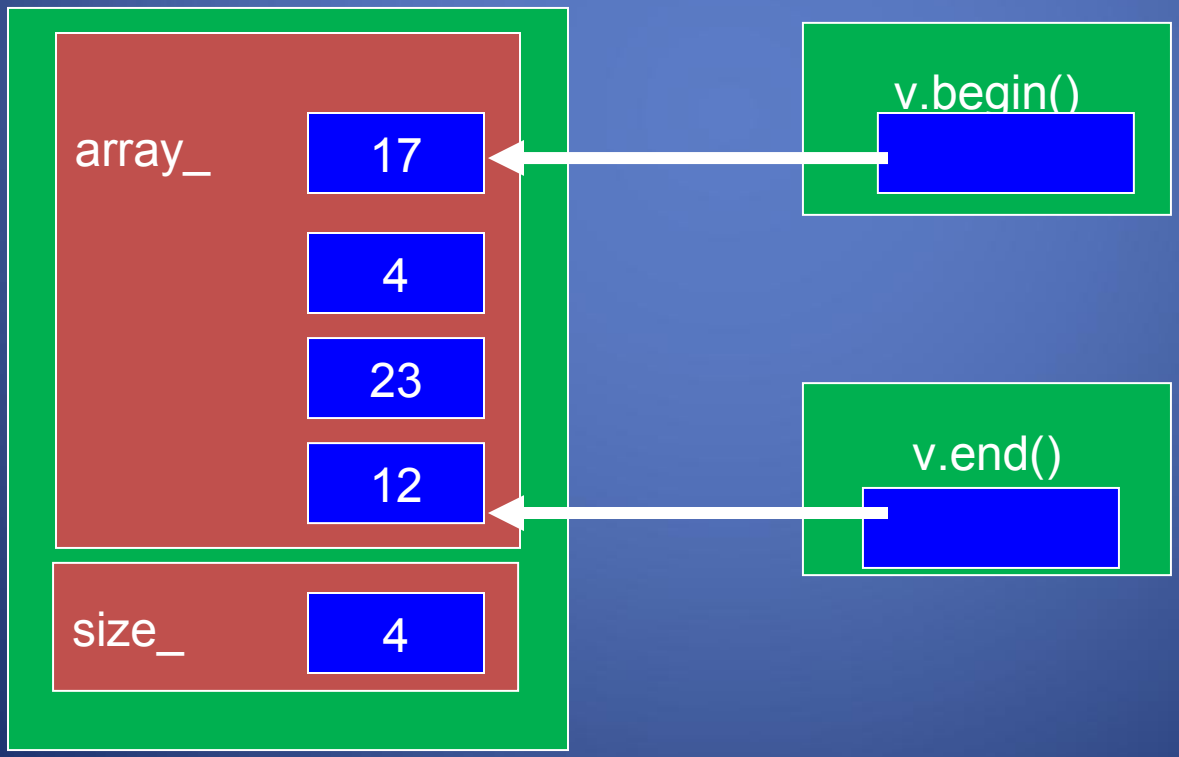
- Iterators are pointer-like entities that are used to access individual elements in a container.
- Often they are used to move sequentially from element to element, a process called *iterating* through a container.



Theiterator corresponding to the class vector<int> is of the typevector<int>::iterator

Iterators contd-

- The member functions begin() and end() return an iterator to the first and past the last element of a container



Iterators Categories

- Not every iterator can be used with every container for example the list class provides no random access iterator
- Every algorithm requires an iterator with a certain level of capability for example to use the [] operator you need a random access iterator
- Iterators are divided into five categories in which a higher (more specific) category always subsumes a lower (more general) category, e.g. An algorithm that

  accepts a forward iterator will also work with a bidirectional iterator and a random access iterator