The Standard C++ Library – Iterators and Algorithms

Version 1: Dr. Ofir Pele
Version 2: Dr. Erel Segal-Halevi

Iterators

Input Iterator

++, input

Output Iterator

++, output

Forward Iterator

++, I/O

Bi-directional Iterator

++, --, I/O

Random-Access Iterator

Pointer arithmetic

Iterator Types

	Output	Input	Forward	Bi-directional	Random
Read		x = *i	x = *i	x = *i	x = *i
Write	*i = x		*i = x	*i = x	*i = x
Iteration	++	++	++	++, --	++, --, +, -, +=, -=
Comparison		==, !=	==, !=	==, !=	==, !=, <, >, <=, >=

- Output: write only and can write only once
- Input: read many times each item
- Forward supports both read and write
- Bi-directional support also decrement
- Random supports random access
(just like C pointer)

Iterators & Containers

Input/output/forward iterators:

- iostreams (folder 3)

Bidirectional iterators:

- list, map, set

Random access iterators:

- vector

Iterators & Containers

```
class NameOfContainer {
  …
  typedef … iterator;    // iterator type
  iterator begin();      // first element
  iterator end();        // element after last
```

```
NameOfContainer<...> c

…

NameOfContainer<...>::iterator it;

for( it= c.begin(); it!=c.end(); ++it)

      // do something that changes *it
```

Iterators & Containers: c++11

```
class NameOfContainer {

 …
 typedef … iterator; // iterator type
 iterator begin();      // first element
 iterator end();        // element after last
```

```
NameOfContainer<...> c

…


for(auto it= c.begin(); it!=c.end(); ++it)

      // do something that changes *it
```

Iterators & Containers: c++11

```cpp
class NameOfContainer {
  …
  typedef … iterator; // iterator type
  iterator begin();     // first element
  iterator end();        // element after last
```

```cpp
NameOfContainer<...> c

…


for(auto& val : c)

    // do something that changes val
```

const_iterators & Containers

```
class NameOfContainer {
  …
typedef … const_iterator; // iterator type
const_iterator begin() const;      // first element
const_iterator end() const;        // element after last
```

```
NameOfContainer<...> c

…

NameOfContainer<...>::const_iterator it;

for( it= c.begin(); it!=c.end(); ++it)

      // do something that does not change *it
```

const_iterators & Containers: c++11

```
class NameOfContainer {
  …
typedef … const_iterator; // iterator type
const_iterator cbegin() const;      // first element
const_iterator cend() const;        // element after last
```

```
NameOfContainer<...> c

…


for(auto it= c.cbegin(); it!=c.cend(); ++it)

    // do something that does not change *it
```

const_iterators & Containers: c++11

```
class NameOfContainer {
  …
typedef … const_iterator; // iterator type
const_iterator cbegin() const;      // first element
const_iterator cend() const;        // element after last
```

```
NameOfContainer<...> c

…


for(const auto& val : c)

    // do something that does not change val
```

const_iterators & Containers

```
...
const_iterator cbegin() const;
const_iterator cend() const;
const_iterator begin() const;
const_iterator end() const;
```

```
...

iterator begin();
iterator end();
```
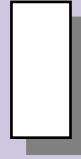
Note that the begin() and end() methods that return regular iterator are not **const** methods. i.e: if we get a container by const (const ref, ...) we can't use these methods. We have to use the methods that return **const_iterator** [end of first week]

IntBufferSwap example revisited

- See folder 4.
- Focus on iterator and const_iterator.
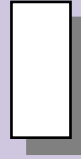
Iterators & Sequence Containers

```
SeqContainerName<...> c;
SeqContainerName<...>::iterator i,j;
```

- `c.insert(i,x)` – inserts **x** before `i`
- `c.insert(i,first,last)`
  – inserts elements in `[first,last)` before `i`
- `c.erase(i)` – erases the element that `i` points to
- `c.erase(i,j)`
  – erase elements in range `[i,j)`

Iterators & Sequence Containers c++11

```
SeqContainerName<...> c;
SeqContainerName<...>::iterator i,j;
```

- `c.emplace(i,p1,...,pn)`:
Constructs and inserts before `i` an object
with a constructor that gets `p1,...,pn`
parameters

Iterators & other Containers

- insert and erase has the same ideas, except they keep the invariants of the specific container.
- For example, a Sorted Associative Container will remain sorted after insertions and erases.

Iterators & other Containers

- So what does `c.`**`insert(`****`pos`****`,`****`x`****`)`** does, when c is a Unique Sorted Associative Container ?

- Inserts x into the set, using **pos** as a **hint** to where it will be inserted.

Iterators & other Containers: **c++11**

- So what does c.**emplace_hint(pos,x)** does, when c is a Unique Sorted Associative Container?

- Constructs and Inserts x into the set, using **pos** as a **hint** to where it will be inserted.

Iterator validity

- When working with iterators, we have to remember that their validity can change What is wrong with this code?

```
Container<...> c;
...
for(auto i= c.begin(); i!=c.end(); ++i )
    if( f( *i ) ) { // some test
        c.erase(i);
    }
```
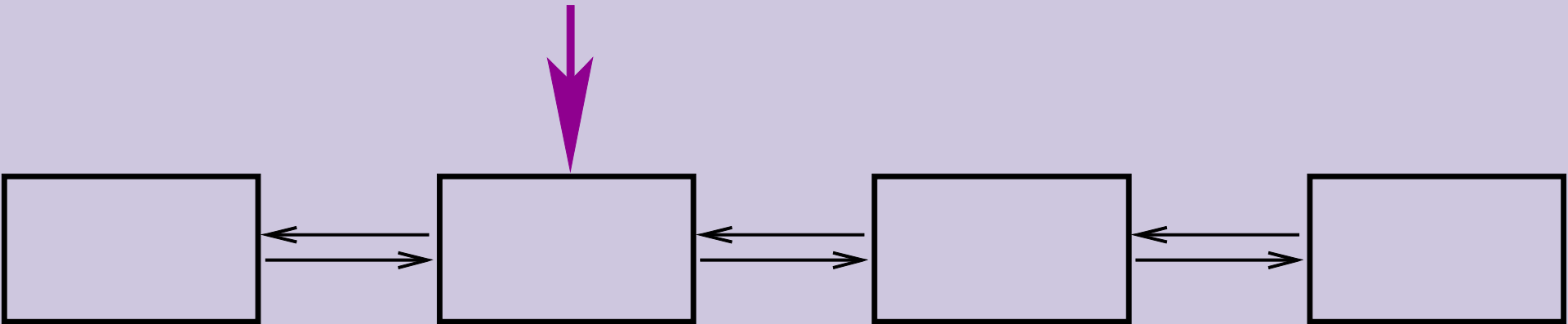
Iterator validity
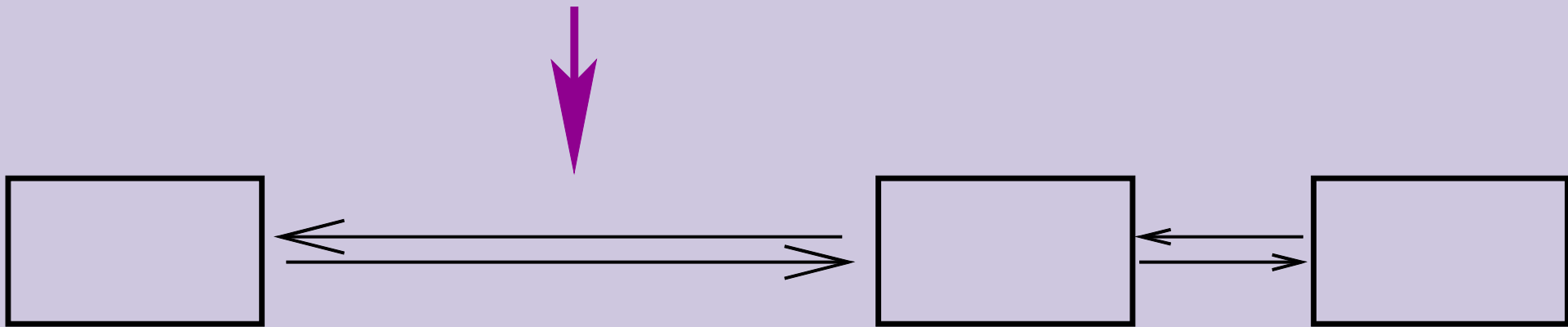
Two cases:

- list, set, map

  - `i` is not a legal iterator

Iterator validity

Two cases:

- **list**, set, map

  - `i` is not a legal iterator
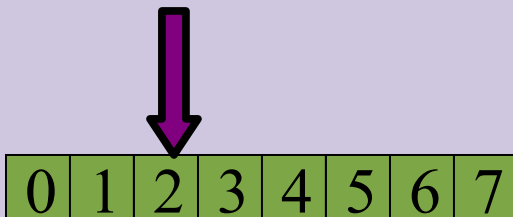
Iterator validity

Two cases:

- **list**, set, map

  - `i` is not a legal iterator

Iterator validity

Two cases:

- list, set, map
    - `i` is not a legal iterator
- **vector**
    - `i` points to the element after

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

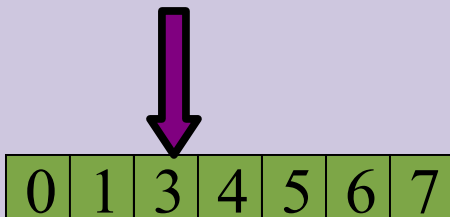# Iterator validity

Two cases:

- list, set, map
  - `i` is not a legal iterator

- **vector**
  - `i` points to the element after

Iterator validity

Two cases:

- list, set, map

  - `i` is not a legal iterator

- **vector**

  - `i` points to the element after

**In either case,
this is not what we want…**

Erasing during iteration (folder 5)

```
Container<...> c;
...
for(auto i= c.begin(); i!=c.end();/*no ++i*/ )
  if( f( *i ) ) { // some test
    i = c.erase(i);
  } else {
    ++i;
  }
```

Iterators & Map

Suppose we work with:

```
map<string,int> dictionary;
map<string,int>::iterator it;
…
it = dictionary.begin();
```

What is the type of `*it` ?

Iterators & Map

Every STL container type Container defines

`Container::value_type`

Type of elements stored in container

- This is the type returned by an iterator

`Container::value_type operator*();`

Iterators & Map

- Ok, so what type of elements does a map return?

- `map<KeyType,ValueType>` keeps pairs
  - `KeyType key` – "key" of entry
  - `ValueType value` – "value" of entry

Pairs

```cpp
template< typename T1, typename T2>
struct pair {
  typedef T1 first_type;
  typedef T2 second_type;

  T1 first;
  T2 second;

  pair( const T1& x, const T2& y )
     : first(x), second(y)
  {}
};
```

Map value_type

```cpp
template< typename Key, typename T,
          typename Cmp = less<Key> >
class map {
public:
  typedef pair<const Key, T> value_type;

  typedef Key key_type;
  typedef T mapped_type;
  typedef Cmp key_compare;
};
```

Using map iterator

```cpp
map<string,int> dict;
…

for( auto i = dict.cbegin();
     i != dict.cend();
     ++i )
{
  cout << i->first << " "
       << i->second << "\n";
}
```

Using map iterator

```cpp
map<string,int> dict;
…

for( const auto& val : dict) {
    cout << val.first << " "
            << val.second << "\n";
}
```

Iterators and Assoc. Containers (folder 6)

Additional set of operations:

- `iterator C::find(key_type const& key)`

Return iterator to first element with `key`.

Return `end()` if not found

- `iterator C::lower_bound(key_type const& key)`

Return iterator to first element greater or equal to `key`

- `iterator C::upper_bound(key_type const& key)`

Return iterator to first element greater than `key`

Adaptors

- Good functionality, wrong interface

- For example, adaptors of basic containers

with limited interface:

stack<T,SequentialContainer>

**queue**<T,SequentialContainer>

stack<T,SequentialContainer**>**

- provides `emplace`, `push`, `pop`, `top`, `size`, `empty,...`

- Notice that unlike java, `pop` , is not returning a value. i.e: it's a void function.

- The reason (historic with c++-11?):

  to make pop return a value it would be either inefficient or wrong:
  http://www.sgi.com/tech/stl/stack.html#3

Algorithms

- Most STL algorithms works on sequences

- Sequences are passed as two iterators:

  - beginning element

  - element one after last

p   q

sequence [p,q)
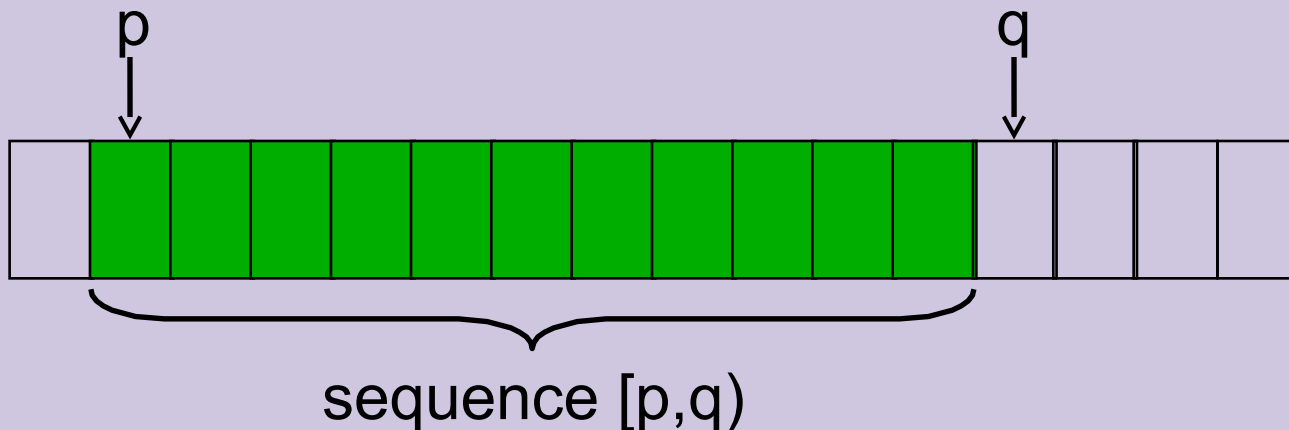
- Algorithms depend on iterator type
  not on container type

Example – merge documentation

copy

```cpp
template< typename In, typename Out>
Out copy(In first, In last, Out res)
{
  while (first != last)
    *res++ = *first++;
  return res;
}
```

copy

```
template< typename In, typename Out>
Out copy(In first, In last, Out res)
{
    while (first != last)
        *res++ = *first++;
    return res;
}
```

What's wrong with this ?

```
void foo(const vector<char>& v) {
    vector<char> v2;
    ...
    copy(v.begin(),v.end(), v2.begin());
```

copy

```
template< typename In, typename Out>
Out copy(In first, In last, Out res)
{
  while (first != last)
    *res++ = *first++;
  return res;
}
```

What's wrong with this ?

```
void foo(const vector<char>& v) {
  vector<char> v2;
  ...
  copy(v.begin(),v.end(), v2.begin());
```

OUT OF BOUND

So how can we copy and insert ?

Solution #1: Use insert explicitly

```
void foo(const vector<char>& v) {
  vector<char> v2;

  ...
  v2.insert(v2.end(),v.begin(),v.end());
```

So how can we copy and insert ?

Solution #2: Use back_inserter, which returns an iterator that knows to "push_back". See folder 6.

```cpp
void foo(const vector<char>& v) {
  vector<char> v2;
  ...
  copy(v.begin(),v.end(), back_inserter(v2));
```

sort – using operator <

```
template <class RandomAccessIterator>
void sort(RandomAccessIterator first,
          RandomAccessIterator last);
```

Example usage(the hard way):

```
sort< vector<int>::iterator >
                (vec.begin(), vec.end());
```

sort – using operator <

```
template <class RandomAccessIterator>
void sort(RandomAccessIterator first,
          RandomAccessIterator last);
```

Example usage:

```
sort(vec.begin(), vec.end());
```

sort – using operator <

```
template <class RandomAccessIterator>
void sort(RandomAccessIterator first,
          RandomAccessIterator last);
```

Example usage with primitive arrays:

```
int arr[5];
...
sort(arr, arr+5);
```

sort – using operator <

```
template <class RandomAccessIterator>
void sort(RandomAccessIterator first,
          RandomAccessIterator last);
```

Example usage with primitive arrays (C++11):

```
int arr[5];
...
sort(begin(arr), end(arr));
```

sort – using comparator

```
template <class RandomAccessIterator,
          class StrictWeakOrdering>
void sort(RandomAccessIterator first,
          RandomAccessIterator last,
          StrictWeakOrdering comp);
```

Example usage:

```
sort(vec.begin(), vec.end(), greater<int>());
```
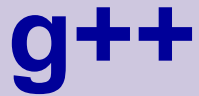
sort – compile error

```
list<int> l(nums, nums+SIZE);
sort(l.begin(), l.end());
```

sort – compile error

```
list<int> l(nums, nums+SIZE);
sort(l.begin(), l.end());
```

list iterators are bidirectional and not random a

g++

/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../include/c++/4.1.2/bits/stl_algo.h: In function 'void std::sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = std::_List_iterator<int>]':

Main.cpp:17:   instantiated from here

/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../include/c++/4.1.2/bits/stl_algo.h:2713: error: no match for 'operator-' in '__last - __first'

/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../include/c++/4.1.2/bits/stl_bvector.h:182: note: candidates are: ptrdiff_t std::operator-(const std::_Bit_iterator_base&, const std::_Bit_iterator_base&)

/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../include/c++/4.1.2/bits/stl_algo.h: In function 'void std::__final_insertion_sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = std::_List_iterator<int>]':

/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../include/c++/4.1.2/bits/stl_algo.h:2714:   instantiated from 'void std::sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = std::_List_iterator<int>]'

Main.cpp:17:   instantiated from here

/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../include/c++/4.1.2/bits/stl_algo.h:2357: error: no match for 'operator-' in '__last - __first'

. . .

g++

/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../include/c++/4.1.2/bits/stl_algo.h: In function 'void std::sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = std::_List_iterator<int>]':

Main.cpp:17:   instantiated from here

/usr/lib/gcc/i486-linux-g[...]13: error: no match for 'operator-' in '__last[...]

/usr/lib/gcc/i486-linux-g[...]82: note: candidates are: ptrdiff_t std::operato[...]ator_base&)

/usr/lib/gcc/i486-linux-g[...]unction 'void std::__final_insertio[...]ator) [with _RandomAccessIter[...]

???

/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../include/c++/4.1.2/bits/stl_algo.h:2714:   instantiated from 'void std::sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = std::_List_iterator<int>]'

Main.cpp:17:   instantiated from here

/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../include/c++/4.1.2/bits/stl_algo.h:2357: error: no match for 'operator-' in '__last - __first'

    . . .

g++

/usr/lib/gcc/i486-linux-
gnu/4.1.2/../../../../include/c+
+/4.1.2/bits/stl_algo.h: In function 'void
std::sort(_RandomAccessIterator,
_RandomAccessIterator) [with
**_RandomAccessIterator =
std::_List_iterator<int>**]':

Main.cpp:17:   instantiated from here

/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../include/c++/4.1.2/bits/stl_algo.h:2713: error: no match for
    'operator-' in '__last - __first'

...

g++ -D_GLIBCXX_CONCEPT_CHECKS and STLFilt

BD Software STL Message Decryptor v2.47a for gcc

stl_algo.h: In function 'void sort(_List_iterator<int>, _List_iterator<
          int>)':

Main.cpp:17:   instantiated from here

stl_algo.h:2713: error: no match for 'operator-' in '__last - __first'

stl_algo.h: In function 'void __final_insertion_sort(
      _List_iterator<int>, _List_iterator<int>)':

stl_algo.h:2714:   instantiated from 'void sort(
      _List_iterator<int>, _List_iterator<int>)'

Main.cpp:17:   instantiated from here

. . .

g++ -D_GLIBCXX_CONCEPT_CHECKS and STLFilt

...
Main.cpp:17:   instantiated from here
boost_concept_check.h:223: error: **conversion from '**

bidirectional_iterator_tag' to non-scalar type '

random_access_iterator_tag' requested

Cryptic error messages

STLFilt:

An STL Error Message Decryptor for C++:

http://www.bdsoft.com/tools/stlfilt.html

Cryptic error messages

Different compilers:

   clang++ (free)

   intel c++ (not free)

Strings

What is a string ?

- a typedef for basic_string<char>
- The basic_string class represents a Sequence of characters.
- It contains:
    - all the usual operations of a Sequence.
    - standard string operations such as search and concatenation.

Rope: non standard!

- Scalable string implementation.

- Efficient operation that involve the string as a whole.

- A rope is a reasonable representation for very long strings such as edit buffers or mail messages.

Rope: non standard!

- Scalable string implementation.

- Efficient operation that involve the string as a whole.

- A rope is a reasonable representation for very long strings such as edit buffers or mail messages.

How to convert something to a string?

- c++11: to_string for primitives

- Using std::ostringstream
- We can encapsulate the string conversion and format into stringify functions – **stringify example**

More?

- Lots of other features, especially in c++11 (threads,…)

- Other libraries:
  - Boost
  - opencv, dlib, armadillo, zlib, …