STL

CSE2013-17F

slides are from CSE2013-16S at SKKU

C++ STL

- Standard template library
  - Set of C++ template classes
  - Provide general-purpose classes and functions
- Containers, algorithms, iterators

C++ STL

- Containers manage collections of objects of a certain kind
  - List, vector, deque, map etc.
    - List: doubly-linked list
    - Map: (key, value)
- Algorithms act on containers
  - Sorting, initialization, and transforming
- Interators are used to step through the elements of collections of objects
  - e.g. containers

General Class Methods

- begin, end, ,rbegin, rend

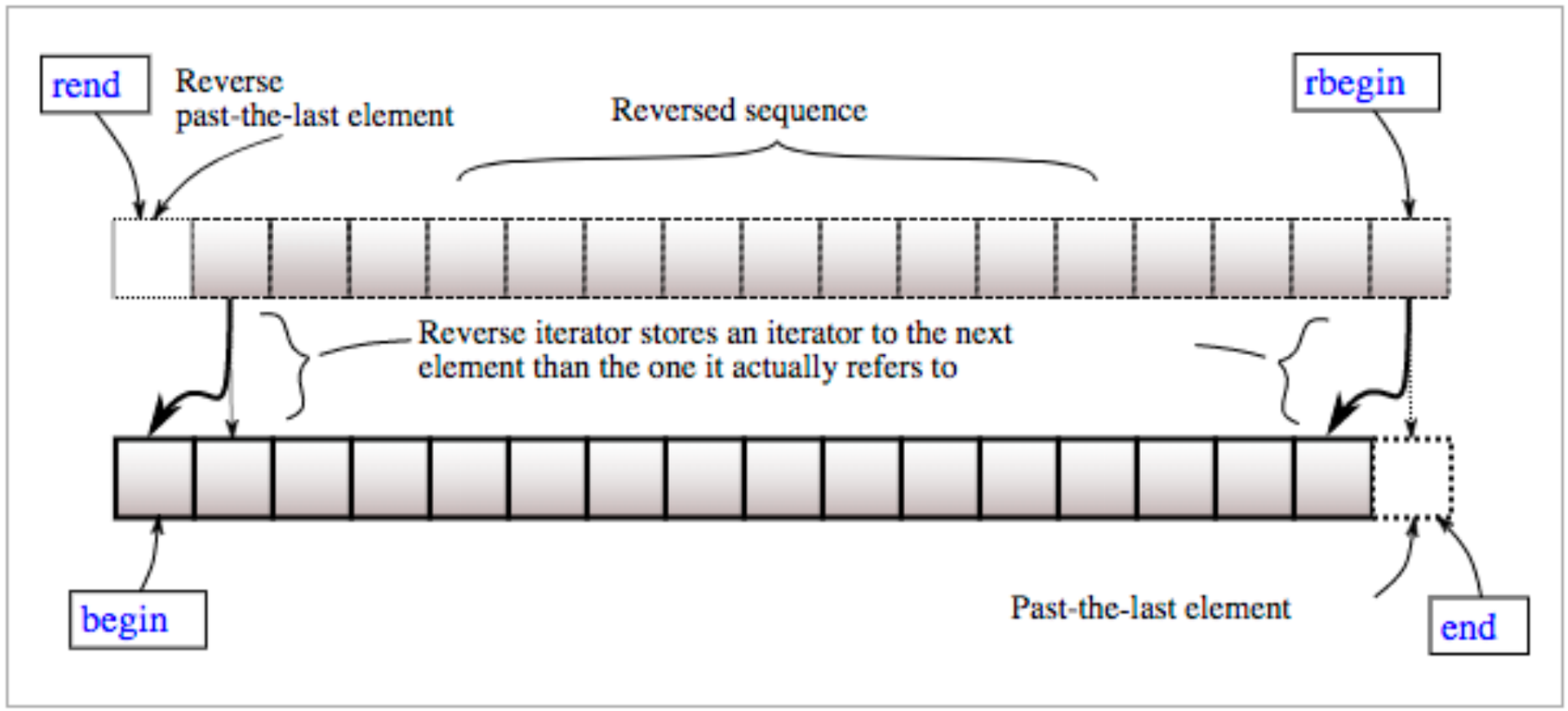


**image from https:/en.cppreference.com/*

General Class Methods

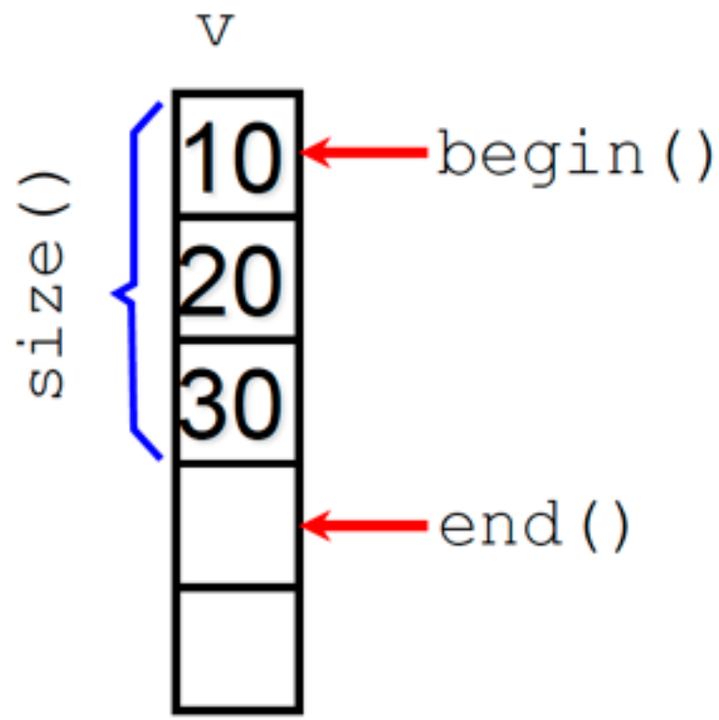
- empty - return “True” if the collection is empty
- size - return # of elements in the collection
- clear - erases all elements in the collection
- erase - erase an element or range of the collections
- insert - insert an element

vector, deque, list

- front - get a reference to the first element
- back - get a reference to the last element
- push_front - add an element to the first (Not vector)
- push_back - add an element to the last
- pop_front - remove an element from the first (Not vector)
- pop_back - remove an element from the last

Iterators

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

using namespace std;

int main() {
  vector<int> v;
  int input;

  while (cin >> input) {
    if (input == 0) break;
    v.push_back (input);
  }

  sort (v.begin(), v.end());

  for (int i=0; i<v.size(); i++)
    cout << v[i] << endl;

  vector<int>::iterator ii;
  for (ii=v.begin(); ii!=v.end(); ii++)
    cout << *ii << endl;

  return 0;
}
```



Operator [ ]

- Vector, map can use [ ] to access an element
  - Similar semantics in C array element accesses
  - Not for list
- at - similar to operator [ ]
  - v.at(index) vs. v[index]
  - Available for vector
  - Perform bound checking unlike operator [ ]

Vector

- #include<vector>
- Constructors
  - vector<Type> vec; // empty vector
  - vector<Type> vec(10); // vector with 10 elements
  - vector<Type> vec(10, 0);
    // vector with 10 elements and they are initialized with 0
- push_back(e), pop_back(), insert(pos, e), erase(pos)

Vector

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

using namespace std;

int main() {
  vector<int> v1;
  vector<int> v2(10);
  vector<int> v3(10,1);

  cout << “v1 :” << v1.capacity();
  cout << “v2 :” << v2.capacity();

  for (int i=0; i<v3.size(); i++)
    cout << v3.at(i) << endl;

  return 0;
}
```

// vector declarations

// capacity returns the size of the vector

List

- #include <list>
- Doubly linked list
- Constructor
  - list<Type> l1; // empty list (size() == 0)
  - list<Type> l2(10); // list with size = 10
  - list<Type> l3(10, 1); // initialized with 1
- push_front(e), push_back(e), insert(pos, e), pop_front(), pop_back(), erase(pos)

List

- Useful algorithms for list
  - sort() sorts the list
  - reverse() reverses the order of the list
  - unique() removes duplicated elements of the list

List

```
#include <iostream>
#include <list>

using namespace std;

int main() {
  list<int> l;

  l.push_back(0);
  l.push_back(1);
  l.insert(l.begin(), 2);
  l.push_back(3);
  l.push_back(4);

  list<int>::iterator i;
  for (i=l.begin(); i!=l.end; i++)
    cout << *i << endl;

  return 0;
}
```

./program
2
0
1
3
4

Set

```
#include <iostream>
#include <set>

using namespace std;

int main() {
  set<int> s;
  s.insert(0);
  s.insert(-2);
  s.insert(5);

  set<int>::const_iterator i;
  i = s.begin();
  while (i != s.end()) {
    cout << *i << endl;
    i++;
  }

  return 0;
}
```

./program
-2
0
5

Map

- <Key, Value> pairs
- Unique element (Key)
- Sorted on key values

Map

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main() {
  map<string, double> market;

  market[“apple”] = 2.30;
  market[“orange”] = 1.20;
  market[“melon”] = 3.30;

  map<string, double>::const_iterator i;
  i = market.begin();
  while (i != market.end()) {
    cout << i->first << “: “ << i->second << endl;
    i++;
  }

  return 0;
}
```

apple: 2.3
melon: 3.3
oragne: 1.2