Associative Containers

Associative Containers (1/2)

- Variable sized container
- Elements are ordered
- Does not provide insertions at certain position because of sorting of elements
- Each element has a value and a key
- Elements looked-up by their keys
- Not all containers provide assignable values

2

Associative Containers (2/2)

- Does not provide mutable iterators
- Use binary search or hashing
- Sorted associative containers are:
  - set
  - multiset
  - map
  - multimap

3

Associative Container Interface

- Finding an element:
  - find(k)
- Counting the number of elements:
  - count(k)
- Returning elements with a certain key:
  - equal_range(k)
- Erasing elements:
  - erase(k), erase(p), erase(p, q)

4

Associative Container Models

- set
- multiset
- map
- multimap

5

Associative Container Types

- Unique associative container
- Multiple associative container
- Simple associative container
- Pair associative container
- Sorted associative container
- (Hashed associative container)

6

Unique Associative Container

- Refinement of Associative Container
- Each key in the container is unique
- The following are unique containers:
  - set
  - map
  - (hash_set)
  - (hash_map)

7

Simple Associative Container

- Refinement of Associative Container
- The value of the elements is also the key value
- The following are simple containers:
  - set
  - multiset

8

Pair Associative Container (1/2)

- Refinement of Associative Container
- Associates a key with some other object
- Value typeid of the form:
  pair<const key_type, mapped_type>
- Cannot provide mutable iterators
- The change element use:
  iterator->second=new_val;

9

Sorted Associative Container

- The following are sorted containers:
  - set
  - map
  - multiset
  - multimap

10

Sorted Associative Container Operations

- Return a key comparison object:
  - key_comp()
- Return the value comparison object:
  - value_comp()
- Using the boundaries:
  - lower_bound(k), upper_bound(k)
- Insertion with hint:
  - insert(p, t)

11

Sets (1/2)

- A set is a model of:
  - Simple Associative Container
  - Unique Associative Container
  - Sorted Associative Container
- Defined in <set> header
- set<Key,Compare, Allocator>
- Duplicate keys not supported
- The data items are the keys

12

Sets (2/2)

- Ordering can be determined by user supplied comparitor function object
- Returns constant BI-directional iterators
  - Cannot modify elements through iterators
- Well suited for the set algorithms
  - Set algorithms accept sorted ranges and a set is always sorted
  - Set algorithms return sorted ranges and inserting them in sets is a fast operation

13

Set Main Member Functions (1/2)

- Constructor:
  - default, copy and with a given size
- Accessors:
  - begin(), end(), rbegin(), rend()
- Sizes:
  - size(), max_size(), empty(), count()

14

Set Main Member Functions (2/2)

- Insertion:
  - insert()
- Deletion:
  - erase()
- Lookup:
  - find(), lower_bound(), upper_bound(), equal_range()
- Comparison: operator == (), operator < ()

15

Set like Operations on Sorted Structures (1/2)

- includes()
  - Is a range a subset of another range
- set_union()
  - Union of first with second (s1 U s2)
- set_intersection()
  - Intersection of first with second (s1 s2)

16

Set like Operations on Sorted Structures (2/2)

- set_difference()
  - Difference of first with second
- set_symmetric_difference()
  - Union of difference s1,s2 and difference s2,s1
- See Algorithms Chapter

17

Example
Set

```
template <typename T, typename C> void PrintSet(const std::set<T, C>& s)
{
  std::copy(s.begin(), s.end(), std::ostream_iterator<T>(std::cout, ", "));
  std::cout<<std::endl;
}

void main()
{
  // Fill two arrays with values
  double arr1[6]={8.5, 9.2, 4.1, 3.5, 8.5, 8.1};
  double arr2[6]={5.3, 6.2, 4.1, 8.5, 6.5, 1.1};

  std::set<double> s1(arr1, arr1+6);          // Create set of arr1
  std::set<double> s2(arr2, arr2+6);          // Create set of arr2
  std::set<double, std::greater<double> > s3; // Set with sort FO

  // Make union of s1 and s2 and insert result in s3
  std::set_union(s1.begin(), s1.end(), s2.begin(), s2.end(),
                 std::inserter(s3, s3.begin()));
  // Print sets
  PrintSet(s1);    // 3.5, 4.1, 8.1, 8.5, 9.2
  PrintSet(s2);    // 1.1, 4.1, 5.3, 6.2, 6.5, 8.5
  PrintSet(s3);    // 9.2, 8.5, 8.1, 6.5, 6.2, 5.3, 4.1, 3.5, 1.1
}
```

18

Maps (1/2)

- A map is a model of
  - Pair Associative Container
  - Unique Associative Container
  - Sorted Associative Container
- Defined in <map> header
- map<Key, T, Compare, Allocator>

19

Maps (2/2)

- Element consist of key and data
  pair<const Key, T>
- Pairs ordered on key based on user supplied comparator function object
- Duplicate keys not supported
- Returns constant BI-directional iterators
  - Not completely constant: it->second = x

20

Map Main Functions (1/2)

- Constructor:
  - default, copy and with a given size
- Accessors:
  - begin(), end(), rbegin(), rend()
- Sizes:
  - size(), max_size(), *empty()*, count()
- Insertion:
  - insert()

21

Map Main Functions (2/2)

- Deletion:
  - erase()
- Lookup:
  - find(), lower_bound(), upper_bound(), equal_range()
- Comparison:
  - operator == (), operator < ()
- Access:
  - operator []

22

Example
Map (1/2)

```
// Compare function object for const char*
struct Comp
{
  bool operator () (const char* s1, const char* s2) const
  {
    return strcmp(s1, s2)<0;
  }
};

// Print function for maps
template <typename T1, typename T2, typename C>
void PrintMap(const std::map<T1, T2, C>& m)
{
  std::map<T1, T2, C>::const_iterator it=m.begin();
  for (it=m.begin(); it!=m.end(); it++)
        std::cout<<it->first<<": "<<it->second<<std::endl;

  std::cout<<std::endl;
}
```

23

Example
Map (2/2)

```
void main()
{
  // Create map with key: const char*, val: double and functor
  std::map<const char*, double, Comp> price;
  std::map<const char*, double, Comp>::iterator it;

  // Set the value of elements. (prices of products)
  // If key doesn't exist element will be added automatically.
  price["Milk"]=1.29;
  price["Butter"]=0.99;
  price["Pudding"]=2.29;
  price["Ice"]=4.29;
  price["Milk"]=1.25;      // This will change the price

  PrintMap(price);         // Print map

  // Find the price of Milk
  std::cout<<"Milk price: "<<price["Milk"]<<std::endl;

  // Find the price of butter
  it=price.find("Butter");
  std::cout<<"Butter price: "<<it->second<<std::endl;
}
```

24

Multisets (Bags) (1/2)

- A multiset is a model of:
  - Simple Associative Container
  - Multiple Associative Container
  - Sorted Associative Container
- Declared in <set> header
- multiset<Key, Compare, Allocator>
- Fast retrieval based on keys
- Duplicate keys supported

25

Multisets (Bags) (2/2)

- The data items are the keys
- Ordering can be determined by user supplied comparitor function object
- Returns constant BI-directional iterators
  - Cannot modify elements through iterators
- Well suited for the set algorithms

26

Multiset Main Functions (1/2)

- Constructor:
  - default, copy and with a given size
- Accessors:
  - begin(), end(), rbegin(), rend()
- Sizes:
  - size(), max_size(), empty(), count()

27

Multiset Main Functions (2/2)

- Insertion:
  - insert()
- Deletion:
  - erase()
- Lookup:
  - find(), lower_bound(), upper_bound(), equal_range()
- Comparison:
  - operator == (), operator < ()

28

Example
Multiset

```
// Fill two arrays with values
double arr1[6]={8.5, 9.2, 4.1, 3.5, 8.5, 8.1};
double arr2[6]={5.3, 6.2, 4.1, 8.5, 6.5, 1.1};

// Create multisets of arr1 and arr2
std::multiset<double> s1(arr1, arr1+6);
std::multiset<double> s2(arr2, arr2+6);

// Multiset with sort function object
std::multiset<double, std::greater<double> > s3;

// Make union of s1 and s2 and insert result in s3
std::set_union(s1.begin(), s1.end(), s2.begin(), s2.end(),
               std::inserter(s3, s3.begin()));

// Print multisets
PrintSet(s1);    // 3.5, 4.1, 8.1, 8.5, 8.5, 9.2
PrintSet(s2);    // 1.1, 4.1, 5.3, 6.2, 6.5, 8.5
PrintSet(s3);    // 9.2, 8.5, 8.5, 8.1, 6.5, 6.2,
                 // 5.3, 4.1, 3.5, 1.1
```

29

Multimaps (1/2)

- Multimap is a model of
  - Pair Associative Container
  - Multiple Associative Container
  - Sorted Associative Container
- Defined in <map> header
- multimap<Key, T, Compare, Allocator>

30

Multimaps (2/2)

- Manages a set of ordered key/value pairs
- Pairs ordered by key (based on user-supplied comparitor function)
- More than one value may be associated with a given key
- Returns constant BI-directional iterators
  - Not completely constant: it->second = x

31

Main Member Functions (1/2)

- Constructor:
  - default, copy and with a given size
- Accessors:
  - begin(), end(), rbegin(), rend()
- Sizes:
  - size(), max_size(), empty(), count()

32

Main Member Functions (2/2)

- Insertion:
  - insert()
- Deletion:
  - erase()
- Lookup:
  - find(), lower_bound(), upper_bound(), equal_range()
- Comparison:
  - operator == (), operator < ()

33

Example
Multimap

```
// Create multimap with key: char, val: int
std::multimap<char, int> m;
std::multimap<char, int>::iterator it;

// Fill multimap
m.insert(std::make_pair('a', 5));
m.insert(std::make_pair('c', 4));
m.insert(std::make_pair('b', 3));
m.insert(std::make_pair('a', 1));
m.insert(std::make_pair('c', 9));

// Print number of same key elements (2, 1, 2)
std::cout<<"Nr. of 'a' elements: "<<m.count('a')<<std::endl;
std::cout<<"Nr. of 'b' elements: "<<m.count('b')<<std::endl;
std::cout<<"Nr. of 'c' elements: "<<m.count('c')<<std::endl;

// Print all elements
for (it=m.begin(); it!=m.end(); it++)
    std::cout<<it->first<<", "<<it->second<<std::endl;
```

34