Sequence Containers

Sequence Containers

- Container whose elements are arranged in strict linear order
- Can write code that is independent of particular internal details
- This approach improves software maintainability and flexibility
- Sequence containers are vector, deque and list
- Sequence containers do not do any ordering

2

Remarks

- STL embodies family of abstractions idea: sequences have similar interfaces
- STL Containers are template classes
- Choose best sequence based on performance criteria
- If you understand one type well (e.g. vector) the others will be easy to understand

3

General Interface

- All sequence containers have a same kind of interface
  - Constructors for creating a filled container
  - Different insertion functions
  - Different erase functions
- Sequence containers have two refinements
  - Front insertion sequence
  - Back insertion sequence

4

Front Insertion Sequence

- Get an element from the front
  - front()
  - precondition: !empty()
- Push an element at the front
  - push_front()
- Remove an element from the front
  - pop_front()
  - precondition: !empty()
- Front insertion in constant time

5

Front Type Containers

- The following containers support front operations
  - list (doubly linked list)
  - deque

6

Back Insertion Sequence

- Get an element from the back
  - back()
  - precondition: !empty()
- Push an element to the back
  - push_back()
- Remove an element from the back
  - pop_back()
  - precondition: !empty()
- Back Insertion in constant time

7

Back Type Containers

- The following containers support back operations
  - vector
  - list
  - deque

8

Container Types

- Each container defines its own
  - Element type
  - Pointer to element
  - Reference to element
  - Iterators to traverse container

9

Vector (1/2)

- Fast random access to a sequence of dynamically varying length
- Fast insertions/deletions at end of sequence
- Insertions/deletions at front take linear time
- In this case use a deque
- Random access iterators provided

10

Vector (2/2)

- A vector has two internal sizes
  - The number of elements
  - Its capacity

11

Main Member Functions (1/2)

- Constructor:
  - default, copy and with a given size
- Accessors:
  - begin(), end(), rbegin(), rend()
- Insertion:
  - push_back(), insert()
- Deletion:
  - pop_back(), erase()

12

Main Member Functions (2/2)

- Sizes:
  - size(), max_size(), capacity()
- Elements:
  - operator[]
- Ability to reserve storage for future extensions
  - reserve (size_type n)

13

Example
Vector

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

void main()
{
  std::vector<int> v;  // Create vector with ints
  v.reserve(10);       // Reserve space for 10 elements

  v.push_back(10);     // Add element
  v.push_back(20);     // Add element
  v[0]=30;             // Change first element

  std::cout<<"element 0: "<<v[0]<<std::endl;        // 30
  std::cout<<"element 1: "<<v[1]<<std::endl;        // 20
  std::cout<<"Size: "<<v.size()<<std::endl;         // 2
  std::cout<<"Capacity: "<<v.capacity()<<std::endl; // 10
  v.clear();                                       // Clear vector
}
```

14

List (1/2)

- Is a doubly linked list
- Supports forward and backward traversal
- No random iterators
  - Some key generic algorithms cannot be used
- Insertion never invalidates iterators

15

List (2/2)

- Deletion only invalidates the element being deleted
- Splicing possible
  - Transferring elements from one sequence to another

16

Main Member Functions in List

- Constructor:
  - default, copy and with a given size
- Accessors:
  - begin(), end(), rbegin(), rend()
- Insertion:
  - push_front(), push_back(), insert()
- Deletion:
  - pop_front(), pop_back(), erase()

17

Other Specials Functions in List

- splice() // copies part into another list
- unique() // makes elements unique
- remove() // remove all elements with a given value
- merge() // merging of two lists
- sort() // sort a list based on a comparison function object
- swap() // swap contents of two lists

18

Example
List

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

void main()
{
  std::list<int> l;   // Create list

  l.push_back(10);    // Add element
  l.push_front(20);   // Add element
  l.push_back(30);    // Add element

  l.sort();           // Sort the list
  l.reverse();        // Reverse the list

  // Print the list
  std::copy(l.begin(), l.end(),
            std::ostream_iterator<int>(std::cout, " "));
  l.clear();
}
```

19

Deque
(double ended queue)

- Similar to vector in terms of functionality
- Main difference is performance
- Insertion/deletion at start of deque take constant time
- Provide random access iterators
- Insertions/deletions in middle take linear time

20

Main Member Functions in Deque

- Many functions the same as for vector
- Does not have capacity() and reserve()
- Additional functions: push_front(), pop_front()

21

Example
Deque

```
#include <iostream>
#include <deque>

void main()
{
  std::deque<int> q;       // Create deque with ints

  q.push_back(10);         // Add element
  q.push_front(20);        // Add element
  q[0]=30;                 // Change first element

  std::cout<<"element 0: "<<q[0]<<std::endl;   // 30
  std::cout<<"element 1: "<<q[1]<<std::endl;   // 10
  std::cout<<"Front: "<<q.front()<<std::endl;  // 30
  std::cout<<"Back: "<<q.back()<<std::endl;    // 10
  std::cout<<"Size: "<<q.size()<<std::endl;    // 2
  q.clear();                                   // Clear deque
}
```

22