The Standard Template Library (STL) Containers

Curtis Larsen

DSU-CIT

Spring 2014
1 Containers
   What is a container?
   What types of containers exist?
   Using containers

2 Iterators
   What is an iterator?
   Using iterators

3 Challenges
   array
   vector
   stack/queue/list
   array
   iterator
What is a container?

- Containers are data types that hold other information.
- A program that needs to manage a collection of students enrolled in a course will need a container to store all of the student objects.
- A spam filter program will need a container to store the messages being considered, and another to store all of the words and their spam scores.
Types of containers in the STL

http://www.cplusplus.com/reference/stl/

- vector
- stack
- queue
- list
- deque
- priority_queue
- map
- set
- others not listed here
Using the vector

Common operations

• `std::vector<type> v;` // create a vector
• `v.push_back(x);` // add item to end of vector
• `v.size();` // get number of items in vector
• `y = v[i]; v[i] = y;` // read/write item value
Using the stack

Common operations

- `std::stack<type> s;` // create a stack
- `s.empty();` // is the stack empty?
- `s.push(x);` // add new item to top of stack
- `y = s.top();` // retrieve copy of top item
- `s.pop();` // remove the top item (do not retrieve it)
Using the queue

Common operations

• `std::queue<type> q; // create a queue`
• `q.empty(); // is the queue empty?`
• `q.push(x); // add new item to end of queue`
• `y = q.front(); // retrieve copy of front item`
• `q.pop();  // remove the front item (do not retrieve it)`
Using the list

Common operations

- `std::list<type> l;` // create a list
- `l.empty();` // is the list empty?
- `l.clear();` // remove all items
- `l.size();` // get number of items in list
- `l.push_front(x);` // add new item to front of queue
- `l.push_back(x);` // add new item to end of queue
- `l.back();` // retrieve copy of end item
- `l.front();` // retrieve copy of front item
- `l.pop_back();` // remove the end item (do not retrieve it)
- `l.pop_front();` // remove the front item (do not retrieve it)
What is an iterator?

- Iterators are “pointer-like” data that allow iteration over the items in a container.
- A program that needs to manage a collection of students enrolled in a course will need a container to store all of the student objects. It will need to iterate over the students to calculate final grades.
- A spam filter program will need a container to store the messages being considered, and another to store all of the words and their spam scores. It will need to iterate over the messages to classify each one as spam or ham.
Indexing

- With arrays, a typical traversal uses a for loop and the legal index values:

```c++
int i, array[5];
for(i = 0; i < 5; i++) {
    array[i] = i*i;
}
```

- With vectors, indexes may also be used:

```c++
std::vector<int> v; v.resize(5);
int i;
for(i = 0; i < 5; i++) {
    v[i] = i*i;
}
```

- However, lists cannot use indexing.
Element pointers

- With arrays, we can also traverse using a pointer to each item:

```c
int *aptr, array[5], i;
for(aptr = array, i = 0; aptr < array + 5;
    aptr++, i++) {
    *aptr = i*i;
}
```

- With vectors, we can also traverse using a pointer-like interface:

```c
std::vector<int> v;  v.resize(5);
std::vector<int>::iterator vitr;
int i;
for(vitr = v.begin(), i = 0; vitr != v.end();
    vitr++, i++) {
    *vitr = i*i;
}
```
Element pointers

• Even better, with lists, we can also use this technique:

```cpp
std::list<int> l; l.resize(5);
std::list<int>::iterator litr;
int i;
for(litr = l.begin(), i = 0; litr != l.end(); litr++, i++) {
    *litr = i*i;
}
```
Write programs that:

- Generates 100 random numbers from -1000 to 1000, storing them in an array. After generating them, display the values from the array.

- Like the previous problem, but write a function to sort the numbers in ascending order before displaying them.
Write programs that:

- Generates 100 random numbers from -1000 to 1000, storing them in a vector. After generating them, display the values from the vector.
- Asks the user how many numbers to randomly generate, and the range of the numbers. Then as the previous problem.
- Like the previous problem, but write a function to sort the numbers in ascending order before displaying them.
Write programs that:

- Generates 100 random numbers from -1000 to 1000, storing them in a stack/queue/list. After generating them, display the values from the stack/queue/list.

- Asks the user how many numbers to randomly generate, and the range of the numbers. Then as the previous problem.

- (list only) Can you write a function that sorts the numbers in ascending order?

- Read a collection of numbers from the user (until a 0 is given). Use a queue and a stack to store the numbers. If the set of numbers is the same forwards and backwards, report “true” to the output. Otherwise, report “false”.
Write programs that:

- Generates 100 random numbers from -1000 to 1000, storing them in an array. After generating them, display the values from the array. Do not use any indexing, only use pointer arithmetic.

- Like the previous problem, but write a function to sort the numbers in ascending order before displaying them. Do not use any indexing, only use pointer arithmetic.
Write programs that:

- Generates 100 random numbers from -1000 to 1000, storing them in a vector/list. After generating them, display the values from the array. Do not use any indexing, only use iterators.

- Like the previous problem, but write a function to sort the numbers in ascending order before displaying them. Do not use any indexing, only use iterators.