The Standard Template Library Classes

Lecture 38
Sections 9.7, 9.8

Robb T. Koether

Hampden-Sydney College

Thu, Apr 23, 2009
Outline

1. The Standard Template Library
2. The Container Classes
3. The vector Class
4. The stack Class
5. The map Class
6. Assignment
The Standard Template Library (STL) was added to the C++ standard in 1994.

It contains

- Container classes - Hold collections of objects.
- Iterators - Iterate through containers.
- Generic algorithms - Perform standard procedures on objects.
The Container Classes

- The basic container classes are
  - vector
  - deque (double-ended queue)
  - list

- The adaptor classes use container classes.
- The adaptor classes are
  - stack
  - queue
  - priority_queue
Other container classes

- set
- multiset
- map
- multimap
- bitset
The Container Classes

Visit the website

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

for a full description of the STL.
### vector Constructors

- **vector()**  
  Constructs the default constructor.

- **vector(vector)**  
  Constructs the copy constructor.

- **vector(sz, value)**  
  Constructs a vector of size `sz`, filled with `value`.

- **vector(begin, end)**  
  Constructs a vector with range of values given by the iterators `begin` and `end`.

We will use the `vector` class as an example.
The **vector** Container Class

### The Assignment Operator

```plaintext
vector& operator=(vector);
Assigns one vector to another.
```
The **vector** Container Class

### The Capacity Functions

- **size()**
  - Returns the number of elements in the vector.

- **max_size()**
  - Returns the maximum size possible.

- **resize(sz, value)**
  - Changes the size to `sz`, filling in with `value`.

- **capacity()**
  - Returns current capacity.

- **empty()**
  - Determines whether the vector is empty.

- **reserve(cap)**
  - Sets the capacity to `cap`. 
### Element Access Functions

- **`operator[](i)`**
  - Returns the element in position `i`.

- **`at(i)`**
  - Same as `operator[](i)`, but with range checking.

- **`front()`**
  - Returns the element in the first position.

- **`back()`**
  - Returns the element in the last position.
The vector Container Class

Mutator Functions

- **assign(n, value)**
  Replaces the contents with \( n \) copies of value.

- **assign(begin, end)**
  Replaces the contents with the range of values from begin to end.

- **push_back(value)**
  Appends \( value \) to the end of the vector.

- **pop_back()**
  Removes the last element.
The `vector` Container Class

**Mutator Functions**

- `insert(it, value)`
  Inserts `value` in position given by the iterator `it`.

- `insert(it, n, value)`
  Inserts `n` copies of `value` starting in position given by the iterator `it`.

- `insert(it, begin, end)`
  Starting in position given by iterator `it`, inserts values in range given by iterators `begin` and `end`. 
The **vector** Container Class

### Mutator Functions

- **erase(it)**
  Removes element in position given by the iterator `it`.

- **erase(begin, end)**
  Removes range of elements given by the iterators `begin` and `end`.

- **swap(vector)**
  Swaps this vector with the given `vector`.

- **clear()**
  Removes all elements.
The **vector** Container Class

**Iterator Functions**

- **begin()**
  Returns iterator set to beginning.

- **end()**
  Returns iterator set to end.

- **rbegin()**
  Returns reverse iterator set "reverse beginning."

- **rend()**
  Returns reverse iterator set to "reverse end."
The vector Container Class

Example (Programming with vectors)

- Write a program that creates a vector of ints, adds some ints to it, and then prints the list.
The `vector` Container Class

Example (Programming with `vectors`)

```cpp
#include <vector>

int main()
{
    vector<int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    vector<int>::iterator it;
    for (it = v.begin(); it != v.end(); it++)
        cout << *it << endl;
}
```
Example

- Download and run the program `STLVectorTest.cpp`. 
The stack Adaptor Class

- An adaptor class uses a container class.
- We may construct a stack in any of the following ways.

### Ways to Construct a Stack

```cpp
#include <stack>

int main()
{
    stack<int> s1;
    stack<int, vector<int> > s2;
    stack<int, deque<int> > s3;
    stack<int, list<int> > s4;
}
```
The stack class has the following member functions (besides the fundamental four).

**stack Member Functions**

- `bool empty() const;`
- `int size() const;`
- `T& top();`
- `void push(const T& value);`
- `void pop();`
The map Container Class

- A map is an associative list.
- Each member has
  - A key.
  - A value.
- The key must be unique for that member.
- The value is accessed through the key, by matching the key.
Suppose we want to store a list of students and their declared majors.

<table>
<thead>
<tr>
<th>Name</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Mathematics</td>
</tr>
<tr>
<td>Tim</td>
<td>Computer Science</td>
</tr>
<tr>
<td>Betty</td>
<td>Chemistry</td>
</tr>
<tr>
<td>Ann</td>
<td>Mathematics</td>
</tr>
</tbody>
</table>
The map Container Class

- If we intend to locate members by name, then
  - The name is the key
  - The major is the value.
- We construct the (empty) map:

```
#include <map>
map<string, string> major;
```
The map Container Class

- To add the data, we may use the subscript operator:

```
Initialize the map

major["John"] = "Mathematics";
major["Tim"] = "Computer Science";
major["Betty"] = "Chemistry";
major["Ann"] = "Mathematics";
```
The map Container Class

- To find "John", we use the `find()` function.
- It returns an iterator to John’s location in the map.

```cpp
map<string, string>::iterator it;
it = major.find("John");
```
The map Container Class

- The data members first and second store the key and the value.

Print the map

```cpp
map<string, string>::iterator it;
for (it = major.begin(); it != major.end(); it++)
    cout << it.first << " is majoring in " << it.second << endl;
```
The map Container Class

- Download and run the program `STLMapTest.cpp`.
Assignment

Homework

- Read Section 9.7, pages 503 - 510.
- Read Section 9.8, pages 511 - 515.