### Brief Aside on STL
(e.g. used in umlcc code examples)

- STL = “Standard Template Library”
- adopted as part of the C++ standard library
- provides a set of utility ADTs (Abstract Data Types)
- supports the idea of “Generic Algorithm”
- makes efficiency guarantees (using “O” notation)
- hides much storage allocation stuff

### Generic Algorithm

- Tries to separate interface issues from efficiency ones
- so that more efficient implementations of data abstractions can be substituted in a matter transparent to the algorithm.
- Based on ideas such as:
  - container classes
  - iterators
  - plug compatibility

### STL Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containers</td>
<td>store objects of arbitrary type</td>
</tr>
<tr>
<td>Iterators</td>
<td>sequence through containers</td>
</tr>
<tr>
<td>Generic Algorithms</td>
<td>do things with components</td>
</tr>
<tr>
<td>Function Objects</td>
<td>modify the interface of other components</td>
</tr>
<tr>
<td>Adaptors</td>
<td>modify the interface of other components</td>
</tr>
<tr>
<td>Allocators</td>
<td>abstract the memory model</td>
</tr>
</tbody>
</table>

### STL More Containers
(T is a type parameter)

- Sorted Associative Containers:
  - `set<Key>`: unique keys
  - `multiset<Key>`: multiple keys of same value
  - `map<Key, T>`: unique keys with associated value
  - `multimap<Key, T>`: multiple keys with associated values
  - About 40 varieties of container in general

```cpp
// Create an array of strings.
string word[] = {"The", "quick", "brown", "fox", "just", "plays"};

// Compute the number of elements.
int numElements = sizeof(word)/sizeof(char*);

STL Code Example

// STL stuff starts here.
// Copy the array into a list
list<string> s;
for( int i = 0; i < numElements; i++ )
{
    s.push_back(word[i]);
}  // push on "back" (end) of list
```
STL Iterators

- Behave as an abstract form of pointers
- *, ++, -- operations, etc.
- 25 or so varieties of iterator

```
list<string>::iterator p = s.begin(); // point to beginning of list
if ( p != s.end() )
{
    cout << *p;                                    // output first string, if there is one
    p++;}
while( p != s.end() )
{
    cout << *p;                        // output other strings, with spacing
    p++;
}
```

Generic Algorithms in STL
(a few of over 80)

- sort
- find
- merge
- reverse
- for_each
- accumulate
- binary_search
- next_permutation

```
// Output list of strings with a blank between each, but none at the end.
// Add a period at the end.
list<string>::iterator p = s.begin(); // point to beginning of list
if ( p != s.end() )
{
    cout << *p; // output first string, if there is one
    p++;
}
while( p != s.end() )
{
    cout << *p << *p; // output other strings, with spacing
    p++;
}
```

Function Objects in STL
(30 or so, 10 general classes)

- less<T> function object comparing two elements of type T for <
- equal<T>
- greater<T>
- divides<T>
- plus<T>

Adapters in STL

- stack< list<T> >
  adapts a list<T> for use as a stack

- queue< vector<T> >
  adapts a vector<T> for use as a queue.