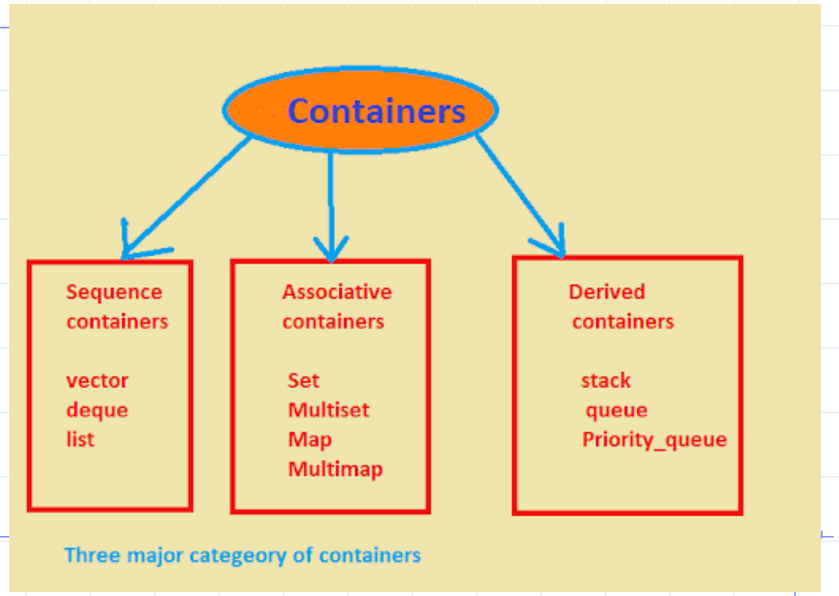
STL Basics

(no, not really)



The C++ Standard Template Library

- A collection of template based containers and algorithms including:
  - list - based on the linked list concept
  - vector - a resizable array
  - set - based on a balanced binary tree
  - multiset - set that allows duplicate entries
  - map - tracks <key, value> pairs

The STL list class

- Some useful methods (there are many others):
  - size() - returns number of elements in list
  - front(), back() - returns (reference to) first or last elements
  - push_front(x), push_back(x) - adds x to front or end of list
  - pop_front(), pop_back() - removes first or last element
  - clear() - empties list
  - remove(x) - removes x from list
  - sort() - sorts list (according to <)
  - reverse() - reverses the list.
  - begin() - returns an iterator “pointing” to beginning of list
  - end() - returns an iterator “pointing” to end of list
    - note: there is a “dummy node” at end of STL lists

Iterators In STL

- An iterator can be used to “walk” through an STL Container.
- Declared as:

  {container-type}::iterator {iterator-variable-name}

  or

  {container-type}::const_iterator {iterator-variable-name}
- where
  - {container-type} would be an instantiated container type
    - list<int>
    - list<string>
  - {iterator-variable-name} is a C++ variable name
- use const_iterator when the associated container is const
- Some examples:
  - list<int>::iterator lii;
  - list<string>::const_iterator cstr_iter;

More on STL Iterators

- The most common thing to do with iterators is
  - overloaded++ : move to next item in container.
  - overloaded-- : move to previous item in container.
  - overloaded * (unary) : returns node contents

- For example:

```
list<string> someStrings;
someStrings.push_back("90");
someStrings.push_back("73");
someStrings.push_front("42"); // list now 42,90,73
for(list<string>::iterator lsi=someStrings.begin();
    lsi!=someStrings.end();
    lsi++)
  cout << *lsi << endl;
```

The STL vector class

- Some useful methods (there are many others):
  - size() - returns number of elements in vector
  - push_back(x) - adds x to end of vector
  - pop_back() - removes last element
  - clear() - empties vector
  - begin() - returns an iterator “pointing” to beginning of vector
  - end() - returns an iterator “pointing” to end of vector

- many vector methods not available here
- … but some others are:
  - operator[ ] is overloaded.
  - data() returns internal C++ array equivalent of vector.

The STL set class

- Some useful methods (there are many others):
  - size() - returns number of elements in set
  - insert(x) - adds x to set; position is not guaranteed
  - erase(x) - removes element x
  - clear() - empties set
  - begin() - returns an iterator “pointing” to beginning of set
  - end() - returns an iterator “pointing” to end of set

- others methods are:
  - find(x) returns iterator “pointing” at x in set
  - count(x) returns count of how many times x is in set (0 or 1)

- All operations are O(log n) … really!

The STL multiset class

- same as set, except allows duplicate values
- Some useful methods (there are many others):
  - size() - returns number of elements in multiset
  - insert(x) - adds x to multiset; position is not guaranteed
  - erase(x) - removes element x
  - clear() - empties multiset
  - begin() - returns an iterator “pointing” to beginning of multiset
  - end() - returns an iterator “pointing” to end of multiset

- others methods are:
  - find(x) returns iterator “pointing” at first x in multiset
  - count(x) returns count of how many times x is in multiset (>=0)

- All operations are O(log n) … really!

The STL map class

- Requires two templated eyes - the key and the value
  - size() - returns number of elements in map
  - clear() - empties list
  - erase(k) - removes key, value pair with key=k
  - begin() - returns an iterator “pointing” to beginning of map
  - end() - returns an iterator “pointing” to end of map
  - overloads operator[ ] with key value

- Example:

```
map<char, string> myMap;
myMap['a']="an element";
myMap['c']="another element";
std::cout << "mymap['a'] is " << mymap['a'] << endl;
std::cout << "mymap['b'] is " << mymap['b'] << endl;
std::cout << "mymap['c'] is " << mymap['c'] << endl;
```