# Lecture 22

- STL overview
- Sequence containers
  - vector<T>
  - Performance
  - Big-O notation
  - Amortized constant time for appending elements



# Standard Template Library Overview

- Collection of commonly used container classes
  - Sequence containers (elements have predef. locations)
    - **vector, list, deque, basic_string, slist, rope**
  - Associative containers (elem. location based on key)
    - **set, multiset, map, multimap, hash_set, hash_multiset, hash_map, hash_multimap**
- Iterators
  - Pointer-like types used for traversing STL containers
- Collection of container-independent algorithms
  - **sort, find, merge, random_shuffle ...**



# Examples

```
#include <vector>
#include <algorithm>
using namespace std;

int main()
{  // CONTAINER + ALGORITHM
   vector<int> v(10);  // vector of 10 ints, fill with...
   generate(v.begin(), v.end(), rand); //... random values
   v[0] = 6; // access like array

   // loop through vector using ITERATORS
   vector<int>::iterator it = v.begin();
   vector<int>::iterator en = v.end();
   int sum=0;
   for (; it != en; ++it) sum += *it;

   // ALGORITHM: shuffle elements randomly
   random_shuffle(v.begin(), v.end());

   // MEMBER FUNCTIONS: if non-empty, erase first element
   if (!v.empty()) v.erase(v.begin());
}
```

- Several implementations exist
- g++ comes with SGI version
- Web-sites
  - SGI STL site: **www.sgi.com/tech/stl** - good!
  - STLport site: **www.stlport.org**
  - Boost site: **www.boost.org** Other useful libraries
- Literature on C++ templates and STL
  - **Meyers, "Effective STL"** - eye-opening & witty!
  - **Vandervoorde, "C++ Templates: The Complete Guide"** - well, complete ;-)
  - **Alexandrescu, "Modern C++ Design"**- mind-boggling

# Sequence Containers

- vector<T>
  - vector template, array functionality
  - element type is T
- list<T>
  - doubly linked list template
  - data associated with node is T



# vector<T>

- #include <vector>
- Sequence that allows **random access** to elements
- Simplest STL container, often most efficient one
- **Amortized constant time** insertion/deletion at the end
- **Linear time** insertion/removal at the beginning and in the middle
- Vectors can grow and shrink
- Compatible with arrays: elements are laid out consecutively in memory
- Iterators which refer to vector elements are **invalidated** after insert/delete operations



# vector<T> Example

```
#include <vector>
using namespace std;

int main()
{
   const int N = 1000;
   vector<int> v;    // empty vector

   v.reserve(N);     // reserve memory for N elements
                     // saves time and memory! v.size() still 0

   // append N elements
   for (int i=0; i < N; ++i) v.push_back(i);

   // add up all elements, array syntax
   int s = 0;
   for (size_t i=0; i < v.size(); ++i) sum += v[i];
   // remove all elements one by one
   for (int i=0; i < N; ++i) v.pop_back();
   assert(v.empty());
   // v is destroyed here; if v contains pointers,
   // destructors are not called on the objects!
}
```

# Commonly used vector members

- iterator **begin**() : returns iterator to first element
- iterator **end**() : returns iterator to end (last element + 1)
- size_type **size**() : # of elements in vector
- bool **empty**() const : true iff vector is empty (faster than !size())
- void **push_back**(const T&) : inserts new element at the end (amortized constant time)
- void **pop_back**() : removes last element
- reference **operator[]**(size_type n): returns n-th element (starts with 0)
- reference **back**() : returns reference to last element
- void **clear**() : remove all elements
- void **erase**(iterator pos) : removes element at position pos
- void **reserve**(size_type n) : allocates memory for n elements
- bool **operator==**(const vector&, const vector&) : equality