C++ STL Continued

CSE 333

Guest Lecturer: Hal Perkins

Instructor: Alex Sanchez-Stern

Teaching Assistants:
Justin Tysdal
Sayuj Shahi
Nicholas Batchelder
Leanna Mi Nguyen

STL Containers ☺ (review)

- A container is an object that stores (in memory) a collection of other objects (elements)
  - Implemented as class templates, so hugely flexible
  - More info in *C++ Primer* §9.2, 11.2

- Several different classes of container
  - Sequence containers (`vector`, `deque`, `list`, `...`)
  - Associative containers (`set`, `map`, `multiset`, `multimap`, `bitset`, `...`)
  - Differ in algorithmic cost and supported operations

STL `iterator` (review)

- Each container class has an associated iterator class (*e.g.* `vector<int>::iterator`) used to iterate through elements of the container
  - http://www.cplusplus.com/reference/std/iterator/
  - Iterator range is from `.begin()` up to `.end()`
    - `end` is one past the last container element!
  - Some container iterators support more operations than others
    - All can be incremented (`++`), copied, copy-constructed
    - Some can be dereferenced on RHS (*e.g.* `x = *it;`)
    - Some can be dereferenced on LHS (*e.g.* `*it = x;`)
    - Some can be decremented (`--`)
    - Some support more (`[]`, `+`, `-`, `+=`, `-=`, `<`, `>` operators)

STL Algorithms (review)

- A set of functions to be used on ranges of elements
  - Range: any sequence that can be accessed through *iterators* or *pointers*, like arrays or some of the containers
  - General form: `algorithm(begin, end, ...);`
- Algorithms operate directly on range *elements* rather than the containers they live in
  - Make use of elements' copy ctor, =, ==, !=, <
  - Some do not modify elements
    - *e.g.* find, count, for_each, min_element, binary_search
  - Some do modify elements
    - *e.g.* sort, transform, copy, swap

Vector Algorithms Example (review)

vectoralgos.cc

```
#include <vector>
#include <algorithm>
#include "Tracer.h"
using namespace std;

void PrintOut(const Tracer& p) {
  cout << " printout: " << p << endl;
}

int main(int argc, char** argv) {
  Tracer a, b, c;
  vector<Tracer> vec;

  vec.push_back(c);
  vec.push_back(a);
  vec.push_back(b);
  cout << "sort:" << endl;
  sort(vec.begin(), vec.end());
  cout << "done sort!" << endl;
  for_each(vec.begin(), vec.end(), &PrintOut);
  return EXIT_SUCCESS;
}
```

STL `list`

- A generic doubly-linked list
  - http://www.cplusplus.com/reference/stl/list/
  - Elements are ***not*** stored in contiguous memory locations
    - Does not support random access (*e.g.* cannot do `list[5]`)
  - Some operations are much more efficient than vectors
    - Constant time insertion, deletion anywhere in list
    - Can iterate forward or backwards
  - Has a built-in sort member function
    - Doesn't copy! Manipulates list structure instead of element values

list Example

listexample.cc

```
#include <list>
#include <algorithm>
#include "Tracer.h"
using namespace std;

void PrintOut(const Tracer& p) {
  cout << " printout: " << p << endl;
}

int main(int argc, char** argv) {
  Tracer a, b, c;
  list<Tracer> lst;

  lst.push_back(c);
  lst.push_back(a);
  lst.push_back(b);
  cout << "sort:" << endl;
  lst.sort();
  cout << "done sort!" << endl;
  for_each(lst.begin(), lst.end(), &PrintOut);
  return EXIT_SUCCESS;
}
```

STL `map`

- One of C++'s *associative* containers: a key/value table, implemented as a search tree
  - http://www.cplusplus.com/reference/stl/map/
  - General form: `map<key_type, value_type> name;`
  - Keys must be *unique*
    - `multimap` allows duplicate keys
  - Efficient lookup (O(log n)) and insertion (O(log n))
    - Access `value` via `name[key]`
  - Elements are type `pair<key_type, value_type>` and are stored in *sorted* order (key is field `first`, value is field `second`)
    - Key type must support less-than operator (`<`)

map Example

mapexample.cc

```
void PrintOut(const pair<Tracer,Tracer>& p) {
  cout << "printout: [" << p.first << "," << p.second << "]" << endl;
}

int main(int argc, char** argv) {
  Tracer a, b, c, d, e, f;
  map<Tracer,Tracer> table;
  map<Tracer,Tracer>::iterator it;

  table.insert(pair<Tracer,Tracer>(a, b));
  table[c] = d;
  table[e] = f;
  cout << "table[e]:" << table[e] << endl;
  it = table.find(c);

  cout << "PrintOut(*it), where it = table.find(c)" << endl;
  PrintOut(*it);

  cout << "iterating:" << endl;
  for_each(table.begin(), table.end(), &PrintOut);

  return EXIT_SUCCESS;
}
```

Unordered Containers (C++11)

- `unordered_map`, `unordered_set`
  - And related classes `unordered_multimap`, `unordered_multiset`
  - Average case for key access is O(1)
    - But range iterators can be less efficient than ordered `map`/`set`
  - See *C++ Primer*, online references for details