CPSC 427a: Object-Oriented Programming

Michael J. Fischer

Lecture 17
November 4, 2010

The C++ Standard Library (cont.)

The C++ Standard Library (cont.)

Iterators

Iterators are like generalized pointers into containers.

Most pointer operations `*`, `->`, `++`, `==`, `!=`, etc. work with iterators.

- `begin()` returns an iterator pointing to the first element of the vector.
- `end()` returns an iterator pointing past the last element of the vector.

Iterator example

Here's a program to store and print the first 10 perfect squares.

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

int main() {
  vector<int> tbl(10);
  for (unsigned k=0; k<10; k++) tbl[k] = k*k;
  vector<int>::iterator pos;
  for (pos = tbl.begin(); pos != tbl.end(); pos++)
    cout<< *pos<< endl;
}
```

Using iterator inside a class

```
#include <iostream>
#include <vector>
using namespace std;
class Squares : vector<int> {
public:
  Squares(unsigned n) : vector<int>(n) {
    for (unsigned k=0; k<n; k++) (*this)[k] = k*k; }
  ostream& print(ostream& out) const {
    const_iterator pos;   // must be const_iterator
    for (pos=begin(); pos!=end(); pos++) out<< *pos<< endl;
    return out; }
};
int main() {
  Squares sq(10);
  sq.print(cout);
}
```

Using subscripts and size()

```
#include <iostream>
#include <vector>
using namespace std;
class Squares : vector<int> {
public:
  Squares(unsigned n) { for (unsigned k=0; k<n; k++) push_back(k
  ostream& print(ostream& out) const {
    for (unsigned k=0; k<size(); k++) out<< (*this)[k]<< endl;
    return out; }
};
int main() {
  Squares sq(10);
  sq.print(cout);
}
```

Algorithms

STL has algorithms as well as data structures.

You must `#include <algorithm>`.

Commonly used: `copy`, `fill`, `swap`, `max`, `min`, `max_element`, `min_element`, but there are many many more.

We'll look at `sort` in greater detail.

STL `sort` algorithm

`sort` works only on randomly-accessible containers such as `vector`. (`list` has its own sort method.)

`sort` takes two iterator arguments to designate the sort range.

It can also take an optional third "comparison" argument to define the sort order.

Reverse sort example

```
class Squares : vector<int> {
public:
  Squares(unsigned n) {for (unsigned k=0; k<n; k++) push_back(k*

  // decreasing order; *** must be static ***
  static bool cmp( const int& x1, const int& x2) {
    return x1 > x2; }

  void rsort() { sort(begin(), end(), cmp); }

  ostream& print(ostream& out) const {
    for (unsigned k=0; k<size(); k++) out<< (*this)[k]<< endl;
    return out; }
};
```

Reverse sort example (cont.)

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

class Squares : vector<int> {
  ...
};

int main() {
  Squares sq(10);
  sq.rsort();
  sq.print(cout);
}
```

pair<T1, T2>

A pair<T1, T2> is an ordered pair of elements of type T1 and T2, respectively.

Class pair<T1, T2> has public data members first and second.

Example:

```
pair<string, double> item("book", 49.95);
                              // makes pair <"book", 49.95>
cout<< item.first;            // prints "book"
cout<< item.second;           // prints 49.95
```

map<Key,Val>

map<Key,Val> associates a value with each key.

More precisely, it is an ordered collection of elements of type pair<Key,Val>.

You must #include <map>.

Can use standard subscript notation to access map contents, where subscript is the key.

Can also use a map iterator, which returns a pointer to a pair.

Using a map<Key,Val>

Example:

```
typedef map<string,double> myMap; // alias for convenience
myMap::iterator pos;
myMap m;                          // a map from strings to doubles
m["dog"];                         // puts pair <"dog",0.0> into m
m["bird"]=5.2;                    // puts pair <"bird",5.2> into m
pos = m.find("cat");              // returns m.end() for not found
cout<< (pos==m.end())<< endl;// prints 1 (true)
pos = m.find("bird");             // pos points to <"bird",5.2>
if (pos!=m.end()) {
  cout<< pos->first<< endl;  // prints "bird"
  cout<< pos->second<< endl; // prints 5.2;  }
}
```

Copying from one container to another

Two ways to copy multiple elements in one statement.

Suppose `m` is a map and `v` a vector of pairs compatible with `m`.

1. v.assign(m.begin(), m.end());
2. Supply `m.begin()` and `m.end()` as arguments to the `v` constructor.

Copying from one container to another – example

A `map` can be copied into a vector of pairs.

```
#include <iostream>
#include <map>
#include <vector>
#include <string>
using namespace std;
int main() {
  map<string,double> m;
  m["dog"]=3; m["cat"]=2;
  // construct p from m
  vector<pair<string,double> > p(m.begin(),m.end());
  // declare iterator
  vector<pair<string,double> >::const_iterator pos;
  // print p
  for (pos=p.begin(); pos!=p.end(); ++pos)
    cout<< pos->first<< " "<< pos->second<< endl;
}
```

`string` class

The standard `string` class tries to make strings behave like other built-in data types.

Like `vector<char>`, strings are growable, but they are not implemented using `vector`, and they support many special string operations.

They can be assigned (`=`, `assign()`), compared (`==`, `!=`, `<`, `<=`, `>`, `>=`, `compare()`), concatenated (`+`), read and written (`>>`, `<<`), searched (`find()`, ...), extracted (`[]`, `substr()`), modified (`+=`, `append()`, ...), and more.

Their length can be found (`size()`, `length()`).

`s.c_str()` returns a copy of `s` as a C string.

You must `#include <string>`.