CSCI-1200 Computer Science II — Spring 2006
Lecture 14 — Associative Containers (Maps), Part 2

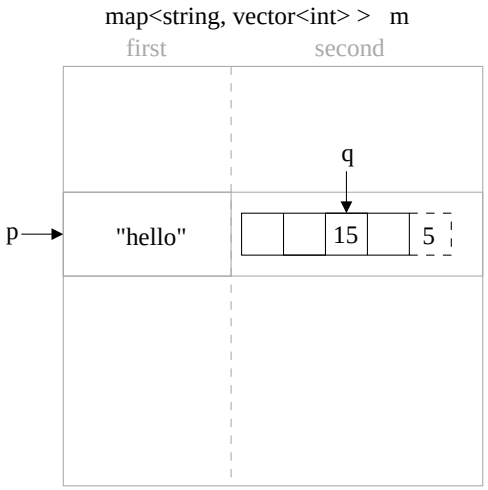
Review of Lecture 13

- Maps are associations between keys and values.
- Maps have fast insert, access and remove operations.
- Maps store pairs; map iterators refer to these pairs.
- The primary map member functions we discussed are `operator[]`, `find`, `insert`, and `erase`.
- The choice between maps, vectors and lists is based on naturalness, ease of programming, and efficiency of the resulting program.

Today's Class — Maps, Part 2

- Maps containing more complicated values.
- Example: index mapping words to the text line numbers on which they appear.
- Maps whose keys are class objects.
- Example: maintaining student records.
- Summary discussion of when to use maps.

14.1 More Complicated Values



- Let's look at the example:

```
map<string, vector<int> > m;
map<string, vector<int> >::iterator p;
```

  Note that the space between the `> >` is **required**. Otherwise, `>>` is treated as an operator.

- Here's the syntax for entering the number 5 in the vector associated with the string `"hello"`:

```
m[string("hello")].push_back(5);
```

- Here's the syntax for accessing the size of the vector stored in the map pair referred to by map iterator p:

```
p = m.find(string("hello"));
p->second.size()
```

  Now, if you want to access (and change) the $\texttt{i}^{th}$ entry in this vector you can either using subscripting:

```
(p->second)[i] = 15;
```

  (the parentheses are needed because of precedence) or you can use vector iterators:

```
vector<int>::iterator q = p->second.begin() + i;
*q = 15;
```

  Both of these, of course, assume that at least `i+1` integers have been stored in the vector (either through the use of `push_back` or through construction of the vector).

- We can figure out the correct syntax for all of these by drawing pictures to visualize the contents of the map and the pairs stored in the map. We will do this during lecture, and you should do so **all the time** in practice.

14.2 Exercise

Write code to count the odd numbers stored in the map

```
map<string, vector<int> > m;
```

This will require testing all contents of each vector in the map. Try writing the code using subscripting on the vectors and then again using vector iterators.

14.3 A Word Index in a Text File

```
// Given a text file, generate an alphabetical listing of the words in
// the file and the file line numbers on which each word appears.  If
// a word appears on a line more than once, the line number is listed
// only once.  A map of strings to vectors stores the information.

#include <algorithm>
#include <cctype>
#include <iostream>
#include <map>
#include <string>
#include <vector>
using namespace std;

vector<string> breakup_v2(const string& line);

int main() {
  map<string, vector<int> > words_to_lines;
  string line;
  int line_number = 0;

  while (getline(cin, line)) {
    line_number++;
    // Break the string up into words
    vector<string> words = breakup_v2(line);

    // Find if each word is already in the map.
    for (vector<string>::iterator p = words.begin(); p!= words.end(); ++p) {
      // If not, create a new entry with an empty vector (default) and
      // add to index to the end of the vector
      map<string, vector<int> >::iterator map_itr = words_to_lines.find(*p);
      if (map_itr == words_to_lines.end())
        words_to_lines[*p].push_back(line_number);  // could use insert here
      // If it is, check the last entry to see if the line number is
      // already there.  If not, add it to the back of the vector.
      else if (map_itr->second.back() != line_number)
        map_itr->second.push_back(line_number);
    }
  }

  // Output each word on a single line, followed by the line numbers.
  map<string, vector<int> >::iterator map_itr;
  for (map_itr = words_to_lines.begin(); map_itr != words_to_lines.end(); map_itr++) {
    cout << map_itr->first << ":\t";
    for (unsigned int i = 0; i < map_itr->second.size(); ++i)
      cout << (map_itr->second)[ i ] << " ";
    cout << "\n";
  }
  return 0;
}
```

```
// this breaks up a line into a vector of alphabetic-char-only strings
vector<string> breakup_v2(const string& line) {
  vector<string> strings;
  string::const_iterator p = line.begin();
  string one_word;

  while (p != line.end()) {
    // Find the beginning of the next alphabetic string
    while (p != line.end() && !isalpha(*p)) p++;

    // If haven't reached the end of the line
    if (p != line.end()) {
      // Restart the word with *p
      one_word = "";
      one_word += tolower(*p);

      // Add remaining letters
      for (p++; p != line.end() && isalpha(*p); p++)
        one_word += tolower(*p);

      // Add word to the vector of strings
      strings.push_back(one_word);
    }
  }
  return strings;
}
```

14.4 Our Own Class as the Map Key

- So far we have used `string` (mostly) and `int` (once) as the key in building a `map`. Intuitively, it would seem that `string` is used quite commonly.
- More generally, we can use any class we want as long as it has an `operator<` defined on it. For example, consider:

```
class Name {
public:
  Name(const string& first, const string& last) :
    m_first(first), m_last(last) {}

  const string& first() const { return m_first; }
  const string& last() const { return m_last; }

private:
  string m_first;
  string m_last;
};
```

- Suppose we wanted a map of names and associated student records, where the student record class stores things like address, courses, grades, and tuition fees and calculates things like GPAs, credits, and remaining required courses. To make this work, we need to add an `operator<`. This is simple:

```
bool operator< (const Name& left, const Name& right) {
  return left.last() < right.last() ||
    (left.last() == right.last() && left.first() < right.first());
}
```

  Now you can define a map:

```
map<Name, StudentRecord> students;
```

- Of course, you would still need to write the `StudentRecord` class!
- Also, some older compilers require the definition of `operator==` as well, so be alert to this.

14.5 Typedefs

- One of the painful aspects of using maps is the syntax. For example, consider a constant iterator in a map associating strings and vectors of ints:

```
map< string, vector<int> > :: const_iterator p;
```

- Typedefs are a syntactic means of shortening this. For example, if you place the line:

```
typedef map< string, vector<int> > map_vect;
```

  before your main function (and any function prototypes), then anywhere you want the map you can just use the identifier `map_vect`:

```
map_vect :: const_iterator p;
```

  The compiler makes the substitution for you.

14.6 An Additional Example: Organizing MP3's

- Let's store information about the MP3 files stored on various computers in a network,
- We'll need to dynamically update this information as computers are added to and removed from the network.
- We'd like to find songs and the fastest connection for downloading them.
- The solution stores the information in a `map`, with a unique identifier for each computer forming the "key" and a class object that maintains information about a computer and its MP3s as the "value".

---

mp3.h

```
#ifndef _mp3_h_
#define _mp3_h_

// An MP3 representation.

#include <string>
using namespace std;

class MP3 {
public:
  MP3() {}
  MP3(const string& artist, const string& title) : artist_(artist), title_(title) {}
  const string& get_artist() const { return artist_; }
  const string& get_title() const { return title_; }
private:
  string artist_;
  string title_;
};

#endif
```

Computer.h

```
#ifndef _computer_h_
#define _computer_h_

// Each computer has the capability of adding an MP3 file, printing
// its MP3 files, determining if a song MP3 is on the computer, or
// finding the songs by a given artist.

#include <list>
#include <string>
#include "mp3.h"
```

```
class Computer {

public:
  Computer(double speed) : speed_(speed) { }

  bool add_mp3(const string& artist, const string& title);
  void print_mp3s() const;
  bool has_song(const string& artist, const string& title, MP3& requested_song) const;
  list<MP3> songs_by_artist(const string& artist) const;
  double getSpeed() const { return speed_; }

private:
  double speed_;     // network connection speed in bytes per second
  list<MP3> mp3s_;   // the songs on the computer; could be a vector just as easily

};

#endif
```

Computer.cpp

```
#include <iostream>
#include "computer.h"

// Add an MP3 to the list of MP3s.  Return false if the song is already there.
bool Computer::add_mp3(const string& artist, const string& title) {
  list<MP3>::iterator p;
  for (p = mp3s_.begin(); p != mp3s_.end(); ++p) {
    if (p->get_artist() == artist && p->get_title() == title) break;
  }
  if (p != mp3s_.end()) {
    return false;
  } else {
    mp3s_.push_back(MP3(artist, title));
    return true;
  }
}

// Iterate through the list of MP3, printing each.
void Computer::print_mp3s() const {
  for(list<MP3>::const_iterator p = mp3s_.begin(); p != mp3s_.end(); ++p) {
    cout << "  ARTIST: " << p->get_artist() << endl
         << "  TITLE: " << p->get_title() << endl;
  }
}

// Returns true if the Computer has the requested song, returns reference to song via parameter
bool Computer::has_song(const string& artist, const string& title, MP3& requested_song) const {
  for (list<MP3>::const_iterator p = mp3s_.begin(); p != mp3s_.end(); ++p) {
    if (p->get_artist() == artist && p->get_title() == title) {
      requested_song = *p;
      return true;
    }
  }
  return false;
}

// Build a list of all songs by the indicated artist.
list<MP3> Computer::songs_by_artist(const string& artist) const {
  list<MP3> results;
  for(list<MP3>::const_iterator p = mp3s_.begin(); p != mp3s_.end(); ++p) {
    if (p->get_artist() == artist) results.push_back(*p);
  }
  return results;
}
```

---

main.cpp

```
// This program processes queries about the MP3 files available on various computers.
// Each computer is identified by a unique id (a string).  Information about the computer,
// including connection speed and a list of MP3's are stored in the computer class object.

#include <algorithm>
#include <iostream>
#include <list>
#include <map>
#include <string>
#include <cctype>
#include "mp3.h"
#include "computer.h"

typedef map<string, Computer> MP3_map;    // since the names of maps tend to be rather long.

void add_computer(MP3_map& mp3_system);   // prototypes of helper functions
void print_mp3s_on_computer(const MP3_map& mp3_system);
void find_song(const MP3_map& mp3_system);
void find_songs_by_artist(const MP3_map& mp3_system);

int main() {
  // the mp3 database
  MP3_map mp3_system;
  bool quit = false;
  // process the input
  char option;
  while (cin >> option) {
    switch (option) {
    case 'a':  add_computer(mp3_system); break;
    case 'p':  print_mp3s_on_computer(mp3_system); break;
    case 'f':  find_song(mp3_system); break;
    case 'F':  find_songs_by_artist(mp3_system); break;
    case 'q':  return 1; break;
    default:
      cout << "Error: " << option << " is not a valid command" << endl;;
      break;
    }
  }
}

// Add a computer to the map of MP3's
void add_computer(MP3_map& mp3_system) {
  string computer_id, line;
  double speed;
  // read id & speed
  cin >> computer_id >> speed;
  getline(cin, line); // read the rest of that line

  // Create a new computer & read its MP3's
  Computer new_computer(speed);
  do {
    string artist,title;
    getline(cin, artist);
    if (artist == "") break;
    getline(cin, title);
    new_computer.add_mp3(artist, title);
  } while (true);

  // Try to add the computer using the map insert function.
  pair<MP3_map::iterator, bool> result_pair = mp3_system.insert(make_pair(computer_id, new_computer));
  // make sure the computer isn't already there!
  assert(result_pair.second);
  cout << "Computer " << computer_id << " added." << endl;
}
```

```
// Print the MP3's for a given computer.
void print_mp3s_on_computer(const MP3_map& mp3_system) {
  string computer_id, line;
  // read id
  cin >> computer_id;
  getline(cin, line);

  // Find the computer
  MP3_map::const_iterator c_itr = mp3_system.find(computer_id);
  if (c_itr == mp3_system.end()) {
    cout << "Computer " << computer_id << " is not in the system." << endl;
  } else {
    cout << "Songs found on " << computer_id << ":\n";
    c_itr->second.print_mp3s();
  }
}

// Find all computers with a particular song.
void find_song(const MP3_map& mp3_system) {
  string line, artist, title;
  getline(cin, line); // read the rest of request line
  // get the artist & title
  getline(cin, artist);
  getline(cin, title);

  // Iterate through the map, looking for the song.
  bool found = false;
  string computer_id;
  double speed;

  MP3_map::const_iterator c_itr;
  for (c_itr = mp3_system.begin(); c_itr != mp3_system.end(); c_itr++) {
    MP3 request;
    if (c_itr->second.has_song(artist, title, request)) {
      if (!found || c_itr->second.getSpeed() > speed) {
        found = true;
        computer_id = c_itr->first;
        speed = c_itr->second.getSpeed();
      }
    }
  }

  if (found)
    cout << "Fastest download for \"" << title << "\" by " << artist << " from " << computer_id << endl;
  else
    cout << "No computer found with \"" << title << "\" by " << artist << endl;
}

// Find and output information about all songs by a given artist.
void find_songs_by_artist(const MP3_map& mp3_system) {
  string line, artist;
  getline(cin, line); // read the rest of request line
  // Get the artist.
  getline(cin, artist);

  // you need to write this for lab




}
```

14.7 When to Use Maps, Reprise

- Maps are an association between two types, one of which (the key) must have a `operator<` ordering on it.
- The association may be immediate:
  - Words and their counts.
  - Words and the lines on which they appear
- Or, the association may be created by splitting a type:
  - Splitting off the name (or student id) from rest of student record.
  - Splitting off the computer id from the rest of the information about the computer and its MP3 files.