

# Computer Science II — CSci 1200

## Lecture 13

### Associative Containers (Maps), Part 2

#### Review of Lecture 12

- 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.

#### More Complicated Values

Let's look at the example of

```
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 is 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  $i^{th}$  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 help 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.

## 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.

## A Word Index in a Text File

Problem: find each word in a text file and record all of the line numbers for each word.

- The implementation uses a map of strings and vectors.
- Each map entry stores a string and a vector of line numbers.
- The code is attached to the lecture notes. We will discuss details during lecture.
- One new piece of C++ code is `getline`, which grabs a line of input up to and including the end-of-line character, and stores it in a string. The end-of-line character is not stored in the string. Anything that was in the string is over-written (unless the end-of-file is reached).

## Exercise

Outline a solution to the word index problem we just solved using a map using a list or a vector instead. Hint: you will need to declare a new class.

## 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<`. As you know, 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, student_record > students;
```

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

## 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.

## An Additional Example

An example solution to a programming problem from a previous semester will be posted on the course web site. The problem involves:

- recording information about the MP3 files stored on various computers in a network,
- dynamically updating this information as computers are added to and removed from the network, and
- finding 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”.

## 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.