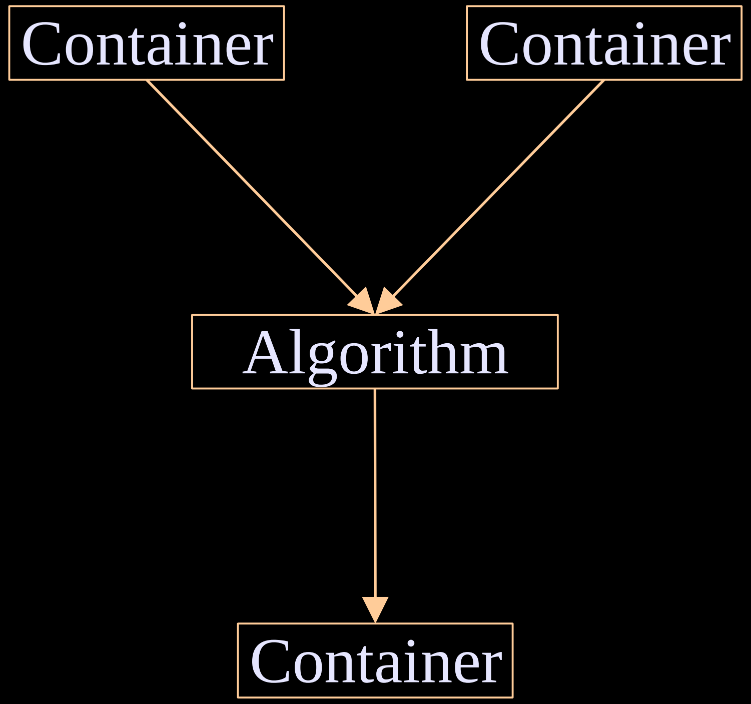
C++ STL Tutorial

Ratchadaporn Kanawong

Silpakorn University

STL components overview

- Data storage, data access and algorithms are separated
  - *Containers* hold data
  - *Iterators* access data
  - *Algorithms*, *function objects* manipulate data
  - *Allocators*... allocate data (mostly, we ignore them)



Sequence Containers

- `vector<T>` - dynamic array
  - Offers random access, back insertion
  - Should be your ***default choice***, ***but choose wisely***
  - Backward compatible with C : `&v[0]` points to the first element
- `deque<T>` - double-ended queue (usually array of arrays)
  - Offers random access, back and front insertion
  - Slower than vectors, no C compatibility
- `list<T>` - 'traditional' doubly linked list
  - Don't expect random access, you can insert anywhere though
- `string` - yes, it is a STL container (a typedef actually)
- Nonstandard containers : `slist`, `rope`…

Associative Containers

- Offer *O(log n)* insertion, suppression and access
- Store only weakly strict ordered types (eg. numeric types)
  - Must have `operator<()` and `operator==()` defined and !(a<b) && !(b<a) ≡ (a==b)
- The sorting criterion is also a template parameter
- `set<T>` - the item stored act as key, no duplicates
- `multiset<T>` - set allowing duplicate items
- `map<K,V>` - separate key and value, no duplicates
- `multimap<K,V>` - map allowing duplicate keys
- hashed associative containers *may* be available
  - Dinkumware and SGI did things differently though

Container Adaptors

- There are a few classes acting as wrappers around other containers, adapting them to a specific interface
  - `stack` - ordinary LIFO
  - `queue` - single-ended FIFO
  - `priority_queue` - the sorting criterion can be specified
  - Programmers can specify the underlying data type
  - usually a `deque`

Tip : vector<bool>

- Meyers: "As an STL container, there are really only two things wrong with `vector<bool>`. First it's not an STL containers. Second it doesn't hold `bool`s. Other than that, there's not much to object to." (Effective STL, p79)
  - `vector<bool>` does not conform to STL requirements
  - it stores `bool`s in a *packed* representation (e.g. bitfield)
  - Accessing it returns proxy objects to `bool`s, not true bools
  - Use a `deque<bool>` or a `bitset` to store `bools`
  - You won't get C compatibility (but C doesn't have `bool`s anyways)

Tip : size() and empty()

- You *may* check whether a container is empty by writing `c.size() == 0` or `c.empty()`
- However, with lists, which have a `splice()` function, if `splice()` is O(1), `size()` must be O(n) and conversely.
- Therefore, while `empty()` will always run in O(1), `size()` may not. You should thus prefer calling `empty()` to checking `size()` against zero.

Template

- Template เป็นพื้นฐานสำหรับการเขียนโปรแกรมเพื่อใช้งานโดยไม่ยึดติดกับชนิดข้อมูล เช่นการเรียงลำดับข้อมูล จะเรียงตัวเลข หรือตัวอักษรก็สามารถทำได้ เรียกว่า generic programming
- Template เป็นเหมือนแม่แบบสำหรับสร้าง generic class หรือ generic function

ตัวอย่าง container

- vector เราสามารถกำหนดชนิดข้อมูลที่จะสร้างเป็นเวกเตอร์ได้
- list เราสามารถสร้างลิสต์ของชนิดข้อมูลใดก็ได้

definition	meaning
vector<int>	เวกเตอร์จำนวนเต็ม
vector<String>	เวกเตอร์เก็บข้อความ

definition	meaning
list<int>	ลิสต์จำนวนเต็ม
list<float>	ลิสต์จำนวนจริง

Example of the vector container

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    // create a vector to store int
    vector<int> vec;
    int i;
    // display the original size of vec
    cout << "vector size = " << vec.size() << endl;
    // push 5 values into the vector
    for(i = 0; i < 5; i++){
        vec.push_back(i);
    }
    // display extended size of vec
    cout << "extended vector size = " << vec.size() << endl;
```

```
    // access 5 values from the vector
    for(i = 0; i < 5; i++){
        cout << "value of vec [" << i << "] = " << vec[i]
<< endl;
    }
    // use iterator to access the values
    vector<int>::iterator v = vec.begin();
    while( v != vec.end()) {
        cout << "value of v = " << *v << endl;
        v++;
    }
    return 0;
}
```

Example of constructing lists

```
#include <iostream>
#include <list>
void main (){
  // constructors used in the same order as described above:
  std::list<int> first;                                // empty list of ints
  std::list<int> second (4,100);                       // four ints with value 100
  std::list<int> third (second.begin(),second.end());  // iterating through second
  std::list<int> fourth (third);                       // a copy of third
  // the iterator constructor can also be used to construct from arrays:
  int myints[] = {16,2,77,29};
  std::list<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );
  std::cout << "The contents of fifth are: ";
  for (std::list<int>::iterator it = fifth.begin(); it != fifth.end(); it++)
    std::cout << *it << ' ';
  std::cout << '\n';}
```

ตัวอย่าง container

- vector เราสามารถกำหนดชนิดข้อมูลที่จะสร้างเป็นเวกเตอร์ได้
- list เราสามารถสร้างลิสต์ของชนิดข้อมูลใดก็ได้

definition	meaning
vector<int>	เวกเตอร์จำนวนเต็ม
vector<String>	เวกเตอร์เก็บข้อความ

definition	meaning
list<int>	ลิสต์จำนวนเต็ม
list<float>	ลิสต์จำนวนจริง

Function Template

สร้างฟังก์ชันเทมเพลตได้ด้วยรูปแบบนี้

```
template <class type> re-type func-name(parameter list){
  // body of function
}
```

เราจะต้องกำหนดชื่อใดๆ แทนคำว่า type สำหรับแทนชนิดข้อมูลที่เรา
จะอ้างในฟังก์ชัน

Example of function template

```
#include <iostream>
#include <string>
using namespace std;
template <typename T>
inline T const& Max (T const& a, T const& b)  {
   return a < b ? b:a;
}
int main () {
   int i = 39;
   int j = 20;
   cout << "Max(i, j): " << Max(i, j) << endl;
   string s1 = "Hello";
   string s2 = "World";
   cout << "Max(s1, s2): " << Max(s1, s2) << endl;
   return 0;}
```

Class Template

สร้างคลาสเทมเพลตได้ด้วยรูปแบบนี้

```
template <class type> class class-name{
  // body of class
}
```

เราจะต้องกำหนดชื่อใดๆ แทนคำว่า type สำหรับแทนชนิดข้อมูลที่เราจะอ้างในคลาส

Example of class Stack<>

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <string>
#include <stdexcept>
using namespace std;
template <class T>
class Stack {
   private:
      vector<T> elems;     // elements
```

```
public:
      void push(T const&);   // push element
      void pop();                    // pop element
      T top() const;                 // return top
element
      bool empty() const{            // return true if
empty.
         return elems.empty();
      }
};
```

Example of class Stack<> (cont’)

```
template <class T>
void Stack<T>::push (T const& elem) {
   // append copy of passed element
   elems.push_back(elem);
}

template <class T>
void Stack<T>::pop () {
   if (elems.empty()) {
      throw out_of_range("Stack<>::pop(): empty stack");
   }
   // remove last element
   elems.pop_back();
}
```

```
template <class T>
T Stack<T>::top () const {
   if (elems.empty()) {
      throw out_of_range("Stack<>::top(): empty
stack");
   }
   // return copy of last element
   return elems.back();
}
```

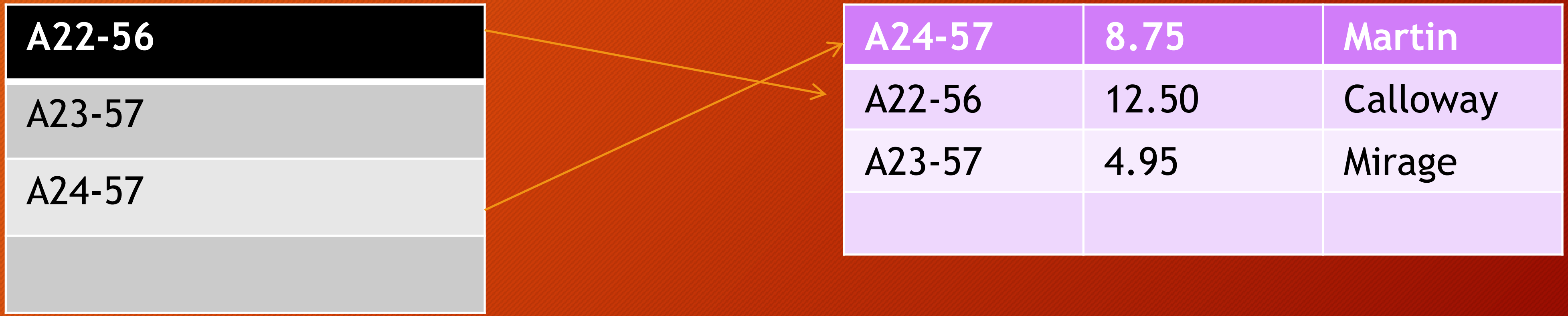
Example of class Stack<> (cont’)

```
int main() {
   try {
      Stack<int>    intStack;  // stack of ints
      Stack<string> stringStack;    // stack of strings
      // manipulate int stack
      intStack.push(7);
      cout << intStack.top() <<endl;
      // manipulate string stack
      stringStack.push("hello");
      cout << stringStack.top() << std::endl;
      stringStack.pop();
      stringStack.pop();
   }catch (exception const& ex) {
      cerr << "Exception: " << ex.what() <<endl;
      return -1; }
}
```

```
7
hello
Exception: Stack<>::pop(): empty stack
```

Map Container

- สร้างความสัมพันธ์ระหว่างคีย์หลักกับค่าข้อมูล
- เราสามารถใช้คีย์หลักในการหาค่าที่สัมพันธ์กับคีย์หลัก
- ตัวอย่าง คีย์หลักเป็นรหัสชิ้นส่วนอิเล็กทรอนิกส์ สัมพันธ์กับราคา 8.75 ดอลล่าร์ ผลิตโดยบริษัท Martin



A22-56
A23-57
A24-57

A24-57	8.75	Martin
A22-56	12.50	Calloway
A23-57	4.95	Mirage

STL map – Associative Arrays

- STL map class เป็นการกำหนดคีย์กับข้อมูล โดยคีย์เป็นข้อมูลที่ไม่ซ้ำกันเลย เช่น เลขบัตรประชาชนกับชื่อสกุล เป็นต้น
- STL map class สามารถใช้ชนิดข้อมูลใดๆ ก็ได้เป็นคีย์หรือเป็นข้อมูล
- 1 to 1 mapping (หนึ่งคีย์สัมพันธ์กับข้อมูลหนึ่งข้อมูล)
- การใช้ STL map class จะต้อง #include <map>

STL map – Associative Arrays

รูปแบบการกำหนดคือ

```
std::map <key_type, data_type, [comparison_function]>
```

หมายเหตุ comparison_function ถ้าไม่ระบุจะใช้ค่า default เป็น less <

ตัวอย่าง

```
map(char,int> m; // keys of type char, values of type int
// less than is already defined for char
```

STL Maps: Constructors

```
map<char,int> m;
map<char,int> m2(m);
```

(1) Standard constructor

```
    map<template,fields> mapName(const Comp &cmp = Comp(),
    const Allocator &a = Allocator());
```

Note that allocator, cmpfn are constructed automatically if not passed in

(2) Copy constructor

```
map<template,fields> mapName(const map<Key, T, Comp, Allocator> &anotherMap)
```

STL Maps: Data Storage

- An STL map is implemented as a tree-structure, where each node holds a “pair”
- Most important to know when retrieving data from the table
  - Some functions return the pair, not just the value
- A pair has two fields, *first* (holding the key) and *second* (holding the value)

STL Map: Data Storage

- If you have a ***pair object***, you can use the following code to print the key and value:

  ```
  cout << myPairObject.first << " " << myPairObject.second;
  ```

- If you have a ***pointer to the pair object***, use the arrow operator instead

  ```
  cout << myPairObject->first << " " << myPairObject->second;
  ```

STL Map: Data Storage

- Any time a function returns an **iterator**, the iterator is a pointer to the pair [so you'll use the -> operation most frequently].
- Tree structure
  - logarithmic time inserts, finds, deletes

Example of map class

```
#include <map>
int main() {
  std::map <string, char> grade_list;
  grade_list["John"] = 'A';
  // Should be John
  std::cout<<grade_list.begin()->first<<endl;
  // Should be A
  std::cout<<grade_list.begin()->second<<endl;
  if(grade_list.find("Tim") == grade_list.end()){
    std::cout<<"Tim is not in the map!"<<endl;
  }
}
```

STL Map: Available Methods

Methods	description
void clear()	remove all elements
bool empty()	Returns true if empty, false otherwise
size_type max_size()	returns max number of elements map can hold (usually an integer returned) [capacity]
size_type size()	return the number of elements currently in the map (usually an integer returned) [actual size]
iterator begin()	returns an iterator to the first element in the map (the first when sorted, due to storage mechanism)
iterator end()	returns an iterator to the last element in the map (the last when sorted)
reverse_iterator rbegin()	returns a reverse iterator to the end of the map
reverse_iterator rend()	returns a reverse iterator to the start of the map

STL Map: Available Methods

pair<iterator,bool> insert(const value_type &val)

Insert val into the map, if it’s not already there. Return pair<iterator,true> if successful, pair<iterator,false> if fails.

iterator insert(iterator i, const value_type &val)

Insert val into the map, after the value specified by i. Iterator to inserted element is returned.

template <class InIter> void insert(InIter start, InIter end)

Insert a range of elements

STL Map: Available Methods

void erase (iterator i)

Remove the element pointed to by i.

size_type erase(const key_type & k)

Remove from the map elements that have keys with the value k.

void erase(iterator start, iterator end)

Remove the elements in the range start to end

STL Map: Available Methods

iterator find(const key_type &k)

Returns an iterator to the specified key. If the key is not found, an iterator to the end of the map is returned.

size_type count(const key_type &k) const

Returns the number of times a key k occurs in the map (0 or 1)

reference operator[](const key_type &k)

Returns a reference to the value associated with the key k.

If the key is not found in the map, the key and a default constructed instance of the value type is inserted in the amp.

STL Map: Available Methods

```
iterator lower_bound(const key_type &k)
```

Returns an iterator to the first element in the map with a key >= k

```
iterator upper_bound(const key_type &k) const
```

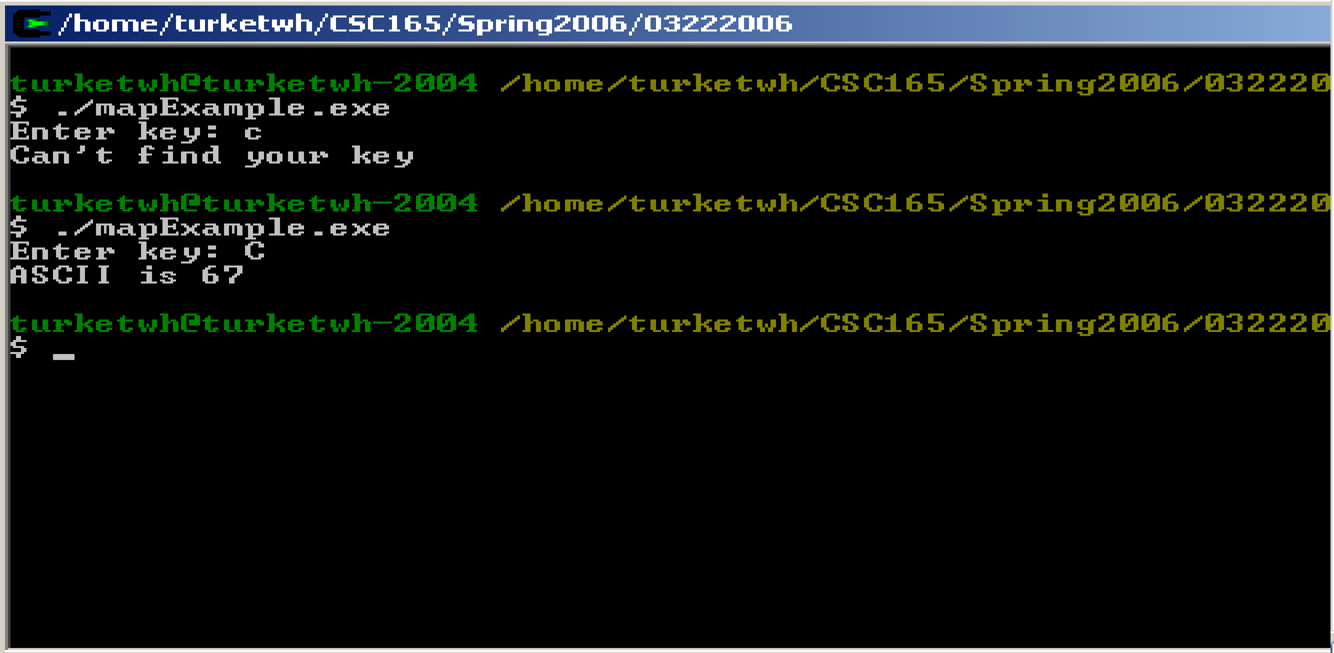
Returns an iterator to the first element in the map with a key strictly > k

```
pair<iterator, iterator> equal_range(const key_type &k)
```

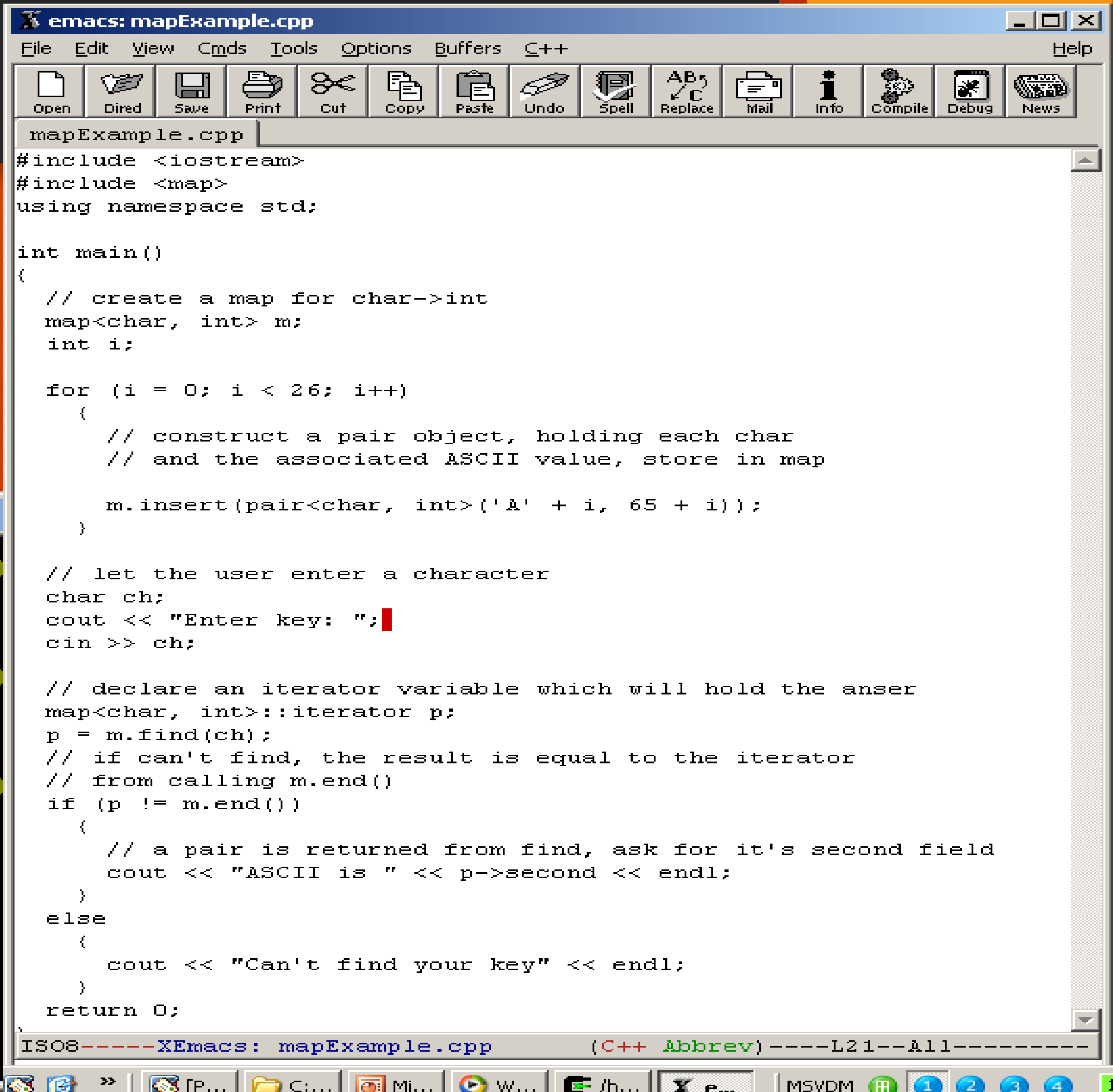
Returns a pair of iterators that point to the upper bound and the lower bound in the map for the specified key

STL Map: Example Programs

Map between characters and ASCII representations



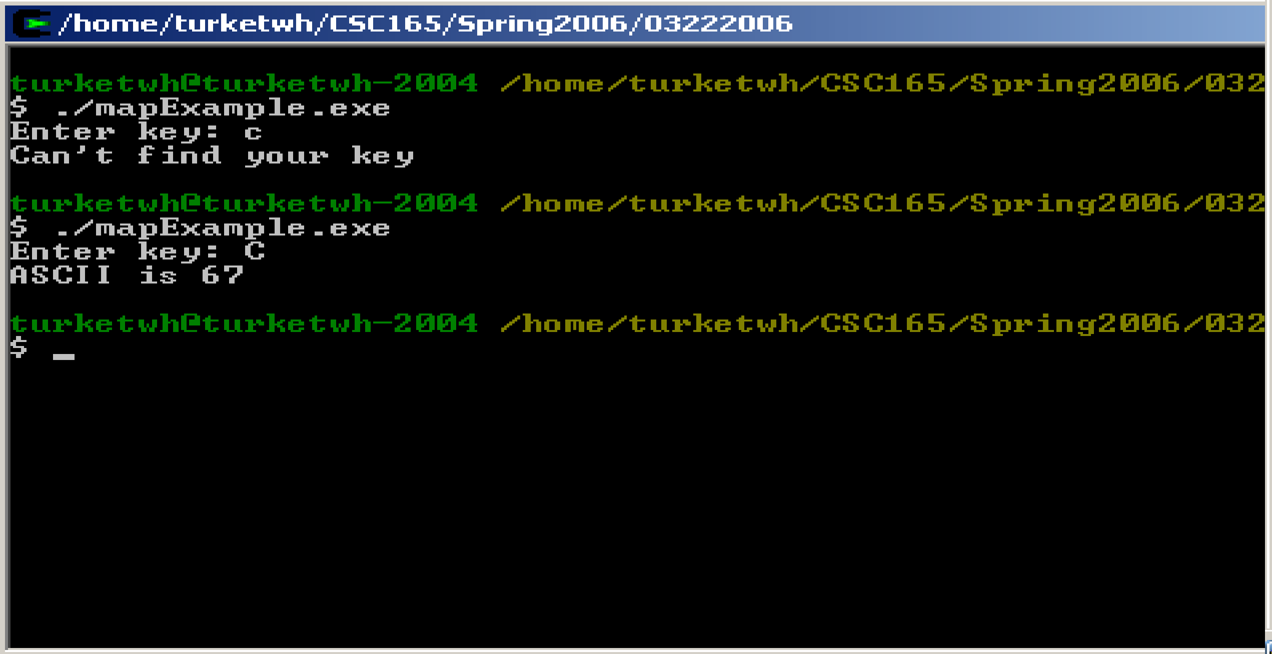
```
turketwh@turketwh-2004 /home/turketwh/CSC165/Spring2006/032220
$ ./mapExample.exe
Enter key: c
Can't find your key

turketwh@turketwh-2004 /home/turketwh/CSC165/Spring2006/032220
$ ./mapExample.exe
Enter key: C
ASCII is 67

turketwh@turketwh-2004 /home/turketwh/CSC165/Spring2006/032220
$ _
```


emacs: mapExample.cpp
File Edit View Cmds Tools Options Buffers C++ Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News
mapExample.cpp
ISO8-----XEmacs: mapExample.cpp (C++ Abbrev)----L21--All---------

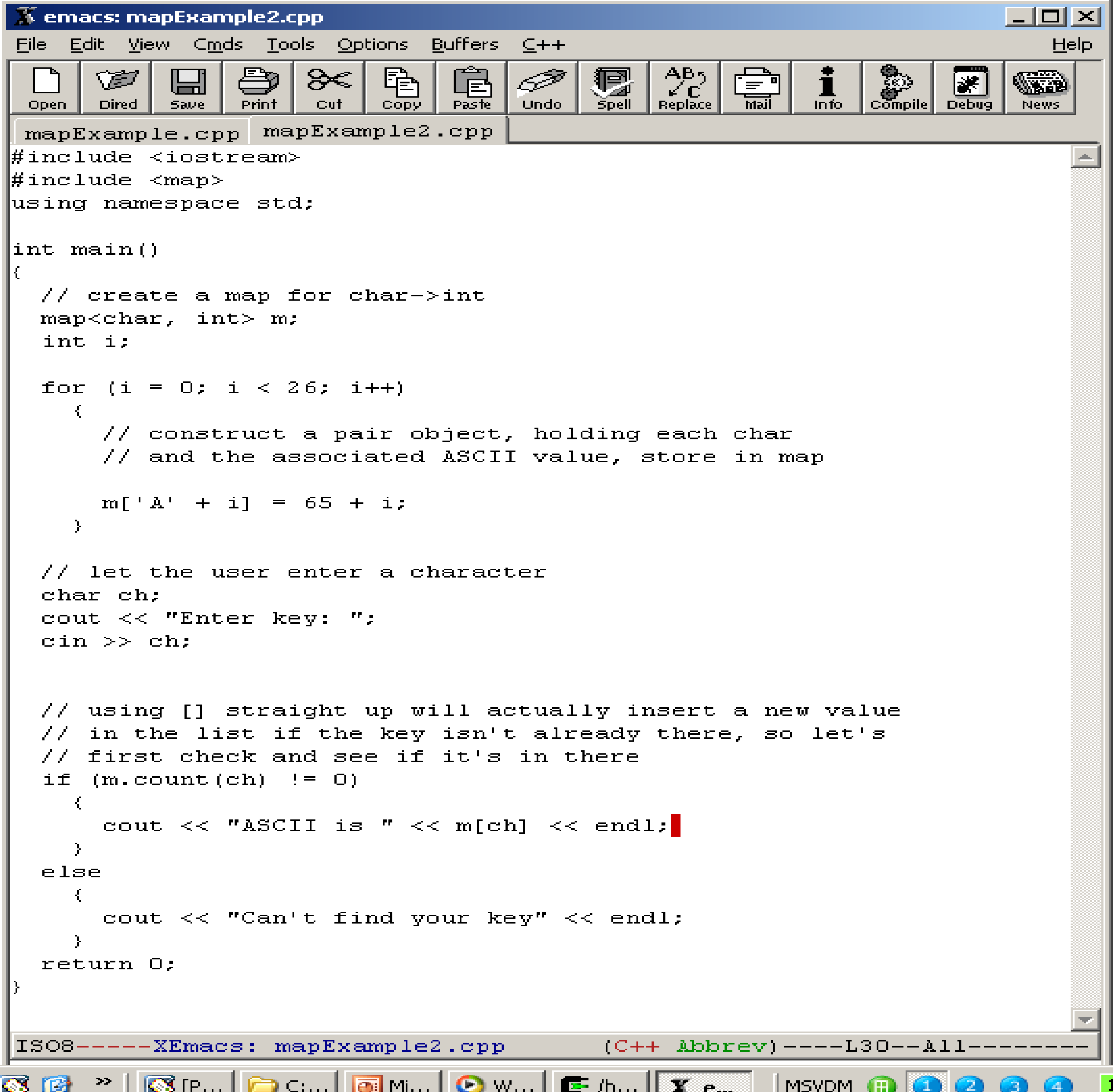
```
#include <iostream>
#include <map>
using namespace std;

int main()
{
  // create a map for char->int
  map<char, int> m;
  int i;

  for (i = 0; i < 26; i++)
    {
      // construct a pair object, holding each char
      // and the associated ASCII value, store in map

      m.insert(pair<char, int>('A' + i, 65 + i));
    }

  // let the user enter a character
  char ch;
  cout << "Enter key: ";
  cin >> ch;

  // declare an iterator variable which will hold the anser
  map<char, int>::iterator p;
  p = m.find(ch);
  // if can't find, the result is equal to the iterator
  // from calling m.end()
  if (p != m.end())
    {
      // a pair is returned from find, ask for it's second field
      cout << "ASCII is " << p->second << endl;
    }
  else
    {
      cout << "Can't find your key" << endl;
    }
  return 0;
```

STL Map: Example Programs

Same program, exploiting []



```
turketwh@turketwh-2004 /home/turketwh/CSC165/Spring2006/032
$ ./mapExample.exe
Enter key: c
Can't find your key

turketwh@turketwh-2004 /home/turketwh/CSC165/Spring2006/032
$ ./mapExample.exe
Enter key: C
ASCII is 67

turketwh@turketwh-2004 /home/turketwh/CSC165/Spring2006/032
$ _
```

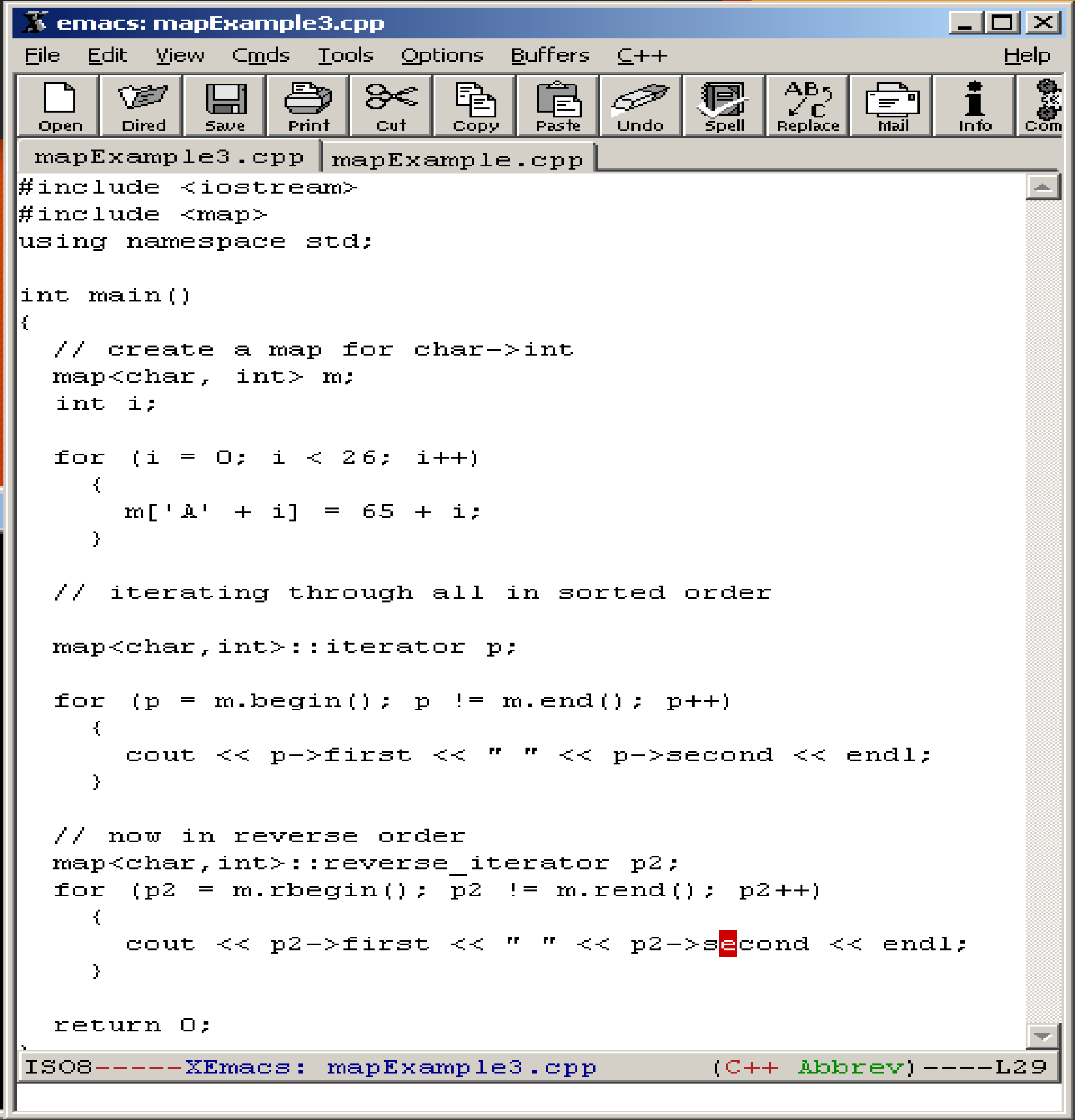


```cpp
#include <iostream>
#include <map>
using namespace std;

int main()
{
  // create a map for char->int
  map<char, int> m;
  int i;

  for (i = 0; i < 26; i++)
    {
      // construct a pair object, holding each char
      // and the associated ASCII value, store in map

      m['A' + i] = 65 + i;
    }

  // let the user enter a character
  char ch;
  cout << "Enter key: ";
  cin >> ch;


  // using [] straight up will actually insert a new value
  // in the list if the key isn't already there, so let's
  // first check and see if it's in there
  if (m.count(ch) != 0)
    {
      cout << "ASCII is " << m[ch] << endl;
    }
  else
    {
      cout << "Can't find your key" << endl;
    }
  return 0;
}
```

```
ISO8-----XEmacs: mapExample2.cpp       (C++ Abbrev)----L30--All--------
```

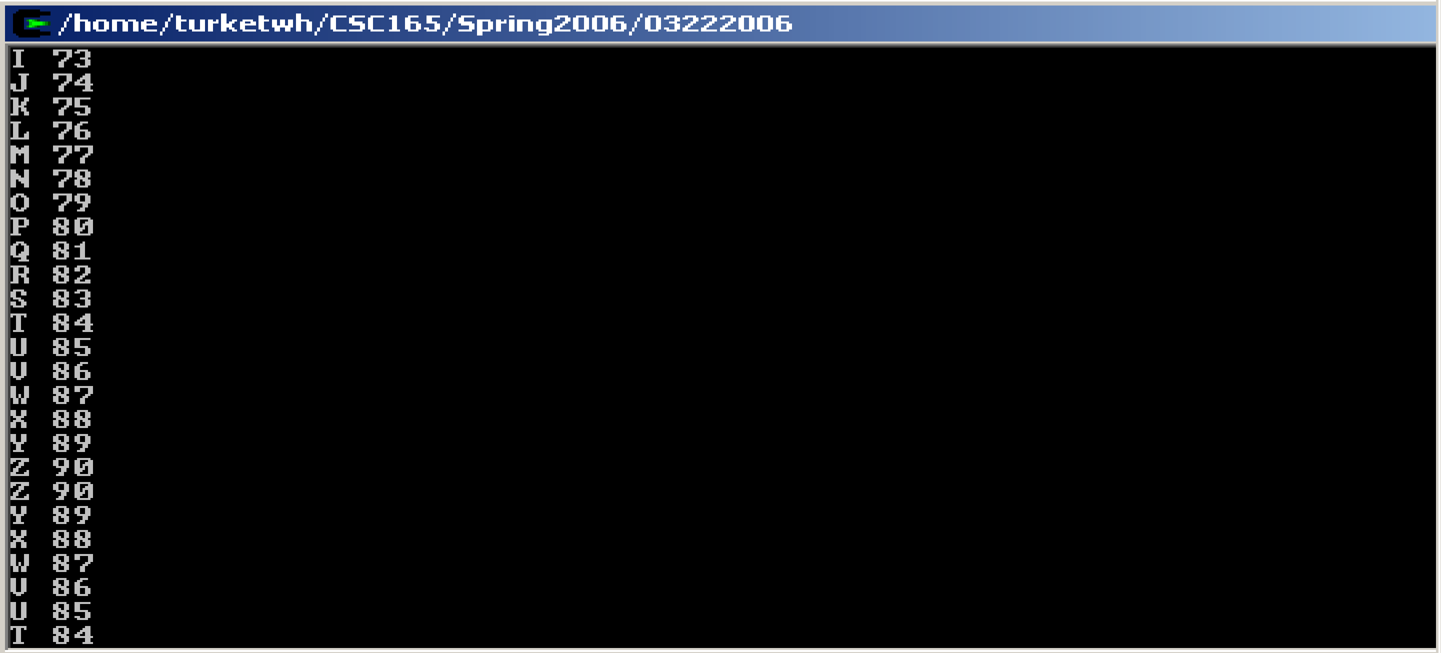
STL Map: Example Programs

Print all entries in map in forward and reverse order



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

int main()
{
  // create a map for char->int
  map<char, int> m;
  int i;

  for (i = 0; i < 26; i++)
    {
      m['A' + i] = 65 + i;
    }

  // iterating through all in sorted order

  map<char,int>::iterator p;

  for (p = m.begin(); p != m.end(); p++)
    {
      cout << p->first << " " << p->second << endl;
    }

  // now in reverse order
  map<char,int>::reverse_iterator p2;
  for (p2 = m.rbegin(); p2 != m.rend(); p2++)
    {
      cout << p2->first << " " << p2->second << endl;
    }

  return 0;
```

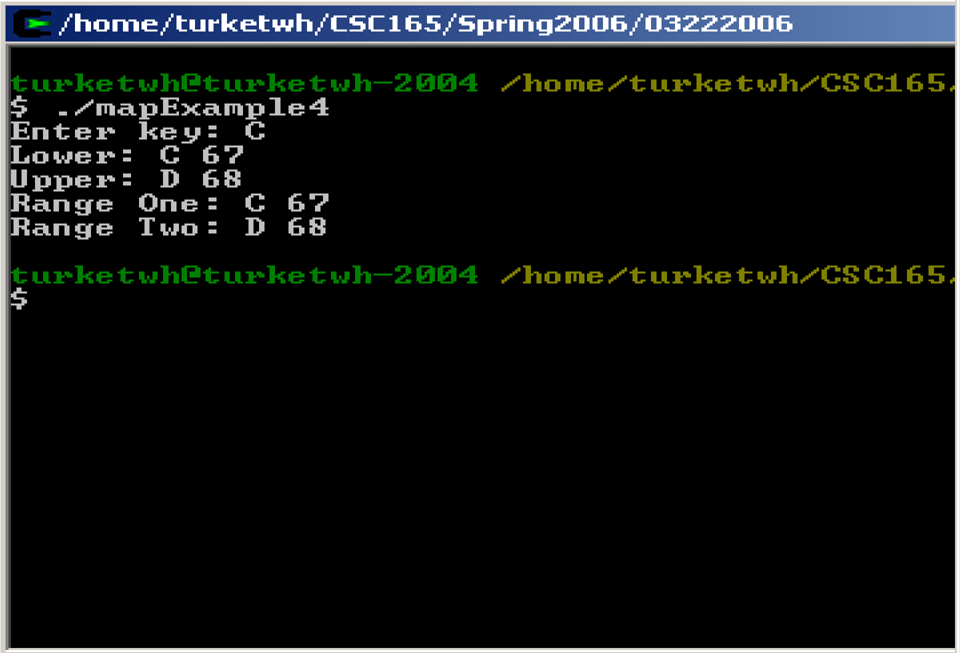
```
ISO8-----XEmacs: mapExample3.cpp      (C++ Abbrev)----L29
```



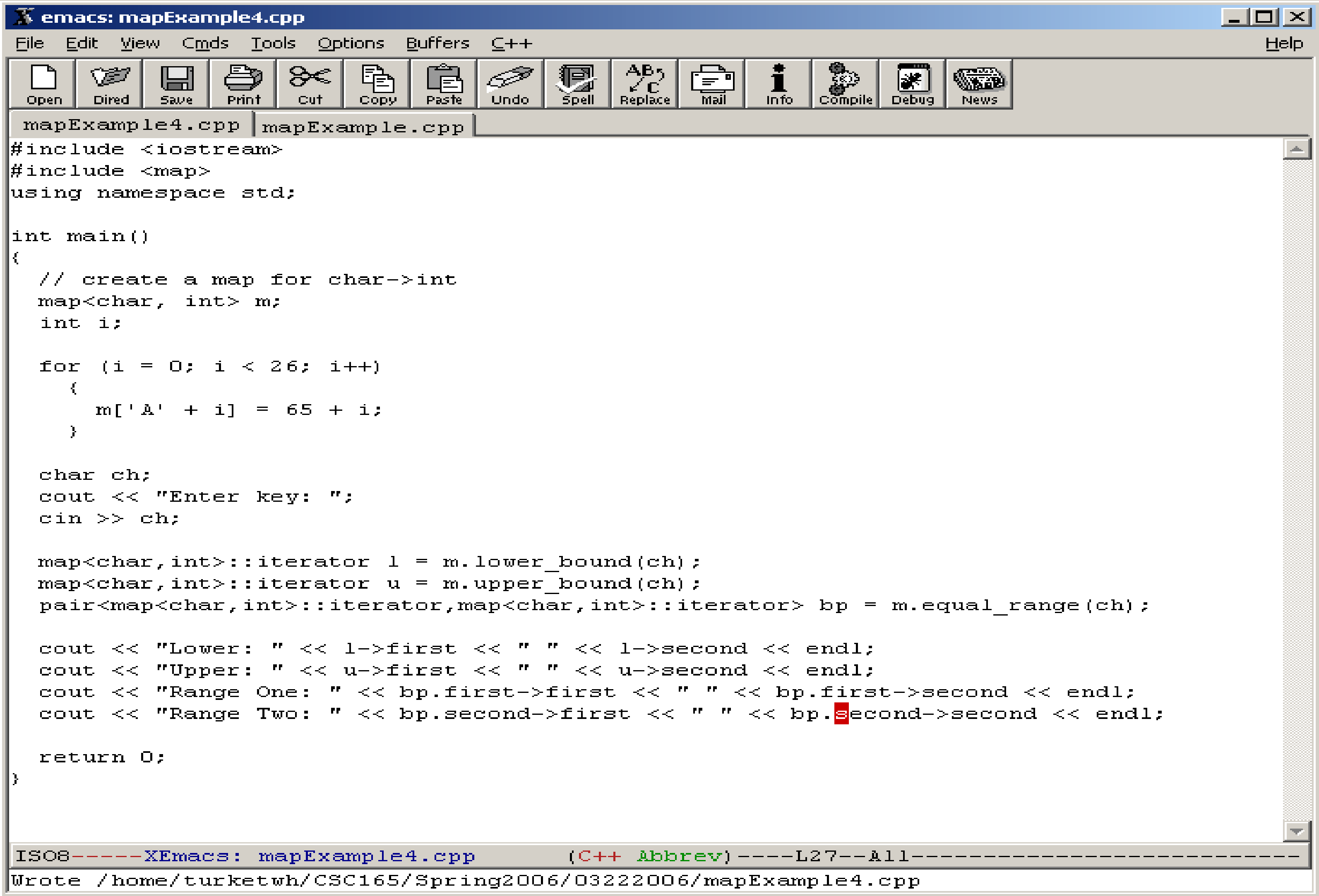
```
I 73
J 74
K 75
L 76
M 77
N 78
O 79
P 80
Q 81
R 82
S 83
T 84
U 85
V 86
W 87
X 88
Y 89
Z 90
Z 90
Y 89
X 88
W 87
V 86
U 85
T 84
```

STL Map: Example Programs

Print lower bound, upper bound, and equal range given a char typed in



```
turketwh@turketwh-2004 /home/turketwh/CSC165/
$ ./mapExample4
Enter key: C
Lower: C 67
Upper: D 68
Range One: C 67
Range Two: D 68

turketwh@turketwh-2004 /home/turketwh/CSC165/
$
```

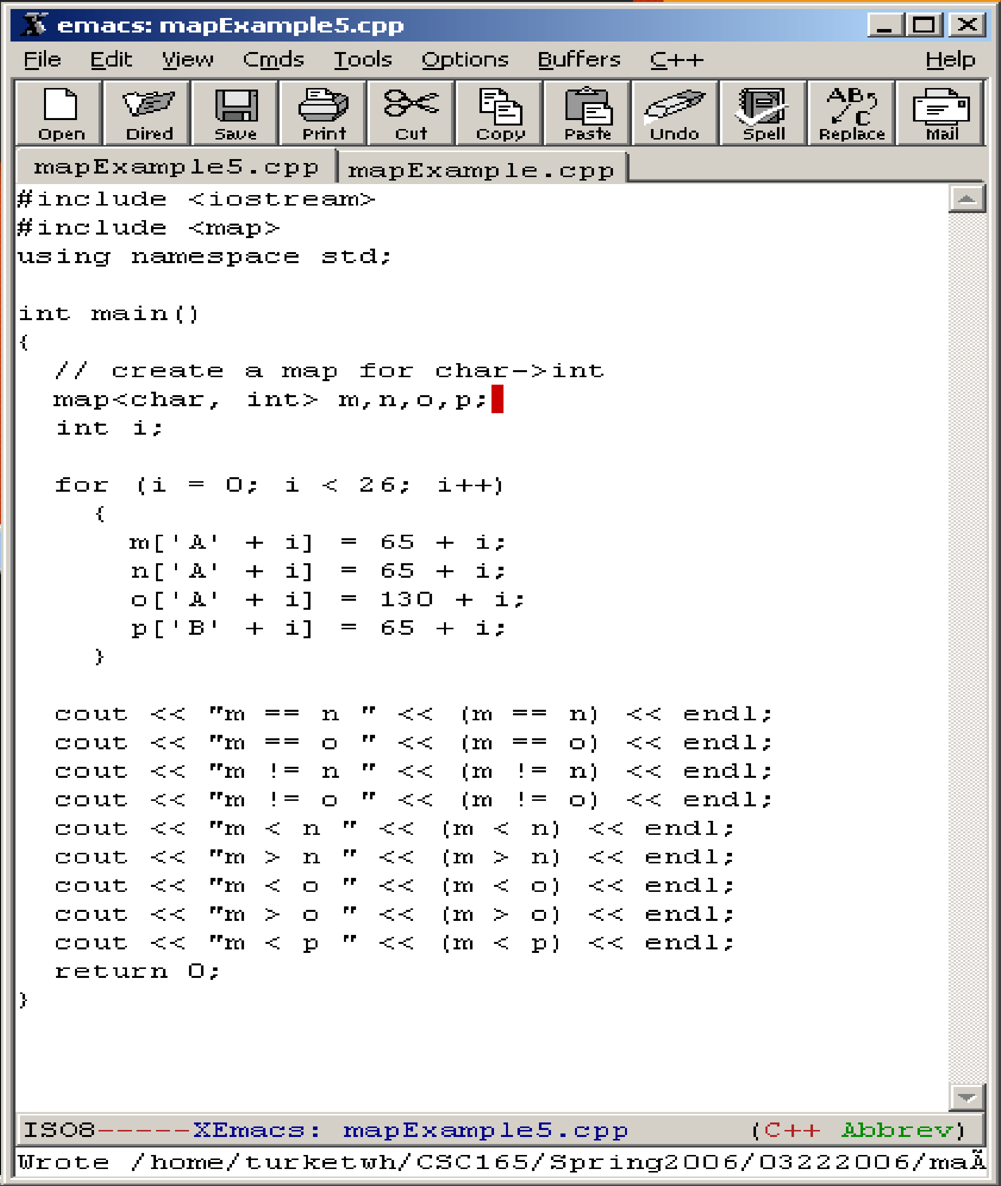


```
#include <iostream>
#include <map>
using namespace std;

int main()
{
  // create a map for char->int
  map<char, int> m;
  int i;

  for (i = 0; i < 26; i++)
    {
      m['A' + i] = 65 + i;
    }

  char ch;
  cout << "Enter key: ";
  cin >> ch;

  map<char,int>::iterator l = m.lower_bound(ch);
  map<char,int>::iterator u = m.upper_bound(ch);
  pair<map<char,int>::iterator,map<char,int>::iterator> bp = m.equal_range(ch);

  cout << "Lower: " << l->first << " " << l->second << endl;
  cout << "Upper: " << u->first << " " << u->second << endl;
  cout << "Range One: " << bp.first->first << " " << bp.first->second << endl;
  cout << "Range Two: " << bp.second->first << " " << bp.second->second << endl;

  return 0;
}
```

Wrote /home/turketwh/CSC165/Spring2006/03222006/mapExample4.cpp

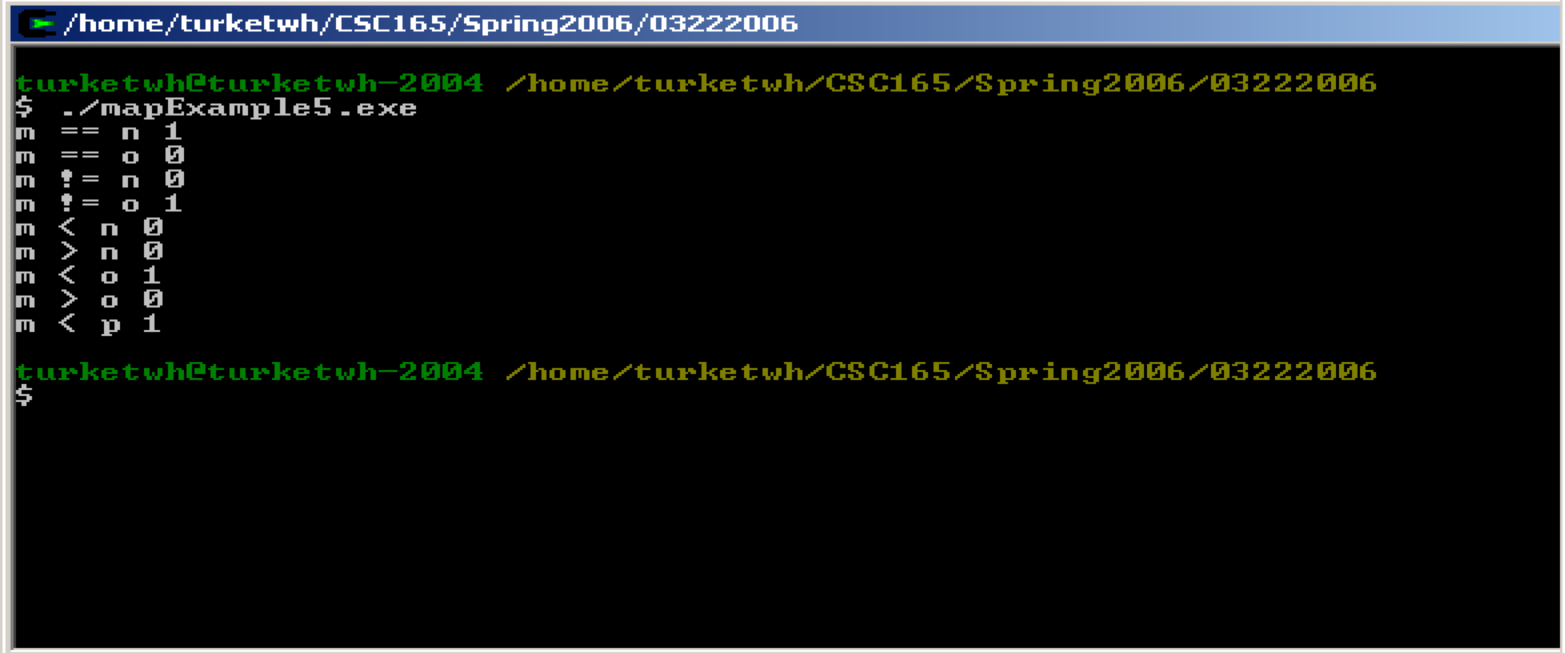
STL Map: Example Programs

Using the <,>, ==, != functions on map objects



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

int main()
{
  // create a map for char->int
  map<char, int> m,n,o,p;
  int i;

  for (i = 0; i < 26; i++)
    {
      m['A' + i] = 65 + i;
      n['A' + i] = 65 + i;
      o['A' + i] = 130 + i;
      p['B' + i] = 65 + i;
    }

  cout << "m == n " << (m == n) << endl;
  cout << "m == o " << (m == o) << endl;
  cout << "m != n " << (m != n) << endl;
  cout << "m != o " << (m != o) << endl;
  cout << "m < n " << (m < n) << endl;
  cout << "m > n " << (m > n) << endl;
  cout << "m < o " << (m < o) << endl;
  cout << "m > o " << (m > o) << endl;
  cout << "m < p " << (m < p) << endl;
  return 0;
}
```

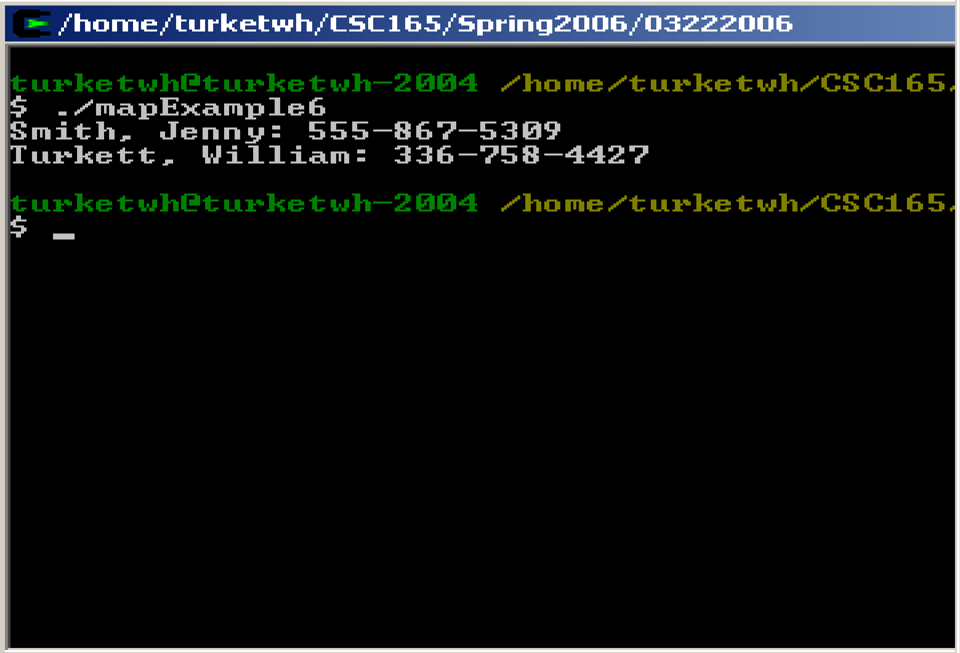
```
ISO8-----XEmacs: mapExample5.cpp          (C++ Abbrev)
Wrote /home/turketwh/CSC165/Spring2006/03222006/maÅ
```



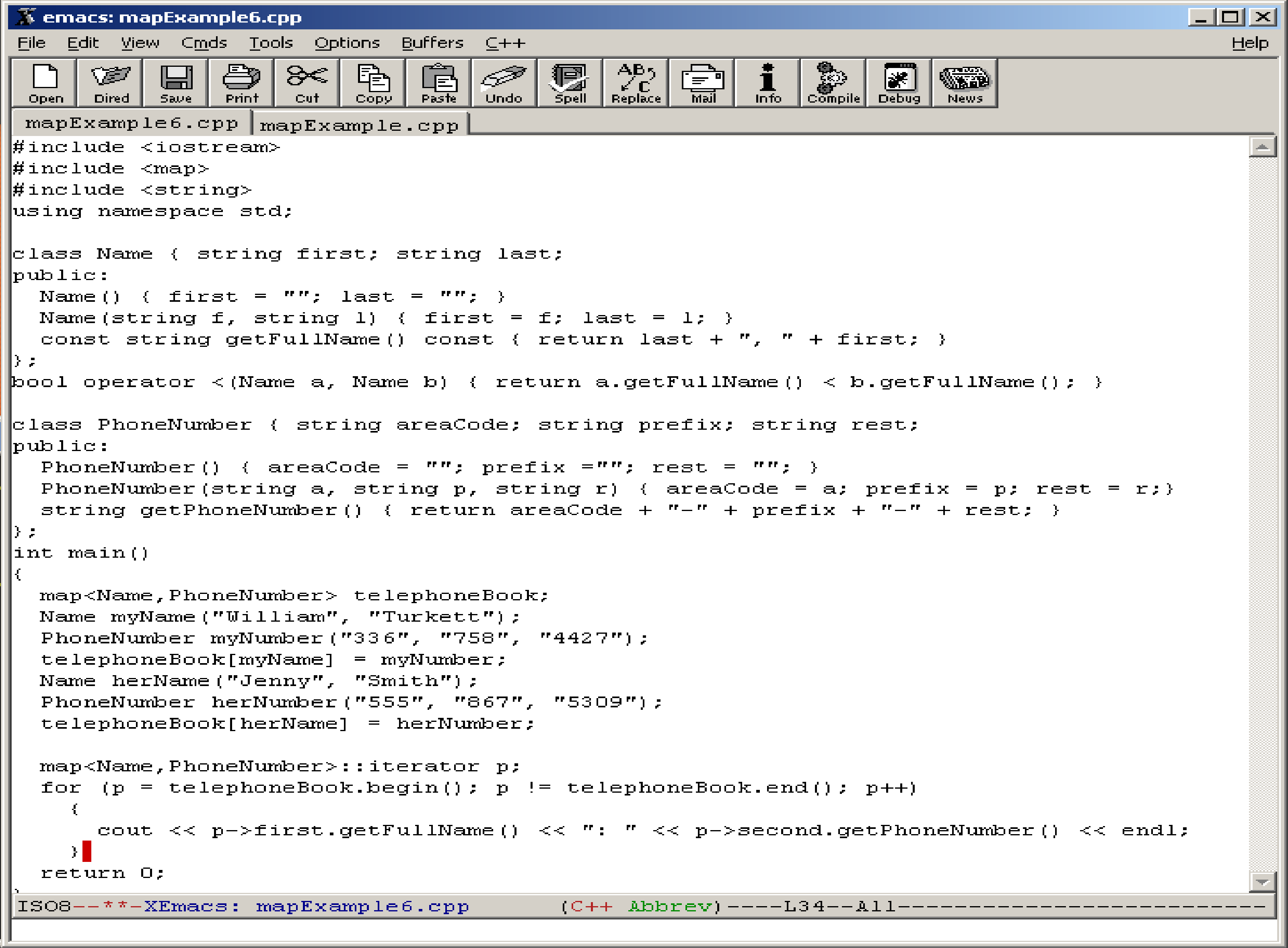
```
turketwh@turketwh-2004 /home/turketwh/CSC165/Spring2006/03222006
$ ./mapExample5.exe
m == n 1
m == o 0
m != n 0
m != o 1
m < n 0
m > n 0
m < o 1
m > o 0
m < p 1

turketwh@turketwh-2004 /home/turketwh/CSC165/Spring2006/03222006
$
```

STL Map: Example Programs

Storing objects, requiring an overload of the < operator for the key type



```
turketwh@turketwh-2004 /home/turketwh/CSC165/
$ ./mapExample6
Smith, Jenny: 555-867-5309
Turkett, William: 336-758-4427

turketwh@turketwh-2004 /home/turketwh/CSC165/
$ _
```



```cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;

class Name { string first; string last;
public:
  Name() { first = ""; last = ""; }
  Name(string f, string l) { first = f; last = l; }
  const string getFullName() const { return last + ", " + first; }
};
bool operator <(Name a, Name b) { return a.getFullName() < b.getFullName(); }

class PhoneNumber { string areaCode; string prefix; string rest;
public:
  PhoneNumber() { areaCode = ""; prefix =""; rest = ""; }
  PhoneNumber(string a, string p, string r) { areaCode = a; prefix = p; rest = r;}
  string getPhoneNumber() { return areaCode + "-" + prefix + "-" + rest; }
};
int main()
{
  map<Name,PhoneNumber> telephoneBook;
  Name myName("William", "Turkett");
  PhoneNumber myNumber("336", "758", "4427");
  telephoneBook[myName] = myNumber;
  Name herName("Jenny", "Smith");
  PhoneNumber herNumber("555", "867", "5309");
  telephoneBook[herName] = herNumber;

  map<Name,PhoneNumber>::iterator p;
  for (p = telephoneBook.begin(); p != telephoneBook.end(); p++)
    {
      cout << p->first.getFullName() << ": " << p->second.getPhoneNumber() << endl;
    }
  return 0;
```

```
ISO8--**-XEmacs: mapExample6.cpp      (C++ Abbrev)----L34--All--------------------------
```

Multi-map Container

- Similar to a map container
- Multi-map container allows duplicates
- Sorted Associative Container
- Multiple Associative Container, there is no limit on the number of elements with the same key.

Example inserting

```
int main () {
  std::multimap<char,int> mymultimap;
  std::multimap<char,int>::iterator it;
  // first insert function version (single parameter):
  mymultimap.insert ( std::pair<char,int>('a',100) );
  mymultimap.insert ( std::pair<char,int>('z',150) );
  it=mymultimap.insert ( std::pair<char,int>('b',75) );
  // second insert function version (with hint position):
  mymultimap.insert (it, std::pair<char,int>('c',300)); //max efficiency
  mymultimap.insert (it, std::pair<char,int>('z',400)); //no max efficiency
```

```
mymultimap contains:
a => 100
b => 75
c => 300
z => 400
z => 150
anothermultimap contains:
a => 100
b => 75
```

Example inserting con’t

```
 // third insert function version (range insertion):
  std::multimap<char,int> anothermultimap;
  anothermultimap.insert(mymultimap.begin(),mymultimap.find('c'));
  // showing contents:
  std::cout << "mymultimap contains:\n";
  for (it=mymultimap.begin(); it!=mymultimap.end(); ++it)
    std::cout << (*it).first << " => " << (*it).second << '\n';
  std::cout << "anothermultimap contains:\n";
  for (it=anothermultimap.begin(); it!=anothermultimap.end(); ++it)
    std::cout << (*it).first << " => " << (*it).second << '\n';
  return 0;
}
```

Example finding

```
int main () { std::multimap<char,int> mymm;
mymm.insert (std::make_pair('x',10));
mymm.insert (std::make_pair('y',20));
mymm.insert (std::make_pair('z',30));
mymm.insert (std::make_pair('z',40));
std::multimap<char,int>::iterator it = mymm.find('x');
mymm.erase (it);
mymm.erase (mymm.find('z'));
• // print content:
• std::cout << "elements in mymm:" << '\n';
• std::cout << "y => " << mymm.find('y')->second << '\n';
• std::cout << "z => " << mymm.find('z')->second << '\n';
• return 0; }
```

```
Elements in mymm:
y => 20
z => 40
```

Example erasing

```
int main (){
  std::multimap<char,int> mymultimap;
  // insert some values:
  mymultimap.insert(std::pair<char,int>('a',10));
  mymultimap.insert(std::pair<char,int>('b',20));
  mymultimap.insert(std::pair<char,int>('b',30));
  mymultimap.insert(std::pair<char,int>('c',40));
  mymultimap.insert(std::pair<char,int>('d',50));
  mymultimap.insert(std::pair<char,int>('d',60));
  mymultimap.insert(std::pair<char,int>('e',70));
  mymultimap.insert(std::pair<char,int>('f',80));

  std::multimap<char,int>::iterator it = mymultimap.find('b')
```

```
a => 10
b => 30
c => 40
```

Example erasing con’t

```
  std::multimap<char,int>::iterator it = mymultimap.find('b');
  mymultimap.erase (it);     // erasing by iterator (1 element)
  mymultimap.erase ('d');    // erasing by key (2 elements)
  it=mymultimap.find ('e');
  mymultimap.erase ( it, mymultimap.end() ); // erasing by range
  // show content:
  for (it=mymultimap.begin(); it!=mymultimap.end(); ++it)
    std::cout << (*it).first << " => " << (*it).second << '\n';
  return 0;
}
```

Question