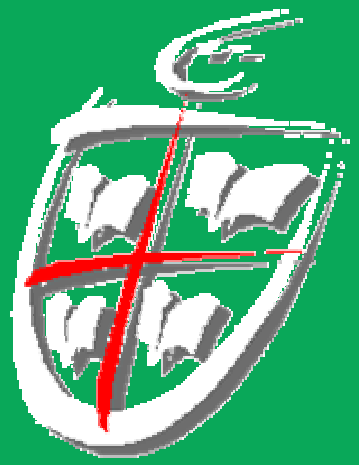


CS900-3a
Computer Science –
A Multifaceted Introduction

Lecture #12
A Computer Learns English

File I/O ñ What is a file?

- Files come in a variety of flavors (binary and ascii).
- We will focus on text files.
- A text file is just a sequence of ASCII encoded characters

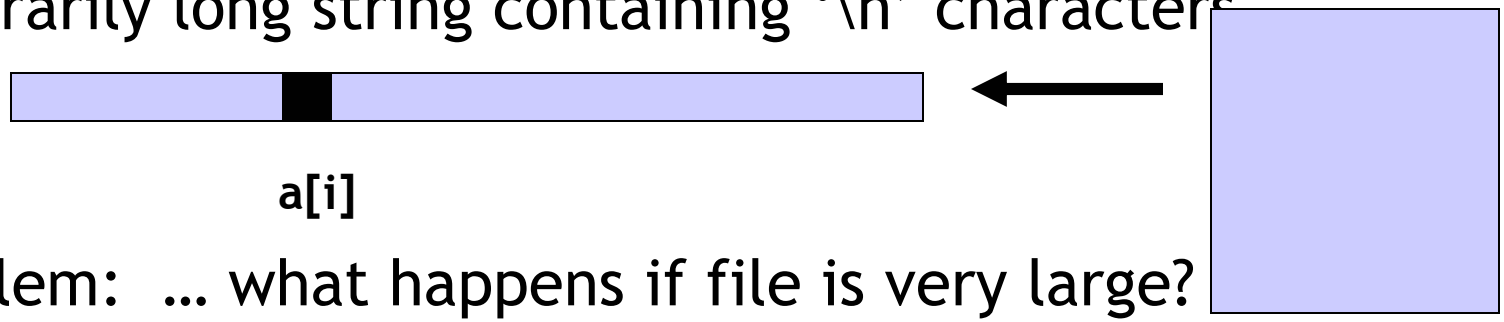
In the beginning God created
the heaven and the earth.

…

- How does a program access this data?

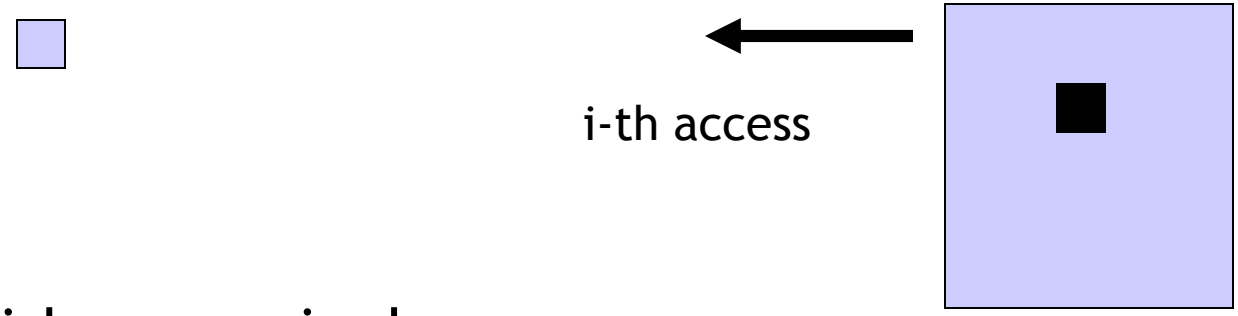
File I/O – What is a file?

- Tentative solution 1: represent a file using an arbitrarily long string containing '\n' characters



- Problem: … what happens if file is very large?

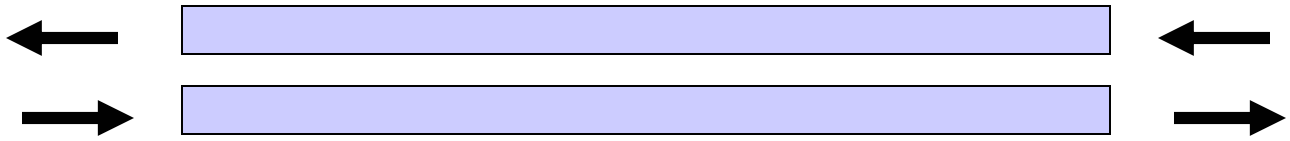
- Tentative solution 2: access the characters of the file one at a time



- Problem: … disk access is slow

File I/O - What is a file?

- Solution: streams



  - Maintain a buffer of some size N
  - Fill input stream buffers only when necessary (empty)
  - Write output buffers only when necessary (full)

- Sound complicated …? Don't worry.
- C++ supports i/o via streams

cout and cin

- Streams can also be used to write to the screen and to read from the keyboard. This replaces (to some extend) the C functions printf and scanf
- The most important operator is “<<“ (for output streams) and “>>” (for input streams), respectively

```
#include <iostream>
#include <string>
int main(){
   string name = “Dr. No”;
   int num;
   cout << “Enter a number “ << name << “:” << endl;
   cin >> num;
   cout << “The number is “ << num << endl;
   return 0;
}
```

Streams to/from Files

- Streams can be used to read/write data from/to files
- A file stream needs to be opened before use and closed afterwards

```
#include <fstream>
#include <string>
int main() {
   string token;
   ifstream in("file.txt"); //-- opens file.txt
   in >> token;             //-- reads token from file
   in.close();              //-- closes file
   cout << token << endl;
   return 0;
}
```

Streams to/from Files

- One can also explicitly open a stream using the open method with a CString

```
#include <fstream>

ifstream in;              //-- empty constructor
in.open("file.txt");      //-- open file stream
//...
in.close();               //-- closes file

string fname = "file.txt";
ofstream out;
out.open(fname.c_str()); //-- need to convert C++ string
                         //-- to a C str using .c_str()
//...
out.close();
```

Streams to/from Files

- It is sometimes useful to check whether a file access was successful
- Example:

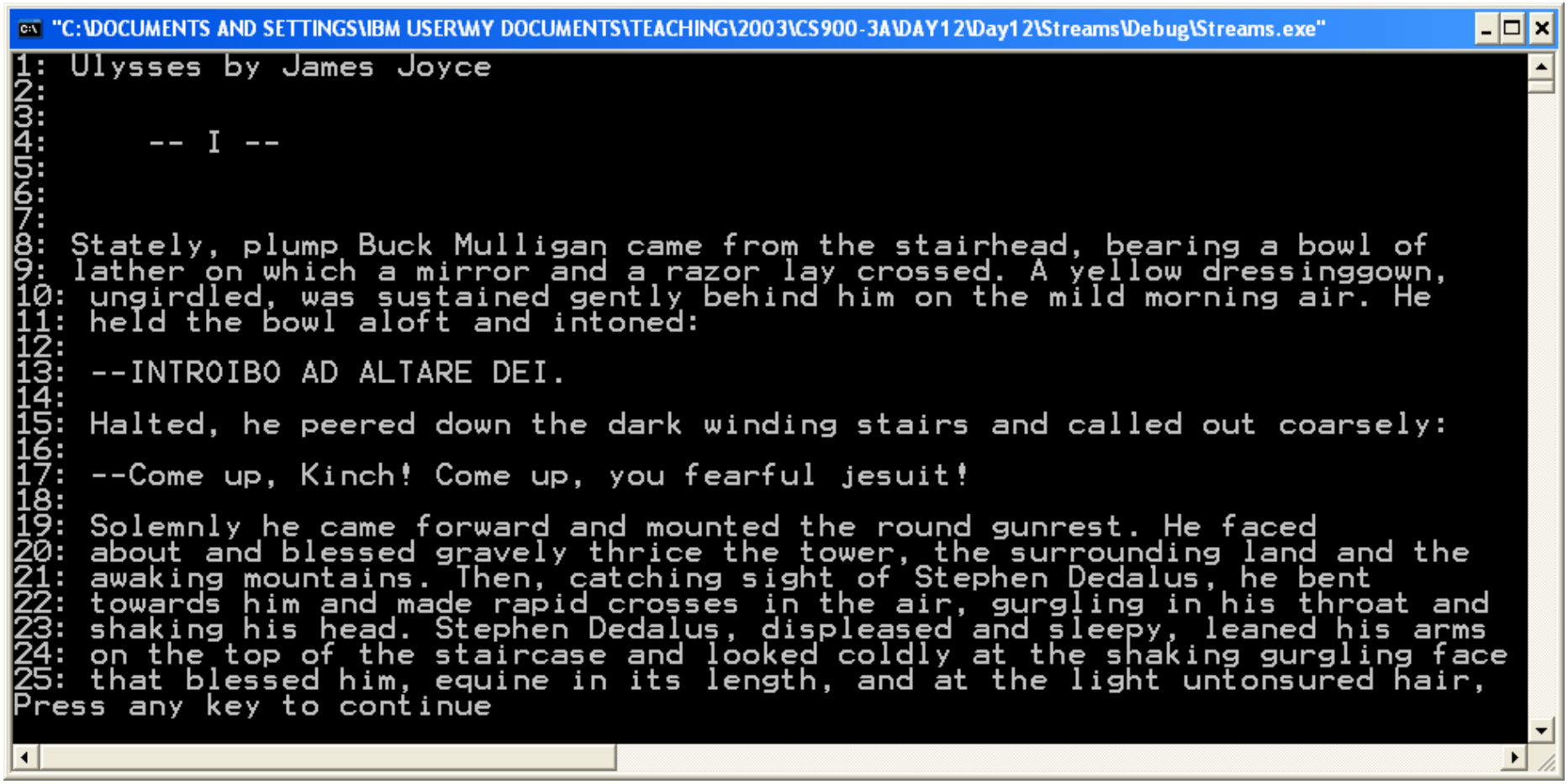
```
#include <fstream>
#include <iostream>
#include <string>
int main() {
   string token;
   ifstream in("file.txt");
   if (in.fail()) {
     cout << "file not found" << endl;
   }
[...]
}
```

Reading in Line by Line

- It is possible to read a file line by line, example:

```
#include <fstream>
#include <iostream>
#include <string>
int main() {
   int count = 1;
   string line;
   ifstream in("file.txt");
   while (1) {
      getline(in,line);
      if (in.fail()) break;
      cout << line << endl;
      count++;
   }
}
```

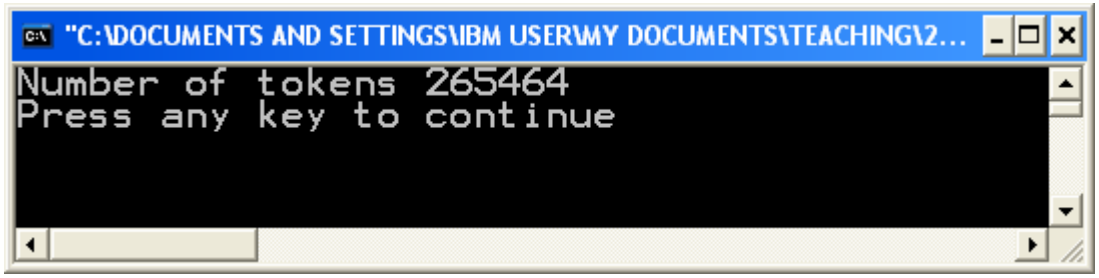
Example Run


"C:\DOCUMENTS AND SETTINGS\IBM USER\MY DOCUMENTS\TEACHING\2003\CS900-3A\DAY12\Day12\Streams\Debug\Streams.exe"
1: Ulysses by James Joyce
2:
3:
4: -- I --
5:
6:
7:
8: Stately, plump Buck Mulligan came from the stairhead, bearing a bowl of
9: lather on which a mirror and a razor lay crossed. A yellow dressinggown,
10: ungirdled, was sustained gently behind him on the mild morning air. He
11: held the bowl aloft and intoned:
12:
13: --INTROIBO AD ALTARE DEI.
14:
15: Halted, he peered down the dark winding stairs and called out coarsely:
16:
17: --Come up, Kinch! Come up, you fearful jesuit!
18:
19: Solemnly he came forward and mounted the round gunrest. He faced
20: about and blessed gravely thrice the tower, the surrounding land and the
21: awaking mountains. Then, catching sight of Stephen Dedalus, he bent
22: towards him and made rapid crosses in the air, gurgling in his throat and
23: shaking his head. Stephen Dedalus, displeased and sleepy, leaned his arms
24: on the top of the staircase and looked coldly at the shaking gurgling face
25: that blessed him, equine in its length, and at the light untonsured hair,
Press any key to continue

Counting the Number of Tokens

- A simple program to count the number of tokens

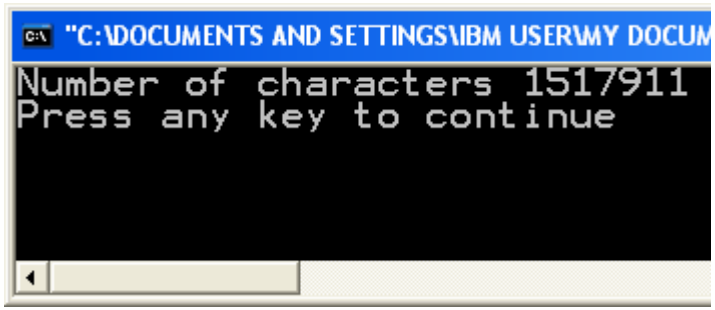
```
#include <fstream>
#include <iostream>
#include <string>
int main() {
   int count = 0;
   string token;
   ifstream in("ulyss.txt");
   do {
      in >> token;
      if (in.fail()) break;
      else           count++;
   } while(1);
   cout << "Number of tokens " << count << endl;
   return 0;
}
```



Counting the Number of Characters

- A simple program that reads one character at a

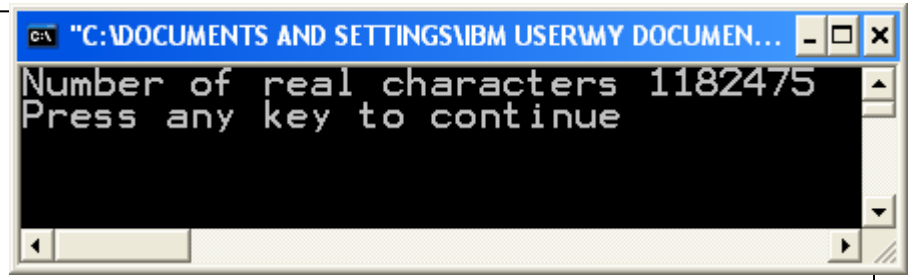
```
#include <fstream>
#include <iostream>
#include <string>
int main() {
   int count = 0; char ch;
   ifstream in("ulyss.txt");
   do {
      ch=in.get();
      if (ch==EOF) break;
      else         count++;
   } while (1);
   cout << "Number of characters " << count << endl;;
   return 0;
}
```



Counting the Number of Characters

- Counting “real” characters

```
int main() {
   int count = 0; char ch;
   ifstream in(“ulyss.txt”);
   do {
      ch=in.get();
      if (ch==EOF) break;
      else if ( (ch>=‘A’ && ch<=‘Z’)||
                (ch>=‘a’ && ch<=‘z’)  ) count++;
   } while (1);
   cout << "Number of real characters "
        << count << endl;;
   return 0;
}
```



Word Statistics

- We would like to write software to compute simple word occurrence statistics from a text file.
- In particular, we would like to count how often each word occurred in the text.
- *First step*: Normalize tokens by removing punctuation marks and by converting to lower case
- *Second step*: Counting the word occurrences using a data structure known as a map (part of the so-called C++ standard template library)

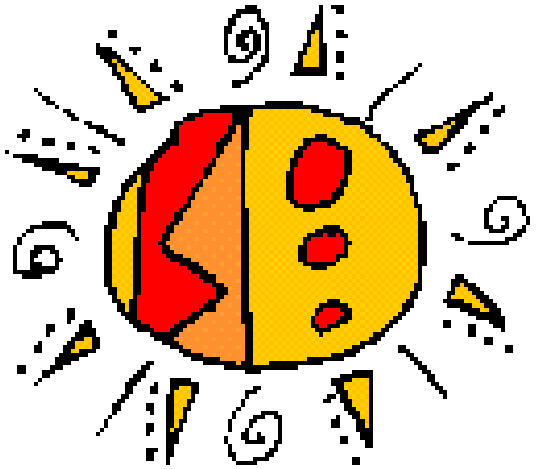
An Extension of the `string` Class

- We propose to extend the C++ **`string`** class as follows:

```
class Word : public string {
public:
  Word()                    : string() {}
  Word(const string& s) : string(s) {}
  //-- remove prefix and suffix of invalid characters
  void clean();
  //-- convert to all lowercase
  void lowercase();
private:
  //-- checks whether a character is (in-)valid
  bool isInvalid(char ch) const;
} ;
```

break

- It’s time for a break – 10 minutes!



An Extension of the `string` Class

- Comments:

```
class Word : public string {
public:
//...
  Word();
//...
};

Word::Word() : string()
{
}
```

The class Word is an extension of the C++ string class

The default constructor without argument

Calls default constructor of the base class string

An Extension of the `string` Class

- Comments:

```
class Word : public string {
public:
//...
  Word(const string& s);
//...
};

Word::Word(const string& s)
: string(s)
{
}
```

Copy constructor from an existing string

The copy constructor simply calls the copy constructor of the base class

clean()

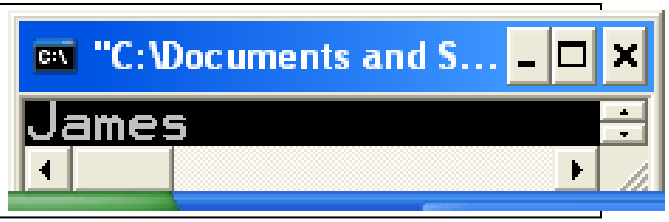
```
void Word::clean()
{
  int len = length(), fi, la;

  //-- find prefix of invalid characters
  for (fi=0; fi<len && isInvalid(at(fi)); fi++);
  //-- find suffix of invalid characters
  for (la=len-1; la>=0 && isInvalid(at(la)); la--);

  (*this) = substr(fi,la+1-fi);
}
```

- Uses the string methods
  - **length()**: length of string
  - **at(int i)**: character at i-th position
  - **substr(int from, int len)**: substring from of length

substr()

- More details on the substring method

```
string name = “James Joyce”;

cout << name.substr(0,5) << endl;
```




```
string name = “James Joyce”;

cout << name.substr(6,3) << endl;
```

isInvalid(char)

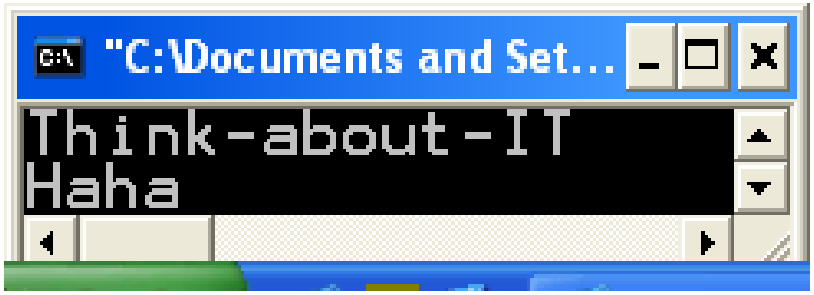
- How we define the isInvalid method depends on what we want to call a word, e.g.

```
bool Word::isInvalid(char ch) const
{
  return ( ch == '.' || ch == ',' || ch == ';' ||
           ch == ':' || ch == '-' || ch == '_' ||
           ch == '`' || ch == '\''|| ch == '\"'||
           ch == '?' || ch == '!' || ch == ' ' ||
           ch== '\n' || ch == '&' || ch == '(' ||
           ch == ')
         );
}
```

Example of `clean()`

- Let's apply the new method

```
Word w1 = "—-Think-about-IT!!!!!!";
Word w2 = "...Haha!:)";

w1.clean();
cout << w1 << endl;

w2.clean();
cout << w2 << endl ;
```



lowercase()

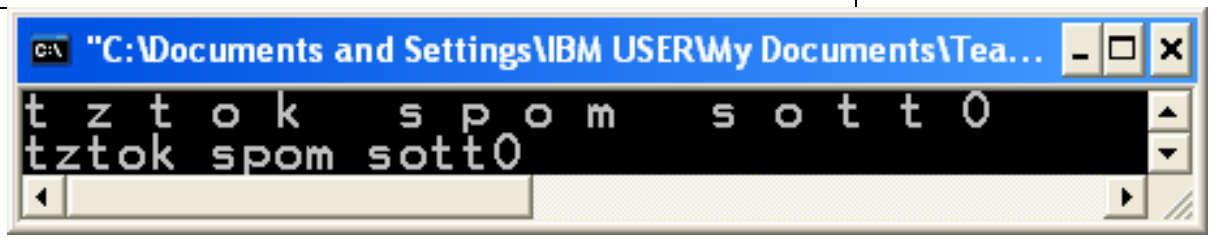
```
void Word::lowercase() {
  for (int i=0; i<length(); i++)
    if (at(i)>='A' && at(i)<='Z')
       at(i) = tolower(at(i)); //-- from cctype library
       //-- equivalently:
       // if(at(i)>='A' && at(i)<='Z') at(i) += 'a'-'A';
}
```

- Uses the string methods
  - **length()**: length of string
  - **at(int i)**: character at i-th position

at() and operator[]

- Here is how to use the at method and the [] operator for C++ strings.

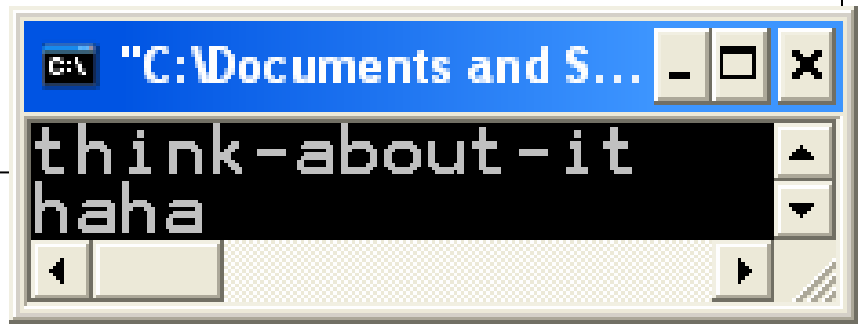
```
int i;
Word w = "Ottos mops kotzt";

for (i = w.length()-1; i>=0; i--)
  cout << w.at(i) << " " ;
cout << endl;
for (i = w.length()-1; i>=0; i--)
  cout << w[i] ;
cout << endl;
```



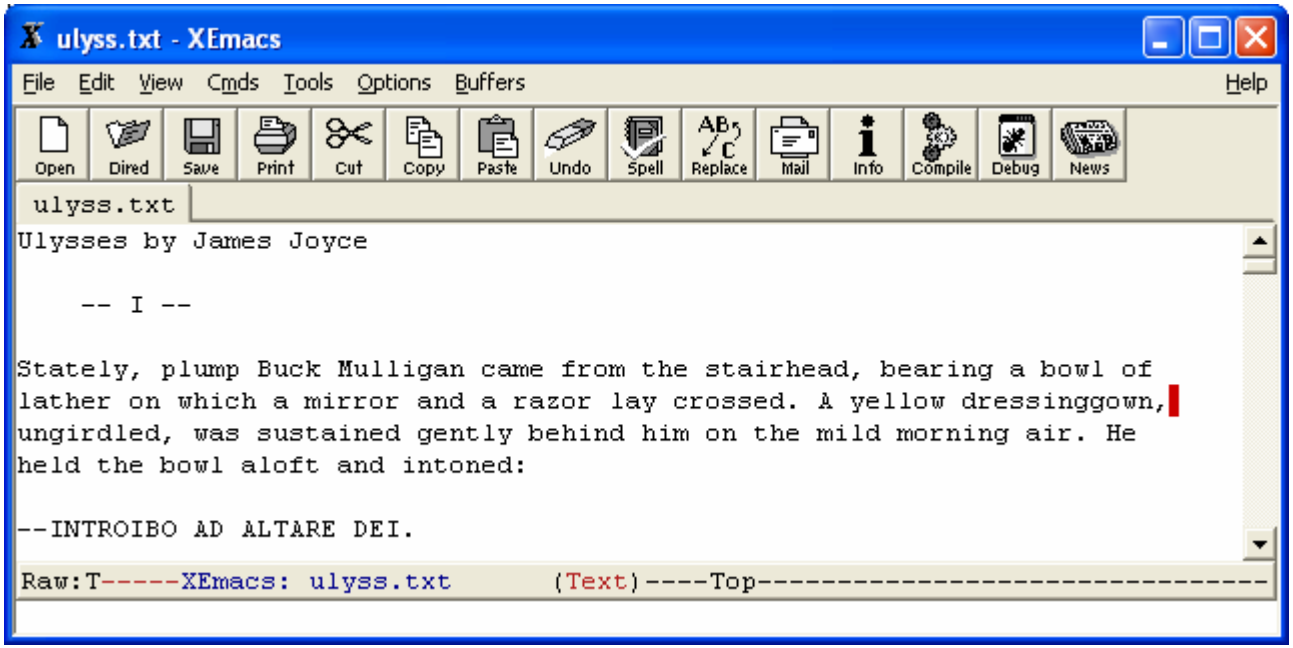
Example of `clean()` and `lowercase()`

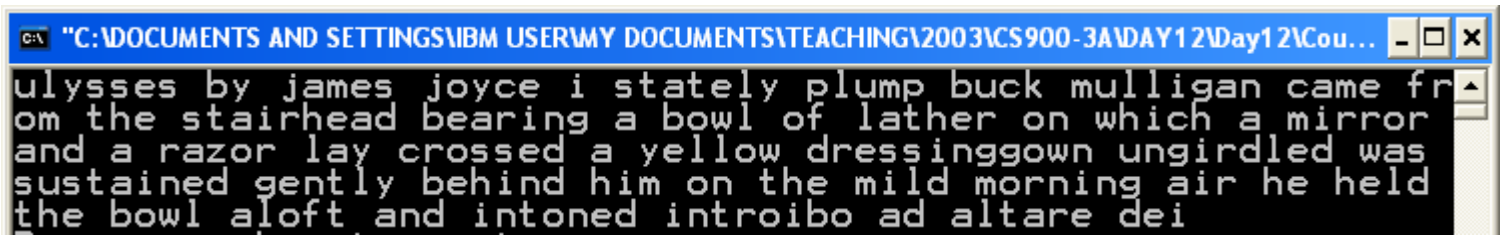
- Let's apply the new methods in conjunction

```
Word w1 = "—-Think-about-IT!!!!!!";
Word w2 = "...Haha!:)";

w1.clean();
w1.lowercase();
cout << w1 << endl;

w2.clean();
w2.lowercase();
cout << w2 << endl ;
```



What we have so far...





Building a Map

- In order to be able to count word occurrences, we would like to have a generalized version of a array which could indexed by strings instead of integers.
- This can be accomplished with a map.
- We can write things like:

```
myMap["happiness"] = 10;
```

- How is this possible?

The Standard Template Library

- C++ has a powerful new library called the STL (standard template library)
- We are interested in a data structure called a map (but there are many others!).
- A map can associate keys from any class/type with values from some class/type

Types are defined in angular brackets

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

map<string,int> myMap;
myMap["happiness"] = 10;
```

Example

- A simple example on how to build a map:

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

map<string,int> myMap;
myMap["van Beethoven"] = 1770;
myMap["Bach"] = 1685;
myMap["Stravinsky"] = 1882;
myMap["Ravel"] = 1875;
myMap["Bartok"] = 1881;
myMap["Hindemith"] = 1895;
```

Iterators

- How do we get to the things stored in a map?
- Let's built a "for" loop:

```
"for (i=first_string; i<last_string; i++)"
```

- In an array we could count i from 0 to length-1, but in a map, that doesn't work.
- Hence, there are special objects called iterators to loop over all the elements in a map.

```
map<string,int>::iterator i;
for (i=myMap.begin(); i!=myMap.end(); i++)
{
  //...
}
```

Const_Iterators

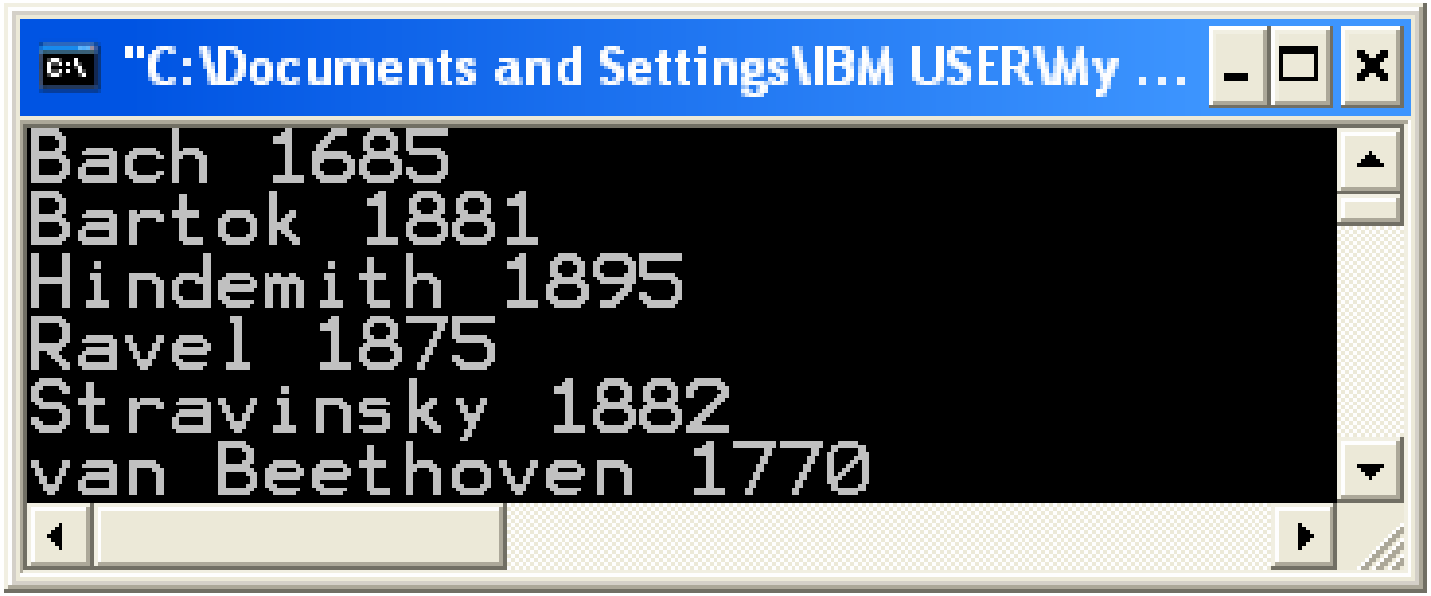
- If the iteration does not change the map (or more generally container), then we should use a const_iterator:

```
map<string,int>::const_iterator i;
for (i=myMap.begin(); i!=myMap.end(); i++)
{
  //...
}
```

Iterators

- Now we also need to access the element pointed to by an iterator. For that we use the * operator.
- We can actually get the key and the value from an iterator. The former is stored as a member first, the latter as a member variables second.
- Example:

```
map<string,int>::const_iterator i;
for (i=myMap.begin(); i!=myMap.end(); i++)
{
  cout << (*i).first  << " "
       << (*i).second << endl ;
}
```


"C:\Documents and Settings\IBM USER\My ...
Bach 1685
Bartok 1881
Hindemith 1895
Ravel 1875
Stravinsky 1882
van Beethoven 1770

Defining a Word map

- A map allows us to store key, value pairs (sorted by key).
- We can use an arbitrary object as a key (such as a **Word** object) and store a corresponding value (such as the word count).

```
#include <map>
class WordMap: public map<Word,int>
{
public:
  void operator+=(const Word& w);
  void print() const ;
};
void WordMap::operator +=(const Word& w)
{
  (*this)[w]++;
}
```

class WordMap

- Comments:

WordMap is derived from a map (a special kind of map)

```
#include <map>
class WordMap: public map<Word,int>
{
public:
  void operator+=(const Word& w);
};
```

We define an operator that we can call every time a word occurs

operator+=

- The implementation of the operator is simple

```
void WordMap::operator +=(const Word& w)
{
  (*this)[w]++;
}
```

Creates a new map entry, if entry is not yet present.

Otherwise, the word count gets incremented by one.

Using the WordMap

```
Word token;
WordMap wm;
WordMap::iterator it;
do {
  in >> token;              //-- read in next token
  if (in.fail()) break;     //-- was that successful?
  token.clean();            //-- clean token
  token.lowercase();        //-- convert to all lowercase
  wm += token;              //-- add occurrence
} while (1);

//-- print out content of map
for (it = begin(); it != end(); it++)
  cout << (*it).first << " " << (*it).second << endl;
```

WordMap counts on

ulyss.txt