Lecture 4a
Using The Standard Template Library

Jack Applin, Guest Lecturer
November 28th, 2017
Announcements

• PA9 is online
  – Efficiency assignment
    • Same as PA8
    • But graded on speed as well as correctness
  – Since Bruce is traveling, use Piazza for questions

• Recitation this week
  – Installing & using Eclipse for C++
    – Mandatory

• Recitation next week
  – Optional help session
STL

1. Containers
   - Abstractions of data structures
2. Iterators
   - Abstractions of pointers
   - Allow you to iterate through containers
3. Algorithms
   - Universal algorithms (e.g. sort)
     • Without reference to object type being acted on
     • Without reference to data structure holding the objects
4. Other
   - Multi-threading  CS370 will cover multi-threading
   - Numerics  Sorry, skipping this part
   - Smart pointers  Thursday & next Tuesday!
Containers (Review)

- Containers are data structures

**Sequences**
- Array (static)
- Vector (dynamic, 1-sided)
- Deque (dynamic, 2-sided)
- Slist (singly linked)
- List (doubly linked)

**Binary Trees**
- Set (unique items)
- Multi-set (repeated items)
- Map (unique keys, key/value pairs)
- Multimap (repeated keys)

**Hash Tables**
- Unordered_set (unique items)
- Unordered_multiset (repeated)
- Unordered_map (unique keys)
- Unordered_multimap (repeated)
Iterators (Review)

• Abstractions of Pointers
  – Overload *, ->, ==, !=, and ++
  – Some overload --, +, and -

• Similar to iterators in Java
  – They support iteration
  – Within the semantics of a container

• Every container defines an iterator type
  – vector<int>::iterator
  – unordered_multiset<Quagga>::iterator
STL Algorithms (Review)

• Goals:
  – Provide useful basic algorithms (e.g. ‘sort’)
  – Without regard to the data type operated on
  – Without regard to the data structure it is stored in
  – Over a whole container or a fragment thereof

• Example

```cpp
template<typename ITER>
ITER max_element(ITER start, ITER end)
{
    ITER max_iter = start;
    for(ITER iter=start; iter!=end; ++iter)
        if ((*iter)>(*max_iter)) max_iter = iter;
    return max_iter;
}
```
An Example

• A simplified version of PA4
• Our example is a Document class with 2 methods:
  – Constructor: reads a text file and computes stemmed term frequencies
  – Print(): writes terms and frequencies to std::cout
• It uses a simplified Stemmer class
  – Removes initial capitals and trailing ‘s’
  – Not a real stemmer, just a placeholder
#ifndef STEM_H_DEFINED
#define STEM_H_DEFINED

#include<string>
using std::string;

class Stemmer {
public:
    //Note: Defined as a Functor
    string operator () (const string& str) const;
};

#endif // STEM_H_DEFINED
class Document {
public:
    Document(ifstream& str);
    void Print() const;
private:
    vector<string> unique_words;
    vector<int> counts;
};

Many of you store terms & frequencies this way
Document Constructor (Part 1)

Document::Document(ifstream& istr)
{
    vector<string> raw_words;
    while(!istr.fail()) {
        string word;
        istr >> word;
        if (!istr.fail()) raw_words.push_back(word);
    }
}

Stemmer stemmer;
vector<string> stemmed_words;
for(unsigned int i=0; i < raw_words.size(); i++) {
    stemmed_words.push_back(stemmer(raw_words[i]));
}
// Step #3: sort the terms
std::sort(stemmed_words.begin(), stemmed_words.end());

// Step #4: keep unique terms, count frequencies
unique_words.push_back(stemmed_words.at(0));
counts.push_back(1);
for(unsigned int i=1; i < stemmed_words.size(); i++) {
    if (0 == stemmed_words.at(i).compare(stemmed_words.at(i-1)))
        counts.back()++;
    else {
        unique_words.push_back(stemmed_words.at(i));
        counts.push_back(1);
    }
}
}
Using STL

• STL algorithm can make this code
  – Simpler
  – More efficient

• Terms & frequencies are key/value pairs
  – For every term (key) we have a count (value)

• Replace the term & frequency vectors with a map<string, int>
```cpp
#include<map>
using std::map;

class Document {
public:
    Document(ifstream& str);
    void Print() const;

private:
    map<string, int> frequencies;
};
```
Document::Document(ifstream& istr)
{
    vector<string> raw_words;

    while(!istr.fail()) {
        string word;
        istr >> word;
        if (!istr.fail()) raw_words.push_back(word);
    }

    Stemmer stemmer;
    vector<string> stemmed_words;
    for(unsigned int i=0; i < raw_words.size(); i++) {
        stemmed_words.push_back(stemmer(raw_words[i]));
    }

    for(unsigned int i=0; i < stemmed_words.size(); i++) {
        frequencies[stemmed_words.at(i)]++;
    }
}
Using STL (II)

• Step 2 copies words from one vector to another while stemming them.
• This is what the std::transform does
  – Conveniently, the stemmer is a functor

• Potential STL trap: memory
  – Transform doesn’t allocate new memory
    • It assigns, it doesn’t push
  – Therefore pre-allocate destination array
Document::Document(ifstream& istr)
{
  vector<string> raw_words;

  while(!istr.fail()) {
    string word;
    istr >> word;
    if (!istr.fail()) raw_words.push_back(word);
  }

  Stemmer stemmer;
  vector<string> stemmed_words(raw_words.size());
  transform(raw_words.begin(), raw_words.end(), stemmed_words.begin(), stemmer);

  for(unsigned int i=0; i < stemmed_words.size(); i++) {
    frequencies[stemmed_words.at(i)]++;
  }
}
Using STL (III)

• Can we combine the loops?
• Trickier: not just iteration
  – First loop copies from one vector to another
    • While stemming the strings
  – Second loop increments frequencies
• Solution
  – Use the local memory in the functor
  – Replace transform with for_each (side effects only)
using std::string;
#include<map>
using std::map;

class Stemmer {
public:
    Stemmer(map<string, int>& freq) : store(freq) {}
    void operator () (string str) const;
protected:
    map<string, int>& store;
};
void Stemmer::operator() (string str) const
{
    // This is same as before
    if (isupper(str.at(0)))
        str.at(0) = tolower(str.at(0));
    if (str.back() == 's')
        str = str.substr(0, str.length()-1);
    // This is new
    store[str]++;
}
Document Constructor

Document::Document(ifstream& istr)
{
    vector<string> raw_words;

    while(!istr.fail()) {
        string word;
        istr >> word;
        if (!istr.fail()) raw_words.push_back(word);
    }

    Stemmer stemmer(frequencies);
    for_each(raw_words.begin(), raw_words.end(), stemmer);
}
Using STL (IV)

• How about the I/O loop?
• Memory must be handled…
  – Don’t know length of file in advance
  – Can’t preallocate array
  – Create a new functor, have it push data
class Reader {
public:
    Reader(vector<string>& word_vector) words(word_vector) {}  
    void operator() (const string& str) con {words.push_back(str);}

protected:
    vector<string>& words;
};
Document Constructor

Document::Document(ifstream& istr)
{
    vector<string> raw_words;
    Reader reader(raw_words);
    for_each(istream_iterator<string>(istr), istream_iterator<string>(), reader);

    Stemmer stemmer(frequencies);
    for_each(raw_words.begin(), raw_words.end(), stemmer);
}
Using STL (V)

• Why create stemmer & reader as local variables?
  – We don’t re-use them

• Why not just allocate them inline?
Document Constructor

Document::Document(ifstream& istr)
{
    vector<string> raw_words;
    for_each(istream_iterator<string>(istr),
             istream_iterator<string>(), Reader(raw_words));

    for_each(raw_words.begin(), raw_words.end(),
             Stemmer(frequencies));
}

2017-12-04
Using STL (VI -- and last)

• In PA4, you needed
  – a vector of words
    • To resolve capitalization
    • To judge reading level
  – Frequencies of terms
    • To compute TF-IDF scores

• But for this example I can fuse the loops
Document Constructor

// Yes, this is one line of top-level code...
Document::Document(ifstream& istr)
{
    for_each(istream_iterator<string>(istr),
             istream_iterator<string>(),
             Stemmer(frequencies));
}