CS900 Lab - Day 13
July 16th, 2003
Instructor: Thomas Hofmann

New Topics Covered
Using STL maps and implementing a trigram language model, Lecture: Day 12/13

Installing
Copy the file trigram.cpp from the course directory

```
cp /course/cs900/day13/src/trigram.cpp .
```

Compiling
You should use the gnu C++ compiler g++. Your program will be in trigram.cpp . To compile type

```
g++ trigram.cpp -o trigram -Wall -O4
```

The -O4 option generates speed optimized code and -Wall makes sure the compiler outputs all warnings.

Problem 1: Bigrams

In the first part of the lab, we want you to implement a bigram class, following the steps presented in class and implementing the missing methods and functions. Basically, you should at least define the following methods:

```
class BigramMap : public map<Word,WordMap>
{
public:
      void   add(const Word& w, const Word& v);
      void   print() const ;
      string random(const Word& w);
};
```

The Word and WordMap class is provided to you in the source file, since this was largely the topic of yesterday's lab.

The add method has been discussed in class, the random method is something you have to implement. *Hint: Use the random method of WordMap.*

Write a main function that reads in a text file like the ones provided in the course directory

```
/course/cs900/day12/data/ulyss.txt, ./bible.txt, ./kant.txt
```

It then preprocesses all tokens and adds all consecutive word pairs to the bigram map. After yiu have done this, the program shpuld generate a specified number of words by sampling from the bigram model stored in the generated BigramMap. This means, you first sample the start word from the standard WordMap (which you should also generate) and then use the most recently generated word as input to BigramMap::random() to generate the next word in the sequence. This process is then repeated.

Problem 2: Trigrams

Repeat what you did in Problem 1, where instead of a BigramMap, you implement and use a TrigramMap. The WordPair class is provided to you in trigram.cpp. Basically you only need to implement the add and random() methods of TrigramMap

```
class TrigramMap : public map<WordPair,WordMap>
{
public:
      void   add(const WordPair& w, const Word& v);
      void   print() const ;
      string random(const WordPair& wp);
};
```

Sample new text using the trigram model adapting the procedure developed in Problem 1.

Problem 3: Trigram Assistant

Use your trigram map in a creative way. For example, ask the user to enter a sentence or piece of text, tokenize it and normalize the tokens using clean() and lowercase(). Then output the sentence by putting each word in a separate line and adding the most likely prediction based on the trrigram model in parantheses. For example, a program run could look like this:

```
Enter a sentence: The white horse crosses the river.

the
white
horse (house)
crosses (runs)
the (the)
river (bridge)
```

It would be best to implement a method best which takes a word pair and outputs the word with the highest word count for this word pair from the TrigramMap.

Think of other ways of using the trigram model, e.g. for an automatic spell checker.