COMP 11 – Introduction to Computer Science – Fall 2002

Project pp10

Programming project due week of Mon 11/18 (wk. 12)

Due dates:
Section 01 (Block H): Wed 11/20 11:00 pm
Section 02 (Block E): Thu 11/21 11:00 pm
Submissions may begin on Mon 11/18 at 5pm.

Project pp10: 8 points. (1 additional style point.) Word Co-locations

Introduction

In recent years work in computational linguistics relies on programs that manipulate text and help us discover properties of our language. One simple property is “word-co-location” identifying words which occur together many times in the text. These could be indicators of special word combinations or peculiarities of the text in question. In this programming project you will write a program making a first step towards such an analysis. You will also see why efficient data structures are important.

Your program will read words from the input, count how many times each consecutive pair of words appears in the text, and will print the top 10 such pairs. For example, if the input is

test file for programming project alpha
this is a nice day for programming project alpha
mr day is a project alpha
this is a test
this is a nice day
test file

The output will indicate that the most frequent pair is “is a” (5 times), then “this is” (4 times), etc., listing the top pairs with their frequencies. Ties are resolved arbitrarily.

A Word-Counting Class

In order to perform this you will first implement a word counting class we call ctable. The class will allow its user to “insert” strings into the table, counting how many time each such string appears. The class should also keep track of how many different strings have been inserted. The interface is as follows:

- `ctable()`: a constructor to initialize the class.
- `void insert(string k)`: if the key `k` has not been seen before it should be inserted to the table with count 1, and `size` should be changed accordingly. Otherwise, the count in the appropriate position should be incremented by 1.
- `int getCount(string k)`: if key `k` has already been seen, its count is returned; otherwise 0 is returned.
- `void display(int n)`: the `n` strings with highest count are displayed together with their counts. See below for the exact format. If the array has less than `n` entries then only those entries are displayed.
Your class should be able to handle up to 3000 different strings. You should implement the class in files ctable.h and ctable.cpp
Notice that we have only specified the interface of the class. The actual representation and implementation is up to you. It is important that you first analyze what representation will support these operations and in what way. This might suggest other functions that should exist in the class. For example, one possible implementation is to use parallel arrays for the strings and their counts as well as a variable the keeps track of the size. You might also want to implement a sorting utility which is used by the display function. This and all other functions you add should be private members of the class.

Example: We have prepared a main program in order to help you test your class. It is available at /comp11/lab/pp10test1.cpp. You should copy this file into your directory, and make sure that your class compiles and runs correctly with this main program. Once it runs correctly, it would be a good idea to make modified main programs with which to test your program further before going to the next step of the project. Here is the main function from this file:

```cpp
int main()
{
    ctable mytext;
    string a, b, c;

    a = "day";
    b = "where";
    c = "comp11";
    mytext.insert(a);
    mytext.insert(b);
    mytext.insert(b);
    mytext.insert(c);
    mytext.insert(a);
    mytext.insert(c);
    mytext.insert(c);
    mytext.insert(a);
    mytext.insert(c);
    mytext.insert(c);

    cout << "------------------------\n";
    mytext.display(2);
    cout << "------------------------\n";
    mytext.display(9);
    cout << "------------------------\n";
    cout << a << " " << mytext.getCount(a) << endl;
    cout << b << " " << mytext.getCount(b) << endl;
    cout << c << " " << mytext.getCount(c) << endl;
    cout << "when" << " " << mytext.getCount("when") << endl;
    cout << "------------------------\n";

    return 0;
}
```
Here is the output that should be obtained with this main program (note that no input is needed here):

```
----------------------
comp11 5
day 4
----------------------
comp11 5
day 4
where 3
----------------------
day 4
where 3
comp11 5
when 0
----------------------
```

**Word Co-locations**

Once your class is working you should write a main program to perform the word co-location task. Your program should read a sequence of words from the input (until the end of file marker is found), count occurrences of consecutive pairs of words in the sequence, and then display the pairs with the 10 top counts. Each pair and count should appear on a separate line and nothing else should be printed by the program. See below for sample output.

To help you test your program we have prepared two text files. The data given in the introduction is available as /comp/11/lab/pp10data1. A much longer data file which includes a section from Jane Austen’s “Pride and Prejudice” is available as /comp/11/lab/pp10data2. In the latter we have removed all punctuation and very common words (like “this”, “is” etc) so as to get more interesting results. You should put your main program in a file named coloc.cpp

Here are some suggestions on how to write and use your code:

- Your **main** function should read strings until the end of file is reached. This can be done by:

  ```
  cin >> mystring;
  while (!cin.eof()) {
      // do whatever is necessary
      // ...
      // ...
      cin >> mystring;
  }
  ```

- Instead of typing in words as input to the program simply run it with input redirected as in **a.out < mydatafile**. To simplify debugging you may at first want to write your own data file with only a few words.

- The strings you put into the table should be composed of concatenations pairs of consecutive words with an underscore between them, e.g. ‘‘nice_day’’.
Examples: If you run the program with input from the file `pp10data1` given above the output should be as follows (but since ties are resolved arbitrarily the order of lines with count 2 is allowed to be different).

```python
is_a 5
this_is 4
project_alpha 3
test_file 2
for_programming 2
programming_project 2
a_test 2
alpha_this 2
a_nice 2
nice_day 2
```

Here is the result you should get using the second data file:

```python
mr_bingley 29
mr_bennet 16
my_dear 14
mrs_bennet 12
mrs_long 10
mr_darcy 9
but_he 8
miss_bingley 7
miss_bennet 6
young_man 5
```

Submitting your program

You must use filenames as specified above. Assuming these are in the current directory on andante, you should submit by typing

```
provide comp11 pp10 ctable.h ctable.cpp coloc.cpp
```

Following the structure above, the grader will run two kinds of tests on your program: (1) we will use different main programs and run them with your class, and (2) we will provide different text files and run your main program and class to get co-location counts.

**Programming style guidelines for this week**
To get the style point, you must satisfy all of the following requirements: (1) Write the class with public and private sections as described above, (2) Have an introductory comment at the top of your program, for each function and for each logical section, (3) Have a comment for each variable and function describing their purpose, (4) Organize your code clearly with white space separating functions and logical sections and indenting the program properly.

Your first submission will be used for style grading, even if your best grading score is on a different submission.

**For the Curious:**
(1) You will notice that the text from “Pride and Prejudice” contains only a small part of the book. If you try to run your program on larger texts you will notice a long wait before it completes. This is due to the fact that we are not using the most efficient data structures for the tables. If you continue to COMP15 you will learn how to create better data structures so that much larger texts can be handled.

(2) More information on co-locations and other statistical tools in language processing can be found in *Statistical Natural Language Processing* by C. Manning and H. Schutze, MIT Press, 2000.