FOLIA SCANDINAVICA
VOL. 4 POZNAŃ 1997

THREE TEXT SEARCHING UTILITIES

JAKUB SATERNUS

INTRODUCTION

During my stay at the University of Gothenburg I had access to a large collection of Swedish texts (nearly 30e6 words). My Master's thesis was consequently based on a computational research on Swedish lexicalised verb phrases. In order to carry out such an automated survey, one has to find an appropriate tool for it. I wrote the three programs because I found the standard UNIX power tools to be of a too general nature.

I considered writing so-called "wrappers" i.e., usually shell scripts that at some stage invoke other programs in a specially created environment, but after a few attempts I decided to implement a very specialised set of small programs that, connected together, would give the desired result. I am planning to discuss the linguistic goal of my research in a separate paper. The results are very interesting and deserve a proper presentation. Here I would like to focus on the programs and programming issues only.

This collection of small UNIX utilities was not meant to be a fully customisable set of programs. It was created in order to satisfy my own needs. Each of the programs is in a way finite i.e., it performs only one task and thus any other uses of this software should be considered an abuse of it, not predicted by their author.

In the final stage of the code development the main `Makefile` knew how to make three programs:

1. `fras`
2. `cntwrds`
3. `conc`

This paper cannot be considered as a proper documentation of the above mentioned programs. Firstly, it is being written 2 months after the programs were finally "checked in" using the Revision Control System, secondly it is being written as a separate paper, not firmly attached to the source code. To be exact I am editing this text in MS Word for Windows 2.0a (at the editor's request), while the programs were developed on UNIX machines. The source files, on the other hand, do contain some valuable comments but they are not suitable for generating documentation. To cut a long story short, I have not written the programs in CWEB.

As for the languages that I used: choosing UNIX means choosing C. I went a bit further and used C++ because the GNU C++ library provides very functional implementations of a dozen classes operating on the most widely used data structures. I also used lex for generating lexical analysers for all applications. All of the programs also rely heavily on a couple of common C++ string handling classes.

Each program takes as an obligatory parameter one or more (in case of **fras**) word lists i.e., files containing a number of words separated by a newline character. Trailing spaces are stripped but otherwise the list should be prepared very carefully, which in most cases implies that it should not be edited by hand. Here is a sample:

```
anställa
anställde
anställer
anställt
använda
använde
använder
använt
avvisa
avvisade
```

The order does not matter as the whole list is always sorted before it is used. This approach may seem not too economical as it implies that each list fed into the program is sorted on every run but this is due to the discrepancy between the **sort** standard utility and the standard **strcmp()** function on the system that I originally developed my programs – they handled the Swedish national characters differently.

Every program takes its input from **stdin** and produces output on **stdout**. Redirection is necessary in order to feed data and save the results. This can usually be done by typing:

```
$ prog_name arg1 arg2 < input_file > output_file
```

at the UNIX shell prompt.

1. FRAS

usage: fras verbs adverbs nouns prepositions

Each of the arguments should be a valid file name containing a list of words to be searched for. The program was primarily used to search for patterns of the form:

verb [adverb] *noun* [*preposition*]
(square brackets denote an optional component)

and present it in the following way:

verb [adverb] noun preposition (following words)

provided that the program has found an adverb and a preposition. Otherwise the output might look like:

verb noun (following words)

so it is rather highly specialised and thus does its job quite well. Extending it to handle different combinations should not be too difficult (one could invoke it by providing three adverb lists but the question is whether such combinations are valid syntactic structures). Even a small grammar (specified in an additional file) could be implemented which would mean turning the program into an intelligent parser. So far it was not necessary to do so.

Here is a sample search session:

```
sympatiserade man med (deras kamp,)
var [ens] plikt (och man skulle vara aktiv i)
liknande grupper bland (tyska flyktingar som kämpade inbördes)
snara förändringar (och trodde att detta var socialismens)
bottnar [också] besvikelsen (och bitterheten,)
kunde man (beskylla någon för.)
blev mötet med (Sverige?)
```

The amount of the right context depends on whether a punctuation mark comes right after the found construction or a few words later.

2. CNTWRDS I.E., COUNT WORDS

Counting all the words in a file is trivial. This simple command does the job:

$ wc - w filename

counting the number of occurrences of the word "the" in a file is also easy:

```
$ grep 'the' filename | wc -l
```

but what if we want to count all the occurrences of several words in several files? There are a number of solutions based on different standard and non-standard UNIX tools but they all mean writing some sort of a script in some ugly and cryptic (e.g. Perl) language. That is why I decided to write my own little word counting program. If it proves to be bug-free I may well release it . . .

```
usage: cntwrds word_list
```

The only argument is a file containing a list of words that we are interested in counting in the input stream. It can even handle hyphenated words by simply ignoring hyphenation marks (which can be dangerous in some cases). The output is very simple and well suited to further automatic processing.

```
 0 anställa
 0 anställde
 0 anställer
 0 anställt
54 använda
14 använde
41 använder
 7 använt
```

3. CONC I.E., IMITATION OF A CONCORDANCE PROGRAM

This program was inspired by an interactive concordance program that I once had access to. The main drawback of that original program was that a user could not supply more than one word to look up at a time. In my case the number of words I needed to look up, in order to analyse the context they occur in, was over 300, so there really was no other way than to write "a quick hack" to solve the problem. I needed to find out how often Swedes omit certain grammatical contructions before past participle forms of verbs.

```
usage: conc [-w width] [-t tab] -f word_list
```

Taking a short list of random words:

```
are
file
files
to
```

the program generates the following output:

```
 are defined in the system header _file sys/stat.h. The magic number tests are
  /stat.h. The magic number tests _are used to check for files with data in
  The magic number tests are used _to check for files with data in particular
      tests are used to check for _files with data in particular fixed formats.
         (compiled program) a.out _file, whose format is defined in a.out.h and
standard include directory. These _files have a 'magic number' stored in a
  place near the beginning of the _file that tells the UNIX operating system
   UNIX operating system that the _file is a binary executable, and which of
   'has been applied by extension _to data files. Any file with some invariant
been applied by extension to data _files. Any file with some invariant
  by extension to data files. Any _file with some invariant identifier at a
```

This might not seem too interesting but in fact what we get here as a result is the actual usage of these four words. The list can easily be further processed by the standard UNIX utilities.

The `-w` parameter specifies the width of the output and `-t` specifies the position of the underscore. (I chose the underscore because it is very seldom used in ordinary texts and on the other hand it is very easy to spot.)

I hope that this brief description of the three text utilities will draw the reader's attention to the enormous possibilities of text processing in the UNIX environment.

REFERENCE

Kernighan, Brian W. and Dennis M. Ritchie, 1988, *The C Programming Language*, Prentice Hall Software Series.

Knuth D. E., 1973, *The Art of Computer Programming*, volume III Sorting and Searching, Addison-Wesley Publishing Company.

Levine, John R. et al., 1992, *lex & yacc*, O'Reilly & Associates, Inc.

Oram Andrew and Steve Talbott, 1991, *Managing Projects with* make, O'Reilly & Associates, Inc.

Sedgewick Robert, 1990, *Algorithms in C++*, Addison-Wesley Publishing Company.