OpinionMining

1. **Outline of plan**.
The method to read from a file and identify separate words, before storing them in a list, may go as follows:

> **⟨ 1. Start. ⟩**
> **⟨ 2. Prepare pointers to word-list and to file. ⟩**
> **⟨ 3. Read words into list of words. ⟩**
> **⟨ 4. Alphabetise the word-list. ⟩**
> **⟨ 5. Starting from the top of the list, search for matches in the "adjectives" file, and print same. ⟩**
> **⟨ 6. Stop. ⟩**

1.1 **Creating a list of words**
This time, we start from the near-beginning, unlike some others we have done. So,

> **⟨ 2. Prepare pointers to word-list and to file. ⟩:-**
> **char** *word_list* **[1000][1000];**
> **FILE ****F* ← *fopen*(*argv*[1], `"r"`);

For the next step, we must also store the characters into a buffer as we read them (except for the spaces and the puntuations.) We modify a 4-state DFA into a structured block as specified below. On matching a *word* the DFA stores it in *word_list*. After reading the entire file referenced by *F*, it goes to a step which alphabetises the list.

> **⟨ 3. Read words into list of words. ⟩:-**
>
> *word_no* ← **0;**

```
while( ~ feof( F ) )
{

   fscanf( F, "%s", &word_list[ word_no ] );
   word_no ← word_no + 1;

   c ← word_list
            [ word_no - 1 ]
            [ strlen( word_list [ word_no - 1 ] ) - 1 ];
   if( ispunct( c ) )
   word_list[ word_no - 1 ]
         [ strlen( word_list [ word_no - 1 ] ) - 1 ] ← '\0' ;

}
```

At the moment that the control goes to the alphabetisation step, *word_list* is populated with every word in the file, i.e. every string which is bounded at at least one end by a *terminator*, which is either a *blank space* or a *punctuation mark*. The parameter *word_no* is equal to the number of words in the file (and subsequently in *word_list*), and *word_list* [*word_no*] is an empty pointer.
Since we are not altogether concerned with the efficiency of the alphabetisation, we arbitrarily choose a sorting algorithm- insertion sort. Referring to the very nice, (and simple) implementation on *Wikipedia*, (which I am almost completely certain originated in Jon Bentley's *Programming Pearls*) I have replicated it, more or less, over here- but applied to *strings* rather than *numbers*.

```
⟨ 4. Alphabetise the word-list. ⟩:-
word_count ← word_no;

i ← 1;

while( i < word_count )
{
   j ← i;
```

```
    while( (j > 0) ∧ (word_list [ j - 1 ] > word_list [ j ]) )
    {
      (Swap:)
      strcpy( temp, word_list [ j ]);
      strcpy( word_list [ j ], word_list [ j - 1 ]);
      strcpy( word_list [ j - 1 ], temp);
      j ← j - 1;
    }

  i ← i + 1;
  }
```

1.2 **Identifying adjectives**

We should like to use the variable name *word_no* as the index in what follows. We will perform a sequence of linear searches, by searching for each *word* in *word_list* inside the file referenced by *Adj_file*- the file that contains adjectives. It will report a match when the *strcmp* function of C's standard library evaluates to 0, in this (C/C++) implementation. Further, we have *not* assumed that the file with the adjectives is lexicographically well-ordered, so the below linear search can be further optimised for such a file. However, as can be seen, the program is independent of the data set of adjectives, and we are feeding both the file to be mined *as well as* the file with the list of adjectives to this program.

⟨ 5. Starting from the top of the list, search for matches in the "adjectives" file, and print same. ⟩:-

```
for(
     word_no ← 0;
     word_no < word_count ;
     word_no ← word_no + 1
    )
{

   strcpy ( T, word_list [ word_no ] );
```

```
    (Linear Search:)
    fscanf ( Adj_file, "%s", &adj );

    while( ~feof ( Adj_file ) )
    {
      if( strcmp ( adj, T ) = 0 ) printf ( "\n%s", adj );
      fscanf ( Adj_file, "%s", &adj );
    }

    (Start again at top of file.)
    fseek ( Adj_file, 0, SEEK_SET );

}
```

Are there any loose ends? Let us do steps 1 and 6.

```
⟨ 1. Start. ⟩:-
#include<stdio.h>
#include<string.h>
#include<ctype.h>


int main( int argc, char *argv [] )
{

   FILE *Adj_file ← fopen ( argv [2], "r" );
   char c, temp [1000], adj [1000], T [1000] ;
   int i, j, word_no, word_count ;
```

Remember that **goto** statement we dropped earlier?

Let's take care of that-

⟨ 6. Stop. ⟩:-

End:
return 0;
}