CS 136 Fall 2008
Mark Giesbrecht

Assignment 6
Due Friday, November 21, 2008 at 4pm.

All coding questions are to be implemented in the C language. The prototypes of all functions which are required in the questions should be stored in an appropriately named `.h` file, which you should `#include` into your `.c` file of solutions, and into your `.c` file of drivers.

1. You are to write a program which keeps statistics of the number of times each alphabetic character (a-z, A-Z) appears in the standard input. Your output should be a list of 26 numbers consisting of the number of times each letter "a" through "z", in order, appears. Capital letters and small letters are counted as the same. Other, non-alphabetic, characters are ignored.

   For example, if the standard input contains

   ```
   The quick brown fox
   jumps over the
   lazy dog.
   ```

   followed by EOF, the output should be

   ```
   1 1 1 1 3 1 1 2 1 1 1 1 1 1 4 1 1 2 1 2 2 1 1 1 1 1
   ```

   Use the `getchar()` function, which reads the next character from the standard input. It returns a `char`, or `EOF` if the end-of-file is reached.

   Your function should have prototype `void collate_ch()`, and should be placed, along with any helper functions in a file called `collchars.c`. You should include a file containing the header for `collate_ch` called `collchars.h`. You should also write a driver program with a `main` function and appropriate tests in `collchars-driver.c`.

   Hint: It might be useful to note that if `ch` is an upper case alphabetic character (i.e., a member of {'A', 'B', ..., 'Z'}), then `ch-'A'` has a value in $0 \ldots, 25$.

2. One way to store a list of words (of different lengths) is to first allocate a large array of `char` to contain all the characters in all the words, and an array of pointers to `char` to point to the beginning of each word (stored in `mem`). I.e.,

   ```
   #define CHARS_MAX 100000
   #define WORDS_MAX 1000
   char mem[CHARS_MAX];
   char *words[WORDS_MAX];
   ```

   Then `words[0]` is the first word, `words[1]` the second, etc. It is probably useful to maintain an index and pointer

   ```
   int next_word_idx;
   char *next_word_start;
   ```

   which contains the index in `mem` of the next word (initialized to 0), and a pointer to the next available character, respectively.

(a) Write a function `add_word`, which consumes a string `wd` and returns an `int i`, such that `wd` is added to `words` and `strcmp(wd,words[i])==0` (i.e., they are equal). Typically the returned `i` would be the old value of `next_word_idx`, following which `next_word_idx` is incremented. If no more space is available in either `mem` or `words`, then -1 is returned.

It is probably a good idea to store the string and the trailing `'0'`.

(b) Write a function `find_word`, which consumes a string `char *wd` and returns the index `int i` in `words` such that `strcmp(wd,words[i])==0` (equal) if `wd` is stored in `words`, and `-1` otherwise.

(c) Write a function `unique_words`, with header `void unique_words();`, which reads from standard input and then prints all the unique words in the file, each on its own line. A word consists of printable characters, surrounded by white-space (and printable characters are defined as those with codes between 41 and 126 inclusive).

Please make use of the standard functions `isprint`, `isspace`, `isalpha`, etc., to determine what type of character you are looking at.

Throughout, you may assume that all words have less than 30 characters.

You should put your three functions `add_word`, `find_word`, and `unique_words` in a file `collwords.c`, and their prototypes in `collwords.h`. You should also write a driver program with a `main` function and appropriate tests in `collwords-driver.c`.