

Computer Science 215
Tools and Techniques for Software Development
Summer 2002
Project 4 – Index

Due: Monday, 6/17/2002 1:00 p.m.

Description

For this assignment, you will write a program called `index.c` to produce an index for a body of text. You must use structures, dynamic memory allocation, and linked lists.

From the command line, the user will invoke the program in the following manner:

```
index <input file name>
```

where `<input file name>` refers to the file containing the text body. If the input file cannot be found, the program should print an error message and exit.

Your program will then read each line of the file, removing punctuation and other extraneous characters along the way. From each line, individual words will be extracted and stored in the program's index structure (discussed in more detail below).

Associated with each word will be a list of numbers representing the location of the word within the document. For our program, these numbers represent section numbers (rather than the usual page numbers). The beginning of a new section is indicated in the input file in the following way:

```
<blank line>  
<section number> <section title>
```

For example,

```
3 Structures in C
```

Any part of the file that has a blank line followed by a number in the left-most non-white space position indicates a new section. Obviously, no duplicate words, or duplicate section numbers for a particular word, should appear in the index.

After all of the words have been stored in the index, your program will read a file called `omit.txt`, containing a list of words (e.g. "a", "and," "the," etc.) to delete from the index. The format of this file is one word per line, not necessarily in alphabetical order. If the file cannot be found, the program should continue and not omit any words.

Finally, your program will write the index to a file called `index.txt`. If the file cannot be opened for writing, the program should print an error message and exit. All words in the index must be lower-case and in alphabetical order, with one word and its section numbers (comma-separated) per line. Write a single letter (e.g., "A") to head each letter

segment of the index. Follow each list of words for a particular letter with a blank line. (See the example output on the webpage.)

Internals

For this program, you **must** use **structs** and **typedefs** for the following types:

WORD_T – a struct containing three fields: a character array **word**, an integer array **sections**, and a pointer **next** pointing to the next word in the list
INDEX_T – a struct containing one field: an array **head** of pointers to **WORD_T**

You may assume that the input file will have no more than 30 sections, that lines in the input file will be no longer than 100 characters, and that words will never be longer than 30 characters. The **head** array in **INDEX_T** will contain one entry for each letter in the alphabet. This entry will be the head of a linked list containing sorted words beginning with that letter.

For this program, you **must** define functions with the **exact** prototypes listed below (these will *help* you):

```
void insert_word (INDEX_T *index, WORD_T *word);  
WORD_T *find_word (const INDEX_T *index, char *word);  
void exclude_words (INDEX_T *index);  
void write_index (const INDEX_T *index);
```

insert_word takes **word** (which is already initialized with the word, a single section number, and a **NULL** **next** pointer) and inserts it into **index**

find_word searches through **index** for **word** and returns a pointer to that word in **index** or **NULL** if **word** is not found

exclude_words reads **omit.txt** and deletes those words from **index**

write_index traverses **index** and writes out the formatted index in **index.txt**

You must define additional functions for other parts of the program, but you are free to design them as you like, as long as you employ good design techniques.

Extra Credit

There are two opportunities for extra credit:

1. [3 points] Modify the code so that **word** in the **WORD_T** struct is a **char** pointer (rather than an array). Dynamically allocate the minimum amount of memory for each word as it is stored in the index.
2. [7 points] Modify the program so that the section numbers are stored in a linked list rather than an array. Create a section number struct and dynamically allocate memory for each section number before inserting it into the list.

If you decide to submit either of these modifications for extra credit, please state that in the comment section at the beginning of your code. Also place special comments around those sections of your code which implement these changes.