CIS*2520 Data Structures Assignment 2

Due: Sunday, Oct. 20, 2019, 11:59pm
Version 1.00 (changes highlighted in yellow)

For this assignment you will be using linked lists to do a document analysis.

For the purposes of this assignment, we will be using texts from the Project Gutenberg web-site (https://www.gutenberg.org) which is a library containing of 60,000 free eBooks. You can search for a book or pick one of the most popular ones: https://www.gutenberg.org/ebooks/search/%3Fsort_order%3Ddownloads. We will be using the "Plain Text UTF-8" versions of the texts.

You will divide the file up into words, and place those words, individually, into linked lists (one word per node). For the purposes of this assignment, *a* word is defined as:

- Any contiguous sequence of characters, numbers, single hyphens and single apostrophes,
- Any contiguous sequence of identical punctuation symbols (including double hyphens), or
- A blank line.

Blank spaces and new-lines at the end of a line of text will be used to determine word boundaries, but are not, themselves, considered words. New-lines with no text on them *are* considered words.

Some examples:

Text	Words
`[EBook #1342]`	`[ EBook # 1342 ]`
`*** START`	`*** START`
`Bennet," said`	`Bennet , " said`
`"_You_ want`	`" _ You _ want`
`three-and-twenty years`	`three-and-twenty years`
`Hurst's gown--"`	`Hurst's gown -- "`

Assume that each line of text contains less than 256 characters including the new-line character ('\n') at the end.

You will write a c file, called "text.c", containing the functions detailed below, and an accompanying header file, called "text.h", with data structures and function prototypes.

```
struct node_struct
{
  void *data;
  struct node_struct *next;
};
```

This structure will represent your text. The attribute “data” will point to malloced memory holding a string of text followed by the ‘\0’ character.

It will also be used to store lists of node structures that will be created by searching the words in the main list. This structure will be used to represent the elements in a linked list that represent the result of a search of a linked list. Each result pointer will point to an item that was found and still exists in the word_struct list.

```
struct node_struct *txt2words( FILE *fp );
```

This function will read text from the file pointer fp, one line at a time. It will split each line into words allocating a string for each word and storing the address of the string in the “data” pointer. It will assemble the words in the original order (first word at the head of the list) into a linked list. It will return a pointer to the first node in the linked list.

```
struct node_struct *search( struct node_struct *list, char *target, int (*compar)(const void *, const void *) );
```

This function will traverse the linked list given by “list”, inspecting each node in turn. For each node it will run the function “compar” with the “data” in the node as the first argument, and the “target” as the second argument. If the “compar” function returns 0 (success), then the node will be considered found. As the function proceeds through the “list” it will construct a second list for each word node that was found. In the second list, “data” will contain a pointer to the found node (not the string) from the first list. It will return the list of nodes that are found.

```
struct node_struct *copy( struct node_struct *start, struct node_struct *end );
```

This function will construct a copy of a list. The copy of the list will begin copying at the node given by “start” and finish just prior to “end” or when it reaches a NULL pointer. A shallow copy will be created, meaning that only the “data” pointers will be copied from the original list to the new list. The data contained in “start” will be the first data also contained in the copy. The

data contained in “end” will not be in the copy (but the data just before it will be). The function will return a pointer to the first node in the copy.

---

void ftext(FILE *fp, struct node_struct *list);

This function will print out the strings pointed to by the “data” pointers in the linked list in order starting at the head of the list. It will insert a white space:

- after a word that is a comma, semicolon, exclamation mark, double quotation mark, or period, provided the next word is not a double quotation mark or a double dash,
- between a word that ends with a letter or a number and another word that begins with a letter or a number,

The white space will be either a single space character or a single new-line character according to the following rule: each line will contain as many words as possible without exceeding a width of 80 characters (not including the final new-line).

---

struct node_struct *sort(struct node_struct *list, int (*compar)(const void *, const void *));

This function will sort the data in the given “list” creating a new list and returning a pointer to the head of the new list. The order of the nodes in the returned list will be in ascending order according to the comparison function “compar” which must return an integer less than, equal to, or greater than zero if the first string argument is considered to be respectively less than, equal to, or greater than the second string argument. “compar” will be passed two “data” pointers from nodes in the original list.

You will use a merge sort with lists to sort the data.

---

void remove_repeats (struct node_struct *list, int (*compar)(const void *, const void *));

This function will remove repeated entries from a list. If the “compar” function returns zero for a node’s “data” and its immediate next node’s “data”, then the second node should be removed from the list. The process should continue for any additional duplicates (which should all be removed), and onward with the next unique data. Any nodes that are removed must have the structure’s memory freed. (Since the first word will never be removed, there is no need for a double pointer as the first argument to this function.)

---

int length(struct node_struct *list);

This function returns the number of nodes in the list given by “list”.

---

void free_list(struct node_struct *list, int free_data);

This function frees the nodes in the list given by “list”. If free_data is non-zero, it will also free the data pointed to by “data”.

Submission

You will submit all of your assignment’s files by depositing them to the School’s git repository in the A2 submission directory. Submit exactly the following files:

text.c text.h

Put your full name, student ID number, and uoguelph e-mail at the top of each file.

Use consistent indenting, formatting, commenting, naming and other good programming practises to make your code readable.