Project 4
CS221: C and Systems Programming – Fall 2018

Deadline: December 7, 2018 at 11:59pm

Doubly Linked List of Sorted Strings

For this project you should write a doubly linked list program that implements `Insert`, `Member`, `Delete`, and `Free_list`, stores a list of strings, and stores the strings in increasing alphabetical order. Your program should not store duplicate strings, and it should be efficient in the following two senses:

1. It minimizes the use of memory for the list. So the individual nodes in the list only store as many characters as are actually in the string (including the terminating null character).
2. It minimizes search times by exploiting the alphabetical ordering in the list. So `Insert`, `Member`, and `Delete` only search through the list as far as needed. For example, suppose the list contains the following strings:

   ```
   cs221 hello how sanfrancisco usf
   ```

   Then a call to `Insert`, `Member`, or `Delete` with argument "hi" should only search the list as far as "how" since when we reach "how" we know that "hi" cannot be in the list. So `Member` and `Delete` can return, and `Insert` can insert "hi" in a node that precedes the node containing "how".

For this project you will be implementing a doubly linked list. So each node in the list is linked to both its immediate successor and its immediate predecessor. The first node in the list has a `NULL` predecessor pointer, and the last node in the list has a `NULL` successor pointer. Furthermore, the list is defined by two pointers: a head pointer, which references the first node in the list, and a tail pointer which references the last node in the list.

The Program

Input to the program is the commands the user enters. Note that understanding the code skeleton is part of this project, which includes how the program take the users input and how it prints the linked list. You don't need to worry about how the string library orders things like "Animal" and "animal". You can assume that the strings will consist only of the lower case letters 'a', 'b', . . . , 'z';. In order to store variable length strings, you can use the following struct definition:

```
struct list_node_s {
    char* data;
    struct list_node_s* pred_p;
    struct list_node_s* next_p;
};
```

The string that's read in by the program does not have to be "efficiently" stored. You can simply declare a fixed-length array of `char`. However, when it's copied into a node in the list, it must be stored efficiently. Any input string will not contain more than 99 characters. So when the null character ('\0') is appended, you won't need to store more than 100 characters for the input string entered by the user.

For the functions that have the following prototypes, if properly implemented, you can simply replace calls to `malloc` and `free` in any other functions in the linked list program with calls to these functions.

```
struct list_node_s* Allocate_node(int size); /* you will implement this */
void Free_node(struct list_node_s* node_p); /* provided to you */
```

The `Allocate_node` function should first allocate storage for a `struct list_node_s`. It can then allocate storage for a string of the desired length. For example, if we were allocating storage for a node that would contain the string "hello", we could do something like this.

```
struct list_node_s* temp_p;
temp_p = (struct list_node_s*) malloc(sizeof(struct list_node_s));
temp_p->data = (char *) malloc(6*sizeof(char));
```

We allocate storage for 6 chars instead of 5 so that there will be room for the terminating null character.
The `Free_node` function (that is provided to you) should first free the storage for the string, and then free the storage for the node:

```
free(node_p->data);
free(node_p);
```

It's most convenient to store the head and tail pointers in a single struct. Then a pointer to this struct can be passed as a single argument to a function. For example, you should define:

```
struct list_s {
    struct list_node_s* h_p;
    struct list_node_s* t_p;
};
```

Now in the main you can define:

```
struct list_s list;
/* Empty list has NULL h_p, t_p pointers */
list.h_p = NULL;
list.t_p = NULL;
```

And you can call the Insert function (for example) with

```
Insert(&list, value);
```

Note that with a sorted, doubly linked list, it is not necessary to use a predecessor pointer in the `Insert` or `Delete` functions, since the `pred_p` member of a node refers to the preceding node in the list. In fact, you should not declare a predecessor pointer in these or any other functions.

You may find it much easier to write the program in stages.

1. Implement this program so that it works with unsorted lists of strings.
2. Then modify the `Insert` function so that it inserts new strings in alphabetical order and doesn't insert duplicates.
3. Finally modify the `Member` and `Delete` functions so that they are "efficient" in the sense outlined in the specification.

Grading
Your code must compile and run correctly on the Linux lab machines. If we cannot compile your code on the lab machines, you will receive no credit.

Feature	Points
Insert	40
Member	15
Delete	20
Free_list (Extra Credit)	10
Allocate_node	20
Coding Style	5

Submission Guidelines:
Prior to the deadline, upload one zip file containing only two files: your `doubly_linked_list.c` source code, and status.txt, to Canvas. The ZIP file name must be in the following format: LastName_FirstName_StudentID_-proj4.zip For instance if my student ID is 123456789 and I am submitting my solution for project 4, then I am going to compress status.txt and `doubly_linked_list.c`, and rename the zip file to: Pournaghshband_Vahab_-123456789_proj4.zip Do not submit/include any other file such as the executable file. A good sanity check is to check your zip file for corruption by extracting (unzipping) it and testing whether it did compress it successfully. If we cannot unzip your submission, you will receive no credit.

Sample status.txt file:

```
Vahab Pournaghshband - Project 4 The program works as required. It compiles/runs and the
output matches the correct format to the letter. However, the style and formatting is incorrect
because I DIDNT comment it (didnt even put my name in the file).
```