CS/240/Lab/4

Linked list

This lab is the first part of a word frequency counter that analyzes the trends (i.e., the most popular topics) of tweeter feeds. For such an analysis, we first build a linked list to store the words in tweeter feeds.

In this lab, you will implement a data type `struct lnode`, which is a node in a linked list. Each node contains a word, along with an integer which stores the number of times the word has been seen (the count), and another integer which stores the last line the word was seen on. Words in this lab are sequences of alphabetical characters. For instance, `"1this is_a@test?"` contains the words `"this"`, `"is"`, `"a"`, `"test"`.

On Piazza, you'll find: `list.h` (prototypes), `list.c` (template for you to fill in), and `liblist_ref.a` (reference implementation of `liblist.a`).

How you implement your `struct lnode` is up to you. The head pointer (`struct lnode *head`) is a variable of the *caller*; as such, the functions you implement will take a pointer to the head pointer (`struct lnode **head`). The `*head` pointer should initially be `NULL`. All strings passed to or returned from these functions *must* be null-terminated. You will implement the following functions:

```
struct lnode *newNode(char* word, int line)
```

Description: Returns a pointer to a new linked list node filled in with the given word and line, and sets the count to be 1. Make sure to duplicate the word, as the original word may be modified by the calling function.

```
void pushNode (struct lnode** head, struct lnode* node)
```

Description: In a linked list with *head as the head pointer, adds the given node to the front of the list.

```
void deleteNode (struct lnode** head, struct lnode* node)
```

Description: Removes the specified node from the list, and `free`s all memory the node is using. If `*head` points to `node`, then it should be updated to point to the new list head (or NULL if the list is empty)!

```
struct lnode *nodeGetNext(struct lnode *node)
```

Description: Simply returns a pointer to the next node in the list, or `NULL` if there are no further nodes.

```
char *nodeGetWord(struct lnode *node)
```

Description: Simply returns a pointer to the word in the given node.

`int nodeGetLine(struct lnode *node)`

Description: Simply returns the line number in the given node.

`void nodeSetLine (struct lnode*, int line)`

Description: Sets the line in the node to be the given line.

`int nodeGetCount(struct lnode *node)`

Description: Simply returns the frequency count in the given node.

`void nodeSetCount(struct lnode*, int count)`

Description: Sets the frequency count in the given node.

`struct lnode* getNode(struct lnode* head, char* word)`

Description: Returns a pointer to the node containing the given word from the linked list with head as the head pointer. If a node with the given word cannot be found, the function returns NULL.

`void deleteList(struct lnode **head)`

Description: Deletes *every* node in the list pointed to by the `*head` pointer. After calling this function, all memory used by the list should be `freed`, and `*head` should become `NULL`.

Compiling & Testing

Before submitting your `list.c` to Autograder, unit test first and make sure that it functions correctly. To create liblist.a do:

```
gcc -c list.c; ar rcu liblist.a list.o
```

The command to link your `liblist.a` with your own unit testing suite in `test.c` is:

```
gcc -o test test.c -L. -llist
```

The command to link our reference `liblist_ref.a` with your own unit testing suite in `test.c` is:

```
gcc -o test test.c -L. -llist_ref
```

If Autograder fails your code, please print out the linked list to check the results.

Turning in

Go to the submission web page, under "currently open projects" will be listed "lab 4 list". Click "lab 4 list", use the file selector to choose your `list.c` file, then click "submit".

Lab is due Monday, February 18th before midnight. No late labs accepted.

Grading criteria

List

- source file is named `list.c`

- code compiles and runs without error
- newNode must create a new node with the specified value of word and line at the head of the list. The word string to store must be a copy of the passed one.
- pushNode must insert the supplied node at the front of the linked list.
- getNode must return a pointer to the requested node if there is one in the list. getNode must return NULL otherwise.
- nodeGetNext, nodeGetWord, nodeGetCount, nodeSetCount, nodeGetLine and nodeSetLine must work as described.
- deleteNode must work on all cases when the node to delete is head, or tail, or in the middle of the list. You can assume the node to delete is always one on the list. All memory belongs to the node must be freed, which includes the word string and the node itself.
- deleteList: after called, the list must be empty, and all memory belongs to the list must be freed.