CSCI 312-01, Fall 2025
A doubly linked list in C

Project description

In this project you will implement a doubly linked list in C with bare-bones functionality. There are two parts, the public interface, which I specify, and the rest of the implementation, where you are free to express yourself.

The public interface

Here is the public interface for the list. Do not change the function signatures (i.e., the type of function, the types of its arguments, and the order of the arguments) since testing will assume this interface. Also, do not change the code defining the `NODE` and `LIST` types.

list.h

```
/* DO NOT CHANGE THIS FILE! */

#ifndef __LIST_H__
#define __LIST_H__

typedef struct node {
  char *value;             // Pointer to the string stored in this node.
  struct node *previous;   // Pointer to the preceding node in the list.
  struct node *next;       // Pointer to the next node in the list.
} NODE;

typedef struct list {
  NODE *head;  // Pointer to the first node in the list.
  NODE *tail;  // Pointer to the last node in the list.
} LIST;

// Function prototypes of the public interface.
LIST *new_list(const char* const value);
void prepend(LIST* const list, const char* const value);
void append(LIST* const list, const char* const value);
void delete_list(LIST *list);
void print_list(const LIST* const list);

#endif
```

`LIST *new_list(const char* const value)`

This function

- creates a new list,
- creates a node in the new list that stores the string `value`,
- updates the `head` and `tail` pointers in the list, and
- returns a pointer to the list.

`void prepend(LIST* const list, const char* const value)`

This function adds a new node at the head of the list. It

- creates a new node that contains the string `value`,
- sets the appropriate pointers in the new head and the old head, and
- updates the `head` pointer in the list.

`void append(LIST* const list, const char* const value)`

This function adds a new node at the tail of the list. It

- creates a new node that contains the string `value`,
- sets the appropriate pointers in the new tail and the old tail, and
- updates the `tail` pointer in the list.

`void delete_list(LIST *list)`

This function deletes a list and frees all dynamically allocated memory associated with it.

Creating a new node

When you create a new node, you will need to perform the following operations:

1. Allocate space to hold the node.
2. Allocate space in the node (pointed to by `char *value`) to hold the string.
3. Copy the string into the space just allocated. You can create copies of strings using the `strdup()` function (you'll need to include `string.h` to use `strdup()`).

   The `strdup()` function has the signature

   `char *strdup(const char *src),`

   where `src` is the string we are cloning. It returns a pointer to a copy of its input.
4. Set the `next` and `previous` pointers.

Marking the ends of a list

You should use `nullptr` to indicate the termini of a list. That is, if `list` is our list, then `list->head->previous` and `list->tail->next` should both be `nullptr`.

What to do

Create your functions in a file named `list.c` Submit this file and a corresponding object file `list.o`, compiled under Linux for an Intel x86-64, to the autograder on Gradescope for grading. Do not submit `list.h`.

Working on the CS system

I have built the most recent versions of two compilers, `clang` 20 and `gcc` 15, on the CS system. They reside in `~rml/llvm/bin` and `~rml/gcc/bin`, respectively.

For convenience you can add these directories to your `PATH` shell variable. When you try to execute a Linux command, the shell searches the directories listed in your `PATH` environment variable for an executable of the same name.[1] By adding directories to your `PATH` you can just enter commands like `clang` and `gcc` without having to use their full pathnames.

To modify your `PATH`, you will edit the file `~/.bashrc`.[2] Add the following to the end of the file:

```
PATH=~rml/llvm/bin:~rml/gcc/bin:$PATH
export PATH
```

Very important: do not leave spaces around the equals sign.

To have this change have immediate effect, use the command

```
source ~/.bashrc
```

The `source` command tells the shell to read and execute the shell commands in your `.bashrc` file. Future shells will read `.bashrc` at start-up and execute the change in `PATH` for you.

[1] Actually, the shell creates a hash table of commands and looks up commands there.

[2] The tilde is *nix shorthand for your home directory.