ECE 223 Programming Assignment 2

Linked List Program

Construct a double-linked list package.
Use the double-linked list to implement a guitar database
The guitar record itself is a datatype with its own header file.

The user interface of the main program (lab2.c) should be extended to include the following commands:

ADD, DELETE, LOOKUP, FIRST, LAST, NEXT, PREV, LIST, HELP, QUIT

The program should FIRST read one of these commands from the command line, and THEN prompt the user for any required arguments one at a time. The program should store the guitars in the list in order sorted on the id number (key)

```
/*-------------------------*/
/* guitar.h                */
```

#if 0
The guitardb_guitar should now be a first class ADT, including a compare function for use in “insert_after,” “find” and “remove.” Return 0 or non-NULL on success and -1 or NULL on failure.
#endif

```
typedef int key_t;

typedef enum guitartype_e
{
      S = 'S' /* solid body */,
      A = 'A' /* arch-top hollow body */,
      T = 'T' /* semi-hollow (thinline) */
} guitartype_t

typedef enum pickup_e
{
      H = 'H' /* humbucker */,
      C = 'C' /* single coil */,
      P = 'P' /* P90 */,
      F = 'F' /* Filtertron */,
      N = 'N' /* none */
} pickup_t;

typedef struct guitar_s {
      key_t id_number;
      char make[20];
      char model [20];
      char submodel[15];
      guitartype_t gtype;
      int year;        /* year of manufacture */
      char finsh[25];  /* description of finish incuding color,
                          binding, etc. */
      int strings;     /* number of strings - usually 6 or 7 */
      int pickups;     /* number of pickups */
      pickup_t neck;   /* pickup type */
      pickup_t middle; /* pickup type */
      pickup_t bridge; /* pickup type */
} *guitar_t;

/* allocate a new struct guitar_s and return a pointer to it */
guitar_t guitar_init(void);

/* read from user all fields for a guitardb_guitar */
int guitar_fill(guitar_t);

/* set the id of a guitardb_guitar */
int guitar_setid(guitar_t, key_t);

/* get the id of a guitardb_guitar */
key_t guitar_getid(guitar_t);
```

```
/* prints a guitar */
int guitar_print(guitar_t);

/* returns 0 if equal, <0 if less than, >0 if greater than */
int guitar_compare(guitar_t, guitar_t);

void guitar_free(guitar_t);



/*-------------------------*/
/* guitardb.h              */

#if 0
This is the guitar database – it may call functions from the guitar
module or from the linked list module.  To add a guitar, first call
guitar_init to get a new record, fill it in with guitar_fill, then call
guitardb_add to insert it into the database – which uses the linked
list rougtines to store it.
#endif

#include “guitar.h”

typedef struct guitardb_s
{
      /* fill this in as needed */
} *guitardb_t;

/* initialize a new guitar database */
guitardb_t guitardb_init(void);
/* add a guitar to the database */
key_t guitardb_add(guitardb_t, guitar_t);
/* lookup a guitar by id return a pointer to it */
guitar_t guitardb_lookup(guitardb_t, key_t);
/* delete guitar from database, return pointer to it, do not free */
guitar_t guitardb_delete(guitardb_t, key_t);
.* reports all of the guitars in the database with it’s key, make,
model, and year, etc. *;
void guitardb_report(guitardb_t);
/* free all resources used by the guitar database, remove everything
from the linked list and free all pointers there before finalizing the
list */
void guitardb_finalize(guitardb_t);
```

```
/*-------------------------*/
/* list.h                  */

#if 0
```
Here is the header for the list ADT. Note all functions are in terms to data_t. This must use a double-linked list. You code must accept as many entries as possible until memory is exhausted. Your program must not have a memory leak – meaning that at no time may data you have allocated be inaccessible or put another way, you must always call free() on any memory your program will no longer keep up with. Find, first, next, prev, and last return NULL if the requested item doesn't exist.
```
#endif

typedef struct list *list_t;
typedef void *data_t;
typedef int (*cmpfunc)(data_t, data_t);

/* create a new empty list */
list_t list_init();
/* insert at head of list */
int list_insert(list_t, data_t);
/* append to tail of list */
int list_append(list_t, data_t);
/* find and sets current item using callback compare function */
data_t list_find(list_t, data_t, cmpfunc cmp);
/* return item at head of list, set current item */
data_t list_first(list_t);
/* return next item after current item */
data_t list_next(list_t);
/* return prev item before current item */
data_t list_prev(list_t);
/* return item at tail of list, set current item */
data_t list_last(list_t);
/* before current item */
int list_insert_before(list_t, data_t);
/* after current item */
int list_insert_after(list_t, data_t);
/* remove current item */
int list_remove(list_t);
/* free all resources allocated by the list */
int list_finalize(list_t);

/*-------------------------*/
```