```c
// Name: Rodrigo Ignacio Rojas Garcia
// Course Number: ECE 2230
// Section: 001
// Semester: Spring 2017
// Assignment Number: 2

// Library Declartion Section
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "guitar.h"
#include "guitardb.h"
#include "list.h"

typedef struct node
{
    struct node *next;
    struct node *previous;
    data_t item;
} *node_t;

typedef struct list
{
    node_t head;
    node_t tail;
    node_t current;
} *list_t;

// Allocates dynamic memory for a new list which and will set the addresses of poin
ter head, current, and tail to NULL since there is no data in them yet
list_t list_init()
{
   list_t new_list;
   new_list = (list_t)calloc(1,sizeof(struct list));
   new_list->head = NULL;
   new_list->current = NULL;
   new_list->tail = NULL;
   return new_list;
}

// Function list_insert will fisrt check if there a database has been created on th
e list, if not it will create it and set the item passed in the function
// to it, and set the head, tail, and current pointers of the database to the new d
atabase. If not true, it will check if the database head and the tail pointer
// point to the same database, meaning one item of the whole list, then it will mak
e sure to set the pervious of the tail to the new head and set the new head next
// to point to the tail which is the database next to it. If none of this are true,
 it means that there is more items on the list and a new database will be
// created and become the head, this will change the old head previous to point to
the new head and the new head next to point to the old head
int list_insert(list_t database, data_t item)
{
    // If first time adding an item to the database, this if statemenet will run
    if (database->head == NULL)
    {
        // Creates a new node
        node_t node;
        // Allocates dynamic memory for the new node
        node = (node_t)calloc(1,sizeof(struct node));
        // Sets item passed to function to the new node created
        node->item = item;
        // Sets the previous in the new node to NULL signifyingt only item on datab
ase
        node->previous = NULL;
        // Sets the next in the new node to NULL signifying only item on the databa
se
        node->next = NULL;
        // Sets head, tail, and current to point to the only node on the database
        database->head = node;
        database->tail = node;
        database->current = node;
        return 0;
    }
    else if (database->head == database->tail)
    {
        // Creates a new node
        node_t node;
        // Allocates dynamic memory for the new node created
        node = (node_t)calloc(1, sizeof(struct node));
        // Sets the item passed by the list_insert function into the item of the no
de created
        node->item = item;
        // Sets the previous pointer to NULL signifying that this will become the n
ew head
        node->previous = NULL;
        // Sets the next pointer of the new node created to the database that point
er head is pointing to
        node->next = database->head;
        // Sets the previous pointer of the pointer tail to the new node created
        database->tail->previous = node;
        // Sets the head pointer in the database to point to the new node created
        database->head = node;
        // Sets the current pointer in the database to point to the new node create
d
        database->current = node;
        return 0;
    }
    else
    {
        // Creates a new node
        node_t node;
        // Allocates dynamic memory for the new node created
        node = (node_t)calloc(1, sizeof(struct node));
        // Sets the item passed by the list_insert function into the item of the no
de created
        node->item = item;
        // Sets the previous pointer of the new node created to NULL signifying tha
t this new node will become the new head
        node->previous = NULL;
        // Sets the next pointer of the new node created to the databse that pointe
r head is pointing to
        node->next = database->head;
        // Sets the head pointer of the database to the new node created, meaning t
hat this new node is the new head
        database->head = node;
        // Sets the current cursor pointer of the database to the new node to keep
track
        database->current = node;
        return 0;
    }
}

// Function list_append will first check if a database has been added to the list,
if not, it will create a new database and set the head, current, and tail
// pointer to it. If not true, it means that a new database will be created after t
he current tail, and the curren tail next pointer will point to the new
// database, the new database previous will point to the current tail and the next
to NULL, and the tail pointer will point to the new database, making it
// the new tail
int list_append(list_t database, data_t item)
{
    // If first time adding an item to the database, this if statemenet will run
    if (database->head == NULL)
    {
```

```
        // Creates a new node
        node_t node;
        // Allocates dynamic memory for the new node
        node = (node_t)calloc(1,sizeof(struct node));
        // Sets item passed to function to the new node created
        node->item = item;
        // Sets the previous in the new node to NULL signifyingt only item on datab
ase
        node->previous = NULL;
        // Sets the next in the new node to NULL signifying only item on the databa
se
        node->next = NULL;
        // Sets head, tail, and current to point to the only node on the database
        database->head = node;
        database->tail = node;
        database->current = node;
    }

    else
    {
        // Creates a new node
        node_t node;
        // Allocates dynamic memory for the new node
        node = (node_t)calloc(1, sizeof(struct node));
        // Sets the item passed by the list_append function to the new node's item
pointer
        node->item = item;
        // Sets the next pointer in the new node created to NULL signifying that th
is will become the new tail
        node->next = NULL;
        // Sets the previous pointer of the new node created to point to same datab
ase that the current cursors is pointing to
        node->previous = database->current;
        // Sets the database next pointer of the current cursors that is ponting to
 to the new node
        database->current->next = node;
        // Sets the current cursor pointer of the database to the new node to keep
track
        database->current = node;
        // Sets the tail pointer of the database to point to the new node created,
which means that his new node becomes the tail
        database->tail = node;
        return 0;
    }
    return 0;
}

// Function list_find first declares a void pointer called temp_item which will cal
l the list_first function which will return the first item on the database or it wi
ll return a NULL
// If the function returns the first item, then it will go into a while loop compar
ing the id numbers of the item passed in the list_find function to the item that li
st_first returned
// If both have the same id_number, then it will return the address of the item, if
 not it will keep looking. If there was never an item on the database, then functio
n returns
// a NULL
data_t list_find(list_t database, data_t item, cmpfunc cmp)
{
    data_t temp_item;
    temp_item = list_first(database);
    while(temp_item != NULL)
    {
        if (cmp(temp_item, item) == 0)
        {
            return temp_item;
        }
        else
        {
            temp_item = list_next(database);
        }
    }
    return NULL;
}

// Function list_first will check if the head database pointer is pointing to an it
em on the list, if truye, then it will return the address of it's item, if false re
utrns a NULL
data_t list_first(list_t database)
{
    if (database->head != NULL)
    {
        if (database->head->item != NULL)
        {
            database->current = database->head;
            return database->head->item;
        }
        else
        {
            return NULL;
        }
    }
    else
    {
        return NULL;
    }
}

// Function list_next will check if the current cursors pointer of the database is
pointing to an item, if so, it will return the item next to it's location, if false
// it will return a NULL
data_t list_next(list_t database)
{
    if (database->current != NULL)
    {
        if (database->current->next != NULL)
        {
            if (database->current->next->item != NULL)
            {
                database->current = database->current->next;
                return database->current->item;
            }
            else
            {
                return NULL;
            }
        }
        else
        {
            return NULL;
        }
    }
    else
    {
        return NULL;
    }
}

// Function list_prev will first check if the current cursor is pointing to an item
 in the database, if true, then it will check it there is an item before it's locat
ion, if that is
// also true, then it will check if that database has an item, if true, it returns
the address of the item, if false returns NULL
data_t list_prev(list_t database)
```

```
{
    if (database->current != NULL)
    {
        if (database->current->previous != NULL)
        {
            if (database->current->previous->item != NULL)
            {
                database->current = database->current->previous;
                return database->current->item;
            }
            else
            {
                return NULL;
            }
        }
        else
        {
            return NULL;
        }
    }
    else
    {
        return NULL;
    }
}

// Function list_last will first check if the tail pointer of the database is poitn
ing to a database, if true, it will check if there is an item stored in it, if this
 also
// true, it will return the item located at the tail of the database, if not true,
it returns a NULL
data_t list_last(list_t  database)
{
    if (database->tail != NULL)
    {
        if (database->tail->item != NULL)
        {
            database->current = database->tail;
            return database->tail->item;
        }
        else
        {
            return NULL;
        }
    }
    else
    {
        return NULL;
    }
}

// Function list_inser_before will first check if the head pointer of the database
is pointing to a database, if not true, it will allocate memory for a node
// and will store the item passed in the function in that database. If there has be
en already been an item stored on the database, then it will check if the
// current cursors pointer and the head pointer of the database point to the same t
hing, if true it will create a new node and set the item there and become the new h
ead
// Lastly, the else statement determined what happens when insreting before an item
 that is not the head, which will create a new node and store the item before it
// as well as changing the previous and next pointers of the items that were before
 and after the new node created betweem them.
int list_insert_before(list_t database, data_t item)
{
    // If there has no item been inserted on the database, this if statement will r
un and will allocate memory for the databse and store the item
    // on the database
    if (database->head == NULL)
    {
        list_append(database, item);
        return 0;
    }
    // If there is data in the list, and the item entered has an ID less than the c
urrent head, then this will add the item to the databse and
    // set the head to this new item entered.
    else if (database->current == database->head)
    {
        list_insert(database, item);
        return 0;
    }
    else
    {
        // Creates a new node
        node_t node;
        // Allocates memory for a new node
        node = (node_t)calloc(1, sizeof(struct node));
        // Sets the new node's item to the item passed by the function
        node->item = item;
        // Sets the node's next pointer to the current database
        node->next = database->current;
        // Sets the node's previous to the previous database of the current databas
e
        node->previous = database->current->previous;
        // Sets the database pointer next of the database before the current databa
se to the new database
        database->current->previous->next = node;
        // Sets the previous pointer of the current database to the new database cr
eated
        database->current->previous = node;
        database->current = node;
        return 0;
    }

}

// Function list_insert_after will firs check if there has been a databse added to
the list, if not it will create one and store the item passed in the function
// to it with list_append. If not true, it will check if the current cursor pointer
 of the database is pointing where the head pointer of the database is pointing
// as well, if true it will create a new database and store the item passed in the
function to it and make that new database the new head. If neither of those true,
// it will create a new database, set the item passed in the function to it, and ch
ange the pointers previous and next from the databases after and before them accord
ingly
int list_insert_after(list_t database, data_t item)
{
    // If there has no item been inserted on the database, this if statement will r
un and will allocate memory for the databse and store the item
    // on the database
    if (database->head == NULL)
    {
        list_append(database, item);
        return 0;
    }
    // If there is data in the list, and the item entered has an ID less than the c
urrent head, then this will add the item to the databse and
    // set the head to this new item entered.
    else if (database->current == database->head)
    {
        list_insert(database, item);
        return 0;
    }
    else
    {
```

```
        // Creates a new dode
        node_t node;
        // Allocates memory for a new node
        node = (node_t)calloc(1,sizeof(struct node));
        // Sets the new node's item to the item passed in the function
        node->item = item;
        // Sets the new node's previous pointer to the current cursors pointer data
base
        node->previous = database->current;
        // Sets the new node's next to the current cursor next pointer which is the
 database next to the current database
        node->next = database->current->next;
        // Sets the current cursor next database to point to the new node databse
        database->current->next = node;
        // Sets the current cursor next database previous pointer to point to the n
ew node database
        database->current->next->previous = node;
        // Sets the current cursor pointer to the new node to keep track
        database->current = node;
        return 0;
    }
}

// Function list_remove will first check that the cursor current pointer of the dat
abse is not NULL, if true it will check if the current cursor and the head pointer
are
// pointing to the same database, if true, it will check if there is an database ne
xt to it, if true it will change the head pointer to the database next to it, free
the
// database and move the current to the next database. If not true, it means that t
here is only one item on the database, therefore everything set to NULL and databas
e is freed.
// The second case if is the current cursors pointer of the database is pointing to
 the same database as the tail. If true it will check if there is an item previous
to it, if true
// it will change the tail position to the item before it, set everything needed to
 NULL, free the database, and change the current cursors to the database before. If
 not true,
// it will set everything to NULL and free the database. The last case is if there
database that wants to be removed is between to databases, if true, it will change
the next
// of previous database and the previous of the next database to point to each othe
r, free the current databse, and change the current cursors to NULL.
int list_remove(list_t database)
{
    // Checks that the current cursor of the database is pointing to a database
    if (database->current != NULL)
    {
        // Checks if the current cursors of the database is pointing to the same da
tabase as the head, if true, it will check if there is an item next to the head, if
 this
        // true it will change the head to the database next to it, set the previou
s of the item next to it to NULL, free database, and change the current cursors poi
nter
        // to the new head. If not true, it means that it is the only item on the l
ist, therefore everything is set to NULL and database is freed.
        if (database->current == database->head)
        {
            if (database->current->next != NULL)
            {
                database->head = database->current->next;
                database->current->next->previous = NULL;
                free(database->current);
                database->current = database->head;
                return 0;
            }
            else
            {
                database->current->next = NULL;
                database->current->previous = NULL;
                database->current->item = NULL;
                free(database->current);
                database->current = NULL;
                database->head = NULL;
                database->tail = NULL;
                return 0;
            }
        }
        // Checks if the current cursors of the database is pointing to the same da
tabase as the tail, if true, it will check if there is an item before the tail, if
        // this is true, it will change the tail to the database before it, set the
 next of the previous database to NULL, free the database, and change the current c
ursor pointer to the
        // the new tail. If not true, it means that it is the only item on the list
, therefore everything is set to NULL and database is freed.
        else if (database->current == database->tail)
        {
            if (database->current->previous != NULL)
            {
                database->tail = database->current->previous;
                database->tail->next = NULL;
                database->current->item = NULL;
                free(database->current);
                database->current = database->tail;
                return 0;
            }
            else
            {
                database->current->next = NULL;
                database->current->previous = NULL;
                database->current->item = NULL;
                free(database->current);
                database->current = NULL;
                database->head = NULL;
                database->tail = NULL;
                return 0;
            }
        }
        // If none of the other if statements are true, this means that it is erasi
ng a database between two other databases. In this case it will change the previous
 and the next of the
        // database next to it and the database before the current database accordi
ngly. Then, it will free the current database and set the current cursors database
to NULL.
        else
        {
            database->current->next->previous = database->current->previous;
            database->current->previous->next = database->current->next;
            database->current->item = NULL;
            free(database->current);
            database->current = NULL;
            return 0;
         }
    }
    return -1;
}

// Function list_finalize will create two temporary node_t variabes to free the dat
abases that have been allocated with memory until it reaches
// a NULL, meaning that there is no more items in the list. At the end it will free
 the whole list
int list_finalize(list_t database)
{
    node_t current;
```

```
    node_t next;
    next = database->head;
    while(next != NULL)
    {
        current = next;
        next = next->next;
        free(current);
    }
    free(database);
    return 0;
}
```