```
/* list.c
 * Christopher Brant
 * cbrant
 * ECE 2230
 * Section 001
 * Spring 2017
 * Programming Assignment #2
 * Due on 2/15/17 at 11:59 PM
 * Professor Walt Ligon
 */

#include <stdio.h>
#include <stdlib.h>
#include "list.h"

struct list_node {
        struct list_node *head;
        struct list_node *tail;
        data_t node_info;
};

struct list {
        int list_size;
        struct list_node *current;
        struct list_node *top;
        struct list_node *bottom;
};

/* Create a new empty list */
list_t list_init(void)
{
        list_t new_list = (list_t)malloc(sizeof(struct list));

        new_list->list_size = 0;
        new_list->current = NULL;
        new_list->top = NULL;
        new_list->bottom = NULL;

        return new_list;
}

/* Insert if list is empty */
static void list_empty_insert(list_t lpoint, data_t dpoint, struct list_node *insert)
{
        struct list_node *add_node = insert;
        add_node->node_info = dpoint;
        lpoint->current = add_node;
        lpoint->top = add_node;
        lpoint->bottom = add_node;
        lpoint->current->head = NULL;
        lpoint->current->tail = NULL;
        lpoint->list_size++;
}

/* Insert at head of list */
int list_insert(list_t lpoint, data_t dpoint)
{
        int success = 1;
        struct list_node *new_node = (struct list_node *)malloc(sizeof(struct list_node));

        if (new_node == NULL)
                return 0;
        if (lpoint->list_size == 0)
                list_empty_insert(lpoint, dpoint, new_node);
        else
        {
                new_node->node_info = dpoint;

                lpoint->top->head = new_node;
                new_node->tail = lpoint->top;
                new_node->head = NULL;
                lpoint->top = new_node;
                lpoint->list_size++;
        }

        return success;
}

/* Append to tail of list */
int list_append(list_t lpoint, data_t dpoint)
{
        int success = 1;
        struct list_node *new_node = (struct list_node *)malloc(sizeof(struct list_node));

        if (new_node == NULL)
                return 0;

        if (lpoint->list_size == 0)
                list_empty_insert(lpoint, dpoint, new_node);
        else
        {
                new_node->node_info = dpoint;

                lpoint->bottom->tail = new_node;
                new_node->head = lpoint->bottom;
                new_node->tail = NULL;
                lpoint->bottom = new_node;
                lpoint->list_size++;
        }

        return success;
}

/* Find and sets current item using callback compare function */
data_t list_find(list_t lpoint, data_t dpoint, cmpfunc cmp)
{
        int i, found = 2;
        data_t match;

        lpoint->current = lpoint->top;

        for (i = 0; i < lpoint->list_size && found != 0 && lpoint->current != NULL; i++)
        {
                found = (*cmp)(lpoint->current->node_info, dpoint);

                if (found != 0)
                        lpoint->current = lpoint->current->tail;
        }

        if (found == 0)
                match = lpoint->current->node_info;
        else
                match = NULL;

        return match;
```

```
}

// Return item at head of list, set current item
data_t list_first(list_t lpoint)
{
        data_t first_node = lpoint->top->node_info;
        lpoint->current = lpoint->top;

        return first_node;
}

// Return next item after current item
data_t list_next(list_t lpoint)
{
        data_t next_node = NULL;

        if (lpoint->list_size > 1 && lpoint->current->tail != NULL)
        {
                next_node = lpoint->current->tail->node_info;
                lpoint->current = lpoint->current->tail;
        }

        return next_node;
}

// Return prev item before current item
data_t list_prev(list_t lpoint)
{
        data_t prev_node = NULL;

        if (lpoint->list_size > 1 && lpoint->current->head != NULL)
        {
                prev_node = lpoint->current->head->node_info;
                lpoint->current = lpoint->current->head;
        }

        return prev_node;
}

// Return item at tail of list, set current item
data_t list_last(list_t lpoint)
{
        data_t last_node = lpoint->bottom->node_info;
        lpoint->current = lpoint->bottom;

        return last_node;
}

// Insert item before current item
int list_insert_before(list_t lpoint, data_t dpoint)
{
        int success = 1;

        struct list_node *new_node = (struct list_node *)malloc(sizeof(struct list_node));

        if (new_node == NULL)
                return 0;

        if (lpoint->list_size == 0)
                list_empty_insert(lpoint, dpoint, new_node);
        else
        {
                new_node->node_info = dpoint;
                new_node->head = lpoint->current->head;
                new_node->tail = lpoint->current;
                new_node->head->tail = new_node;
                new_node->tail->head = new_node;
                lpoint->current = new_node;
                lpoint->list_size++;
        }

        return success;
}

// Insert item after current item
int list_insert_after(list_t lpoint, data_t dpoint)
{
        int success = 1;

        struct list_node *new_node = (struct list_node*)malloc(sizeof(struct list_node));

        if (new_node == NULL)
                return 0;

        if (lpoint->list_size == 0)
                list_empty_insert(lpoint, dpoint, new_node);
        else
        {
                new_node->node_info = dpoint;

                new_node->tail = lpoint->current->tail;
                new_node->head = lpoint->current;
                new_node->tail->head = new_node;
                new_node->head->tail = new_node;
                lpoint->current = new_node;
                lpoint->list_size++;
        }

        return success;
}

// Remove current item
int list_remove(list_t lpoint)
{
        int success = 1;
        struct list_node *removed = lpoint->current;

        if (lpoint->list_size > 1)
        {
                if (lpoint->current->tail == NULL)
                {
                        lpoint->current = removed->head;
                        lpoint->current->tail = NULL;
                        removed->head = NULL;
                        removed->tail = NULL;
                        lpoint->bottom = lpoint->current;
                }

                else if (lpoint->current->head == NULL)
                {
                        lpoint->current = removed->tail;
                        lpoint->current->head = NULL;
                        removed->head = NULL;
                        removed->tail = NULL;
                        lpoint->top = lpoint->current;
                }

                else
                {
                        lpoint->current = removed->head;
                        lpoint->current->tail = removed->tail;
                        removed->tail->head = lpoint->current;
                        removed->head = NULL;
                        removed->tail = NULL;
                }

                lpoint->list_size--;

                free(removed);
                removed = NULL;

                if (removed != NULL)
                        success = 0;
        }

        else if (lpoint->list_size == 1)
        {
                lpoint->current = NULL;
                lpoint->top = NULL;
                lpoint->bottom = NULL;

                lpoint->list_size--;

                free(removed);
                removed = NULL;

                if (removed != NULL)
                        success = 0;
        }

        return success;
}

// Free all resources allocated by the list
int list_finalize(list_t lpoint)
{
        int i, success = 1;
        struct list_node *removed;

        removed = lpoint->top;

        for (i = 0; i < lpoint->list_size-1 && removed != NULL; i++)
        {
                lpoint->top = lpoint->top->tail;
                lpoint->top->head = NULL;
                removed->tail = NULL;
                free(removed);

                removed = lpoint->top;
        }

        free(removed);

        if (lpoint->top != lpoint->bottom)
                success = 0;

        lpoint->top = NULL;
        lpoint->bottom = NULL;
        lpoint->current = NULL;
        lpoint->list_size = 0;

        free(lpoint);
        lpoint = NULL;

        return success;
}
```