```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#include "list.h"

/* interface functions */
list_t* create_list(void)
{
    return get_node(0);
}

status_t insert_start(list_t* p_list, data_t new_data)
{
    generic_insert(p_list, get_node(new_data), p_list->next);
    return (SUCCESS);
}

status_t insert_end(list_t* p_list, data_t new_data)
{
    node_t* p_run = NULL;

    p_run = p_list;
    while(p_run->next != NULL)
        p_run = p_run->next;
    generic_insert(p_run, get_node(new_data), p_run->next);
    return (SUCCESS);
}

status_t insert_after(list_t* p_list, data_t e_data, data_t new_data)
{
    node_t* p_enode = NULL;

    p_enode = search_node(p_list, e_data);
    if(p_enode == NULL)
        return (LIST_DATA_NOT_FOUND);
    generic_insert(p_enode, get_node(new_data), p_enode->next);
    return (SUCCESS);
}

status_t insert_before(list_t* p_list, data_t e_data, data_t new_data)
{
    node_t* p_enode = NULL;

    p_enode = search_node(p_list, e_data);
    if(p_enode == NULL)
        return (LIST_DATA_NOT_FOUND);
    generic_insert(p_enode->prev, get_node(new_data), p_enode);
    return (SUCCESS);
}

status_t get_start(list_t* p_list, data_t* p_start_data)
{
    if(is_empty(p_list) == TRUE)
        return (LIST_EMPTY);
    *p_start_data = p_list->next->data;
    return (SUCCESS);
}

status_t get_end(list_t* p_list, data_t* p_end_data)
{
    node_t* p_run = NULL;

    if(is_empty(p_list) == TRUE)
        return (LIST_EMPTY);

    p_run = p_list;
    while(p_run->next != NULL)
        p_run = p_run->next;
    *p_end_data = p_run->data;

    return (SUCCESS);
}

```

```c
status_t pop_start(list_t* p_list, data_t* p_start_data)
{
    if(is_empty(p_list) == TRUE)
        return (LIST_EMPTY);
    *p_start_data = p_list->next->data;
    generic_delete(p_list->next);
    return (SUCCESS);
}

status_t pop_end(list_t* p_list, data_t* p_end_data)
{
    node_t* p_run = NULL;

    if(is_empty(p_list) == TRUE)
        return (LIST_EMPTY);

    p_run = p_list;
    while(p_run->next != NULL)
        p_run = p_run->next;

    *p_end_data = p_run->data;
    generic_delete(p_run);
    return (SUCCESS);
}

status_t remove_start(list_t* p_list)
{
    if(is_empty(p_list) == TRUE)
        return (LIST_EMPTY);
    generic_delete(p_list->next);
    return (SUCCESS);
}

status_t remove_end(list_t* p_list)
{
    node_t* p_run = NULL;

    if(is_empty(p_list) == TRUE)
        return (LIST_EMPTY);

    p_run = p_list;
    while(p_run->next != NULL)
        p_run = p_run->next;
    generic_delete(p_run);

    return (SUCCESS);
}

status_t remove_data(list_t* p_list, data_t r_data)
{
    node_t* p_node = NULL;

    p_node = search_node(p_list, r_data);
    if(p_node == NULL)
        return (LIST_DATA_NOT_FOUND);
    generic_delete(p_node);

    return (SUCCESS);
}

status_t is_empty(list_t* p_list)
{
    return (p_list->next == NULL && p_list->prev == NULL);
}

status_t find(list_t* p_list, data_t f_data)
{
    node_t* p_node = NULL;

    p_node = search_node(p_list, f_data);
    return (p_node != NULL);
}
```

```c
len_t get_length(list_t* p_list)
{
    node_t* p_node = NULL;
    len_t len = 0;

    for(p_node = p_list->next; p_node != NULL; p_node = p_node->next, ++len)
        ;
    return (len);
}

void show(list_t* p_list, const char* msg)
{
    node_t* p_run = NULL;

    if(msg)
        puts(msg);

    printf("[START]<->");
    p_run = p_list->next;
    while(p_run != NULL)
    {
        printf("[%d]<->", p_run->data);
        p_run = p_run->next;
    }
    puts("[END]");
}

status_t destroy_list(list_t** pp_list)
{
    node_t* p_run = NULL;
    node_t* p_run_next = NULL;
    list_t* p_list = NULL;

    p_list = *pp_list;
    p_run = p_list->next;
    while(p_run != NULL)
    {
        p_run_next = p_run->next;
        free(p_run);
        p_run = p_run_next;
    }

    free(p_list);
    p_list = NULL;
    *pp_list = NULL;

    return (SUCCESS);
}

/* concat immuable */
list_t* concat_lists_imm(list_t* p_list_1, list_t* p_list_2)
{
    list_t* p_concat_list = NULL;
    node_t* p_run = NULL;

    p_concat_list = create_list();
    for(p_run = p_list_1->next; p_run != NULL; p_run = p_run->next)
        assert(insert_end(p_concat_list, p_run->data) == SUCCESS);

    for(p_run = p_list_2->next; p_run != NULL; p_run = p_run->next)
        assert(insert_end(p_concat_list, p_run->data) == SUCCESS);

    return (p_concat_list);
}
```

```c
/* concat mutable */
status_t concat_list_m(list_t* p_list_1, list_t** pp_list_2)
{
    list_t* p_list_2 = NULL;
    node_t* p_run = NULL;

    p_list_2 = *pp_list_2;
    if(is_empty(p_list_2) == TRUE)
    {
        free(p_list_2);
        p_list_2 = NULL;
        *pp_list_2 = NULL;
        return (SUCCESS);
    }

    p_run = p_list_1;
    while(p_run->next != NULL)
        p_run = p_run->next;

    p_run->next = p_list_2->next;
    p_list_2->next->prev = p_run;

    free(p_list_2);
    p_list_2 = NULL;
    *pp_list_2 = NULL;

    return (SUCCESS);
}

/* merge sorted lists */
list_t* merge_lists(list_t* p_list_1, list_t* p_list_2)
{
    node_t* p_run_1 = NULL;
    node_t* p_run_2 = NULL;
    list_t* p_merged_list = NULL;

    p_merged_list = create_list();
    p_run_1 = p_list_1->next;
    p_run_2 = p_list_2->next;

    while(TRUE)
    {
        if(p_run_1 == NULL)
        {
            while(p_run_2 != NULL)
            {
                assert(insert_end(p_merged_list, p_run_2->data) == SUCCESS);
                p_run_2 = p_run_2->next;
            }
            break;
        }

        if(p_run_2 == NULL)
        {
            while(p_run_1 != NULL)
            {
                assert(insert_end(p_merged_list, p_run_1->data) == SUCCESS);
                p_run_1 = p_run_1->next;
            }
            break;
        }

        if(p_run_1->data <= p_run_2->data)
        {
            assert(insert_end(p_merged_list, p_run_1->data) == SUCCESS);
            p_run_1 = p_run_1->next;
        }
        else
        {
            assert(insert_end(p_merged_list, p_run_2->data) == SUCCESS);
            p_run_2 = p_run_2->next;
        }
    }

    return (p_merged_list);
}
```

```c
/* reversal */
/* immutable */
list_t* get_reversed_list(list_t* p_list)  /* immutable version */
{
    list_t* p_reversed_list = NULL;
    node_t* p_run = NULL;

    p_reversed_list = create_list();
    if(is_empty(p_list))
        return (p_reversed_list);

    p_run = p_list->next;
    while(p_run != NULL)
    {
        assert(insert_start(p_reversed_list, p_run->data) == SUCCESS);
        p_run = p_run->next;
    }

    return (p_reversed_list);
}

status_t reverse_list(list_t* p_list)     /* mutable version */
{
    node_t* p_run = NULL;
    node_t* p_run_prev = NULL;
    node_t* p_original_last = NULL;
    node_t* p_current_last = NULL;

    p_original_last = p_list;
    while(p_original_last->next != NULL)
        p_original_last = p_original_last->next;

    p_run = p_original_last->prev;
    p_current_last = p_original_last;
    while(p_run && p_run != p_list)
    {
        p_run_prev = p_run->prev;
        p_current_last->next = p_run;
        p_run->prev = p_current_last;
        p_current_last = p_run;
        p_current_last->next = NULL;
        p_run = p_run_prev;
    }

    if(p_list != p_original_last)
    {
        p_list->next = p_original_last;
        p_original_last->prev = p_list;
    }

    return (SUCCESS);
}

/* list axuillary functions */

static void generic_insert(node_t* p_beg, node_t* p_mid, node_t* p_end)
{
    p_mid->next = p_end;
    p_mid->prev = p_beg;
    if(p_beg != NULL)
        p_beg->next = p_mid;
    if(p_end != NULL)
        p_end->prev = p_mid;
}

static void generic_delete(node_t* p_delete_node)
{
    if(p_delete_node == NULL)
        return;

    if(p_delete_node->next != NULL)
        p_delete_node->next->prev = p_delete_node->prev;
    if(p_delete_node->prev != NULL)
        p_delete_node->prev->next = p_delete_node->next;

    free(p_delete_node);
    p_delete_node = NULL;
}
```

```c
static node_t* search_node(list_t* p_list, data_t s_data)
{
    node_t* p_run = NULL;

    p_run = p_list->next;
    while(p_run != NULL)
    {
        if(p_run->data == s_data)
            return (p_run);
        p_run = p_run->next;
    }

    return (NULL);
}

static node_t* get_node(data_t new_data)
{
    node_t* p_node = NULL;

    p_node = (node_t*)xcalloc(1, sizeof(node_t));
    p_node->data = new_data;
    p_node->prev = NULL;
    p_node->next = NULL;

    return (p_node);
}

/* auxillary function */
static void* xcalloc(size_t nr_elements, size_t size_per_element)
{
    void* p = NULL;

    p = calloc(nr_elements, size_per_element);
    if(p == NULL)
    {
        fprintf(stderr, "calloc:fatal:out of memory\n");
        exit(EXIT_FAILURE);
    }

    return (p);
}

status_t remove_all(list_t* p_list, data_t remove_data)
{
    node_t* p_run = NULL;
    node_t* p_run_next = NULL;

    p_run = p_list->next;
    while(p_run != NULL)
    {
        p_run_next = p_run->next;
        if(p_run->data == remove_data)
            generic_delete(p_run);
        p_run = p_run_next;
    }

    return (SUCCESS);
}
```