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

int main(void)
{
    list_t* p_list = NULL;
    list_t* p_reversed_list = NULL;
    list_t* p_list_1 = NULL;
    list_t* p_list_2 = NULL;
    list_t* p_concat_list = NULL;
    list_t* p_merged_list = NULL;
    data_t data;
    status_t status;

    p_list = create_list();
    assert(p_list);
    show_list(p_list, "After creation:");

    assert(is_list_empty(p_list) == TRUE);
    assert(get_list_length(p_list) == 0);
    assert(get_start(p_list, &data) == LIST_EMPTY);
    assert(get_end(p_list, &data) == LIST_EMPTY);
    assert(pop_start(p_list, &data) == LIST_EMPTY);
    assert(pop_end(p_list, &data) == LIST_EMPTY);
    assert(remove_start(p_list) == LIST_EMPTY);
    assert(remove_end(p_list) == LIST_EMPTY);

    for(data = 0; data < 5; ++data)
        assert(insert_start(p_list, data) == SUCCESS);
    show_list(p_list, "After insert_start:");

    for(data = 5; data < 10; ++data)
        assert(insert_end(p_list, data) == SUCCESS);
    show_list(p_list, "After insert_end:");

    assert(
        insert_after(p_list, -10, 100) ==
        LIST_DATA_NOT_FOUND
    );

    assert(
        insert_before(p_list, 300, 200) ==
        LIST_DATA_NOT_FOUND
    );

    assert(insert_after(p_list, 0, 100) == SUCCESS);
    show_list(p_list, "Afteter insert_after:");
    assert(insert_before(p_list, 0, 200) == SUCCESS);
    show_list(p_list, "After insert_before:");

    assert(get_list_length(p_list) > 0);
    data = -1;
    assert(get_start(p_list, &data) == SUCCESS);
    printf("Start Data:%d\n", data);
    show_list(p_list, "After get_start:");

    data = -1;
    assert(get_end(p_list, &data) == SUCCESS);
    printf("End Data:%d\n", data);
    show_list(p_list, "After get_end:");

    data = -1;
    assert(pop_start(p_list, &data) == SUCCESS);
    printf("Poped Start Data:%d\n", data);
    show_list(p_list, "After pop_start:");

    data = -1;
    assert(pop_end(p_list, &data) == SUCCESS);
    printf("Poped End Data:%d\n", data);
    show_list(p_list, "After pop_end:");

    assert(remove_start(p_list) == SUCCESS);
    show_list(p_list, "After remove_start:");

    assert(remove_end(p_list) == SUCCESS);
    show_list(p_list, "After remove_end:");

```

```
    assert(remove_data(p_list, -10) == LIST_DATA_NOT_FOUND);
    assert(remove_data(p_list, 0) == SUCCESS);
    show_list(p_list, "After remove_data");

    printf("Length = %d\n", get_list_length(p_list));
    assert(is_list_empty(p_list) == FALSE);

    if(find_in_list(p_list, -10) == FALSE)
        puts("-10 is not present in list");

    if(find_in_list(p_list, 2) == TRUE)
        puts("2 is present in list");

    show_list(
        p_list,
        "p_list before immutable reverseal:get_reversed_list():"
    );

    p_reversed_list = get_reversed_list(p_list);
    show_list(
        p_reversed_list,
        "show_listing reversed version of p_list"
    );
    show_list(
        p_list,
        "p_list after get_reversed_list():it should be same before and after:"
    );
    assert(
        destroy_list(&p_reversed_list) == SUCCESS &&
        p_reversed_list == NULL
    );

    show_list(p_list, "Before mutable reverse:reverse_list():");
    assert(reverse_list(p_list) == SUCCESS);
    show_list(p_list, "After mutable reversal:reverse_list():");

    while(is_list_empty(p_list) != TRUE)
        assert(remove_end(p_list) == SUCCESS);

    assert(is_list_empty(p_list) == TRUE);

    show_list(p_list, "Should be empty:");
    assert(reverse_list(p_list) == SUCCESS);
    show_list(p_list, "Reversed of empty list should be empty as well:");

    assert(insert_end(p_list, 100) == SUCCESS);
    show_list(p_list, "Should contain one element:");
    assert(reverse_list(p_list) == SUCCESS);
    show_list(
        p_list,
        "Reversed version of list with one element is same list:"
    );

    assert(
        destroy_list(&p_list) == SUCCESS &&
        p_list == NULL
    );

    puts("Testing inter-list routines");

    p_list_1 = create_list();
    p_list_2 = create_list();
    assert(
        is_list_empty(p_list_1) == TRUE &&
        is_list_empty(p_list_2) == TRUE
    );
    p_merged_list = merge_lists(p_list_1, p_list_2);
    assert(is_list_empty(p_merged_list) == TRUE);
    assert(
        destroy_list(&p_merged_list) == SUCCESS &&
        p_merged_list == NULL
    );

    for(data = 5; data <= 95; data += 10)
        assert(insert_end(p_list_1, data) == SUCCESS);

    for(data = 10; data <= 60; data += 10)
        assert(insert_end(p_list_2, data) == SUCCESS);
```

```
    show_list(p_list, "Should be empty:");
    assert(reverse_list(p_list) == SUCCESS);
    show_list(p_list, "Reversed of empty list should be empty as well:");

    assert(insert_end(p_list, 100) == SUCCESS);
    show_list(p_list, "Should contain one element:");
    assert(reverse_list(p_list) == SUCCESS);
    show_list(
        p_list,
        "Reversed version of list with one element is same list:"
    );

    assert(
        destroy_list(&p_list) == SUCCESS &&
        p_list == NULL
    );

    puts("Testing inter-list routines");

    p_list_1 = create_list();
    p_list_2 = create_list();
    assert(
        is_list_empty(p_list_1) == TRUE &&
        is_list_empty(p_list_2) == TRUE
    );
    p_merged_list = merge_lists(p_list_1, p_list_2);
    assert(is_list_empty(p_merged_list) == TRUE);
    assert(
        destroy_list(&p_merged_list) == SUCCESS &&
        p_merged_list == NULL
    );

    for(data = 5; data <= 95; data += 10)
        assert(insert_end(p_list_1, data) == SUCCESS);

    for(data = 10; data <= 60; data += 10)
        assert(insert_end(p_list_2, data) == SUCCESS);

    show_list(p_list_1, "p_list_1:Before get_concated_lists():");
    show_list(p_list_2, "p_list_2:Before get_concated_lists():");
    p_concat_list = get_concated_list(p_list_1, p_list_2);
    show_list(p_concat_list, "p_concat_list:after concat");
    show_list(p_list_1, "p_list_1:After get_concated_lists():");
    show_list(p_list_2, "p_list_2:After get_concated_lists():");

    p_merged_list = merge_lists(p_list_1, p_list_2);
    show_list(p_merged_list, "p_merged_list:after merge_lists():");
    show_list(p_list_1, "p_list_1:After merge_lists()");
    show_list(p_list_2, "p_list_2:After merge_lists():");

    puts("p_list_1 and p_list_2 : after merge_lists() = before concat_lists():");
    assert(
            concat_lists(p_list_1, &p_list_2)
            == SUCCESS
        );
    show_list(p_list_1, "p_list_1:After concat_lists():");

    assert(clear_list(p_concat_list) == SUCCESS);
    show_list(p_concat_list, "p_concat_list:after clear_list():");
    assert(is_list_empty(p_concat_list) == TRUE);

    assert(
        destroy_list(&p_concat_list) == SUCCESS &&
        p_concat_list == NULL
    );
    assert(
        destroy_list(&p_merged_list) == SUCCESS &&
        p_merged_list == NULL
    );
    assert(
        destroy_list(&p_list_1) == SUCCESS &&
        p_list_1 == NULL
    );

    puts("Implementation successful");

    return (EXIT_SUCCESS);
}
```