

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <assert.h>
4  #include "list.h"
5
6  int main(void)
7  {
8      list_t* p_list = NULL;
9      list_t* p_reversed_list = NULL;
10     list_t* p_list_1 = NULL;
11     list_t* p_list_2 = NULL;
12     list_t* p_concat_list = NULL;
13     list_t* p_merged_list = NULL;
14     data_t data;
15     status_t ...
16
17     p_list = create_list();
18     assert(p_list);
19     show_list(p_list, "After creation:");
20
21     assert(is_list_empty(p_list) == TRUE);
22     assert(get_list_length(p_list) == 0);
23     assert(get_start(p_list, &data) == LIST_EMPTY);
24     assert(get_end(p_list, &data) == LIST_EMPTY);
25     assert(pop_start(p_list, &data) == LIST_EMPTY);
26     assert(pop_end(p_list, &data) == LIST_EMPTY);
27     assert(remove_start(p_list) == LIST_EMPTY);
28     assert(remove_end(p_list) == LIST_EMPTY);
29
30     for(data = 0; data < 5; ++data)
31         assert(insert_start(p_list, data) == SUCCESS);
32     show_list(p_list, "After insert_start:");
33
34     for(data = 5; data < 10; ++data)
35         assert(insert_end(p_list, data) == SUCCESS);
36     show_list(p_list, "After insert_end:");
37
38     assert(
39         insert_after(p_list, -10, 100) ==
40         LIST_DATA_NOT_FOUND
41     );
42
43     assert(
44         insert_before(p_list, 300, 200) ==
45         LIST_DATA_NOT_FOUND
46     );
47
48     assert(insert_after(p_list, 0, 100) == SUCCESS);
49     show_list(p_list, "Afteter insert_after:");
50     assert(insert_before(p_list, 0, 200) == SUCCESS);
51     show_list(p_list, "After insert_before:");
52
53     assert(get_list_length(p_list) > 0);
54     data = -1;
55     assert(get_start(p_list, &data) == SUCCESS);
56     printf("Start Data:%d\n", data);
57     show_list(p_list, "After get_start:");
58
59     data = -1;
60     assert(get_end(p_list, &data) == SUCCESS);
61     printf("End Data:%d\n", data);
62     show_list(p_list, "After get_end:");
63
64     data = -1;
65     assert(pop_start(p_list, &data) == SUCCESS);
66     printf("Poped Start Data:%d\n", data);
67     show_list(p_list, "After pop_start:");
68
69     data = -1;
70     assert(pop_end(p_list, &data) == SUCCESS);
71     printf("Poped End Data:%d\n", data);
72     show_list(p_list, "After pop_end:");
73
74     assert(remove_start(p_list) == SUCCESS);
75     show_list(p_list, "After remove_start:");
76
77     assert(remove_end(p_list) == SUCCESS);
78     show_list(p_list, "After remove_end:");
79
```

```

80     assert(remove_data(p_list, -10) == LIST_DATA_NOT_FOUND);
81     assert(remove_data(p_list, 0) == SUCCESS);
82     show_list(p_list, "After remove_data");
83
84     printf("Length = %d\n", get_list_length(p_list));
85     assert(is_list_empty(p_list) == FALSE);
86
87     if(find_in_list(p_list, -10) == FALSE)
88         puts("-10 is not present in list");
89
90     if(find_in_list(p_list, 2) == TRUE)
91         puts("2 is present in list");
92
93     show_list(
94         p_list,
95         "p_list before immutable reverseal:get_reversed_list():"
96     );
97
98     p_reversed_list = get_reversed_list(p_list);
99     show_list(
100        p_reversed_list,
101        "show_listing reversed version of p_list"
102    );
103    show_list(
104        p_list,
105        "p_list after get_reversed_list():it should be same before and after:"
106    );
107    assert(
108        destroy_list(&p_reversed_list) == SUCCESS &&
109        p_reversed_list == NULL
110    );
111
112    show_list(p_list, "Before mutable reverse:reverse_list():");
113    assert(reverse_list(p_list) == SUCCESS);
114    show_list(p_list, "After mutable reversal:reverse_list():");
115
116    while(is_list_empty(p_list) != TRUE)
117        assert(remove_end(p_list) == SUCCESS);
118
119    assert(is_list_empty(p_list) == TRUE);
120
121    show_list(p_list, "Should be empty:");
122    assert(reverse_list(p_list) == SUCCESS);
123    show_list(p_list, "Reversed of empty list should be empty as well:");
124
125    assert(insert_end(p_list, 100) == SUCCESS);
126    show_list(p_list, "Should contain one element:");
127    assert(reverse_list(p_list) == SUCCESS);
128    show_list(
129        p_list,
130        "Reversed version of list with one element is same list:"
131    );
132
133    assert(
134        destroy_list(&p_list) == SUCCESS &&
135        p_list == NULL
136    );
137
138    puts("Testing inter-list routines");
139
140    p_list_1 = create_list();
141    p_list_2 = create_list();
142    assert(
143        is_list_empty(p_list_1) == TRUE &&
144        is_list_empty(p_list_2) == TRUE
145    );
146    p_merged_list = merge_lists(p_list_1, p_list_2);
147    assert(is_list_empty(p_merged_list) == TRUE);
148    assert(
149        destroy_list(&p_merged_list) == SUCCESS &&
150        p_merged_list == NULL
151    );
152
153    for(data = 5; data <= 95; data += 10)
154        assert(insert_end(p_list_1, data) == SUCCESS);
155
156    for(data = 10; data <= 60; data += 10)
157        assert(insert_end(p_list_2, data) == SUCCESS);

```

```

121     show_list(p_list, "Should be empty:");
122     assert(reverse_list(p_list) == SUCCESS);
123     show_list(p_list, "Reversed of empty list should be empty as well:");
124
125     assert(insert_end(p_list, 100) == SUCCESS);
126     show_list(p_list, "Should contain one element:");
127     assert(reverse_list(p_list) == SUCCESS);
128     show_list(
129         p_list,
130         "Reversed version of list with one element is same list:"
131     );
132
133     assert(
134         destroy_list(&p_list) == SUCCESS &&
135         p_list == NULL
136     );
137
138     puts("Testing inter-list routines");
139
140     p_list_1 = create_list();
141     p_list_2 = create_list();
142     assert(
143         is_list_empty(p_list_1) == TRUE &&
144         is_list_empty(p_list_2) == TRUE
145     );
146     p_merged_list = merge_lists(p_list_1, p_list_2);
147     assert(is_list_empty(p_merged_list) == TRUE);
148     assert(
149         destroy_list(&p_merged_list) == SUCCESS &&
150         p_merged_list == NULL
151     );
152
153     for(data = 5; data <= 95; data += 10)
154         assert(insert_end(p_list_1, data) == SUCCESS);
155
156     for(data = 10; data <= 60; data += 10)
157         assert(insert_end(p_list_2, data) == SUCCESS);
158
159     show_list(p_list_1, "p_list_1:Before get_concated_lists():");
160     show_list(p_list_2, "p_list_2:Before get_concated_lists():");
161     p_concat_list = get_concated_list(p_list_1, p_list_2);
162     show_list(p_concat_list, "p_concat_list:after concat");
163     show_list(p_list_1, "p_list_1:After get_concated_lists():");
164     show_list(p_list_2, "p_list_2:After get_concated_lists():");
165
166     p_merged_list = merge_lists(p_list_1, p_list_2);
167     show_list(p_merged_list, "p_merged_list:after merge_lists():");
168     show_list(p_list_1, "p_list_1:After merge_lists()");
169     show_list(p_list_2, "p_list_2:After merge_lists()");
170
171     puts("p_list_1 and p_list_2 : after merge_lists() = before concat_lists():");
172     assert(
173         concat_lists(p_list_1, &p_list_2)
174         == SUCCESS
175     );
176     show_list(p_list_1, "p_list_1:After concat_lists():");
177
178     assert(clear_list(p_concat_list) == SUCCESS);
179     show_list(p_concat_list, "p_concat_list:after clear_list():");
180     assert(is_list_empty(p_concat_list) == TRUE);
181
182     assert(
183         destroy_list(&p_concat_list) == SUCCESS &&
184         p_concat_list == NULL
185     );
186     assert(
187         destroy_list(&p_merged_list) == SUCCESS &&
188         p_merged_list == NULL
189     );
190     assert(
191         destroy_list(&p_list_1) == SUCCESS &&
192         p_list_1 == NULL
193     );
194
195     puts("Implementation successful");
196
197     return (EXIT_SUCCESS);
198 }
```