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

#define uerror(fun_name) printf("%s:%d:Unexpected Error:", fun_name, __LINE__ )

int main(void)
{
    list_t *lst = create_list(), *lst1, *lst2, *cat_list=NULL, *merge_list=NULL;
    data_t data, *arr=NULL;
    result_t rs;
    len_t lst_len, arr_len;
    int i;

    if(is_empty(lst))
        puts("1:List is empty");
    else{
        uerror("is_empty");
        exit(EXIT_FAILURE);
    }

    if(delete_beg(lst) == LIST_EMPTY)
        puts("2:Cannot del_beg from empty list");
    else{
        uerror("del_beg");
        exit(EXIT_FAILURE);
    }

    if(delete_end(lst) == LIST_EMPTY)
        puts("3:Cannot del_end from empty list");
    else{
        uerror("del_end");
        exit(EXIT_FAILURE);
    }

    if(examine_beg(lst, &data) == LIST_EMPTY)
        puts("4:Cannot examine_beg from empty list");
    else{
        uerror("examine_beg");
        exit(EXIT_FAILURE);
    }

    if(examine_end(lst, &data) == LIST_EMPTY)
        puts("5:Cannot examine_end from empty list");
    else{
        uerror("examine_end");
        exit(EXIT_FAILURE);
    }

    if(examine_and_delete_beg(lst, &data) == LIST_EMPTY)
        puts("6:Cannot examine_and_delete_beg from empty list");
    else{
```

```
53          uerror("examine_and_delete_beg");
54          exit(EXIT_FAILURE);
55      }
56
57      if(examine_and_delete_end(lst, &data) == LIST_EMPTY)
58          puts("7:Cannot examine_and_delete_end from empty list");
59      else{
60          uerror("examine_and_delete_end");
61          exit(EXIT_FAILURE);
62      }
63
64      for(data = 0; data < 5; data++){
65          if(insert_beg(lst, data) != SUCCESS){
66              uerror("insert_beg");
67              exit(EXIT_FAILURE);
68          }
69      }
70      printf("8:display:insert_beg:");
71      display(lst);
72
73      for(data = 5; data < 10; ++data){
74          if(insert_end(lst, data) != SUCCESS){
75              uerror("insert_end");
76              exit(EXIT_FAILURE);
77          }
78      }
79      printf("9:display:insert_end:");
80      display(lst);
81
82      if(insert_after_data(lst, 1000, 100) != DATA_NOT_FOUND){
83          uerror("insert_afer_data");
84          exit(EXIT_FAILURE);
85      }
86      puts("10:insert_after_data:1000 is not in list");
87
88      if(insert_before_data(lst, -435, 200) != DATA_NOT_FOUND){
89          uerror("inesrt_before_data");
90          exit(EXIT_FAILURE);
91      }
92      puts("11:insert_before_data:-435 is not in list");
93
94      if(insert_after_data(lst, 0, 100) != SUCCESS){
95          uerror("insert_after_data");
96          exit(EXIT_FAILURE);
97      }
98      printf("12:insert_after_data:");
99      display(lst);
100
101     if(insert_before_data(lst, 0, 200) != SUCCESS){
102         uerror("insert_before_data");
103         exit(EXIT_FAILURE);
104     }
```

```
105     printf("13:insert_before_data:");
106     display(lst);
107 
108     if(delete_beg(lst) != SUCCESS){
109         uerror("delete_beg");
110         exit(EXIT_FAILURE);
111     }
112     printf("14:del_beg:");
113     display(lst);
114 
115     if(delete_end(lst) != SUCCESS){
116         uerror("delete_end");
117         exit(EXIT_FAILURE);
118     }
119     printf("15:del_end:");
120     display(lst);
121 
122     if(delete_data(lst, -234) != DATA_NOT_FOUND){
123         uerror("delete_data");
124         exit(EXIT_FAILURE);
125     }
126     printf("16:delete_data:");
127     display(lst);
128 
129     if(delete_data(lst, 0) != SUCCESS){
130         uerror("delete_data");
131         exit(EXIT_FAILURE);
132     }
133     printf("17:delete_data:0:");
134     display(lst);
135 
136     if(examine_beg(lst, &data) != SUCCESS){
137         uerror("examine_beg");
138         exit(EXIT_FAILURE);
139     }
140     printf("18:examine_beg:%d\n", data);
141 
142     if(examine_end(lst, &data) != SUCCESS){
143         uerror("examine_end");
144         exit(EXIT_FAILURE);
145     }
146     printf("19:examine_end:%d\n", data);
147 
148     if(examine_and_delete_beg(lst, &data) != SUCCESS){
149         uerror("examine_and_delete_beg");
150         exit(EXIT_FAILURE);
151     }
152     printf("20:examine_and_delete_beg:");
153     display(lst);
154 
155     if(examine_and_delete_end(lst, &data) != SUCCESS){
156         uerror("examine_and_delete_end");
```

```
157         exit(EXIT_FAILURE);
158     }
159     printf("21:examine_and_delete_end:");
160     display(lst);
161 
162     if(is_empty(lst) == TRUE){
163         uerror("is_empty");
164         exit(EXIT_FAILURE);
165     }
166     puts("22:is_empty:List is not empty");
167 
168     lst_len = len(lst);
169     printf("23:len:length:%d\n", lst_len);
170 
171     if(find(lst, -1) != FALSE){
172         uerror("find");
173         exit(EXIT_FAILURE);
174     }
175     puts("24:find:-1 is not in the list");
176 
177     if(find(lst, 6) != TRUE){
178         uerror("find");
179         exit(EXIT_FAILURE);
180     }
181     puts("25:find:6 is present in the list");
182 
183     if((arr = to_array(lst, &arr_len)) == NULL){
184         uerror("to_array");
185         exit(EXIT_FAILURE);
186     }
187 
188     printf("26:to_array:");
189     for(i=0; i < arr_len; ++i)
190         printf("[%d]", arr[i]);
191     printf("\n");
192 
193     rs = destroy_list(&lst);
194     if(rs == SUCCESS && lst == NULL)
195         puts("27:destroy_list:List is successfully destroyed");
196     else{
197         uerror("destroy_list");
198         exit(EXIT_FAILURE);
199     }
200 
201     lst1 = create_list();
202     lst2 = create_list();
203 
204     for(data = 1; data < 50000; ++data)
205         if(insert_end(lst1, 10 * data) != SUCCESS){
206             uerror("insert_end");
207             exit(EXIT_FAILURE);
208         }
```

```
209
210     for(data = 5; data < 56000; data += 10)
211         if(insert_end(lst2, data) != SUCCESS){
212             uerror("insert_end");
213             exit(EXIT_FAILURE);
214         }
215
216     printf("28:lst1:");
217     display(lst1);
218
219     printf("29:lst2:");
220     display(lst2);
221
222     if((cat_list = concat(lst1, lst2)) == NULL){
223         uerror("concat");
224         exit(EXIT_FAILURE);
225     }
226     printf("30:concat:");
227     display(cat_list);
228
229     if((merge_list = merge(lst1, lst2)) == NULL){
230         uerror("merge");
231         exit(EXIT_FAILURE);
232     }
233     printf("31:merge:");
234     display(merge_list);
235
236     if((rs = destroy_list(&lst1)) == SUCCESS && lst1 == NULL)
237         puts("32:destroy_list:lst1 is destroyed successfully");
238     else{
239         uerror("destroy_list");
240         exit(EXIT_FAILURE);
241     }
242
243     if((rs = destroy_list(&lst2)) == SUCCESS && lst2 == NULL)
244         puts("33:destroy_list:lst2 is destroyed successfully");
245     else{
246         uerror("destroy_list");
247         exit(EXIT_FAILURE);
248     }
249
250     if((rs = destroy_list(&cat_list)) == SUCCESS && cat_list == NULL)
251         puts("34:destroy_list:cat_list is destroyed successfully");
252     else{
253         uerror("destroy_list");
254         exit(EXIT_FAILURE);
255     }
256
257     if((rs = destroy_list(&merge_list)) == SUCCESS && merge_list == NULL)
258         puts("35:destroy_list:merge_list is destroyed successfully");
259     else{
260         uerror("destroy_list");
```

```
261             exit(EXIT_FAILURE);
262         }
263
264         exit(EXIT_FAILURE);
265     }
266
267
```