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

/* List Interface Routines */

list_t *create_list(void)
{
    list_t *lst = get_node(0);
    lst->next = lst->prev = lst;
    return (lst);
}

result_t insert_beg(list_t *lst, data_t new_data)
{
    g_insert(lst, get_node(new_data), lst->next);
    return (SUCCESS);
}

result_t insert_end(list_t *lst, data_t new_data)
{
    g_insert(lst->prev, get_node(new_data), lst);
    return (SUCCESS);
}

result_t insert_after_data(list_t *lst, data_t e_data, data_t new_data)
{
    node_t *e_node = search_node(lst, e_data);
    if(!e_node)
        return (DATA_NOT_FOUND);
    g_insert(e_node, get_node(new_data), e_node->next);
    return (SUCCESS);
}

result_t insert_before_data(list_t *lst, data_t e_data, data_t new_data)
{
    node_t *e_node = search_node(lst, e_data);
    if(!e_node)
        return (DATA_NOT_FOUND);
    g_insert(e_node->prev, get_node(new_data), e_node);
    return (SUCCESS);
}

result_t delete_beg(list_t *lst)
{
    if(is_empty(lst))
        return (LIST_EMPTY);
    g_delete(lst->next);
    return (SUCCESS);
}

result_t delete_end(list_t *lst)
```

```c
{
    if(is_empty(lst))
        return (LIST_EMPTY);
    g_delete(lst->prev);
    return (SUCCESS);
}

result_t delete_data(list_t *lst, data_t e_data)
{
    node_t *e_node = search_node(lst, e_data);
    if(!e_node)
        return (DATA_NOT_FOUND);
    g_delete(e_node);
    return (SUCCESS);
}

result_t examine_beg(list_t *lst, data_t *p_data)
{
    if(is_empty(lst))
        return (LIST_EMPTY);
    *p_data = lst->next->data;
    return (SUCCESS);
}

result_t examine_end(list_t *lst, data_t *p_data)
{
    if(is_empty(lst))
        return (LIST_EMPTY);
    *p_data = lst->prev->data;
    return (SUCCESS);
}

result_t examine_and_delete_beg(list_t *lst, data_t *p_data)
{
    if(is_empty(lst))
        return (LIST_EMPTY);
    *p_data = lst->next->data;
    g_delete(lst->next);
    return (SUCCESS);
}

result_t examine_and_delete_end(list_t *lst, data_t *p_data)
{
    if(is_empty(lst))
        return (LIST_EMPTY);
    *p_data = lst->prev->data;
    g_delete(lst->prev);
    return (SUCCESS);
}

result_t find(list_t *lst, data_t f_data)
{
```

```c
    node_t *f_node = search_node(lst, f_data);
    if(f_node)
        return (TRUE);
    return (FALSE);
}

void display(list_t *lst)
{
    node_t *run;

    printf("[beg]<->");

    for(run = lst->next; run != lst; run = run->next)
        printf("[%d]<->", run->data);

    printf("[end]\n");
}

result_t is_empty(list_t *lst)
{
    return (lst->next == lst && lst->prev == lst);
}

len_t len(list_t *lst)
{
    node_t *run = lst->next;
    len_t len = 0;

    for(; run != lst; run = run->next, ++len)
        ;

    return (len);
}

data_t *to_array(list_t *lst, len_t *p_len)
{
    len_t lst_len = len(lst);
    data_t *arr;
    node_t *run;
    int i;

    if(lst_len <= 0)
        return (NULL);

    arr = (data_t*)xcalloc(lst_len, sizeof(data_t));

    for(run = lst->next, i = 0; run != lst; run = run->next, ++i)
        arr[i] = run->data;

    *p_len = lst_len;
    return (arr);
}
```

```c

list_t *to_list(data_t *p_data, len_t len)
{
    list_t *new_list = create_list();
    int i;

    for(i = 0; i < len; i++)
        insert_end(new_list, p_data[i]);

    return (new_list);
}

list_t *merge(list_t *lst1, list_t *lst2)
{
    list_t *lst3 = create_list();
    node_t *run1 = lst1->next, *run2 = lst2->next;
    flag_t from_lst1 = FALSE, from_lst2 = FALSE;

    while(TRUE){

        if(run1 == lst1){
            from_lst1 = TRUE;
            break;
        }

        if(run2 == lst2){
            from_lst2 = TRUE;
            break;
        }

        if(run1->data <= run2->data){
            insert_end(lst3, run1->data);
            run1 = run1->next;
        }
        else{
            insert_end(lst3, run2->data);
            run2 = run2->next;
        }
    }

    if(from_lst1){
        while(run2 != lst2){
            insert_end(lst3, run2->data);
            run2 = run2->next;
        }
    }
    else if(from_lst2){
        while(run1 != lst1){
            insert_end(lst3, run1->data);
            run1 = run1->next;
        }
    }
```

```
209
210     return (lst3);
211 }
212
213 /*
214 void ncat(list_t **pp_lst, int nr_lists, ...);
215
216 main()
217 {
218     list_t *lst1, lst2, lst3, lst4, lst5;
219     list_t *master_lst;
220     //  populate lst1 to lst5
221
222     ncat(&master_lst, 5, lst1, lst2, lst3, lst4, lst5);
223 }
224 */
225
226 list_t *concat(list_t *lst1, list_t *lst2)
227 {
228     list_t *new_list = create_list();
229     node_t *run;
230
231     for(run = lst1->next; run != lst1; run = run->next)
232         insert_end(new_list, run->data);
233
234     for(run = lst2->next; run != lst2; run = run->next)
235         insert_end(new_list, run->data);
236
237     return (new_list);
238 }
239
240 result_t destroy_list(list_t **pp_list)
241 {
242     list_t *p_list = *pp_list;
243     node_t *run, *run_next;
244
245     for(run = p_list->next; run != p_list; run = run_next){
246         run_next = run->next;
247         free(run);
248     }
249
250     free(p_list);
251     *pp_list = NULL;
252     return (SUCCESS);
253 }
254
255 /* List auxillary routines */
256
257 void g_insert(node_t *beg, node_t *mid, node_t *end)
258 {
259     mid->next = end;
260     mid->prev = beg;
```

```
261     beg->next = mid;
262     end->prev = mid;
263 }
264
265 void g_delete(node_t *e_node)
266 {
267     e_node->next->prev = e_node->prev;
268     e_node->prev->next = e_node->next;
269     free(e_node);
270 }
271
272 node_t *search_node(list_t *lst, data_t s_data)
273 {
274     node_t *run;
275
276     for(run = lst->next; run != lst; run = run->next)
277             if(run->data == s_data)
278                     return (run);
279
280     return (NULL);
281 }
282
283 static node_t *get_node(data_t new_data)
284 {
285     node_t *new_node = (node_t*)xcalloc(1, sizeof(node_t));
286     new_node->data = new_data;
287     return (new_node);
288 }
289
290 /* Auxillary routines */
291 static void *xcalloc(int nr_elements, int size_per_element)
292 {
293     void *p = calloc(nr_elements, size_per_element);
294     if(!p){
295         fprintf(stderr, "xcalloc:fatal:out of memory\n");
296         exit(EXIT_FAILURE);
297     }
298     return (p);
299 }
300
301
302
```