ilya@ilya-VirtualBox:~/sanya:*/sem2courseworks/Inf/8$ cat lin-2-list-barrier.h

```
#ifndef _LIN_2_LIST_BARRIER_H_
#define _LIN_2_LIST_BARRIER_H_

#include <stdlib.h>

typedef int data_type;

typedef struct Node{
    data_type data;
    struct Node *next;
    struct Node *prev;
} Node;

typedef struct List {
    int size;
    Node *barr;
} List;

List* list_create(void);
void list_destroy(List **lst);
int list_remove(List *lst, int i);
int list_remove_k(List *lst, int i);
int list_insert(List *lst, int i, data_type value);
void list_push_front(List *lst, data_type value);
void list_push_back(List *lst, data_type value);
data_type list_get_out(List *lst, int i);
data_type list_pop_front(List *lst);
data_type list_pop_back(List *lst);
data_type list_peak(List *lst, int i);
void list_print(List *lst);
int list_size(List *lst);

#endif
```

ilya@ilya-VirtualBox:~/sanya:*/sem2courseworks/Inf/8$ cat lin-2-list-barrier.c

```
#include <stdio.h>
#include "lin-2-list-barrier.h"

List* list_create(void)
{
  List *lst;
  Node *nod;
  lst = (List*)malloc(sizeof(List));
  nod = (Node*)malloc(sizeof(Node));
  lst->size = 0;
  lst->barr = nod;
  nod->next = lst->barr;
  nod->prev = lst->barr;
  return(lst);
}

void list_push_front(List *lst, data_type value)
{
    Node *Next;
    Next = (Node*)malloc(sizeof(Node));
    Next->data = value;
    lst->size++;
    Next->prev = (lst->barr)->prev;
    (lst->barr)->prev = Next;
    Next->next = lst->barr;
    (Next->prev)->next = Next;
}

void list_push_back(List *lst, data_type value)
{
    Node *Next;
```

```
    Next = (Node*)malloc(sizeof(Node));
    Next->data = value;
    lst->size++;
    Next->next = (lst->barr)->next;
    (lst->barr)->next = Next;
    Next->prev = lst->barr;
    (Next->next)->prev = Next;
}

int list_insert(List *lst, int i, data_type value)
{
        if (i == 0 || abs(i) > lst->size + 1) {
                return 1;
        } else {
      lst->size++;
   }
        Node *Inter;
        Node *Tmp = lst->barr;
        Inter = (Node*)malloc(sizeof(Node));
        if (i > 0) {
      for (int k = 0; k < i; k++) {
                    Tmp = Tmp->next;
      }
        } else {
      for (int k = 0; k < -i; k++) {
                    Tmp = Tmp->prev;
          }
      Tmp = Tmp->next;
   }
        Inter->data = value;
        Inter->next = Tmp;
        Inter->prev = Tmp->prev;
        (Tmp->prev)->next = Inter;
        Tmp->prev = Inter;
        return 0;
}

data_type list_pop_front(List *lst)
{
   if (lst->size == 0) {
      puts("Error: stack is empty");
      exit(100);
   }
   data_type tmp = list_peak(lst, -1);
   list_remove(lst, -1);
   return tmp;
}

data_type list_pop_back(List *lst)
{
   if (lst->size == 0) {
      puts("Error: stack is empty");
      exit(100);
   }
   data_type tmp = list_peak(lst, 1);
   list_remove(lst, 1);
   return tmp;
}

data_type list_get_out(List *lst, int i)
{
   if (lst->size == 0) {
      puts("Error: stack is empty");
      exit(100);
   }
   data_type tmp = list_peak(lst, i);
```

```
    list_remove(lst, i);
    return tmp;
}

int list_remove(List *lst, int i)
{
        if (i == 0 || abs(i) > lst->size) {
                return 1;
        } else {
      lst->size--;
    }
        Node *Tmp = lst->barr;
    if (i > 0) {
      for (int k = 0; k < i; k++) {
                Tmp = Tmp->next;
      }
        } else {
      for (int k = 0; k < -i; k++) {
                Tmp = Tmp->prev;
          }
    }
        (Tmp->prev)->next = Tmp->next;
        (Tmp->next)->prev = Tmp->prev;
        free(Tmp);
        return 0;
}

int list_remove_k(List *lst, int i)
{
        if (i == 0 || abs(i) > lst->size) {
                return 1;
        }
    if (i > 0) {
      for (int k = i; k <= lst->size; k = k + i) {
         list_remove(lst, k);
         k -= 1;
      }
        } else {
      for (int k = -i; k <= lst->size; k = k - i) {
         list_remove(lst, -k);
         k -= 1;
      }
    }
    return 0;
}

data_type list_peak(List *lst, int i)
{
    if (lst->size == 0) {
      puts("Error: stack is empty");
      exit(100);
    }
        Node *Tmp = lst->barr;
    if (i > 0) {
      for (int k = 0; k < i; k++) {
                Tmp = Tmp->next;
      }
        } else {
      for (int k = 0; k < -i; k++) {
                Tmp = Tmp->prev;
          }
    }
        return Tmp->data;
}

void list_print(List *lst)
```

```
{
    Node *tmp = (lst->barr)->next;
    for (int i = 0; i < lst->size; i++) {
        printf("%d ", tmp->data);
        tmp = tmp->next;
    }
    putchar('\n');
}


void list_destroy(List **lst)
{
    Node *tmp = ((*lst)->barr)->next;
    Node *next = NULL;
    while (tmp != (*lst)->barr) {
        next = tmp->next;
        free(tmp);
        tmp = next;
    }
    free((*lst)->barr);
    free(*lst);
    (*lst) = NULL;
}

int list_size(List *lst)
{
    return lst->size;
}
ilya@ilya-VirtualBox:~/sanya:*/sem2courseworks/Inf/8$ cat main.c
#include<stdio.h>
#include "lin-2-list-barrier.h"

#define LISTS_NUM 10

int list_no(int *st);

int main(void)
{
        char c;
    char fb;
        int st;
        int no;
        int val;
        List *A[LISTS_NUM];
        for (int i = 0; i < LISTS_NUM; i++) {
                A[i] = NULL;
        }
        while (1){
                scanf("%c", &c);
                switch (c) {
                        case 'c':
                                if (list_no(&st)) {printf("?\n");break;}
            if (A[st] == NULL) {
                A[st] = list_create();
            } else {
                printf("?\n");
            }
                                break;

         case 'd':
                                if (list_no(&st)) {printf("?\n");break;}
                                if (A[st] != NULL) {
                                        list_destroy(&A[st]);
                                } else {
                                        printf("List dose not exist\n");
                                }
```

```
                        break;
                case 'a':
                        if (list_no(&st)) {printf("?\n");break;}
                        if (A[st] == NULL) {
                                printf("?\n");
                                break;
                        }
    if (scanf(" %c", &fb) != 1) {printf("?\n");;break;}
    switch (fb) {
        case 'f':
            while (scanf("%d", &val) == 1) {
                list_push_front(A[st], val);
                if (getchar() == '\n') {
                    break;
                }
            }
            break;
        case 'b':
            while (scanf("%d", &val) == 1) {
                list_push_back(A[st], val);
                if (getchar() == '\n') {
                    break;
                }
            }
            break;
        case 'i':
            if (scanf("%d %d",&no , &val) != 2) {printf("?\n");break;}
            if (list_insert(A[st], no, val)) {
                printf("?\n");
            }
                                break;
        default:
            printf("?\n");
            break;
    }
                        break;
case 'o':
                        if (list_no(&st)) {printf("?\n");break;}
                        if (A[st] == NULL) {
                                printf("?\n");
                                break;
                        }
    if (scanf(" %c", &fb) != 1) {printf("?\n");break;}
    if (list_size(A[st]) == 0) {printf("?\n");break;}
    switch (fb) {
        case 'f':
            printf("%d\n", list_pop_front(A[st]));
            break;
        case 'b':
            printf("%d\n", list_pop_back(A[st]));
            break;
        case 'i':
            if (scanf("%d",&no) != 1) {printf("?\n");break;}
            if (abs(no) > list_size(A[st]) || no == 0) {
                printf("?\n");
            } else {
                printf("%d\n", list_get_out(A[st], no));
            }
            break;
        case 'p':
            if (scanf("%d",&no) != 1) {printf("?\n");break;}
            if (abs(no) > list_size(A[st]) || no == 0) {
                printf("?\n");
            } else {
                printf("%d\n", list_peak(A[st], no));
            }
```

```
                break;
            default:
                printf("?\n");
                break;
        }
                        break;
                case 'r':
                        if (list_no(&st)) {printf("?\n");break;}
                        if (A[st] == NULL) {
                                printf("?\n");
                                break;
                        }
                        if (scanf("%d",&no) != 1) {printf("?\n");break;}
                        if (list_remove(A[st], no)) {
                                printf("?\n");
                        }
                        break;
        case 'k':
                        if (list_no(&st)) {printf("?\n");break;}
                        if (A[st] == NULL) {
                                printf("?\n");
                                break;
                        }
                        if (scanf("%d",&no) != 1) {printf("?\n");break;}
                        if (list_remove_k(A[st], no)) {
                                printf("?\n");
                        }
                        break;
                case 'p':
                        if (list_no(&st)) {printf("?\n");break;}
                        if (A[st] == NULL) {
                                printf("?\n");
                                break;
                        }
                        list_print(A[st]);
                        break;
                case 'q':
                        for (int i = 0; i < 10; i++) {
                                if (A[i] != NULL) {
                                        list_destroy(&A[i]);
                                }
                        }
                        return 0;
                        break;
        case 's':
            if (list_no(&st)) {printf("?\n");break;}
            if (A[st] == NULL) {
                                printf("?\n");
                                break;
                        }
            printf("%d\n", list_size(A[st]));
            break;
                case ' ':
                        break;
                case '\n':
                        break;
                default:
                        printf("Unknown command\n");
                        break;
            }

        }
}

int list_no(int *st)
{
```

```
    if (scanf("%d", st) != 1) {return 1;}
    return *st > LISTS_NUM ? 1 : 0;
}

root@Kali:~/Study/Courseworks/sem2courseworks/Inf/8# valgrind ./main.out
==2639== Memcheck, a memory error detector
==2639== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2639== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==2639== Command: ./main.out
==2639==
c 1
a 1 f 5 6 7 8 9
p 1
5 6 7 8 9
s 1
5
a 1 b 4 3 2 1 0
p 1
0 1 2 3 4 5 6 7 8 9
r 1 2
r 1 -2
p 1
0 2 3 4 5 6 7 9
a 1 i 2 -1
a 1 i -2 -8
p 1
0 -1 2 3 4 5 6 7 -8 9
k 1 2
p 1
0 2 4 6 -8
c 4
p 4

p 2
?
p 22
?
c 44
?
d 44
?
c 2
a 1 f 1 2 3
d 2
p 1
0 2 4 6 -8 1 2 3
k 1 9
?
k 1 4
p 1
0 2 4 -8 1 2
s 1
6
o 1 f
2
o 1 f
1
o 1 b
0
p 1
2 4 -8
o 1 b
2
o 1 b
4
o 1 b
-8
o 1 b
?
o 1 b
?
p 1
a 1 f 1 2 3 4 2 3 4
p 1
1 2 3 4 2 3 4
a 1 i 4 -4
p 1
1 2 3 -4 4 2 3 4
q
==2639==
==2639== HEAP SUMMARY:
==2639== in use at exit: 0 bytes in 0 blocks
==2639== total heap usage: 31 allocs, 31 frees, 2,720 bytes
allocated
==2639==
==2639== All heap blocks were freed -- no leaks are
possible
==2639==
```