**单循环链表的实现**

#include <stdio.h>

#include <malloc.h>

#include <assert.h>

#define ElemType int

typedef struct Node {

ElemType data;

struct Node\* next;

}Node,\*PNode;

typedef struct List {

PNode first;

PNode last;

int size;

}List;

void InitSCList(List\* list) {

PNode s = (PNode)malloc(sizeof(Node));

assert(s != NULL);

list->first = list->last = s;

list->last->next = list->first;

list->size = 0;

}

PNode \_buynode(ElemType x){

PNode s = (PNode)malloc(sizeof(Node));

assert(s != NULL);

s->data = x;

s->next = NULL;

return s;

}

void push\_back(List\* list, ElemType x) {

PNode s = \_buynode(x);

list->last->next = s;

list->last = s;

list->last->next = list->first;

list->size++;

}

void push\_front(List\* list, ElemType x) {

PNode s = \_buynode(x);

s->next = list->first->next;

list->first->next = s;

if (list->first == list->last) { //注意要判断是不是第一个数据

list->last = s;

}

list->size++;

}

void show\_list(List\* list) {

Node\* p = list->first->next;

while (p != list->first) {

printf("%d-->", p->data);

p = p->next;

}

printf("Nul.\n");

}

void pop\_back(List\* list) {

if (list->size == 0) {

return;

}

PNode p = list->first;

while (p->next != list->last) {

p = p->next;

}

free(list->last);

list->last = p;

list->last->next = list->first;

list->size--;

}

void pop\_front(List\* list) {

if (list->size == 0) {

return;

}

PNode p = list->first->next;

list->first->next = p->next;

free(p);

if (list->size == 1) {

list->last = list->first; //注意判断是不是链表里面仅有一个数据

}

list->size--;

}

void insert\_val(List\* list, ElemType x) { //前提是链表已经有序

PNode p = list->first;

while (p->next != list->last && p->next->data < x) {

p = p->next;

} //移动指针p的过程

if (p->next== list->last && p->next->data < x) {

push\_back(list,x);

}

else {

PNode s = \_buynode(x);

s->next = p->next;

p->next = s;

list->size++;

}

}

Node\* find(List\* list, ElemType key) {

if (list->size == 0) {

return NULL;

}

PNode p = list->first->next;

while (p != list->first && p->data != key) { //循环链表的循环条件（是否等于头指针）

p = p->next;

}

if (p == list->first) { //说明找了一圈没找着

return NULL;

}

return p; //如果最后并不是p==list->first,那么此时一定是因为值相等才退出了循环

}

int length(List\* list) {

return list->size;

}

void delete\_val(List\* list, ElemType key) {

if (list->size == 0) {

return;

}

PNode p = find(list, key); //先找到位置

if (p == NULL) {

printf("要删除的数据不存在。\n");

return;

}

if (p == list->last) {

pop\_back(list);

}

else {

PNode q = p->next;

p->data = q->data;

p->next = q->next;

free(q);

list->size--;

}

}

void sort(List\* list) {

if (list->size == 0 || list->size == 1) {

return;

}

PNode s = list->first->next;

PNode q = s->next;

//下面进行断开链表的操作

list->last->next = NULL; //第一步 先不要让整个链表循环

list->last = s;

list->last->next = list->first; //构成一个小的单循环链表

//下面一个一个进行按值插入

while (q != NULL) {

s = q;

q = q->next;

PNode p = list->first;

while (p->next != list->last && p->next->data < s->data) {

p = p->next;

} //移动指针p的过程

if (p->next == list->last && p->next->data < s->data) {

s->next = list->last->next;

list->last->next = s;

list->last = s;

}

else {

s->next = p->next;

p->next = s;

}

//由于是先断开再插入 则不用考虑链表size的变化

}

}

void resver(List\* list) {

if (list->size == 0 || list->size == 1) {

return;

}

PNode p = list->first->next;

PNode q = p->next;

//断开链表

list->last->next = NULL;

list->last = p;

list->last->next = list->first;

while (q != NULL) {

p = q;

q = q->next;

p->next = list->first->next;

list->first->next = p;

}

}

void clear(List\* list) {

PNode p = list->first->next;

while (p != list->first) { //链表循环一圈

list->first->next = p->next;

free(p);

p = list->first->next;

}

list->last = list->first;

list->last->next = list->first; //让最后的链表循环

list->size = 0;

}

void destroy(List\* list) {

clear(list);

free(list->first); //释放头结点的空间

list->first = list->last = NULL; //是我们在构建链表结构体的时候就定义的

}

void main() {

List mylist;

InitSCList(&mylist);

int select = 1;

int Item;

PNode p;

while (select) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\*[1]push\_back [2]push\_front \*\n");

printf("\*[3]show\_list [4]pop\_back \*\n");

printf("\*[5]pop\_front [6]insert\_val \*\n");

printf("\*[7]find [8]length \*\n");

printf("\*[9]delete\_val [10]sort \*\n");

printf("\*[11]resver [12]clear \*\n");

printf("\*[13]destroy [0]quit\_system \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("请选择：>");

scanf\_s("%d", &select);

if (select == 0) {

break;

}

switch (select) {

case 1:

printf("请输入要插入的数据（-1结束）:>");

while (scanf\_s("%d", &Item), Item != -1) {

push\_back(&mylist, Item);

}

break;

case 2:

printf("请输入要插入的数据（-1结束）:>");

while (scanf\_s("%d", &Item), Item != -1) {

push\_front(&mylist, Item);

}

break;

case 3:

show\_list(&mylist);

break;

case 4:

pop\_back(&mylist);

break;

case 5:

pop\_front(&mylist);

break;

case 6:

printf("请输入要插入的数据:>");

scanf\_s("%d", &Item);

insert\_val(&mylist, Item);

break;

case 7:

printf("请输入要查找的数据:>");

scanf\_s("%d", &Item);

p = find(&mylist, Item);

if (p == NULL) {

printf("查找的数据在链表中不存在.\n");

}

break;

case 8:

printf("单链表的长度为:> %d \n", length(&mylist));

break;

case 9:

printf("请输入要删除的值:>");

scanf\_s("%d", &Item);

delete\_val(&mylist, Item);

break;

case 10:

sort(&mylist);

break;

case 11:

resver(&mylist);

break;

case 12:

clear(&mylist);

break;

case 13:

destroy(&mylist);

break;

default:

printf("输入命令错误，请重新输入。\n");

break;

}

}

}