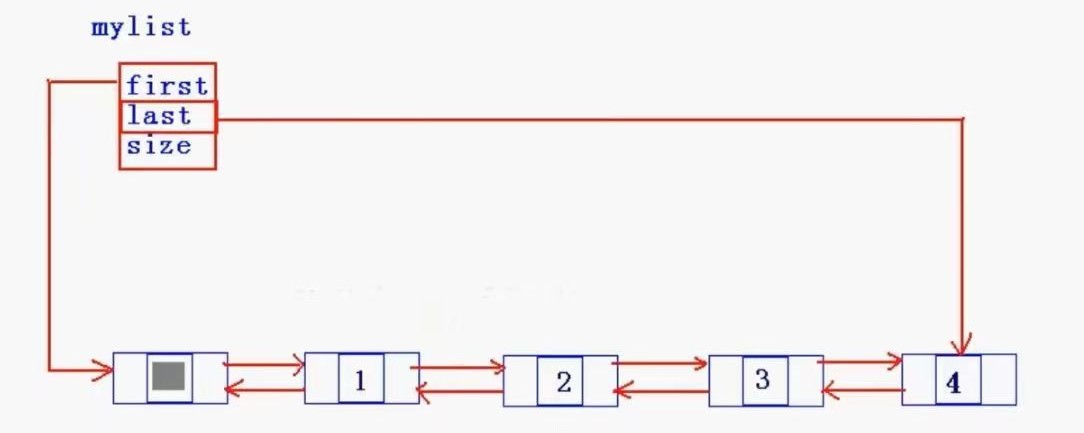
**双向链表的实现**



数据域 后趋域 前趋域

#include <stdio.h>

#include <malloc.h>

#include <assert.h>

typedef int ElemType;

typedef struct Node {

ElemType data;

struct Node\* prio;

struct Node\* next;

}Node,\*PNode;

typedef struct List {

PNode first;

PNode last;

int size;

}List;

void InitDList(List \*list) {

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

assert(s != NULL);

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

list->last->next = NULL; //后趋指针为空

list->first->prio = NULL; //前趋指针为空

list->size = 0;

}

PNode \_buynode(ElemType x) {

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

assert(s != NULL);

s->data = x;

s->next = s->prio=NULL; //和先前单链表不同 这里购买结点的时候要考虑前趋和后趋

return s;

}

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

PNode s = \_buynode(x);

s->prio = list->last;

list->last->next = s;

list->last = s;

list->size++;

}

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

PNode s = \_buynode(x);

if (list->first == list->last) {

list->last = s;

}

//先连上后面的再断开 修改四个指针

else {

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

s->next->prio = s;

}

s->prio = list->first;

list->first->next = s;

list->size++;

}

void show\_list(List\* list) {

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

while (p != NULL) { //非循环链表的while循环终止条件为空指针

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 = NULL;

list->size--;

}

void pop\_front(List\* list) {

if (list->size == 0) {

return;

}

PNode p = list->first->next;

if (list->first->next == list->last) {

list->last = list->first;

list->last->next = NULL;

}

else {

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

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

}

free(p);

list->size--;

}

void insert\_val(List\* list, ElemType x) {

PNode p = list->first; //插入数据要从头结点开始遍历

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

p = p->next;

}

if (p->next == NULL) {

push\_back(list, x);

}

else {

PNode s = \_buynode(x);

s->next = p->next;

s->next->prio = s;

s->prio = p;

p->next = s;

list->size++;

}

}

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

PNode p = list->first->next; //寻找数据则从第一个有数据的结点开始

while (p != NULL && p->data != key) { //两个条件不可以交换顺序

p = p->next;

}

return p; //这一句上面两个条件都囊括在内了

}

int length(List\* list) {

return list->size;

}

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

if (list->size == 0) { return; }

PNode p = find(list, key);

if (p == NULL) {

printf("要删除的值不存在.\n");

return;

}

//与单向链表不同的是不需要用后面的数据覆盖要删除的数据

if (p == list->last) {

list->last = p->prio;

list->last->next = NULL;

/\*free(p);\*/

}

else {

p->prio->next = p->next;

p->next->prio = p->prio;

/\*free(p);\*/

}

free(p);

list->size--;

}

void sort(List\* list) {

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

return;

}

PNode s = list->first->next;

PNode q = s->next;

list->last = s;

list->last->next = NULL;

while (q != NULL) {

s = q;

q = q->next;

PNode p = list->first; //插入数据要从头结点开始遍历

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

p = p->next;

}

if (p->next == NULL) {

//s结点已经存在，所以尾插的方法我们要自己重新编写

s->next = NULL;

s->prio = list->last;

list->last->next = s;

list->last = s;

}

else {

//常规插入 修改四个指针

s->next = p->next;

s->next->prio = s;

s->prio = p;

p->next = s;

}

}

}

void resver(List\* list) {

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

return;

}

PNode p = list->first->next;

PNode q = p->next;

list->last = p;

list->last->next = NULL;

while (q != NULL) {

p = q;

q= q->next;

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

p->next->prio = p;

p->prio = list->first;

list->first->next = p;

}

}

void clear(List\* list) {

//进行头删 直到删除所有结点

if (list->size == 0) {

return;

}

PNode p = list->first->next;

while (p != NULL) {

if (p == list->last) { //判断是否删除的为最后一个结点

list->last = list->first;

list->last->next = NULL;

}

else { //常规头删操作

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

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

}

free(p);

p = list->first->next;

list->size--;

}

}

void destroy(List\* list) {

clear(list);

free(list->first);

list->first = list->last = NULL;

}

void main() {

List mylist;

InitDList(&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;

}

}

}