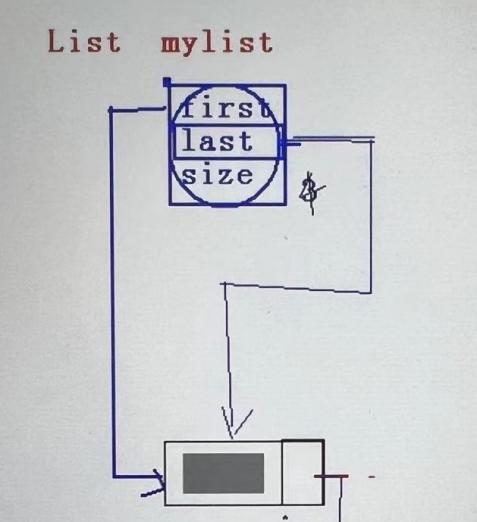
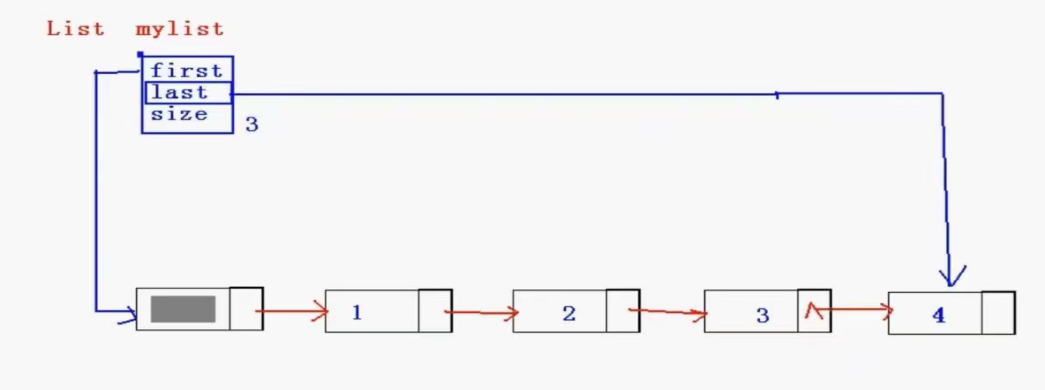
单链表的实现

**一、图示**



（初始化链表的状态）



**二、代码部分：**

#include<stdio.h>

#include <malloc.h>

#include <assert.h>

typedef int ElemType;

typedef struct Node { //结点的类型

ElemType data;

Node\* next;

}Node, \* PNode; //PNode=Node\*，PNode为结点类型的指针，可以这样定义

//typedef Node\* PNode;

typedef struct List { //创建一个单链表（具有管理方式的链表）

PNode first; //first last均为指向结点的指针

PNode last;

size\_t size; //size\_t在64位系统上定义为unsighed int,其类型表示C中任何对象所能达到的最大长度

}List;

void InitList(List\* list) { //下面用了&，上面就要用\*，list是List\*型的变量

list->first = list->last = (Node\*)malloc(sizeof(Node));

//刚开始first和last指向的是同一个结点

assert(list->first != NULL);

list->first->next = NULL;

list->size = 0;

}

//-----------优化部分start-------------

//-----------One------------------

Node\* \_buynode(ElemType x) {

Node\* s = (Node\*)malloc(sizeof(Node));

assert(s != NULL);

s->data = x;

s->next = NULL;

return s;

}

//---------Two----------------------

typedef Node\* It;

//编写迭代器 使代码更加简洁

It begin(List\* list) {

return list->first->next; //指向第一个结点

}

It end(List\* list) {

return list->last->next; //指向最后一个结点的下一个结点

}

//插入的位置给的是某个结点的地址 此时我们要在所给位置结点的前面进行插入工作

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

Node\* p = list->first;

while (p->next != pos) {

p = p->next;

}

Node\* s = \_buynode(x);

s->next = p->next;

p->next = s;

if (pos == NULL) {

list->last = s;

}

list->size++;

}

//则此时尾部插入则可以改成

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

insert(list, end(list), Item);

}

//头部插入可变为

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

insert(list, begin(list), Item); //insert(begin(),Item);

}

//-------------优化部分end-----------------

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

// /\*Node\* s = (Node\*)malloc(sizeof(Node));

// assert(s != NULL);

// s->data = Item;

// s->next = NULL; \*/

//

// Node\* s=\_buynode(Item); //对代码的改进 下面的以此类推

//

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

// list->last = s;

// list->size++;

//}

//

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

// Node\* s = (Node\*)malloc(sizeof(Node));

// assert(s != NULL);

// s->data = Item;

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

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

// //刚开始first和last指向的是同一个结点，所以当插入第一个结点的时候要保证last的指向要改变

// //在之后的插入中就无需再动

// //因此在插入之前我们要进行判断

// if (list->size == 0) {

// list->last = s;

// }

// list->size++;

//}

void show\_list(List\* list) {

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

while (p != NULL) {

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

p = p->next;

}

printf("Nul \n");

}

void pop\_back(List\* list) {

if (list->size == 0) {

return;

}

Node\* 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;

}

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

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

free(p);

//要考虑链表里面只有一个结点的时候，头删之后要更改last指向

if (list->size == 1) {

list->last = list->first;

}

list->size--;

}

//插入数据前要保证链表是有序的 方便Item寻找合适的位置插入

//无序的链表无法按值插入

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

//先申请一个结点

Node\* s = (Node\*)malloc(sizeof(Node));

assert(s != NULL);

s->data = Item;

s->next = NULL;

Node\* p = list->first; //思考运动的指针是指向自身好还是指向它的前驱好

while ( p->next != NULL && p->next->data < Item) { //括号里的条件顺序千万不能交换，交换了如果出现p->next=NULL的情况 是不会有p->next->data值的，会导致程序的崩溃

p = p->next;

}

//判断特殊情况：是否是尾部插入(是的话要修改last指针的指向)

//因为跳出循环有两种情况：一、找到合适的位置 / 二、p走到头了

if (p->next == NULL) {

list->last = s;

}

s->next = p->next;

p->next = s;

list->size++;

}

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

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

//包含了两种情况 一、没找到，p指向空 /二、找到了并返回当前的地址

while (p!= NULL && p->data != key) { //括号里的条件顺序千万不能交换，交换了如果出现p=NULL的情况 是不会有data值的，会导致程序的崩溃

p = p->next;

}

return p;

}

int length(List\* list) {

return list->size;

}

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

if (list->size == 0) {

return;

}

Node\* p = find(list, key);

if (p == NULL) {

printf("要删除的数据不存在.\n");

return;

}

if (p == list->last) {

pop\_back(list);

}

else {

Node\* q = p->next; //新思路

p->data = q->data; //先将q所指向的数据复制到p所指向的数据，将其覆盖掉

p->next = q->next; //然后再删除q所指向的指针

free(q);

list->size--; //这句要写道括号里面 因为尾部删除的函数里已经减过了

}

}

void sort(List\* list) {

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

return;

}

//将整个链表分成两个链表

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

Node\* q = s->next;

list->last = s;

list->last->next = NULL;

//-----------------------

//从q链表中将每一个结点拆下来分别插入第一段链表中

while (q != NULL) {

s = q;

q = q->next;

//拷贝的上面值插入的代码

Node\* p = list->first; //思考运动的指针是指向自身好还是指向它的前驱好

while (p->next != NULL && p->next->data < s->data) { //括号里的条件顺序千万不能交换，交换了如果出现p->next=NULL的情况 是不会有p->next->data值的，会导致程序的崩溃

p = p->next;

}

//判断特殊情况：是否是尾部插入(是的话要修改last指针的指向)

//因为跳出循环有两种情况：一、找到合适的位置 / 二、p走到头了

if (p->next == NULL) {

list->last = s;

}

s->next = p->next;

p->next = s;

}

}

//链表里的排序或逆置一般需要对结点整体进行操作

void resver(List\* list) {

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

return;

}

//将链表分成两个链表

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

Node\* q = p->next;

list->last = p;

list->last->next = NULL;

while (q != NULL) {

p = q;

q = q->next;

//然后再对后面链表里的结点进行头插

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

list->first->next = p;

}

}

void clear(List\* list) {

if (list->size == 0) {

return;

}

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

while (p != NULL) {

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

free(p);

p = list->first->next;

}

list->last = list->first;

list->size = 0;

}

void destroy(List\* list) {

clear(list);

free(list->first);

list->first = list->last = NULL; //将头指针尾指针悬空

}

void main() {

List mylist;

InitList(&mylist);

int select = 1;

int Item;

Node\* 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");

}

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:> \n");

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;

}

}

}