线性表的顺序表示和实现

线性表的顺序表示指的是用一组地址连续的存储单元依次存储线性表的数据元素。

线性结构：

1 2 3 4 5 6 7 8

1没有前驱数据

8没有后驱数据

除1和8之外的数都有前驱和后驱

#include <iostream>

#include <malloc.h>

#include<assert.h>

using namespace std;

#define SEQLIST\_INIT\_SIZE 8

#define INC\_SIZE 3

typedef int ElemType; //typedef作用是为现有的数据类型创建一个新的名字。使代码方便阅读和理解；

typedef struct SeqList {

ElemType\* base; //int类型的指针，指向顺序表空间

int capacity; //容量

int size; //大小

}SeqList;

bool Inc(SeqList\* list) {

ElemType\* newbase = (ElemType\*)realloc(list->base, sizeof(ElemType) \* list->capacity + INC\_SIZE);

if (newbase == NULL) {

printf("增配空间失败，内存不足。\n");

return false;

}

list->base = newbase;

list->capacity += INC\_SIZE;

return true;

}

void InitSetList(SeqList\* list) {

list->base = (ElemType\*)malloc(sizeof(ElemType)\* SEQLIST\_INIT\_SIZE);

assert(list->base != NULL); //断言，保证指针不悬空（即不指向任何位置）

list->capacity = SEQLIST\_INIT\_SIZE;

list->size = 0;

}

void push\_back(SeqList\* list, ElemType x) { //难点：在哪个表插入(base)？ 在表中的哪个位置插入(size)？

if (list->size >= list->capacity && Inc(list)==false) {

printf("顺序表空间已满，不能尾部插入数据。\n");

return;

}

list->base[list->size] = x; //base指向的线性表，下标为size的大小

list->size++;

}

void show\_list(SeqList\* list) {

for (int i = 0; i < list->size; i++) {

printf("%d", list->base[i]);

}

printf("\n");

}

void push\_front(SeqList\* list, ElemType x) { //此时要移动数据，要移动多少个呢？

if (list->size >= list->capacity && Inc(list) == false) {

printf("顺序表空间已满，不能头部插入数据。\n");

return;

}

for (int i = list->size; i > 0; --i) {

list->base[i] = list->base[i - 1];

}

list->base[0] = x;

list->size++;

}

void pop\_back(SeqList\* list) {

if (list->size == 0) {

printf("顺序表已空，不能尾部删除数据。\n");

return;

}

list->size--; //尾部删除只要将size减掉1就行；不一定要从逻辑里删掉

}

void pop\_front(SeqList\* list) {

if (list->size == 0) {

printf("顺序表已空，不能头部删除数据。\n");

return;

}

for (int i = 0; i < list->size - 1;i++) {

list->base[i] = list->base[i + 1];

} //移动数据，注意要先移动第二个数据，防止被覆盖

list->size--;

}

void insert\_pos(SeqList\* list, ElemType pos, ElemType Item) {

//判断位置是否合法，插入的位置能否满足构成线性表的条件

if (pos > list->size || pos < 0 && Inc(list) == false) {

printf("插入数据的位置是非法的，不能插入数据。\n");

return;

}

for (int i = list->size; i > pos; i--) {

list->base[i] = list->base[i - 1];

}

list->base[pos] = Item;

list->size++;

}

int find(SeqList\* list, ElemType key) {

for (int i = 0; i < list->size; i++) {

if (list->base[i] == key) { //此方法对应的是顺序表里没有重复的数据

return i;

}

}

return -1;

}

int length(SeqList\* list) {

return list->size;

}

void delete\_pos(SeqList\* list, ElemType pos) {

if (pos < 0 || pos >= list->size) {

printf("删除数据的位置不合法，不能删除数据。\n");

}

for (int i = pos; i < list->size-1; i++) {

list->base[i] = list->base[i + 1];

}

list->size--;

}

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

int pos = find(list, key);

if (pos == -1) {

printf("要删除的数据不存在，不能删除数据。\n");

return;

}

delete\_pos(list, pos);

list->size--;

}

void sort(SeqList\* list) {

for (int i = 0; i < list->size - 1; i++) {

for (int j = 0; j < list->size - i - 1; j++) {

if (list->base[j] > list->base[j + 1]) {

ElemType tmp = list->base[j];

list->base[j] = list->base[j + 1];

list->base[j + 1] = tmp;

}

}

}

}

void resver(SeqList\* list) {

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

return;

}

ElemType low = 0;

ElemType high = list->size - 1;

ElemType tmp;

while (low < high) {

tmp = list->base[low];

list->base[low] = list->base[high];

list->base[high] = tmp;

low++;

high--;

}

}

void clear(SeqList\* list) {

list->size = 0; //清除数据，表是存在的，可以继续执行添加操作

}

void destroy(SeqList\* list) {

free(list->base); //将指针所指向的空间释放

list->base = NULL; //指针变量指向地址0

list->capacity = 0; //容量设置为0

list->size = 0; //大小设置为0

}

void main() {

SeqList mylist;

InitSetList(&mylist);

ElemType Item;

int pos;

int select = 1;

while (select) {

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

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

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

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

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

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

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

printf("\*[13] clear [14] destroy \*\n");

printf("\*[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(-1结束)：>\n"); //用while循环 省的一个一个的插入

while (scanf\_s("%d", &Item), Item != -1) { //逗号表达式，只看最后面的条件

push\_back(&mylist, Item);

}

break;

case 2:

printf("请输入要插入的数(-1结束)：>\n");

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：>\n");

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

printf("请输入要插入的位置：>\n");

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

insert\_pos(&mylist,pos,Item);

break;

case 7:

printf("请输入要查找的数据：>\n");

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

pos = find(&mylist, Item);

if (pos == -1) {

printf("查找的数据%d在顺序表中不存在。\n");

}

else

printf("查找的数据在顺序表中的%d的下标位置。\n", Item);

break;

case 8:

printf("顺序表的长度为:>&d。\n", length(&mylist));

break;

case 9:

printf("请输入要删除数据的位置:>");

scanf("%d", &pos);

delete\_pos(&mylist, pos);

break;

case 10:

printf("请输入要删除的数据:>");

scanf("%d", &Item);

delete\_val(&mylist, Item);

break;

case 11:

sort(&mylist);

break;

case 12:

resver(&mylist);

break;

case 13:

clear(&mylist);

break;

case 14:

destroy(&mylist);

break;

default:

printf("输入的选择错误，请重新输入。\n");

break;

}

}

}

//两个有序顺序表的合并

void merge(SeqList\* lt, SeqList\* la, SeqList\* lb) {

lt->capacity = la->size + lb->size;

lt->base = (ElemType\*)malloc(sizeof(ElemType) \* lt->capacity); //一定要为新的顺序表开辟空间啊！

assert(lt->base != NULL);

ElemType ia=0;

ElemType ib = 0;

ElemType ic = 0;

while (ia < la->size && ib < lb->size) {

if (la->base[ia] < lb->base[ib]) {

lt->base[ic++] = la->base[ia++]; //后置++，先传值再++；

}

else {

lt->base[ic] = la->base[ib];

ib++;

ic++;

}

}

//考虑线性表大小不同的情况

while (ia < la->size) {

lt->base[ic++] = la->base[ia++];

}

while (ib < lb->size) {

lt->base[ic++] = la->base[ib++];

}

lt->size = la->size + lb->size; //三要素千万不能忘

}

void main() {

SeqList mylist, youlist, list;

InitSetList(&mylist);

InitSetList(&youlist);

push\_back(&mylist,1);

push\_back(&mylist,3);

push\_back(&mylist, 5);

push\_back(&mylist, 7);

push\_back(&mylist, 8);

push\_back(&mylist, 9);

push\_back(&youlist, 2);

push\_back(&youlist, 4);

push\_back(&youlist, 6);

push\_back(&youlist, 8);

push\_back(&youlist, 10);

merge(&list, &mylist, &youlist);

}