//定义常量

#define LIST\_ININ\_SIZE 100

#define LISTINCREMENT 100

typedef int ElemType;

#include <stdio.h>

#include<windows.h>

//结构体声明

typedef struct

{

ElemType \*elem;

int length;

int listsize;

}Sqlist;

bool InitList(Sqlist &L);

bool ListInsert\_Sq(Sqlist &L, int i, ElemType e);

bool visit(int e);

bool UpData(Sqlist &L, int i, int e);

int ListTraverse(Sqlist L, bool visit(int e));

int GetElem(Sqlist L, int i, int e);

int ListLength(Sqlist &L);

int ListDelete(Sqlist &L, int i, int &e);

//主函数

void main()

{

int m=0;

Sqlist L;

//初始化线性表

InitList(L);

//插入线性表

ListInsert\_Sq(L, 1, 5);

ListInsert\_Sq(L, 2, 4);

ListInsert\_Sq(L, 3, 3);

ListInsert\_Sq(L, 4, 0);

ListInsert\_Sq(L, 5, -1);

//取出线性表的特定元素

printf("%d", GetElem(L,3, m));

//遍历线性表

ListTraverse(L,visit);

printf("\n");

//输出线性表的长度

//printf("%d", ListLength(L));

//删除元素

//printf("删除的的元素为%d\n", ListDelete(L, 3, m));

UpData(L,3,10);

ListTraverse(L, visit);

//摧毁线性表

DestroyList(L);

}

/\*

基本操作

\*/

//线性表初始化

bool InitList(Sqlist &L)

{

L.elem = (ElemType \*)malloc(LIST\_ININ\_SIZE \* sizeof(ElemType));

if (!L.elem) exit(0);

L.length = 0;

L.listsize = LIST\_ININ\_SIZE;

return true;

}

//摧毁线性表

bool DestroyList(Sqlist &L)

{

free(L.elem);

L.elem = NULL;

L.length = 0;

L.listsize = 0;

return true;

}

//更新线性表元素

bool UpData(Sqlist &L,int i,int e)

{

if (i<1 || i>L.length)//i值不合法

exit(0);

int \*p;

p = NULL;

p = &(L.elem[i - 1]);

\*p = e;

}

//删除元素

int ListDelete(Sqlist &L,int i,int &e)

{

ElemType \*p, \*q;

if (i<1 || i>L.length)//i值不合法

exit(0);

p = L.elem + i - 1;//p为被删除元素的位置

e = \*p;//被删除元素的值赋给e

q = L.elem + L.length - 1;//表尾元素的位置

for (++p; p <= q; ++p)

\*(p - 1) = \*p;

L.length--;

return e;

}

//清空线性表（将当前元素个数赋值0，遍历不出任何一个元素，相当于清空线性表）

bool ClearList(Sqlist &L)

{

L.length = 0;

return true;

}

//返回线性表的长度

int ListLength(Sqlist &L)

{

return L.length;

}

//判满线性表

bool ListFull(Sqlist L)

{

return (L.length = L.listsize) ? true : false;

}

//判空线性表

bool ListEmpty(Sqlist L)

{

return (L.length==0) ? true : false;

}

//在线性表L中的第i个位置之前插入新的元素e

bool ListInsert\_Sq(Sqlist &L, int i, ElemType e)

{

ElemType \*newbase,\*q,\*p;

q = NULL;

p = NULL;

if (i<1 || i>L.length + 1) exit(0);

if (L.length >= L.listsize)

{

newbase = (ElemType \*)realloc(L.elem, (L.listsize + LISTINCREMENT) \* sizeof(ElemType));

if (!newbase) exit(0);

L.elem = newbase;

L.listsize += LISTINCREMENT;

}

q= &(L.elem[i - 1]);

for (p = &(L.elem[L.length - 1]); p >= q; --p);

\*(p + 1) = \*p;

\*q = e;

++L.length;

return true;

}

//若线性表存在,用e返回L第i个元素的值,求前驱,求后继可由此扩展

int GetElem(Sqlist L, int i, int e)

{

if (i<1 || i>L.length)

exit(0);

else

{

if (!ListEmpty(L))

{

e = L.elem[i - 1];

return e;

}

else

return 0;

}

}

int compare(ElemType e1, ElemType e2)

{

if (e1==e2) return 0;

else if (e1 > e2) return 1;

else return -1;

}

//返回L中的第一个元素与e满足关系compare()的数据元素的位序,若这样的元素不存在,则返回0;

int LocateElem(Sqlist L, int e, int(\*compare)(int, int))

{

int \*p = L.elem;

int i = 1;

while (i <= L.length && !compare(\*p++, e))

{

if (i <= L.length)

return i;

else

return 0;

}

}

bool visit(int e)

{

printf("%d\n", e);

return true;

}

int ListTraverse(Sqlist L, bool visit(int e))

{

for (int i = 0; i < L.length; i++)

visit(L.elem[i]);

return 0;

}