1.顺序表的定义

使用结构体来构造一个顺序表。

typedef struct

{

int length;//当前顺序表长度

int Maxsize;//顺序表最大长度

int\* data;//定义顺序表中元素类型的数组指针

}SqList;

2.顺序表的初始化

顺序表的初始化是使用动态分配数组空间方式构造一个空的线性表。

#include<stdio.h>

#include<stdlib.h>

#define InitSize 10

void InitList(SqList &L)

{

L.data = (int \*)malloc(InitSize\*sizeof(int));//用malloc函数申请一片空间

L.length = 0;//把顺序表的当前长度设为0

L.Maxsize = InitSize;//这是顺序表的最大长度

}

malloc函数：malloc函数的返回值是void \*，使用malloc函数要在返回的时候转化为我们需要的类型。malloc(InitSize \* sizeof（int))这代表的是申请了InitSize个int型大小的空间。malloc函数的使用要引头文件#include<stdlib.h>。

分配成功则返回指向被分配内存的指针，分配失败则返回空指针NULL。

3.增加顺序表的长度

操作步骤：新建一个\* p用来指向原来顺序表的地址，然后新申请一片更大的空间，再把\*p指向的值（原先的顺序表元素）一个个放入到新申请空间的顺序表里，最后销毁原来的顺序表。

void IncreaseSize(SqList &L)

{

int len;

int \*p = L.data;//\*p指向的地址和顺序表的首地址是一样的

printf("请输入你要增加的顺序表的长度:");

scanf("%d", &len);

L.data = (int \*)malloc((L.Maxsize + len)\*sizeof(int));//新申请一片空间

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

L.data[i] = p[i];//把值一个个复制过去

L.Maxsize = L.Maxsize + len;//顺序表最大长度增加len

free(p);//释放空间

}

4.1顺序表的元素查找（按位查找）

顺序表有随机存取的功能，因此按位查找元素可以直接通过数组下标定位取得。

bool GetElem(SqList &L)

{

int i;

printf("你要找第几个元素：");

scanf("%d", &i);

if (i<1 || i>L.length + 1)//判断输入的i值是否合法

{

printf("查找失败\n");

return false;//i值不合法，返回一个false

}

printf("第%d个元素是%d\n", i, L.data[i - 1]);

return true;//返回一个true

}

false/true是bool型变量，C++独有，一般将非零值看做true，将零值看做false。

4.2顺序表的元素查找（按值查找）

顺序表按值查找，只能采用依次遍历的方法。

void LocateElem(SqList &L)

{

int e;

int k = 1;

printf("输入你要查找的元素：");

scanf("%d", &e);

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

if (L.data[i] == e)

{

printf("找到了，是第%d个元素\n", i + 1);

k = 0;

break;

}

if (k)

printf("找不到元素%d\n", e);

}

5.顺序表的元素插入

顺序表的元素插入和插队是一个意思的。想象一下，有一个人要插队，他要插到第3个位置去，那么他前面的两个人不用动，而他后面的人都得动。具体步骤是：最后面的那个人后退一个位置，倒数第二个人后退到原来最后一个人的位置，这样子后面的每个人依次后退，最后就空出来了一个位置，这个人就插队进去了。顺序表也是这么插入的。在插入操作完成后表长+1（多了一个人）。

元素插入有一些要求：

1.元素下标是否越界（有没有插队到奇怪的位置）

2.顺序表存储空间是否满了（有没有位置让你插队）

bool ListInsret(SqList &L)

{

int i, e;

printf("请输入要插入顺序表的元素和元素位置：");

scanf("%d %d", &e, &i);

if (i<1 || i>L.length + 1)//判断元素下标是否越界

return false;

if (L.length > L.Maxsize)//判断顺序表存储空间是否满了

return false;

for (int j = L.length; j >= i; j--)

{

L.data[j] = L.data[j-1];//从后往前逐个后移元素

}

L.data[i-1] = e;//将新元素放入下标为i-1的位置

L.length++;//表长+1

printf("插入的元素是%d，插入的位置是%d\n", e, i);

return true;

}

6.顺序表的元素删除

删除和插入的操作类型，这里借用插队的例子说明。一群人在排队，有一个人有事临时走了，那么这个人的位置就空出来了，后面的人就一个个往前一步，补上这个空位。在删除操作完成后表长-1（少了一个人）。

元素删除有一些要求：

1.元素下标是否越界（走的人是不是这个排队里面的人）

2.顺序表存储空间是否为空（有没有人可以走）

bool ListDelete(SqList &L)

{

int i, e;

printf("请输入要删除的元素位置：");

scanf("%d",&i);

if (i<1 || i>L.length + 1)//判断元素下标是否越界

return false;

if (!L.data)//判断是不是空表

{

printf("空表\n");

return false;

}

e = L.data[i - 1];

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

{

L.data[j-1] = L.data[j];

}

L.length--;//表长-1

printf("删除的元素是%d，这个元素的位置是%d\n", e, i);

return true;

}

7.顺序表的打印

bool PrintList(SqList &L)

{

if (!L.data)//判断是不是空表

return false;

printf("顺序表里的元素有：");

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

printf("%d ", L.data[i]);

printf("\n");

return true;

}

8.求顺序表的表长

int Length(SqList &L)//求表长

{

if (L.length == 0)

return 0;

return L.length;

}

9.顺序表的销毁

顺序表初始化的时候是用malloc函数向系统申请的空间，malloc函数申请的空间是在内存的堆区，堆区的空间不会被系统自动回收，只把L.length改为0是不够的，还需要用free函数释放空间。与malloc一样，要引头文件#include<stdlib.h>。

————————————————