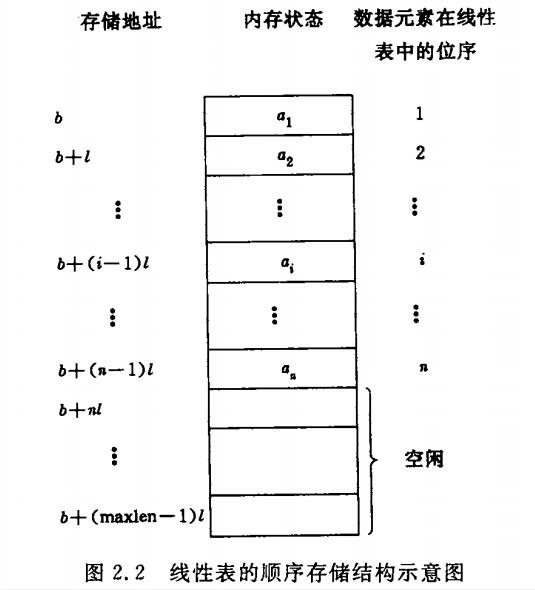
顺序表

[TOC]

# # 1 顺序表的基本概念

　　顺序存储的线性表称为**顺序表**。表中元素的逻辑顺序与物理顺序相同。

　　假设顺序表L存储的起始位置是b，每个数据元素所占用存储空间大小是l，则表L所对应的顺序存储如下图。（***本文规定：****顺序表元素位序从1开始，而数组元素下标从0开始*）



　　顺序表中元素的一维数组可以是**静态分配**，也可以是**动态分配**。

## ##1.1 静态存储

在静态分配时，由于数组的大小和空间已经固定，一旦空间占满，再加入新的数据产生溢出，就会导致程序崩溃。

#include<stdio.h>

#define MaxSize 50 //预定义顺序表最大长度

#define ElemType int //预定义顺序表元素类型

typedef struct{

ElemType data[MaxSize]; //顺序表的元素

int length; //顺序表的当前长度

}SqList; //顺序表的类型定义

int main(){

SqList L;L.length=0; //声明定义顺序表L，并设置当前长度为0

printf("%d\n",L.length);

return 0;

}

## ##1.2 动态存储

在动态分配时，存储数组的空间是在程序执行中通过动态存储分配的，一旦已分配空间占满，可以另外开辟一块更大的存储空间，用以代换原来的存储空间，从而达到扩充存储数组空间的目的。

#include<stdio.h>

#include<stdlib.h> //定义malloc和realloc函数的头文件

#define InitSize 100 //预定义顺序表初始长度

#define ListIncrement 10 //预定义顺序表扩充增量

#define ElemType int //预定义顺序表元素类型

typedef struct{

ElemType \*data; //动态分配数组的指针

int length,MaxSize; //顺序表的最大容量和当前个数

}SeqList; //动态分配的顺序表的类型定义

SeqList L; //声明定义顺序表L，全局变量默认长度为0

int main(){

L.data=(ElemType\*)**malloc**(sizeof(ElemType)\*InitSize); //C语言初始动态分配语句

// L.data=new ElemType(InitSize); //C++的初始动态分配语句

L.MaxSize=InitSize; //设置顺序表最大容量

L.data=(ElemType\*)realloc(**L.data**,sizeof(ElemType)\*(InitSize+ListIncrement));//顺序表扩充

L.MaxSize+=ListIncrement; //顺序表扩充后最大容量也要设置

printf("%d\n",L.length);

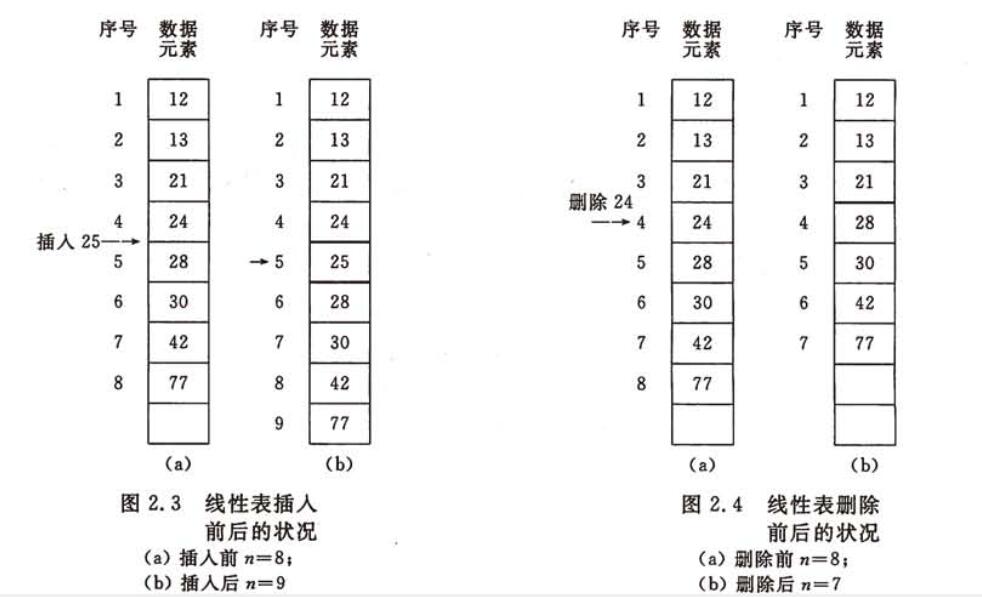
return 0;

}

# # 2 顺序表的基本操作

## ## 2.1 插入操作

在顺序表L的第i（1<=i<=L.length+1）个位置插入新元素e。如果i的输入不合法，则返回false，表示插入失败。如果顺序表当前长度已达到数组的最大容量，则扩充分配存储空间。否则将顺序表的第i个元素以及其后的元素右移1个位置，腾出一个空位置插入新元素。



#include<stdio.h>

#include<stdlib.h> //定义malloc和realloc函数的头文件

#define InitSize 1 //预定义顺序表初始长度

#define ListIncrement 10 //预定义顺序表扩充增量

#define ElemType int //预定义顺序表元素类型

typedef struct{

ElemType \*data; //动态分配数组的指针

int length,MaxSize; //顺序表的最大容量和当前个数

}SeqList; //动态分配的顺序表的类型定义

SeqList L; //声明定义顺序表L，全局变量默认长度为0

//1.插入操作

bool ListInsert\_Seq(SeqList &L,int i,ElemType e){ //第i个位置（1<=i<=L.length+1）插入元素e

if(i<1||i>L.length+1) return false; //i的位置不合法

if(L.length>=L.MaxSize){ //当前长度已达到最大容量，扩充分配存储空间

ElemType \*newbase=(ElemType\*)realloc(L.data,sizeof(ElemType)\*(InitSize+ListIncrement));//顺序表扩充

if(!newbase)return false; //存储分配失败

L.data=newbase; //新基址

L.MaxSize+=ListIncrement; //增加存储容量

}

for(int j=L.length;j>=i;j--) //第i个元素以及其后的元素右移1个位置

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

L.data[i-1]=e; //腾出一个空位置插入新元素

L.length++; //表当前长度+1

return true;

}

int main(){

L.data=(ElemType\*)malloc(sizeof(ElemType)\*InitSize);

L.MaxSize=InitSize;

ListInsert\_Seq(L,1,10);

ListInsert\_Seq(L,1,11);

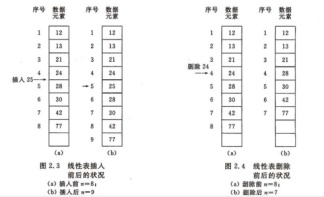
printf("%d\n",L.length);

return 0;

}

## ## 2.2 删除操作

　　删除顺序表L的第i（1<=i<=L.length）个位置的元素。成功返回true，失败返回false，并将被删除的元素用引用变量e返回。



#include<stdio.h>

#include<stdlib.h> //定义malloc和realloc函数的头文件

#define InitSize 1 //预定义顺序表初始长度

#define ListIncrement 10 //预定义顺序表扩充增量

#define ElemType int //预定义顺序表元素类型

typedef struct{

ElemType \*data; //动态分配数组的指针

int length,MaxSize; //顺序表的最大容量和当前个数

}SeqList; //动态分配的顺序表的类型定义

SeqList L; //声明定义顺序表L，全局变量默认长度为0

//2.删除操作

bool ListDelete\_Seq(SeqList &L,int i,ElemType& e){ //删除第i个位置（1<=i<=L.length）元素

if(i<1||i>L.length)return false; //i的位置不合法

e=L.data[i-1]; //将被删除的元素用引用变量e返回

for(int j=i;j<L.length;j++) //第i个位置之后元素前移

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

L.length--; //表长度-1

return true;

}

int main(){

L.data=(ElemType\*)malloc(sizeof(ElemType)\*InitSize);

L.MaxSize=InitSize;

ListInsert\_Seq(L,1,10); //

int e;

if(ListDelete\_Seq(L,1,e))

printf("delete:%d,length:%d\n",e,L.length);

return 0;

}

## ## 2.3 查找操作

在顺序表L中查找第一个元素的值等于e的值，并返回其位号。

//3.按值查找

int LocateElem\_Seq(SeqList L,ElemType e){

for(int i=1;i<=L.length;i++)

if(L.data[i-1]==e)return i; //返回第i个元素(下标为i-1值为e)的位号i

return 0; //返回0表明查找失败

}

## ## 2.4 顺序表并集

　　假设两个顺序表La和Lb分别表示两个集合，现将所有不在顺序表Lb但不在La的元素插入到La中，即：La = La∪Lb

//## 2.4 顺序表并集

void Union\_Seq(SeqList &La,SeqList Lb){

int La\_len=La.length,Lb\_len=Lb.length; //求顺序表长度

for(int i=1;i<=Lb\_len;i++){

int e=Lb.data[i-1]; //取Lb第i个元素赋给e

if(!LocateElem\_Seq(La,e)){ //在La查找不到e,则插入

ListInsert\_Seq(La,++La\_len,e);

}

}

}

## ## 2.5 顺序表合并

已知顺序表La和Lb的元素按值非递减排列（即La和Lb都是**有序表**），现归并La和Lb得到有序表Lc。

//## 2.5 顺序表合并

void MergeList\_Seq(SeqList La,SeqList Lb,SeqList &Lc){

Lc.MaxSize=La.length+Lb.length;

Lc.data=(ElemType\*)malloc(sizeof(ElemType)\*Lc.MaxSize);

if(!Lc.data)exit(0); //存储分配失败

int ai=0,bi=0,ci=0;

while(ai<La.length&&bi<Lb.length){

if(La.data[ai]<Lb.data[bi]) //按非递减排列归并

ListInsert\_Seq(Lc,++ci,La.data[ai++]);

else

ListInsert\_Seq(Lc,++ci,Lb.data[bi++]);

}

while(ai<La.length)ListInsert\_Seq(Lc,++ci,La.data[ai++]); //插入La剩余元素

while(bi<Lb.length)ListInsert\_Seq(Lc,++ci,Lb.data[bi++]); //插入Lb剩余元素

}