**第二章作业**

**第一题（2.1）若将顺序表中记录其长度的分量 listlen 改为指向最后一个元素的位置 last，在实现 各基本运算时需要做那些修改？**

算法分析

因为listlen= last+1;所以程序要做以下修改

算法实现

#define MAXIEN 100

typedef struct sList

{

elementType data[MAXLEN];

int listlen;//(此处修改为 int last;)

}seqlist;

//创建顺序表

void initiallist(seqlist &L)

{

L->listlen=0;//(此处修改为 L->last=-1;)

}

//顺序表长度

int listlength (seqlist &L)

{

return L.listlen;//(此处修改为return L.last+1;)

}

//按位置插入元素

int getelement (seqlist &L,int i; elementType x)

{

if(i<1||i>L.listlen)//(此处修改为if(i<1||i>L.last+1))

{

return 0;

}

else

{

x=L.data[i-1];

return 1;

}

}

//按值查询

int getlocate(seqList &L, elementType x)

{

int i;

for(i=0; i< L.listlen;i++)//（此处改为i<=L.last）

{

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

return i+1;

}

return 0;

}

//按序号取值

int getnum( seqList &L, int i)

{

int x;

if(i<1 || i> L.listlen) // (此处修改为if(i<1||i>L.last+1))

cout<<”超出取值范围”<<endl;

return 0;

else

{

x=L.data[i-1];

return x;

}

}

//打印顺序表

void printList(seqList &L)

{

int i;

for(i=0;i<L.listlen;i++)//(此处改为i<L.last+1)

cout<<L.data[i]<<" ";

cout<<endl;

}

**第二题（2.7）假设顺序表L中的元素按从小到大的次序排列，设计算法以删除表中重复的元素, 并 要求时间尽可能少。要求：**

1. **对顺序表（1,1,2,2,2,3,4,5,5,5,6,6,7,7,8,8,8,9）模拟执行本算法，并统计移动元素的 次数。**

算法描述

时间性能最好要避免使用for循环嵌套进行数据遍历。又因为该表是按顺序排列，(n+1=m时)前后数据只有可能data[n]=data[m]或者data[n]<data[m]两种情况。data[n]=data[m]时，保留前者,执行m++；data[m]继续与data[n]比较，若不相等则data[n+1]=data[m]，若相等则不进行操作;之后n++,m++；重复比较步骤，一直比较到m=listlen-1。

算法实现

void removesame(seqList & L)

{

if(L.listLen<=1)

cout<<"该表无重复元素，无需进行操作"<<endl;

else{

int n=0,m=1;

while(m<L.listLen)

{

if(L.data[n]==L.data[m])

m++;

else {

if((n+1)==m)

{

n++;

m++;

}

else

{ L.data[n+1]=L.data[m];

n++;

m++;

}

}

}

L.listLen=n+1;

}

}

**（2）分析算法的时间性能。**

该算法只对数据进行了一次遍历，所以时间复杂度为O(n)

**第二题（2.21）设计算法将链表 L 就地逆置，即利用原表各结点的空间实现逆置。**

算法分析描述：

不申请新节点，需要将原数据链表拆开每次取下一个节点数据，因为要逆序，所以采用头插法，在原链表头结点后联结新元素。首先创建临时节点p指向第二个元素，来将原链表的头节点head和第一个元素a1，与后面的a2…an分开，将a1->next置空作为链表的尾。创建临时节点t指向p之后的元素，这样就可以将p取下，头插法插到head和a1直接，p->next = A.head->next; A.head->next = p; p= t 重新指向下一个元素，循环上述步骤直到an插入链表。

算法实现

void linkedList::nilist(linkedList& A)

{

List \*p, \*t;

p = A.head->next->next; //指向第二个元素

A.head->next->next=*NULL*;//第一个元素next置空 head-> a1 .....

while (p != *NULL*) {

t = p->*next*;//t指向p后面的元素

p->*next* = A.head->next; // p头插法插到头结点后head->p->a1

A.head->next = p; //

p = t; //p指向下一个元素

}

//循环直到head->an->an-1......->a1

}

**第二题（2.23）设计算法以判断带头结点的双循环链表 L 是否是对称的，即从前往后和从后往前的 输出序列是相同的。若对称，返回 1，否则返回 0。**

算法描述

创建新指针节点f和b指向前驱和后驱，分别从前后开始比较数据，如果f->data等于b->data, b=b->next; f=f->prior;准备比较下一对数据，如果不相等直接return 0；直到前驱f等于后驱b的地址，结束比较。如果能比较到最后说明完全对称，return 1;另外当链表为空时直接判定为不匹配

算法实现

int judgeDuichen(Linklist \*L)

{

if(L->next==L)

{

return 0 ;

}

else

{

Linklist \*b=L->next;

Linklist \*f=L->prior;

while(b!=f)

{

if(f->data==b->data)

{

b=b->next;

f=f->prior;

}

else

{

return 0;

}

}

return 1;

}

}