实验内容

1．建立单链表，并在单链表上实现插入、删除和查找操作（验证性内容）。

2．建立双向链表，并在双向链表上实现插入、删除和查找操作（设计性内容）。

3．计算已知一个单链表中数据域值为一个指定值x的结点个数（应用性设计内容）。

**任务一：**

1．实验要求

编程实现如下功能

（1）根据输入的一系列整数，以0标志结束，用***头插法***建立单链表，并输出单链表中各元素值，观察输入的内容与输出的内容是否一致。

（2）根据输入的一系列整数，以0标志结束，用***尾插法***建立单链表，并输出单链表中各元素值，观察输入的内容与输出的内容是否一致。

（3）在单链表的第i个元素之后插入一个值为x的元素，并输出插入后的单链表中各元素值。

（4）删除单链表中第i个元素，并输出删除后的单链表中各元素值。

（5）在单链表中查找第i个元素，如果查找成功，则显示该元素的值，否则显示该元素不存在。

2. 实验相关原理：

线性表的链式存储结构是用一组任意的存储单元依次存放线性表中的元素，这组存储单元可以是连续的，也可以是不连续的。为反映出各元素在线性表中的前后逻辑关系，对每个数据元素来说，除了存储其本身数据值之外，还需增加一个或多个指针域，每个指针域的值称为指针，又称作链，它用来指示它在线性表中的前驱或后继的存储地址。数据值本身和指针两个部分一起组成一个数据元素完整的存储结构，称为结点，若干个结点链接成链表。如果一个结点中只含一个指针的链表，则称单链表。单链表的存储结构一般化描述如下：

struct Lnode {

Elemtype data;/\*数据域\*/

struct Lnode \*next;/\*指针域\*/

} LNODE, \*Linklist; /\*其中Lnode为结点类型名，LNODE为结点变量名，Linklist为指向结点的指针名\*/

【核心算法提示】

1．链表建立操作的基本步骤：链表是一个动态的结构，它不需要事先分配空间，因此建立链表的过程是一个结点“逐个插入” 的过程。先建立一个只含头结点的空单链表”（也可以没有头结点，只有值为空的头指针，包含头结点是常用的实现方式），然后依次生成新结点，再不断地将其插入到链表的头部或尾部，分别称其为“头插法”和“尾插法”。

2．链表查找操作的基本步骤：因链表是一种"顺序存取"的结构，则要在带头结点的链表中查找到第 i个 元素，必须从头结点开始沿着后继指针依次"点数"，直到点到第 i 个结点为止,如果查找成功，则用e返回第i个元素值。头结点可看成是第0个结点。

3．链表指定位置插入操作的基本步骤：先确定要插入的位置，如果插入位置合法，则再生成新的结点，最后通过修改链将新结点插入到指定的位置上。

4．链表指定位置删除操作的基本步骤：先确定要删除的结点位置，如果位置合法，则再通过修改链使被删结点从链表中“卸下”，最后释放被删结点的空间。

【核心算法参考】

void creat1(Lnode \* &L) /\*输入一系列整数，以0标志结束，将这些整数作为data域并用头插法建立一个带头结点的单链表的函数\*/

{ L=new Lnode;/\*生成头结点\*/

L->next=NULL;/\*头结点的指针域初始为空\*/

int node;

cin>>node;

while(node!=0)

{ p= new Lnode;/\*为一个新结点的分配空间\*/

p->data=node; /\*为新结点数据域赋值\*/

p->next=L->next;/\*新结点指针域指向开始结点\*/

L->next=p; /\*头结点指针域指向新结点，即新结点成为开始结点\*/

cin>>node;

}

}

void creat2(Lnode \* &L) /\*输入一系列整数，以0标志结束，将这些整数作为data域并用尾插法建立一个带头结点的单链表的函数\*/

{ L=new Lnode;/\*为头结点分配空间\*/

L->next=NULL; /\*头结点的指针域初始为空\*/

r=L; /\*尾指针初始指向头结点\*/

cin>>node;

while(node!=0)

{ p= new Lnode;/\*为一个新结点分配空间\*/

p->data=node; /\*新结点数据域赋值\*/

p->next=NULL; /\*新结点指针域为空\*/

r->next=p;/\*尾结点指针域指向新结点\*/

r=p; /\*尾指针指向新结点，即新结点成为尾结点\*/

cin>>node;

}

}

status list\_search(Lnode \* L, int i; Elemtype &e)

/\*在带头结点的单链表L中查找第i个元素，如果查找成功则用e返回其值\*/

{ p=L->next; /\*使指针p指向第1个元素结点\*/

j=1; /\*计数器赋初值为1\*/

while (p&& j<i) /\*顺着后继指针查找第i个元素结点\*/

{ p=p->next;

j++;

}

if (!p&& j>i)

return ERROR; /\*如果i值不合法，即i值小于1或大于表长，则出错\*/

e=p->data; /\*如果第i个元素存在，则将该元素值赋给e\*/

return OK;

}

status list\_insert(Lnode \* &L, int i; Elemtype x)

/\*在带头结点的单链表L中第i个位置之前插入新元素x\*/

{ p=L; j=0;

while(p!=NULL&&j<i-1) /\*寻找插入位置，并使p指向插入位置的前驱结点，即L中的第i-1个位置\*/

{ p=p->next;

++j;

}

if(p==NULL||j>i-1) return ERROR; /\*若位置不正确,即i小于1或大于表的长度加1，则出错\*/

s=new Lnode; /\*分配一个新结点的空间\*/

s->data=x; /\*为新结点数据域赋值\*/

/\*下面两条语句就是完成修改链，将新结点s插入到p结点之后\*/

s->next=p->next; /\*新结点指针域指向p的后继结点\*/

p->next=s; /\*新结点成为p的后继结点\*/

return OK;

}

status list\_delete(Lnode \* &L,int i,Elemtype &x)

/\*在带头结点的单链表L中，删除第i个元素结点,并用x返回其值\*/

{ p=L;j=0;

while(p->next!=NULL&&j<i-1) /\*寻找被删结点，并使p指向被删结点的前驱\*/

{ p=p->next;

++j;

}

if (p->next==NULL||j>i-1)

return ERROR; /\*若位置不正确，即i小于1或大于表长，则出错\*/

q=p->next; /\* q指向p的后继结点\*/

p->next=q->next; /\*q的后继结点成为p的后继结点\*/

x=q->data; /\*用x返回第i个位置的元素\*/

free(q); /\*释放q所指的被删结点的空间\*/

return OK;

}

**3．主函数测试验证参考**

main()/\*主函数\*/

{ Lnode \* L=NULL; int i,x;

L=creat(L);/\*调用单链表建立函数\*/

cout<<"the Linklist is:\n";

display(L); /\*调用单链表元素输出(遍历)函数\*/

cout<<"please input the position you want to insert:";/\*请求输入插入操作位置\*/

cin>>i;

cout<<"please input the node you want to insert:");/\*请求输入需要插入的元素\*/

cin>>x;

L=insert(L,i,x);/\*调用单链表插入函数\*/

cout<<"the Linklist after inserted is:\n";

display(L);/\*调用单链表元素输出(遍历)函数\*/

cout<<"please input the node position you want to delete:"; /\*请求输入删除操作位置\*/

cin>>i;

L=delete(L,i); /\*调用单链表删除函数\*/

cout<<"the Linklist after deleted is:\n";

display(L); /\*调用单链表元素输出(遍历)函数\*/

cout<<"please input the position you want to search:"; /\*请求输入待查找元素的位置\*/

cin>>i;

x=search(L,i); /\*调用单链表查找函数\*/

if (x) /\*如果查找成功，则显示第i个元素的值，否则显示第i个元素不存在\*/

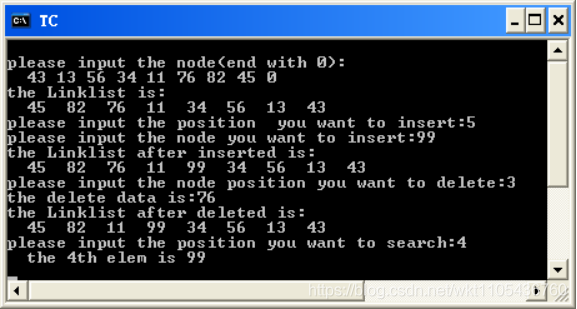
cout<<" the elem is \n",i,x);

else

cout<<"the elem is not exsit\n");

}

4．运行结果参考



**任务二：**

在实验任务一的基础上，实现插入排序

**任务三（选做）：**

1.实验要求

（1）输入链表的长度和各元素的值，用尾插法建立双向循环链表，并输出链表中各元素值，观察输入的内容与输出的内容是否一致。

（2）在双向循环链表的第i个元素之前/后插入一个值为x的元素，并输出插入后的链表中各元素值。

（3）删除双向循环链表中第i个元素，并输出删除后的链表中各元素值。

（4）在双向循环链表中查找值为x元素，如果查找成功，则显示该元素在链表中的位置，否则显示该元素不存在。

2, 对比单链表核心算法提示

双向循环链表采用的存储结构描述如下：

struct DULNODE

{ Elemtype data; /\*数据域\*/

struct DULNODE \*prior; /\*指向前驱结点的指针域\*/

struct DULNODE \*next;/\*指向后继结点的指针域\*/

} DulNODE,\*DuLinklist;

不论是建立双向循环链表还是在双向循环链表中进行插入、删除和查找操作，其操作方法和步骤都跟单链表类似。只不过要注意两点：

（1）凡是在操作中遇到有修改链的地方，都要进行前驱和后继两个指针的修改。

（2）单链表操作算法中的判断条件：p= =NULL 或p!=NULL ,在循环链表的操作算法中则需改为：p!= L,其中L为链表的头指针。