第6次实验

## 单链表概念

单链表用一组地址任意的存储单元存放线性表中的数据元素。由于逻辑上相邻的元素其物理位置不一定相邻，为了建立元素间的逻辑关系，需要在线性表的每个元素中附加其后继元素的地址信息。这种地址信息称为指针。

head

data

next

a1

a2

an

∧

图1 单链表的结点 图2 带头结点的单链表

head

38

数据域

数据域

38

22

94

86

a2 86

… …

94

… …

a3 NULL

a1 22

存储地址

图3 单链表存储结构示意图

头结点空置不用，主要是为了编程的方便。

结点的形式定义如下：

struct LNode

{

ElemType data; // 数据域，ElemType是某种数据类型

struct LNode \*next; // 指针域

} ;

* **语句p=p->next则将p自身从指向结点ai的指针变为指向结点ai+1的指针**
* 单链表的主要算法

⑴求单链表的长度

单链表长度不定，要确定链表长度需要走遍表中所有结点才能算出。

int Length( )

{

LNode \*p=head->next; //p指向第一个元素所在结点

int len=0;

while( p!=NULL ) { //逐个检测p结点存在与否

len++;

p=p->next; //指针后移

}

return len;

}

⑵从单链表中删除第i个结点

为了从单链表中删除第i个结点，需进行如下操作：

第一步，如果第i个结点存在则找到第i和第i-1个结点的指针p和q；

第二步，通过语句q->next=p->next将第i-1个结点的指针**赋值为**第i个结点的指针，从而将第i个结点从链表中断开；

第三步，释放第i个结点所占空间以便于重用。

图4 显示了删除结点前后链表中指针的变化。

(a) 删除前 (b) 删除后

图4 在单链表中删除结点ai

ai-1

ai

ai+1

q

p

ai-1

ai

ai+1

p

q

void Delete(LNode \*head，int i )

{

if(i<1) cout<<”不存在第”<<i<<”个元素”;

else {

LNode \*p=head; //p指向头结点(第0个结点)

LNode \*q; //q和p最终分别指向第i-1和第i个结点

int k=0;

while( p!=NULL&&k<i ) {

q=p;

p=p->next;

k++;

}

if(p==NULL) cout<< i<<”超出链表长度”;

else {

q->next=p->next; //从链表中删除该结点

delete p; //释放结点p

}

}

}

这里位置i从1开始计数。

⑶在第i个位置插入新结点x

为了在链表的第i个位置插入一个新结点，需进行如下操作：

第一步，首先找到第i-1个结点的指针p；

第二步，建立新结点s并通过语句**s->next=p->next**将其指针指向第i个结点；

第三步，通过语句**p->next=s**将第i-1个结点的指针指向新结点。

图5 给出了插入新结点前后链表指针的变化。

ai-1

ai

p

x

s

p

ai-1

x

ai

s

(a) 插入前 (b) 插入后

图5 在单链表中插入结点x

void Insert(LNode \*head, int i, ElemType x)

{

if(i<1) {

cout<<"不存在第"<<i<<"个位置";

} else {

LNode \*p=head; //p最终将指向第i-1个结点

int k=0; //p目前指向第0个结点(头结点)

while( p!=NULL&&k<i-1 )

{

p=p->next;

k++;

}

if(p==NULL) cout<< i<<"超出链表最大可插入位置";

else {

LNode \*s=new LNode; //建立新结点s

s->data = x;

s->next=p->next; //修改结点s的指针

p->next=s; //修改结点p的指针

}

}

}

这里位置i从1开始计数。

## 学生信息管理（单链表）

用单链表形式实现前一次用顺序表实现的学生信息管理，其中学生信息包含学号、姓名，是结构体形式。

// 结点的一种定义形式

struct LNode{

char name[15]; // 姓名

char tel[10]; // 电话

struct LNode \*next; // 指针域

};

// 创建带有头节点的空链表

LNode \*head,\*p; // 定义头指针、临时指针

head = new LNode; // 定义头结点

head->next = NULL; // 头结点指针域为空

1）生成单链表（在仅头部插入，不排序）

【算法】

// 在头部插入结点函数

void Insert(LNode\* head, char\* Name,char\* Tel)

{

p指向头结点

建立新结点s 并赋值

修改结点s的指针

修改结点p（头结点）的指针

}

2）实现按学号查找元素

【算法】

void Display( LNode\* head )

{

LNode \*p;

p指向头结点之后的首元结点

输出提示信息

while( p!=NULL )

{

输出元素信息

移动到下一个元素

}

}

3）实现按学号删除元素

【算法】

// 删除结点函数

void Delete(LNode\* head, char\* Name)

{

p指向头结点之后的首元结点

定义结点指针变量q;

while( p不为空 且 p的学号与要查找的学号不同 )

{

q=p ; // q最终将指向被删除结点的前驱结点

p=p->next; //p最终将指向被删除结点

}

if(p为空)

输出"没有这条记录，无法删除！";

else {

从链表中删除该结点

释放结点p

}

}