西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

链表运算

1、遍历链表

2、销毁链表

3、查找结点

4、链表逆序

9.3 链表的运算

- 双链表与单链表的基本运算大多数是相同的，这里仅讨论单链表的情形。

9.3.1 链表的遍历

- （1）链表遍历ListTraverse(L,visit())
- 与数组不同，链表不是用下标而是用指针运算查找数据元素的。通过链表的头结点L可以访问开始结点p=L->next，令p=p->next，即p指向直接后继结点，如此循环可以访问整个链表中的全部结点，这就是链表的遍历（traverse）。链表的输出、销毁、查找和逆序等运算都需要遍历链表。

9.3.1 链表的遍历

- 链表遍历算法的实现步骤为：
- ①令指针p指向L的开始结点。
- ②若p为0，表示已到链尾，遍历结束。
- ③令p指向直接后继结点，即p=p->next。重复②～③步骤直至遍历结束。

9.3.1 链表的遍历

链表遍历的算法如下：

```
1 void ListTraverse(LinkList L,void(*visit)(ElemType*))
2 { //遍历L中的每个元素且调用函数visit访问它
3   LinkList p=L->next; //p指向开始结点
4   while(p!=NULL) { //若不是链尾继续
5     visit(&(p->data)); //调用函数visit()访问结点
6     p=p->next; //p指向直接后继结点
7   }
8 }
```

其中visit是函数指针。当调用ListTraverse遍历结点时，通过调用visit()对每个结点完成定制的操作。

9.3.1 链表的遍历

- （2）输出链表
- 设计遍历结点时的visit函数：

```
1 void visit(ElemType *ep) //实现链表遍历时结点访问的定制函数
2 { //在函数中对结点*ep实现定制的操作，例如输出
3   printf("%d ",*ep);
4 }
```

- 调用ListTraverse(L,visit)时扫描链表中的每个结点，并调用visit()输出结点的数据域。

9.3.1 链表的遍历

- （3）计算链表长度ListLength(L)
- 应用遍历算法逐个统计链表结点个数的算法如下：

```
1  int ListLength(LinkList L) //返回L中数据元素个数
2  {
3    int cnt=0;
4    LinkList p=L->next; //p指向开始结点
5    while(p!=NULL) { //若不是链尾继续
6      cnt++;
7      p=p->next; //指向直接后继结点
8    }
9    return cnt; //返回0表示无数据结点
10 }
```

9.3.1 链表的遍历

（4）返回链表尾结点元素LastElem(L,&e)

应用遍历算法移动到尾结点，返回尾结点元素的算法如下：

```
1  int LastElem(LinkList L,ElemType *e) //用e返回尾结点元素
2  {
3    LinkList q=NULL, p=L; //指向头结点
4    while (p!=NULL) { //若不是链尾继续
5      q=p;
6      p=p->next; //指向直接后继结点
7    }
8    if (q!=NULL) {
9      *e=q->data;
10     return 1; //操作成功返回真（1）
11   }
12   return 0; //操作失败返回假（0）
13 }
```

9.3.1 链表的遍历

（5）检测是否为循环链表LinkRing(L)

循环链表的特征是尾结点的next是头结点，所以应用遍历算法判断链表是否为循环链表的算法如下：

```
1 int LinkRing(LinkList L) //判断链表是否为循环链表
2 {
3   LinkList p=L; //指向头结点
4   while (p!=NULL) { //若是链尾结束
5     p=p->next; //指向直接后继结点
6     if (p==L) return 1; //是循环链表返回真（1）
7   }
8   return 0; //不是循环链表返回假（0）
9 }
```

9.3.1 链表的遍历

（6）两个链表相连LinkContact(L1,*L2)

通过让链表L1的尾结点指向L2开始结点，将两个链表连接起来的算法如下：

```
1  void LinkContact(LinkList L1,LinkList *L2) //两个链表相连
2  {
3    LinkList q=NULL,p=L1; //p指向链表1头结点
4    while (p!=NULL) { //若是链表1链尾结束
5      q=p;
6      p=p->next; //指向直接后继结点
7    }
8    if (q!=NULL && (*L2)!=NULL) {
9      q->next=(*L2)->next; //链表1尾结点指向链表2开始结点
10     free(*L2); //释放链表2头结点
11     *L2=NULL;
12   }
13 }
```

9.3.2 销毁链表

- （1）销毁链表DestroyList(&L)
- 按照动态内存的使用要求，当不再使用链表时或程序结束前，需要将创建链表时分配的所有结点的内存释放掉，即销毁链表。
- 销毁链表的步骤如下：
- ①若*L为0，表示已到链尾，销毁链表结束。
- ②令指针p指向结点*L的next，释放内存*L。
- ③*L置换为p，即*L指向直接后继结点，重复①～③步骤直至销毁链表结束。

9.3.2 销毁链表

- 应用遍历算法销毁链表的算法如下：

```
1  void DestroyList(LinkList *L) //销毁单链表L
2  {
3    LinkList q,p=*L;//p指向头结点
4    while(p!=NULL) { //若不是链尾继续
5      q=p->next; //指向直接后继结点
6      free(p); //释放结点存储空间
7      p=q; //直接后继结点
8    }
9    *L=NULL;//置为空表
10 }
```

9.3.2 销毁链表

（2）置空链表ClearList(&L)

将一个链表重置为空表（即没有数据结点）的算法如下：

```
1  void ClearList(LinkList *L) //将L重置为空表
2  {
3    LinkList p,q;
4    p=(*L)->next; //p指向开始结点
5    while(p!=NULL) { //若不是链尾继续
6      q=p->next; //指向直接后继结点
7      free(p); //释放结点存储空间
8      p=q; //直接后继结点
9    }
10   (*L)->next=NULL; //初始时为空表
11 }
```

9.3.3 查找结点

- （1）用e返回链表中第i个数据元素GetElem(L,i,&e)
- 应用遍历算法可以实现链表结点的查找，找出指定位置的元素。其步骤为：
- ①令指针p指向L。
- ②若p为0，表示已到链尾，查找结束，未发现给定元素的结点。
- ③若计数器与给定i相同，查找结束，找到给定元素的结点。
- ④令p指向直接后继结点，即p=p->next。重复②～④步骤直至查找结束。

9.3.3 查找结点

应用遍历算法定位链表结点，返回第i个数据元素的算法如下：

```
1  int GetElem(LinkList L,int i,ElemType *e)
2  { //用*e返回L中第i个数据元素
3    LinkList p=L; //p指向头结点
4    while(p!=NULL && i>0) {
5      //顺指针向后查找直到p指向第i个元素或链尾结束
6      p=p->next; //指向直接后继结点
7      i--;
8    }
9    if(p==NULL||p==L) return 0; //第i个元素不存在返回假（0）
10   if (e!=NULL) *e=p->data; //用*e返回第i个元素
11   return 1; //操作成功返回真（1）
12 }
```

9.3.3 查找结点

- （2）返回链表中满足指定数据元素的位序 LocateElem(L,e,compare())
- 应用遍历算法查找链表结点，返回第一个满足定制关系数据元素的位序的算法如下：

9.3.3 查找结点

```
1  int LocateElem(LinkList L,ElemType e,
                  int(*compare)(ElemType*,ElemType*))
2  { //返回L中第1个与e满足关系compare()的数据元素的位序
3    int i=0;
4    LinkList p=L->next; //p指向开始结点
5    while(p!=NULL) { //若不是链尾继续
6      i++;
7      //关系成立时找到指定元素的位序
8      if(compare(&(p->data),&e)) return i;
9      p=p->next; //指向直接后继结点
10   }
11   return 0; //关系不存在返回0
12 }
```

9.3.3 查找结点

- 其中compare是函数指针。当调用LocateElem遍历结点时，通过调用compare()对每个结点与给定完成定制的关系比较，关系成立返回真，否则返回假。如相等比较为

```
1 //实现两个数据元素关系比较的定制函数
2 int compare(ElemType *ep1,ElemType *ep2)
3 { //在函数中对数据元素进行定制的关系比较，如相等，大于或小于
4   if (*ep1==*ep2) return 1; //满足相等关系返回真（1）
5   return 0; //不满足关系返回假（0）
6 }
```

9.3.3 查找结点

- （3）返回链表中指定元素的前驱元素
PriorElem(L,cur_e,&pre_e)
- 应用遍历算法查找链表结点，返回链表中指定元素的前驱元素的算法如下：

9.3.3 查找结点

```
1  int PriorElem(LinkList L,ElemType cur_e,ElemType *pre_e)
2  { //用pre_e返回cur_e元素的前驱
3    LinkList q,p=L->next; //p指向开始结点
4    while(p!=NULL) { //若不是链尾继续
5      q=p->next; //q为p的直接后继结点
6      if(q!=NULL&&q->data==cur_e&&pre_e!=NULL) {
7        *pre_e=p->data; //*pre_e返回前驱元素
8        return 1; //操作成功返回真（1）
9      }
10     p=q; //p指向直接后继结点
11   }
12   return 0; //不存在cur_e返回假（0）
13 }
```

9.3.3 查找结点

- （4）返回链表中指定元素的后继元素
NextElem(L,cur_e,&next_e)
- 应用遍历算法查找链表结点，返回链表中指定元素的后继元素的算法如下：

9.3.3 查找结点

```
1  int NextElem(LinkList L,ElemType cur_e,ElemType *next_e)
2  { //用next_e返回cur_e元素的后继
3    LinkList p=L->next; //p指向开始结点
4    while(p!=NULL) { //若不是链尾继续
5      if(p->data==cur_e) {
6        if (p->next!=NULL&&next_e!=NULL)
7          *next_e=p->next->data; //*next_e返回后继元素
8        return 1; //操作成功返回真（1）
9      }
10     p=p->next; //p指向直接后继结点
11   }
12   return 0; //不存在cur_e返回假（0）
13 }
```

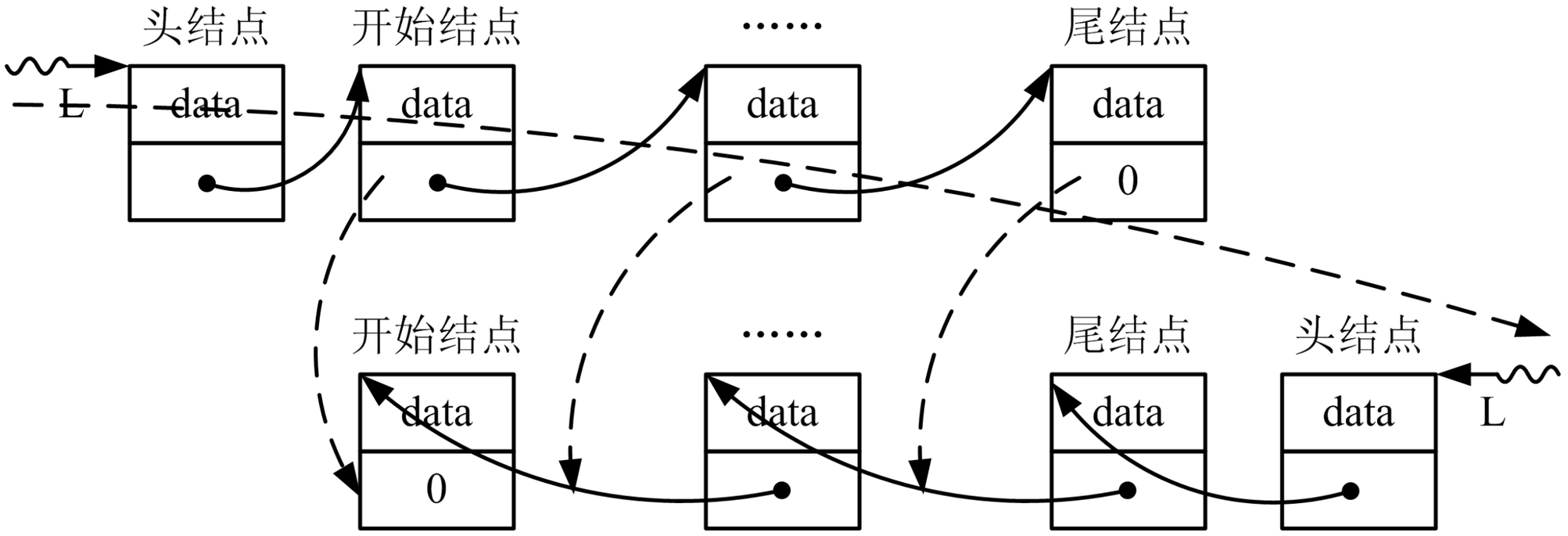
9.3.4 链表的逆序

- 链表的逆序是指将原始链表中开始结点变为尾结点，尾结点变为开始结点。

- 逆序的实现是从链表开始结点建立前向链p，移动前向链判断是否到链尾，移动中建立后向链q，最后修改头结点指向后向链。如果是循环链表，则逆序实现起来要方便一些。

9.3.4 链表的逆序

图9.7 链表逆序示意




程序设计