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

用链表表示动态“数组”

1、链表的基本概念

2、创建链表

- 使用数组时必须事先定义好数组长度（即元素个数），这个长度一经定义就是固定不变的。如果事先难以确定元素个数，则必须把数组长度定义得足够大，这将占用许多内存。另一方面，在数组中若要插入或删除某个元素，需要移动插入点或删除点后面所有的数组元素，这将运行大量时间。

- 链表是一种存储空间能动态进行增长或缩小的数据结构。链表主要用于两个目的：
- 一是建立不定长度的数组。
- 二是链表可以在不重新安排整个存储结构的情况下，方便且迅速地插入和删除数据元素。

- 由于这些原因，链表广泛地运用于数据管理中。

9.1 链表概述

- 首先设计一种称为结点（node）的数据类型：

```
typedef struct tagNODE { //结点数据类型
    ElemType data; //数据域
    struct tagNODE *link; //指针域
} NODE;
```

9.1.1 链表的概念

- 这个结构体类型中，data成员表示数据域，代表结点的数据信息。ElemType可以是简单的内置数据类型，也可以是复杂的数据类型，如

```
typedef struct tagElemType { //复杂的数据元素类型
   ..... //任意数目、任意组合、任意类型的数据成员
} ElemType;
```

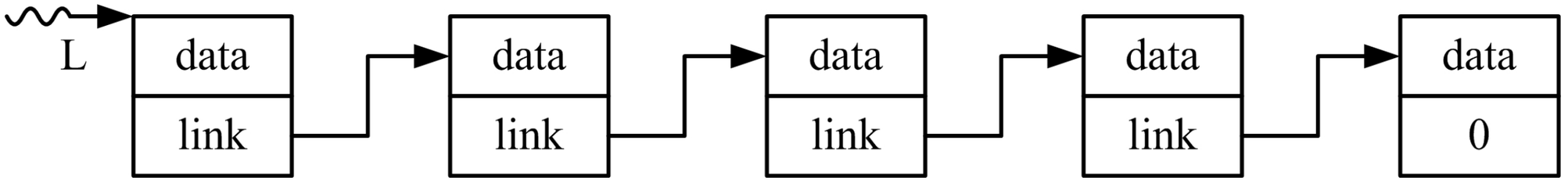
9.1.1 链表的概念

- 数据域是链表中的信息对象（元素），实际应用中结合具体要求设计其数据类型。为方便介绍，这里将ElemType简单设定为int型，即

```
typedef int ElemType; //简单的数据元素类型
```

9.1.1 链表的概念

- link成员表示指针域，存放另一个结点的地址，是链表中的组织者。假定有一个NODE类型的对象指针L，将一个新结点的地址赋给L的link成员，则L可以通过它的link成员“链接”到新结点上，重复这个过程可以得到如图的链表结构。



9.1.1 链表的概念

- 可以看出，链表是一种物理存储非连续（非顺序）的存储结构，数据元素的逻辑顺序是通过链表中的指针域实现的。

- 链表由一系列结点组成，结点可以在运行时动态生成，因而链表可以动态增长或缩短。同时，结点是按指针域连接起来的，插入或删除结点非常简便和迅速。通常，链表包含创建、遍历、插入、删除等运算。

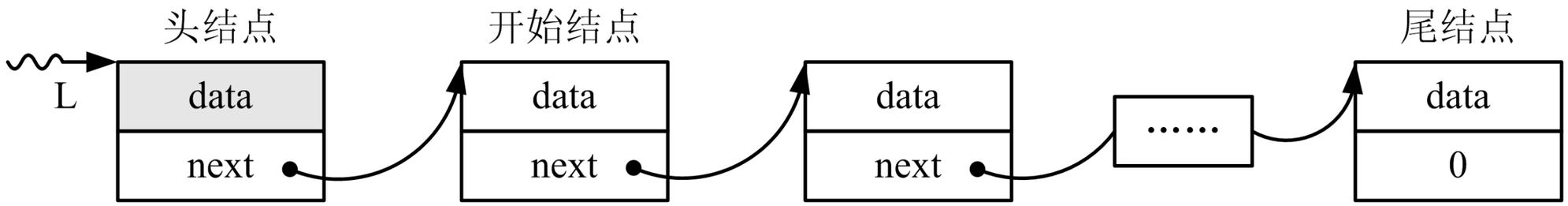
9.1.2 单链表与双链表

- 1. 单链表
- 单链表每个结点包含一个指向直接后继结点的指针域，其形式为：

```
typedef struct tagLNode { //单链表结点类型
    ElemType data; //数据域
    struct tagLNode *next; //指针域：指向直接后继结点
}LNode,*LinkList;
//LNode为单链表结构体类型，LinkList为单链表指针类型
```

9.1.2 单链表与双链表

- next指向直接后继结点，由它构成了一条链。



9.1.2 单链表与双链表

- 指针L指向单链表头结点，头结点指向开始结点，开始结点又指向下一个结点，……，直到最后一个尾结点。尾结点的next为0，表示NULL指针，约定单链表的结点的next为0时表示尾结点。上述链表称为带头结点的单链表，若开始结点为头结点，则称这样的链表为不带头结点的单链表。

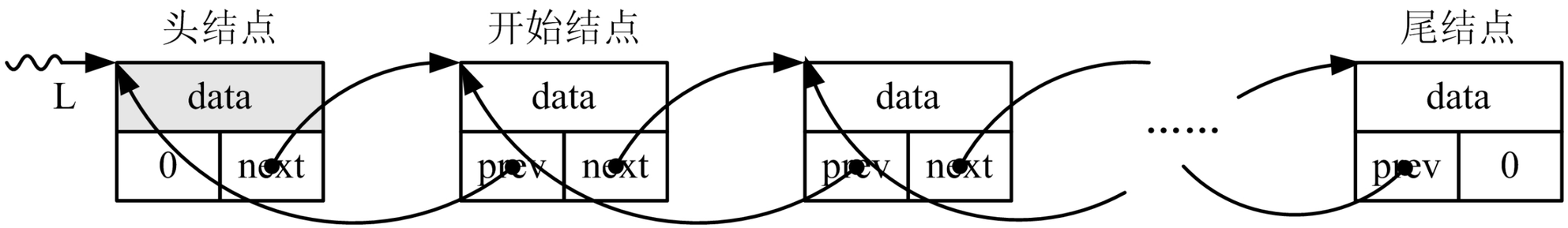
9.1.2 单链表与双链表

- 2．双链表
- 双链表每个结点包含指向前驱结点和指向直接后继结点的指针域，其形式为：

```
typedef struct tagDNode { //双链表结点类型
    ElemType data; //数据域
    struct tagDNode *prev,*next;
  //指针域：分别指向前驱结点和直接后继结点
} DNode, *DLinkList;
//DNode为双链表结构体类型，DLinkList为双链表指针类型
```

9.1.2 单链表与双链表

- next指向直接后继结点，prev指向前驱结点，由它们构成了两条链。



9.1.2 单链表与双链表

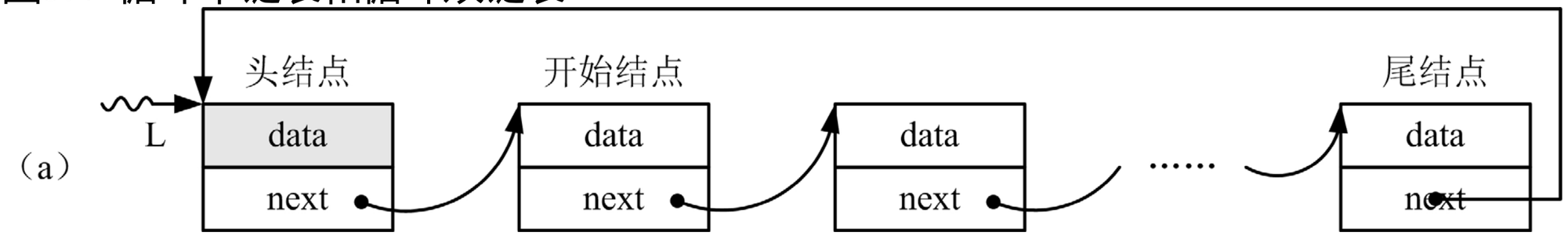
- 指针L指向双链表头结点，其每个结点分别有指向前一个结点和后一个结点的指针。沿着next指针，头结点指向开始结点，开始结点又指向下一个结点，……，直到尾结点，尾结点的next为0。沿着prev指针，尾结点指向前一个结点，直到头结点head，头结点的prev为0。约定双链表的结点的next为0时表示尾结点，prev为0时表示头结点。双链表也有带头结点和不带头结点之分。

9.1.2 单链表与双链表

- 与单链表相比，双链表可以从前向链和后向链遍历整个链表，这样简化了链表排序方法及运算。同时，当一个链数据受损（如数据库设备故障）时可以根据另一个链来恢复它。

- 3．循环链表
- 若单链表尾结点指向头结点而不是0，则该链表是循环单链表。

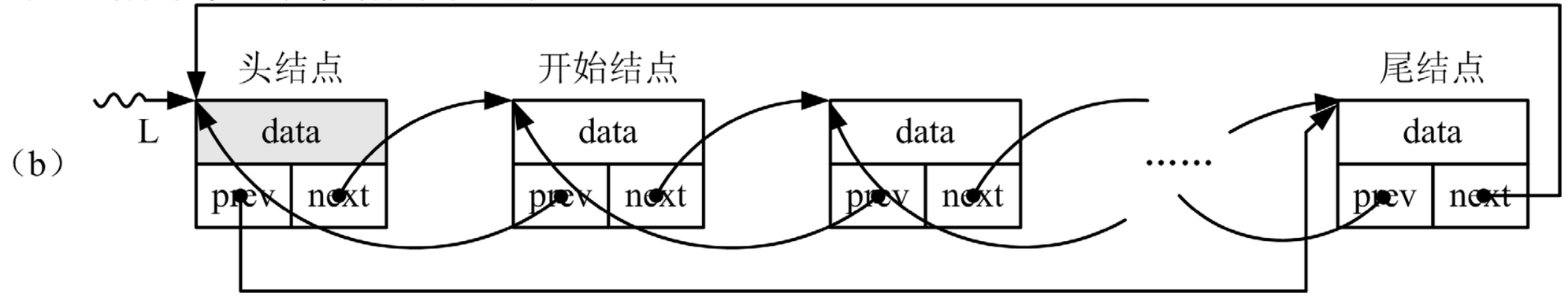
图9.4 循环单链表和循环双链表



9.1.2 单链表与双链表

- 若双链表尾结点next指向头结点而不是0，头结点prev指向尾结点而不是0，则该链表是循环双链表。

图9.4 循环单链表和循环双链表



9.2 链表的创建

- 在程序中欲要使用链表，需要创建链表。

9.2.1 创建单链表

- 通过前面的内存动态分配技术可以产生新结点的内存单元，例如：

```
LinkList p; //链表指针
p=(LinkList)malloc(sizeof(LNode));
//分配LNode类型内存单元且将地址保存到p中
```

9.2.1 创建单链表

- 调用malloc、realloc内存分配函数或free释放函数时，在程序中需要文件包含命令：

```
#include <stdlib.h>
```

9.2.1 创建单链表

- 创建链表常用两种方法：头插法和尾插法。

9.2.1 创建单链表

- （1）头插法建立链表CreateLinkF(&L,n,input())
- 该方法先建立一个头结点*L，然后产生新结点，设置新结点的数据域；再将新结点插入到当前链表的表头，直至指定数目的元素都增加到链表中为止。其步骤为：
- ①创建头结点*L，设置*L的next为0。
- ②动态分配一个结点s，输入s的数据域。
- ③将s插入到开始结点之前，头结点之后，即s->next=p->next，p->next=s。
- ④重复②～④步骤加入更多结点。

9.2.1 创建单链表

采用头插法创建单链表的算法如下：

```
1  void CreateLinkF(LinkList *L,int n,void(*input)(ElemType*))
2  { //头插法创建单链表，调用input输入函数输入数据
3    LinkList s;
4    *L=(LinkList)malloc(sizeof(LNode)); //创建头结点
5    (*L)->next=NULL; //初始时为空表
6    for (; n>0; n--) { //创建n个结点链表
7      s=(LinkList)malloc(sizeof(LNode)); //创建新结点
8      input(&s->data); //调用input输入数据域
9      s->next=(*L)->next; //将s增加到开始结点之前
10     (*L)->next=s; //头结点之后
11   }
12 }
```

9.2.1 创建单链表

- 参数L表示将要创建出来的单链表指针，之所以是LinkList类型的指针类型（即指针的指针），其原因是需要将链表返回到调用函数中。n表示创建链表时需要输入的元素数目，由实际应用中的具体要求确定。

9.2.1 创建单链表

- 考虑到数据域可能是复杂的数据类型，输入不止是一个简单的scanf。因此设计input函数指针，CreateLinkF调用它实现定制的数据域输入。例如

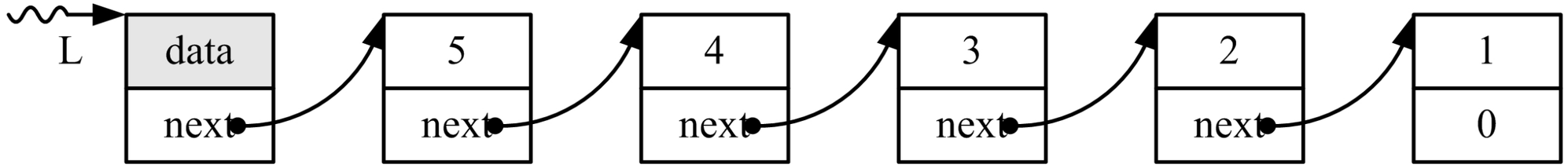
```
void input(ElemType *ep) //实现数据域元素输入的定制函数
{     //在函数中可以写更加复杂、任意形式、任意数目的输入
    scanf("%d",ep);
}
```

9.2.1 创建单链表

运行时若输入5个元素：1、2、3、4、5，则调用

```
LinkList L;
CreateLinkF(&L,5,input); //创建单链表L
```

建立的单链表如图所示。



9.2.1 创建单链表

- （2）尾插法建立链表CreateLinkR(&L,n,input())
- 头插法建立的链表中结点的次序与元素输入的顺序相反，若希望两者次序一致，可采用尾插法建立链表。该方法是将新结点插到当前链表的末尾上，其步骤为：
- ①创建头结点*L，设置*L的next为0，且令指针p指向*L。
- ②动态分配一个结点s，输入s的数据域。
- ③将s插入到当前链表末尾，即p->next=s，p=s。
- ④重复②～④步骤加入更多结点。

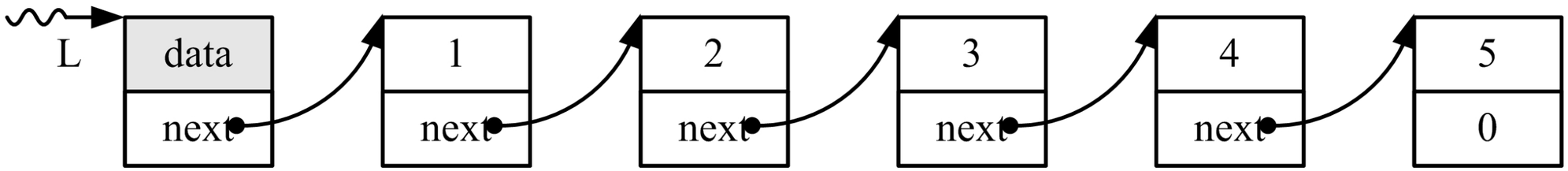
9.2.1 创建单链表

采用尾插法创建单链表的算法如下：

```
1  void CreateLinkR(LinkList *L,int n,void(*input)(ElemType*))
2  { //尾插法创建单链表，调用input输入函数输入数据
3    LinkList p,s;
4    p=*L=(LinkList)malloc(sizeof(LNode)); //创建头结点
5    for (; n>0; n--) { //创建n个结点链表
6      s=(LinkList)malloc(sizeof(LNode)); //创建新结点
7      input(&s->data); //调用input输入数据域
8      p->next=s, p=s; //将s插入到当前链表末尾
9    }
10   p->next=NULL; //尾结点
11 }
```

9.2.1 创建单链表

运行时若输入5个元素：1、2、3、4、5，则调用CreateLinkR(&L,5,input)建立的单链表如图所示。



9.2.1 创建单链表

- （3）初始化链表InitList(&L)
- 构造一个空链表（即没有数据结点）的算法如下：

```
1 void InitList(LinkList *L) //构造一个空的单链表L
2 {
3  *L=(LinkList)malloc(sizeof(LNode));//创建头结点，*L指向头结点
4  (*L)->next=NULL; //初始时为空表
5 }
```

9.2.1 创建单链表

- （4）判断链表是否为空表ListEmpty(L)
- 检测一个链表是否为空表的算法如下：

```
1 int ListEmpty(LinkList L) //检测L是否为空表
2 { //若L为空表返回TRUE，否则返回FALSE
3   return L->next==NULL;
4 }
```

9.2.2 创建双链表

- 双链表每个结点有两个指针域，next指向直接后继结点，prev指向前驱结点。建立双链表的过程与单链表相似，只是需要再处理prev指针，建立前向链即可。

9.2.2 创建双链表

采用头插法创建双链表的算法如下：

```
1  void CreateLinkF(DLinkList *L,int n,
                    void(*input)(ElemType*))
2  { //头插法创建双链表，调用input输入函数输入数据
3    DLinkList s;
4    *L=(DLinkList)malloc(sizeof(DNode)); //创建头结点
5    (*L)->prev=(*L)->next=NULL; //初始时为空表
6    for (; n>0; n--) { //创建n个结点链表
7     s=(DLinkList)malloc(sizeof(DNode)); //创建新结点
8     input(&s->data); //调用input输入数据域
9     s->next=(*L)->next; //将s增加到开始结点之前
10    if ((*L)->next!=NULL) (*L)->next->prev=s;//开始结点之前是s
11     (*L)->next=s , s->prev=*L; //头结点之后是s，s之前是头结点
12   }
13 }
```

9.2.2 创建双链表

采用尾插法创建双链表的算法如下：

```
1  void CreateLinkR(DLinkList *L,int n,
                    void(*input)(ElemType*))
2  { //尾插法创建双链表，调用input输入函数输入数据
3    DLinkList p,s;
4    p=*L=(DLinkList)malloc(sizeof(DNode)); //创建头结点
5    (*L)->prev=(*L)->next=NULL; //初始时为空表
6    for (; n>0; n--) { //创建n个结点链表
7      s=(DLinkList)malloc(sizeof(DNode)); //创建新结点
8      input(&s->data); //调用input输入数据域
9      s->next=NULL; //当前结点是尾结点
10     p->next=s , s->prev=p , p=s; //将s增加到链尾
11   }
12 }
```

9.2.2 创建双链表

- 构造一个空的双链表（即没有数据结点）的算法如下：

```
1 void InitList(DLinkList *L) //构造一个空的单链表L
2 {
3   *L=(DLinkList)malloc(sizeof(DNode));
    //创建头结点，*L指向头结点
4   (*L)->prev=(*L)->next=NULL; //初始时为空表
5 }
```


程序设计