西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C++程序设计
Programming in C++

1011018

主讲：魏英，计算机学院

自定义数据类型的应用——链表

1、链表的概念和分类

2、创建单链表

- 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。
- 链表广泛地运用于数据管理中。

- 首先设计一种称为结点（node）的数据类型：

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

- 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。

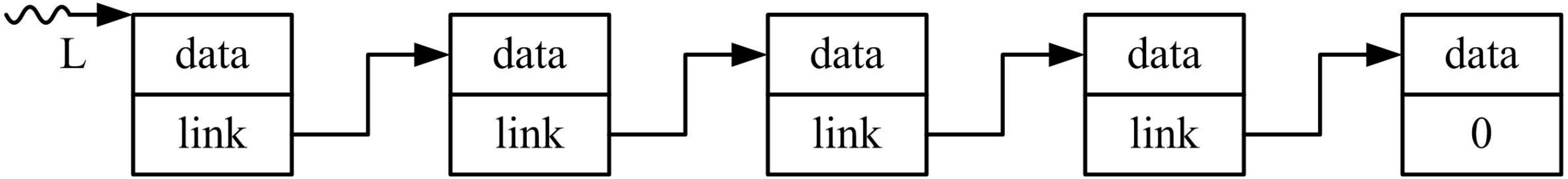
- ElemType可以是简单的内置数据类型，也可以是复杂的数据类型，如

```
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;
```

- 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，将ElemType简单设定为int型，即

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

- 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链表结构。

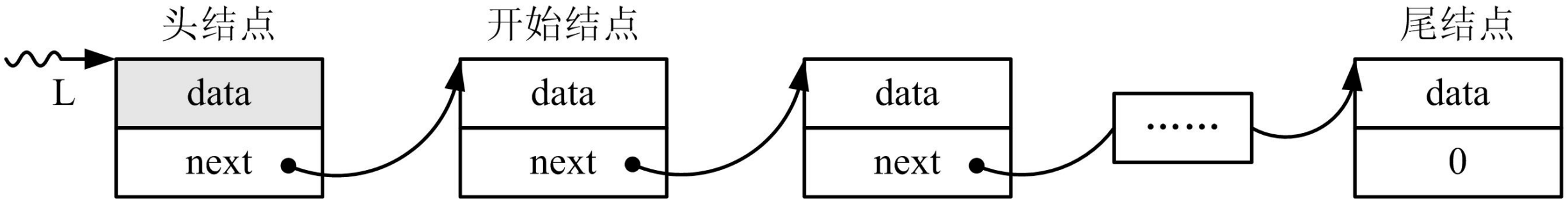


- 链表的分类：
- （1）单链表
- 单链表每个结点包含一个指向直接后继结点的指针域，其形式为：

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

- next指向直接后继结点，由它构成了一条链。

图22.1 单链表结构

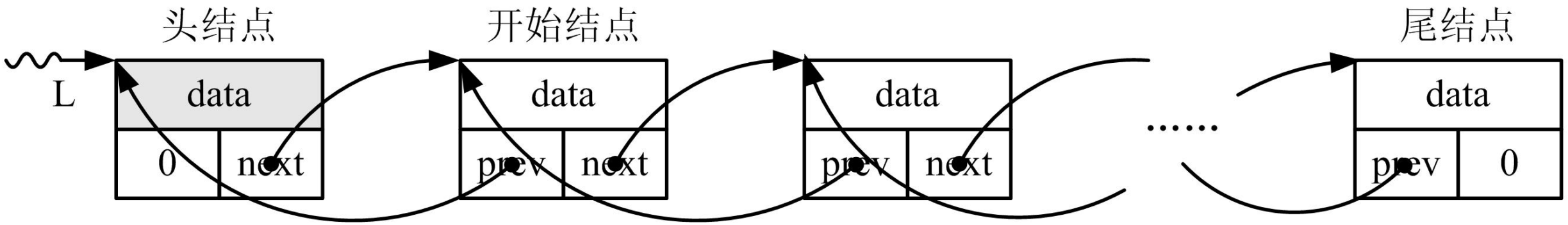


- 指针L指向单链表头结点，头结点指向开始结点，开始结点又指向下一个结点，……，直到最后一个尾结点。尾结点的next为0，表示NULL指针，约定单链表的结点的next为0时表示尾结点。上述链表称为带头结点的单链表，若开始结点为头结点，则称这样的链表为不带头结点的单链表。

- （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：

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

图22.2 双链表结构



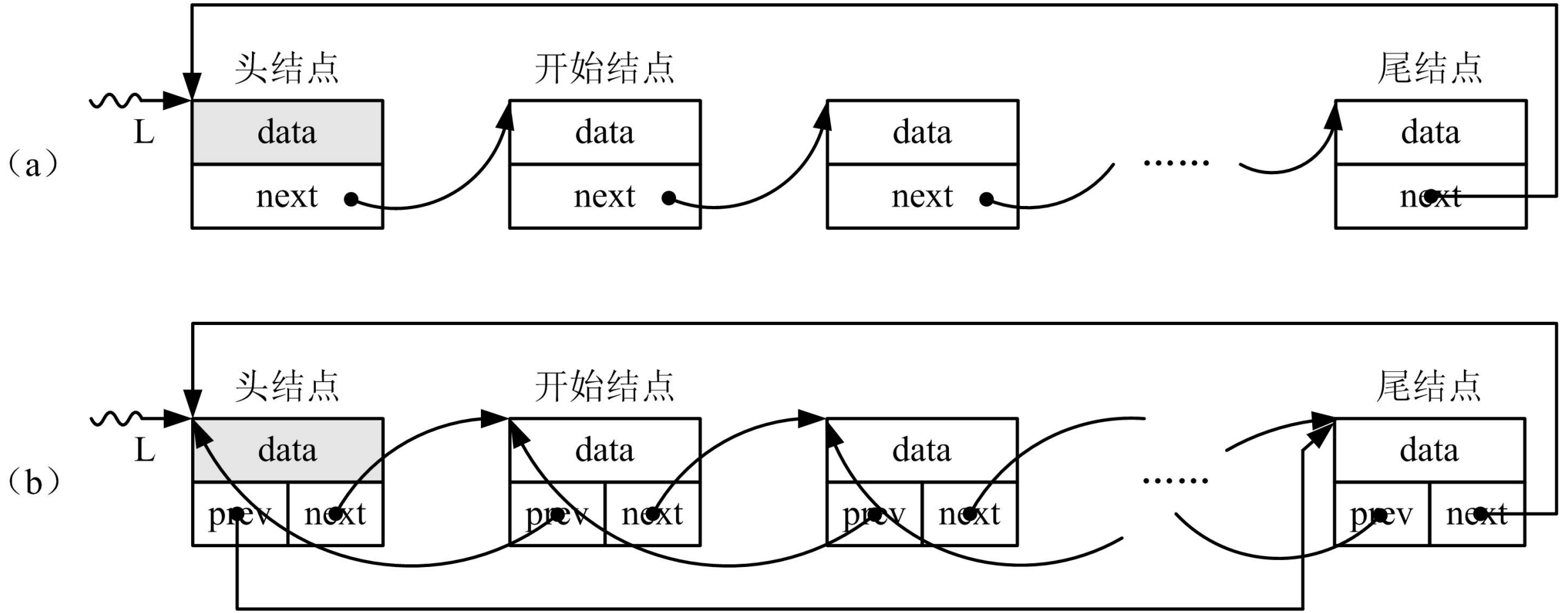
- 指针L指向双链表头结点，其每个结点分别有指向前一个结点和后一个结点的指针。沿着next指针，头结点指向开始结点，开始结点又指向下一个结点，……，直到尾结点，尾结点的next为0。沿着prev指针，尾结点指向前一个结点，直到头结点head，头结点的prev为0。约定双链表的结点的next为0时表示尾结点，prev为0时表示头结点。双链表也有带头结点和不带头结点之分。

- 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。

- 同理，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22.3 循环单链表和循环双链表



- 通过前面介绍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=new LNode;//分配LNode类型内存单元且将地址保存到p中
```

- 创建链表常用两种方法：头插法和尾插法。
- （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。
- ④重复②～④步骤加入更多结点。

```
#include <iostream>
using namespace std;
typedef int ElemType; //简单的数据元素类型
struct LNode { //单链表结点类型
    ElemType data; //数据域
    LNode *next; //指针域：指向直接后继结点
};
typedef LNode* LinkList; //LNode为单链表结构体类型，LinkList为单链表指针类型
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
    cin>>*ep;
}
```

```
13 void CreateLinkF(LinkList *L,int n,void(*input)(ElemType*))
14 { //头插法创建单链表，调用input输入函数输入数据
15       LinkList s;
16     *L=new LNode;//创建头结点
17     (*L)->next=NULL; //初始时为空表
18     for (; n>0; n--) { //创建n个结点链表
19         s=new LNode; //创建新结点
20         input(&s->data); //调用input输入数据域
21         s->next=(*L)->next; //将s增加到开始结点之前
22         (*L)->next=s; //头结点之后
23     }
24 }
25 int main()
26 {
27     LinkList L; int n;  cin>>n;
28     CreateLinkF(&L,n,input);
29 }
```

- （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。
- ④重复②～④步骤加入更多结点。

```
#include <iostream>
using namespace std;
typedef int ElemType; //简单的数据元素类型
struct LNode { //单链表结点类型
    ElemType data; //数据域
    LNode *next; //指针域：指向直接后继结点
};
typedef LNode* LinkList; //LNode为单链表结构体类型，LinkList为单链表指针类型
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
    cin>>*ep;
}
```

```
13 void CreateLinkR(LinkList *L,int n,void(*input)(ElemType*))
14 { //尾插法创建单链表，调用input输入函数输入数据
15     LinkList p,s;
16     p=*L=new LNode; //创建头结点
17     for (; n>0; n--) { //创建n个结点链表
18         s=new LNode; //创建新结点
19         input(&s->data); //调用input输入数据域
20         p->next=s, p=s; //将s插入到当前链表末尾
21     }
22     p->next=NULL; //尾结点
23 }
24 int main()
25 {
26     LinkList L; int n; cin>>n;
27     CreateLinkR(&L,n,input);
28 }
```

- （3）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。

```
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          delete 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 }
```

C程序设计