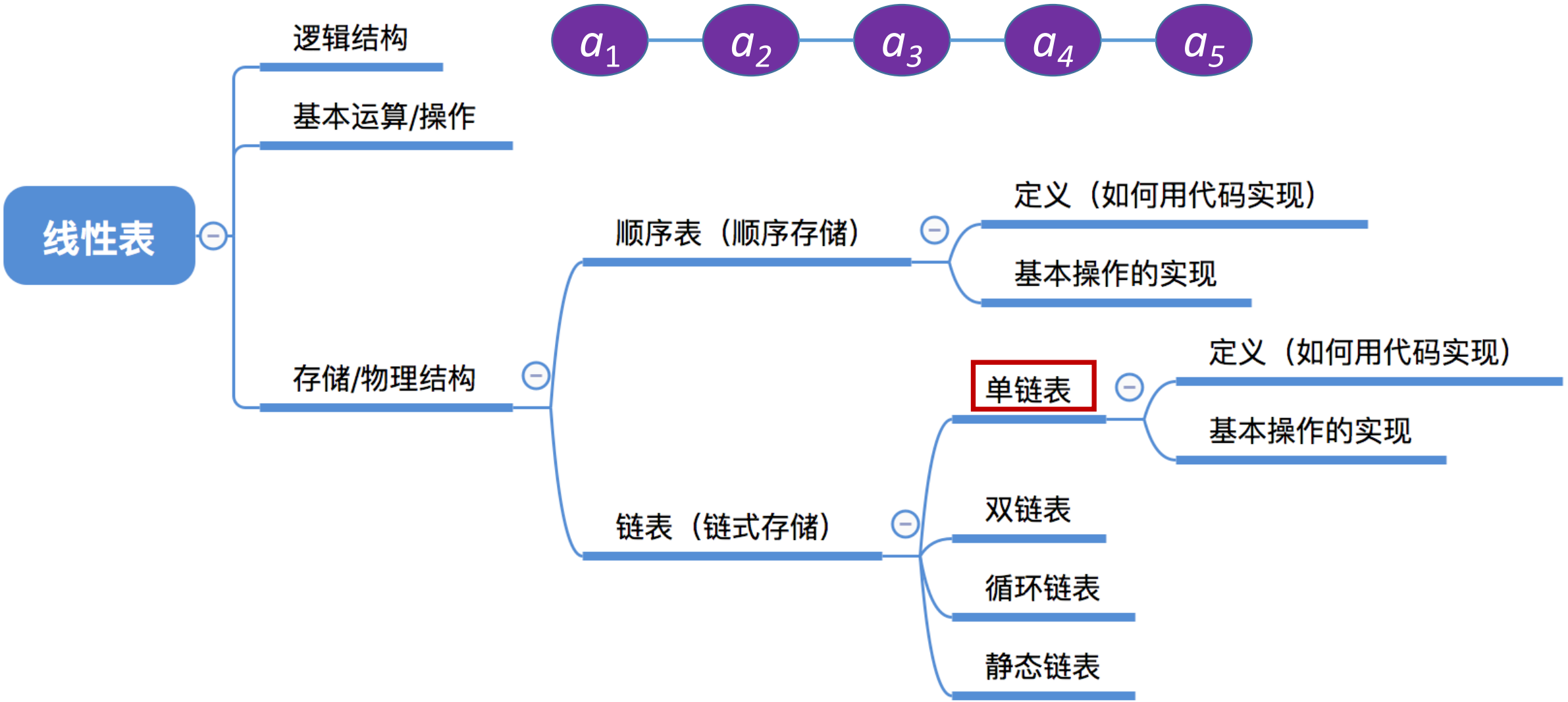
本节内容

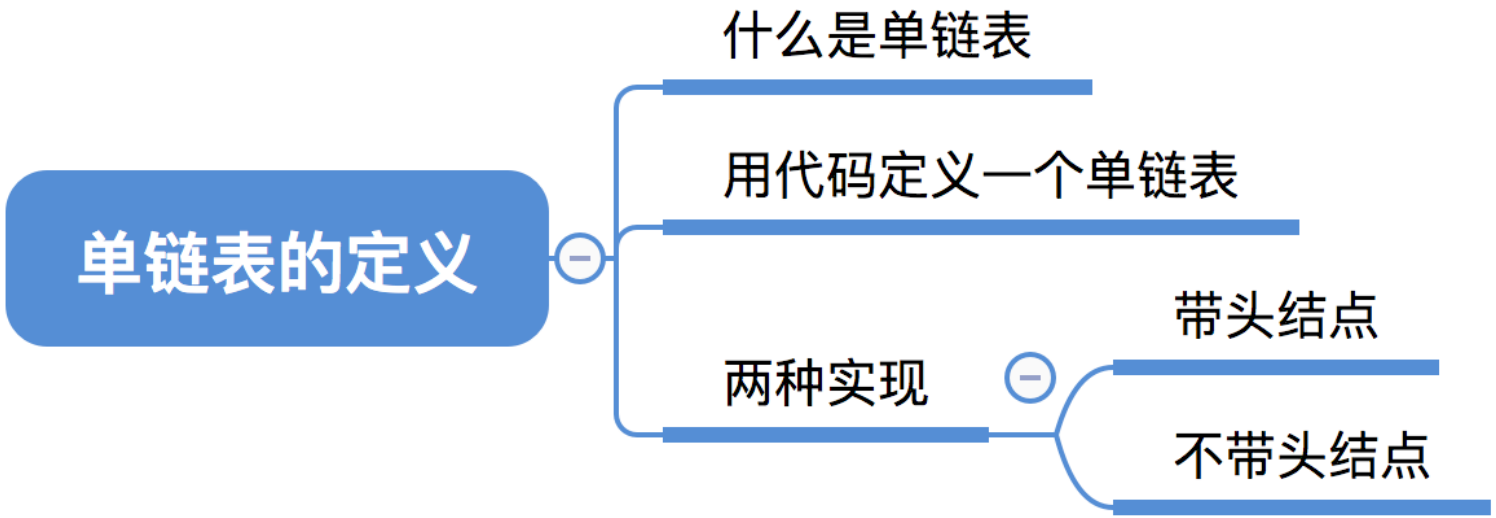
单链表

定义

知识总览

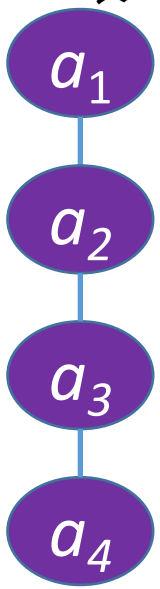


知识总览

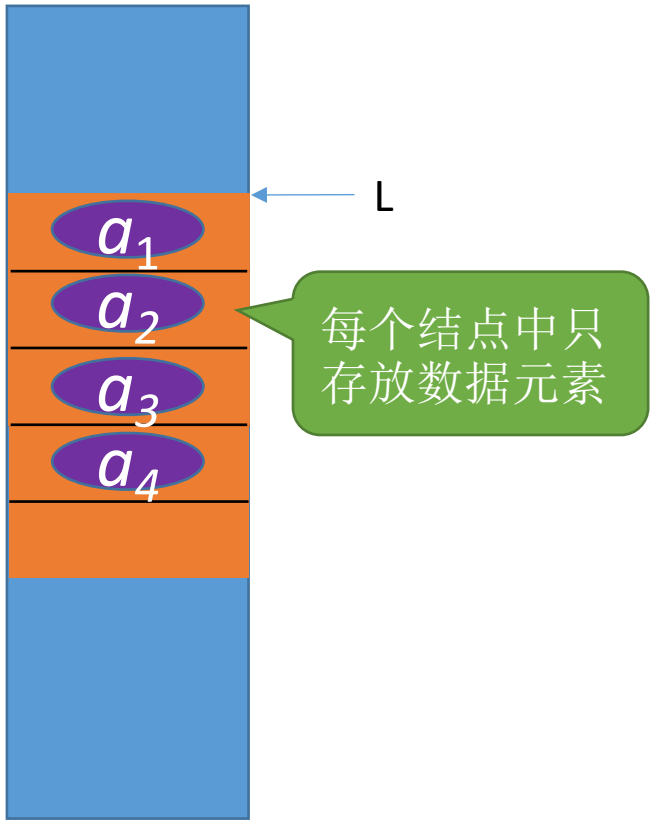


什么是单链表

逻辑结构：线性表

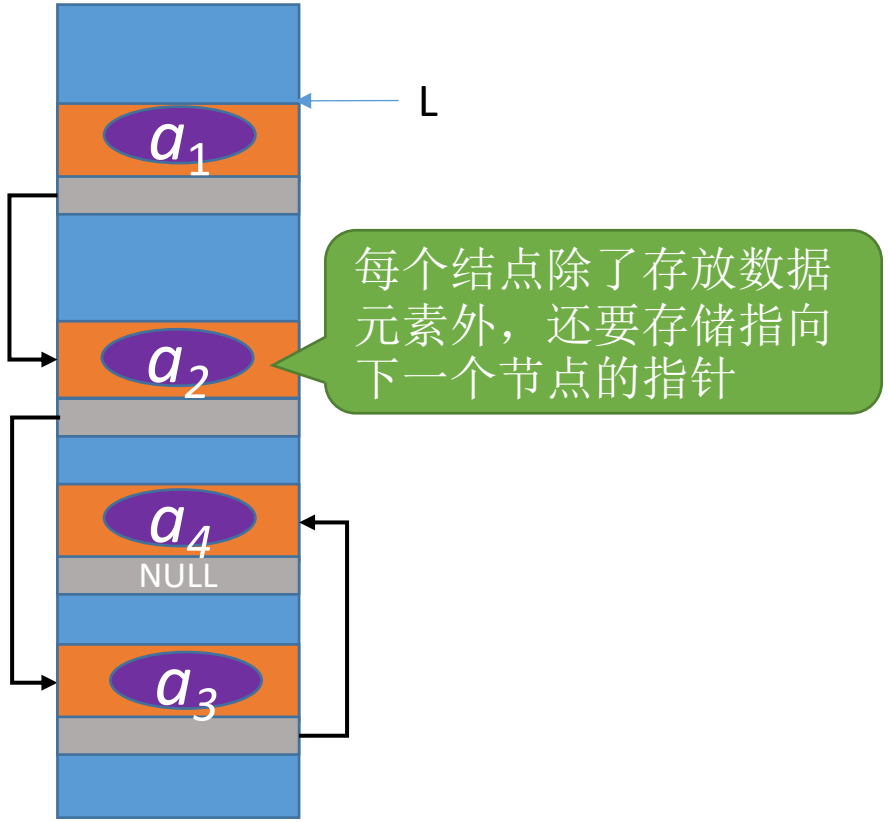


顺序表
（顺序存储）



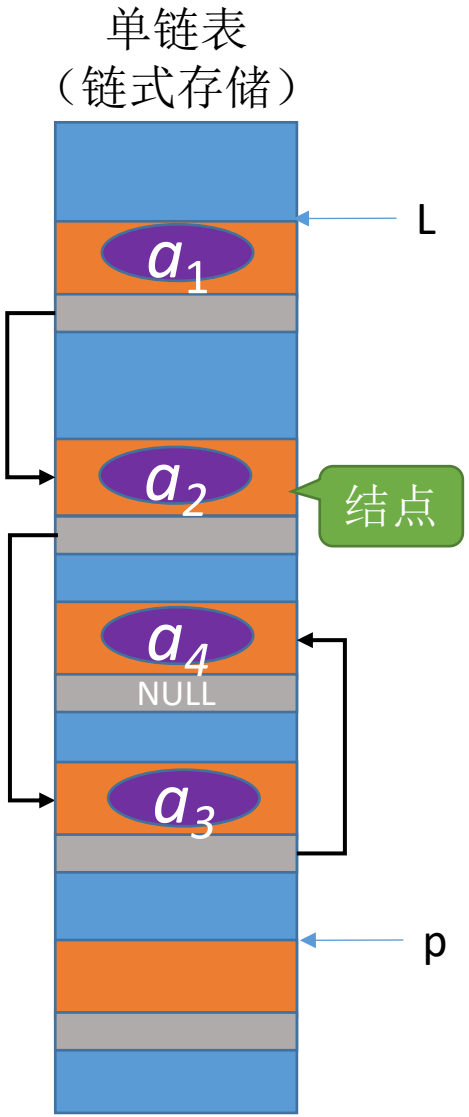
优点：可随机存取，存储密度高
缺点：要求大片连续空间，改变容量不方便

单链表
（链式存储）



优点：不要求大片连续空间，改变容量方便
缺点：不可随机存取，要耗费一定空间存放指针

用代码定义一个单链表



结点

```
struct LNode{                    //定义单链表结点类型
    ElemType data;               //每个节点存放一个数据元素
    struct LNode *next;          //指针指向下一个节点
};
```

数据域

指针域

```
struct LNode * p = (struct LNode *) malloc(sizeof(struct LNode));
```

增加一个新的结点：在内存中申请一个结点所需空间，并用指针 p 指向这个结点

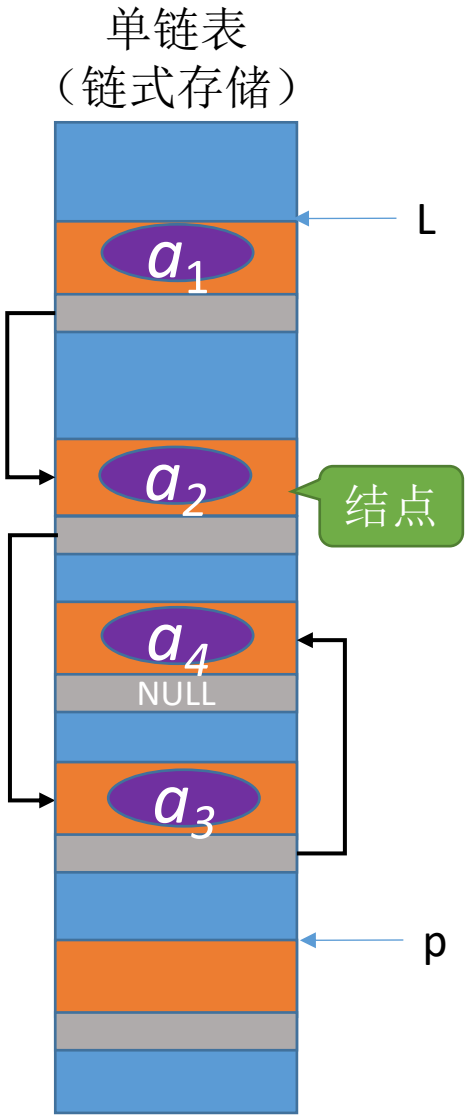
typedef 关键字 —— 数据类型重命名

```
typedef <数据类型> <别名>

typedef int zhengshu;
typedef int *zhengshuzhizhen;

int x = 1;
int *p;
```

等价

```
zhengshu x = 1;
zhengshuzhizhen p;
```

用代码定义一个单链表



结点

```
struct LNode{                   //定义单链表结点类型
    ElemType data;              //每个节点存放一个数据元素
    struct LNode *next;         //指针指向下一个节点
};
```

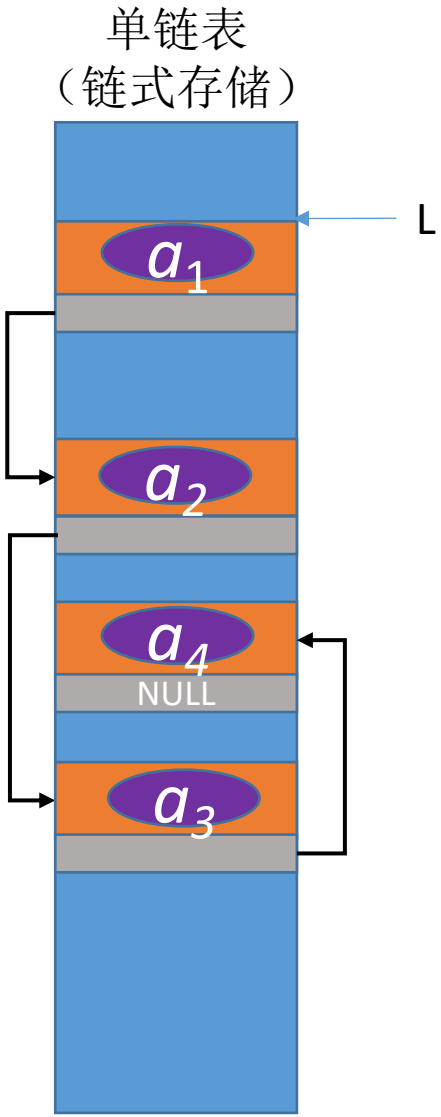
```
struct LNode * p = (struct LNode *) malloc(sizeof(struct LNode));
```

增加一个新的结点：在内存中申请一个结点所需空间，并用指针 p 指向这个结点

typedef 关键字 —— 数据类型重命名

```
typedef  <数据类型>  <别名>
typedef struct LNode LNode;
LNode * p = (LNode *) malloc(sizeof(LNode));
```

用代码定义一个单链表




```
typedef struct LNode{                  //定义单链表结点类型
    ElemType data;                     //每个节点存放一个数据元素
    struct LNode *next;                //指针指向下一个节点
}LNode, *LinkList;
```

LinkList
网络释义
– 单链表

```
struct LNode{                          //定义单链表结点类型
    ElemType data;                     //每个节点存放一个数据元素
    struct LNode *next;                //指针指向下一个节点
};
typedef struct LNode LNode;
typedef struct LNode *LinkList;
```

要表示一个单链表时，只需声明一个头指针 L，指向单链表的第一个结点

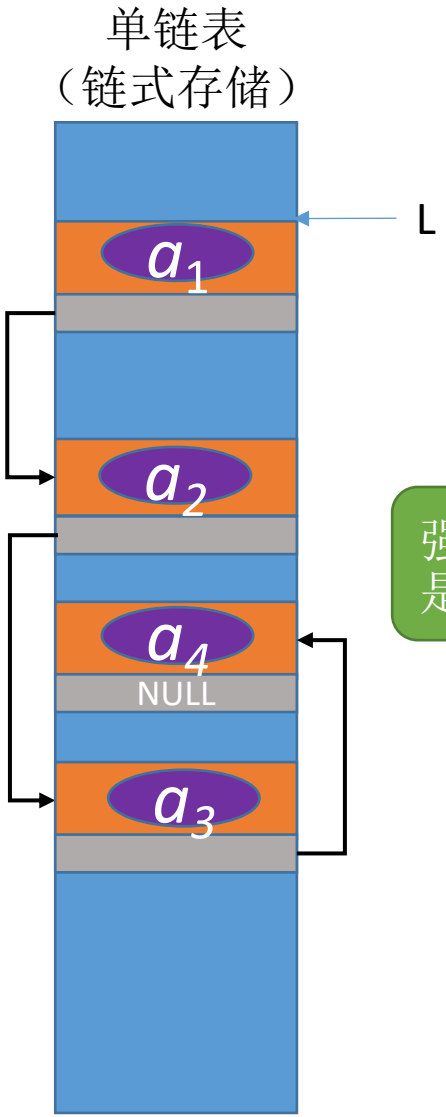
```
LNode * L;  //声明一个指向单链表第一个结点的指针
```

或：

```
LinkList L; //声明一个指向单链表第一个结点的指针
```

代码可读性更强

用代码定义一个单链表




```
typedef struct LNode{                    //定义单链表结点类型
    ElemType data;                       //每个节点存放一个数据元素
    struct LNode *next;                  //指针指向下一个节点
}LNode, *LinkList;

LNode * GetElem(LinkList L, int i){
    int j=1;
    LNode *p=L->next;
    if(i==0)
        return L;
    if(i<1)
        return  NULL;
    while(p!=NULL && j<i){
        p=p->next;
        j++;
    }
    return  p;
}
```

强调返回的是一个结点

强调这是一个单链表

强调这是一个单链表 ——使用 LinkList

强调这是一个结点 ——使用 LNode *

用代码定义一个单链表

头插法建立单链表的算法如下：

```
LinkList List_HeadInsert(LinkList &L){//逆向建立单链表
    LNode *s; int x;
    L=(LinkList)malloc(sizeof(LNode)); //创建头结点
    L->next=NULL;                      //初始为空链表
    scanf("%d",&x);                    //输入结点的值
    while(x!=9999){                    //输入 9999 表示结束
        s=(LNode*)malloc(sizeof(LNode));//创建新结点
        s->data=x;
        s->next=L->next;
        L->next=s;                     //将新结点插入表中，L 为头指针
        scanf("%d",&x);
    }
    return  L;
}
```

强调这是一个单链表　　——使用 LinkList

强调这是一个结点　　——使用 LNode *

不带头结点的单链表

```
typedef struct LNode{                //定义单链表结点类型
    ElemType data;                   //每个节点存放一个数据元素
    struct LNode *next;              //指针指向下一个节点
}LNode, *LinkList;

//初始化一个空的单链表
bool InitList(LinkList &L) {
    L = NULL;       //空表，暂时还没有任何结点
    return true;
}

void test(){
    LinkList L;     //声明一个指向单链表的指针
    //初始化一个空表
    InitList(L);
    //.....后续代码.....
}
```

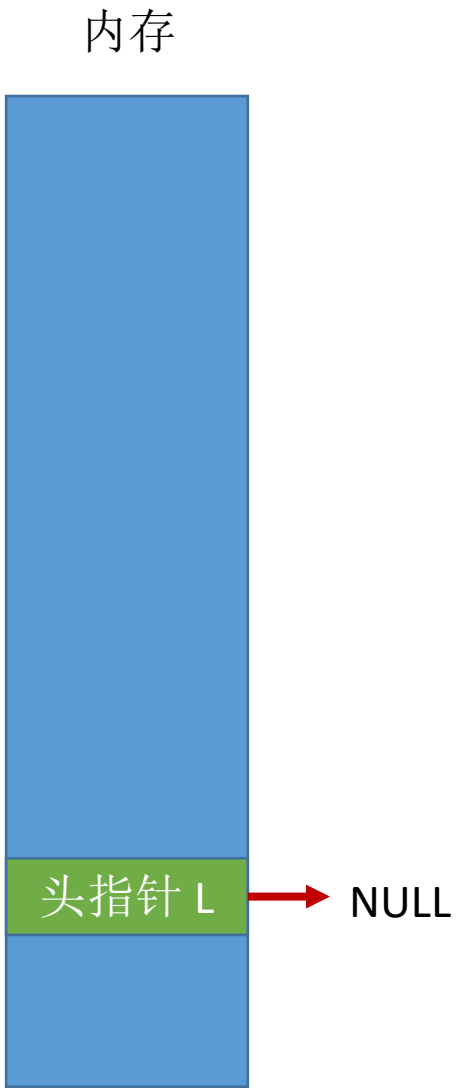
防止脏数据

注意，此处并没有创建一个结点

```
//判断单链表是否为空
bool Empty(LinkList L) {
    if (L == NULL)
        return true;
    else
        return false;
}
```

或：

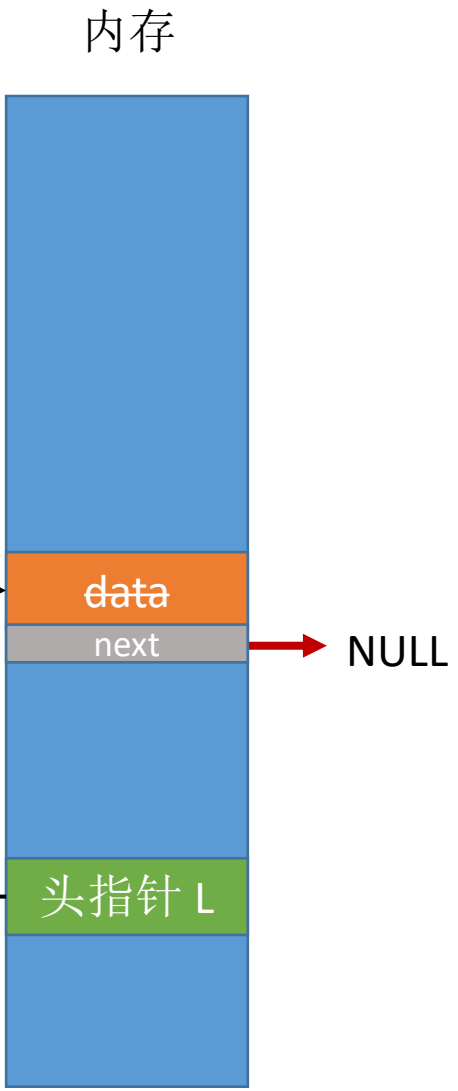
```
bool Empty(LinkList L) {
    return (L==NULL);
}
```



带头结点的单链表

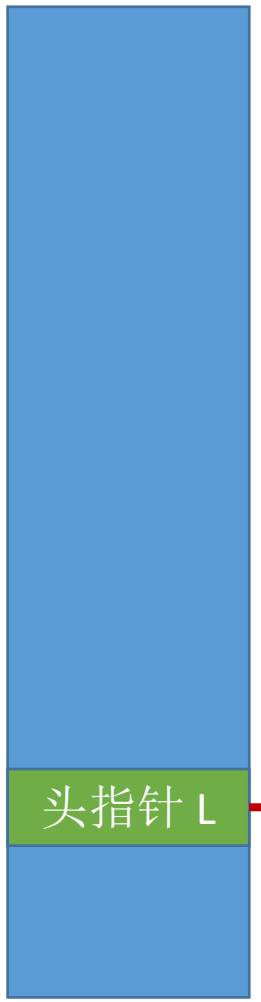
```
typedef struct LNode{              //定义单链表结点类型
    ElemType data;                 //每个节点存放一个数据元素
    struct LNode *next;            //指针指向下一个节点
}LNode, *LinkList;

//初始化一个单链表（带头结点）
bool InitList(LinkList &L) {
    L = (LNode *) malloc(sizeof(LNode));    //分配一个头结点
    if (L==NULL)                   //内存不足，分配失败
        return false;
    L->next = NULL;                //头结点之后暂时还没有节点
    return true;
}

void test(){
    LinkList L;       //声明一个指向单链表的指针
    //初始化一个空表
    InitList(L);
    //.....后续代码.....
}
```

```
//判断单链表是否为空（带头结点）
bool Empty(LinkList L) {
    if (L->next == NULL)
        return true;
    else
        return false;
}
```



不带头结点 V.S. 带头结点

不带头



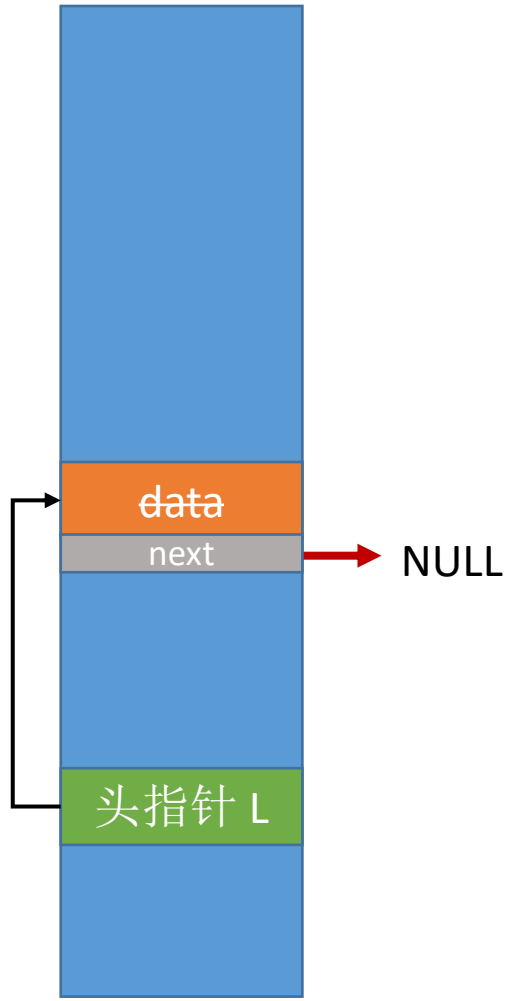
头指针 L

NULL

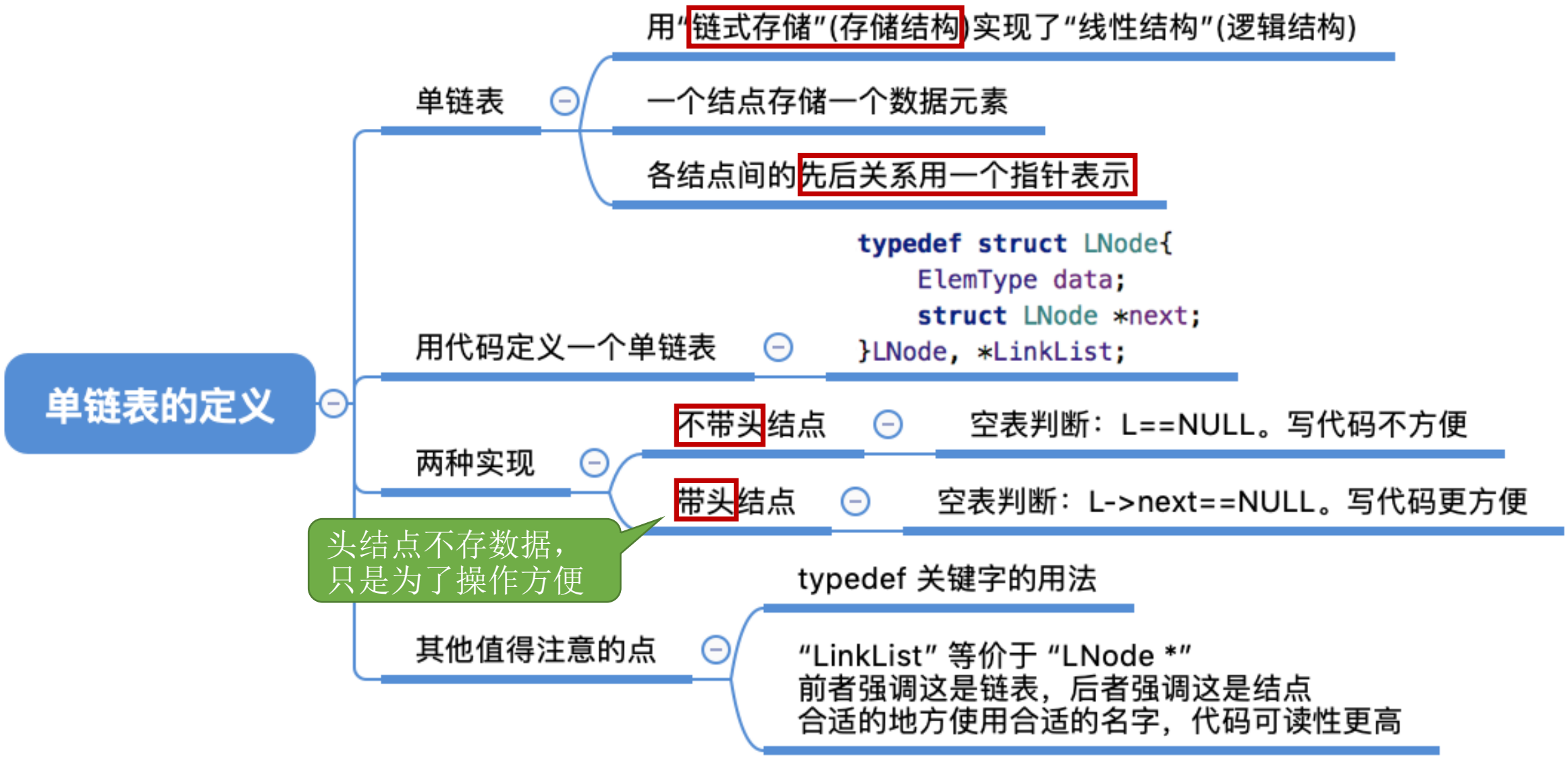
不带头结点，写代码更麻烦
对第一个数据结点和后续数据结点的处理需要用不同的代码逻辑
对空表和非空表的处理需要用不同的代码逻辑

带头结点，写代码更方便，用过都说好

带头



知识回顾与重要考点



欢迎大家对本节视频进行评价~

学员评分：2.3.1 单链表的定义

扫一扫二维码打开或分享给好友

– 腾讯文档 –

可多人实时在线编辑，权限安全可控

公众号：王道在线

b站：王道计算机教育

抖音：王道计算机考研