

本节内容

单链表 定义

知识总览

线性表

逻辑结构

基本运算/操作

存储/物理结构

顺序表（顺序存储）

链表（链式存储）

定义（如何用代码实现）

基本操作的实现

单链表

双链表

循环链表

静态链表

定义（如何用代码实现）

基本操作的实现



知识总览

单链表的定义

什么是单链表

用代码定义一个单链表

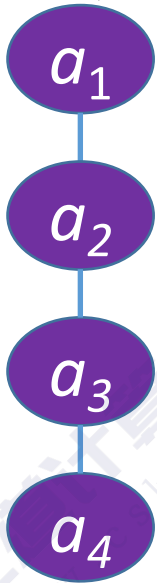
两种实现

带头结点

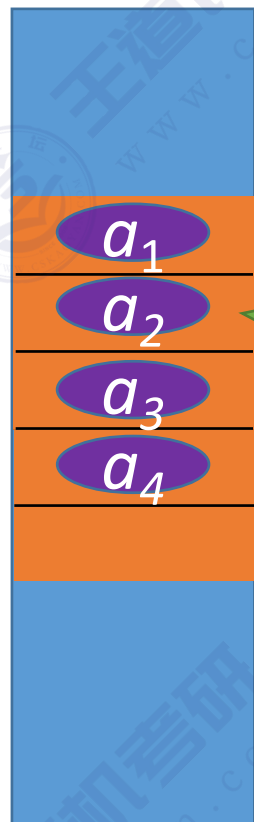
不带头结点

什么是单链表

逻辑结构：
线性表

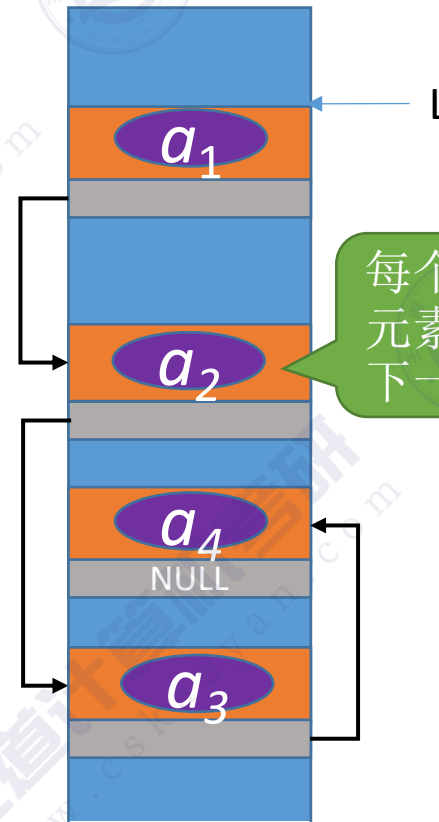


顺序表
(顺序存储)



每个结点中只
存放数据元素

单链表
(链式存储)



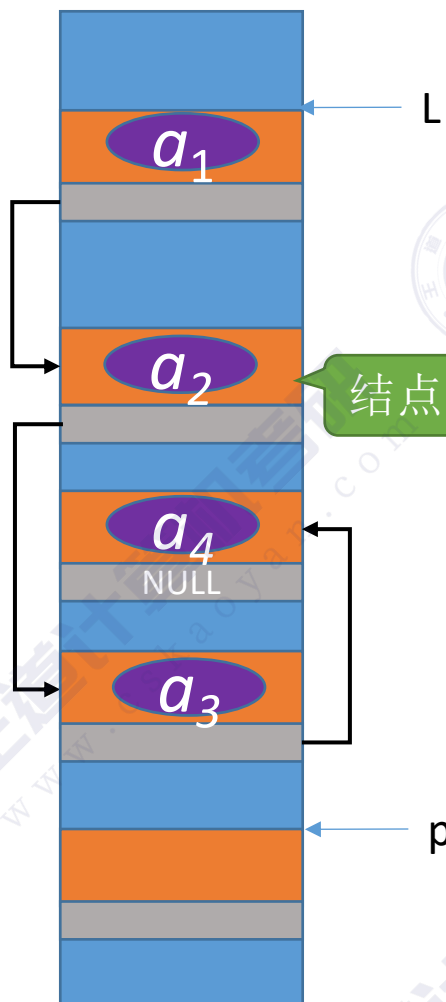
每个结点除了存放数据
元素外，还要存储指向
下一个节点的指针

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

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

单链表
(链式存储)



结点

```
struct LNode{  
    ElemType data;  
    struct LNode *next;  
};
```

数据域

指针域

//定义单链表结点类型

//每个节点存放一个数据元素

//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 int zhengshu;
```

```
typedef int *zhengshuzhizhen;
```

```
int x = 1;
```

```
int *p;
```

```
zhengshu x = 1;
```

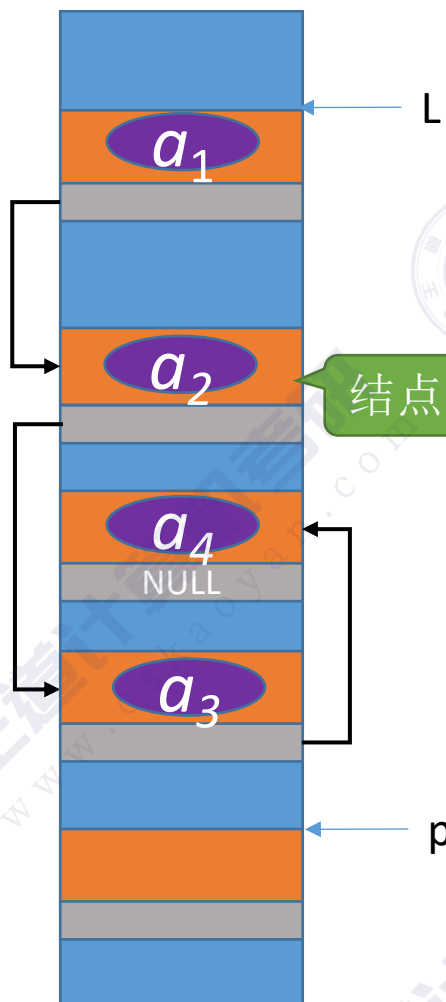
```
zhengshuzhizhen p;
```

等价



用代码定义一个单链表

单链表
(链式存储)



结点

```
struct LNode{  
    ElemType data;  
    struct LNode *next;  
};
```

//定义单链表结点类型
//每个节点存放一个数据元素
//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));
```

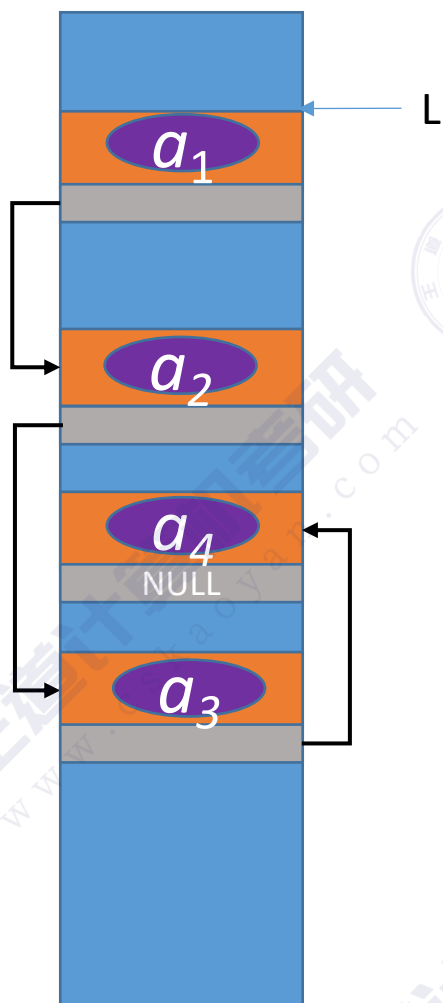
原来如此，简单!



巴啦啦能量

用代码定义一个单链表

单链表
(链式存储)



```
typedef struct LNode{  
    ElemType data;  
    struct LNode *next;  
} LNode, *LinkList;
```

//定义单链表结点类型
//每个节点存放一个数据元素
//指针指向下一个节点

LinkList
网络释义

```
struct LNode{  
    ElemType data;  
    struct LNode *next;  
};
```

//定义单链表结点类型
//每个节点存放一个数据元素
//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

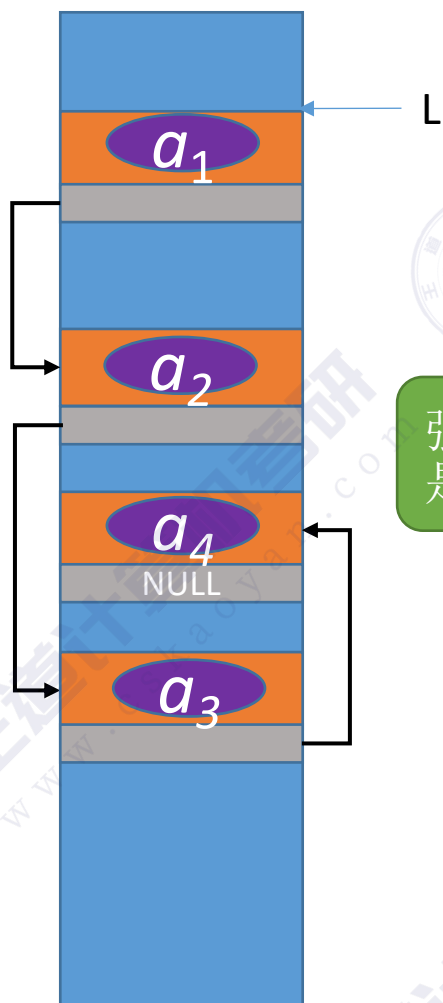
```
LNode * L; //声明一个指向单链表第一个结点的指针
```

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

单链表
(链式存储)



```
typedef struct LNode{  
    ElemType data;  
    struct LNode *next;  
}LNode, *LinkList;
```

//定义单链表结点类型
//每个节点存放一个数据元素
//指针指向下一个节点

```
LNode * GetElem(LinkList L, int i){
```

强调返回的
是一个结点

```
    int j=1;  
    LNode *p=L->next;
```

强调这是一
个单链表

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

强调这是一个单链表
强调这是一个结点

——使用 LinkList
——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：

```
LinkedList List_HeadInsert(LinkedList &L) { // 逆向建立单链表
    LNode *s; int x;
    L = (LinkedList) 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
强调这是一个结点

——使用 LinkedList
——使用 LNode *

不带头结点的单链表

内存

```
typedef struct LNode{  
    ElemType data;  
    struct LNode *next;  
}LNode, *LinkList;
```

//定义单链表结点类型
//每个节点存放一个数据元素
//指针指向下一个节点

//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);
```

```
}
```

头指针 L → NULL

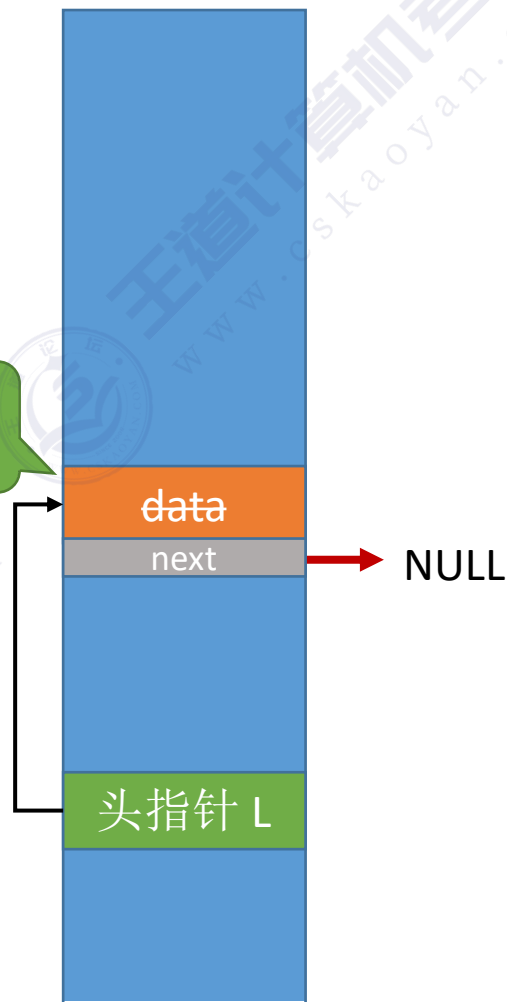
带头结点的单链表

```
typedef struct LNode{  
    ElemType data;  
    struct LNode *next;  
}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;  
}
```

内存

头结点不
存储数据



不带头结点 v.s. 带头结点

不带头

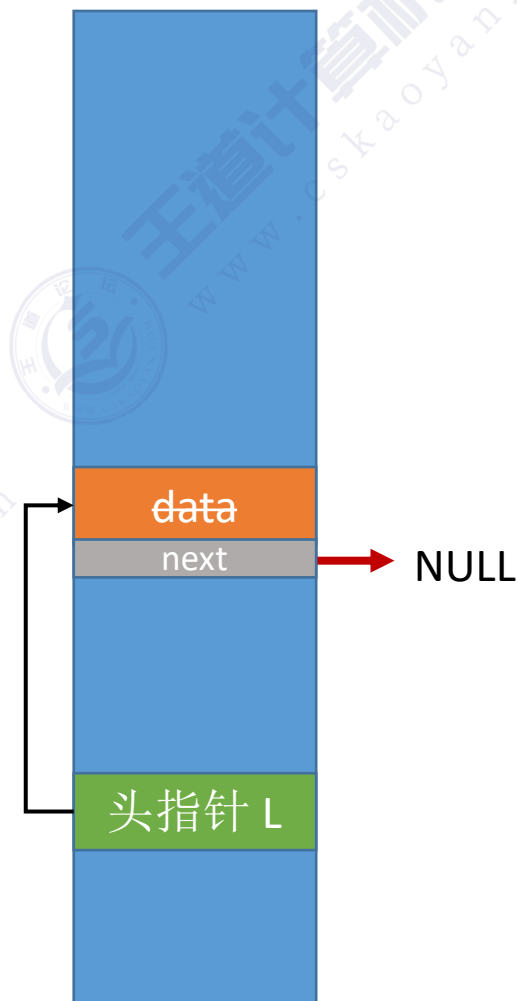


带头结点，写代码更方便，用过都说好



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



知识回顾与重要考点

单链表的定义

单链表

用“**链式存储**”(存储结构)实现了“**线性结构**”(逻辑结构)

一个结点存储一个数据元素

各结点间的**先后关系**用一个**指针表示**

用代码定义一个单链表

```
typedef struct LNode{  
    ElemType data;  
    struct LNode *next;  
}LNode, *LinkList;
```

两种实现

不带头结点

空表判断: $L == \text{NULL}$ 。写代码不方便

带头结点

空表判断: $L \rightarrow \text{next} == \text{NULL}$ 。写代码更方便

头结点不存数据,
只是为了操作方便

其他值得注意的点

typedef 关键字的用法

“LinkList”等价于“LNode *”
前者强调这是链表, 后者强调这是结点
合适的地方使用合适的名字, 代码可读性更高