C程序设计语言精髓

单向链表

哈尔滨工业大学
赵玲玲 *zhaoll@hit.edu.cn*

单向链表---定义

- 下面的结构是什么意思？

```
struct temp
{
        int     data;
        struct temp pt;
};
```

单向链表---定义

- 下面的结构是什么意思？

```
struct temp
{
        int     data;
        struct temp pt;
};
```

结构体声明时不能包含本结构体类型成员，系统将无法为这样的结构体类型分配内存

- CB下的错误提示：
  - field 'pt' has incomplete type
- VC下的错误提示：
  - 'pt' uses undefined struct 'temp'

单向链表---定义

- 下面的结构是什么意思？

```
struct temp
{
        int     data;
        struct temp pt;
};
```

- 下面的结构是什么意思呢？

```
struct temp
{
        int     data;
        struct temp *pt;
};
```

单向链表---定义

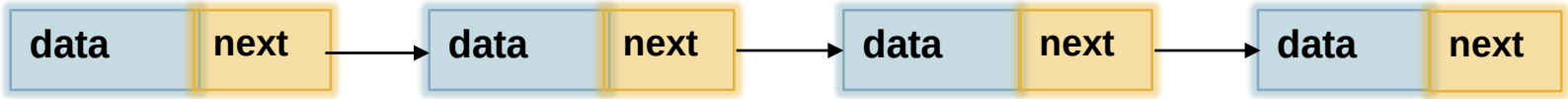
- 链表（Linked list）:线性表的链式存储结构
  - 为表示每个元素与后继元素的逻辑关系，除存储元素本身信息外，还要存储其直接后继信息

```
struct Link
{
        dataType      data;      //数据域：存储数据元素信
        struct Link *next;       //指针域：存储直接后继的节点信息
};
```

单向链表---定义

- 链表（Linked list）
  - 为表示每个元素与后继元素的逻辑关系，除存储元素本身信息外，还要存储其直接后继信息

```
struct Link
{
        dataType      data;
        struct Link *next;
};
```



单向链表---定义

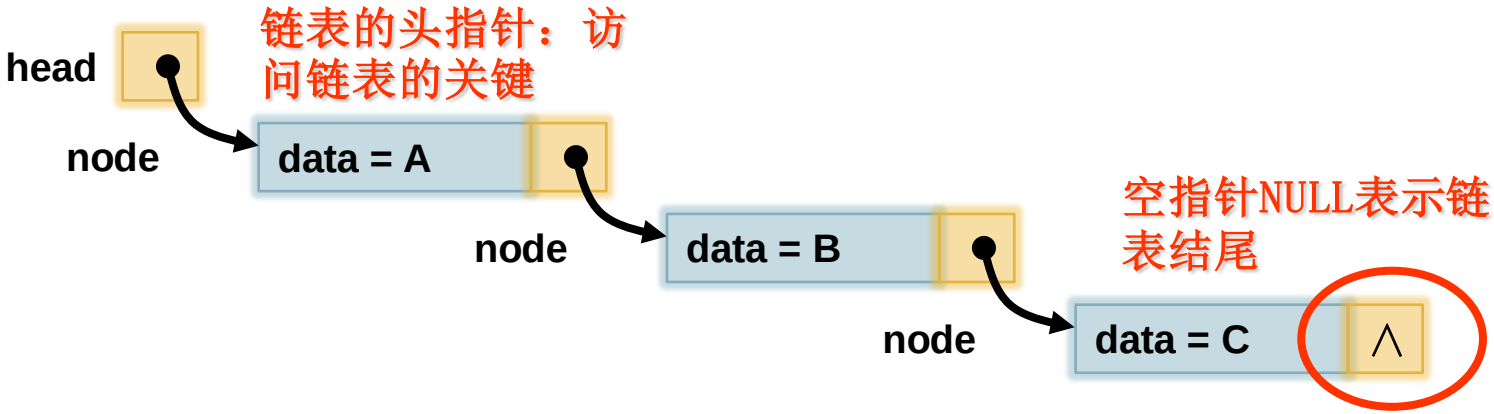
- 链表（Linked list）
  - 为表示每个元素与后继元素的逻辑关系，除存储元素本身信息外，还要存储其直接后继信息

```
struct Link
{
        dataType      data;
        struct Link *next;
};
```

n个节点链接成一个链表（因为只包含一个指针域，故又称线性链表或单向链表）

单向链表---建立

- 向链表中添加一个新节点



单向链表---建立

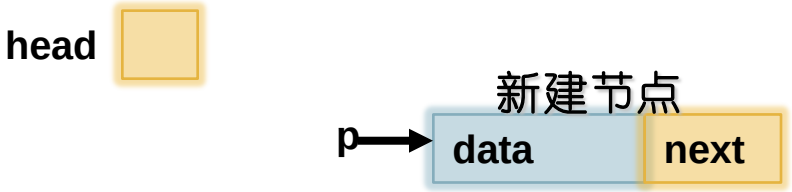
- 创建一个新节点

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

单向链表---建立

* 若原链表为空表(head == NULL)，则将新建节点p置为头节点

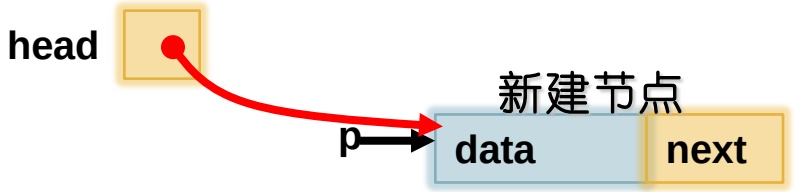
```
struct Link *p;
p = (struct Link *)malloc(sizeof(struct Link));
```



单向链表---建立

- 若原链表为空表(head == NULL)，则将新建节点p置为头节点

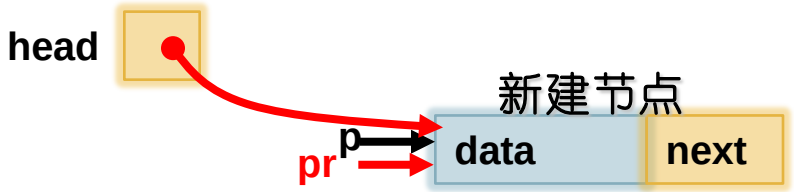
```
struct Link *p;
p = (struct Link *)malloc(sizeof(struct Link));

head = p;
```



单向链表---建立

* 若原链表为空表(head == NULL)，则将新建节点p置为头节点

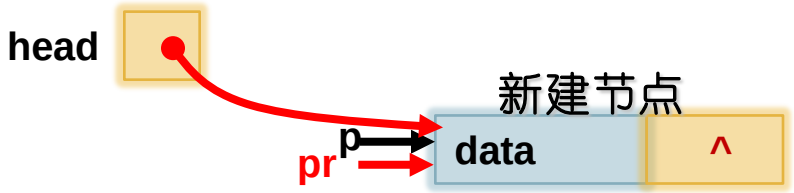
```
struct Link *p;
p = (struct Link *)malloc(sizeof(struct Link));

head = p;
pr = p;
```



单向链表---建立

* 若原链表为空表(head == NULL)，则将新建节点p置为头节点

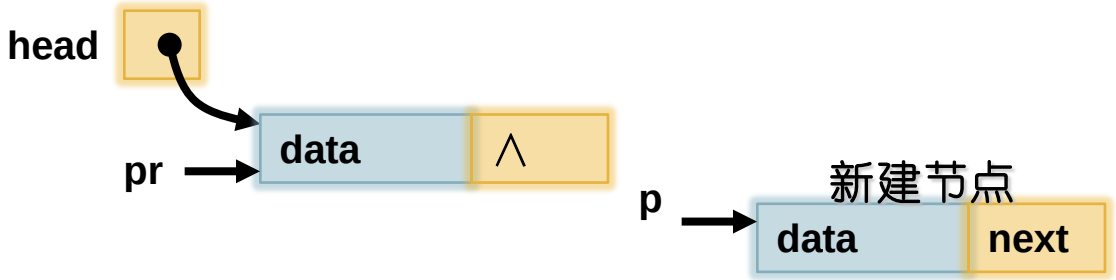
```
struct Link *p;
p = (struct Link *)malloc(sizeof(struct Link));

head = p;
pr = p;
pr->next = NULL;
```



单向链表---建立

* 若原链表为非空，则将新建节点p添加到表尾

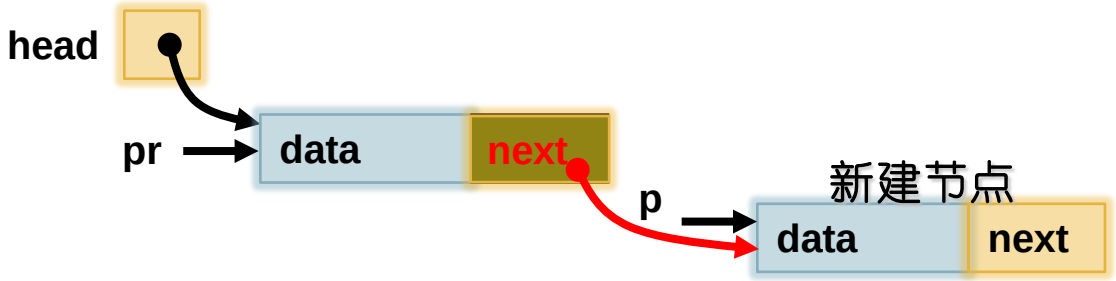
```
struct Link *p;
p = (struct Link *)malloc(sizeof(struct Link));
```



单向链表---建立

* 若原链表为非空，则将新建节点p添加到表尾

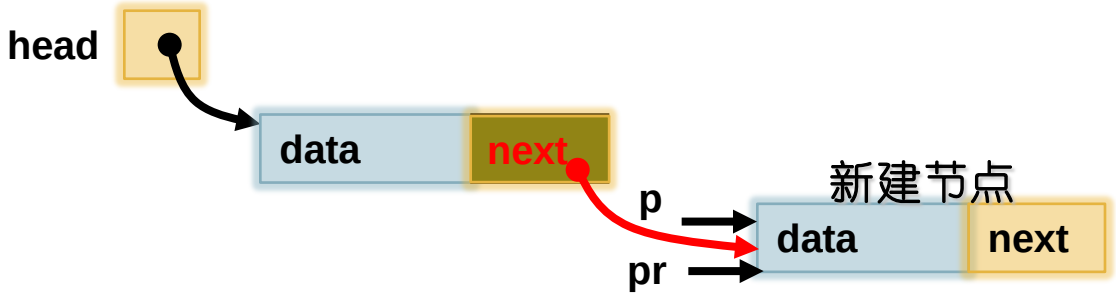
```
struct Link *p;
p = (struct Link *)malloc(sizeof(struct Link));

pr->next = p;
```



单向链表---建立

* 若原链表为非空，则将新建节点p添加到表尾

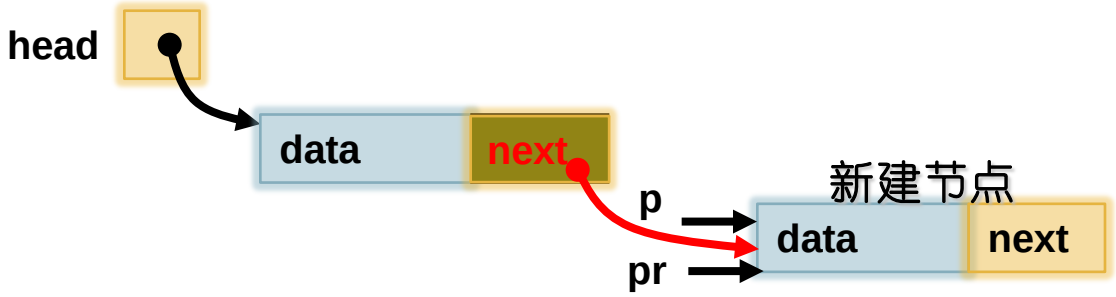
```
struct Link *p;
p = (struct Link *)malloc(sizeof(Link));

pr->next = p;
pr = p;
```



单向链表---建立

* 若原链表为非空，则将新建节点p添加到表尾

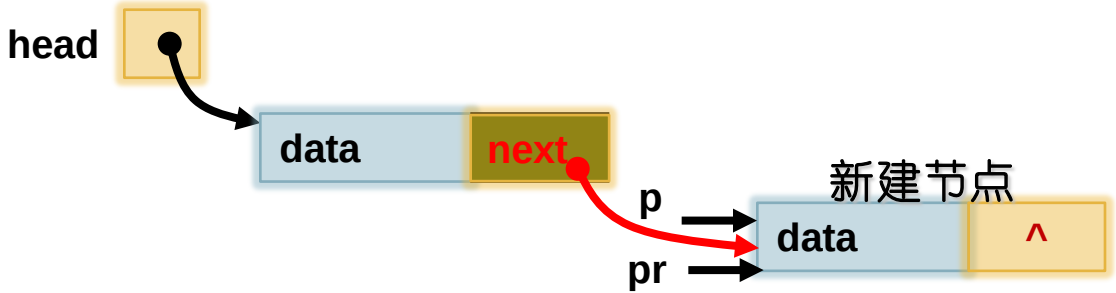
```
struct Link *p;
p = (struct Link *)malloc(sizeof(Link));

pr->next = p;
pr = p;
```



单向链表---建立

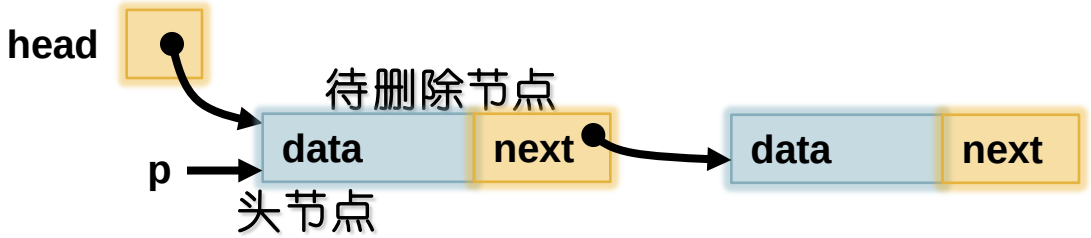
* 若原链表为非空，则将新建节点p添加到表尾

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

pr->next = p;
pr = p;
pr->next = NULL;
```



单向链表---删除

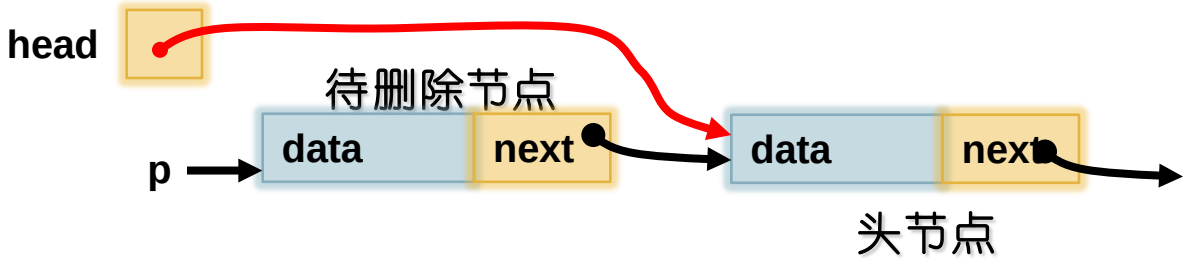
- 若原链表为空表，则退出程序
- 若待删除节点p是头节点，则将head指向当前节点的下一个节点即可删除当前节点



单向链表---删除

* 若原链表为空表，则退出程序
* 若待删除节点p是头节点，则将head指向当前节点的下一个节点即可删除当前节点

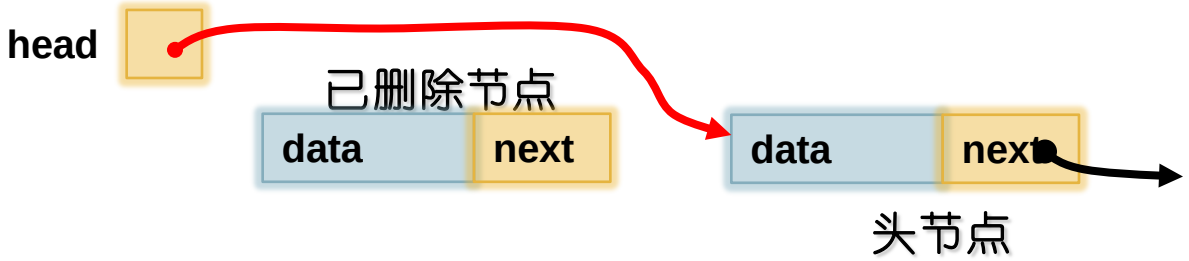
```
(1) head = p->next;
```



单向链表---删除

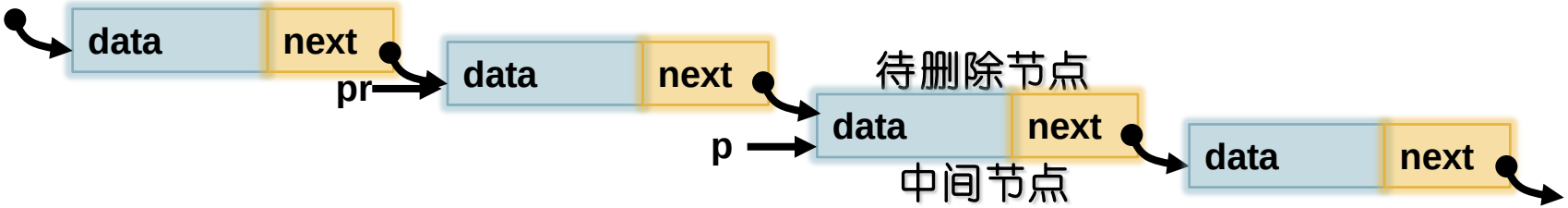
* 若原链表为空表，则退出程序
* 若待删除节点p是头节点，则将head指向当前节点的下一个节点即可删除当前节点

```
(1) head = p->next;
(2) free(p);
```



单向链表---删除

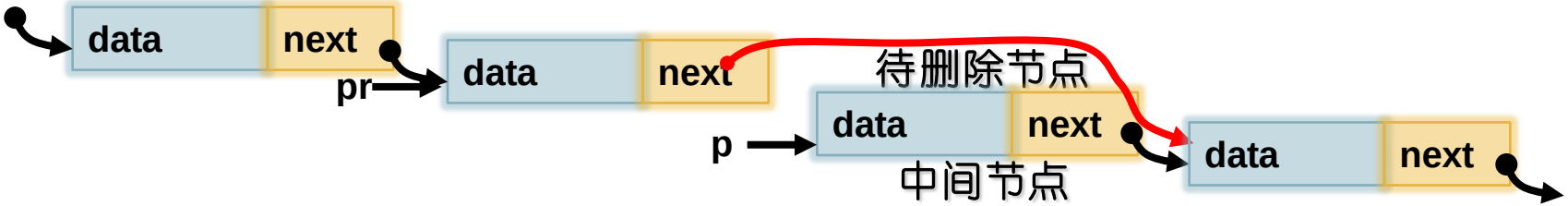
* 若待删除节点不是头节点，则将前一节点的指针域指向当前节点的下一节点即可删除当前节点



单向链表---删除

* 若待删除节点不是头节点，则将前一节点的指针域指向当前节点的下一节点即可删除当前节点

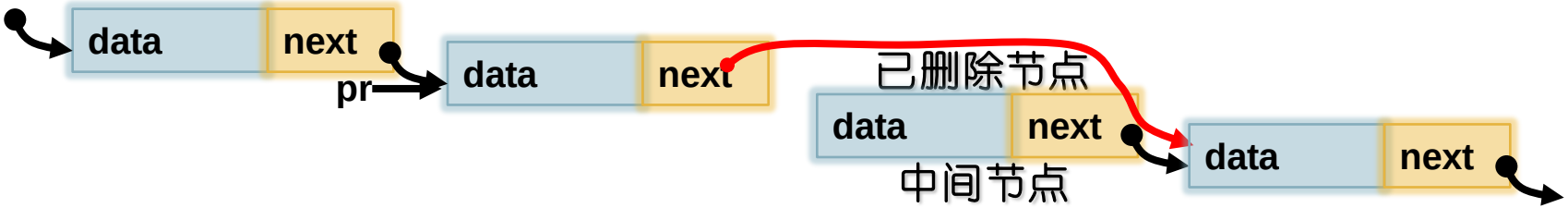
```
(1)pr->next = p->next;
```



单向链表---删除

- 若待删除节点不是头节点，则将前一节点的指针域指向当前节点的下一节点即可删除当前节点

```
(1)pr->next = p->next;
(2)free(p);
```

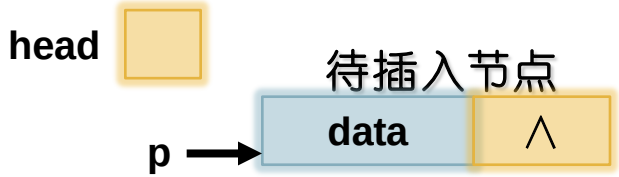


- 若已搜索到表尾（p->next == NULL）仍未找到待删除节点，则显示“未找到”

单向链表---插入

* 若原链表为空表，则将新节点p作为头节点，让head指向新节点p

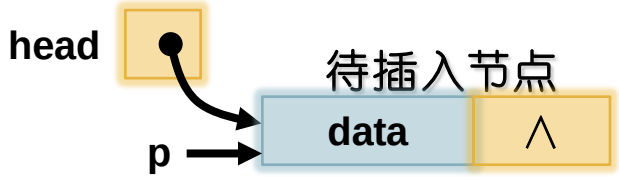
```
p = (struct link *)malloc(sizeof(struct link));
p->next = NULL;
p->data = nodeData;
```



单向链表---插入

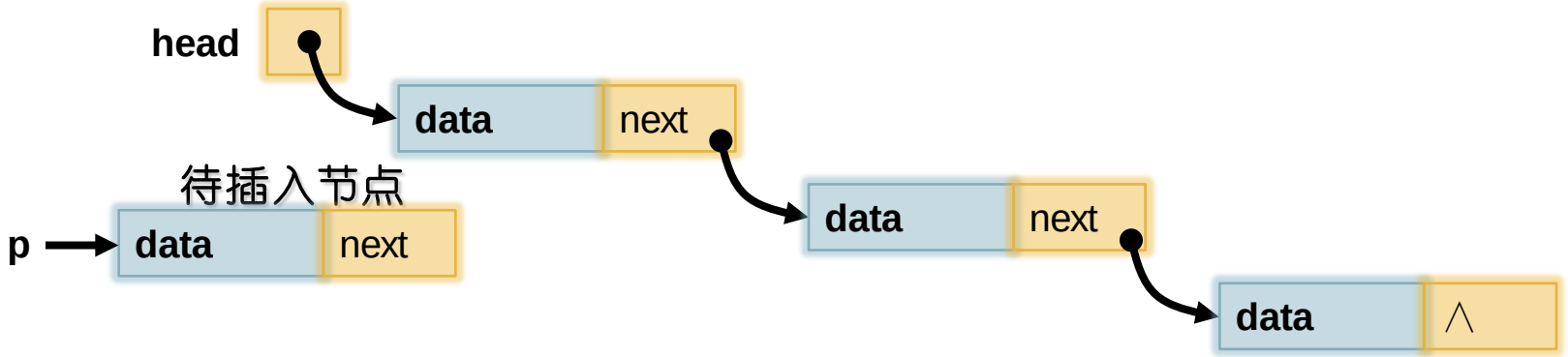
* 若原链表为空表，则将新节点p作为头节点，让head指向新节点p

```
p = (struct link *)malloc(sizeof(struct link));
p->next = NULL;
p->data = nodeData;

head = p;
```



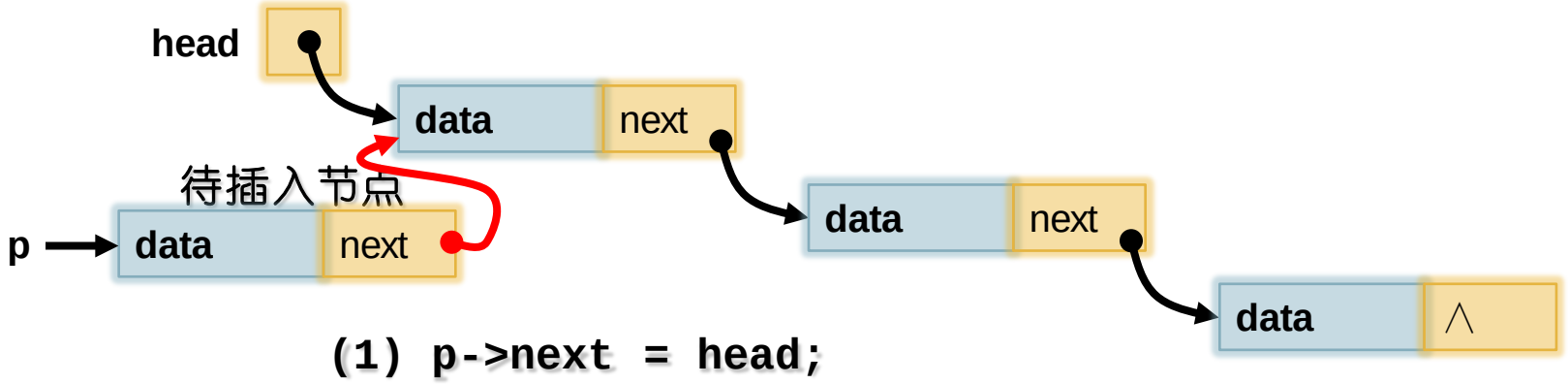
单向链表---插入

- 若原链表为非空，则按节点值（假设已按升序排序）的大小确定插入新节点的位置
- 若在头节点前插入新节点



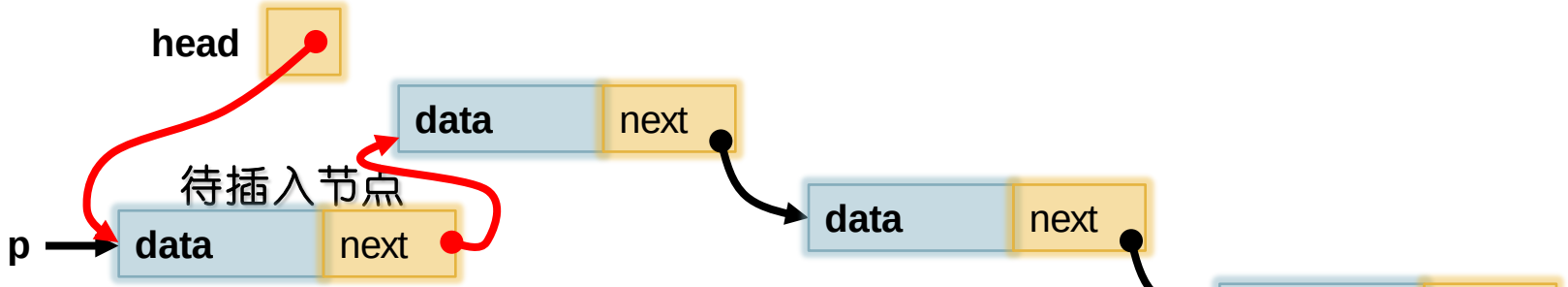
单向链表---插入

- 若原链表为非空，则按节点值（假设已按升序排序）的大小确定插入新节点的位置
- 若在头节点前插入新节点，则将新节点的指针域指向原链表的头节点，且让head指向新节点



单向链表---插入

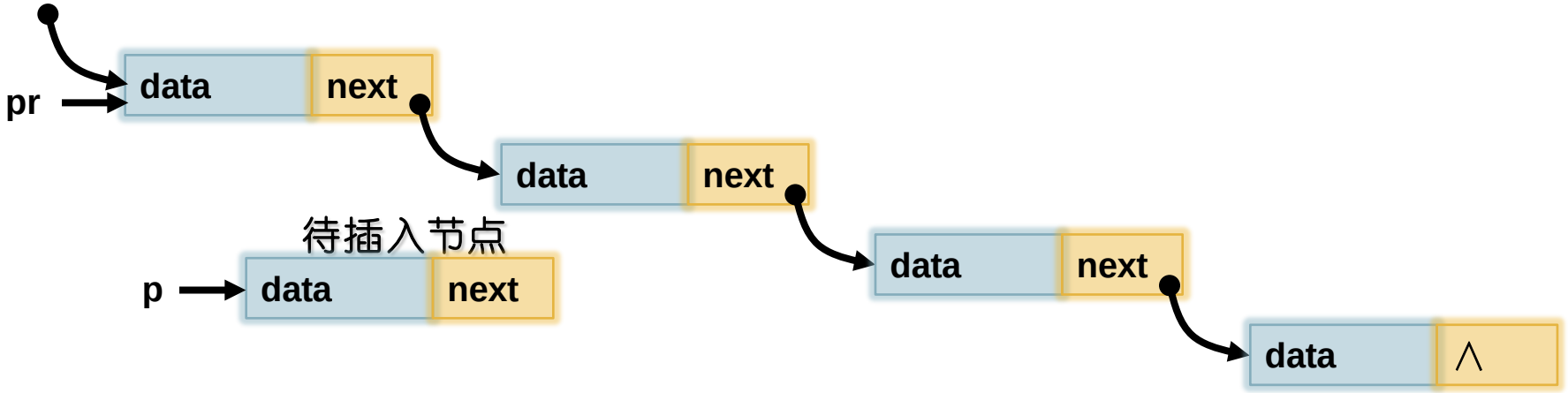
- 若原链表为非空，则按节点值（假设已按升序排序）的大小确定插入新节点的位置
- 若在头节点前插入新节点，则将新节点的指针域指向原链表的头节点，且让head指向新节点




```
(1) p->next = head;
(2) head = p;
```

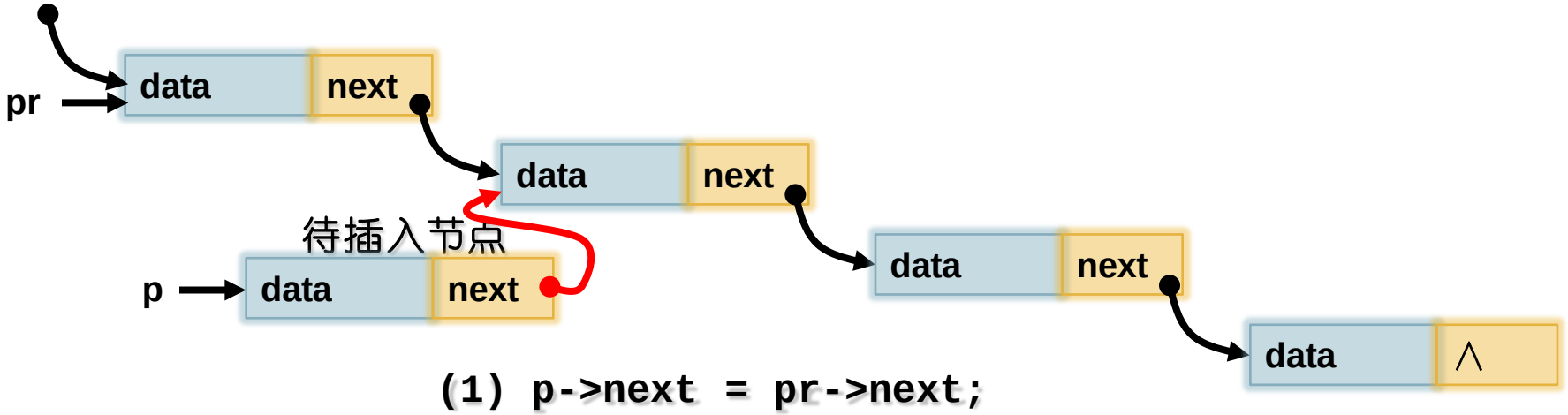
单向链表---插入

* 若在链表中间插入新节点，则将新节点的指针域指向下一节点且让前一节点的指针域指向新节点



单向链表---插入

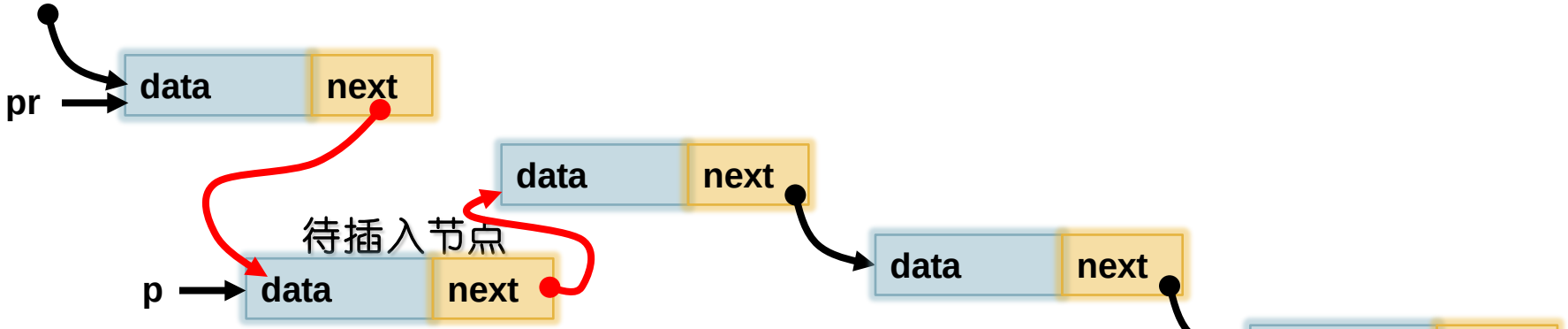
* 若在链表中间插入新节点，则将新节点的指针域指向下一节点且让前一节点的指针域指向新节点




```
(1) p->next = pr->next;
```

单向链表---插入

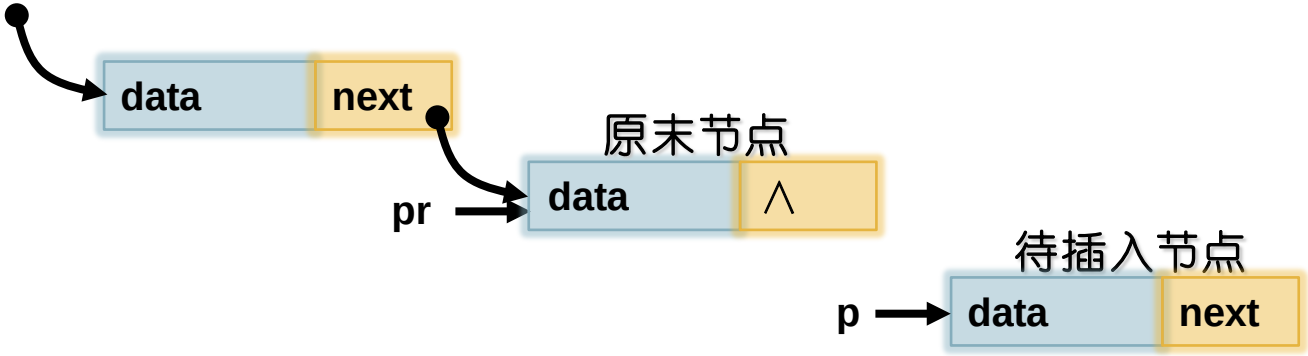
* 若在链表中间插入新节点，则将新节点的指针域指向下一节点且让前一节点的指针域指向新节点




```
(1) p->next = pr->next;
(2) pr->next = p;
```

单向链表---插入

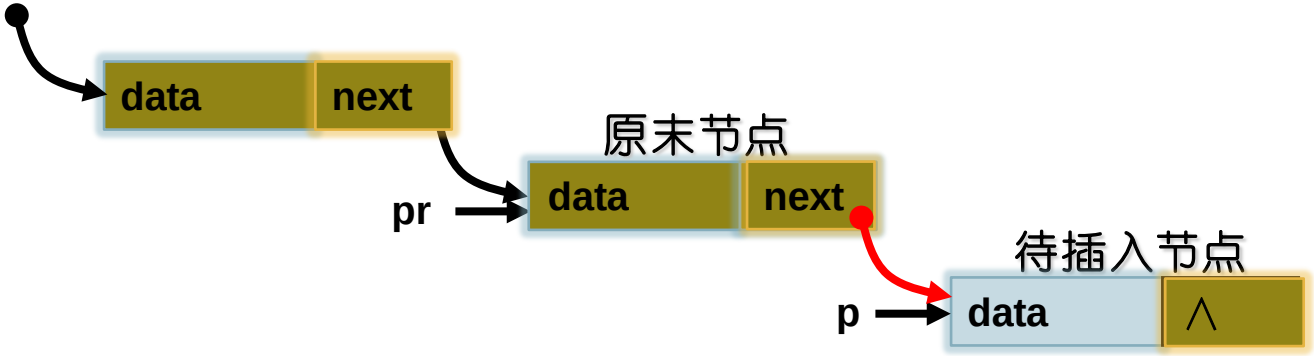
* 若在表尾插入新节点，则末节点指针域指向新节点




```
(1) pr->next = p;
```

单向链表---插入

* 若在表尾插入新节点，则末节点指针域指向新节点

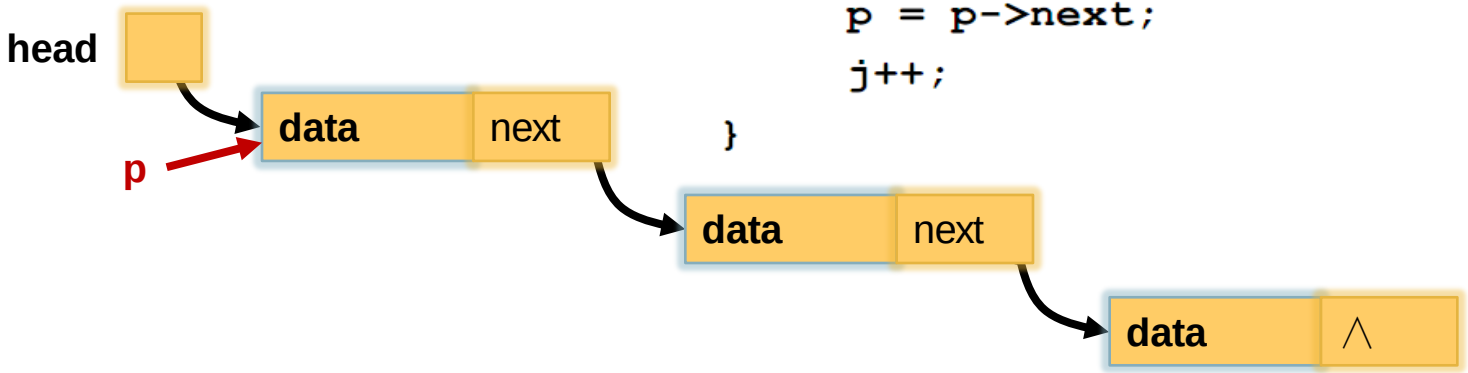



```
(1)pr->next = p;
(2)p->next = NULL;
```

单向链表---输出

* 遍历链表的所有节点

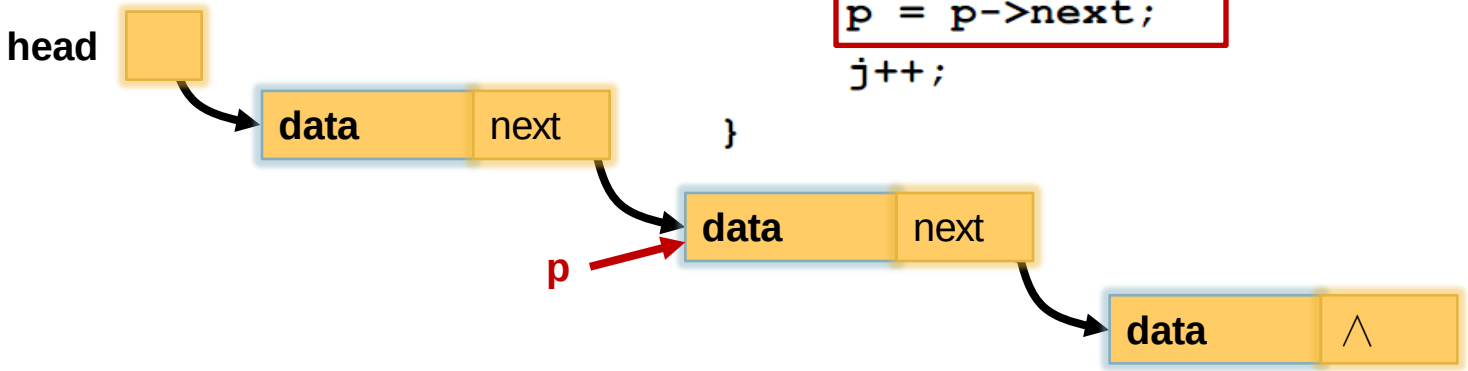
```
struct link *p = head;
int j = 1;
while (p != NULL)
{
      printf("%5d%10d\n", j, p->data);
      p = p->next;
      j++;
}
```



单向链表---输出

* 遍历链表的所有节点

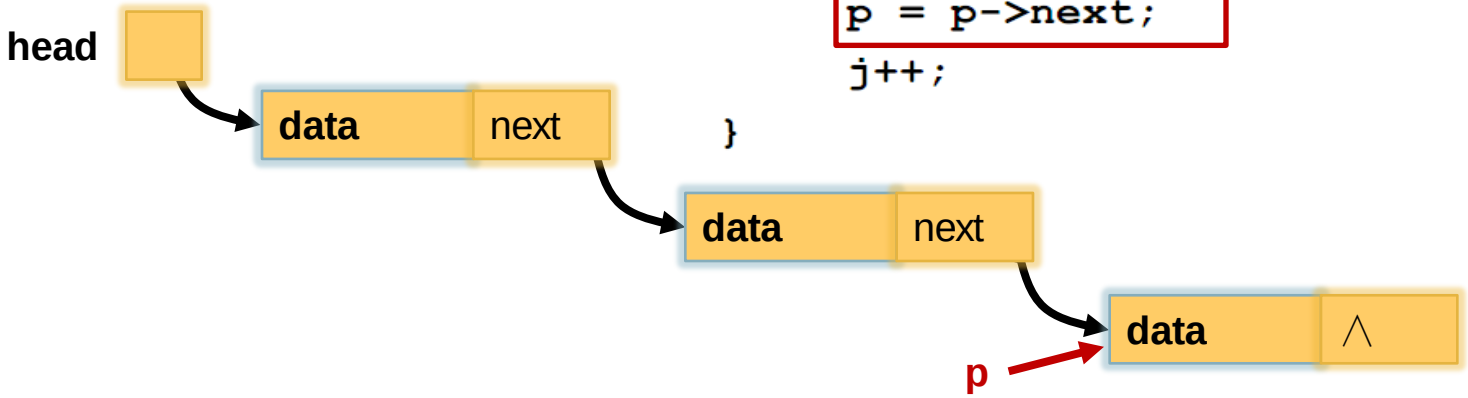
```
struct link *p = head;
int j = 1;
while (p != NULL)
{
      printf("%5d%10d\n", j, p->data);
      p = p->next;
      j++;
}
```



单向链表---输出

* 遍历链表的所有节点

```
struct link *p = head;
int j = 1;
while (p != NULL)
{
      printf("%5d%10d\n", j, p->data);
      p = p->next;
      j++;
}
```



单向链表---输出

* 遍历链表的所有节点

```
struct link *p = head;
int j = 1;
while (p != NULL)
{
      printf("%5d%10d\n", j, p->data);
      p = p->next;
      j++;
}
```