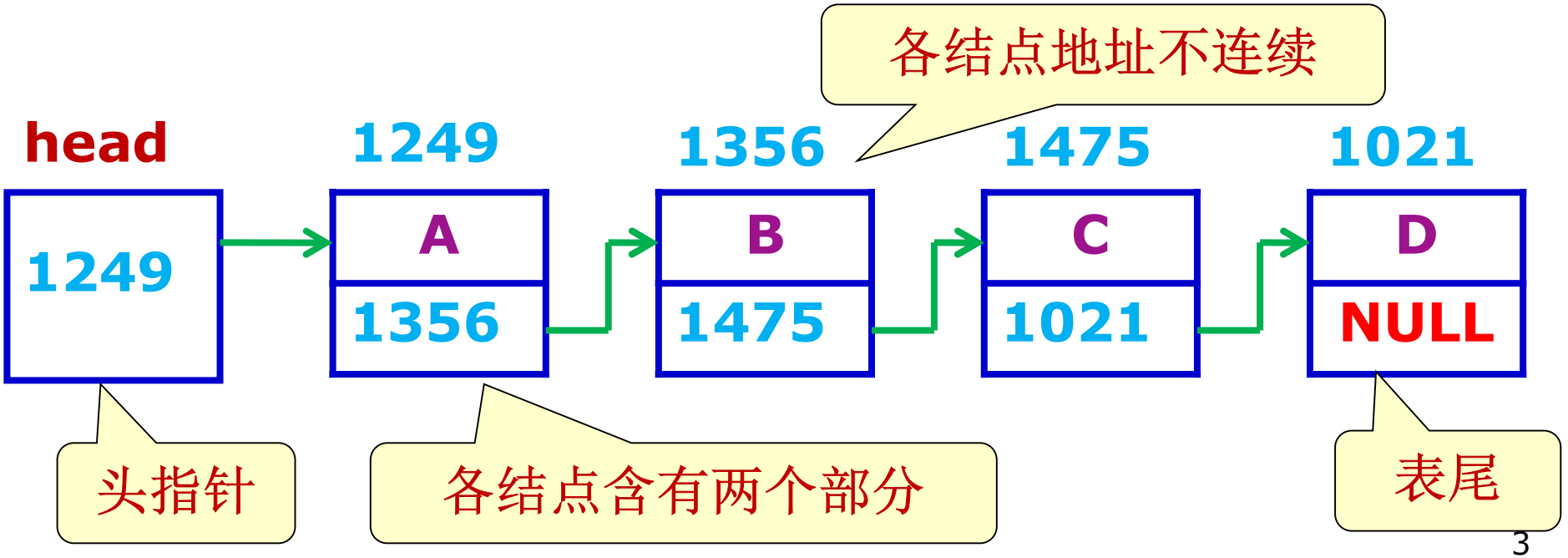
第9章 用户自己建立数据类型

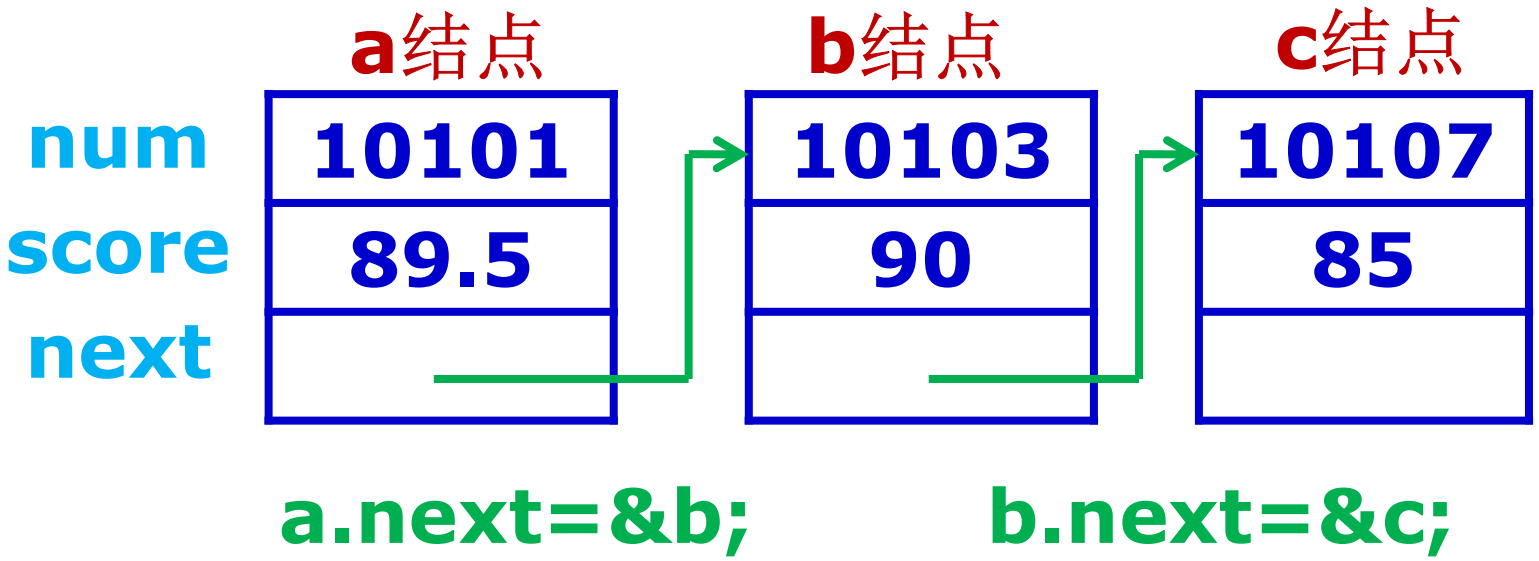
9.4 用指针处理链表

9.4.1 什么是链表

- 链表是一种常见的重要的数据结构
- 它是动态地进行存储分配的一种结构
- 链表必须利用指针变量才能实现

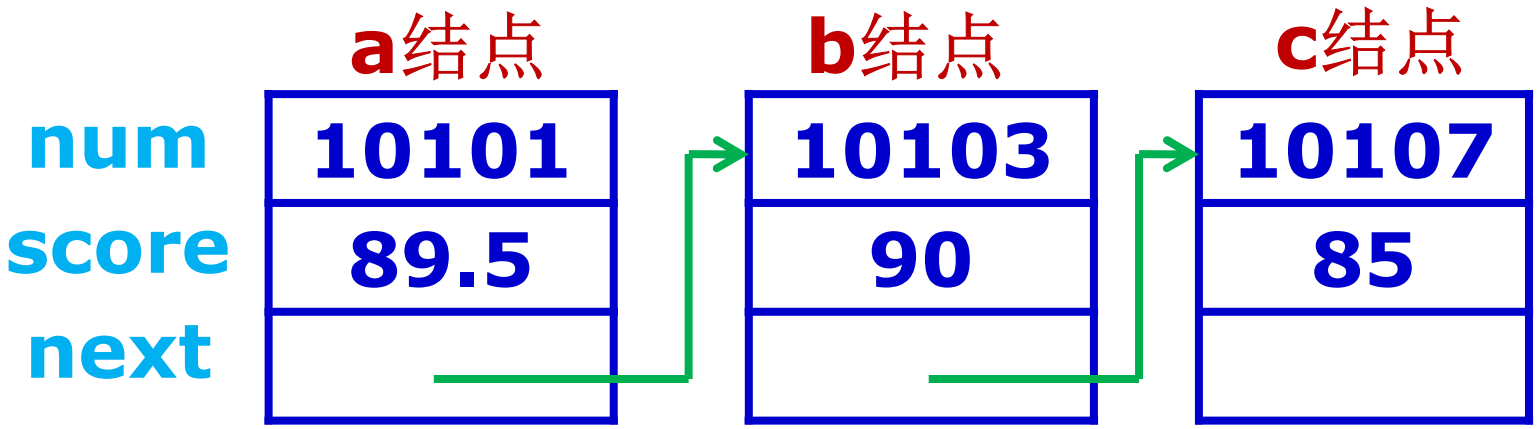


```
struct Student
{  int num;
   float score;
   struct Student *next;
}a,b,c;
```



9.4.2 建立简单的静态链表

例9.5 建立一个如图所示的简单链表，它由3个学生数据的结点组成，要求输出各结点中的数据。

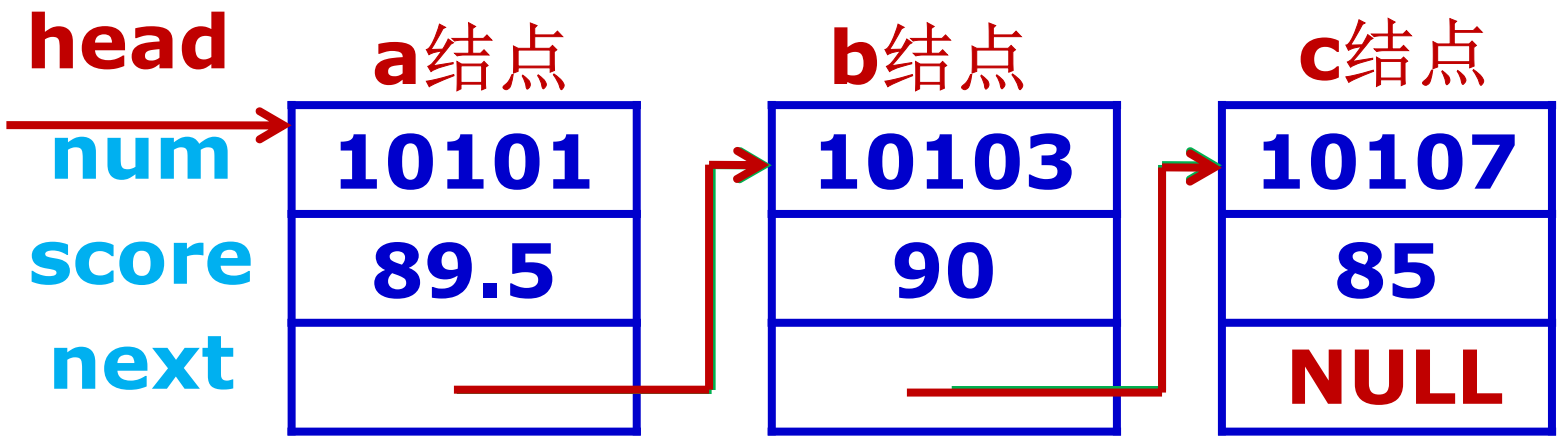


9.4.2 建立简单的静态链表

- 解题思路：

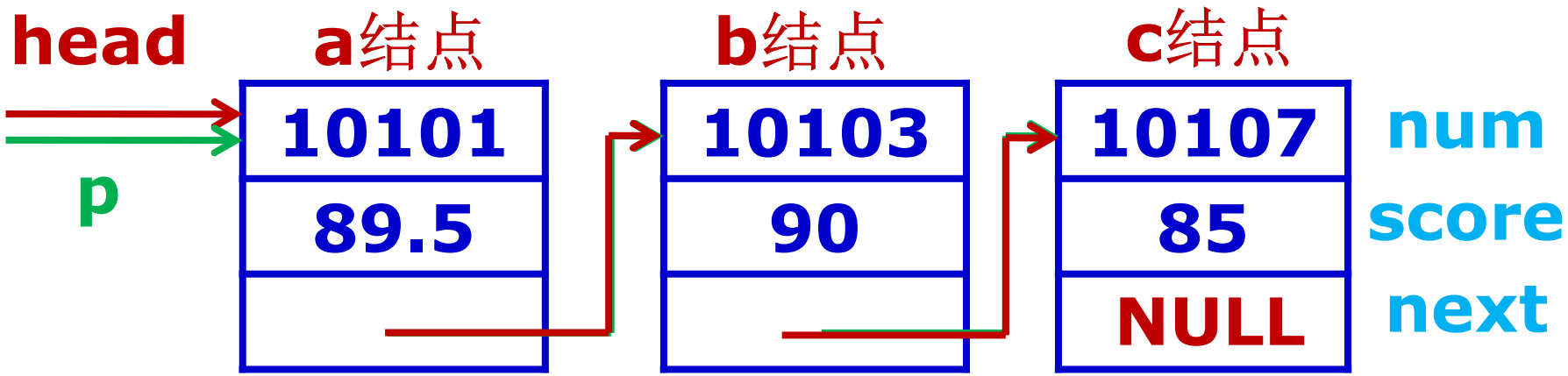
head=&a;　　a.next=&b;

b.next=&c;　c.next=NULL;



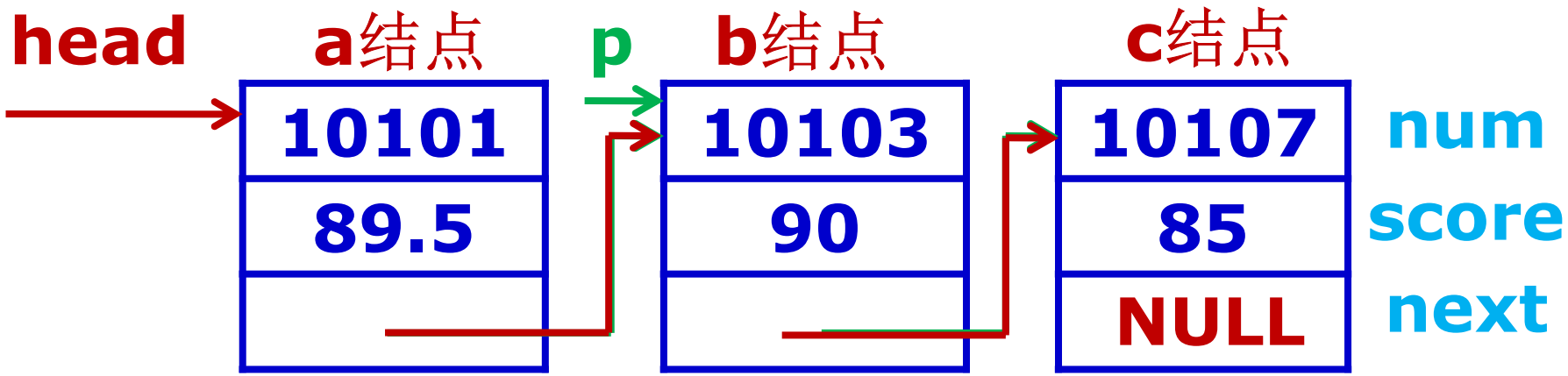
```
#include <stdio.h>
struct Student
{ int num;
  float score;
  struct Student *next;
};
```

```
int main()
{ struct Student a,b,c,*head,*p;
  a. num=10101;  a.score=89.5;
  b. num=10103;  b.score=90;
  c. num=10107;  c.score=85;
  head=&a;          a.next=&b;
  b.next=&c;        c.next=NULL;
  p=head;
  do
  {printf("%ld%5.1f\n",p->num,p->score);
    p=p->next;
   }while(p!=NULL);
  return 0;
}
```



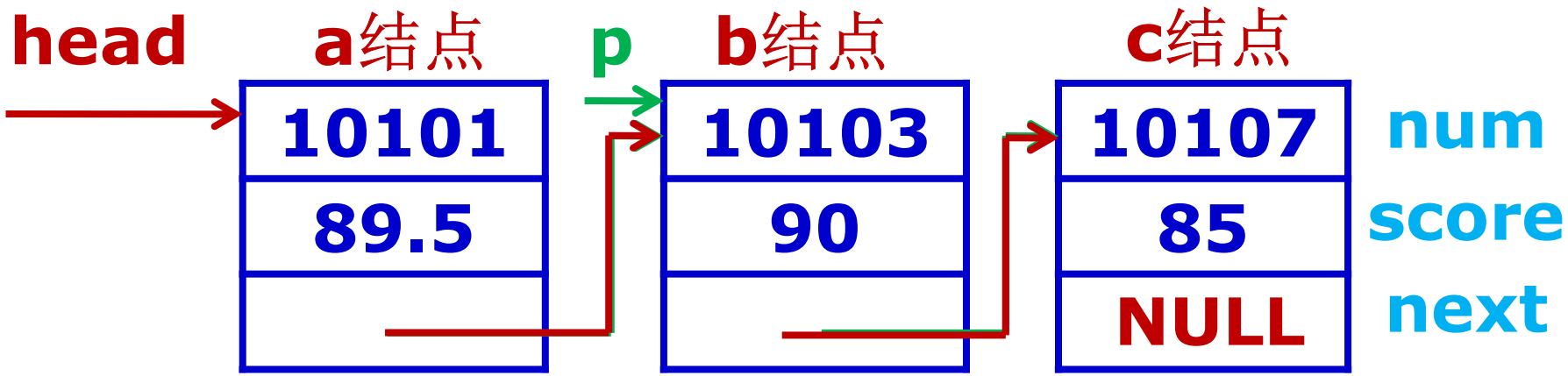

```
  p=head;
  do
  {printf("%ld%5.1f\n",p->num,p->score);
    p=p->next;  相当于p=&b;
   }while(p!=NULL);
  return 0;
}
```

```
10101  89.5
```



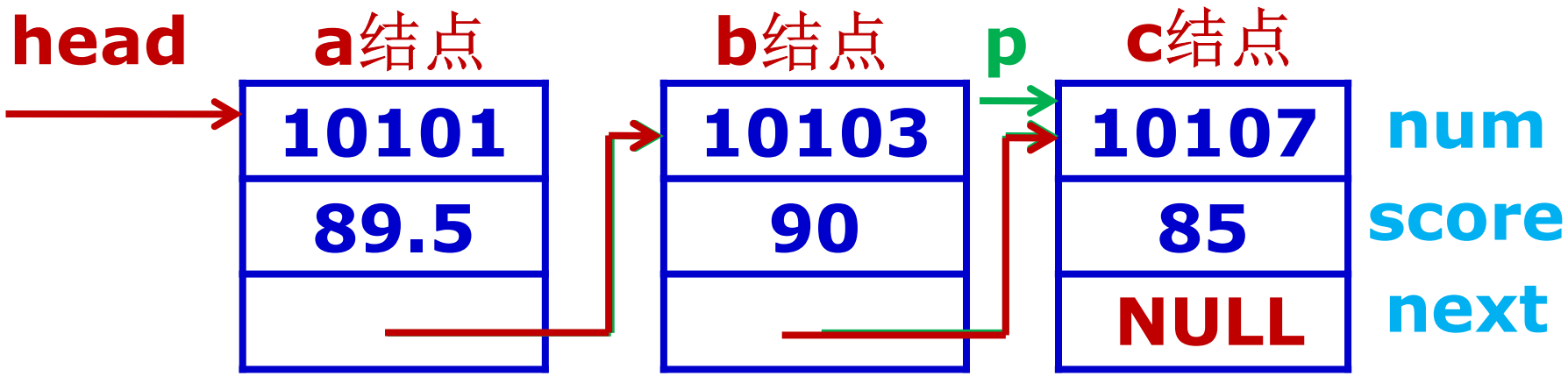

```
  p=head;
  do
  {printf("%ld%5.1f\n",p->num,p->score);
    p=p->next;  相当于p=&b;
   }while(p!=NULL);
  return 0;
}
```

```
10101  89.5
```



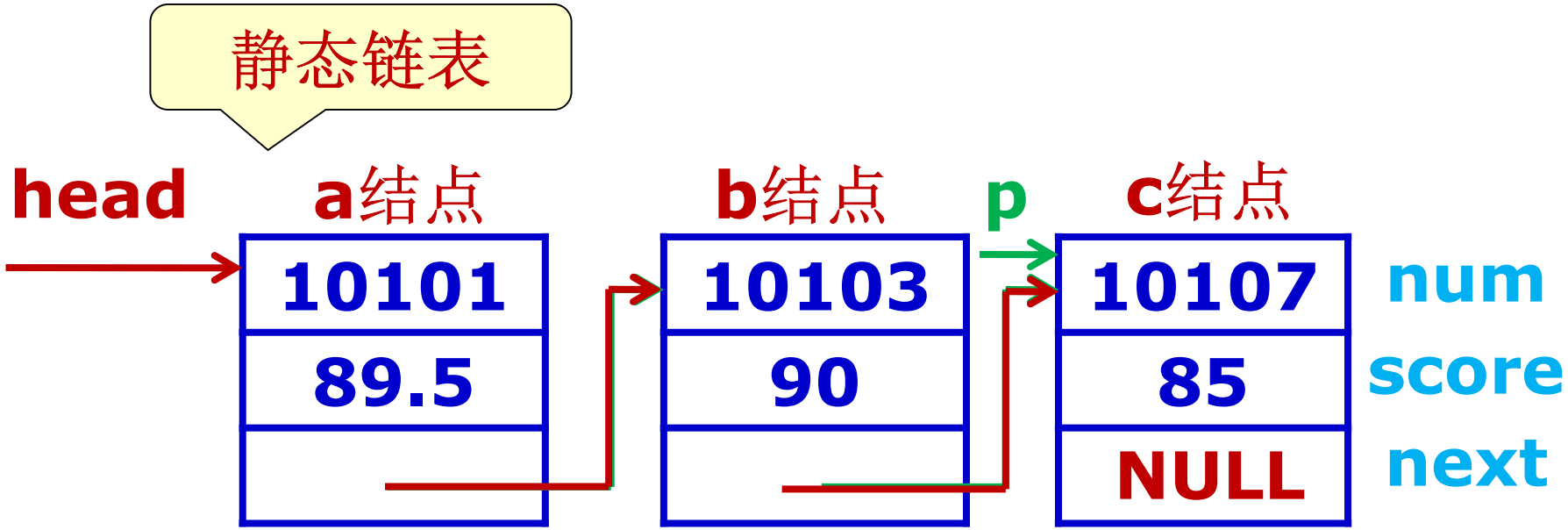

```
  p=head;
  do
  {printf("%ld%5.1f\n",p->num,p->score);
    p=p->next;  相当于p=&c;
   }while(p!=NULL);
  return 0;
}
```

```
10101   89.5
10103   90.0
```




```
  p=head;
  do
  {printf("%ld%5.1f\n",p->num,p->score);
    p=p->next;  相当于p=&c;
   }while(p!=NULL);
  return 0;
}
```

```
10101   89.5
10103   90.0
```




```
   p=head;
   do
   {printf("%ld%5.1f\n",p->num,p->score);
     p=p->next; 相当于p=NULL;
    }while(p!=NULL);
   return 0;
}
```

```
10101   89.5
10103   90.0
10107   85.0
```

9.4.3 建立动态链表

- 所谓建立动态链表是指在程序执行过程中从无到有地建立起一个链表，即一个一个地开辟结点和输入各结点数据，并建立起前后相链的关系。
- 例**9.6** 写一函数建立一个有**3**名学生数据的单向动态链表。

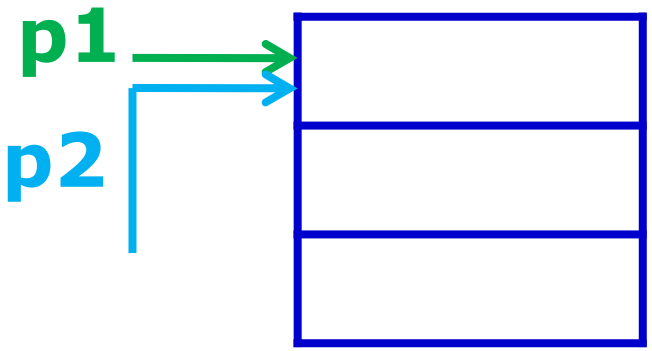
- 解题思路：
  - 定义**3**个指针变量：**head,p1**和**p2**，它们都是用来指向**struct Student**类型数据

```
struct Student *head,*p1,*p2;
```

➢解题思路：

◆用**malloc**函数开辟第一个结点，并使**p1**和**p2**指向它

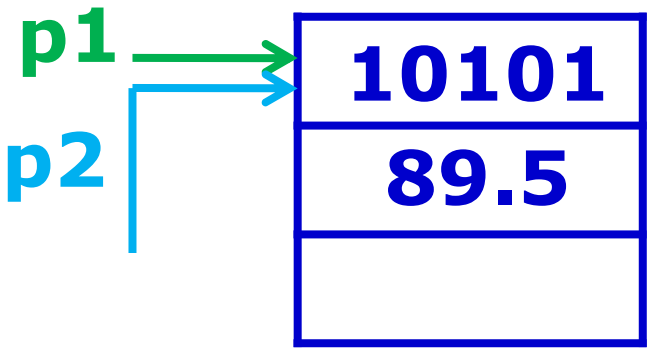
```
p1=p2=(struct Student*)malloc(LEN);
```



➢解题思路：

◆读入一个学生的数据给**p1**所指的第一个结点

```
scanf("%ld,%f",&p1->num,&p1->score);
```

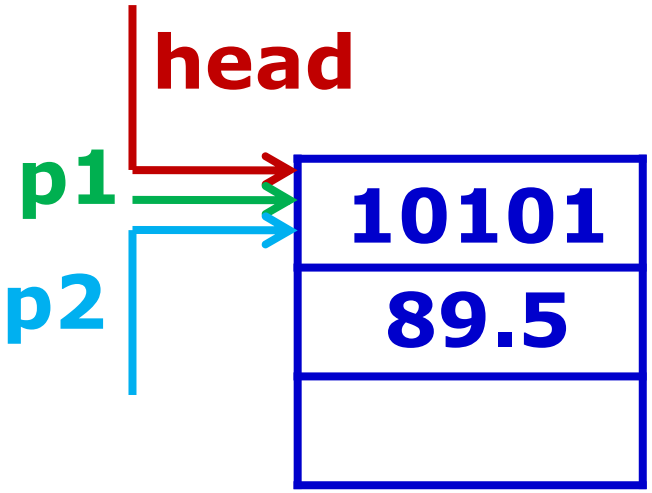


➢解题思路：

◆读入一个学生的数据给**p1**所指的第一个结点

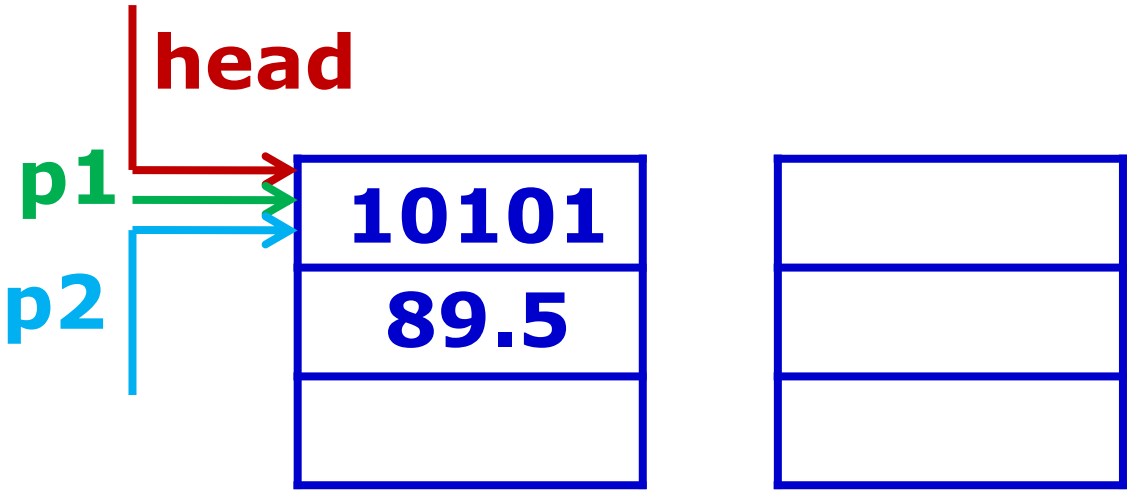
```
scanf("%ld,%f",&p1->num,&p1->score);
```

◆使**head**也指向新开辟的结点



➢解题思路：

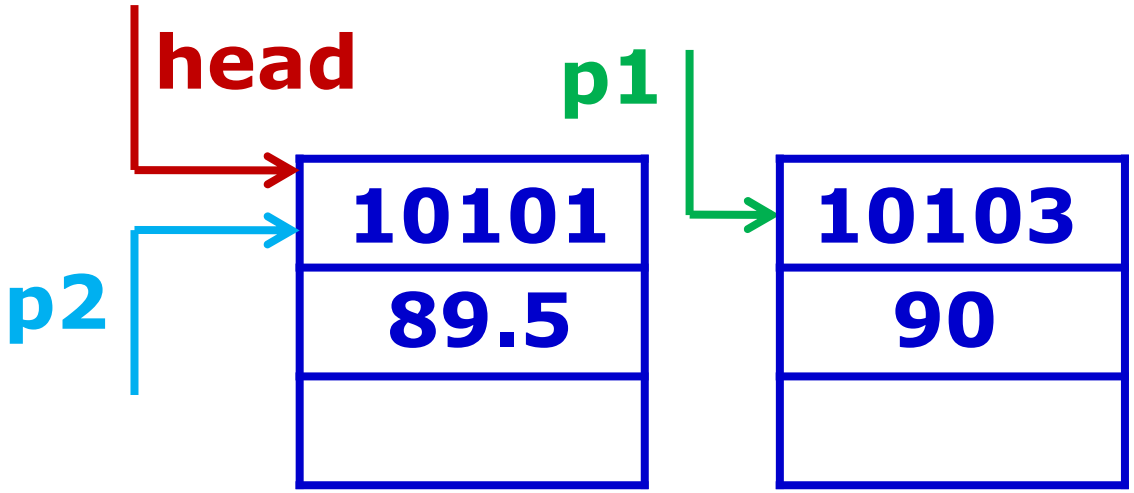
◆再开辟另一个结点并使**p1**指向它，接着输入该结点的数据



➢解题思路：

◆再开辟另一个结点并使**p1**指向它，接着输入该结点的数据

```
p1=(struct Student*)malloc(LEN);
scanf("%ld,%f",&p1->num,&p1->score);
```

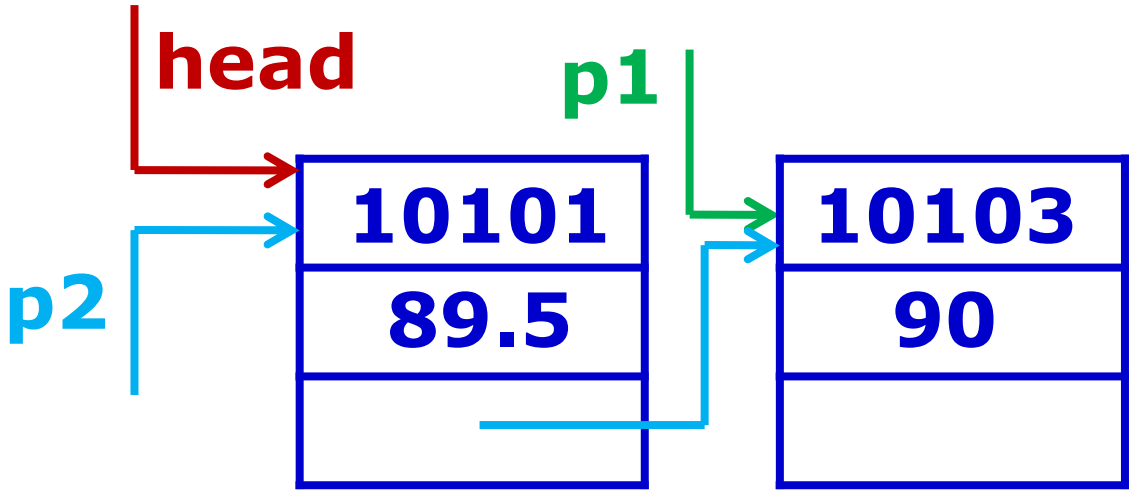


➢解题思路：

◆使第一个结点的**next**成员指向第二个结点，即连接第一个结点与第二个结点

p2->next=p1;

◆使**p2**指向刚才建立的结点

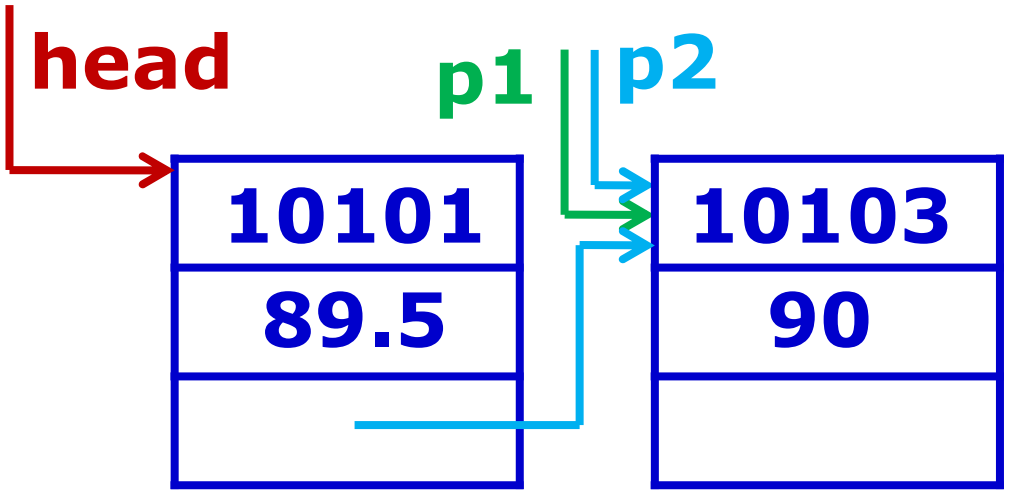


➢解题思路：

◆使第一个结点的**next**成员指向第二个结点，即连接第一个结点与第二个结点

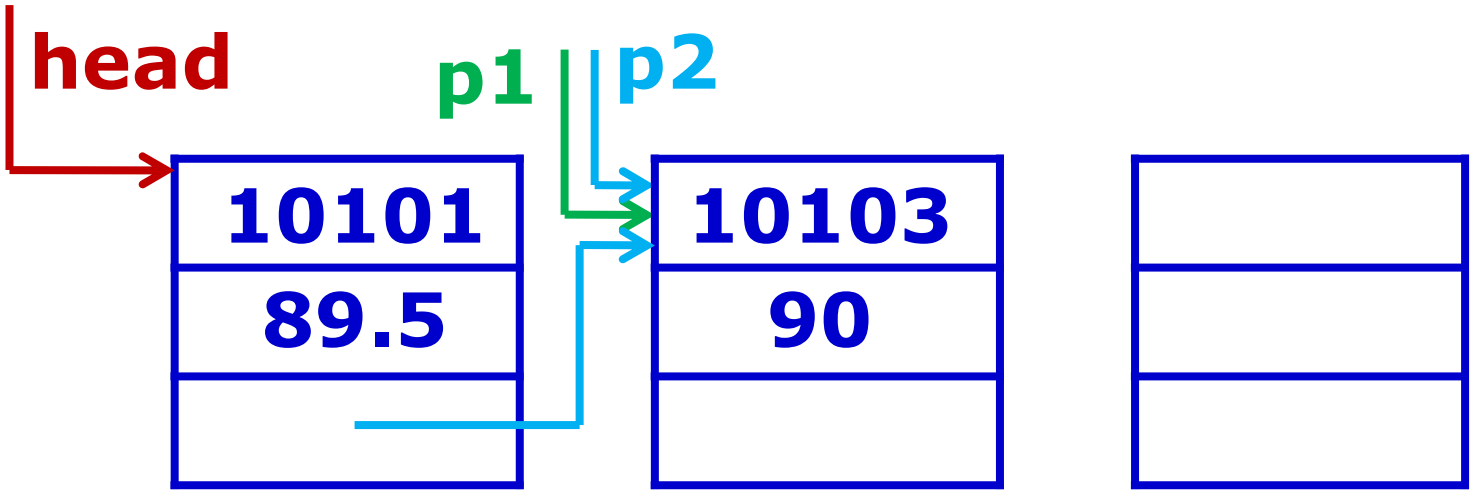
p2->next=p1;

◆使**p2**指向刚才建立的结点　　**p2=p1;**



➢解题思路：

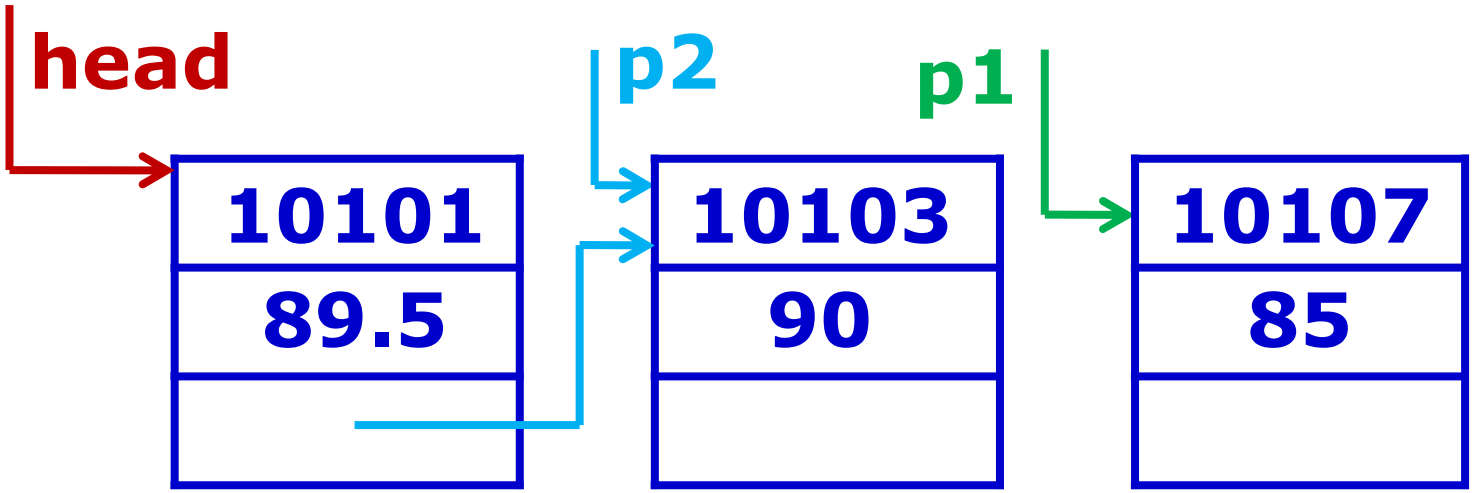
◆再开辟另一个结点并使**p1**指向它，接着输入该结点的数据



➢解题思路：

◆再开辟另一个结点并使**p1**指向它，接着输入该结点的数据

```
p1=(struct Student*)malloc(LEN);
scanf("%ld,%f",&p1->num,&p1->score);
```

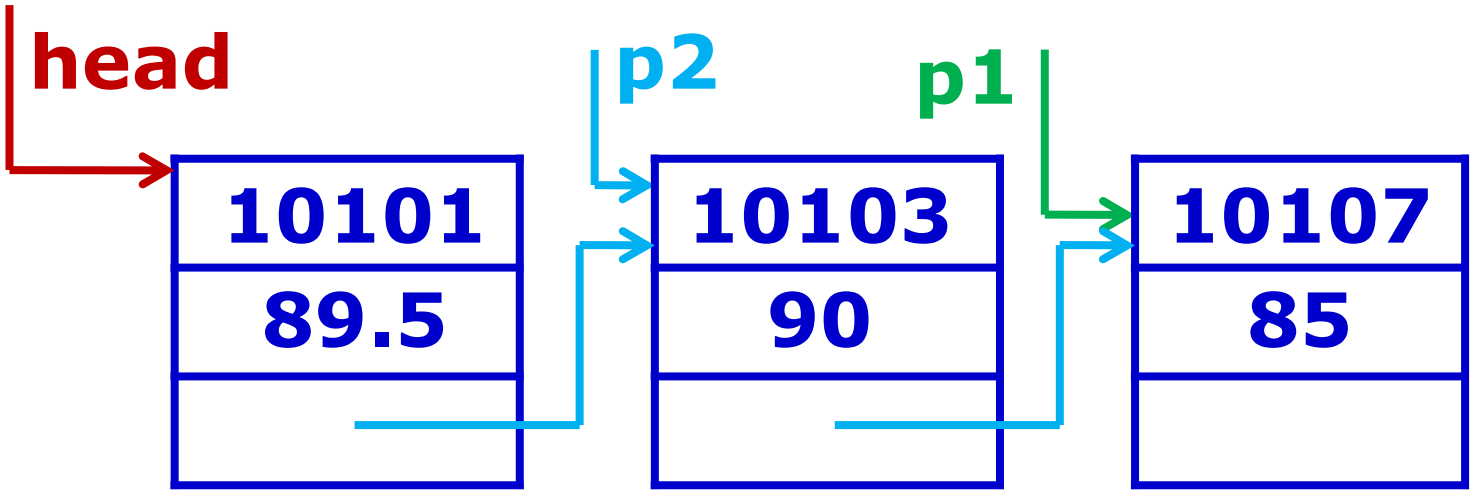


➢解题思路：

◆使第二个结点的**next**成员指向第三个结点，即连接第二个结点与第三个结点

p2->next=p1;

◆使**p2**指向刚才建立的结点

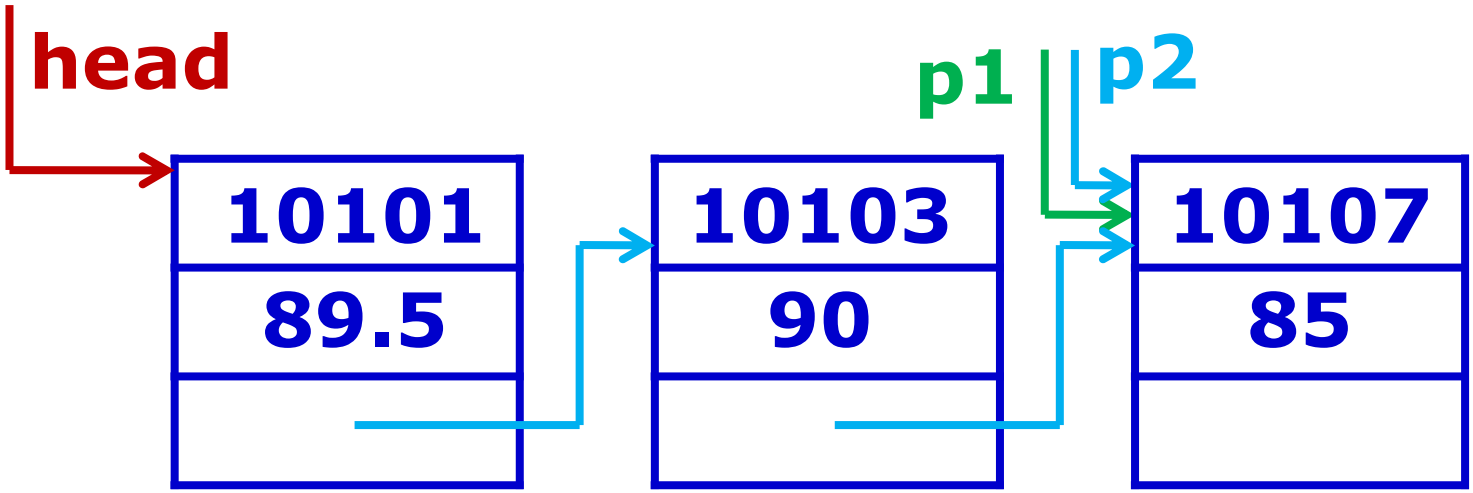


➢解题思路：

◆使第二个结点的**next**成员指向第三个结点，即连接第二个结点与第三个结点

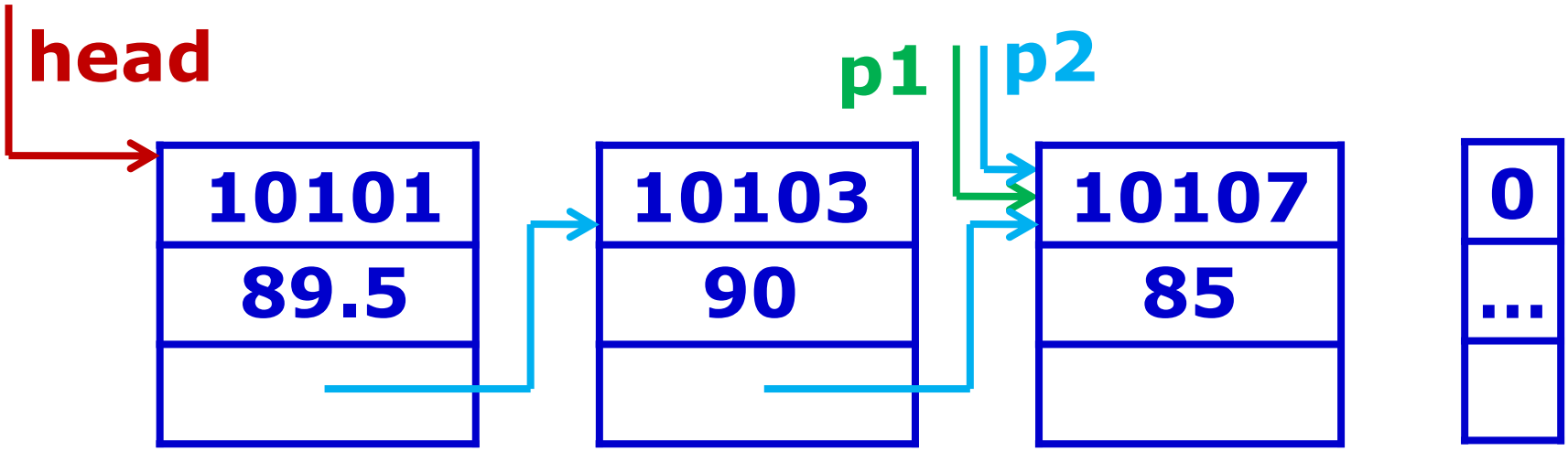
p2->next=p1;

◆使**p2**指向刚才建立的结点　　**p2=p1;**



➢解题思路：

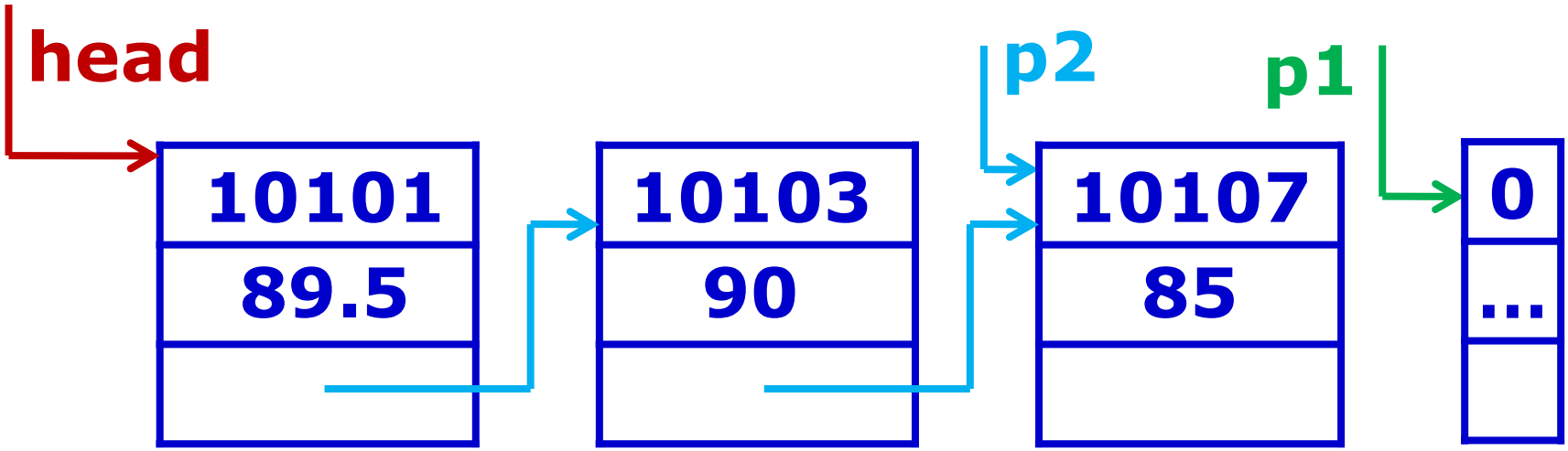
◆再开辟另一个结点并使**p1**指向它，接着输入该结点的数据



➢解题思路：

◆再开辟另一个结点并使**p1**指向它，接着输入该结点的数据

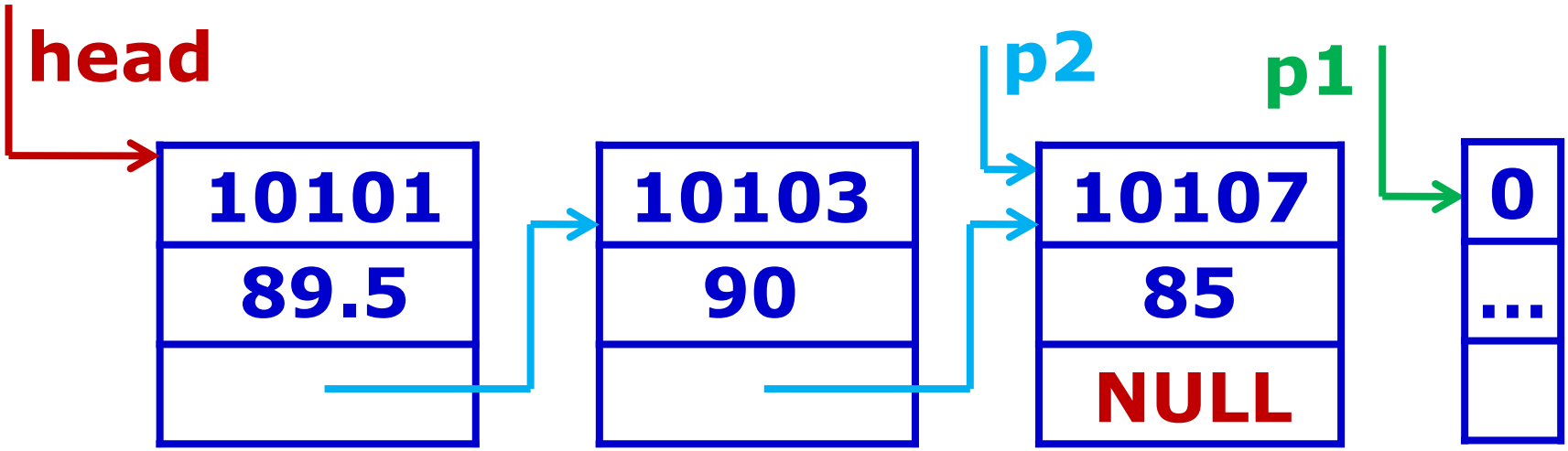
```
p1=(struct Student*)malloc(LEN);
scanf("%ld,%f",&p1->num,&p1->score);
```



➢解题思路：

◆输入的学号为**0**，表示建立链表的过程完成，该结点不应连接到链表中

p2->next=NULL;



struct Student类型数据的长度

```
#include <stdi
#include <st  .h>
#define LEN sizeof(struct Student)
struct Student
{ long num;
  float score;
  struct Student *next;
};
int n;
```

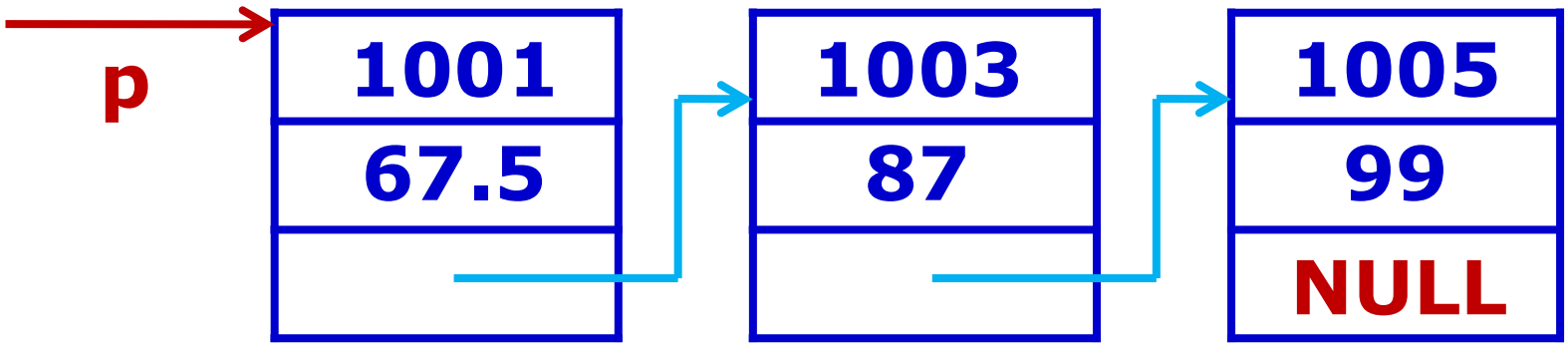
```
struct Student *creat(void)
{ struct Student *head,*p1,*p2;  n=0;
  p1=p2=( struct Student*) malloc(LEN);
  scanf("%ld,%f",&p1->num,&p1->score);
  head=NULL;
  while(p1->num!=0)
  {n=n+1;
   if(n==1) head=p1;
   else  p2->next=p1;
   p2=p1;
   p1=(struct Student*)malloc(LEN);
   scanf("%ld,%f",&p1->num,&p1->score);
  }
  p2->next=NULL;   return(head);
}
```

p1总是开辟新结点
p2总是指向最后结点
用p2和p1连接两个结点

```
int main()
{ struct Student *pt;
  pt=creat();
  printf("\nnum:%ld\nscore:%5.1f\n",
                    pt->num,pt->score);
  return 0;
}
```

9.4.4 输出链表

例**9.10** 编写一个输出链表的函数**print**。



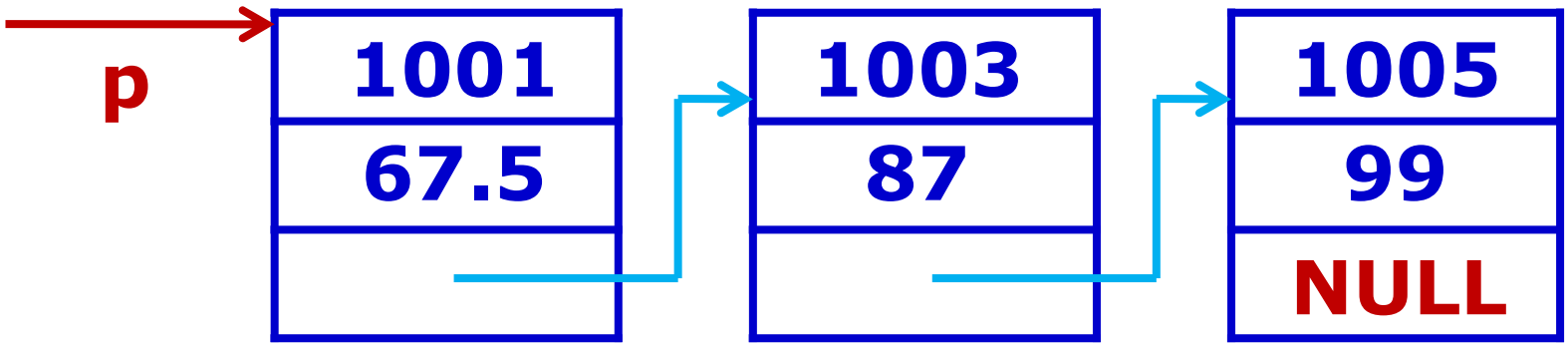
```
1001   67.5
```

➢解题思路：

◆输出**p**所指的结点

```
printf("%ld %5.1f\n",p->num,p->score);
```

◆使**p**后移一个结点



```
1001   67.5
```

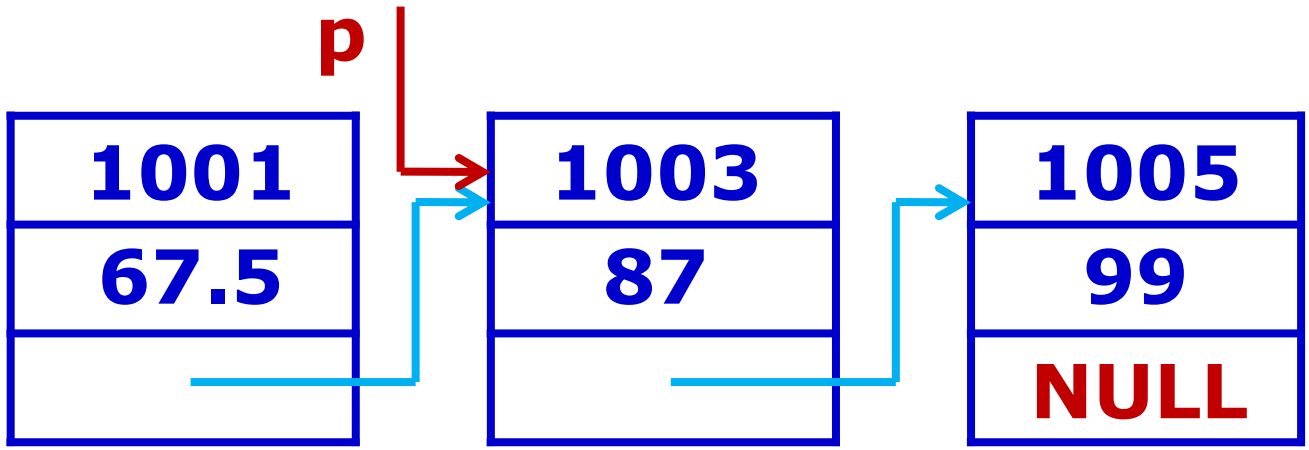
➢解题思路：

◆输出**p**所指的结点

```
printf("%ld %5.1f\n",p->num,p->score);
```

◆使**p**后移一个结点

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



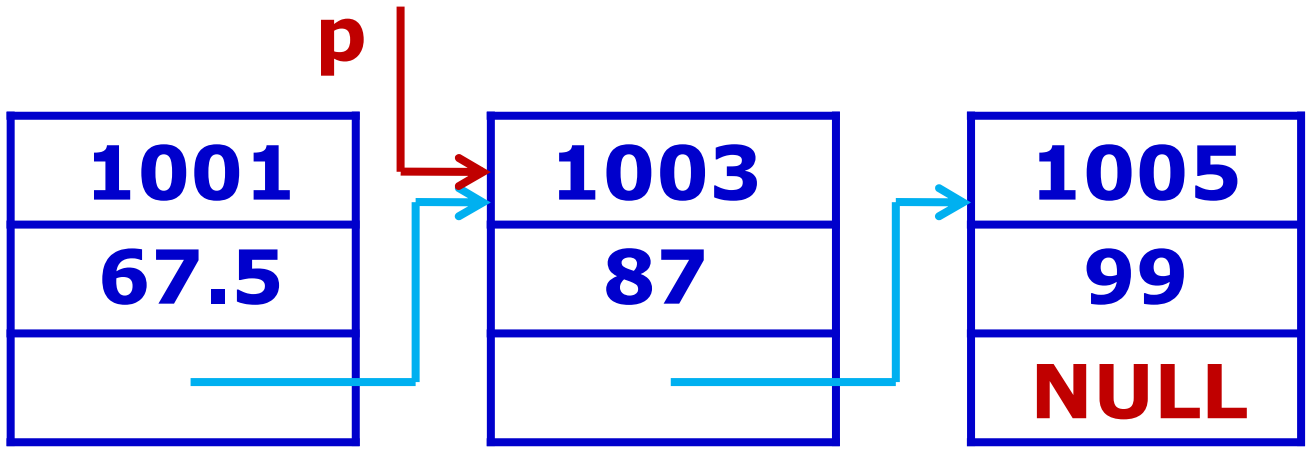
```
1001   67.5
1003   87.0
```

➢解题思路：

◆输出**p**所指的新结点

```
printf("%ld %5.1f\n",p->num,p->score);
```

◆使**p**后移一个结点



```
1001   67.5
1003   87.0
```

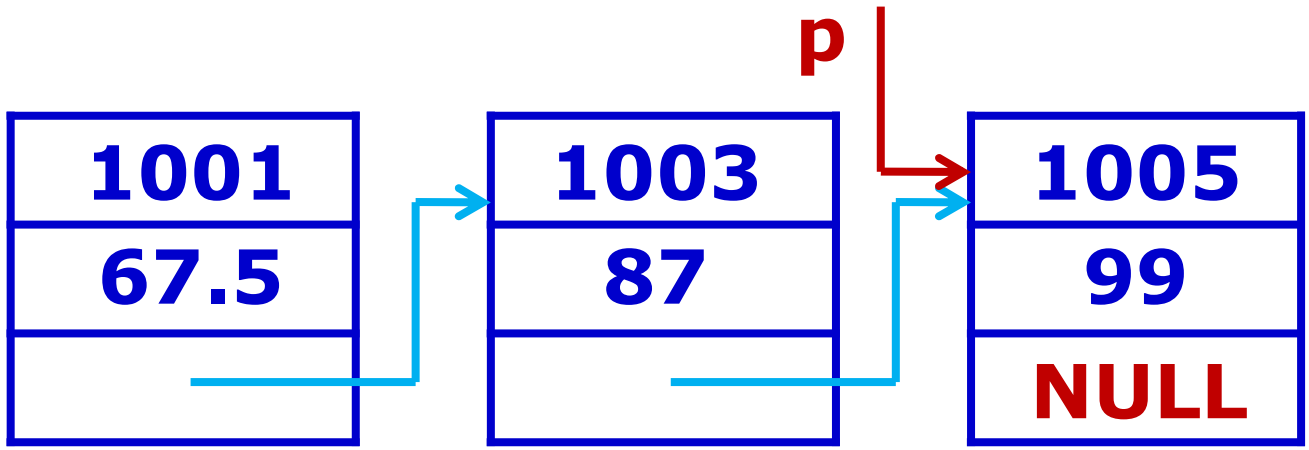
➢解题思路：

◆输出**p**所指的新结点

```
printf("%ld %5.1f\n",p->num,p->score);
```

◆使**p**后移一个结点

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



```
1001   67.5
1003   87.0
1005   99.0
```

➢解题思路：

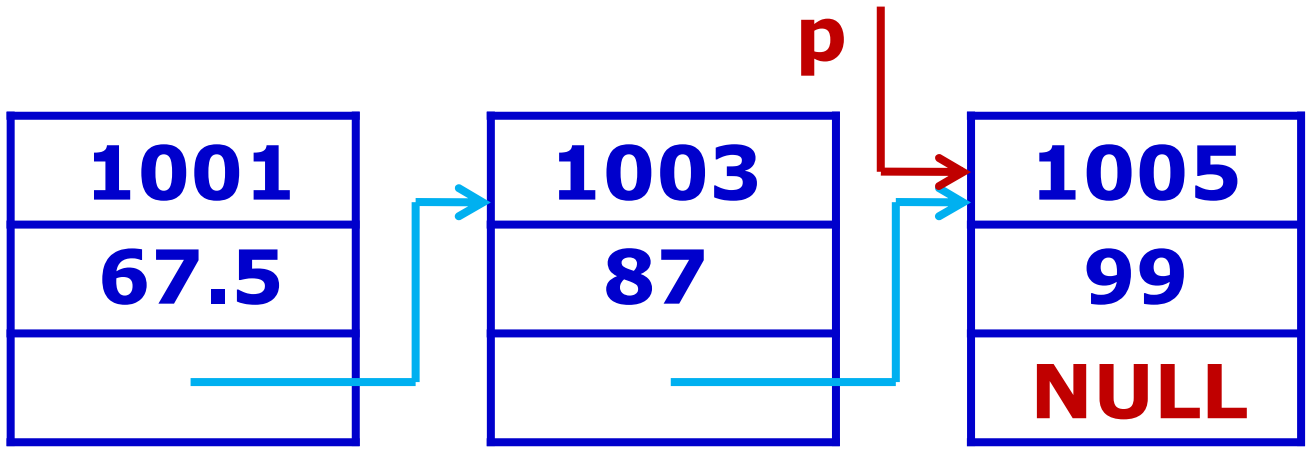
◆输出**p**所指的新结点

```
printf("%ld %5.1f\n",p->num,p->score);
```

◆使**p**后移一个结点

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

相当于**p=NULL;**



```
void print(struct Student *p)
{
  printf("\nThese %d records are:\n",n);
  if(p!=NULL)
    do
    { printf("%ld %5.1f\n",
                    p->num,p->score);
      p=p->next;
    }while(p!=NULL);
}
```