

# 第9章 用户自己建立数据类型

9.1 定义和使用结构体变量

9.2 使用结构体数组

9.3 结构体指针

9.4 用指针处理链表

9.5 共用体类型

9.6 使用枚举类型

9.7 用**typedef**声明新类型名

# 9.1 定义和使用结构体变量

## 9.1.1 自己建立结构体类型

## 9.1.2 定义结构体类型变量

## 9.1.3 结构体变量的初始化和引用



## 9.1.1 自己建立结构体类型

- 用户自己建立由不同类型数据组成的组合型的数据结构，它称为结构体
- 例如，一个学生的学号、姓名、性别、年龄、成绩、家庭地址等项，是属于同一个学生的，因此组成一个组合数据，如**student\_1**的变量，反映它们之间的内在联系



## 9.1.1 自己建立结构体类型

```
struct Student  
{ int num;  
  char name[20];  
  char sex;  
  int age;  
  float score;  
  char addr[30];  
};
```

◆由程序设计者指定了一个结构体类型

**struct Student**

◆它包括

**num,name,sex,age, score,addr**等不同类型的成员



## 9.1.1 自己建立结构体类型

➤ 声明一个结构体类型的一般形式为：

**struct** 结构体名

{ 成员表列 };

类型名 成员名;



## 9.1.1 自己建立结构体类型

➤说明:

(1)结构体类型并非只有一种，而是可以设计出许多种结构体类型，例如

**struct Teacher**

**struct Worker**

**struct Date**等结构体类型

◆各自包含不同的成员



## 9.1.1 自己建立结构体类型

➤说明:

(2) 成员可以属于另一个结构体类型。

```
struct Date  
{ int month; int day; int year; };  
struct Stu  
{ int num;char name[20];  
  char sex;int age;  
  struct Date birthday;  
  char addr[30];  
};
```



## 9.1.1 自己建立结构体类型

➤说明:

**(2)** 成员可以属于另一个结构体类型。

num	name	sex	age	birthday			addr
				month	day	year	





## 9.1.2 定义结构体类型变量

- 前面只是建立了一个结构体类型，它相当于一个模型，并没有定义变量，其中并无具体数据，系统对之也不分配存储单元。
- 相当于设计好了图纸，但并未建成具体的房屋。为了能在程序中使用结构体类型的数据，应当定义结构体类型的变量，并在其中存放具体的数据。



## 9.1.2 定义结构体类型变量

1. 先声明结构体类型，再定义该类型变量

➤ 声明结构体类型 **struct Student**，可以用它来定义变量

**struct Student**   **student1,student2;**

结构体类型名

结构体变量名



## 9.1.2 定义结构体类型变量

1. 先声明结构体类型，再定义该类型变量

➤ 声明结构体类型 **struct Student**，可以用它来定义变量

```
struct Student student1, student2;
```

**student1**

10001	Zhang Xin	M	19	90.5	Shanghai
-------	-----------	---	----	------	----------

**student2**

10002	Wang Li	F	20	98	Beijing
-------	---------	---	----	----	---------



## 9.1.2 定义结构体类型变量

### 2.在声明类型的同时定义变量

```
struct Student  
{ int num;  
    char name[20];  
    char sex;  
    int age;  
    float score;  
    char addr[30];  
} student1,student2;
```

成员的引用格式：

1 赋初值

2 引用

结构体变量.成员

**student1.num=1001;**

**student2.score=70.5;**



## 9.1.2 定义结构体类型变量

### 3. 不指定类型名而直接定义结构体类型变量

➤ 其一般形式为：

**struct**

**{ 成员表列 }变量名表列;**

指定了一个无名的结构体类型。



## 9.1.2 定义结构体类型变量

**(1)** 结构体类型与结构体变量是不同的概念，不要混同。只能对变量赋值、存取或运算，而不能对一个类型赋值、存取或运算。在编译时，对类型是不分配空间的，只对变量分配空间。



## 9.1.2 定义结构体类型变量

- (2)** 结构体类型中的成员名可以与程序中的变量名相同，但二者不代表同一对象。
- (3)** 对结构体变量中的成员（即“域”），可以单独使用，它的作用与地位相当于普通变量。



## 9.1.3 结构体变量的初始化和引用

**例9.1** 把一个学生的信息(包括学号、姓名、性别、住址)放在一个结构体变量中，然后输出这个学生的信息。

➤ 解题思路：

- ◆ 自己建立一个结构体类型，包括有关学生信息的各成员
- ◆ 用它定义结构体变量，同时赋以初值
- ◆ 输出该结构体变量的各成员





```
#include <stdio.h>
```

```
int main()
```

```
{struct Student
```

```
{ long int num;
```

```
char sex;
```

```
}a={10101,
```

```
"Li Lin",
```

```
"M",
```

```
"123 Beijing Road"};
```

```
printf("NO.:%ld\nname:%s\n
```

```
sex:%c\naddress:%s\n",
```

```
a.num,a.name,a.sex,a.addr);
```

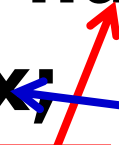
```
return 0;
```

```
}
```

```
NO.:10101
name:Li Lin
sex:M
address:123 Beijing Road
```

char name[20];

char addr[20];



```
#include <stdio.h>
int main()
{struct Student
    { long int num;          char name[20];
      char sex;              char addr[20];
    }a={10101,"Li Lin",'M',
        "123 Beijing Road"};
printf("NO.:%ld\nname:%s\n
        sex:%c\naddress:%s\n",
        a.num,a.name,a.sex,a.addr);
return 0;
}
```



```
#include <stdio.h>
int main()
{struct Student
    { long int num;          char name[20];
      char sex;              char addr[20];
    }a={10101,"Li Lin",'M',
        "123 Beijing Road"};

    a.num=10010; 对
    printf("%s\n",a); 不对

    .....
}
```



```
#include <stdio.h>
int main()
{struct Student
  { long int num;      char name[20];
    char sex;          char addr[20];
  }a={10101,"Li Lin",'M',
      "123 Beijing Road"};

  struct Student b;

  b=a; 对
  b.num++; 对
  .....
}
```



```
#include <stdio.h>
int main()
{struct Student
    { long int num;          char name[20];
      char sex;              char addr[20];
    }a={10101,"Li Lin",'M',
        "123 Beijing Road"};

    scanf("%ld",&a.num); 对
    printf("%o",&a); 对
    scanf("%ld,%s,%c,%s\n",&a); 错
    .....
}
```



```
#include <stdio.h>
int main()
{ struct Date
  { int month; int day; int year; };
  struct Stu
  { int num; char name[20];
    char sex; int age;
    struct Date birthday;
    char addr[30];
  } a, b;

  a.birthday.month=12; 对
  a.age=10; b.age=9; 对
  sum=a.age+b.age; 对
```



**例9.2** 输入两个学生的学号、姓名和成绩，输出成绩较高学生的学号、姓名和成绩

➤ 解题思路：

**(1)**定义两个结构相同的结构体变量**student1**和**student2**；

**(2)**分别输入两个学生的学号、姓名和成绩；

**(3)**比较两个学生的成绩，如果学生**1**的成绩高于学生**2**，就输出学生**1**的全部信息，如果学生**2**的成绩高于学生**1**，就输出学生**2**的全部信息。如果二者相等，输出**2**个学生的全部信息



```
#include <stdio.h>
int main()
{ struct Student
  { int num;
    char name[20];
    float score;
  } student1, student2;
  scanf("%d%s%f", &student1.num,
        student1.name, &student1.score);
  scanf("%d%s%f", &student2.num,
        student2.name, &student2.score);
```

不能加&





```
printf("The higher score is:\n");
if (student1.score>student2.score)
    printf("%d %s %6.2f\n",student1.num,
        student1.name, student1.score);
else if (student1.score<student2.score)
    printf("%d %s %6.2f\n",student2.num,
        student2.name, student2.score);
else
{printf("%d %s %6.2f\n",student1.num,
    student1.name, student1.score);
    printf("%d %s %6.2f\n",student2.num,
        student2.name, student2.score);
}
return 0;
}
```

```
10101 Wang 89
10103 Ling 90
The higher score is:
10103 Ling 90.00
```



## **9.2 使用结构体数组**

### **9.2.1 定义结构体数组**

### **9.2.2 结构体数组的应用举例**



## ➤ 定义2

### (1) 定义结构体数组一般形式是

① **struct** 结构体名

{成员表列} 数组名[数组长度];

② 先声明一个结构体类型，然后再用此类型定义结构体数组：

结构体类型 数组名[数组长度];

如：

**struct Person leader[3];**



➤ 定义2:

**(2)**对结构体数组初始化的形式是在定义数组的后面加上:

= {初值表列} ;

如:

```
struct Person leader[3]=  
    {"Li",0,"Zhang",0,"Fun",0};
```



## 9.2.1 定义结构体数组

**例9.3** 有3个候选人，每个选民只能投票选一人，要求编一个统计选票的程序，先后输入被选人的名字，最后输出各人得票结果。



## 9.2.1 定义结构体数组

### ➤ 解题思路:

- ◆ 设一个结构体数组，数组中包含**3**个元素
- ◆ 每个元素中的信息应包括候选人的姓名(字符型)和得票数(整型)
- ◆ 输入被选人的姓名，然后与数组元素中的“姓名”成员比较，如果相同，就给这个元素中的“得票数”成员的值加**1**
- ◆ 输出所有元素的信息



```
#include <string.h>
#include <stdio.h>
struct Person
{ char name[20];
  int count;
}leader[3]={"Li",0,"Zhang",0,"Sun",0};
```

全局的结构体数组

**leader[0]**      **name    count**

<b>Li</b>	<b>0</b>
<b>Zhang</b>	<b>0</b>
<b>Sun</b>	<b>0</b>



```

int main()
{ int i,j;  char leader_name[20];
  for (i=1;i<=10;i++)
  { scanf("%s",leader_name);
    for(j=0;j<3;j++)
      if(strcmp(leader_name,
                 leader[j].name)==0)
        leader[j].count++;
  }
  for(i=1;i<=10;i++)
    leader[i].count=leader[j].count+1;
  return 0;
}

```





```

int main()
{ int i,j;  char leader_name[20];
  for (i=1;i<=10;i++)
  { scanf("%s",leader_name);
    for(j=0;j<3;j++)
      if(strcmp(leader_name,
                 leader[j].name)==0)
        leader[j].count++;
  }
  for(i=0;i<3;i++)
    printf("%5s:%d\n",leader[i].name,
           leader[i].count);
  return 0;
}

```

```

Li
Li
Fun
Zhang
Zhang
Fun
Li
Fun
Zhang
Li

```

```

Li:4
Zhang:3
Sun:3

```



## 9.2.2 结构体数组的应用举例

**例9.4** 有**n**个学生的信息(包括学号、姓名、成绩)，要求按照成绩的高低顺序输出各学生的信息。

➤ **解题思路：**用结构体数组存放**n**个学生信息，采用选择法对各元素进行排序(进行比较的是各元素中的成绩)。



```

#include <stdio.h>
struct Student
{ int num; char name[20]; float score; };
int main()
{ struct Student
  stu[5]={ {10101,"Zhang",78 },
           {10103,"Wang",98.5},
           {10106,"Li", 86 },
           {10108,"Ling", 73.5},
           {10111,"", 0 } };
  struct Student temp;
  const int n =30; int i,j,k;

```

常变量

若人数变为30



```
#include <stdio.h>
```

```
struct Student
```

```
#define N 5
```

```
{ int num; char name[20]; float score; };
```

```
int main()
```

```
{ struct Student
```

```
    stu[5]={ {10101,"Zhang",78  },
```

```
              {10103,"Wang",98.5},
```

```
              {10106,"Li",    86  },
```

```
              {10108,"Ling", 73.5},
```

```
              {10110,"Fun"
```

注意temp的类型

```
    struct Student temp;
```

```
const int n = 5 ; int i,j,k;
```



```
printf("The order is:\n");
```

```
for(i=0;i<n-1;i++)
```

```
{ k=i;
```

```
  for(j=i+1;j<n;j++)
```

```
    if(stu[j].score>stu[k].score) k=j;
```

```
  temp=stu[k];
```

```
  stu[k]=stu[i];  stu[i]=temp;
```

```
}
```

```
for(i=0;i<n;i++)
```

```
  printf("%6d %8s %5.2f\n",
```

```
    stu[i].num,stu[i].name,stu[i].score);
```

```
printf("\n");
```

```
return 0;
```

写法上与普通变量一致

```
}
```



## 9.3 结构体指针

9.3.1 指向结构体变量的指针

9.3.2 指向结构体数组的指针

9.3.3 用结构体变量和结构体变量的指针作函数参数



## 9.3.1 指向结构体变量的指针

- 指向结构体对象的指针变量既可以指向结构体变量，也可以用来指向结构体数组中的元素。
- 指针变量的基类型必须与结构体变量的类型相同。例如：

```
struct Student *pt;
```



## 9.3.1 指向结构体变量的指针

**例9.5** 通过指向结构体变量的指针变量输出结构体变量中成员的信息。

➤ 解题思路：在已有的基础上，本题要解决两个问题：

- ◆ 怎样对结构体变量成员赋值；
- ◆ 怎样通过指向结构体变量的指针访问结构体变量中成员。





```
#include <stdio.h>  
#include <string.h>  
int main()  
{ struct Student  
    { long num;  
      char name[20];  
      char sex;  
      float score;  
    };  
.....
```



```
struct Student stu_1;
```

```
struct Student * p;
```

```
p=&stu_1;
```

```
stu_1.num
```

```
strcpy(stu_1.name, "Li Lin");
```

```
stu_1.sex = 'M'; stu_1.score = 89.5;
```

```
printf("No.:%ld\n", stu_1.num);
```

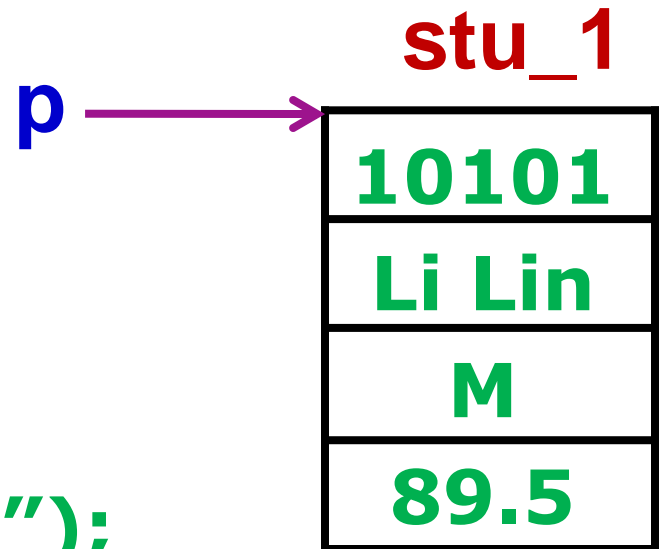
```
printf("name:%s\n", stu_1.name);
```

```
printf("sex:%c\n", stu_1.sex);
```

```
printf("score:%5.1f\n", stu_1.score);
```

```
return 0;
```

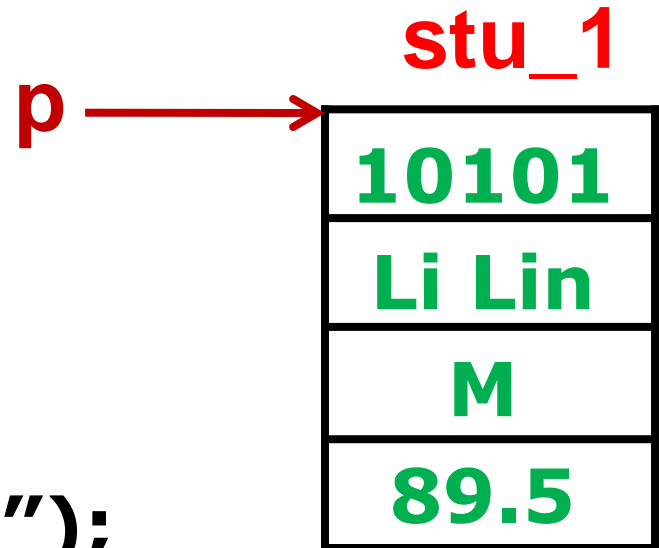
```
}
```



```

struct Student stu_1;
struct Student * p;
p=&stu_1;
stu_1.num=10101;
strcpy(stu_1.name,"Li Lin");
stu_1.sex='M'; stu_1.score=89.5;
printf("No.:%ld\n",stu_1.num);
printf("name:%s\n",stu_1.name);
printf("sex: (*p).num 1.sex);
printf("score (*p).name tu_1.score);
return 0;
}

```



**(\*p).num**

**(\*p).name**

**(\*p).sex**

**(\*p).score**

## ➤说明:

◆为了方便和直观，C语言允许把  
**(\*p).num**用**p->num**来代替

◆**(\*p).name**等价于**p->name**

◆如果**p**指向一个结构体变量**stu**，以下等价：

① **stu.成员名**(如**stu.num**)

② **(\*p).成员名**(如**(\*p).num**)

**p->成员名**(如**p->num**)



## 9.3.2 指向结构体数组的指针

**例9.6** 有3个学生的信息，放在结构体数组中，要求输出全部学生的信息。



➤ 解题思路：用指向结构体变量的指针处理

(1) 声明 **struct Student**，并定义结构体数组、初始化

(2) 定义指向 **struct Student** 类型指针 **p**

(3) 使 **p** 指向数组首元素，输出元素中各信息

(4) 使 **p** 指向下一个元素，输出元素中各信息

(5) 再使 **p** 指向结构体数组的下一个元素，输出它指向的元素中的有关信息



```
#include <stdio.h>
```

```
struct Student
```

```
{ int num;   char name[20];
```

```
    char sex;  int age;
```

```
};
```

```
struct Student stu[3]={
```

```
    {10101,"Li Lin",'M',18},
```

```
    {10102,"Zhang Fun",'M',19},
```

```
    {10104,"Wang Min",'F',20} };
```



No. Name

sex age

```
int main()
{ struct Student *p;
  printf(" No. Name          sex age\n");
  for(p=stu;p<stu+3;p++)
    printf("%5d %-20s %2c %4d\n",
           p->num, p->name,
           p->sex, p->age);
  return 0;
}
```



10101	Li Lin	M	18
10102	Zhang Fang	M	19
10104	Wang Min	F	20

stu[0]

stu[1]

stu[2]



```
No.  Name          sex age
10101 Li Lin       M   18
```

```
int main()
```

```
{ struct Student *p;
```

```
printf(" No.  Name          sex age\n");
```

```
for(p=stu;p<stu+3;p++)
```

```
printf("%5d %-20s %2c %4d\n",
```

```
p->num, p->name,
```

```
p->sex, p->age);
```

```
return 0;
```

```
}
```

**p** →

10101	Li Lin	M	18
10102	Zhang Fang	M	19
10104	Wang Min	F	20

**stu[0]**

**stu[1]**

**stu[2]**



```
int main()
```

```
{ struct Student *p;
```

```
    printf(" No.  Name          sex  age\n");
```

```
    for(p=stu;p<stu+3;p++)
```

```
        printf("%5d %-20s %2c %4d\n",
```

```
            p->num, p->name,
```

```
            p->sex, p->age);
```

```
    return 0;
```

```
}
```

**p** →

10101	Li Lin	M	18
10102	Zhang Fang	M	19
10104	Wang Min	F	20

**stu[0]**

**stu[1]**

**stu[2]**



No.	Name	sex	age
10101	Li Lin	M	18
10102	Zhang Fun	M	19

```
int main()
```

```
{ struct Stude
```

```
printf(" No. Name sex age\n");
```

```
for(p=stu;p<stu+3;p++)
```

```
printf("%5d %-20s %2c %4d\n",
```

```
p->num, p->name,
```

```
p->sex, p->age);
```

```
return 0;
```

```
}
```

10101	Li Lin	M	18
10102	Zhang Fang	M	19
10104	Wang Min	F	20

stu[0]

stu[1]

stu[2]

p



No.	Name	sex	age
10101	Li Lin	M	18
10102	Zhang Fun	M	19
10104	Wang Min	F	20

## 9.3.3 用结构体变量和结构体变量的指针作函数参数

- 将一个结构体变量的值传递给另一个函数，有**3**个方法。



## (1) 用结构体变量的成员作参数。

例如，用**stu[1].num**或**stu[2].name**作函数实参，将实参值传给形参。

- ◆用法和用普通变量作实参是一样的，属于“值传递”方式。
- ◆应当注意实参与形参的类型保持一致。



## (2) 用结构体变量作实参。

- ◆用结构体变量作实参时，将结构体变量所占的内存单元的内容全部按顺序传递给形参，形参也必须是同类型的结构体变量
- ◆在函数调用期间形参也要占用内存单元。这种传递方式在空间和时间上开销较大
- ◆在被调用函数期间改变形参（也是结构体变量）的值，不能返回主调函数
- ◆一般较少用这种方法



**(3)**用指向结构体变量（或数组元素）的指针作实参，将结构体变量（或数组元素）的地址传给形参。



**例9.7** 有**n**个结构体变量，内含学生学号、姓名和**3**门课程的成绩。要求输出平均成绩最高的学生的信息(包括学号、姓名、**3**门课程成绩和平均成绩)。





- 解题思路：将**n**个学生的数据表示为结构体数组。按照功能函数化的思想，分别用**3**个函数来实现不同的功能：
- ◆ 用**input**函数输入数据和求各学生平均成绩
  - ◆ 用**max**函数找平均成绩最高的学生
  - ◆ 用**print**函数输出成绩最高学生的信息
  - ◆ 在主函数中先后调用这**3**个函数，用指向结构体变量的指针作实参。最后得到结果。
  - ◆ 本程序假设**n=3**



```
#include <stdio.h>
#define N 3
struct Student
{ int num;
  char name[20];
  float score[3];
  float aver;
};
```

4个成员

输入前3个成员值

计算最后成员值



```
int main()  
{ void input(struct Student stu[]);  
  struct Student max(struct Student stu[]);  
  void print(struct Student stu);  
  struct Student stu[N], *p=stu;  
  input(p);  
  print(max(p));  
  return 0;  
}
```



```
void input(struct Student stu[])
```

```
{ int i;
```

```
    printf("请输入各学生的信息:
```

**i=0**

```
    学号、姓名、三门课成绩:\n");
```

```
    for(i=0;i<N
```

输入第1个成员

输入第2个成员值

```
    {scanf("%d %s
```

输入第3个成员值

```
        &stu[i].id, &stu[i].name,
        &stu[i].score[0], &stu[i].score[1],
        &stu[i].score[2]);
```

计算第4个成员值

```
    stu[i].aver=(stu[i].score[0]+
        stu[i].score[1]+stu[i].score[2])/3.0;
```

```
}
```

stu

10101	Li	78	89	98	88.33

stu[0]

stu[1]

stu[2]

```
void input(struct Student stu[])
```

```
{ int i;
```

```
printf("请输入各学生的信息:
```

**i=1**

```
学号、姓名、三门课成绩:\n");
```

```
for(i=0;i<N
```

输入第1个成员

输入第2个成员值

```
{scanf("%d %s
```

输入第3个成员值

```
&stu[i].id, &stu[i].name,  
&stu[i].score[0], &stu[i].score[1],  
&stu[i].score[2]);
```

计算第4个成员值

```
stu[i].aver=(stu[i].score[0]+  
stu[i].score[1]+stu[i].score[2])/3.0;
```

```
}
```

**stu**

10101	Li	78	89	98	88.33	stu[0]
10103	Wang	98.5	87	69	84.83	stu[1]
						stu[2]

```
void input(struct Student stu[])
```

```
{ int i;
```

```
printf("请输入各学生的信息:
```

**i=2**

```
学号、姓名、三门课成绩:\n");
```

```
for(i=0;i<N
```

输入第1个成员

输入第2个成员值

```
{scanf("%d %s
```

输入第3个成员值

```
&stu[i].id, &stu[i].name,
```

计算第4个成员值

```
&stu[i].score[0], &stu[i].score[1],
```

```
&stu[i].score[2]);
```

```
stu[i].aver=(stu[i].score[0]+  
stu[i].score[1]+stu[i].score[2])/3.0;
```

```
}
```

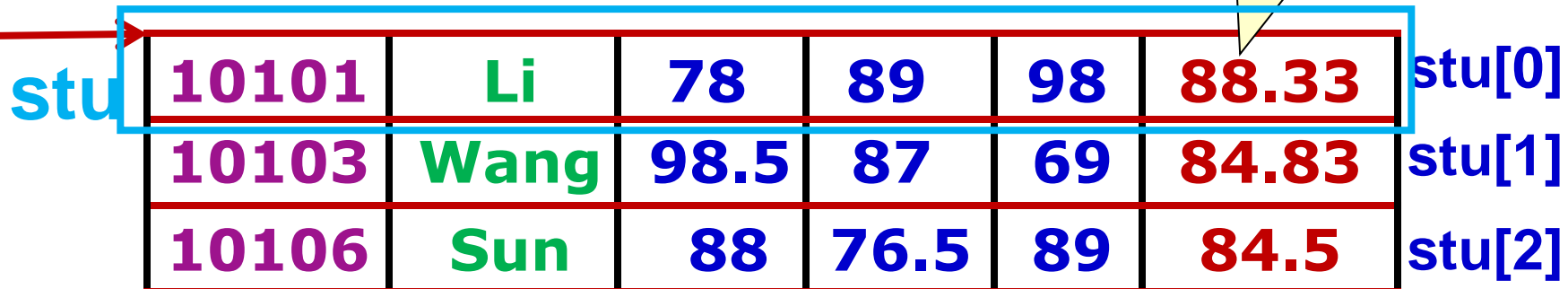
stu

10101	Li	78	89	98	88.33	stu[0]
10103	Wang	98.5	87	69	84.83	stu[1]
10106	Sun	88	76.5	89	84.5	stu[2]

```
struct Student max(struct Student stu[])  
{int i,m=0;  
  for(i=0;i<N;i++)  
    if (stu[i].aver>stu[m].aver) m=i;  
  return stu[m];  
}
```

返回

最大



The diagram shows an array of three student records. A red arrow points to the first element, and a blue arrow points to the last element. A yellow callout bubble points to the average score of the first student, indicating it is the maximum. The array is labeled 'stu' on the left and 'stu[0]', 'stu[1]', and 'stu[2]' on the right.

stu	10101	Li	78	89	98	88.33	stu[0]
	10103	Wang	98.5	87	69	84.83	stu[1]
	10106	Sun	88	76.5	89	84.5	stu[2]

```
void print(struct student stud)
{ printf("\n成绩最高的学生是:\n");
  printf("学号:%d", stud.num);
  printf("姓名:%s", stud.name);
  printf("三门课成绩: %5.1f, %5.1f, %5.1f\n", stud.score[0], stud.score[1], stud.score[2]);
  printf("平均成绩: %6.2f\n", stud.aver);
}
```

```
成绩最高的学生是:
学号:10101
姓名:Li
三门课成绩: 78.0, 89.0, 98.0
平均成绩: 88.33
```

三门课成绩:%5.1f,%5.1f,%5.1f\n

平均成绩:%6.2f\n", stud.num,

stud.name,stud.score[0],

stud.score[1],stud.score[2],stud.aver);

}

	num	name	score			aver	
stud	10101	Li	78	89	98	88.33	stu[0]
	10103	Wang	98.5	87	69	84.83	stu[1]
	10106	Sun	88	76.5	89	84.5	stu[2]





➤ 以上**3**个函数的调用，情况各不相同：

- ◆ 调用**input**函数时，实参是指针变量，形参是结构体数组，传递的是结构体元素的地址，函数无返回值。
- ◆ 调用**max**函数时，实参是指针变量，形参是结构体数组，传递的是结构体元素的地址，函数的返回值是结构体类型数据。
- ◆ 调用**print**函数时，实参是结构体变量，形参是结构体变量，传递的是结构体变量中各成员的值，函数无返回值。



## **9.4 用指针处理链表**

### **9.4.1 什么是链表**

### **9.4.2 建立简单的静态链表**

### **9.4.3 建立动态链表**

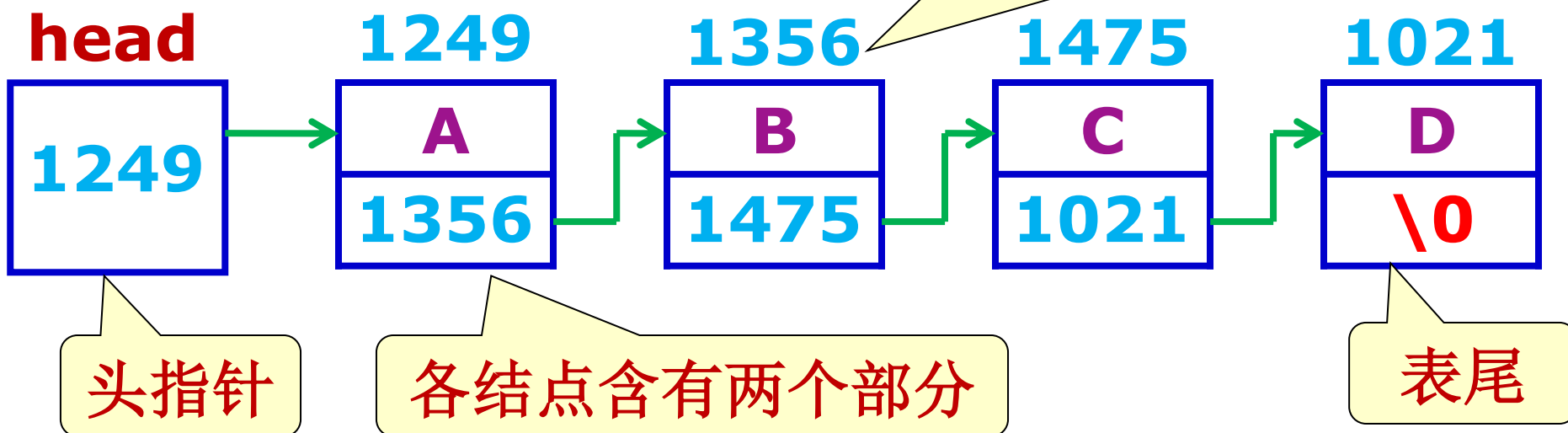
### **9.4.4 输出链表**



## 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

各结点地址不连续

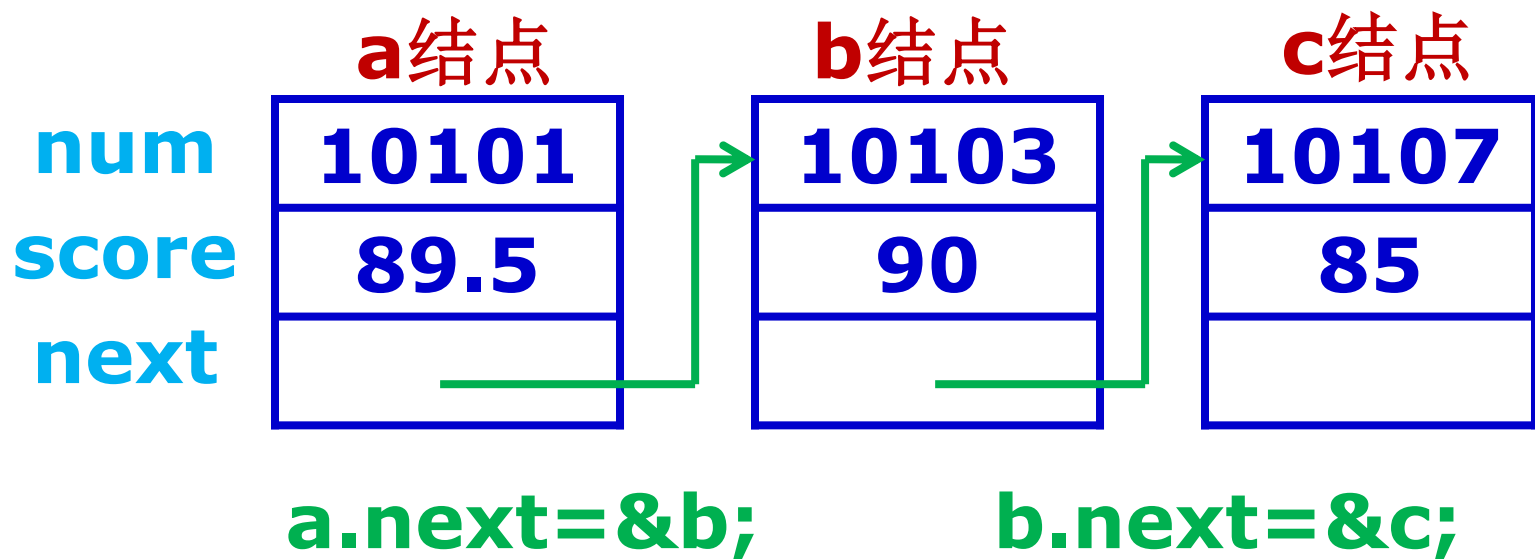


## 9.4.1 什么是链表

- 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

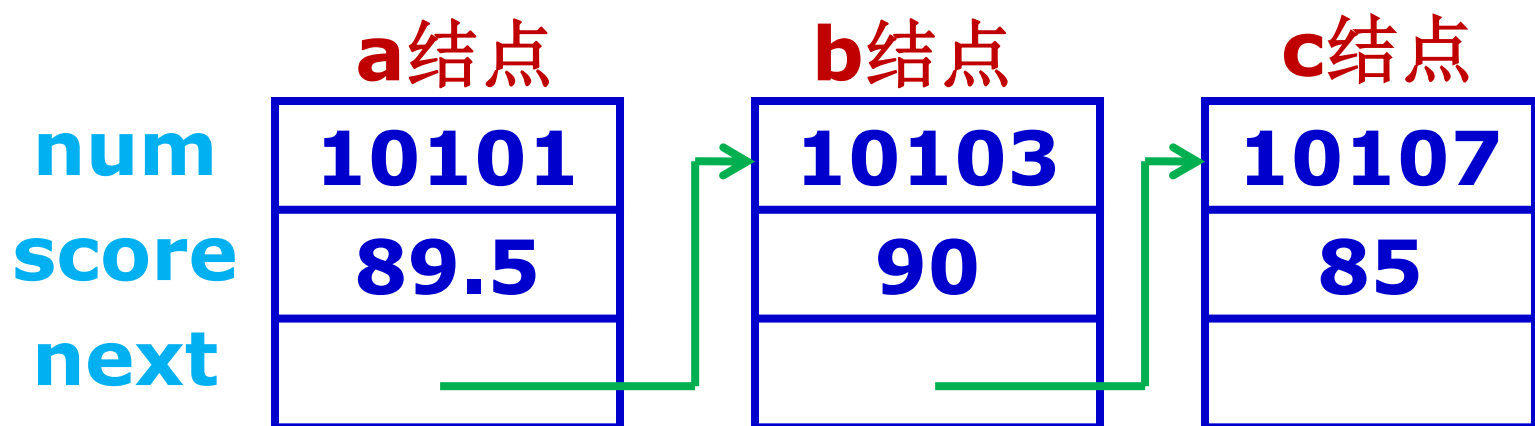


```
struct Student
{
    int num;
    float score;
    struct Student *next;
}a,b,c;
```



## 9.4.2 建立简单的静态链表

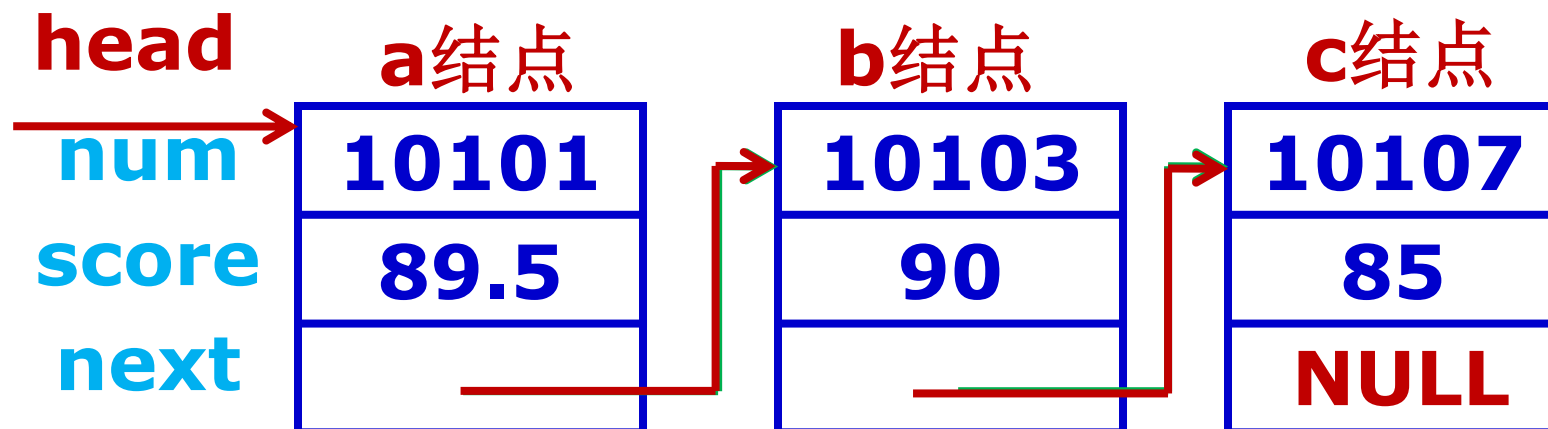
例**9.8** 建立一个如图所示的简单链表，它由**3**个学生数据的结点组成，要求输出各结点中的数据。



## 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;`



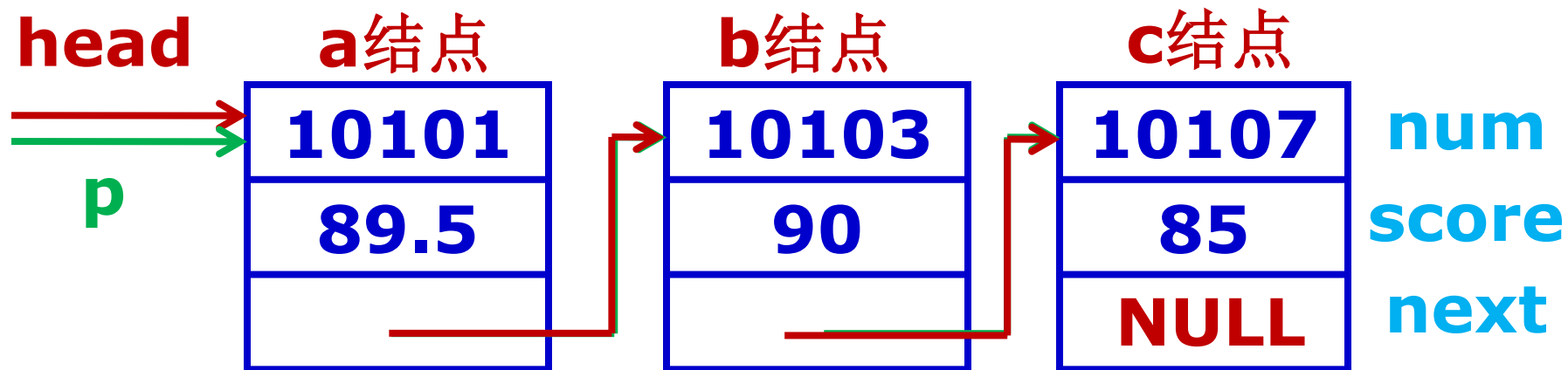
```
#include <stdio.h>  
struct Student  
{ int num;  
    float score;  
    struct Student *next;  
};
```





```
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;  
}
```

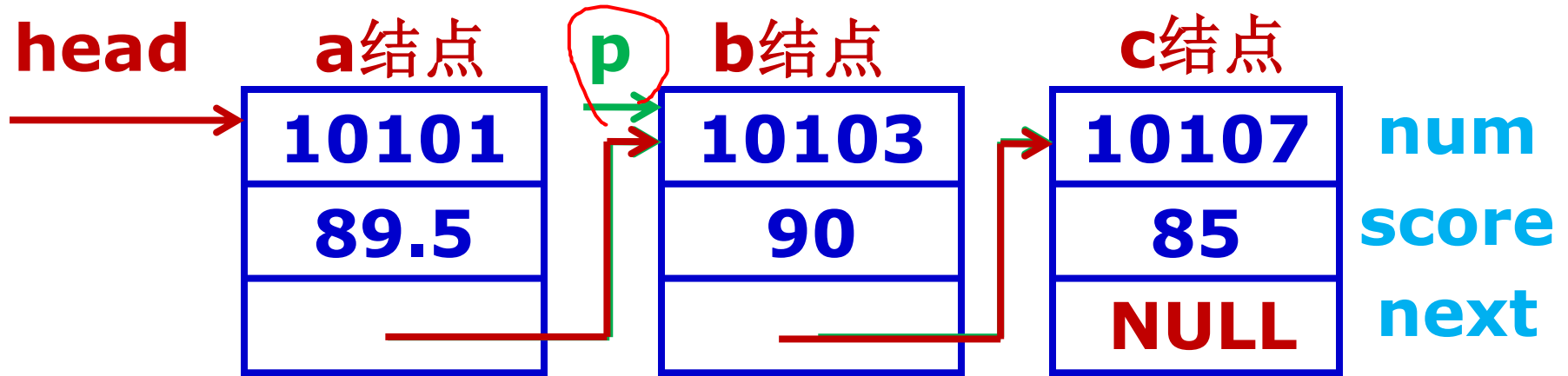




```
p=head;
do
{printf("%ld%5.1f\n",p->num,p->score);
  p=p->next; 相当于p=&b;
}while(p!=NULL);
return 0;
}
```

10101 89.5

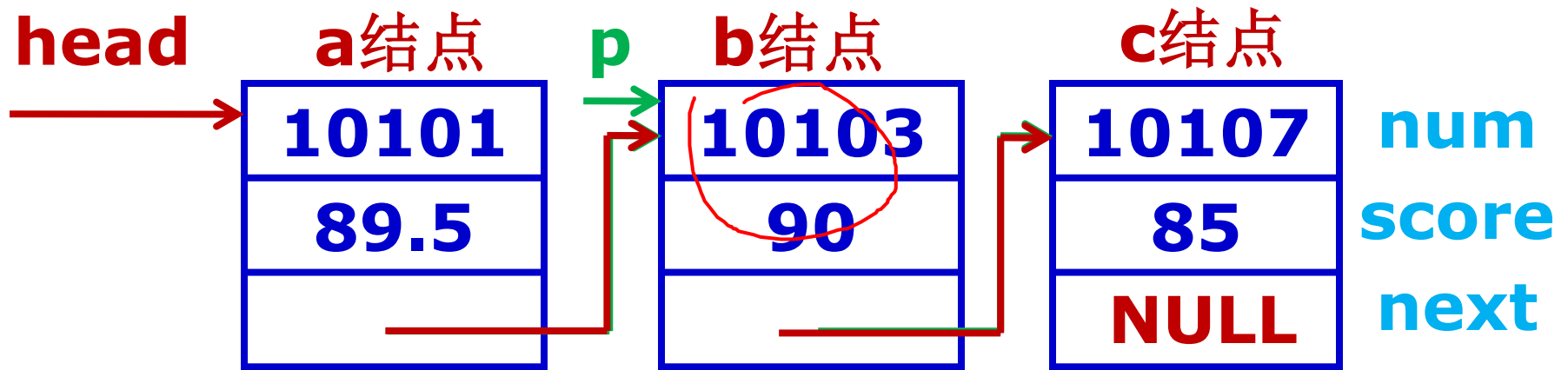




```
p=head;
do
{printf("%ld%5.1f\n",p->num,p->score);
  p=p->next; 相当于p=&b;
}while(p!=NULL);
return 0;
}
```

10101 89.5

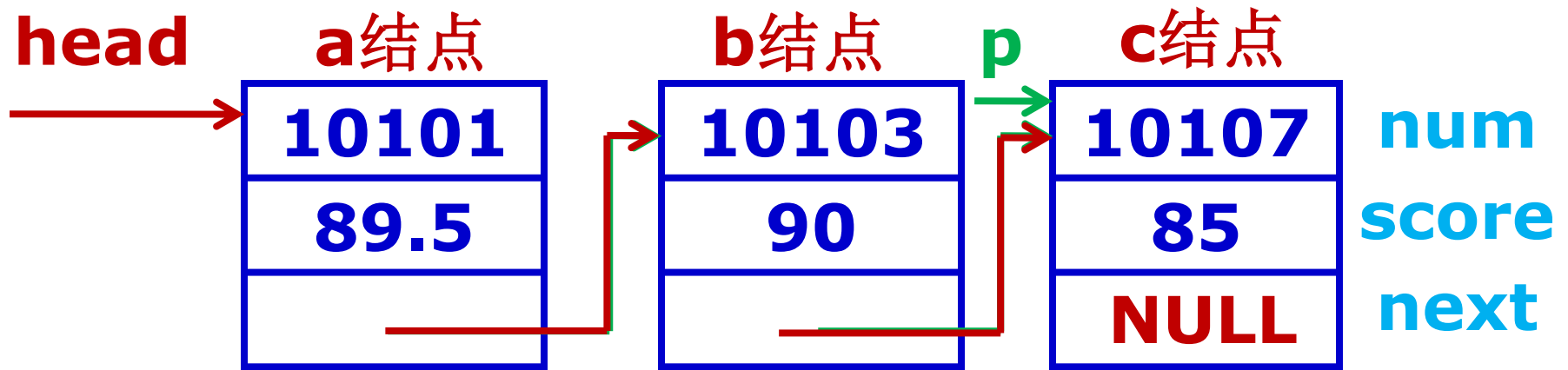




```
p=head;
do
{printf("%ld%5.1f\n",p->num,p->score);
  p=p->next; 相当于p=&c;
}while(p!=NULL);
return 0;
}
```

```
10101 89.5
10103 90.0
```



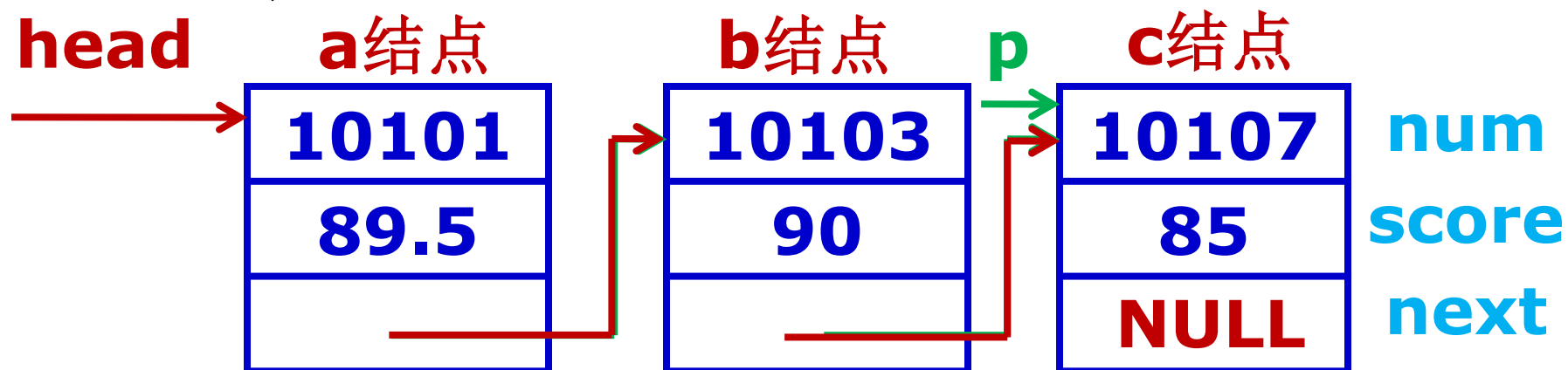


```
p=head;
do
{printf("%ld%5.1f\n",p->num,p->score);
  p=p->next; 相当于p=&c;
}while(p!=NULL);
return 0;
}
```

10101	89.5
10103	90.0



## 静态链表



```
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。



## 9.4.3 建立动态链表

**例9.9** 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

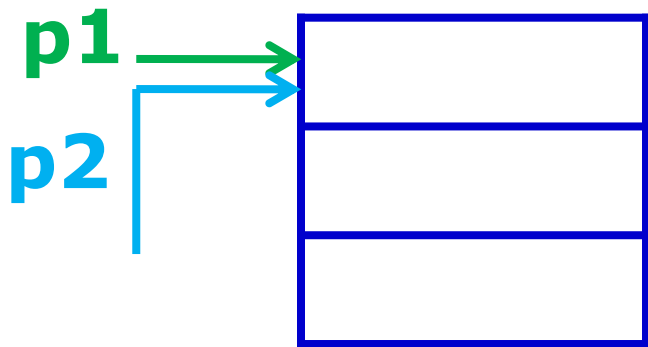
```
struct Student *head, *p1, *p2;
```



## ➤ 解题思路:

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

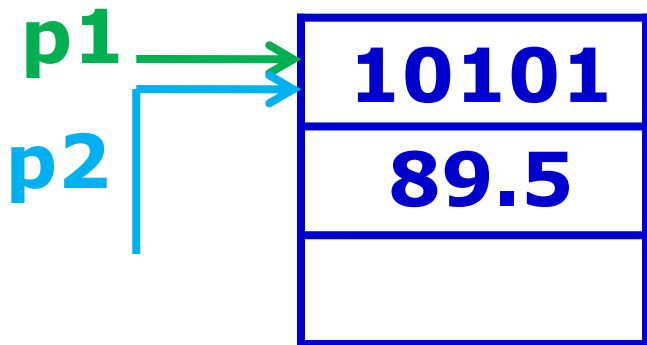
**p1=p2=(struct Student\*)malloc(LEN);**



## ➤ 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);
```

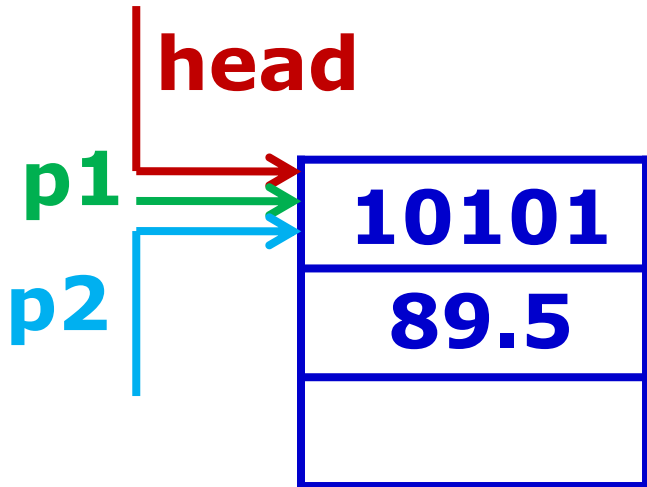


## ➤ 解题思路:

◆ 读入一个学生的数据给 **p1** 所指的第一个结点

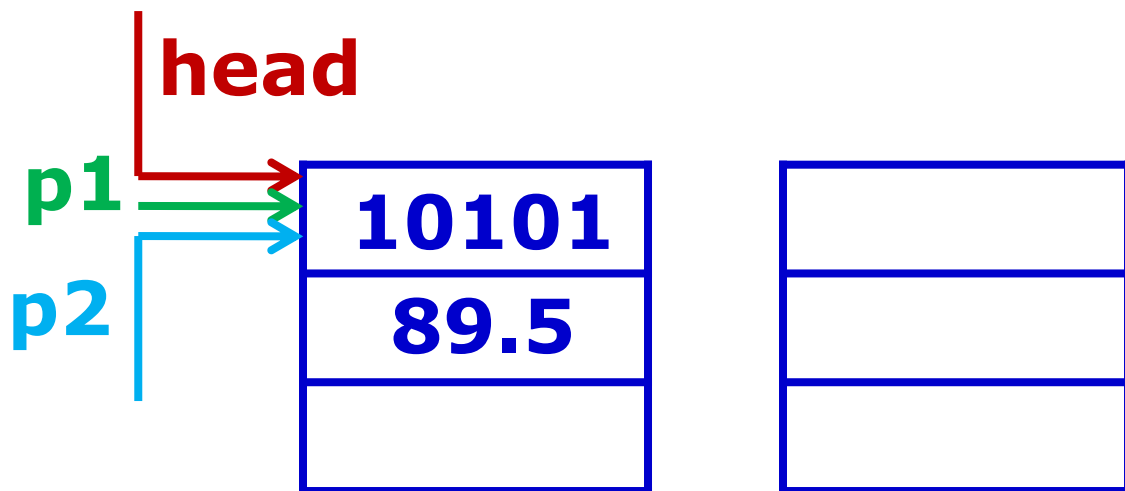
`scanf("%ld,%f",&p1->num,&p1->score);`

◆ 使 **head** 也指向新开辟的结点



## ➤ 解题思路:

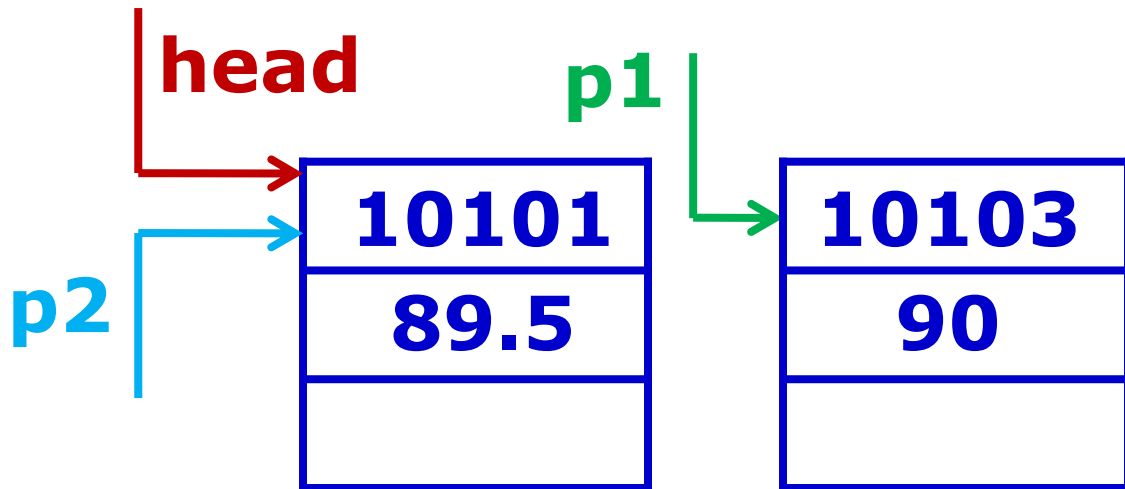
◆ 再开辟另一个结点并使**p1**指向它，接着输入该结点的数据



## ➤ 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);
```

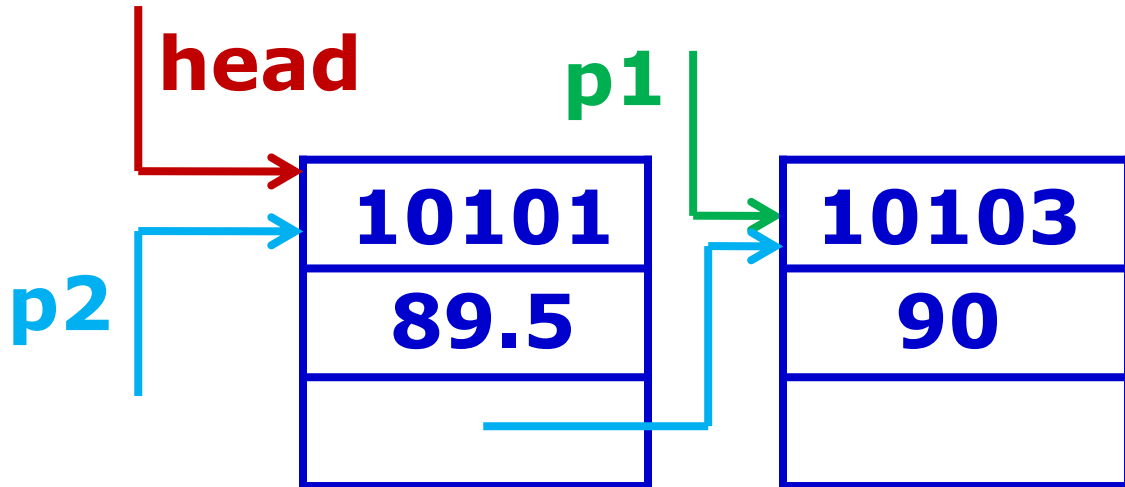


## ➤ 解题思路:

- ◆ 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

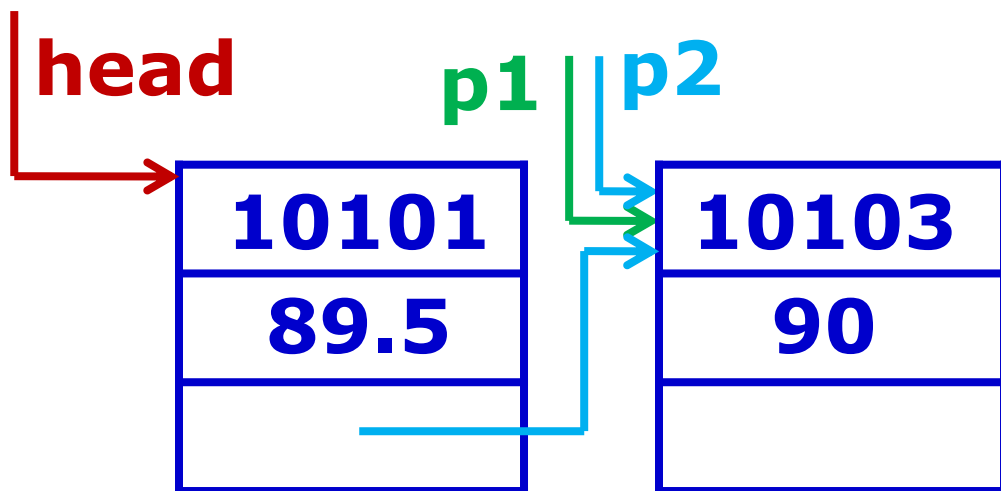


## ➤ 解题思路:

- ◆ 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;**

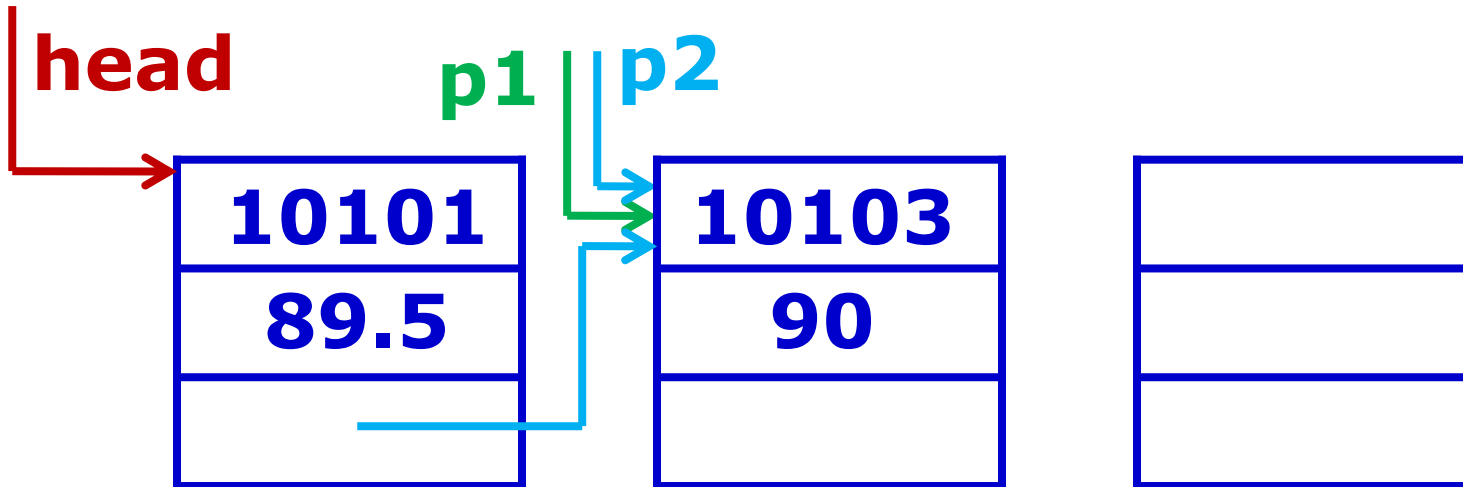
- ◆ 使**p2**指向刚才建立的结点 **p2=p1;**





## ➤ 解题思路:

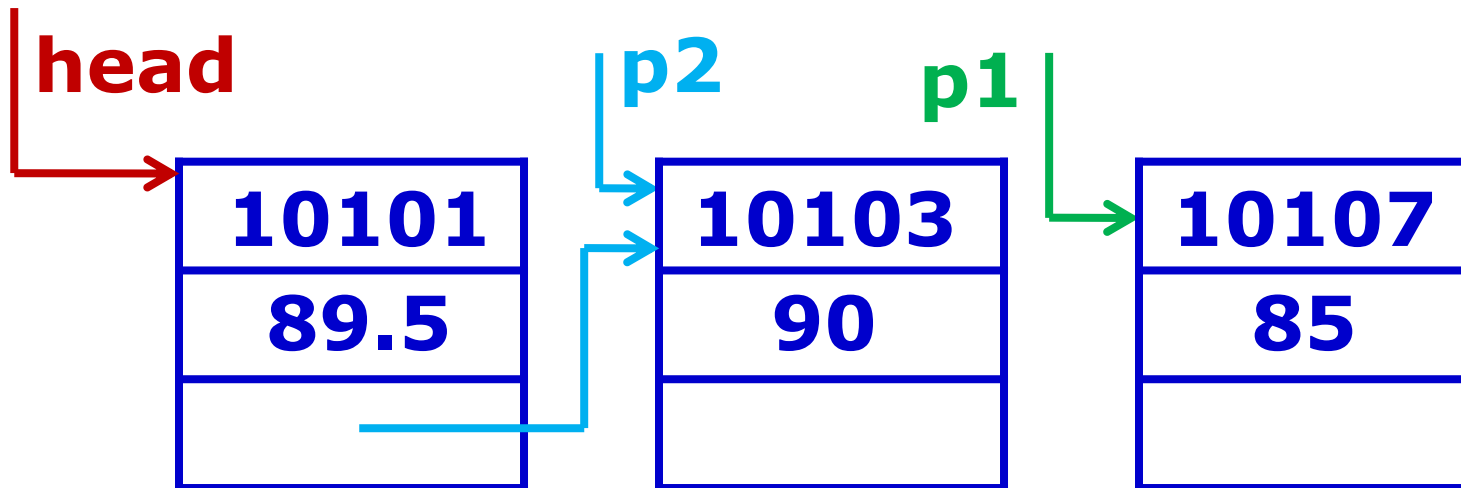
- ◆ 再开辟另一个结点并使**p1**指向它，接着输入该结点的数据



## ➤ 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);
```

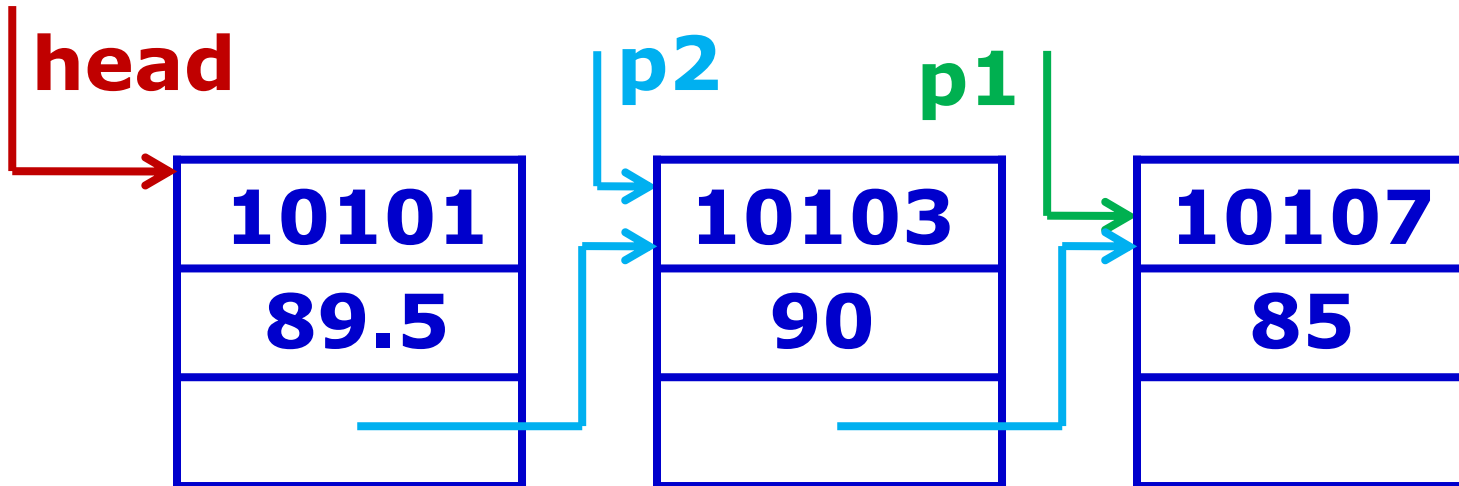


## ➤ 解题思路:

- ◆ 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

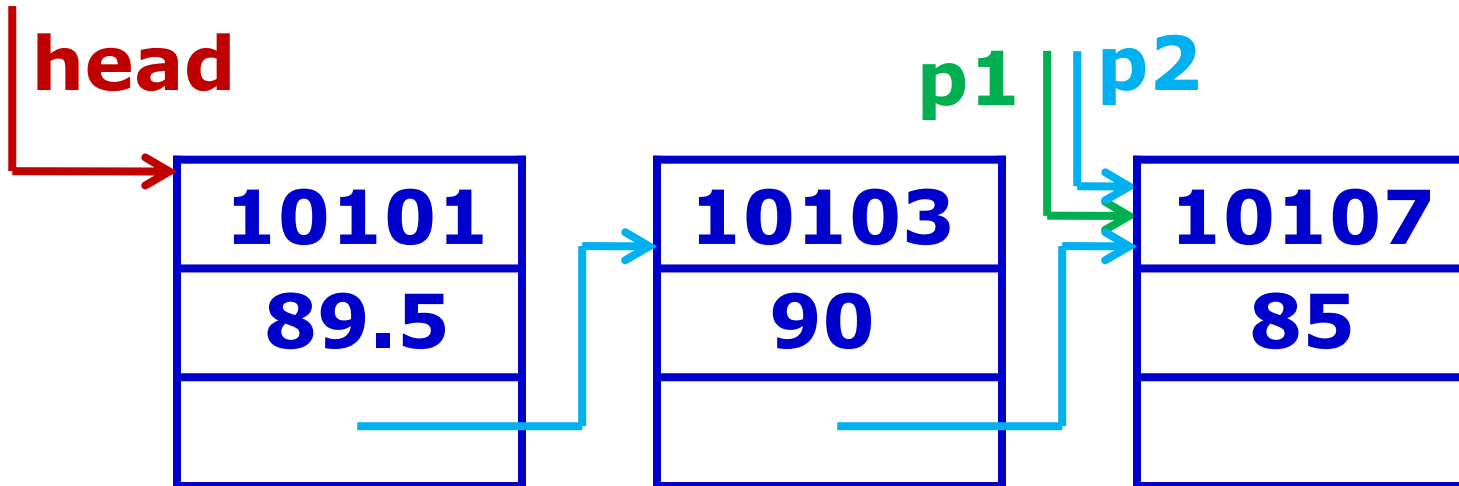


## ➤ 解题思路:

- ◆ 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

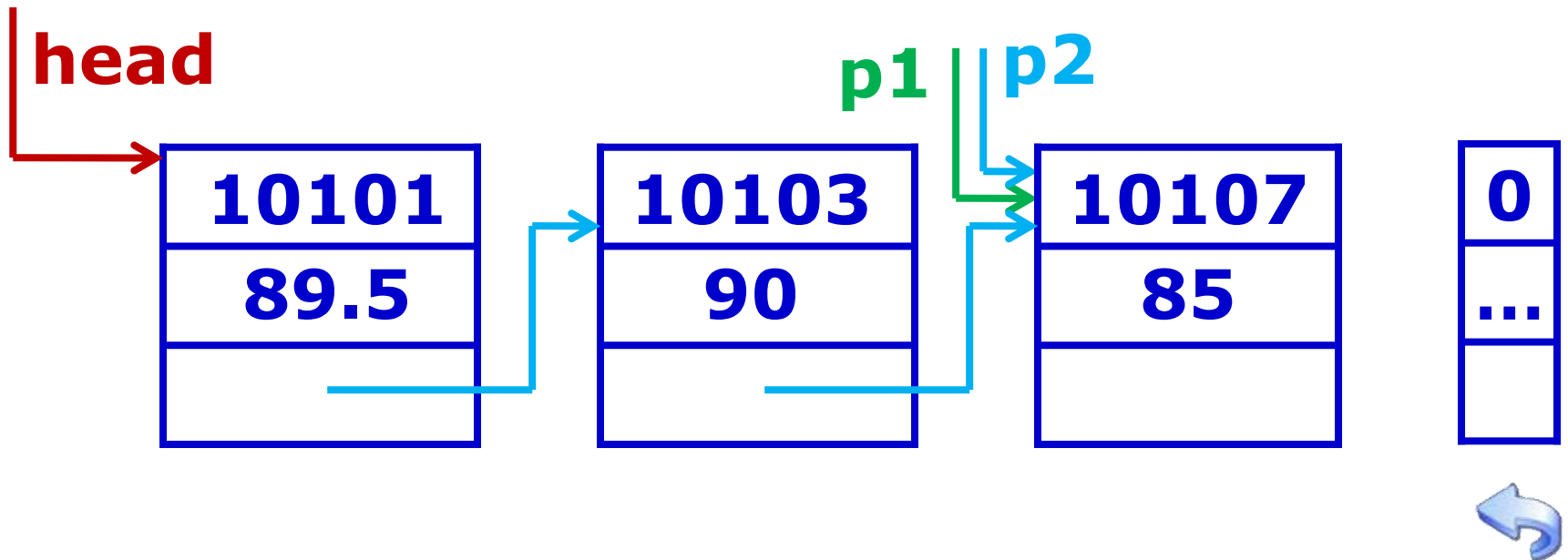
**p2->next=p1;**

- ◆ 使**p2**指向刚才建立的结点 **p2=p1;**



## ➤ 解题思路:

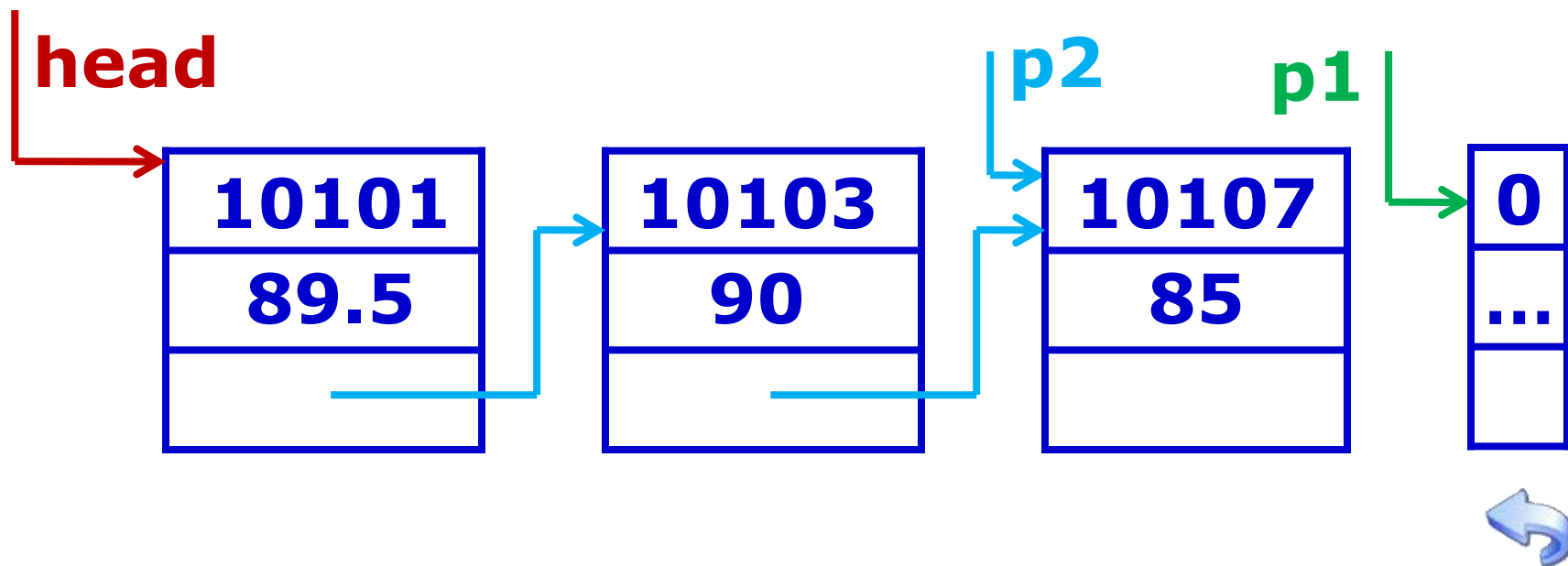
- ◆ 再开辟另一个结点并使**p1**指向它, 接着输入该结点的数据



## ➤ 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

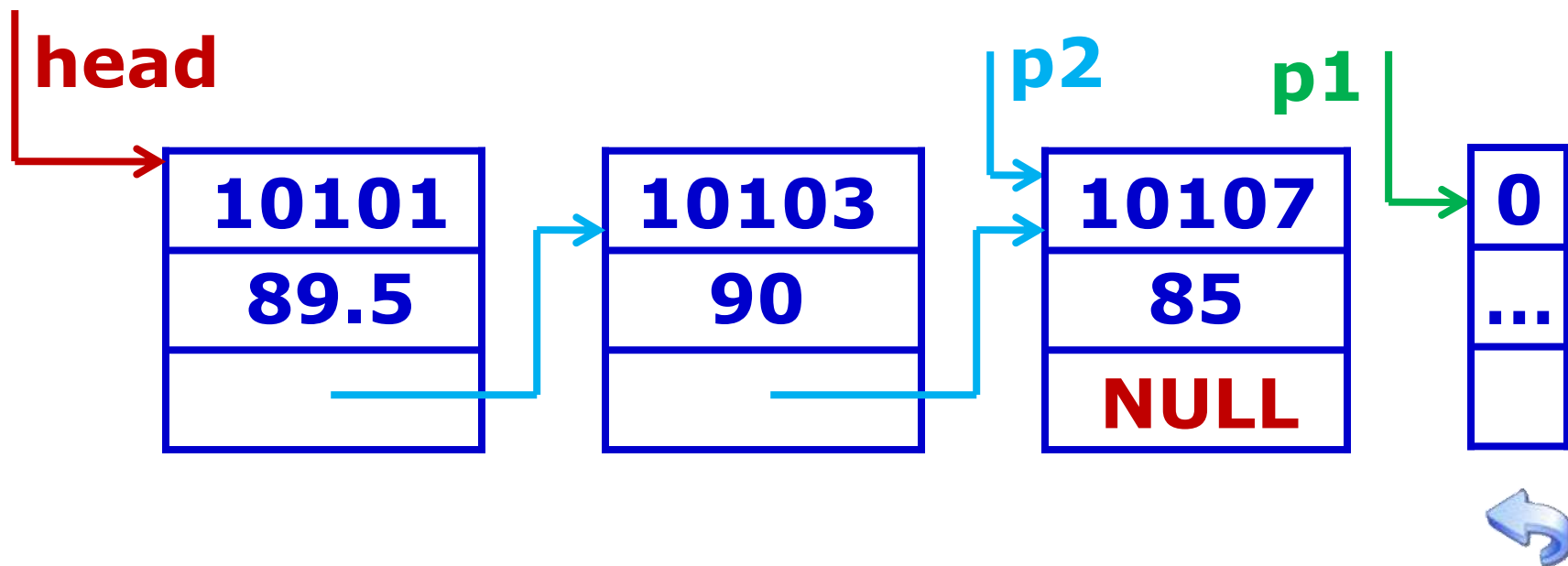
```
p1=(struct Student*)malloc(LEN);  
scanf("%ld,%f",&p1->num,&p1->score);
```



## ➤ 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 <stdio.h>
#include <stdlib.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

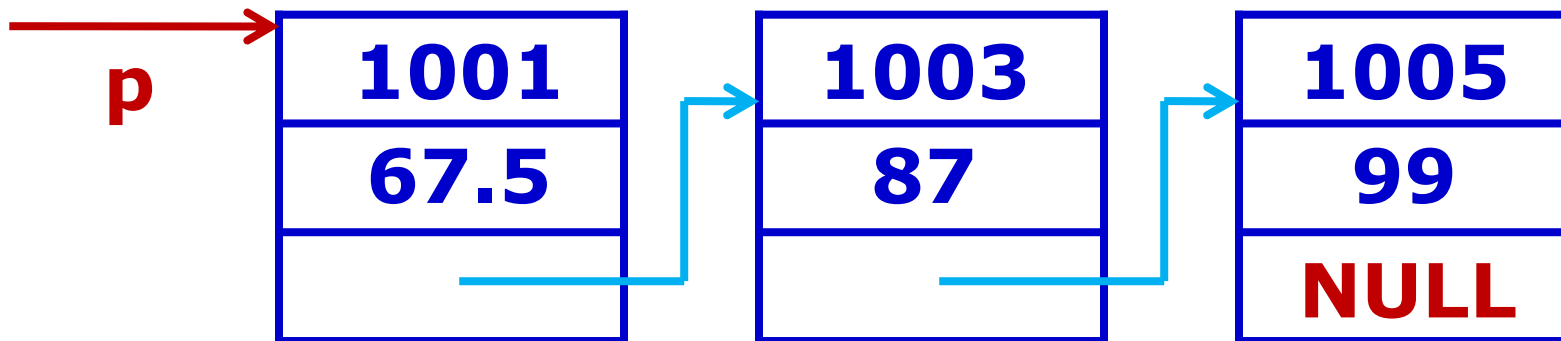


```
int main()  
{ struct Student *pt;  
  pt=creat();  
  printf("\nnum:%ld\nscore:%5.1f\n",  
        pt->num,pt->score);  
  return 0;  
}
```



## 9.4.4 输出链表

例**9.10** 编写一个输出链表的函数print。



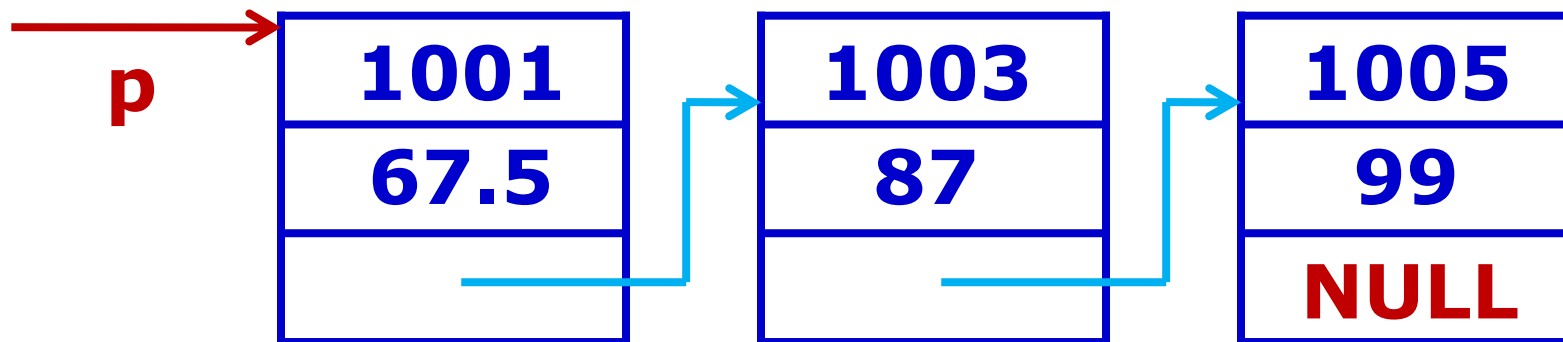
1001 67.5

➤ 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

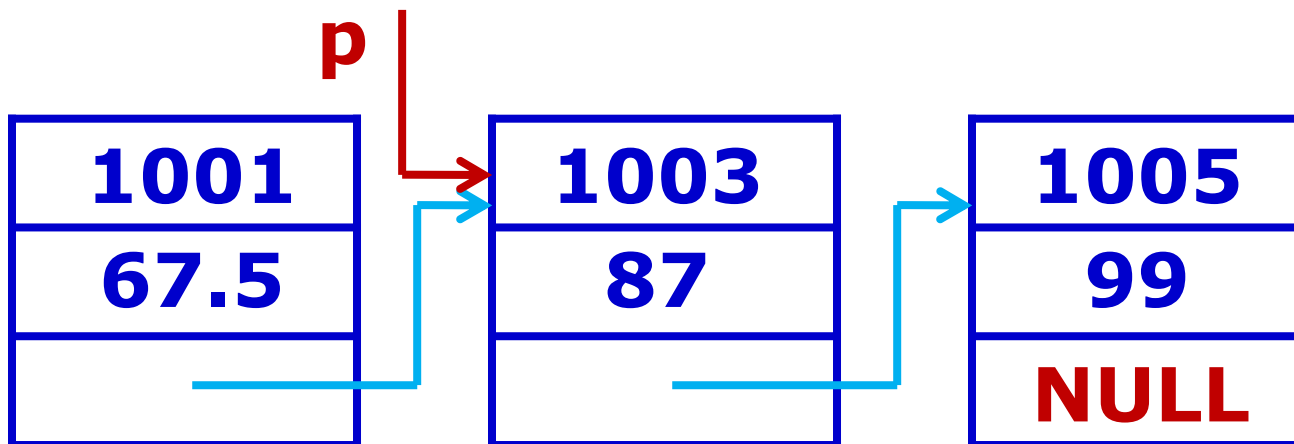
➤ 解题思路:

◆ 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;
```



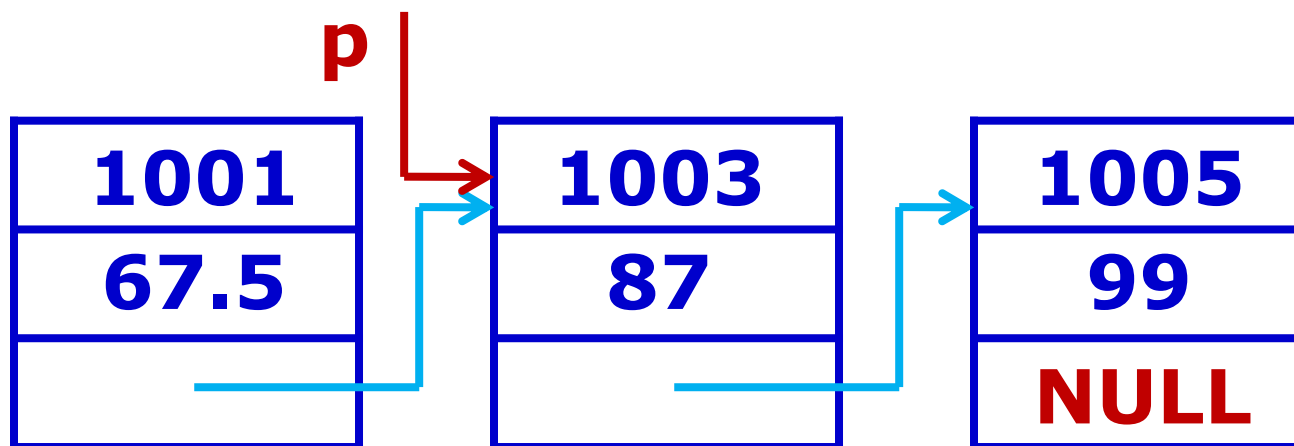
1001	67.5
1003	87.0

➤ 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

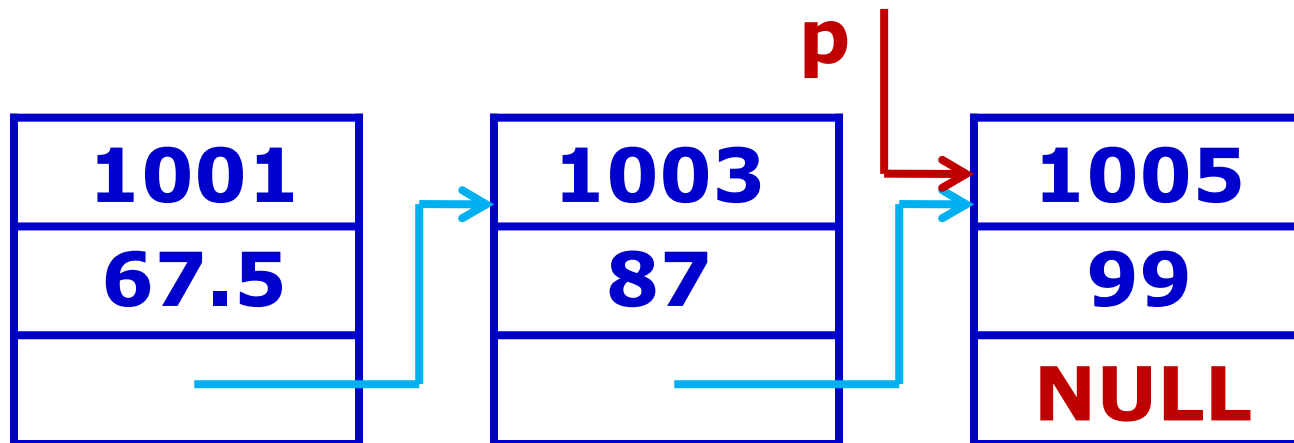
➤ 解题思路:

◆ 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;
```



➤ 解题思路:

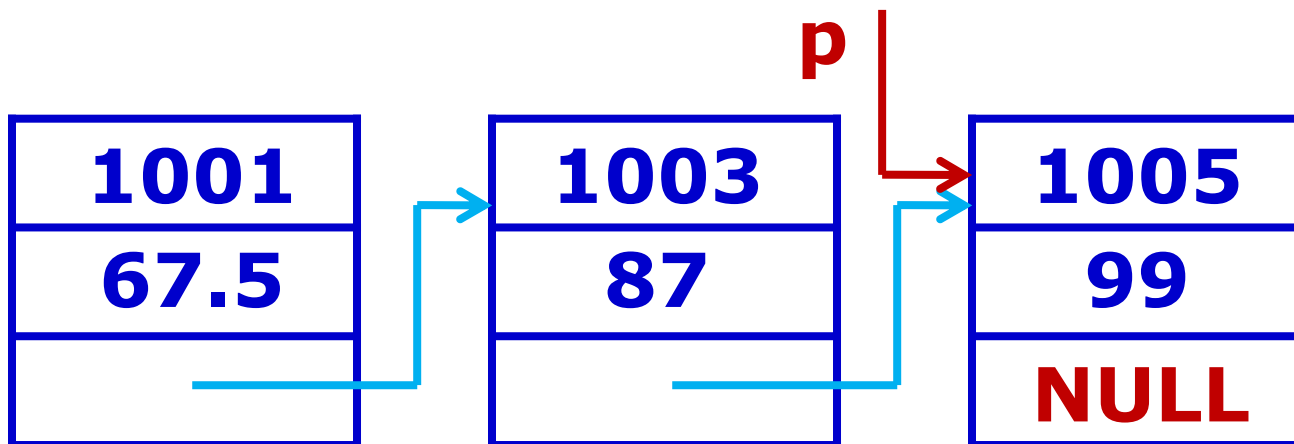
◆ 输出 **p** 所指的 **新** 结点

```
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;**



1001	67.5
1003	87.0
1005	99.0



```
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);  
}
```



给定一个链表，链表包含一个整形数，实现一个函数search，查找值为val的节点并删除，如果查找不到则添加到链表开始。



## **9.5 共用体类型**

**9.5.1 什么是共用体类型**

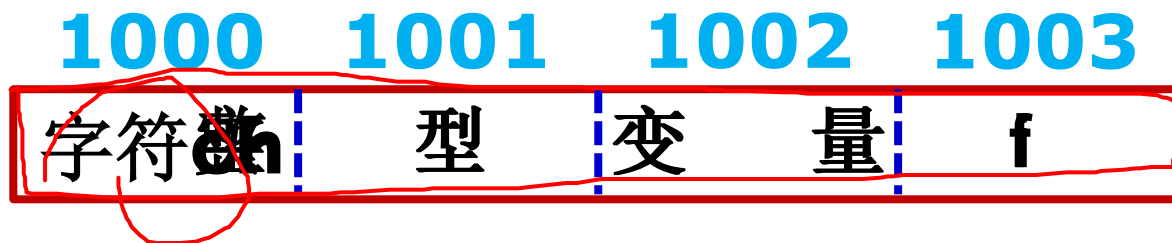
**9.5.2 引用共用体变量的方式**

**9.5.3 共用体类型数据的特点**



## 9.5.1 什么是共用体类型

- 有时想用同一段内存单元存放不同类型的变量。
- 使几个不同的变量共享同一段内存的结构，称为“共用体”类型的结构。



➤ 定义共用体类型变量的一般形式为：

union 共用体名

{ 成员表列

}变量表列;

例如：

union Data

{ int i;

char ch;

float f;

}a,b,c;

union Data

{ int i;

char ch;

float f;

};

union Data a,b,c;



- “共用体”与“结构体”的定义形式相似，但它们的含义是不同的。
- 结构体变量所占内存长度是各成员占的内存长度之和，每个成员分别占有其自己的内存单元。而共用体变量所占的内存长度等于最长的成员的长度。



## 9.5.2 引用共用体变量的方式

- 只有先定义了共用体变量才能引用它，但应注意，不能引用共用体变量，而只能引用共用体变量中的成员。

例如，前面定义了**a,b,c**为共用体变量，下面的引用方式是正确的：

**a.i**

**a.ch**

**a.f**



## 9.5.3 共用体类型数据的特点

➤ 在使用共用体类型数据时要注意以下一些特点：

**(1)** 同一个内存段可以用来存放几种不同类型的成员，但在每一瞬时只能存放其中一个成员，而不是同时存放几个。





## 9.5.3 共用体类型数据的特点

➤ 在使用共用体类型数据时要注意以下一些特点：

**(2)** 可以对共用体变量初始化，但初始化表中只能有一个常量。



## 9.5.3 共用体类型数据的特点

➤ 在使用共用体类型数据时要注意以下一些特点：

**(3)** 共用体变量中起作用的成员是最后一次被赋值的成员，在对共用体变量中的一个成员赋值后，原有变量存储单元中的值就取代。



## 9.5.3 共用体类型数据的特点

➤ 在使用共用体类型数据时要注意以下一些特点：

**(4)** 共用体变量的地址和它的各成员的地址都是同一地址。

**(5)** 不能对共用体变量名赋值，也不能企图引用变量名来得到一个值。



## 9.5.3 共用体类型数据的特点

➤ 在使用共用体类型数据时要注意以下一些特点：

**(6)** 以前的**C**规定不能把共用体变量作为函数参数，但可以使用指向共用体变量的指针作函数参数。**C99**允许用共用体变量作为函数参数。



## 9.5.3 共用体类型数据的特点

➤ 在使用共用体类型数据时要注意以下一些特点：

**(7)** 共用体类型可以出现在结构体类型定义中，也可以定义共用体数组。反之，结构体也可以出现在共用体类型定义中，数组也可以作为共用体的成员。



## 9.5.3 共用体类型数据的特点

**例9.11** 有若干个人的数据，其中有学生和教师。学生的数据中包括：姓名、号码、性别、职业、班级。教师的数据包括：姓名、号码、性别、职业、职务。要求用同一个表格来处理。



➤ 解题思路:

◆ 学生和教师的数据项目多数是相同的，但有一项不同。现要求把它们放在同一表格中

num	name	sex	job	class(班) position(职务)
101	Li	f	s	501
102	Wang	m	t	prof



➤ 解题思路:

◆ 如果**job**项为**s**，则第 5 项为**class**。即**Li**是**501**班的。如果**job**项是**t**，则第 5 项为**position**。**Wang**是**prof**（教授）。

num	name	sex	job	class(班) position(职务)
101	Li	f	s	501
102	Wang	m	t	prof





➤ 解题思路:

◆ 对第5项可以用共用体来处理（将**class**和**position**放在同一段存储单元中）

num	name	sex	job	class(班) position(职务)
101	Li	f	s	501
102	Wang	m	t	prof



```
#include <stdio.h>
```

```
struct
```

```
{ int num;
```

```
  char name[10];
```

```
  char sex;
```

```
  char job;
```

```
  union
```

```
  { int class;
```

```
    char position[10];
```

```
  }category;
```

```
}person[2];
```

共用体变量

外部的结构体数组



```
#include <stdio.h>
```

```
union Categ
```

```
{ int clas;
```

```
    char position[10];
```

```
};
```

```
struct
```

```
{ int num;
```

```
    char name[10];
```

```
    char sex;
```

```
    char job;
```

```
    union Categ category
```

```
}person[2];
```

声明共用体类型

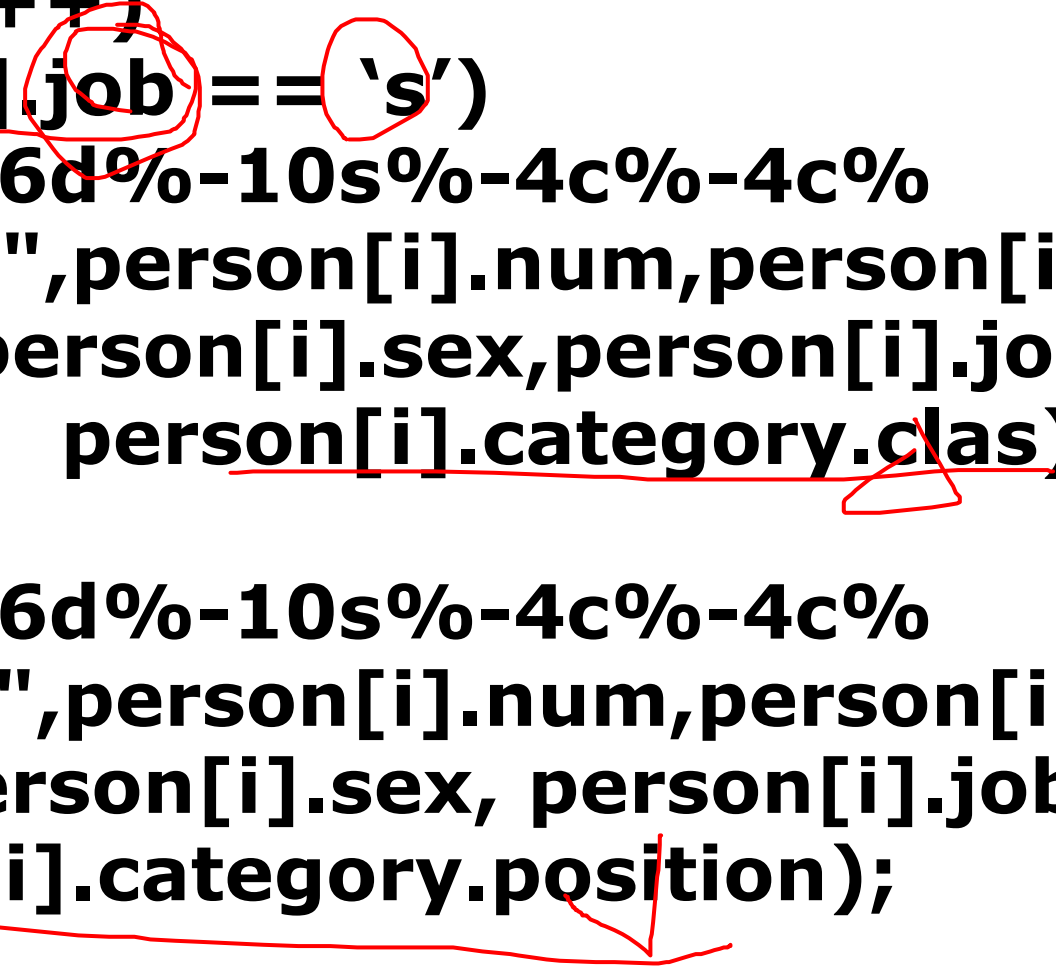
定义共用体类型变量



```
int main()
{int i;
for(i=0;i<2;i++)
{scanf("%d %s %c %c",&person[i].num,
        &person[i].name,
        &person[i].sex,&person[i].job);
if(person[i].job == 's')
    scanf("%d",&person[i].category.clas);
else if(person[i].job == 't')
    scanf("%s",person[i].category.position);
else printf("Input error!");
}
printf("\n");
```



```
for(i=0;i<2;i++)
{if (person[i].job == 's')
    printf("%-6d%-10s%-4c%-4c%-10d\n",person[i].num,person[i].name,person[i].sex,person[i].job,person[i].category.clas);
else
    printf("%-6d%-10s%-4c%-4c%-10s\n",person[i].num,person[i].name,person[i].sex, person[i].job,person[i].category.position);
}
return 0;
}
```



## 9.6 使用枚举类型

- 如果一个变量只有几种可能的值，则可以定义为枚举类型
- 所谓“**枚举**”就是指把可能的值一一列举出来，变量的值只限于列举出来的值的范围内



## 9.6 使用枚举类型

➤ 声明枚举类型用**enum**开头。

枚举元素

➤ 例如：

```
enum Weekday{sun,mon,tue,  
              wed,thu,fri,sat};
```

◆ 声明了一个枚举类型**enum**

枚举变量

◆ 然后可以用此类型来定义变量

```
enum Weekday  weekday,weekend;
```



## 9.6 使用枚举类型

**workday=mon;**

正确

**weekend=sun;**

正确

**weekday=monday;**

不正确





➤说明:

**(1) C编译对枚举类型的枚举元素按常量处理，故称枚举常量。不要因为它们是标识符(有名字)而把它们看作变量，不能对它们赋值。例如:**

sun=0; mon=1; 错误



➤说明:

**(2)** 每一个枚举元素都代表一个整数, C 语言编译按定义时的顺序默认它们的值为 **0,1,2,3,4,5...**

◆在上面定义中, **sun**的值为**0**, **mon**的值为**1,...sat**的值为**6**

◆如果有赋值语句: **workday=mon;**  
相当于**workday=1;**



➤说明:

**(2)** 每一个枚举元素都代表一个整数，C语言编译按定义时的顺序默认它们的值为**0,1,2,3,4,5...**

◆也可以人为地指定枚举元素的数值，例如：

```
enum Weekday{sun=7,mon=1,tue,  
wed,thu,fri,sat}workday,week_end;
```

◆指定枚举常量**sun**的值为**7**，**mon**为**1**，以后顺序加**1**，**sat**为**6**。



➤说明:

**(3)** 枚举元素可以用来作判断比较。例如:

**if(workday==mon)...**

**if(workday>sun)...**

◆枚举元素的比较规则是按其在初始化时指定的整数来进行比较的。

◆如果定义时未人为指定，则按上面的默认规则处理，即第一个枚举元素的值为 0，故

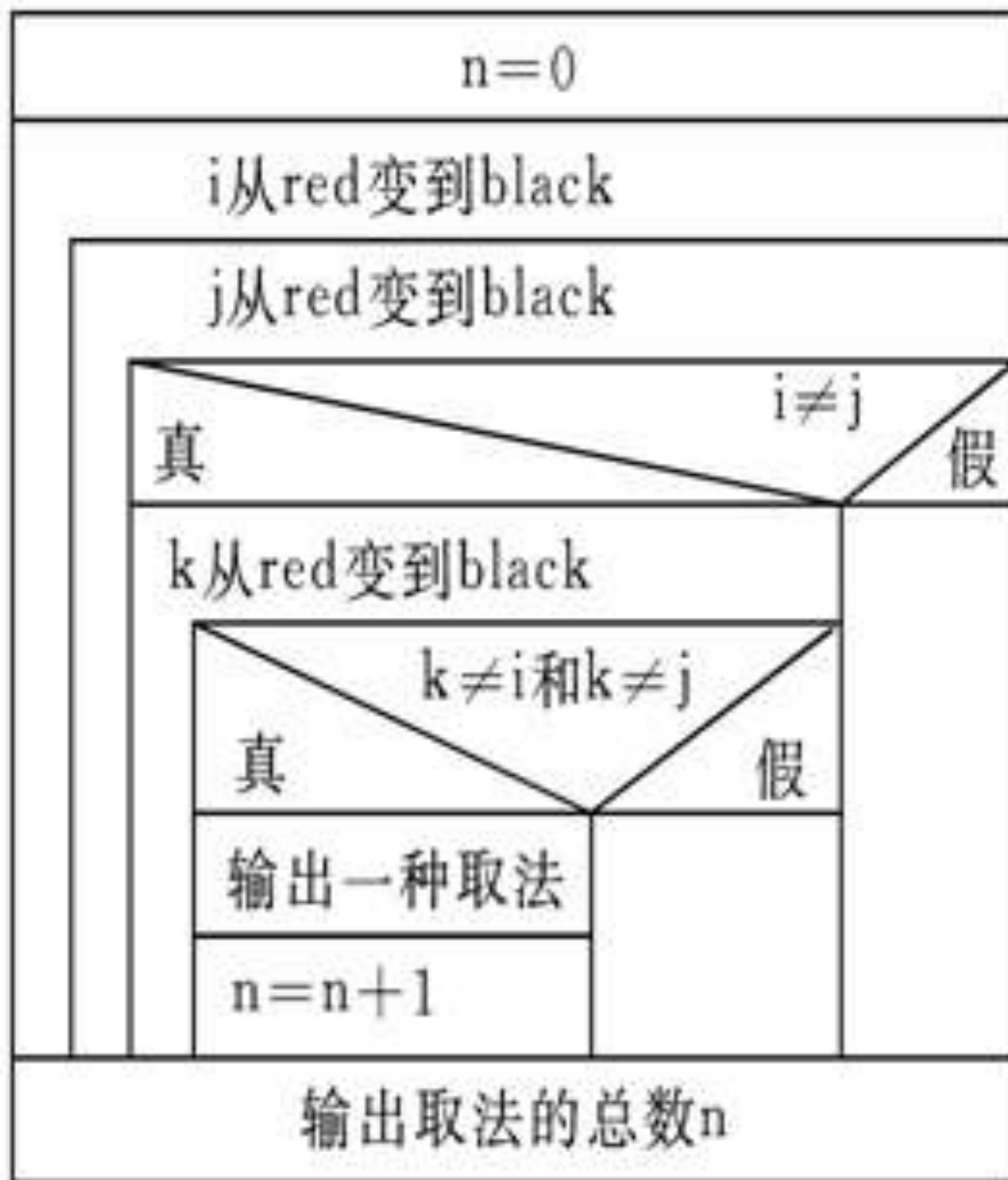
**mon>sun, sat>fri**



**例9.12** 口袋中有~~红、黄、蓝、白、黑~~**5**种颜色的球若干个。每次从口袋中先后取出**3**个球，问得到**3**种不同颜色的球的可能取法，输出每种排列的情况。



➤ 解题思路:



## ► 解题思路:

loop由1到3				
loop的值				
1	2	3		
$i \Rightarrow \text{pri}$	$j \Rightarrow \text{pri}$	$k \Rightarrow \text{pri}$		
pri的值				
red	yellow	blue	white	black
输出 "red"	输出 "yellow"	输出 "blue"	输出 "white"	输出 "black"



```
#include <stdio.h>  
int main()  
{enum Color{red,yellow,blue,white,black};  
enum Color i,j,k,pri;  
int n,loop;  
n=0;  
for (i=red;i<=black;i++)  
for (j=red;j<=black;j++)  
if (i!=j)  
{ for (k=red;k<=black;k++)  
if ((k!=i) && (k!=j))  
{ n=n+1;  
printf("%0-4d",n);
```





```
for (loop=1;loop<=3;loop++)  
{switch (loop)  
  { case 1: pri=i;break;  
    case 2: pri=j;break;  
    case 3: pri=k;break;  
    default:break;  
  }  
}
```



```
switch (pri)
{case red: printf("%-10s","red");
        break;
 case yellow:printf("%-10s","yellow");
        break;
 case blue: printf("%-10s","blue");
        break;
 case white: printf("%-10s","white");
        break;
 case black: printf("%-10s","black");
        break;
}
```



```
    }  
    printf("\n");  
}  
}  
printf("\ntotal:%5d\n",n);  
return 0;  
}
```



## 9.7 用typedef声明新类型名

1. 简单地用一个新的类型名代替原有的类型名

```
typedef int Integer;
```

```
typedef float Real;
```

```
int i,j; float a,b; 与
```

```
Integer i,j; Real a,b;
```

等价



## 9.7 用typedef声明新类型名

### 2.命名一个简单的类型名代替复杂的类型表示方法

(1)命名一个新的类型名代表结构体类型:

```
typedef struct
```

```
{ int month; int day; int year; }Date;
```

```
Date birthday;
```

```
Date *p;
```



## 9.7 用typedef声明新类型名

2.命名一个简单的类型名代替复杂的类型表示方法

(2) 命名一个新的类型名代表数组类型

```
typedef int Num[100];
```

```
Num a;
```



## 9.7 用typedef声明新类型名

2.命名一个简单的类型名代替复杂的类型表示方法

(3)命名一个新的类型名代表一个指针类型

```
typedef char *String;
```

```
String p,s[10];
```



## 9.7 用typedef声明新类型名

### 2.命名一个简单的类型名代替复杂的类型表示方法

(4)命名一个新的类型名代表指向函数的指针类型

```
typedef int (*Pointer)();  
Pointer p1,p2;
```





## 9.7 用typedef声明新类型名

➤ 归纳起来，声明一个新的类型名的方法是

① 先按定义变量的方法写出定义体 (**int i;**)

② 将变量名换成新类型名 (将**i**换成**Count**)

③ 在最前面加**typedef**

(**typedef int Count**)

④ 用新类型名去定义变量



## 9.7 用typedef声明新类型名

➤ 以定义上述的数组类型为例来说明：

① 先按定义数组变量形式书写：**int a[100];**

② 将变量名**a**换成自己命名的类型名：**int  
Num[100];**

③ 在前面加上**typedef**，得到**typedef int  
Num[100];**

用来定义变量：**Num a;**

相当于定义了：**int a[100];**



## 9.7 用typedef声明新类型名

➤ 对字符指针类型，也是：

```
char *p;
```

```
char *String;
```

```
typedef char *String;
```

```
String p;
```



## 9.7 用typedef声明新类型名

➤说明:

(1) 以上的方法实际上是为特定的类型指定了一个同义字(**synonyms**)。例如

① **typedef int Num[100];**

**Num a;**      **Num**是**int [100]**的同义词

② **typedef int (\*Pointer)();**

**Pointer p1;** **Pointer**是**int (\*)()**的同义词



## 9.7 用typedef声明新类型名

➤说明:

**(2)** 用**typedef**只是对已经存在的类型指定一个新的类型名，而没有创造新的类型。

**(3)**用**typedef**声明数组类型、指针类型，结构体类型、共用体类型、枚举类型等，使得编程更加方便。

**(4)****typedef**与**#define**表面上有相似之处



## 9.7 用typedef声明新类型名

➤说明:

(5) 当不同源文件中用到同一类型数据时，常用**typedef**声明一些数据类型。可以把所有的**typedef**名称声明单独放在一个头文件中，然后在需要用到它们的文件中用**#include**指令把它们包含到文件中。这样编程者就不需要在各文件中自己定义**typedef**名称了。



## 9.7 用**typedef**声明新类型名

➤说明：

**(6)** 使用**typedef**名称有利于程序的通用与移植。有时程序会依赖于硬件特性，用**typedef**类型就便于移植。



- 给定一个链表，链表包含一个整形数，且此链表已经按照整数从小到大进行排序，通过链表的指针操作将链表调整成从大到小的顺序（不借助数组，不增加新的节点）。（**20**分）

```
struct SortedNum  
{  
    int Num;  
    SortedNum* pNext;  
};  
//尾指针为nullptr
```