

第 9 章

用户自己建立数据类型

定义和使用结构体变量

自己建立结构体类型

struct 结构体名
{成员表列};

C语言允许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**（structre）。

在程序中建立一个结构体类型：

num	name	sex	age	score	addr
10010	Li Fang	M	18	87.5	Beijing

```
struct Student
{
    int num;           //学号为整型
    char name[20];     //姓名为字符串
    char sex;          //性别为字符型
    int age;           //年龄为整型
    float score;       //成绩为实型
    char addr[30];     //地址为字符串
};                    //注意最后有一个分号
```

结构体类型的名字是由一个关键字**struct**和结构体名组合而成的。结构体名由用户指定，又称“结构体标记”(structure tag)。

花括号内是该结构体所包括的子项，称为结构体的成员(member)。对各成员都应进行类型声明，即 **类型名 成员名;**

“成员表列”(member list)也称为“域表”(field list)，每一个成员是结构体中的一个域。成员名命名规则与变量名相同。

自己建立结构体类型

- (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，各自包含不同的成员。
- (2) 成员可以属于另一个结构体类型。

num	name	sex	age	birthday			addr
				month	day	year	

```
struct Date //声明一个结构体类型 struct Date
{
    int month; //月
    int day; //日
    int year; //年
};
```

```
struct Student //声明一个结构体类型 struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
    struct Date birthday; //成员birthday属于struct Date类型
    char addr[30];
};
```

定义结构体类型变量

1. 先声明结构体类型，再定义该类型的变量

```
struct Student
{
    int num;           //学号为整型
    char name[20];     //姓名为字符串
    char sex;          //性别为字符型
    int age;           //年龄为整型
    float score;       //成绩为实型
    char addr[30];     //地址为字符串
};                    //注意最后有一个分号
```

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

结构体类型名 结构体变量名

student1:	10001	Zhang Xin	M	19	90.5	Shanghai
student2:	10002	Wang Li	F	20	98	Beijing

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;
```

struct 结构体名
{ 成员表列
}变量名表列;

3. 不指定类型名而直接定义结构体类型变量

struct
{ 成员表列
}变量名表列;

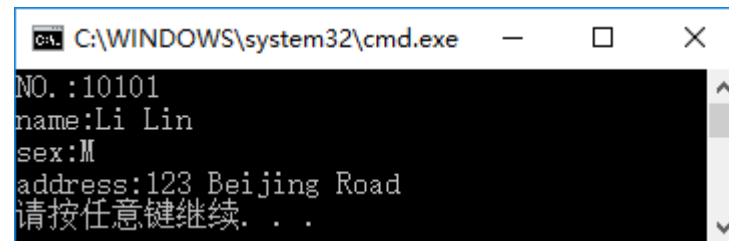
定义结构体类型变量

- (1) 结构体类型与结构体变量是不同的概念，不要混淆。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。
 - (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。
-

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。

```
#include <stdio.h>
int main()
{
    struct Student                //声明结构体类型struct Student
    {
        long int num;            //以下4行为结构体的成员
        char name[20];
        char sex;
        char addr[20];
    }a={10101,"Li Lin",'M',"123 Beijing Road"}; //定义结构体变量a并初始化
    printf("NO.:%ld\nname:%s\nsex:%c\naddress:%s\n",a.num,a.name,a.sex,a.addr);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
NO.:10101
name:Li Lin
sex:M
address:123 Beijing Road
请按任意键继续...
```

结构体变量的初始化和引用

- (1) 在定义结构体变量时可以对它的成员初始化。初始化列表是用花括号括起来的一些常量，这些常量依次赋给结构体变量中的各成员。

注意

对结构体变量初始化，不是对结构体类型初始化

- (2) 可以引用结构体变量中成员的值，引用方式为 **结构体变量名.成员名**

```
student1.num=10010;
```

/*已定义了student1为student类型的结构体变量，则student1.num表示student1变量中的num成员，即student1的num(学号)成员*/

“.”是成员运算符，它在所有的运算符中优先级最高，因此可以把student1.num作为一个整体来看待，相当于一个变量。

```
printf("%s\n",student1);
```

//企图用结构体变量名输出所有成员的值



不能企图通过输出结构体变量名来达到输出结构体变量所有成员的值。只能对结构体变量中的各个成员分别进行输入和输出。

结构体变量的初始化和引用

- (3) 如果成员本身又属一个结构体类型，则要用若干个成员运算符，一级一级地找到最低的一级的成员。只能对最低级的成员进行赋值或存取以及运算。

```
student1.num=10010;           //结构体变量student1中的成员num  
student1.birthday.month=6;    //结构体变量student1中的成员birthday中的成员month
```

- (4) 对结构体变量的成员可以像普通变量一样进行各种运算（根据其类型决定可以进行的运算）。

```
student2.score = student1.score;    //赋值运算  
sum=student1.score+student2.score;  //加法运算  
student1.age++;                     //自加运算
```

- (5) 同类的结构体变量可以互相赋值。

```
student1=student2;                //假设student1和student2已定义为同类型的结构体变量
```

- (6) 可以引用结构体变量成员的地址，也可以引用结构体变量的地址(结构体变量的地址主要用作函数参数，传递结构体变量的地址)。但不能用以下语句整体读入结构体变量。

```
scanf("%d",&student1.num);        //输入student1.num的值  
printf("%o",&student1);           //输出结构体变量student1的起始地址
```

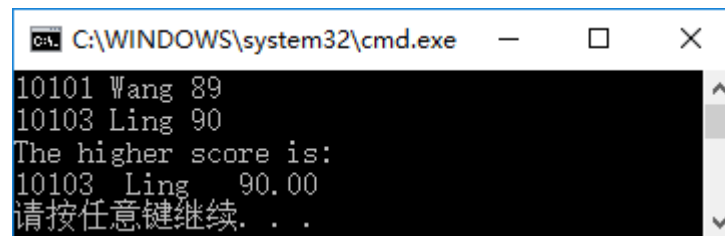
```
scanf("%d,%s,%c,%d,%f,%s\n",&student1);
```



结构体变量的初始化和引用

【例9.2】输入两个学生的学号、姓名和成绩，输出成绩较高的学生的学号、姓名和成绩。

```
#include <stdio.h>
int main()
{
    struct Student          //声明结构体类型struct Student
    {
        int num;
        char name[20];
        float score;
    }student1,student2;      //定义两个结构体变量student1,student2
    scanf("%d%s%f",&student1.num,student1.name,&student1.score); //输入学生1的数据
    scanf("%d%s%f",&student2.num,student2.name,&student2.score); //输入学生1的数据
    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;
}
```



```
C:\WINDOWS\system32\cmd.exe
10101 Wang 89
10103 Ling 90
The higher score is:
10103 Ling 90.00
请按任意键继续. . .
```

使用结构体数组

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。

name	count
Li	0
Zhang	0
Sun	0

```
C:\WINDOWS\system32\cmd.exe
Li
Li
Sun
Zhang
Zhang
Sun
Li
Sun
Zhang
Li

Result:
  Li:4
  Zhang:3
  Sun:3
请按任意键继续. . .
```

```
#include <string.h>
#include <stdio.h>
struct Person //声明结构体类型struct Person
{   char name[20]; //候选人姓名
    int count; //候选人得票数
}leader[3]={"Li",0,"Zhang",0,"Sun",0}; //定义结构体数组并初始化

int main()
{   int i,j;
    char leader_name[20]; //定义字符数组
    for(i=1;i<=10;i++)
    {   scanf("%s",leader_name); //输入所选的候选人姓名
        for(j=0;j<3;j++)
            if(strcmp(leader_name,leader[j].name)==0) leader[j].count++;
    }
    printf("\nResult:\n");
    for(i=0;i<3;i++)
        printf("%5s:%d\n",leader[i].name,leader[i].count);
    return 0;
}
```

定义结构体数组

(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];**

```
struct Person
{   char name[20];
    int count;
} leader[3];
```

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];

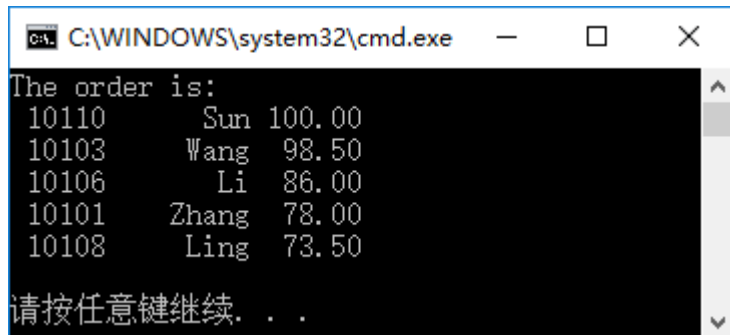
```
struct Person
{   char name[20];
    int count;
};
struct Person leader[3]; //leader是结构体数组名
```

(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};**

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

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。



```
C:\WINDOWS\system32\cmd.exe
The order is:
10110    Sun 100.00
10103    Wang 98.50
10106    Li 86.00
10101    Zhang 78.00
10108    Ling 73.50
请按任意键继续. . .
```


```
#include <stdio.h>
struct Student //声明结构体类型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},{10110,"Sun",100}; //定义结构体数组并初始化
    struct Student temp; //定义结构体变量temp, 用作交换时的临时变量
    const int n=5; //定义常量n
    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) //进行成绩的比较
                k=j;
        temp=stu[k]; stu[k]=stu[i]; stu[i]=temp; //stu[k]和stu[i]元素互换
    }
    for(i=0;i<n;i++)
        printf("%6d %8s %6.2f\n",stu[i].num,stu[i].name,stu[i].score);
    printf("\n");
    return 0;
}
```

结构体指针



结构体指针

所谓结构体指针就是指向结构体变量的指针，一个结构体变量的起始地址就是这个结构体变量的指针。如果把一个结构体变量的起始地址存放在一个指针变量中，那么，这个指针变量就指向该结构体变量。

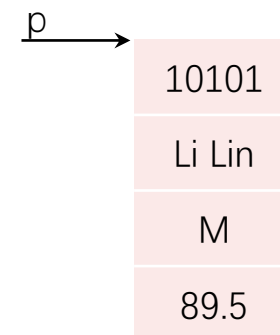


指向结构体变量的指针

【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。

```
#include <stdio.h>
#include <string.h>
int main()
{
    struct Student          //声明结构体类型struct Student
    {
        long num;
        char name[20];
        char sex;
        float score;
    };
    struct Student stu_1;    //定义struct Student类型的变量stu_1
    struct Student *p;       //定义指向struct Student 类型数据的指针变量p
    p=&stu_1;               //p指向stu_1
    stu_1.num=10101;         //对结构体变量的成员赋值
    strcpy(stu_1.name,"Li Lin"); //用字符串复制函数给stu_1.name赋值
    stu_1.sex='M';
    stu_1.score=89.5;
    printf("No.:%ld\nname:%s\nsex:%c\nscore:%5.1f\n",stu_1.num,stu_1.name,stu_1.sex,stu_1.score); //输出结果
    printf("\nNo.:%ld\nname:%s\nsex:%c\nscore:%5.1f\n",(*p).num,(*p).name,(*p).sex, (*p).score);
    return 0;
}
```

(*p).num也可表示为p->num



如果 `p` 指向一个结构体变量 `stu`，以下3种用法等价：

① `stu.成员名`

`stu.num`

② `(*p).成员名`

`(*p).num`

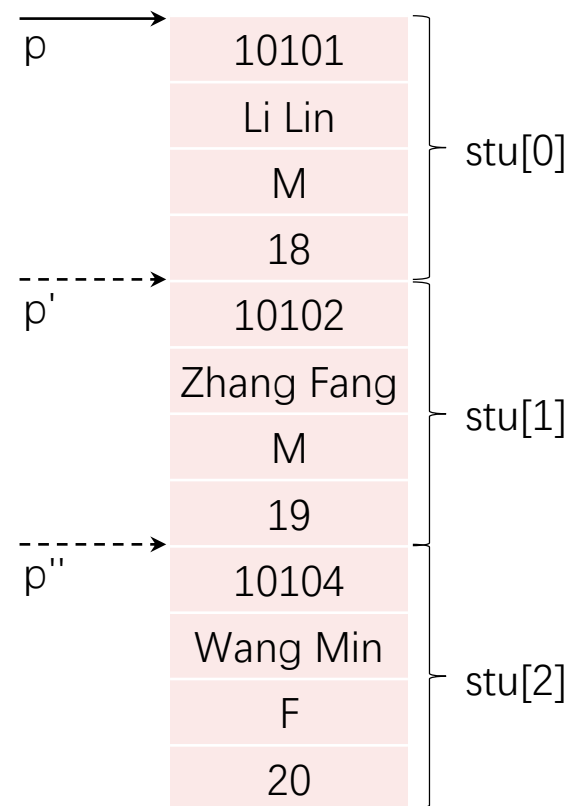
③ `p->成员名`

`p->num`

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。

```
#include <stdio.h>
struct Student          //声明结构体类型struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
};
struct Student stu[3]={{10101,"Li Lin",'M',18},{10102,"Zhang Fang",'M',19},{10104,"Wang Min",'F',20}};
//定义结构体数组并初始化
int main()
{
    struct Student *p;    //定义指向struct Student结构体变量的指针变量
    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);    //输出结果
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
No. Name      sex age
10101 Li Lin   M  18
10102 Zhang Fang M  19
10104 Wang Min F  20
请按任意键继续. . .
```

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，采取的也是“值传递”的方式，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，如果结构体的规模很大时，开销是很可观的。此外，由于采用值传递方式，如果在执行被调用函数期间改变了形参（也是结构体变量）的值，该值不能返回主调函数，这往往造成使用上的不便。因此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。

用结构体变量和结构体变量的指针作函数参数

【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)。

```
C:\WINDOWS\system32\cmd.exe
请输入各学生的信息： 学号、姓名、三门课成绩：
10101 Li 78 89 98
10103 Wang 98.5 87 69
10106 Sun 88 76.5 89

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
请按任意键继续...
```

```
#include <stdio.h>
#define N 3          //学生数为3
struct Student      //建立结构体类型struct Student
{
    int num;         //学号
    char name[20];   //姓名
    float score[3];  //3门课成绩
    float aver;      //平均成绩
};
int main()
{
    void input(struct Student stu[]); //函数声明
    struct Student max(struct Student stu[]); //函数声明
    void print(struct Student stu); //函数声明
    struct Student stu[N],*p=stu; //定义结构体数组和指针
    input(p); //调用input函数
    print(max(p)); //调用print函数,以max函数的返回值作为实参
    return 0;
}
void input(struct Student stu[]) //定义input函数
{
    int i;
    printf("请输入各学生的信息： 学号、姓名、三门课成绩:\n");
    for(i=0;i<N;i++)
```

```

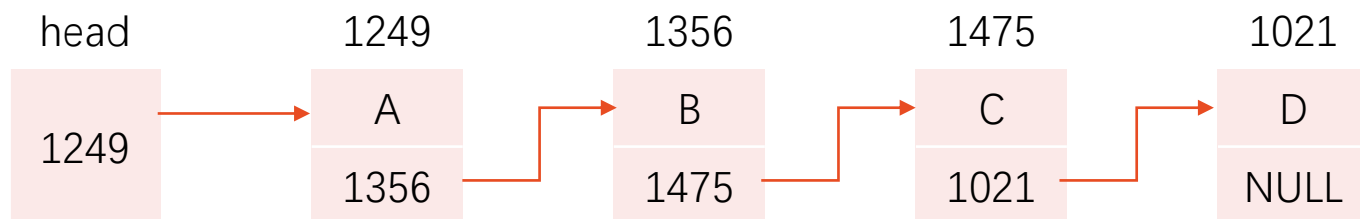
    {
        scanf("%d %s %f %f %f",&stu[i].num,stu[i].name,
            &stu[i].score[0],&stu[i].score[1],&stu[i].score[2]); //输入数据
        stu[i].aver=(stu[i].score[0]+stu[i].score[1]+stu[i].score[2])/3.0;
        //求平均成绩
    }
}
struct Student max(struct Student stu[]) //定义max函数
{
    int i,m=0; //用m存放成绩最高的学生在数组中的序号
    for(i=0;i<N;i++)
        if(stu[i].aver>stu[m].aver) m=i;
    //找出平均成绩最高的学生在数组中的序号
    return stu[m]; //返回包含该生信息的结构体元素
}

void print(struct Student stud) //定义print函数
{
    printf("\n成绩最高的学生是:\n");
    printf("学号:%d\n姓名:%s\n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],st
    ud.aver);
}
```

链表

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。



链表有一个“**头指针**”变量，图中以head表示，它存放一个地址，该地址指向一个元素。

链表中每一个元素称为“**结点**”，每个结点都应包括两个部分：

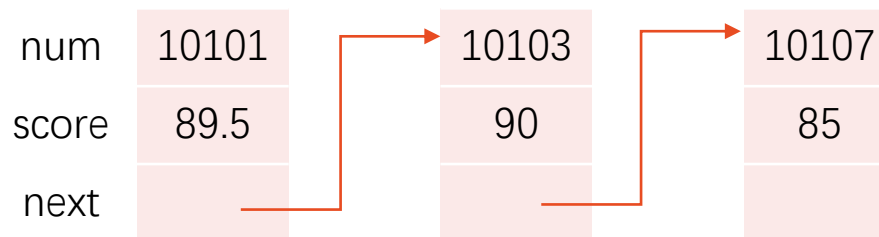
- (1) 用户需要用的实际数据；
- (2) 下一个结点的地址。

head指向第1个元素，第1个元素又指向第2个元素……直到最后一个元素，该元素不再指向其他元素，它称为“**表尾**”，它的地址部分放一个“**NULL**”（表示“空地址”），链表到此结束。

什么是链表

可以用结构体变量建立链表。一个结构体变量包含若干成员，这些成员可以是数值类型、字符类型、数组类型，也可以是指针类型。用指针类型成员来存放下一个结点的地址。

```
struct Student
{
    int num;
    float score;
    struct Student *next;    //next是指针变量，指向结构体变量
};
```



成员num和score用来存放结点中的有用数据（用户需要用到的数据）。

next是指针类型的成员，它指向struct Student类型数据（就是next所在的结构体类型）

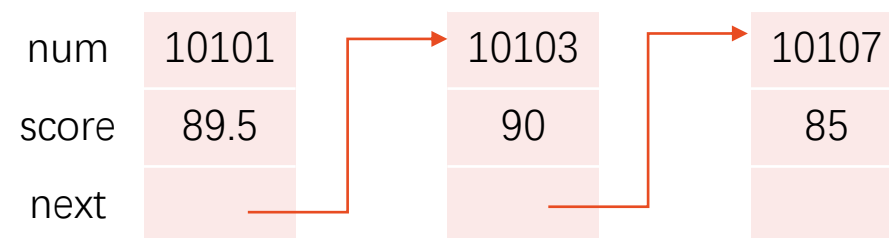
注意

- 上面只是定义了一个struct Student类型，并未实际分配存储空间，只有定义了变量才分配存储单元。

建立简单的静态链表

【例9.8】建立一个简单链表，它由3个学生数据的结点组成，要求输出各结点中的数据。

```
#include <stdio.h>
struct Student          //声明结构体类型struct Student
{
    int num;
    float score;
    struct Student*next;
};
int main()
{
    struct Student a,b,c,*head,*p;    //定义3个结构体变量a,b,c作为链表的结点
    a.num=10101; a.score=89.5;         //对结点a的num和score成员赋值
    b.num=10103; b.score=90;          //对结点b的num和score成员赋值
    c.num=10107; c.score=85;          //对结点c的num和score成员赋值
    head=&a;                          //将结点a的起始地址赋给头指针head
    a.next=&b;                         //将结点b的起始地址赋给a结点的next成员
    b.next=&c;                         //将结点c的起始地址赋给a结点的next成员
    c.next=NULL;                     //c结点的next成员不存放其他结点地址
    p=head;                          //使p指向a结点
    do
    {
        printf("%ld %5.1f\n",p->num,p->score); //输出p指向的结点的数据
        p=p->next;                          //使p指向下一结点
    }while(p!=NULL);                      //输出完c结点后p的值为NULL，循环终止
    return 0;
}
```

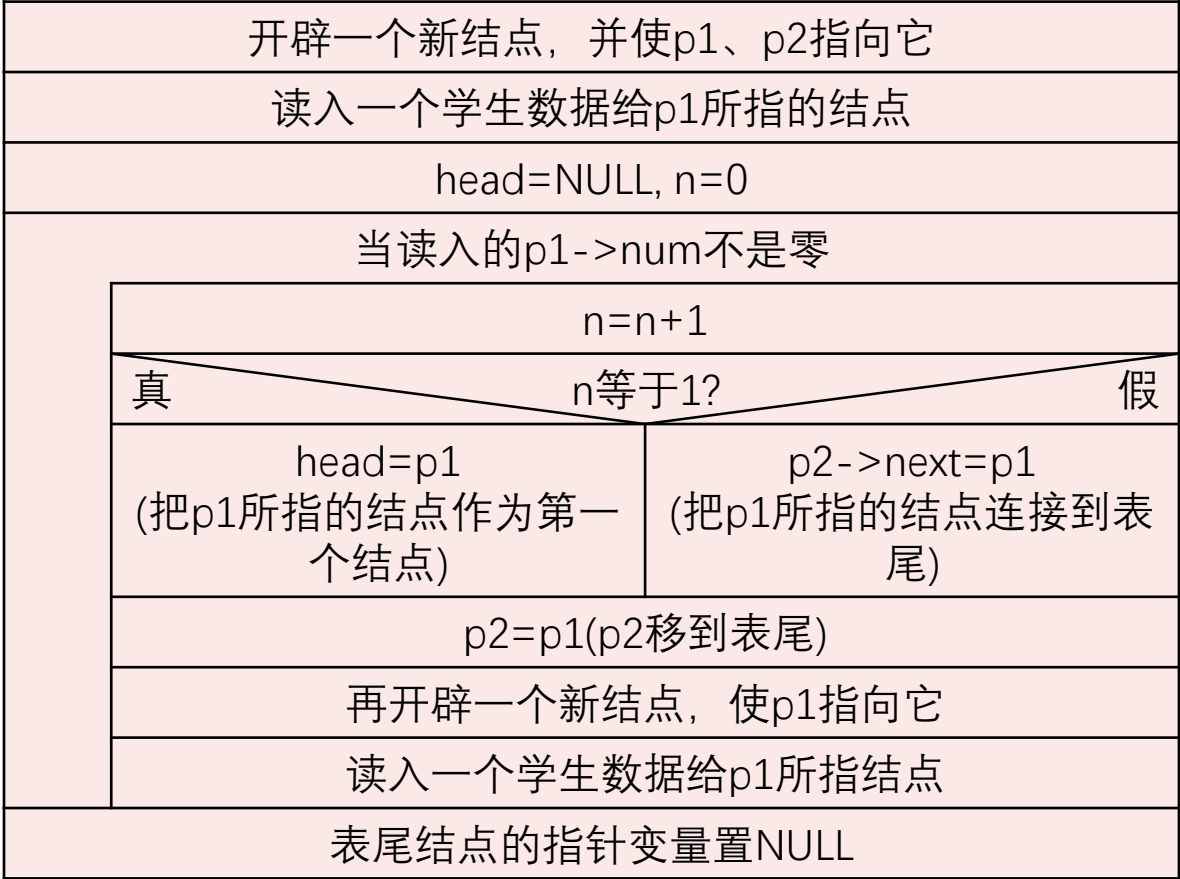


```
C:\WINDOWS\system32\cmd.exe
10101 89.5
10103 90.0
10107 85.0
请按任意键继续. . .
```


建立简单的动态链表

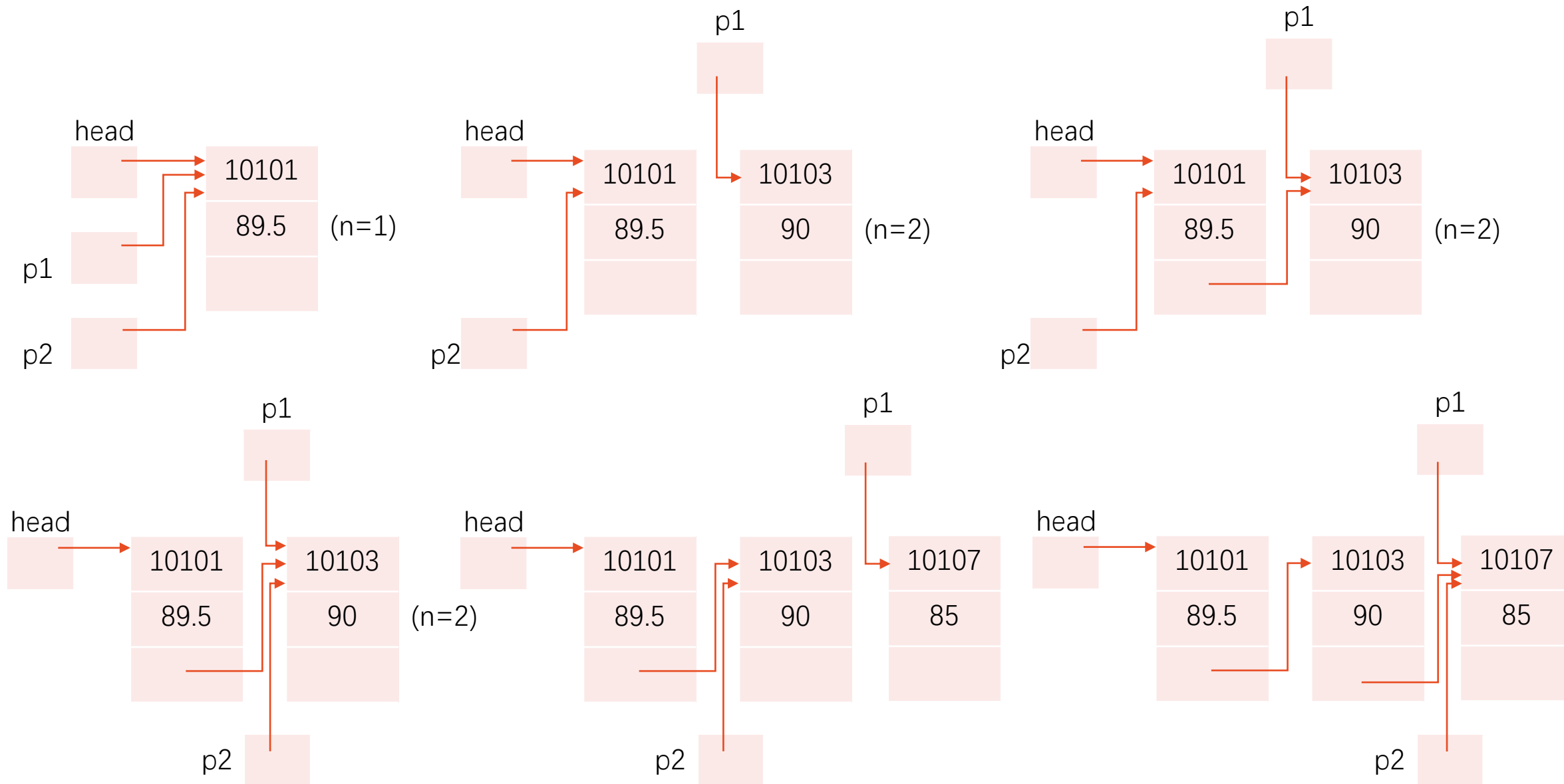
【例9.9】 写一函数建立一个有3名学生数据的单向动态链表。

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。



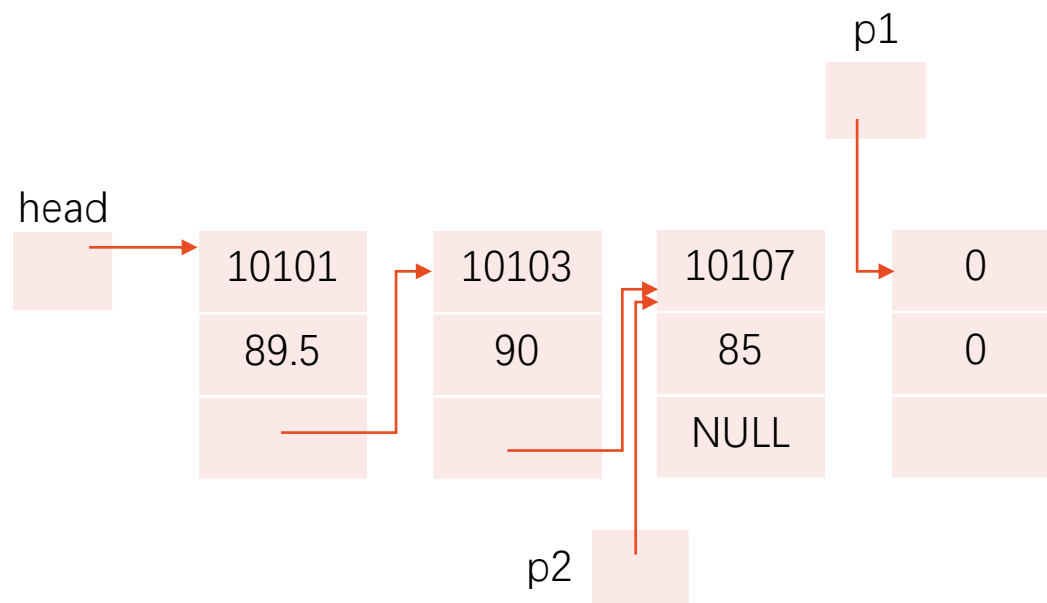
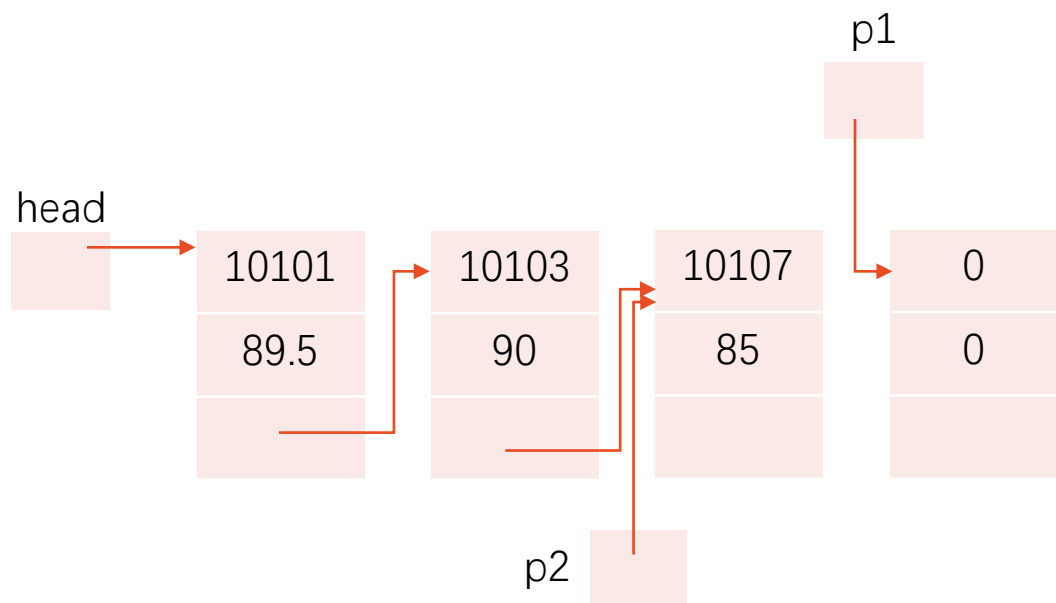
建立简单的动态链表

【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。



建立简单的动态链表

【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。



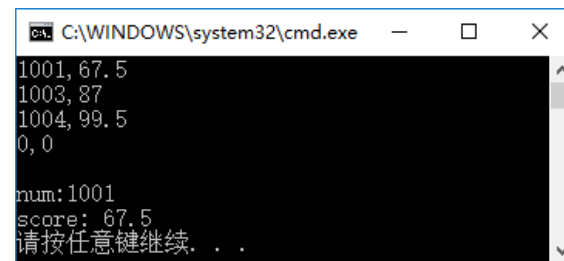
建立简单的动态链表

【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。

```
#include <stdio.h>
#include <stdlib.h>
#define LEN sizeof(struct Student)
struct Student
{
    long num;
    float score;
    struct Student*next;
};
int n;    //n为全局变量，本文件模块中各函数均可使用它
struct Student *creat(void)
//定义函数。此函数返回一个指向链表头的指针
{
    struct Student *head;
    struct Student *p1,*p2;
    n=0;
    p1=p2=(struct Student*) malloc(LEN); //开辟一个新单元
    scanf("%ld,%f",&p1->num,&p1->score);
    //输入第1个学生的学号和成绩
    head=NULL;
    while(p1->num!=0)
```

```

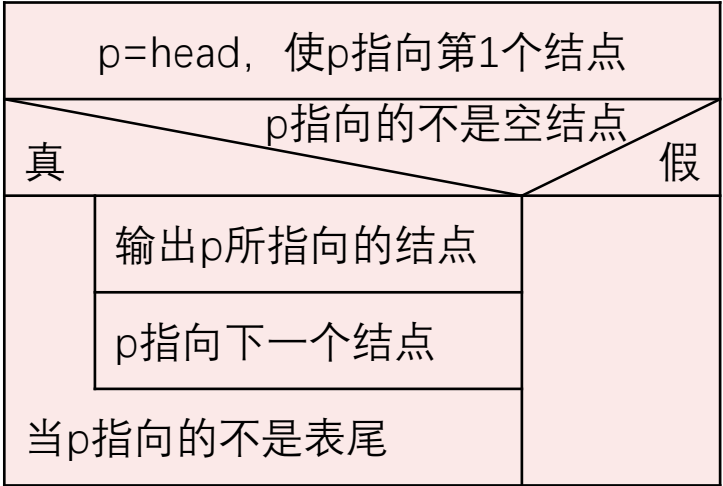
    {
        n=n+1;
        if(n==1) head=p1;
        else p2->next=p1;
        p2=p1;
        p1=(struct Student*)malloc(LEN);
        //开辟动态存储区，把起始地址赋给p1
        scanf("%ld,%f",&p1->num,&p1->score);
        //输入其他学生的学号和成绩
    }
    p2->next=NULL;
    return(head);
}
int main()
{
    struct Student *pt;
    pt=creat();    //函数返回链表第一个结点的地址
    printf("\nnum:%ld\nscore:%5.1f\n",pt->num,pt->score);
    //输出第1个结点的成员值
    return 0;
};
```



```
C:\WINDOWS\system32\cmd.exe
1001, 67.5
1003, 87
1004, 99.5
0, 0
num: 1001
score: 67.5
请按任意键继续. . .
```

输出链表【例9.10】编写一个输出链表的函数print。

```
#include <stdio.h>
#include <stdlib.h>
#define LEN sizeof(struct Student)
struct Student          //声明结构体类型struct Student
{
    long num;
    float score;
    struct Student *next;
};
int n;                  //全局变量n
void print(struct Student*head) //定义print函数
{
    struct Student*p;    //在函数中定义struct Student类型的变量p
    printf("\nNow,These %d records are:\n",n);
    p=head;              //使p指向第1个结点
    if(head!=NULL)        //若不是空表
        do
        {
            printf("%ld %5.1f\n",p->num,p->score); //输出一个结点中的学号与成绩
            p=p->next; //p指向下一个结点
        }while(p!=NULL); //当p不是"空地址"
}
```



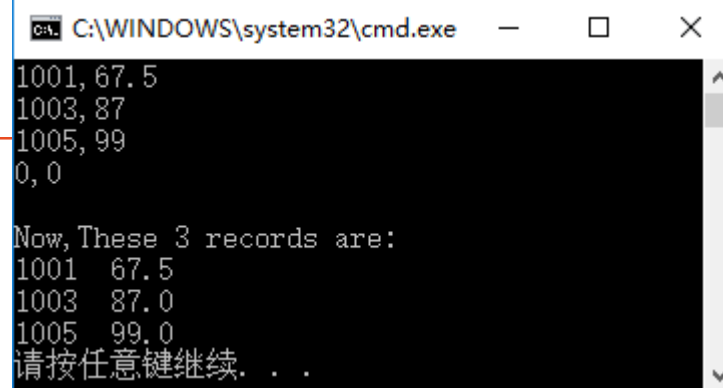
组合【例9.7】和【例9.9】

```
#include <stdio.h>
#include <malloc.h>
#define LEN sizeof(struct Student)
struct Student
{
    long num;
    float score;
    struct Student *next;
};
int n;
struct Student *creat() //建立链表的函数
{
    struct Student *head;
    struct Student *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);
}

void print(struct Student *head) //输出链表的函数
{
    struct Student *p;
    printf("\nNow,These %d records are:\n",n);
    p=head;
    if(head!=NULL)
        do
        {
            printf("%ld %5.1f\n",p->num,p->score);
            p=p->next;
        }while(p!=NULL);
}

int main()
{
    struct Student *head;
    head=creat(); //调用creat函数, 返回第1个结点的起始地址
    print(head); //调用print函数
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
1001, 67.5
1003, 87
1005, 99
0, 0
Now, These 3 records are:
1001  67.5
1003  87.0
1005  99.0
请按任意键继续. . .
```

***共用体类型**

什么是共用体类型

使几个不同类型的变量共享同一段内存的结构，称为“共用体”类型的结构。

union共用体名
{ 成员表列
}变量表列;

1000地址

短整型变量i

字符型变量ch

实型变量f

```
union Data
{
    int i;
    //表示不同类型的变量i,ch,f可以存放到同一段存储单元中
    char ch;
    float f;
}a,b,c; //在声明类型同时定义变量
```

```
union Data //声明共用体类型
{
    int i;
    char ch;
    float f;
};
union Data a,b,c; //用共用体类型定义变量
```

```
union //没有定义共用体类型名
{
    int i;
    char ch;
    float f;
}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。几个成员共用一个内存区。

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。

```
a.i    //引用共用体变量中的整型变量i  
a.ch   //引用共用体变量中的字符变量ch  
a.f    //引用共用体变量中的实型变量f
```

```
printf("%d",a);
```



```
printf("%d",a.i);
```



共用体类型数据的特点

(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。

(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。

```
union Data a={1,'a',1.5};
```



(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被取代。

(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。C 99允许同类型的共用体变量互相赋值。

(6) C 99允许用共用体变量作为函数参数。

```
b=a; //a和b是同类型的共用体变量，合法
```

(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。

```
union Date
{
    int i;
    char ch;
    float f;
}a;
a.i=97;
printf("%d",a.i); //输出整数97
printf("%c",a.ch); //输出字符'a'
printf("%f",a.f); //输出实数0.000000
```

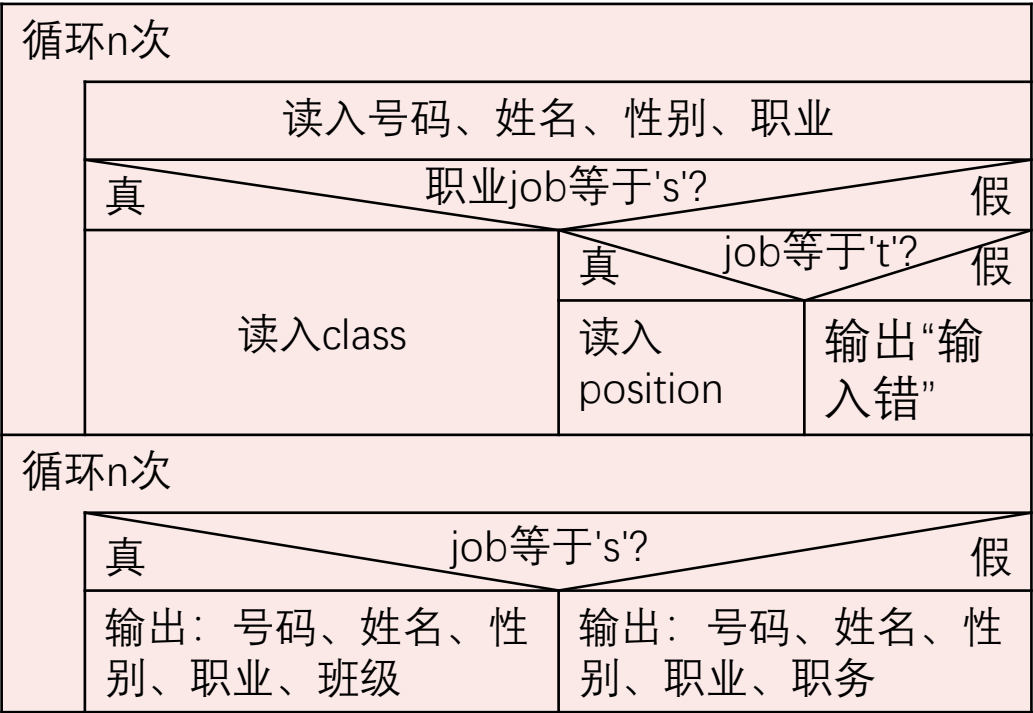
```
a=1; //不能对共用体变量赋值，赋给谁？
m=a; //企图引用共用体变量名以得到一个
      值赋给整型变量m
```



共用体类型数据的特点

【例9.10】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。

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



共用体类型数据的特点

【例9.10】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。

```
#include <stdio.h>
struct                                //声明无名结构体类型
{   int num;                          //成员num(编号)
    char name[10];                    //成员name(姓名)
    char sex;                         //成员sex(性别)
    char job;                         //成员job(职业)
    union                             //声明无名共用体类型
    {   int clas;                     //成员clas(班级)
        char position[10];           //成员position(职务)
    }category;                       //成员category是共用体变量
}person[2];                           //定义结构体数组person，有两个元素
```

```

int main()
{
    int i;
    for(i=0;i<2;i++)
    {
        printf("please enter the data of person:\n");
        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");
    printf("No.namesex job class/position\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;
}

```

```

C:\WINDOWS\system32\cmd.exe
please enter the data of person:
101 Li f s 501
please enter the data of person:
102 Wang m t prof

No.namesex job class/position
101 Li f s 501
102 Wang m t prof
请按任意键继续. . .

```

//输入前4项

//如是学生，输入班级

//如是教师，输入职务

//如job不是's'和't'，显示“输入错误”

//若是学生

//若是教师

使用枚举类型

使用枚举类型

```
enum [枚举名]{枚举元素列表}
```

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

如果一个变量只有几种可能的值，则可以定义为**枚举**(enumeration)**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。

声明枚举类型用enum开头。花括号中的sun,mon,...,sat称为**枚举元素**或**枚举常量**。

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

枚举类型

枚举变量

也可以不声明有名字的枚举类型，而直接定义枚举变量：

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

- (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。
- (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…。也可以在定义枚举类型时显式地指定枚举元素的数值。
- (3) 枚举元素可以用来作判断比较。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。

使用枚举类型

【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。

```
#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)//如果二球不同色
            {
                for(k=red;k<=black;k++)
                    if((k!=i) && (k!=j))
                    {
                        n=n+1;
                        printf("%-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)//根据球的颜色输出相应的文字
                            {
                                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;
                                default:break;
                            }
                        }
                        printf("\n");
                    }
            }

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

//声明枚举类型enum Color
//定义枚举变量i,j,k,pri

//外循环使i的值从red变到black
//中循环使j的值从red变到black

//内循环使k的值从red变到black
//如果3球不同色

//符合条件的次数加1

//输出当前是第几个符合条件的排列

//先后对3个球分别处理

//loop的值从1变到3

//loop的值为1时，把第i个球的颜色输出

//loop的值为2时，把第j个球的颜色输出

//loop的值为3时，把第k个球的颜色输出

//pri=i

//pri=j

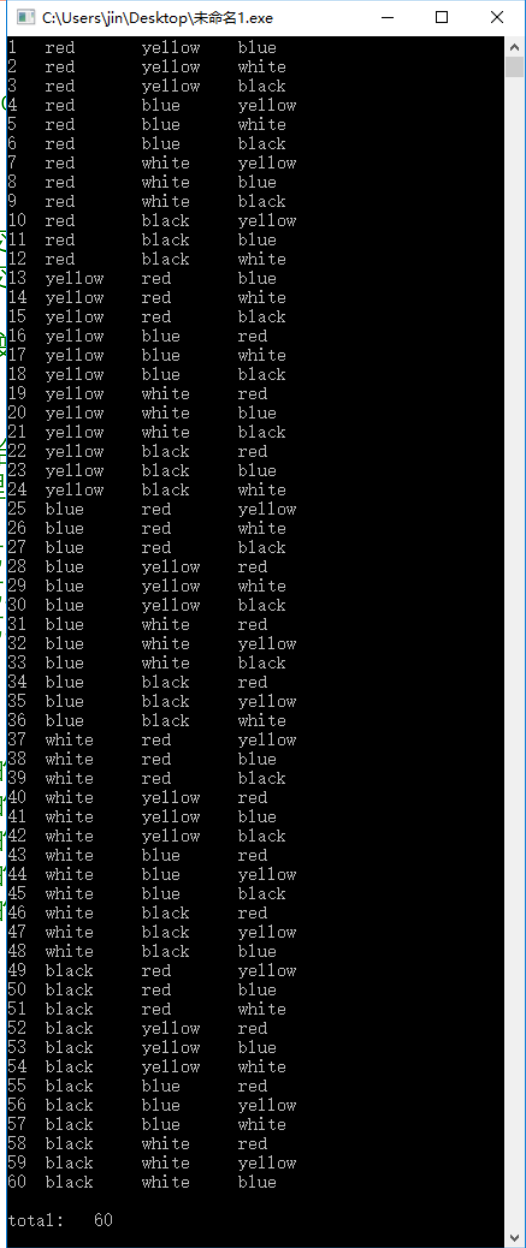
//pri=k

//pri=i

//pri=j

//pri=k

total: 60



*用typedef声明新类型名

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; //指定用Integer为类型名, 作用与int相同  
typedef float Real; //指定用Real为类型名, 作用与float相同
```

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

② 命名一个新的类型名代表数组类型

③ 命名一个新的类型名代表指针类型

④ 命名一个新的类型名代表指向函数的指针类型

```
typedef struct  
{  
    int month;  
    int day;  
    int year;  
}Date;           //声明了一个新类型名Date, 代表结构体类型  
Date birthday;   //定义结构体类型变量birthday, 不要写成struct Date birthday; ①  
Date*p;          //定义结构体指针变量p, 指向此结构体类型数据  
  
typedef int Num[100]; //声明Num为整型数组类型名 ②  
Num a;             //定义a为整型数组名, 它有100个元素  
  
typedef char* String; //声明String为字符指针类型 ③  
String p,s[10];       //定义p为字符指针变量, s为字符指针数组  
  
typedef int (*Pointer)(); //声明Pointer为指向函数的指针类型, 该函数返回整型值 ④  
Pointer p1,p2;         //p1,p2为Pointer类型的指针变量
```

用typedef声明新类型名

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（如： `int i;`）
- ② 将变量名换成新类型名（例如： 将*i*换成Count）
- ③ 在最前面加typedef（例如： `typedef int Count`）
- ④ 然后可以用新类型名去定义变量

简单地说，就是按定义变量的方式把变量名换上新类型名，并在最前面加typedef，就声明了新类型名代表原来的类型。

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]`
- ④ 用来定义变量： `Num a;` 相当于定义了： `int a[100];`

同样，对字符指针类型，也是：

- ① `char *p;` //定义变量p的方式
- ② `char *String;` //用新类型名String取代变量名p
- ③ `typedef char *String;` //加typedef
- ④ `String p;` //用新类型名String定义变量，相当`char *p;`

习惯上，常把用typedef声明的类型名的第1个字母用大写表示，以便与系统提供的标准类型标识符相区别。

用typedef声明新类型名

- (1) typedef的方法实际上是为特定的类型指定了一个同义字(synonyms)。
 - (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tyoedef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表面实质不同的。#define是在预编译时处理的，它只能作简单的字符串替换，而typedef是在编译阶段处理的，且并非简单的字符串替换。
 - (5) 当不同源文件中用到同一类型数据（尤其是像数组、指针、结构体、共用体等类型数据）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。
 - (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。
-