第7章 结构和链表

本章内容

- 结构
  - 结构类型的定义
  - 类型定义
  - 结构变量的定义
  - 结构变量的初始化
  - 结构成员的引用
  - 结构数组
  - 结构指针
- 动态内存分配
  - 动态内存申请
  - 动态内存释放
- 链表
  - 链表的建立
  - 链表的操作

●要求：

1.掌握结构数据类型、结构变量和结构指针的定义，结构成分的引用方法；
2.掌握结构数组的定义和使用方法；
3.掌握编写有结构形参、结构指针形参，以及返回结构的函数的方法；
4.掌握链接存储线性表的概念和基本操作；
5.掌握类型定义和变量定义的方法。

回顾：

在前面的课程中，我们学习了一些基本类型:（整型,实型、字符型）的定义和应用，还学习了构造类型：例如:数组（一维、二维）的定义和应用，这些数据类型的特点是：

当定义某一特定的数据类型，就限定了该类型变量的存储特性和取值范围。对简单数据类型来说，既可以定义单个的变量，也可以定义数组。而数组的全部元素都具有相同的数据类型，或者说是相同数据类型的一个集合。

7.1 结构类型和结构变量

在日常生活中，我们常会遇到一些需要填写的登记表，如住宿表、成绩表、通讯地址等。在住宿表中我们通常会登记上姓名、性别、身份证号码等项目；在通讯地址表中我们会写下姓名、邮编、邮箱地址、电话号码、E - mail等项目。这些表中集合了各种类型的数据.例如:

一个学生信息表

学号	姓名	性别	math	physic	english

通讯地址表由下面的项目构成

姓名	工作单位	家庭地址	邮编	电话号码	E-mail

- 描述一个或一个班的学生信息，包括以下信息：
  - 学号、姓名、性别、成绩
- 以前的解决方案：
  描述一个学生信息

```
int   num; //学号
char  name[20]; //姓名
char  sex       //性别
int   score[3]; //成绩
```

描述一个班的学生信息,则可用数组

```
#define N  50
int   num[N];       //学号
char  name[N][20];  //姓名
char  sex[N]        //性别
int   score[N][3]; //成绩
```

从中看出,这个表中集合了各种标准类型数据，我们还无法用前面学过的任一种数据类型对其进行整体描述，因此，C语言引入一种能集不同数据类型于一体的数据类型——结构类型(structure)。

它相当于其它高级语言中的记录。“结构”是一种构造类型，它是由若干“成员”组成的。每一个成员可以是一个基本数据类型或者又是一个构造类型。结构既然是一种“构造”而成的数据类型，那么在说明和使用之前必须先定义它，也就是要构造它。如同在调用函数之前要先定义函数一样。

假定把num,name, sex, score 合在一起，定义为结构类型st，那么它们在内存中将是连续存放的。

结构st

```
int   num;      //学号
char  name[20]; //姓名
char  sex;      //性别
float score[3]; //成绩
```

结构的功能:

结构被称为是一种构造性数据类型,有若干个成员组成的，通过它可以构造出一种非常灵活的数据类型。

结构的使用，是C 语言的一个很重要的特点。

定义一种结构类型，和基本类型（如int、float、char）一样，可以用来定义一维数组、多维数组、指针、多级指针、指针数组等等。

定义一个结构类型，需要注意如下几个要点：

- 结构类型中由哪些成员组成。
- 每个成员的数据类型是什么。其中，成员的类型也可以是结构类型。
- 怎样标识结构类型。
- 用结构类型定义哪种变量，如变量、指针、数组等。
- 怎样使用结构类型的成员。

结构的定义

可以采用不同的方式来定义结构。在了解定义的概念的基础上，可以根据编程规范或者个人习惯来采用某一种方式。

8

1. 结构类型

定义结构类型的一般形式为：

关键字

```
struct  结构类型名
{
   类型  成员名1;
   类型  成员名2;
   ……
};
```

成员说明表:

花括弧内是该结构类型中的各个成员（或称分量），是一组变量类型，由它们组成一个结构类型。每个成员都是该结构的一个组成部分。对每个成员也必须作类型说明。成员的命名规则与标准变量命名规则相同。

例如：学生数据结构类型为：

引出结构类型的定义。

```
struct student
{
  int   num;
  char  name[20];
  char  sex;
  int   score[3];
};
```

学号 姓名 性别 成绩

结构名:整个结构类型的名称。

成员说明表:指明组成此种结构类型的全部成员。

只是定义类型，没有定义变量。

在这个结构定义中，结构名为student，该结构由4个成员组成。第一个成员为num，整型变量；第二个成员为name，字符数组；第三个成员为sex，字符变量；第四个成员为score数组，整型变量。应注意在括号后的分号是不可少的。结构定义之后，即可进行变量说明。凡说明为结构student的变量都由上述4个成员组成。由此可见，结构是一种复杂的数据类型，是数目固定，类型不同的若干有序变量的集合。

2. 结构变量

在结构类型定义中，详细列出了结构类型所包含的每个成员的名称及其类型。实际上，结构类型定义只是表明一类实体其数据属性的"模式"，并不定义一个特定的数据实体，因此不要求分配存储单元。

程序如果要实际使用结构类型所描述的数据信息，就必须定义结构变量。结构变量要占用存储单元，能存放如结构类型所描述的具体数据。对结构类型和结构变量，我们可以简单地理解为，结构类型是表示数据框架的描述文本，结构变量才能存放实际数据。

● 结构类型变量的定义

(1)先定义结构类型后定义变量名

其一般形式为:

```
struct  结构类型名
{
    类型     成员名1;
    类型     成员名2;
    ……
};
```

形式：

1. struct 结构类型名 结构变量名表; //C句法
2. 结构类型名 结构变量名表; //C++句法

例如：struct student{
int num;
char name[20];
char sex;
int score[3];
};
student s1, s2[2], *s3=s2;

表示：表示s1是一个结构变量，含有4个成员num，name,sex,和score。
s2[2]是一个含有2个结构元素的数组。
s3是一个指向结构对象的结构指针变量。

	num	name	sex	score		
s1	1000	liaoli	F	88	90	98
S2[0]	1001	liaomin	M	98	78	100
S2[1]	1002	liaowan	M	100	69	88

s3 指向student类型的结构对象s2

注意：在编译时，不对结构类型分配空间，只对结构变量分配空间。结构变量s1的内存分配字节情况.

```
struct student
{ int   num;      //4bytes
  char  name[20];//20bytes
  char  sex;      //1(*4)byte
  int   score[3]; //12bytes
}s1;
```

总共需要4+20+1*4+12=40 bytes

```
void main()
{   printf("%d\n",sizeof(s1));
}
```

主要原因是在vc6中有字节对齐的问题.
字节对齐有助于加快计算机的取数速度，否则就得多花指令周期了。为此，编译器默认会对结构体进行处理（实际上其它地方的数据变量也是如此），让宽度为2的基本数据类型（short等）都位于能被2整除的地址上，让宽度为4的基本数据类型（int等）都位于能被4整除的地址上，以此类推。所以整个结构体的sizeof值就增长了。

同理,结构变量s2的内存分配的字节数40*2.

总共需要的字节数为40*3.

(2) 定义类型的同时定义变量
其一般形式为：

```
struct  结构类型名
{
    类型    成员名1;
    类型    成员名2;
    ……
}结构变量名表;
```

例如：

```
struct student{
        int num;
        char name[20];
        char sex;
        int score[3];
}s1, s2[2], *s3;
```

(3) 直接定义结构类型变量
若仅需要结构变量，则定义时可以不出现结构名。其一般形式为：

```
struct
{
    类型    成员名1;
    类型    成员名2;
    ……
}结构变量名表;
```

例如：

```
struct
{
    int num;
    char name[20];
    char sex;
    int score[3];
}s1, s2[2], *s3;
```

第三种方法与第二种方法的区别在于第三种方法中省去了结构类型名，而直接给出结构变量。注意:由于结构类型名省略了,以后将无法使用这种类型来定义其他的变量.

(4) 结构的其他定义方式

除了结构的基本定义方式，还可以用类型定义或者宏定义的方式来定义结构以及结构变量。即给已有类型起别名，提高程序可移植性：

1）利用类型定义(typedef)

```
typedef struct date
{
   short day, year;
   char month[3];
} DATE;
```

DATE s1, s2[3], *s3;

先将DATE 定义成类型，然后可以用来定义变量

2）利用宏定义(define)

```
#define DATE struct date
.....
DATE
{
    short day, year;
    char month[3];
};
```

DATE s1, s2[3], *s3;

先将 DATE 定义成宏，然后可以用来定义变量

(5) 结构的嵌套定义

结构类型的嵌套结构就是成员也可以是一个结构变量，即允许"嵌套"的结构类型。例如，给上述学生信息增加出生日期，并将出生日期定义为一种包含日、月、年3项信息的结构类型，则更完整的学生信息类型就被定义成嵌套的结构类型。

num	name	sex	address	birthday		
				day	month	year

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

```
struct student
{ int   num;
  char  name[20];
  char  sex;
  char  address[40];
  Date birthday;
}a,b;
```

● 结构变量的初始化

结构变量的初始化和一般变量的初始化是一样的.可以在定义结构变量的同时，对结构变量中的各个成员进行初始化。初始化时注意数据类型一致性。

例1:在定义结构类型的同时进行结构变量的定义及初始化。

```
struct  Point { /* 说明绘图程序的坐标类型 */
        int  x;
        int  y;
}p1 = {20, 50},p2;     /* p1的x值为20，p1的y值为50 */
```

也可以在定义结构类型与声明结构变量分开的情况下，在声明结构变量时进行初始化。例如，利用已定义的结构类型Point，结合声明结构变量进行初始化：

Point p3={10,40}，p4={ 20,50};//采用C++句法定义变量

例2:先定义结构类型，再进行结构变量的定义及初始化。

```
struct  student
{
     int num;
     char  name[20];
     char  sex;
     int  score[3];
};
[struct] student a={1000,"Liu Lin",'F' {99,78,100}};
```

结构类型和结构变量一再说明

1. 要注意结构类型名和结构变量名的区别。

不能对结构类型名进行赋值、存取或运算，因为类型不占用存储空间；而结构变量会占用存储空间，定义时可以赋初值，定义后可引用。

2. 结构变量初始化的时间。

静态结构变量初始化遵守与其它静态变量初始化相同的规则。即静态的和全局的结构变量初始化在程序执行之前完成，静态的结构变量未指定初值时，结构变量的每个成员的值自动置二进制代码为全0的值。局部结构变量初始化是程序控制每次进入它所属辖域时创建并初始化，未指定初值的局部结构变量其初值是不确定的。

3.可以定义指向结构的指针变量（结构指针变量简称结构指针）。

设有结构变量s和能指向该结构的指针变量p，当把s所占据的存储块的开始地址赋值给p时(p = &s)，就说结构指针p指向结构变量s。定义结构指针的方法与定义一般指针变量一样，当类型区分符是结构类型时，所定义的指针变量即为结构指针。例如，

```
Date  *pd, date3; //采用C++句法定义变量
pd = &date3;
```

定义结构指针pd和结构变量date3，并使结构指针pd指向结构变量date3，即结构指针pd的内容为结构变量date3所占据的存储块的首地址。

3.结构变量的引用

结构变量定义后,就可以引用结构变量和结构的成员变量.

(1) 引用结构变量

引用结构变量 有两种方法：

1. 用结构变量名直接引用结构变量
2. 指向结构变量的指针间接引用结构变量。

程序引用结构变量有以下多种应用。

- ●相同类型的结构变量相互赋值
- ●让结构指针指向结构变量
- ●将结构变量作为实参调用带结构形参的函数

例如：

```
struct date
{
  int month;
  int day;
  int year;
};
```

```
struct  student
{ int   num;
  char  name[20];
  char  sex;
  struct date birth;
  char  address[40];
};
```

```
student s1 = { 10005, "Yang ming", 'F', {20, 11, 1985},
"15 Nanjing Rd"}, s2;
```

把一个结构变量赋值给同类型的另一个结构变量。例如

```
s2 = s1; /* 将结构变量s1整体赋值给结构变量s2 */
```

也可以将结构变量s1所占据的存储块的首地址赋值给结构指针变量,如:

```
 student *p1=&s1;/*定义指向结构变量s1的结构指针p1*/
```

上述代码定义结构指针变量p1，并让它指向结构变量s1。

(2)引用结构成员

引用结构成员有以下3种方法。

- 使用结构变量和成员运算符
- 使用结构指针和指针运算符
- 使用结构指针和成员运算符

♦结构变量引用规则

● 成员的访问。

对结构变量中各个成员的访问，用操作符"."表示，

其格式为：　　结构变量名.成员名

操作符"."称为成员运算符，具有最高优先级。C允许直接赋值给一个结构变量成员，而不能将一个结构变量作为一个整体进行输入和输出。例如，用代码s1.name直接引用s1结构变量的name成员。s1.name成员可以在程序中单独使用，与普通变量完全相同。

引用结构变量的成员对该成员进行输入输出、赋值、运算等操作。例如:scanf(" %d%s",&s1.num,s1.name);

```
printf("学号：%d　姓名：%s　成绩：%d,%d,%d\n", s1.num, s1.name, s1.score[0],s1.score[1],s1.score[2]);
/*更新学生s1的成绩.*/
```

输出学生student的信息。代码：s1.score[0]+= 5;

strcpy(s1.name, "Li ming");/*则将Li ming 赋给其结构变量s1中的name成员*/

以下语句是错误的：

```
scanf("%s,%ld,%s,%c,%f,%f,  %f ",&s1);
printf("%s,%d,%s,%c,%f,%f,  %f \n",s1);
```

●对成员变量可以象普通变量一样进行各种运算。

下列运算是正确的：

```
s1.num++;//学生s1的学号增加1
++s1.num; //学生s1的学号增加1
```

注意：运算s1.num++是对s1中的num进行自加运算，而不是先对s1进行自加运算。

```
for(i=sum=0;i<3;i++)
    sum+=s1.score[i];//统计学生s1的总分
```

●可以引用成员的地址，也可以引用结构变量的地址。

结构变量成员地址的引用方法

&结构变量名.成员名

结构变量地址的引用方法

&结构变量名

例如:

```
scanf("%d",&s1.num);       /*键盘输入s1.num的值*/
scanf("%d",&s1.score[1]);/*键盘输入s1.score[1]值*/
printf("%d",&s1);          /*输出s1的首地址*/
```

&s1.score[1]表示引用结构变量s1的成员score[1]的地址.

结构变量的地址主要用于作函数参数，传递结构的地址值。

●指向结构变量的指针

定义指向结构类型变量的指针变量：

[struct] 结构类型名 *指针变量名;

也可将所定义的指针变量指向结构类型变量(注意，它只能指向一个结构类型变量，而不能指向结构类型变量的某一成员)。如：　指针变量=&结构变量;

引用形式如下：

指针变量->成员

其中算符"->"称为指向运算符，由一个减号"-"和一个大于号">"组成.（请注意中间不能有空格符），其优先级与成员运算符"."一样，也是最高优先级的运算符。

下面是与指向运算符相关的几种运算：

在C语言中，有以下三种方法引用结构成员：

① 结构变量.成员名;

② p->成员名;

③ (*p).成员名;//使用结构指针和成员运算符引用结构成员：

第③种,圆括号是必要的。若省略，由于成员运算符"."优先级高于"*"，将会导致编译错误。一般采用前两种方法之一。

```
struct student
{   int num;
    char name[20];
    char sex;
    int  age;
    int score;
}st1,*p=&st1;
```

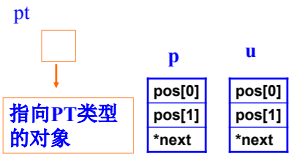
p → 1000 | liupin | F | 20 | 100

```
p->num=1000; strcpy(p->name, "liupin")
(*p).sex='F';st1.age=20;st1.score=100;
```

```
(*p).sex ='F';
/*与代码 p->sex ='F';   等价 */
```

又如:定义结构变量 p 和u，以及结构指针pt：

```
#define PT struct pNode
PT
{     float pos[2];
      PT  *next;
}p, u, *pt;
```



希望产生以下效果的语句，将数值填入到结构成员中去，如何编写程序？

建立静态链表

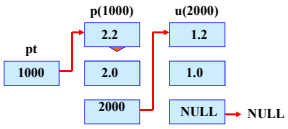


图7.1单向静态链表示意图

```
#include <stdio.h>              //建立静态链表
#define PT struct pNode
void main()
{   PT{  double pos[2]; PT  *next;  }p, u, *pt;
        pt = &p;            // pt 指向p
        p.pos[0] = 2.2; // p的数组成员pos 的首元素pos[0]置值2.2
        pt->pos[1] = 2.0; // pt 对象的成员pos的元素pos[1]置值2.0
        p.next = &u;            // p 的成员next 指向结构变量u
        p.next->pos[0] = 1.2; //结构成员赋值
        u.pos[1] = 1.0;         //结构成员赋值
        u.next = NULL;      // 结构的指针成员赋空地址
        for(;pt;pt=pt->next)//输出数据
            printf("%f,%f\n",pt->pos[0],pt->pos[1]);
}
```

●结构的嵌套引用

结构类型的嵌套结构就是成员也可以是一个结构变量，即允许"嵌套"的结构类型。例如，给上述学生信息增加出生日期，并将出生日期定义为一种包含日、月、年3项信息的结构类型，则更完整的学生信息类型就被定义成嵌套的结构类型。

number	name	sex	birthday		
			day	month	year

```
struct Date  //说明一个日期
{   int  day;     // 日
    int  month;   //月
    int  year;    // 年
};
```

```
struct  student
{ int   numb;       // 学号
  char  name[20];   // 姓名
  char  sex;        // 性别
  Date   birthday;
}s={1000,"Liu Lin",'f',12,7,1988},b;
```

```
/*采用C++句法，birthday的类型是结构类型Date，
如果采用C句法，则在Date之前用struct引导 */
```

首先定义一个结构Date，由day(日)、month(月)、year(年)三个成员组成。在定义并说明变量a 和 b 时，其中的成员birthday被说明为Date结构类型。成员名可与程序中其它变量同名，互不干扰。

上述代码先定义Date结构类型，由day、month和year三个成员组成。然后在定义student结构类型时，将成员birthday指定为Date结构类型，使类型student是一个嵌套的结构类型。

在实际应用中，为表示复杂的数据结构，常常用到这种嵌套的结构类型。在结构类型中有数组和结构成员，数组的元素又是结构，结构中又有结构，嵌套层次会有许多层。如果成员本身又是一个结构则必须逐级找到最低级的成员才能使用。系统只能对最低的成员进行赋值或存储等运算。

例如,对上面定义的struct student结构变量s,可以这样访问各成员：

```
s.name
s. birthday.day
s. birthday.month
s. birthday.year
```

不能是s. Date，因为Date本身是一个结构类型名。

【例7.1】利用结构变量，输入3个学生的姓名、语文成绩和数学成绩，然后计算每个学生的平均成绩并输出。

```
#include <stdio.h>
struct stuScore
{     char name[20];
      int  chinese;
      int math;
};
```

```
void main()
{ float aver1,aver2,aver3;
  stuScore st1,st2,st3;      /* 采用C++句法，定义3个结构变量 */
  printf("请输入3位学生的姓名、语文成绩、数学成绩\n");
  scanf("%s%d%d",st1.name,&st1.chinese,&st1.math);
  scanf("%s%d%d",st2.name,&st2.chinese,&st2.math);
  scanf("%s%d%d",st3.name,&st3.chinese,&st3.math);
  aver1=(st1.chinese+st1.math)/2.0; /* 计算平均成绩 */
  aver2=(st2.chinese+st2.math)/2.0;
  aver3=(st3.chinese+st3.math)/2.0;
  printf("姓名\t语文\t数学\t平均成绩\n");
  printf("%s\t%4d\t%4d\t%6.2f\n",
              st1.name,st1.chinese,st1.math,aver1);
  printf("%s\t%4d\t%4d\t%6.2f\n",
              st2.name,st2.chinese,st2.math,aver2);
  printf("%s\t%4d\t%4d\t%6.2f\n",
              st3.name,st3.chinese,st3.math,aver3);
}
```

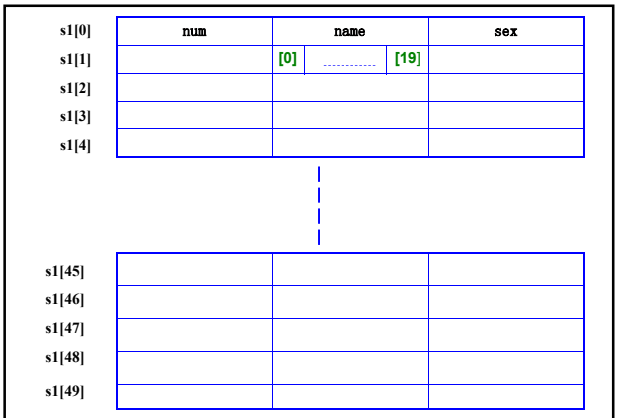
7.2 结构数组

- 通常用结构（变量）描述个体，用数组描述个体的集合。当数组的元素是结构时，这种数组就称为结构数组。在实际应用中，经常用结构数组来表示具有相同数据结构的一个群体。如一个班的学生档案，一个车间职工的工资表等。
- 定义的方法和结构变量相似，只需说明它为数组类型即可。

例如：用结构变量s1描述某学生的有关信息，而用结构数组可表示一个班的学生。

```
struct  student {
int  num;
char name[20];
char  sex;
}s1[50];
```

定义了一个结构数组s1，共有50个元素，s1[0]～s1[49]。每个数组元素都具有struct student的结构形式，即:有num, name和sex成员.



1. 结构数组的定义

(1) 先定义结构类型，再定义结构数组

以下代码先定义结构类型stuScore，然后用这个结构类型定义结构数组：

```
struct stuScore
{     char name[20];
      int chinese;
      int math;
};
stuScore  st[3];
/* 采用C++句法，定义有3个元素的结构数组 */
```

(2) 在定义结构类型的同时定义结构数组

以下代码实现与前面代码同样的效果，在定义结构类型stuScore同时定义结构数组：

```
struct stuScore
{     char name[20];
      int chinese;
      int math;
}st[3];
```

●结构数组的初始化

与结构变量初始化相仿，在定义结构数组时，也可给结构数组赋初值。例如：

```
struct stuScore
{     char name[20];
      int chinese;
      int math;
}st[3]={{"Zhang",80,85},{"Li",85,90},
        {"Wang",90,70}};
```

	name	Chinese	math
st[0]	Zhang	80	85
st[1]	Li	85	90
st[2]	Wang	90	70

图7.2结构数组st的逻辑结构

2. 结构数组的引用

(1)结构数组元素成员的引用

使用结构数组元素和成员运算符引用结构数组元素成员

◆结构数组元素成员的引用方法

结构数组名[元素下标].结构成员名

例如：

```
printf("%s", st[1].name);//输出第2个学生的姓名，即Li
```

◆结构数组元素成员地址的引用方法

&结构数组名[下标].成员名

◆结构数组元素地址的引用方法

&结构数组名[下标]

◆结构数组首地址的引用方法

结构数组名

(2) 使用结构指针和指针运算符

使用结构指针和指针运算符引用结构数组元素成员的一般形式为：

指针变量名->成员名

其中算符“->”称为指向运算符，

由一个减号“-”和一个大于号“>”组成。

例如：

```
stuScore *sp = st;//定义结构指针sp,指向结构数组st首元素
printf("%s\n", sp->name); // 输出第1个学生姓名，Zhang
sp++;        // 结构指针指向下一个元素
printf("%s\n", sp->name); // 输出第2个学生姓名，Li
```

(3)使用结构指针和成员运算符

使用结构指针和成员运算符引用结构数组元素成员的一般形式为：

(*指针变量名).成员名

例如：

```
stuScore *spt = st;
/* 定义stuScore结构指针spt,指向结构数组st首元素*/
printf("%s\n", (*spt).name);// 输出学生st[0]姓名，Zhang
spt++; // 结构指针指向下一个元素，指向st[1]
printf("%s\n", spt->name);// 输出学生st[1] 姓名，Li
```

当指针变量指向数组首地址后，也可以用以下方法取成员值：

a. (*(指针变量+i)).成员名

b. (指针变量+i)->成员名

指向结构数组的指针的引用注意事项：

指向结构变量的指针变量，可以指向结构变量，也可以指向同类型的结构数组的元素。

```
struct student
{   int num;
    char name[30];
    float score;
}stu[30], *p;
```

p=stu;等价与p=&stu[0];

p=&stu[2].num; ✗

```
int *ip;
ip=&stu.num;
```

p只能指向一个struct student类型的数据(某个元素的的起始地址)，不能指向一个成员变量。

例如：

```
#include <stdio.h>
void main()
 {
        struct a
        { int i,*p; }k;
        int j=1234,*ip;
        k.i=j;
        k.p=&k.i;// k.p=&j;
        ip =&k.i;
        printf("%d,%d,%d\n",k.i,*k.p ,*ip);
}
```

注意：

若p的初值为stu，p+1后指向下一元素的起始地址；

(++p)->num 先使p自加1，然后得到它指向元素的num成员值

++p->num 使p指向的结构体变量中的成员num的值加1

(p++)->num 先得到p->num的值，然后使p自加1，指向stu[1]

p->nunm++ 使p指向的结构体变量中的成员num的值加1

48

【例7.2】利用结构数组，输入3个学生的姓名、语文成绩和数学成绩，然后计算每个学生的平均成绩并输出（即用结构数组替换例7.1中的结构变量）。

```
#include <stdio.h>
struct stuScore
{     char name[20];
      int chinese;
      int math;
};
```

```
void main()
{ int i;float aver[3];
  stuScore st[3];  /* 定义结构数组，含3个元素 */
  printf("请输入3位学生的姓名、语文成绩、数学成绩\n");
  for(i=0;i<3;i++)
    scanf("%s%d%d",
st[i].name,&st[i].chinese,&st[i].math);
  printf("姓名\t语文\t数学\t平均成绩\n");
  for(i = 0; i < 3; i++)
  { aver[i]=(st[i].chinese+st[i].math)/2.0;
          /*计算平均成绩 */
    printf("%s\t%4d\t%4d\t%6.2f\n",
      st[i].name,st[i].chinese,st[i].math,aver[i]);
  }
}
```

7.3 结构和函数

函数的形参可以是基本类型变量、指针、数组，也允许是结构。

将一个结构传递给函数有三种方式：

传递单个成员、传递整个结构、传递指向结构的指针。

◆传递单个成员：用结构变量的成员给函数，在C中认为是值调用，即在被调用的函数中尽管修改了形参的值，但不会改变调用函数时提供的实参变量的值。

◆传递结构变量：用结构变量给函数，在C中是值调用用.此外，结构变量作函数参数进行整体传送。但是这种传送要将全部成员逐个按值传送，特别是成员为数组时将会使传送的时间和空间开销很大，严重地降低了程序的效率。

◆传递结构指针：即用指针变量（或结构数组名）作函数参数进行传送，这里由实参传向形参的只是地址，从而减少了时间和空间的开销。

注意：传递单个成员、传递整个结构是值传递，不影响实参的值，即单向传递.

1.结构成员作为函数的参数(单向传递)

【例7.3】日期转换程序，根据输入的年月日，输出是该年中的第几天。

```
#include  <stdio.h>
int dayTable[ ][12]={
      {31,28,31,30,31,30,31,31,30,31,30,31}, /* 平年 */
      {31,29,31,30,31,30,31,31,30,31,30,31}};/* 闰年 */
struct Date /* 定义一个Date结构类型 */
{  int  day;
   int  month;
   int  year;
   int  yearDay;
} date; /* 定义一个Date结构类型的结构变量 */
```

```
int dayofYear(int d, int m, int y)/*计算年中第几天函数 */
{ int  i, leap, day = d;
  leap = (y%4 == 0 && y%100)||y%400 == 0;/* 是否闰年 */
  for (i = 0; i < m-1; i++)
     day += dayTable[leap][i];
  return day;     /* 返回计算的结果 */
}
void main()
{ int leap, days;
  printf("\tDate Conversion Program\n");
  printf("Year = ");
  scanf("%d", &date.year);    /* 输入年份 */
```

```
  for (; ;)    /* 输入月份，并检查是否在1～12之间 */
   {  printf("Month = ");   scanf("%d", &date.month);
      if (date.month >= 1 && date.month <= 12) break;
      printf("输入的月份必须在 1 到 12 之间\n" );
   }
 /* leap=1是闰年，leap=0 不是闰年 */
   leap = (date.year%4 == 0&&date.year%100)||date.year%400==0;
   days = dayTable[leap][date.month-1];
   for (; ;)   /* 输入日期，并检查是否输入正确 */
   {  printf("Day = ");      scanf("%d", &date.day);
      if (date.day >= 1 && date.day <= days) break;
      printf("输入的天必须在 1 到 %d 之间\n",days);
   }
 /* 调用dateofYear函数，实参为结构date的3个成员 */
   date.yearDay = dayofYear(date.day, date.month, date.year);
   printf("The days of the year are: %d\n",date.yearDay);
 }
```

在上述主函数中，调用dayofYear函数使用了date结构的3个成员：

day、 month 和 year

作为实参。

在dayofYear函数中用3个形参：

d、m和y

与之对应，并将最后的计算结果通过执行"return day;"语句返回给主函数，并赋值给Date结构类型变量的yearDay成员。

2.结构作为函数参数(单向传递)

用结构作为实参，将这个结构的所有成员都传递给了被调用函数的形参,分配内存单元。不影响实参值.以例7.3程序为例，对主函数中对函数dayofYear的调用，原来使用表示日、月、年的3个结构成员作为实参改为用一个结构变量date作为实参。即将代码

```
date.yearDay = dayofYear(date.day, date.month, date.year);
```

改写成

```
date.yearDay = dayofYear(date);
```

对应的dayofYear函数也要作如下的修改：

```
int dayofYear(Date d)/* 计算年中第几天，设有结构形参 */
{ int  i, leap, day = d.day;
  leap=(d.year%4 == 0 && d.year%100)||d.year%400 == 0;
    /* 计算是否闰年 */
  for (i = 0; i < d.month-1; i++)
        day += dayTable[leap][i];
  return day;     /* 返回计算的结果 */
}
```

在主函数中，对dayofYear函数的调用直接使用了date为实参，在dayofYear函数中用结构形参Date d。

结构变量或结构成员作为实参传递给函数形参是值传递。dayofYear函数只能把计算结果返回给主函数，而不能简单地将计算结果直接赋值给形参的yearDay成员。

3.结构指针作为函数参数(双向传递)

使用结构变量作为函数的形参，在函数调用时，系统要为结构形参分配存储单元，并为实参结构向形参结构完成值传递等。为了减少系统开销并提高效率，C语言也允许指向结构的指针作为函数的形参。实参可以是结构的地址，也可以是指向结构的指针变量(或结构数组名。）作函数参数进行传送，这里由实参传向形参的只是地址，从而减少了传递时间和空间上的开销。

仍以例7.3程序为例，主函数中对函数dayofYear的调用，由使用结构变量date作为实参现改为用结构变量date的地址作为实参。即，将代码

```
date.yearDay = dayofYear(date);
```

改写成

```
dayofYear(&date);
```

对应的dayofYear函数也要作如下的修改：

```
void dayofYear(Date *dp)/* 计算年中第几天函数，设有结构指针形参，无返回值 */
{ int i, leap, day = dp->day; /*确定是闰年或平年*/
  leap =(dp->year%4 == 0 && dp->year%100)|| dp->year%400 == 0;
  for(i = 0; i < dp->month-1; i++)
      day += dayTable[leap][i];
  dp->yearDay = day;
/* 计算结果回填到yearDay成员，没有return语句 */
}
```

在主函数中，对dayofYear函数的调用使用date结构变量的地址作为实参，因此在dayofYear函数中必须用结构指针形参Date *dp与之对应。被调用函数dayofYear通过结构指针形参引用所指向的结构成员，并将计算结果存放到所指结构的相应成员yearDay，不必再返回结果。

(1) 结构或结构成员作为函数参数是值传递方式，在被调用函数dayofYear中必须用return 语句返回结果。如果不用return语句，写成"d.yearDay = day;"，主函数不能获得函数结果。因为函数的形参是函数的局部变量，函数调用时，将结构date的各成员值依次送给形参d的各成员中，以后函数对d作任何的改变与date无关。见图7.3所示。

main函数调用语句 date结构（实参）　　dayofYear数 d结构（形参）

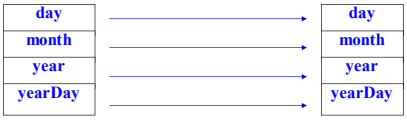


图7.3 结构作为函数参数

(2) 使用结构地址或结构指针作为函数的参数，函数调用时，虽只传递结构的地址给结构指针形参，但通过该结构指针形参间接引用所指向的结构变量。因此，函数既可以引用结构指针形参所指向结构的成员，也可把计算结果存储于结构指针形参所指向结构的成员中。见图7.4所示。

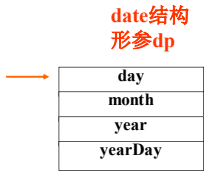


图7.4 结构地址作为函数参数

4. 函数返回结构类型值

在7.3的2中，将结构date作为实参传递给dayofYear函数的结构类型形参d，但计算结果是通过return day;语句返回到主函数的调用语句。如果希望将计算的结果先保存到结构类型形参d的成员yearDay中，然后返回结构类型形参d的值，可以对函数dayofYear重新改写如下：

```
Date dayofYear(Date d)
{ int i, leap;
  d.yearDay = d.day;
  leap = (d.year%4 == 0 && d.year%100) ||
         d.year%400 == 0; /* 计算是否闰年 */
  for(i = 0; i < d.month-1; i++)
       d.yearDay += dayTable[leap][i];
  return d;              /* 返回结构类型 */
}
```

经上述改动后，主函数调用dayofYear函数后，需要把返回的结构赋值给结构变量date，调用语句也作相应的修改。即将

```
date.yearDay = dayofYear(date);
```

改写成

```
date = dayofYear(date);
```

调用带有结构类型形参的函数时，实参结构的各成员的值需要全部拷贝给形参结构的相应成员，费时间又费空间。一般情况下，以传递结构地址或指针为好。有时为了程序的安全性，确保函数不能修改实参结构情况下，可使用结构作为形参。调用函数返回结构值的函数，需要将函数的返回值赋值保存于某个结构变量。对于复杂的结构类型，实现的效率较低。

习题:p196 2, 5

上机:

p197 6

编写一个洗牌程序，实现给四人发牌，最后显示四人所拿的13张牌（牌点和花色，无大小怪）。