# 项目5

## 学生成绩处理

### 项目说明

本项目用C语言编写一个程序进行学生成绩的统计与管理。为了简化项目程序，每位学生只记录3门课程的成绩，要求从键盘输入学生的信息，包括学号，姓名，(计算机基础、程序设计、英语)三门课程的成绩。并实现以下几个功能：

(1)输入学生的学号信息，实现学生成绩信息的查询。

(2)按计算机基础课程成绩从大到小对学生信息进行排序并输出。

(3)统计程序设计课程成绩大于或等于60分的学生个数。

(4)汇总计算出学生的平均成绩，输出汇总后的成绩单。



# 任务1　学生成绩信息的输入与输出

## 一、任务概述

一次期末考试中，一个班级共30位同学参加了计算机基础、程序设计、英语考试，现要将这个班的30位同学的相关信息(包括学号、姓名、三门课的成绩)从键盘上输入，然后输出这30位同学的成绩单。

## 二、预备知识

### (一)结构体类型

问题：如何表示如图5.1所示的名片数据？


图5.1　名片数据

单独分析以上数据，我们可以定义以下类型的变量来分别表示上面的数据：

```
char name[10];              /*张三*/
int age;                    /*20*/
char sex[2];                /*男*/
char xh[20];                /*学号*/
char nation[20];            /*壮*/
char mobile[20];            /*13909090909*/
```

如果还有下面的数据，如何表示？

李四、18 岁、男、学号 010101011、汉族、手机号 13908080808

孙红、19 岁、女、学号 010101012、汉族、手机号 13907070707

……

能否用一种类型来统一描述以上数据？由于必须类型相同才能构造成数组，显然以前学习的数据类型都不能很好地解决问题。在 C 语言中提供的结构体类型可以把这些不同类型的数据组合起来构造成一种新的数据类型，用起来更加方便。

结构体类型的定义形式为：

```
struct 类型名
{
    成员说明表列
};
```

例如，前面[问题]中提到的数据可以表示如下：

```
struct student          /*结构体类型名*/
{
    char name[10];      /*结构体成员，以下都是*/
    int age;
    char sex[2];
    char xh[20];
    char nation[20];
    char mobile[20];
};
```

struct 是结构体关键字，结构体类型定义中的每个成员项都有确定的类型和名称，称为结构体类型的“域”，每个域的定义后面要有“;”号。

结构体类型由用户定义，所以结构体类型不是固定结构的类型，用户可以定义不同结构的结构体类型，也可以定义相同结构的结构体类型，系统均认为是不同的结构体类型，例如下面是两个不同的结构体类型，虽然 aa 和 bb 的结构是一样的：

```
struct aa{int a; int b; char c;}
struct bb{int a; int b; char c;}
```

定义了结构体类型，就可以定义结构体变量、结构体数组了。

### (二)结构体变量的定义与引用

#### 1. 结构体变量的定义

(1)用已定义的结构体类型名定义变量。例如:

struct student wang,zhang;/ * 定义了两个结构体变量 wang 和 zhang * /

(2)在定义结构体类型的同时定义结构体变量。例如:

```
struct student                /* 结构体类型名 */
{
    char name[10];            /* 结构体成员,以下都是 */
    int age;
    char sex[2];
    char xh[20];
    char nation[20];
    char mobile[20];
}wang,zhang;
```

(3)不定义结构体类型名,直接定义结构体变量。例如:

```
struct
{
    char name[10];          /* 结构体成员,以下都是 */
    int age;
    char sex[2];
    char xh[20];
    char nation[20];
    char mobile[20];
}wang,zhang;
```

这种定义形式由于没有给结构体类型命名,只能一次性定义若干结构体变量。

结构体类型的长度可以用 sizeof 运算符计算出来,形式为:

sizeof(结构体类型名)

或者

sizeof(变量名)

#### 2. 结构体变量的引用

数组元素的引用采用数组名和下标结合的引用方法,例如 a[3]、b[4]等。结构体变量的成员的引用则采用成员运算符“.”来完成,格式为:

结构体变量名.成员名

例如前面定义的变量 wang,其成员引用如下:

wang.age

注意:成员运算符“.”的优先级最高。

### (三)结构体数组

结构体类型既可以定义单个的变量,也可以定义结构体数组,用以存储批量的数据,例如一个班级的学生信息。

**1. 结构体数组的定义**

和结构体变量定义一样，结构体数组的定义也有以下 3 种方法：

(1)先定义结构体类型，用结构体类型名定义结构体数组，例如：

```
struct student
{
    char name[10];
    int age;
    char sex[2];
    char xh[20];
    char nation[20];
    char mobile[20];
};
struct student stu[20];
```

(2)定义结构体类型名的同时定义结构体数组，例如：

```
struct student
{
    char name[10];
    int age;
    char sex[2];
    char xh[20];
    char nation[20];
    char mobile[20];
}stu[20];
```

(3)不定义结构体类型名，直接定义结构体数组，例如：

```
struct
{
    char name[10];
    int age;
    char sex[2];
    char xh[20];
    char nation[20];
    char mobile[20];
}stu[20];
```

**2. 结构体数组的初始化**

和普通数组的元素是普通变量一样，结构体数组的每一个元素相当于一个结构体变量，二者的初始化也很类似，例如：

```
struct student stu[2]={
{"张三",20,"男","010101010","汉族","13909090909"},
{"李四",18,"男","010101011","汉族","13908080808"}};
```

**3. 结构体数组的引用**

结构体数组元素的成员表示为:结构体数组名[下标].成员名

例如:stu[i].age　　/* 下标为 i 的数组元素的成员 age */

## 三、任务分析与实施

任务分析:从任务描述中可以知道每一位学生包括学号,姓名和三门课程的成绩等 5 个信息,这符合结构体类型记录信息的特点。共有 30 位同学的信息,要记录多个同学的信息可以用数组。因此,在此可以采用结构体数组记录 30 位同学的信息。要输入和输出 30 位同学的信息,可以采用循环实现。

```
#include<stdio.h>
#define N 5  //为了程序运行方便,假设只有 5 个同学
//声明结构体类型 stu
struct stu
{
    char id[6];
    char name[10];
    int m1,m2,m3;
};

//输入函数
void input_stu(struct stu s[])
{
   int i;
   //循环输入学生信息
   for (i=0;i<N;i++)
   {
   printf("请输入第%d个同学的记录:",i+1);
   scanf("%s%s",s[i].id,s[i].name);
   scanf("%d%d%d",&s[i].m1, &s[i]. m2,&s[i].m3);
   }
}
//输出函数
void output_stu(struct stu x[])
{
   int i;
   printf("他们的成绩单为:\n");
   //循环输出学生信息
   for(i=0;i<N;i++)
   {
       printf("%s\t%s\t",x[i].id,x[i].name);
       printf("%d,%d,%d\n",x[i].m1, x[i].m2,x[i].m3);
```

```
    }
}
main()
{
    //声明结构体数组 student
    struct stu student[N];
    input_stu(student);//调用输入函数录入数据
    output_stu(student);//调用输出函数显示结果
}
```

程序说明与测试：

(1)#define N 5 语句是宏定义一个常量 N 等于 5。

(2)语句 stu student[N]语句声明一个结构体数组 student，其大小为 5。

(3)程序中数据输入由 input_stu()函数完成，输出由 output_stu()函数完成，main()函数通过参数交换批量数据。

(4)程序运行，输入学生的相关信息，结果如图 5.2 所示：

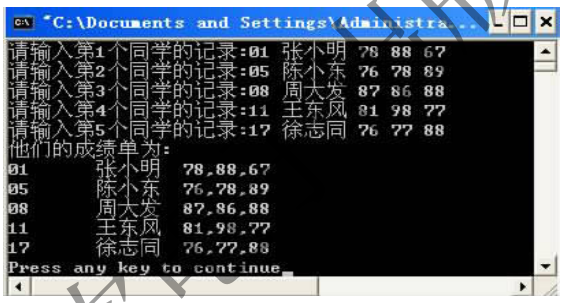


图 5.2　运行结果

## 四、同步训练

**1. 填空题**

(1)“.”称为_____运算符，“->”称为_____运算符。

(2)若有如下定义语句，则变量 w 在内存中所占的字节数是_____。

```
struct st{int a;char name[10];double ave};
```

(3)以下程序用来输出结构体变量 ex 所占存储单元的字节数，请填空。

```
struct st
{
    char name[20];double score;
};
main()
{
    struct st ex;
    printf("ex size: %d\n",sizeof(_____));
}
```

**2. 编程题**

(1)定义一个结构体,成员项包括一个字符型,一个整型。编程实现结构体变量成员项的输入和输出。

(2)建立一个结构体,其中包括学生的姓名,性别和计算机课程的成绩。建立一个有 5 个元素的结构体数组。输入学生信息,输出分数大于平均分的同学的姓名、性别和计算机课程成绩。

# 任务 2　学生成绩信息的查询与排序

## 一、任务概述

一次期末考试中,一个班级共 30 位同学参加了计算机基础、程序设计、英语考试,要求从键盘上输入 30 位同学的相关信息(包括学号、姓名、三门课的成绩),按计算机基础成绩从小到大输出 30 位同学的相关信息;然后在键盘输入任意一位同学的学号,如果后面输入的学号与前面输入的 30 位同学的学号出现匹配,则输出该同学的成绩信息。

## 二、预备知识

**1. 结构体的赋值**

给结构体变量赋值有三种方法:初始化、赋值和从键盘读入。

(1)结构体变量可在声明时直接进行初始化。初始化数据应放在大括号中,并根据成员变量的声明次序排序,同时数据之间的类型应一致。例如:

```
struct student stu={2001,"张华",'M',86.00,92.2};
```

或

```
struct student
{
  int num;
  char name[8];
  char sex;
  float score[2];
}stu={2001,"张华",'M',86.00,92.2};
```

例 5.1 定义结构类型并定义它的变量,在声明时直接进行初始化,最后在终端中输出结构体变量各成员的值。

分析:初始化结构体变量 stu,在初始化时直接给各成员变量赋初值,在主程序中用 printf 直接输出结构体各成员变量的值。

```
//5-1.c
#include<stdio.h>
struct student
{
```

```
    int num;
        char name[8];
        char sex;
        float score[2];
    }stu={2001,"张华",'M',86.00,92.2};
    int main()
    {
        printf("%d\t%s\t%c\t%f\t%f\n",stu.num,stu.name,stu.sex,stu.score[0],stu.score
[1]);
        return 0;
    }
```

编译程序 5-1.c,编译成功后运行程序,程序的输出结果为:

```
2001    张华    M    86.000000    92.199997
```

(2)对结构体变量的数据成员进行逐个赋值。

例 5.2 定义一个结构体类型及结构体变量,逐个给结构体变量赋值,最后输出结构体变量的值。

分析:定义结构体类型 student 及结构体变量 stu,在程序中逐个给成员变量赋值,最后用 printf 直接输出结构体各成员变量的值。

```
    //编辑源程序代码 5-2.c
    #include<stdio.h>
    #include<string.h>
    struct student
    {
        int num;
        char name[8];
        char sex;
        float score[2];
    }stu1;
    int main( )
    {
        stu.num=2001;
        strcpy(stu.name,"张华");
        stu.sex='M';
        stu.score[0]=86.00;
        stu.score[1]=92.2;
        printf("%d\t%s\t%c\t%f\t%f\n",stu.num,stu.name,stu.sex,stu.score[0],stu.score
    [1]);
        return 0;
    }
```

编译程序 5-2.c,编译成功后运行程序,程序的输出结果为:

2001　　张华　M　86.000000　92.199997

(3)从键盘中逐个读入结构体数据成员。

例 5.3 利用结构体变量存储,从键盘中逐个读入结构体数据成员,给成员变量赋值,在终端输出张华同学的个人信息。

分析:初始化结构体变量 stu,在初始化时并不赋初值,在主程序中等待键盘输入,依次把键盘的输入赋值给结构体变量 stu 的各成员变量,最后用 printf 直接输出结构体变量 stu 各成员变量的值。

```
//编辑源程序代码 5-3.c
#include<stdio.h>
struct student
{
    int num;
    char name[8];
    char sex;
    float score[2];
}stu;
int main( )
{
    scanf("%d%s%c%f%f",&stu.num,&stu.name,&stu.sex,&stu.score[0],&stu.score[1]);
    printf("%d\t%s\t%c\t%f\t%f\n",stu.num,stu.name,stu.sex,stu.score[0],stu.score
[1]);
}
```

编译程序 5-3.c,编译成功后运行程序,程序的输入输出结果为:

2001　张华　M　86.00　92.2

2001　张华　0.000000　0.000000

**2. 结构体的比较**

两个结构体变量不能直接比较它们是否相等,也不能直接比较大小,所以一般都是对结构体变量内的某个成员进行比较。

比如,要判断结构体变量 a 的学号是否为 123456,则判断条件为:

```
if(strcmp(a.id, "123456")==0)……
```

要判断结构体变量 a 和 b 的 m1 是否相等,则判断条件为:

```
if(a.m1==b.m1)……
```

## 三、任务分析与实施

**1. 任务分析**

(1)从任务描述中,每位同学的信息包括:学号、姓名和计算机基础、程序设计及英语三门课程的成绩,可以使用结构体变量来存取。

(2)班级共有 30 位同学,因此可以定义结构体数组来存取 30 位同学的信息。

(3)在键盘输入 30 位同学的信息,可以用循环实现。

(4)按计算机基础课程成绩从小到大输出30位同学的信息,可以使用冒泡排序法或选择排序法进行排序。

(5)后面输入的学号要与前面30位同学的学号进行比较,可以用循环的方式,用后面输入的学号与前面输入的30位同学的学号一一进行比较,如果相匹配,就找到该位同学的成绩信息,然后输出。如果没有出现匹配,证明后面输入的学号与前面30位同学的学号都不相匹配,输出不相匹配提示信息。

**2. 任务实施**

```
#include "stdio.h"
//引入"string.h"头文件,函数 strcmp()在此头文件中声明定义
#include "string.h"
#define N 5 //为了程序运行方便,假设只有5个同学
//声明结构体类型 stu
struct stu
{
    char id[6];
    char name[10];
    int m1,m2,m3;
    };

void input_stu(struct stu s[])
{
   int i;
   //循环输入学生信息
   for (i=0;i<N;i++)
   {
       printf("请输入第%d个同学的记录:",i+1);
       scanf("%s%s",s[i].id,s[i].name);
       scanf("%d%d%d",&s[i].m1, &s[i]. m2,&s[i].m3);
   }
}

void sort_stu(struct stu s[])//按计算机基础课程成绩从高到低排序
{
    int i,j;
    struct stu t;
    for(i=0;i<N-1;i++)
    {
        for(j=0;j<N-1-i;j++)
        {
            if(s[j].m1<s[j+1].m1)
            {
                t=s[j];
```

```
                s[j]=s[j+1];
                s[j+1]=t;
            }
        }
    }
}

void output_stu(struct stu s[])
{
    int i;
    printf("排序后的成绩单为:\n");
    printf("学号\t 姓名\t 计算机基础\t 程序设计\t 英语\n");
    for(i=0;i<N;i++)
    {
        printf("%s\t%s",s[i].id,s[i].name);
        printf("\t%d\t\t%d\t\t%d\n",s[i].m1,s[i].m2, s[i].m3);
    }
}

void find_stu(struct stu s[])
{
    char findid[6]; //声明要查找的学号 findid
    int i;
    int flag = 1;//定义一个标记,并赋初值 1 表示假设没找到
    //循环输入学生信息
    printf("请输入要查找的学号\n");
    scanf("%s",&findid);
    for(i=0;i<N;i++)
    {
      //比较学号 findid 是否与 N 位同学中的学号相等
      if(strcmp(findid,s[i].id)==0)
      {
      flag = 0; //如果找到了,标记值赋为 0
          printf("学号为%s 同学的成绩信息:\n",findid);
          printf("%s,%s,%d,%d,%d\n",s[i].id,s[i].name,s[i].m1,s[i].m2,s[i].m3);
          break;
      }
    }
    if(flag == 1)
    {
       printf("该学号不存在\n");
    }
```

```
}


main()
{
        struct stu student[N];
        input_stu(student);
        sort_stu(student);
        output_stu(student);
        find_stu(student);
}
```

程序说明与测试

(1)函数 strcmp(参数 1,参数 2)是比较两个字符串的大小。如果两个字符串相同,返回值为 0;如果参数 1 字符串大于参数 2 字符串,返回值为 1,否则返回值为-1。

(2)程序中用到标记变量 flag,作用是表示找到还是没找到,初始标记值赋为 1,假设没找到,当找到与输入的学号相等的记录时,赋值为 0,表示已经找到。

(3)当找到有相同的学号时,表示已经查询到相关的学生成绩信息,应终止循环的执行。break 语句的功能就是退出循环。

(4)运行程序,输入学生的相关信息和查找的学号信息,为了更好的测试,需要使用存在和不存在的学号分别输入。

## 四、同步训练

**1. 填空题**

(1)若已定义:

```
struct num{int a;int b;float f;} n={1,3,5.789};
struct num *pn=&n;
```

则表达式 pn->b/n.a* ++pn->b 的值是_____,表达式(*pn).a+pn->f 的值是_____。

(2)以下程序段的运行结果分别是_____。

```
struct s{int a; float b; char *c;} x={18,83.5,"zhang"};
   struct s *px =&x;
   printf("%c %s\n",*px->c-1,&px->c[1]);
```

**2. 编程题**

使用结构体数组存放考生记录。考生信息包括:准考证号(register)、姓名(name)、性别(sex)、出生日期(birthday)、成绩(score[]5 门课程和总成绩)。

(1)编写 input 函数,实现考生数据的输入;

(2)编写 print 函数,实现考生数据的输出;

(3)编写 search 函数,找出考分最高的考生信息;

(4)编写 sort 函数,按准考证号从小到大排序输出考生信息。

# 任务 3　学生成绩信息的汇总统计

## 一、任务描述

从键盘中输入一个班级 30 位学生的学号、姓名及计算机基础、程序设计、英语三门课的成绩,计算出每位学生的总成绩和平均成绩,同时统计程序设计课程学生成绩大于或等于 60 分学生的个数。然后输出学生的成绩单(每条记录包含学号、姓名、计算机基础、程序设计、英语、总成绩和平均成绩)和程序设计课程成绩大于或等于 60 分的学生个数。

## 二、预备知识

**1. 指向结构体变量的指针**

根据指针所指数据类型,我们将指针分为不同的类别。结构体指针指向结构体变量,不能指向其他类型的数据。例如

```
struct student
{
    char id[10],name[20];
    float score;
};
struct student s, *p;
p=&s;
```

则定义了一个结构体指针变量 p,并使它指向结构体变量 s。

通过结构体指针访问结构体成员有两种方法:

(1)显式方式:(*指针变量).成员名,如(*p).name

(2)专用方式:指针变量->成员名,如 p->name

它们均等价于

结构体变量名.成员名

例如:

```
struct stu
{
    char id[10],name[20];
    float score1,score2,score3;
};
main()
{
    struct stu stud, *p;
    p=&stud;
    strcpy(p->id, "123456789");
```

```
    strcpy((*p).name, "王老五");
    p->score1=83;
    (*p).score2=87;
    stud.score3=92.5;
    printf("%s,%s,", (*p).id, p->name);
    printf("%6.2f,%6.2f,%6.2f\n",stud.score1,p->score2, p->score3);
}
```

程序运行结果如图 5.3 所示：

```
123456789,王老五, 83.00, 87.00, 92.50
```

**图 5.3　运行结果**

**2. 指向结构体数组的指针**

对于结构体 stu，如果有以下语句：

```
struct student fresh[50], *p;
p=fresh; //或 p=&fresh[0];
```

则定义了一个含 50 个元素的结构体数组 fresh 和一个指向结构体的指针变量，同时设置指针变量指向该数组的起始位置，即指向第 0 个元素，p 每增加 1，则指向下一个元素。即：

```
for(i=0;i<50;i++,p++)
  gets(p->id);          //或 gets((*p).id);或 gets(fresh[i].id);
```

表示输入 50 名新生的学号。而

```
for(p=fresh;p<fresh+50;p++)
   puts(p->name);
```

表示输出 50 名新生的名字。

**3. 指向结构体的指针作函数参数**

用结构体指针作函数参数比用结构体变量作函数参数效率高，因为这种方式无需传递各个成员的值，只需传递一个地址，且函数中的结构体成员并不占据新的内存单元，而与主调函数中的成员共享存储单元。同时还可以通过修改形参所指成员的值影响实参所对应的成员值。

例如：

```
struct book
{
     char bookname[30];
     int quantity;
};
main()
{
     void fun(struct book *p);
     struct book book1={"C 语言程序设计教程",10};
     printf("有%d 册《%s》\n", book1.quantity,book1.bookname);
     fun(&book1);
     printf("还有%d 册《%s》\n", book1.quantity,book1.bookname);
}
void fun(struct book *p)
{
```

```
    p->quantity --2;
}
```

运行结果为如图 5.4 所示：


图 5.4 运行结果

此外，还可以将结构体数组作为函数参数；也可以设计结构体指针型的函数，即返回的地址是指向结构体类型数据的。

## 三、任务分析与实施

**1. 任务分析**

(1)把一个人的有关信息组合起来，可以使用结构体来组织一个人的有关数据，再用结构化数组表示多个人的信息。为了优化结构化的处理，也可以定义指向结构体的指针来处理结构化数据。

(2)定义存放学生程序设计课程成绩大于或等于 60 分的个数变量，判断学生程序设计课程成绩是否大于或等于 60 分，如果成立则变量值加 1。

(3)求每个同学的总分、平均分。

**2. 任务实施**

```
#include <stdio.h>
#define N 5   //为了程序运行方便，假设只有 5 个同学
//声明结构体类型 stu
struct stu
{
   char id[10];
   char name[10];
   int m1,m2,m3;
float sum,avg;
};

//输入函数
void input_stu(struct stu s[])
{
   int i;
   struct stu *p=s;
   //循环输入学生信息
   for (i=0;i<N;i++,p++)
   {
   printf("请输入第%d 个同学的学号:",i+1);
   scanf("%s",p->id);
   printf("姓名:");
```

```
    scanf("%s",p->name);
    printf("三门课程成绩:");
    scanf("%d%d%d",&p->m1, &p->m2,&p->m3);
    }
}

void count_stu_m2(struct stu s[],int n)//统计并显示信息函数
{
    //count 记录程序设计课程成绩大于或等于 60 分学生个数,初始值为 0
    struct stu *p=s;
    int count = 0;
    printf("程序设计课程成绩大于或等于 60 分学生信息\n");
    for(i=0; i<N; i++,p++)
    {
        //判断学生程序设计课程成绩是否大于或等于 60 分
        if(p->m2>=60)
        {
        //学生程序设计课程成绩大于或等于 60 分的个数加 1
            count ++;
        }
    }
    printf("程序设计课程成绩大于或等于 60 分学生人数的个数为:%d\n",count);
}

void sum(struct stu s[])
{
    struct stu *p=s;
    int i;
    for(i=0;i<N;i++,p++)
    {
        p->sum=p->m1+p->m2+p->m3;
        p->avg=p->sum/3.0;
    }
}

//输出函数
void output_stu(struct stu *p,int n)
{
   int i;
   printf("他们的成绩单为:\n");
   //循环输出学生信息
   for(i=0;i<n;i++,p++)
```

```
    {
        printf("%s\t%s\t",p->id,p->name);
        printf("%d,%d,%d,%f,%f\n",p->m1, p->m2,p->m3,p->sum,p->avg);
    }
}

main()
{
    //声明结构体数组 student
    struct stu student[N],*p=student;
    input_stu(p);//调用输入函数录入数据
    count_stu_m2(p,N);//统计
    sum(p,N);//求总分和平均分
    output_stu(p,N);//调用输出函数显示结果
    system("pause");
}
```

程序说明与测试

(1)在输入函数中定义指向结构体指针 p,通过 p=s 把结构体数组 s 的地址赋给 p,即 p 指向结构体数组的第一个结构单元(第一个学生),p++将指向下一个单元(下一个学生)。通过指向结构体的指针访问结构体,采用结构指针运算符->(也称为箭头运算符,由减号"-"和大于号">"组成)。即如果结构体指针 p 指向结构体 s,则 p->id 相当于 s.id,当然也可以用(*p).id 访问结构体 s 的 id 成员。

(2)在统计和输出函数中使用结构体指针作为函数参数,同时为了让输出函数了解结构体数组的个数,增加一个整型参数。主函数以结构体数组作为实参,与结构体指针形参对应。

## 四、同步训练

1.以下程序定义一个结构体变量,然后通过调用函数更换它的值,其运行结果如图 5.5 所示。

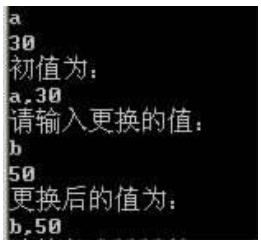


图 5.5 运行结果

请根据以上运行结果,将程序补充完整并调试。

(1)函数用结构体作为函数参数,返回结构体

```
struct st
{
    char name[30];
    int num;
};
struct st fun(struct sty)
{
    printf("\n 请输入更换的值:\n");
    scanf("%s",_____);
```

```
    _____;
    returny;
}
main()
{
    struct st x;
    gets(x.name);
    scanf("%d",&x.num);
    printf("初值为:\n%s,%d", x.name,x.num);
    _____;
    printf("更换后的值为:\n%s,%d\n", x.name,x.num);
}
```

(2)函数用结构体指针作为函数参数,无返回值

```
struct stru
{
    char name[30];
    int num;
};
void fun(struct stru *s)
{
    printf("\n请输入更换的值:\n");
    scanf("%s",_____);
    _____;
}
main()
{
    struct stru x;
    gets(x.name);
    scanf("%d",&x.num);
    printf("初值为:\n%s,%d", x.name,x.num);
    _____;
    printf("更换后的值为:\n%s,%d\n", x.name,x.num);
}
```

2.设计函数 int days(struct date *d),其中形参 d 是指向实参日期结构(由年、月、日三个 int 类型成员组成)变量的指针,函数返回值对应的本年度第几日。如果月份成员超出范围(1～12),则函数返回－1,如果日成员超出范围(1～28/29/30/31,考虑闰年),函数返回－2;自定义主函数进行测试。

# 任务 4　建立动态学生信息表

## 一、任务描述

利用链表,实现动态存储班级 N 位学生的学号、姓名及计算机基础课程的成绩信息,同时还能够实现插入和删除学生的信息运算。

## 二、预备知识

### (一)动态存储分配与链表的概念

**1. 内存单元的分配**

在 C 程序设计中,内存单元的使用是通过定义变量来进行的,若存储的数据量较大,则可定义数组或结构体。用这种方式使用内存,必须事先确定所需内存单元的多少,即变量的个数和数组的大小。对于数组,若需处理的数据个数不确定,则只能将数组定义得足够大,因此会造成内存的浪费。

其实,对内存的使用可以动态地进行。程序运行中需要内存时可临时分配内存单元,不用时可随时将其释放,使数据的存储和处理更加灵活和高效。这就是所谓的动态存储分配。

**2. 链表**

链表是实现动态存储分配的一种方法,它是一种不同于变量和数组的数据结构。

链表由若干结点构成,每一个结点含有两部分内容:数据部分和指针部分。数据部分是程序所需的,指针部分则存放下一个结点的地址。因此可以通过上一个结点访问下一个结点。此外,链表中设一个头指针变量(head),它指向第一个结点,并且末结点(又称为表尾)的指针部分存放的是“空地址”NULL,表示它不指向任何结点,链表到此结束。

单向链表示例如图 5.6 所示:

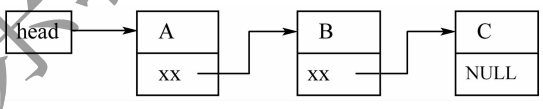


图 5.6　单向链表

与变量和数组不同,变量和数组的访问是随机的,只要给出变量名或数组元素名即可。访问链表中任何一个结点均须由头指针从第一个结点开始顺次进行,直至找到所需结点。

### (二)动态存储分配函数

真正实现动态存储分配,除了利用含指针成员的结构体之外,还需利用 C 语言提供的几个标准库函数(使用时应包含头文件“alloc.h”或“malloc.h”或“stdlib.h”)。

**1. malloc 函数**

开辟指定大小的存储空间,并返回该存储区的起始地址。函数原型为:

```
void *malloc(unsigned int size)
```

其中 size 为需要开辟的字节数。函数返回一个指针,该指针不指向具体的类型,当将该指针赋给具体的指针变量时,需进行强制类型转换。若 size 超出可用空间,则返回空指针值 NULL。例如:

```
float *p1;
int *p2;
p1=(float *)malloc(8);
*p1=3.14;
p2=(int *)malloc(20*sizeof(int));
for(i=0;i<20;i++)
scanf("%d",p2++);
```

分别开辟了 8 字节和 20 个 int 类型单元的存储空间，并向其中存入数据。

**2. calloc 函数**

按所给数据个数和每个数据所占字节数开辟存储空间。函数原型为：

```
void *calloc(unsigned int num, unsigned int size)
```

其中 num 为数据个数，size 为每个数据所占字节数，故开辟的总字节数为 num * size。函数返回该存储区的起始地址。

例如，上例中 p2 可改写为 p2=(int *)calloc(20, sizeof(int))；

例如：calloc(10,20) //开辟 10 个且每个大小均为 20 字节的存储空间。

**3. realloc 函数**

重新定义所开辟内存空间的大小。函数原型为：

```
void *realloc(void *ptr, unsigned int size)
```

其中 ptr 所指的内存空间是用前述函数已开辟的，size 为新的空间大小，其值可比原来大或小。函数返回新存储区的起始地址(该地址可能与以前的地址不同)。例如：

```
p1=(float *)realloc(p1,16);//将原先开辟的 8 个字节调整为 16 个字节
```

**4. free 函数**

将以前开辟的某内存空间释放。函数原型为：

```
void free(void *ptr)
```

其中 ptr 为存放待释放空间起始地址的指针变量，函数无返回值。应注意：ptr 所指向的空间必须是前述函数所开辟的。例如

```
free((void *)p1);
```

将上例开辟的 16 个字节释放。可简写为

```
free(p1);
```

由系统自动进行类型转换。

**(三)链表的建立、插入、删除和查找**

**1. 链表的建立**

(1)定义结点的组成

利用包含指针成员的结构体变量构成结点。例如：

```
struct student_score
{
    long num;
    float score;
    struct stud_score *next;
};
```

定义的结点中,前两个成员存放有效数据(学号、成绩),后一个成员存放下一个结点的地址。由于构成链表的所有结点类型均相同,故指针变量 next 的类型也是 struct student_score。

(2)定义结点

根据以上结点的组成,可定义结点实体,并输入数据。例如:

```
struct student_score stud1,stud2,stud3;
stud1.num=1261001;stud1.score=85;
stud2.num=1261002;stud2.score=95;
stud3.num=1261003;stud3.score=90.5;
```

(3)建立链表

将各个结点链接起来便构成一个链表。例如:

```
struct stud_score *head;
head=&stud1;
stud1.next=&stud2;
stud2.next=&stud3;
stud3.next=NULL;
```

(4)输出链表(访问链表)

通过上一个结点的指针项访问下一个结点。例如:

第一个结点的学号和成绩可表示为:

head->num,head->score

第二个结点的学号和成绩可表示为:

stud1.next->num,stud1.next->score 或

head->next->num,head->next->score

第三个结点的学号和成绩可表示为:

head->next->next->num,head->next->next->score

以上只是讲述建立链表的一个原理,并不具备动态的性质,因为结点的数目即结构体变量的数目是事先定义好的,增加新的结点或删除已有结点都要重新修改程序,因此不是一个通用的方法。

例如,以下程序实现建立一个具有 10 个结点的链表,每个结点的数据成员由一数组给出,并通过对每个结点的访问来显示链表中每个结点的数据。

```
#include<stdio.h>
struct node
{
    int data;
    struct node * next;
};
struct node nd[10];
int a[] = {1,3,5,7,9,11,13,15,17,19};
main()
{
    int i;
    struct node *head,*p;
    head = &nd[0];
    nd[0].data = a[0];
```

```
p = head;
for(i=1; i<10; i++)
    {
        nd[i].data = a[i];
        p->next = &nd[i];
        p=p->next;
    }
    p->next = NULL;
    p = head;
    while(p ! = NULL)
    {
        printf("%d",p->data);
        p = p->next;
    }
}
```

说明：由于链表首指针 head 一般不能改变，所以程序中使用指针 p 指向当前正在处理的结点。

**2. 在链表中查找指定元素**

在对链表进行插入或删除的运算中，总是首先需要找到插入或删除的位置，这就需要对链表进行扫描查找，在链表中寻找包含指定元素值的前一个结点。当找到包含指定元素的前一个结点后，就可以在该结点后插入新结点或删除该结点后的一个结点。

下面是在非空链表中寻找包含指定元素值的前一个结点的 C 语言描述。

```
struct node       /*定义结点类型*/
{
    ET d;       /*ET为数据元素类型名,下同*/
    struct node *next;
};
/*在头指针为 head 的非空链表中寻找包含元素 x 的前一个结点 p(结点 p 作为函数值返回) */
struct node *lookst(head, x)
ET x;
struct node *head;
{
    struct node *p;
    p=head;
    while((p->next! =NULL)&&(((p->next)->d)! =x))  p=p->next;
    return p;
}
```

在这个算法中，从头指针指向的结点开始往后沿指针进行扫描，直到后面已没有结点或下一个结点的数据域为 x 为止。因此，由这个算法返回的结点值 p 有两种可能；当链表中存在包含元素 x 的结点时，则返回的 p 指向第一次遇到的包含元素 x 的前一个结点，当链表中不存在

包含元素的结点时,则返回的 p 指向链表中的最后一个结点。

**3. 链表的插入**

链表的插入是指在原链表中的指定元素之前插入一个新元素。

为了要在链表中插入一个新元素,首先要给该元素分配一个新的结点,以便用于存储该元素的值。新结点可以用 malloc()函数申请,然后将存放新元素的结点链接到链表中指定的位置。

要在链表中包含元素 x 的结点之前插入一个新元素 b。其插入过程如下:

(1)用 malloc()函数申请取得新结点 p,并置该结点的数据域为 b。即令 p->data=b。

(2)在链表中寻找包含元素 x 的一个结点,设该结点的存储地址为 q,链表如图 5.7(b)所示。

(3)最后将结点 p 插入到结点 q 之后。为了实现这一步,只要改变以下两个结点的指针域内容:

①使结点 p 指向包含元素 x 的结点(即结点 q 的后件结点),即令

```
p->next = q->next
```

②使结点 q 的指针域内容改为指向结点 p,即令

```
q->next = p
```

这一步的结果如图 5.7(c)所示。此时插入就完成。

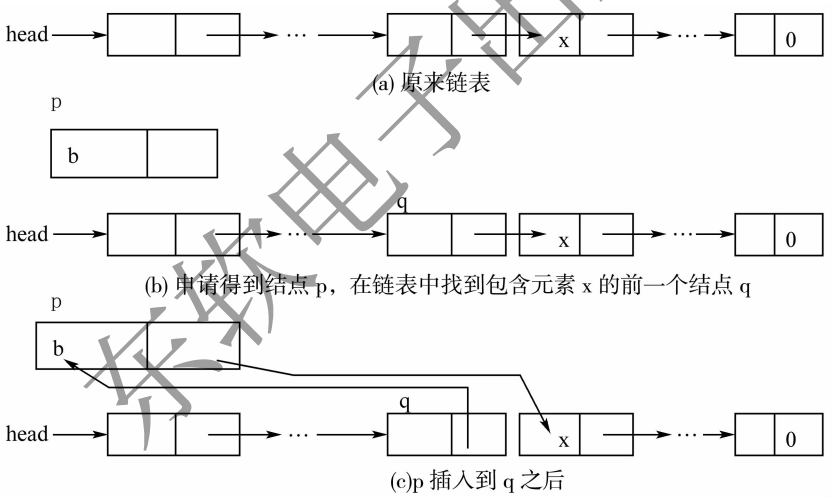


**图 5.7 链表的插入**

由链表的插入过程可以看出,链表在插入过程中不发生数据元素移动的现象,只需改变有关结点的指针即可,从而提高了插入效率。

下面结出在链表中包含元素 x 的结点之前插入新元素 b 的 C 语言描述。

```
#include<stdlib.h>
struct node     /* 定义结点类型 */
{
    ET d;     /* 数据元素类型 */
    struct node *next;
};
/* 在头指针为 head 的链表中包含元素 x 的结点之后插入新元素 b */
```

```
insIst(head, x, b)
ET x,b;
struct node * * head;
{
    struct node * p, * q;
    p=(struct node * )malloc(sizeof(struct node));//申请一个新结点 p
    p->d = b;//置结点的数据域
    if( * head==NULL)//链表为空
    {
     * head = p;
    p->next = NULL;
    return;
}
if(( * head->d)==x)//在第一个结点前插入
{
    p->next= * head;
            * head=p;
           return;
        }
        q = lookst( * head, x);//寻找包含元素 x 的前一个结点 q
        p->next = q->next;q->next = p; //结点 p 插入到结点 q 之后
        return;
    }
```

**4. 链表的删除**

链表的删除是指在链表中删除包含指定元素的结点。

为了在链表中删除包含指定元素的结点，首先要在链表中找到这个结点，然后将要删除的结点放回到可利用栈。

要在链表中删除包含元素 x 的结点。其删除过程如下：

(1)在链表中寻找包含元素 x 的前一个结点，设该结点地址为 q。则包含元素 x 的结点地址 p=q->next。

(2)将结点 q 后的结点 p 从链表中删除，即让结点 q 的指针指向包含元素 x 的结点 p 的指针指向的结点，即令

```
q->next = p->next
```

经过上述两步后，链表如图 5.8 所示。

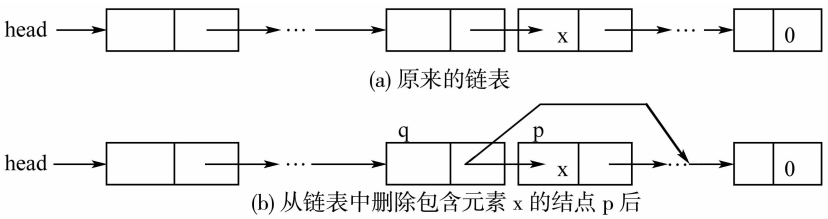


**图 5.8　链表的删除**

(3)将包含元素 x 的结点 p 释放。此时,链表的删除运算完成。

从链表的删除过程可以看出,在链表中删除一个元素后,不需要移动表的数据元素,只需改变被删除元素所在结点的前一个结点的指针域即可。另外,当从链表中删除一个元素后,该元素的存储结点就变为空闲,应将该空闲结点释放。

下面给出在链表中删除包含元素 x 的结点的 C 语言描述。

```
#include<stdlib.h>
struct node       /* 定义结点类型 */
{
    ET d;     /* 数据元素类型 */
    struct node *next;
};
delst(head, x)      /* 在头指针为 head 的链表中删除包含元素 x 的结点 */
ET x;
struct node **head;
{
    struct node *p, *q;
    if(*head==NULL)
    {
        printf("this is a empty list\n");
        return;
    }
    if((*head->d)==x)//删除第一个结点
    {
        p=*head->next; free(*head); *head=p; return;
    }
    q=lookst(*head, x); //寻找包含元素 x 的前一个结点 q
    if(q->next==NULL) //链表中没有包含元素 x 的结点
    {
        printf("No this node int the list\n");
        return;
    }
    p=q->next; q->next=p->next;
    free(p);
    return;
}
```

## 三、任务分析与实施

**1. 任务分析**

(1)一般情况下,各班级的人数是往往都不一样的,在编写处理学生成绩信息程序时,为了适用于各班级的信息处理,往往要把数组设置较大,虽然这样解决了这个问题,但也浪费了不少

的内存空间。更好的做法是根据不同班级人数的多少，在程序运行时动态分配内存空间。

(2)学生信息查询指定按学生学号，学生信息的查询，插入和删除实质是对链表进行查询，插入和删除运算。

**2. 任务实施**

```
#define NULL 0
#define SIZE sizeof(struct stu)
struct stu
{
    int xh;
    char name[10];
    float m1;
    struct stu *next;
} *head=NULL, *p, *q, *r;
void crate()
{
    int i;
    float f;
    printf("建立链表,按学号小到大输入元素,如 3 张三 88(-1\0 0 结束)\n");
    while(1)
    {
       r=(struct stu *)malloc(SIZE);
       scanf("%d%s%f",&r->xh,r->name,&f);
       r->m1 = f;
       if(r->xh<=-1)
       {
           free(r);
           break;
       }
       if(head==NULL)
       {
           head = r;
           p=r;
       }
       else
       {
           p->next=r;
           p=r;
       }
    }
    p->next = NULL;
}
void pr()
{
```

```
printf("\n*******学生信息*****\n");
    p=head;
    while(p! =NULL)
    {
        printf("%d%s%5.1f\n",p->xh,p->name,p->m1);
        p=p->next;
    }
}
void del()
{
    int xh;
    while(1)
    {
        if(head==NULL)
        {
            printf("\n链表已空\n");
            break;
    }
    printf("\n删除链表结点,请输入学号(-1结束):\n");
    scanf("%d",&xh);
    if(xh<=-1) break;
    q=head;
    while(xh>q->xh && q->next! =NULL)
    {
        p=q;
        q=q->next;
    }
    if(xh==q->xh)
    {
        if(q==head) head=q->next;
        else p->next = q->next;
        printf("成功删除%d号学生信息\n",xh);
        free(q);
    }
    else
        printf("%d号学生信息未找到! \n",num);
    }
    return;
}
void insert()
{
    int i;
```

```
    float f;
    while(1)
    {
        printf("\n插入链表结点,请输入学号,姓名,计算机基础成绩(-100)结束\n");
        r=(struct stu *)malloc(SIZE);
        scanf("%d%s%f",&r->xh,r->name,&f);
        r->ml=f;
        r->next=NULL;
        if(r->xh==-1) break;
        q=head;
        p=q;
        if(head==NULL)
        {
            head=r;
            q=r;
            p=r;
        }
        else if(r->xh<head->xh)
        {
            r->next=head;
            head=r;
        }
        else
        {
            while((r->xh>q->xh)&&(q->next!=NULL))
            {
                p=q;q=q->next;
            }
            if(r->xh<q->xh)
            {
                 p->next=r;
                r->next=q;
            }
            else
                q->next=r;
        }
    }
}
main()
{
    int n;
    while(1)
```

```
    {
        printf("\n\n\t\t\t 链表操作\n\n");
        printf("\t\t 1 建立链表  2 输出链表\n");
        printf("\t\t 3 删除链表 4 插入结点\n");
        printf("\t\t 退出\n");
        printf("\t\t\t 请选择(1－5):");
        scanf("%d",&n);
        if(n>4 || n<1) break;
        switch(n)
        {
            case 1: create();break;
            case 2: pr(); break;
            case 3: del(); break;
            case 4: insert(); break;
        }
    }
}
```

**3. 程序说明与测试**

删除要考虑多种情况,如:链表是空链表,或所找结点不在链表中,或在链表中分别是首结点,中间结点,尾结点。

## 四、同步训练

1. 分析程序的功能和运行结果

```
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>
main()
{   int j,n,*p;
    printf("enter number of array:");
    scanf("%d",&n);
    p=(int *)malloc(sizeof(int));
    printf("enter int of %d:",n);
    for(j=0;j<n;j++)
    {
        *(p+j)=j;
    }
    for(j=0;j<n;j++)
    {
```

```
        printf("%5d",*(p+j));
    }
    free(p);
}
```

2.下面的程序意在建立一条单向链表存储线性表(a1－a2－…－an),将程序中的空白处补充完整(程序通过 new node 取一个结构结构空间,前一个结构的指针指向新取得的结构地址,结构的值分别取 5,10,20,30,…)。

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int num;
    struct node *next;
} *head;
void printnode(node *p)
{
    while(p->next! =NULL)
    {    //输出 p 指向的数据
         printf("%5d",_____);
         p=_____;//指向下一个
    }
    printf("%5d\n",p->num);
}

main()
{
    node *p=new node;
    head=_____;//设置头指针
    p->num=5;
    for(int k=1;k<10;k++)
    {    //产生下一个结点
         p->next=_____;
         p=p->next;
    _____=k*10;
    }
    p->next=NULL;
    printnode(head);
}
```

3.编程建立一个带有头结点的简单键表,当输入的数据为 0 时结束链表的建立并依次输入各结点的值,然后删除指定数据的结点并输出链表以查看结果。(要求建立、删除和输出要用单独的函数,写出算法说明,有适当注释的程序清单,并说明测试思路及数据。)

# 知识扩展

## 一、共用体

C 语言编程处理某些问题时,需要用几种不同类型的变量存放到同一段内存单元中。也就是使用覆盖技术,几个变量互相覆盖。这种几个不同的变量共同占用一段内存的结构,在 C 语言中,被称作“共用体”类型结构,简称共用体(某些书籍中可能称之为“联合体”)。

一般定义形式

```
union 共用体名 { 成员表列 } 变量列表;
```

如:

```
union udata     //定义 udata 共用体
{
    int i;
    char ch;
    float f;
}a,b,c;  //定义 a,b,c 三个共用体变量
```

**1. 共用体变量的引用方式**

共用体变量也只有在定义后才能引用它,要注意的是,不能引用共用体变量,而只能引用共用体变量中的成员。如:

```
union udata
    {
    int i;
    char ch;
    float f;
}a,b,c;
```

对于这里定义的共用体变量 a,b,c,正确的引用方式为:

```
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
```

而以下的引用是错误的:printf("%d",a);

因为 a 的存储区内有好几种类型的数据,分别占用不同长度的存储区,这些共用体变量名 a,难以使系统确定究竟输出的是哪一个成员的值。

而应该写成 printf("%d",a.i);或 printf("%c",a.ch);

**2. 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,但是在每一瞬间只能存放其中的一种,而不是同时存放几种。换句话说,每一瞬间只有一个成员起作用,其他的成员不起作用,即不是同时都存在和起作用。

(2)共用体变量中起作用的成员是最后一次存放的成员,在存入一个新成员后,原有成员就失去作用。

(3)共用体变量的地址和它的各成员的地址都是同一地址。

(4)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。

(5)共用体类型可以出现在结构体类型的定义中,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的定义中,数组也可以作为共用体的成员。

## 二、枚举类型

在实际问题中,有些变量的取值被限定在一个有限的范围内。例如,一个星期内只有七天,一年只有十二个月,一个班每周有六门课程等等。如果把这些量说明为整型,字符型或其他类型显然是不妥当的。为此,C 语言提供了一种称为"枚举"的类型。在"枚举"类型的定义中列举出所有可能的取值,被说明为该"枚举"类型的变量取值不能超过定义的范围。应该说明的是,枚举类型是一种基本数据类型,而不是一种构造类型,因为它不能再分解为任何基本类型。

**1. 枚举的定义**

枚举类型定义的一般形式为:

```
enum 枚举名{ 枚举值表 };
```

在枚举值表中应罗列出所有可用值。这些值也称为枚举元素。

例如:enum weekday{Mon,Tue,Wed,Thu,Fri,Sat,Sun};

该枚举名为 weekday,枚举值共有 7 个,即一周中的七天。凡被说明为 weekday 类型变量的取值只能是七天中的某一天。

**2. 枚举变量的说明**

如同结构类型一样,枚举变量也可用不同的方式说明,即先定义后说明,同时定义说明或直接说明。

设有变量 a,b,c 被说明为上述的 weekday,可采用下述任一种方式:

```
enum weekday{ sun,mou,tue,wed,thu,fri,sat };
enum weekday a,b,c;
```

或者为:

```
enum weekday{ sun,mou,tue,wed,thu,fri,sat }a,b,c;
```

或者为:

```
enum { sun,mou,tue,wed,thu,fri,sat }a,b,c;
```

枚举类型在使用中有以下规定:

(1)枚举值是常量,不是变量。不能在程序中用赋值语句再对它赋值。

例如对枚举 weekday 的元素再作以下赋值都是错误的:

```
sun=5;
mon=2;
sun=mon;
```

(2)枚举元素本身由系统定义了一个表示序号的数值,从 0 开始顺序定义为 0,1,2…。如在 weekday 中,sun 值为 0,mon 值为 1,…,sat 值为 6。

如:

```
#include<stdio.h>
main()
{
```

```
    enum weekday { sun,mon,tue,wed,thu,fri,sat } a,b,c;
    a=sun;
    b=mon;
    c=tue;
    printf("%d,%d,%d",a,b,c);
}
```

说明：

只能把枚举值赋予枚举变量，不能把元素的数值直接赋予枚举变量。如：

```
    a=sum;
    b=mon;
```

是正确的。而：

```
    a=0;
b=1;
```

是错误的。如一定要把数值赋予枚举变量，则必须用强制类型转换。

如：

```
    a=(enum weekday)2;
```

其意义是将顺序号为 2 的枚举元素赋予枚举变量 a，相当于：

```
    a=tue;
```

还应该说明的是枚举元素不是字符常量也不是字符串常量，使用时不要加单、双引号。