



清华大学

腾讯会议 ID: 726 444 239/ 2021
Tsinghua University

计算机程序设计基础

第9讲 结构、联合和枚举

沈瑜 (010-62782951)
shenyu@tsinghua.edu.cn

清华大学电机系
2021.11.9





主要内容

- C程序设计举例
- 结构体struct
- 联合体union
- 枚举

参考教材： 第9章（暂时不看9.3、9.4节）





9.1 C程序设计举例

● 例1: 学籍卡信息

```
#include <stdio.h>
#include <string.h>
struct student
{
    char name[20];
    int sex;
    int age;
    float height;
    char addr[100];
};
```

```
int main()
{
    struct student zhang3, li4, wang5;
    //不能直接赋值! zhang3.name = "Zhang Hua" 是错误代码!
    strcpy( zhang3.name, "Zhang Hua");
    zhang3.sex='m';
    zhang3.age=28;
    zhang3.height=1.76;
    strcpy( zhang3.addr, "Park load No.16");
    li4=zhang3; //这句的效果是什么? 试试Watch一下如何?
    strcpy( li4.name, "Li Yong");
    li4.sex='f';
    li4.age=23;
    li4.height=1.68;
    strcpy( li4.addr, "Green House 5-503");
    strcpy( wang5.name, "Wang Jun");
    wang5.sex='m';
    wang5.age=25;
    wang5.height=1.70;
    strcpy( wang5.addr, "Street Peace No.253");

    printf("  name      sex  age  address\n");
    printf("-----\n");
    printf("%-14s%-4c%-4d%-25s\n",
           zhang3.name, zhang3.sex, zhang3.age, zhang3.addr);
    printf("%-14s%-4c%-4d%-25s\n",
           li4.name, li4.sex, li4.age, li4.addr);
    printf("%-14s%-4c%-4d%-25s\n",
           wang5.name, wang5.sex, wang5.age, wang5.addr);
    return 0;
}
```

C:\windows\system			
name	sex	age	address
Zhang Hua	m	28	Park load No.16
Li Yong	f	23	Green House 5-503
Wang Jun	m	25	Street Peace No.253

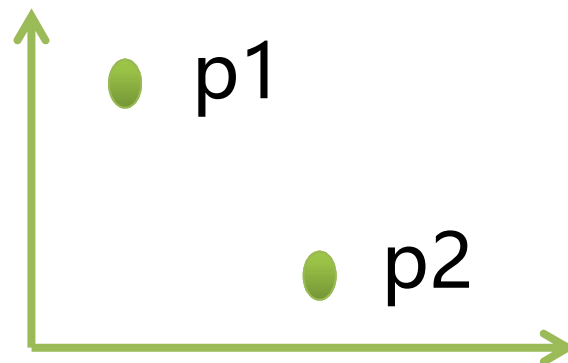


●例2: 求平面上的两点间距离

```
#include <stdio.h>
#include <math.h>
typedef struct
{
    int x, y;
} POINT;

double CalcDistance(POINT p1, POINT p2)
{
    double dx=p2.x-p1.x;
    double dy=p2.y-p1.y;
    return sqrt(dx*dx+dy*dy);
}

int main()
{
    POINT p1={5, 7};
    POINT p2={2, 3};
    printf("distance=%lf\n", CalcDistance(p1, p2) );
    return 0;
}
```





●例3: 复数操作

```
#include <stdio.h>
#include <math.h>
//声明
struct my_complex_struct{ double x, y;};
typedef my_complex_struct my_complex;

//实现
double real_part(my_complex z){ return z.x; };
double img_part(my_complex z){ return z.y; };
double magnitude(my_complex z){ return sqrt(z.x*z.x+z.y*z.y); };
double angle(my_complex z){ return atan2(z.y, z.x); };

//构造复数
my_complex make_from_real_img( double x, double y )
{
    my_complex z;
    z.x = x;
    z.y = y;
    return z;
}

my_complex make_from_mag_ang( double r, double A )
{
    my_complex z;
    z.x = r*cos(A);
    z.y = r*sin(A);
    return z;
}
```





//复数加减

```
my_complex add_my_complex( my_complex z1, my_complex z2 )  
{  
    return make_from_real_img( real_part(z1)+real_part(z2), img_part(z1)+img_part(z2) );  
}
```

```
my_complex sub_my_complex( my_complex z1, my_complex z2 )  
{  
    return make_from_real_img( real_part(z1)-real_part(z2), img_part(z1)-img_part(z2) );  
}
```

//复数乘除

```
my_complex mul_my_complex( my_complex z1, my_complex z2 )  
{  
    return make_from_mag_ang( magnitude(z1)*magnitude(z2), angle(z1)+angle(z2) );  
}
```

```
my_complex div_my_complex( my_complex z1, my_complex z2 )  
{  
    return make_from_mag_ang( magnitude(z1)/magnitude(z2), angle(z1)-angle(z2) );  
}
```

```
int main()  
{  
    my_complex z1={3.1, 4.1}, z2={0.1, 0.1};  
    my_complex z3, z4;  
    z3 = sub_my_complex( z1, z2 );  
    z4 = mul_my_complex( z1, z2 );  
    return 0;  
}
```





● 例4: 四季风景

```
#include <stdio.h>

enum Season{ Spring=1, Summer, Autumn, Winter };

int main() {
    enum Season k;
    printf("Plz input:");
    scanf("%d", &k);
    if(k==Spring)
        printf("春花");
    else if(k==Summer)
        printf("夏风");
    else if(k==Autumn)
        printf("秋月");
    else if(k==Winter)
        printf("冬雪");

    printf("\n");
    return 0;
}
```





预备知识

1. typedef 关键字

- 用法：
 - **typedef** 现有类型 新名字;
 - 举例:

```
void main() {  
    typedef int INTEGER;  
    INTEGER i, sum = 0;  
    for ( i=1; i<=100; i++) sum += i;  
    printf("sum = %d\n",sum);  
}
```

- 作用：
 - 为现有类型创建一个新的名字
 - 用于编写更美观、可读性更强的代码



2. sizeof关键字

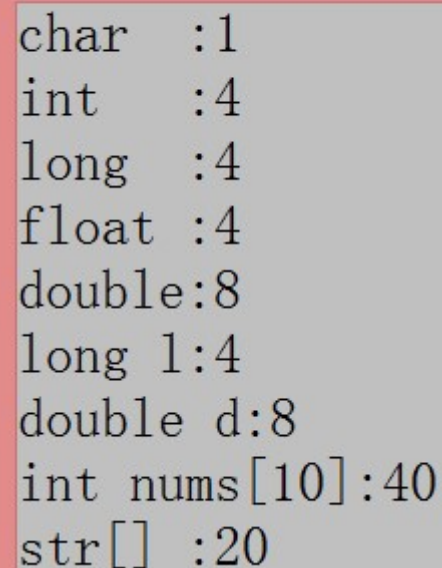
用法: **sizeof** (变量名或者类型)

返回内存大小

```
#include<stdio.h>

int main()
{
    int x;
    char y;
    float z;
    long l;
    double d;
    int nums[10];
    char str[20];
    printf("char   :%d\n", sizeof(char));
    printf("int    :%d\n", sizeof(int));
    printf("long   :%d\n", sizeof(long));
    printf("float  :%d\n", sizeof(float));
    printf("double:%d\n", sizeof(double));
    printf("long l:%d\n", sizeof(l) );
    printf("double d:%d\n", sizeof(d) );
    printf("int nums[10]:%d\n", sizeof(nums) );
    printf("str[]  :%d\n", sizeof(str) );

    return 0;
}
```



```
char   :1
int    :4
long   :4
float  :4
double:8
long l:4
double d:8
int nums[10]:40
str[]  :20
```

sizeof关键字

sizeof是一个特殊的编译预处理，不能看成函数、也不是一元操作符。

- **sizeof**是在编译阶段求值的。
- ```
int a=0;
printf(“%d”,sizeof(a=3));
printf(“%d”, a);
```
- 输出结果是**4,0**而不是我们期望的**4,3**。问题在于**sizeof**在编译阶段处理的特性，**sizeof**不能被编译成机器码，所以**sizeof**作用范围内，也就是**( )**里面的内容也不能被编译，而是被替换成类型。
- **a=3**相当于**int**，而代码也被替换为

```
int a=0;
printf(“%d”, 4);
printf(“%d”, a);
```





## 9.2 结构struct

### 1. 结构的概念 (structure)

- ◆ struct是C语言中的构造类型，是由不同数据类型的数据组成的集合体。
- ◆ 为处理复杂的数据结构提供了手段。
- ◆ 为函数间传递不同类型的参数提供了便利。



## 2. 结构类型的声明

**struct {**

成员变量说明1;

成员变量说明2;

.....

**};**

→ 空白学籍卡

```
struct student
{
 char name[20];
 int sex;
 int age;
 float height;
 char addr[100];
};
```

这里的分号;  
一定不能丢!

学籍卡

姓名: 张三

性别: 男

年龄: 20

身高: 1.70 m

住址: 清华园1号

各科成绩.....



### 3. 结构变量的定义

一种数据类型

- 变量定义形式

- **struct** 结构名 结构变量名;

- 注意:

- 结构变量的存储类型概念、它的寿命、可见性及使用范围与普通变量、数组等完全一致。

- 结构变量定义必须在结构类型声明之后，二者也可同时进行。

- 结构变量占内存大小可用 **sizeof** 运算求出:

**sizeof(运算量)**

其中，运算量可以是 变量名，也可以是类型名。

```
struct student zhang3;
```

```
typedef struct student STUDENT;
STUDENT zhang3;
```



## 4. 结构变量的使用和初始化

- 使用形式

- 一般情况下不能把结构体变量作为一个整体参加数据处理。
- 结构体成员的引用形式：结构变量名.成员名
- “.”成员运算符优先于所有运算符：  
**\*zhang.name 等同于 \*(zhang.name)**

- 初始化

```
struct student
{
```

```
 char name[20];
 int sex;
 int age;
 float height;
 char addr[100];
```

```
};
```

```
struct student
```

```
 wang5={"Wang Fang", 'F', 20, 1.70, "清华园1号"};
```

初始化，不是赋值

## 结构变量的使用:

### ◆ 成员访问

### ◆ 整体复制

数组不能=运算符复制

结构可以=运算符复制  
(包括其中的数组)

```
int main()
{
 struct student zhang3, li4, wang5;
 //不能直接赋值! zhang3.name = "Zhang Hua" 是错误代码!
 strcpy(zhang3.name, "Zhang Hua");
 zhang3.sex='m';
 zhang3.age=28;
 zhang3.height=1.76;
 strcpy(zhang3.addr, "Park load No.16");
 li4=zhang3; //这句的效果是什么? 试试Watch一下如何?
 strcpy(li4.name, "Li Yong");
 li4.sex='f';
 li4.age=23;
 li4.height=1.68;
 strcpy(li4.addr, "Green House 5-503");
 strcpy(wang5.name, "Wang Jun");
 wang5.sex='m';
 wang5.age=25;
 wang5.height=1.70;
 strcpy(wang5.addr, "Street Peace No.253");

 printf(" name sex age address\n");
 printf("-----\n");
 printf("%-14s%-4c%-4d%-25s\n",
 zhang3.name, zhang3.sex, zhang3.age, zhang3.addr);
 printf("%-14s%-4c%-4d%-25s\n",
 li4.name, li4.sex, li4.age, li4.addr);
 printf("%-14s%-4c%-4d%-25s\n",
 wang5.name, wang5.sex, wang5.age, wang5.addr);
 return 0;
}
```

## 结构变量的使用:

### ◆ 成员访问

### ◆ 整体复制

### ◆ 函数间的传递

#### ◆ 参数

#### ◆ 返回值

(更好的办法是采用结构指针, 而不是利用“副本”)

```
//复数加减
my_complex add_my_complex(my_complex z1, my_complex z2)
{
 return make_from_real_img(real_part(z1)+real_part(z2), img_part(z1)+img_part(z2));
}

my_complex sub_my_complex(my_complex z1, my_complex z2)
{
 return make_from_real_img(real_part(z1)-real_part(z2), img_part(z1)-img_part(z2));
}

//复数乘除
my_complex mul_my_complex(my_complex z1, my_complex z2)
{
 return make_from_mag_ang(magnitude(z1)*magnitude(z2), angle(z1)+angle(z2));
}

my_complex div_my_complex(my_complex z1, my_complex z2)
{
 return make_from_mag_ang(magnitude(z1)/magnitude(z2), angle(z1)-angle(z2));
}

int main()
{
 my_complex z1={3.1, 4.1}, z2={0.1, 0.1};
 my_complex z3, z4;
 z3 = sub_my_complex(z1, z2);
 z4 = mul_my_complex(z1, z2);
 return 0;
}
```



单选题 1分



```
struct abc
{
 int a,b,c;
};
```

程序段

```
struct abc s[3]={{1,2,3},{4,5,6},{7,8,9}};
```

```
int t;
```

```
t=s[1].a+s[2].b;
```

```
printf("%d\n",t);
```

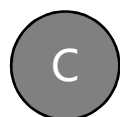
的输出为:



5



6



7



11



12



13

提交



## 5. 结构数组

- 具有相同结构体变量的结构体也可以组成数组

**160张学籍卡 → 数组**

- 结构数组的说明形式

**struct 结构名 结构数组名[元素个数];**

- 结构数组初始化

**struct 结构名 结构数组名[]={初始数据};**

```
struct student stu[3]=
{
 {"wang", 'M', 22, 1.7, "清华园1号"},
 {"zhang", 'F', 23, 1.7, "清华园1号"},
 {"li", 'M', 24, 1.7, "北大**楼"}
};
```



## 6. 结构嵌套

- 结构的成员项是结构
- 参加运算和操作的只能是最内层结构的成员项
- 内层结构成员的引用形式：
  - 结构变量名.外层成员名.内层成员名

```
struct date
```

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

```
struct student
```

```
{
 char name[20];
 char sex;
 struct date birthday;
};
```

```
struct student wang2;
strcpy(wang2.name,"wang2");
wang2.sex='M';
wang2.birthday.year=1980;
wang2.birthday.month=3;
wang2.birthday.day=6;
```



## 9.3 联合union

### 1. 联合的概念（union）

◆ union是C语言中的构造类型，是由几个不同数据变量共享同一段内存的数据类型。

### 2. 定义形式：

```
union 联合名
{ 数据类型 成员名 1;
 数据类型 成员名 2;
 :
};
```

### 3. 引用形式：

— 联合名.成员名





## 4. union与struct的区别

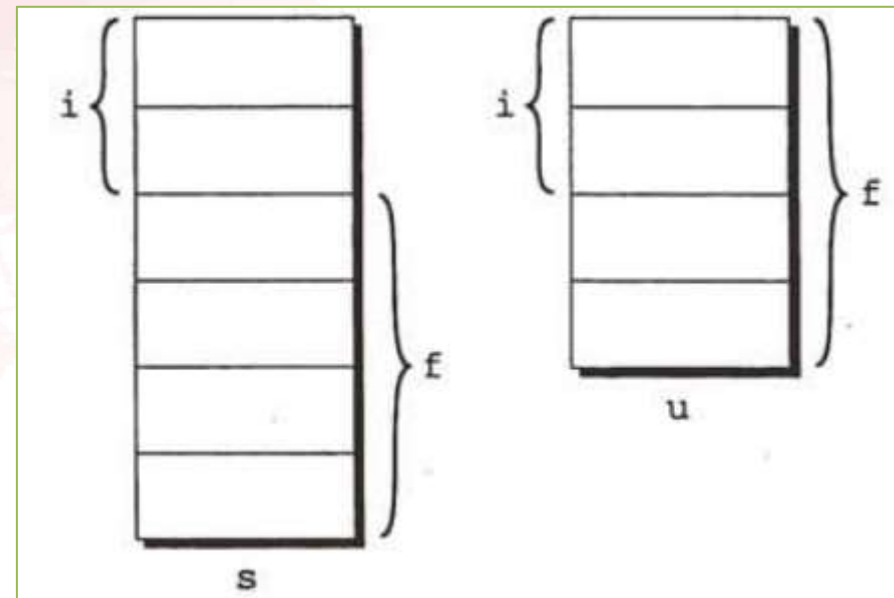
- 编译器只为联合中最大成员分配足够空间
  - 联合的成员在这个空间内彼此覆盖

```
struct {
 short int i;
 float f;
} s;
```

结构

```
union {
 short int i;
 float f;
} u;
```

联合



结构

联合

## union与struct的区别

- 声明方法、操作方法均与结构一致
- 联合的重要应用场合：
  - 节省空间
  - 构造混杂数据结构





## 9.4 枚举enum

### 1. 枚举的概念 (enum)

- 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



# 枚举

- 声明枚举类型用**enum**开头。
- 例如：

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

枚举元素

- 声明了一个枚举类型**enum Weekday**
- 然后可以用此类型来定义变量

枚举变量

```
enum Weekday workday,weekend;
```

workday=mon;      正确

weekend=sun;      正确

workday=monday;    不正确

```
enum Week_day2{ sunday,monday,tuesday,wednesday,thursday};
```



# 枚举

- 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



# 枚举

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

- 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: **weekday=mon;**  
相当于**weekday=1;**

- 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}weekday,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**。



# 枚举

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

- 说明:

**(3)** 枚举元素可以用来作判断比较。例如:

```
if(workday==mon)...
```

```
if(workday>tue)...
```

- 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**



# 枚举

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

- 说明:

## (3) 枚举变量的++运算

- C语言支持
- C++编译提示出错

```
enum Weekday workday,weekend;
workday++; //C语言可以，C++出错
```

Visual Studio中通常按C++编译，报错



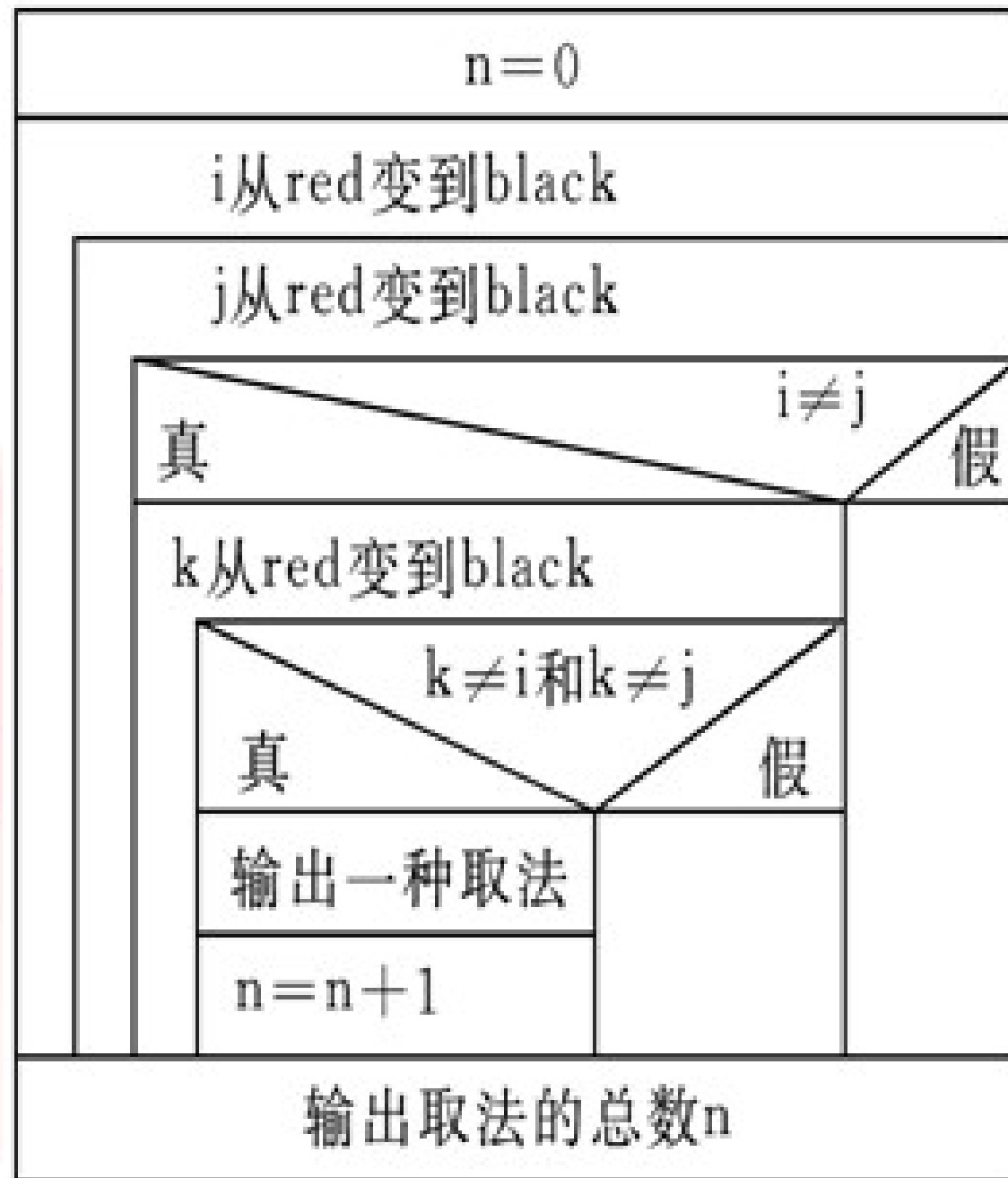
## 例 p325 例9.12

- 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 (输出)

| 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<br>"red"                | 输出<br>"yellow"             | 输出<br>"blue"               | 输出<br>"white" | 输出<br>"black" |



## 例 p325

```
enum Color {red,yellow,blue,white,black};
```

```
//声明枚举类型enum Color
```

```
// enum Color i,j,k,pri; //注释掉该行，改为下一行
```

```
int i,j,k,pri; //定义枚举变量i,j,k,pri
```

原因：Visual Studio中通常按C++编译，不支持枚举变量++操作

不能用 `printf("%s", red);`

必须： `if(i==red) printf("red");`



```

for (loop=1;loop<=3;loop++) //先后对三个球分别处理
{
 switch (loop) //loop的值从1变到3
 {
 case 1: pri=i;break; //loop的值为1时，把第1球的颜色赋给pri
 case 2: pri=j;break; //loop的值为2时，把第2球的颜色赋给pri
 case 3: pri=k;break; //loop的值为3时，把第3球的颜色赋给pri
 default:break;
 }
 switch (pri) //根据球的颜色输出相应的文字
 {
 case red:printf("%-10s","red"); break; //pri的值等于枚举常里red时输出“red”
 case yellow: printf("%-10s","yellow"); break; //pri的值等于枚举常里yellowd时输出“yellow”
 case blue: printf("%-10s","blue"); break; //pri的值等于枚举常里blue时输出“blue”
 case white: printf("%-10s","white"); break; //pri的值等于枚举常里white时输出“white”
 case black: printf("%-10s","black"); break; //pri的值等于枚举常里black时输出“black”
 default :break;
 }
}
printf("\n");

```

|    |        |        |        |
|----|--------|--------|--------|
| 1  | red    | yellow | blue   |
| 2  | red    | yellow | white  |
| 3  | red    | yellow | black  |
| 4  | red    | blue   | yellow |
| 5  | red    | blue   | white  |
| 6  | red    | blue   | black  |
| 7  | red    | white  | yellow |
| 8  | red    | white  | blue   |
| 9  | red    | white  | black  |
| 10 | red    | black  | yellow |
| 11 | red    | black  | blue   |
| 12 | red    | black  | white  |
| 13 | yellow | red    | blue   |
| 14 | yellow | red    | white  |
| 15 | yellow | red    | black  |
| 16 | yellow | blue   | red    |
| 17 | yellow | blue   | white  |
| 18 | yellow | blue   | black  |
| 19 | yellow | white  | red    |
| 20 | yellow | white  | blue   |
| 21 | yellow | white  | black  |
| 22 | yellow | black  | red    |
| 23 | yellow | black  | blue   |
| 24 | yellow | black  | white  |
| 25 | blue   | red    | yellow |
| 26 | blue   | red    | white  |
| 27 | blue   | red    | black  |
| 28 | blue   | yellow | red    |
| 29 | blue   | yellow | white  |
| 30 | blue   | yellow | black  |