第9章 用户自己建立数据类型

9.5 共用体类型

- 每到学期期中前后，课程教学进度过半，学校教学管理部门将开展学生对任课教师(教师1、教师2、教师3等)及教学辅助部门(包括教务部、学工部、后勤部、医务部、网络部、图书馆、保卫部等)的评价，请编写一个程序实现如下主要功能：

  (1)评价对象信息录入

  (2)学生评价

  (3)对评价信息进行统计与分析

  (4)评价结果展示

- 分析
  - ◆评价对象包含**3**项信息：名称、类别、评分
  - ◆评价对象有两类：任课教师和教学服务部门
  - ◆对任课教师的评价以等级实现
    - ●**excellent**、**good**、**unqualified**
  - ◆对教学服务部门的评价以打分为主**(0~10)**
  - ◆最后会得到一个总评表，形式如下：

名称	类别	评分
教师A	'J'	excellent
教务部	'S'	8.76
教师B	'J'	good
后勤部	'S'	7.98
图书馆	'S'	7.88
医务部	'S'	9.12

- 显然，评价对象应使用结构体类型，其形式应该如下：

```
struct Checker{
    T1 name;
    T2 category;
    T3 result;
};
```

- 评价对象的名称，显然应该是一个字符串，故应声明为：

```
char name[20];
```

- 评价对象类型，由上表，也很明确，用字符标注即可：

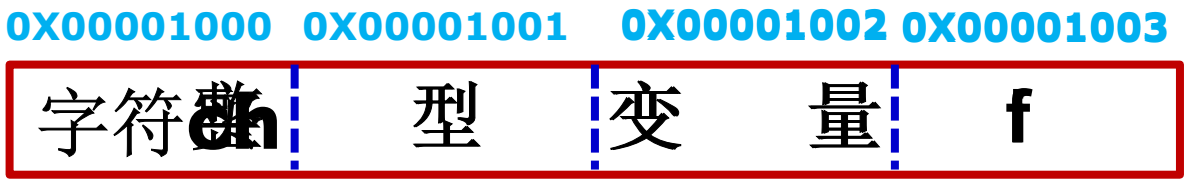
```
char category;
```

- 评价结果呢？有时候是个字符串，有时候是个浮点数

2个(类)值中的一个(类)，不能同时具有2个(类)值

9.5.1 什么是共用体类型

- 同一段内存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。



- 声明共用体类型的一般形式：

```
union  identifier_opt {
    members_list;
};
```

可省略

声明结束，不可省略

```
union  Result {
    char grade[20];
    float score;
} ;
```

共用体类型名

匿名共用体类型

9.5.2 引用共用体变量的方式

- 引用共用体变量及成员的方法同结构体相同，先定义共用体类型的变量，然后再使用共用体变量及其成员
  - **union Result a, b;**
  - **union { char grade[20]; float score;} a;**
- 同结构体相同，可以定义共用体类型的数组、指针等
  - **union Result arr[10], a, *p, *q;**
  - **p = &a; q = arr;**
- 成员运算符
  - **. a.grade, a.score**
  - **-> p->grade, q->score**

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。

  (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**(**为共用体的第一个成员初始化**)**。

  (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。

  (4) 共用体变量的地址和它的各成员的地址都是同一地址。

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：

  (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。

  (6) 同结构体一样，可以定义共用体数组、指针，共用体变量、数组、指针均可作为函数参数。

  (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。

- 引例的程序设计

```
#include <stdio.h>
//声明存储考核等级的共用体类型
union Result{
        char grade[20];
        float score;
};
//声明表示被考核对象的结构体类型
struct Checker{
        char name[20];
        char category;
        union Result rst;
};
```

```
void input(struct Checker *p, int n){
  for(int i=0; i<n; i++)
  {
    scanf("%s %c", p[i].name, &p[i].category);
    if(p[i].category == 'J')
       scanf("%s", p[i].rst.grade);
    else scanf("%f", &p[i].rst.score);
  }
}//输入被考核对象信息
void output(struct Checker *p, int n){
  for(int i=0; i<n; i++)
  {
    printf("%15s%3c", p[i].name, p[i].category);
    if(p[i].category == 'J')
      printf("15%s\n", p[i].rst.grade);
    else printf("%10.2f\n", p[i].rst.score);
  }
}//输出被考核对象信息
```

实际由评价人填写

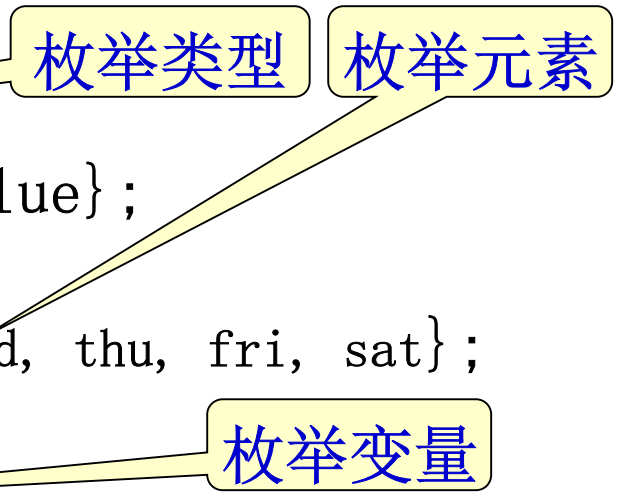
```
int main()
{
    struct Checker a[5];
    input(a, 3);
    ......//各种操作
    output(a, 3);
    return 0;
}
```

9.6 使用枚举类型

- 引子：学院在每学年第一学期接近期末的时候，往往会组织一次文艺晚会，晚会期间有一摇奖环节：摇奖池中有红、黄、蓝、白、黑5种颜色的球若干个，摇奖时摇奖机依次从奖池中摇出3个不同颜色的球。入场时，每从入口领取一张特制的奖券，并在监督人员的监督下依次填写红黄蓝白黑五种颜色中的三种（少于多于3种以及有涂改痕迹，兑奖无效）。问每次摇奖时，每人的中奖概率多大？
- 分析：
  - ◆找到摇奖结果所有可能的情况即可
  - ◆球的颜色是有限的
  - ◆数学上的排列问题
  - ◆球的颜色怎么表示呢？

9.6 使用枚举类型

- 程序设计时，如果把颜色作为一个变量，那么这个变量只有几种可能的值，并且所有可能的值是已知的，那么就可以把所有可能的值一一列举出来，则变量的值仅限于列举出来的值的范围。
- 这种把变量可能的值一一列举出来，作为一种自定义的数据类型，这种数据类型称为“枚举类型”。
- 声明枚举类型(enumeration)使用关键字enum，声明方式同结构体、共用体类似。如：
  - ◆颜色
    - ●enum ColorRGB{red, green, blue};
  - ◆星期
    - ●enum Weekday{sun, mon, tue, wed, thu, fri, sat};
  - ◆定义枚举变量
    - ●enum ColorRGB clr;



➢ 注意事项:

(1) C编译器对枚举类型的枚举元素按常量处理，故称枚举常量，不能对它们赋值。例如:

sun=0; mon=1; 错误

(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**。

(3) 枚举变量的值应使用枚举元素，如**workday=mon;**

(4) 枚举元素的值可以用来做比较，如**mon>sun**为真

```
//声明枚举类型：颜色
enum Color{
   red=6, yellow=1, blue, white, black
};
//获得特定颜色所对应的字符串
const char* getColorString(enum Color clr)
{
   switch(clr){
        case red: return "red";
        case yellow: return "yellow";
        case blue: return "blue";
        case white: return "white";
        case black: return "black";
   }
}
```

```
#include<stdio.h>
int main()
{
   int total = 0;
   enum Color i, j, k;
   for(i=red; i<=black; i++){
     for(j=red; j<=black; j++){
       if(j != i){
          for(k=red; k<=black; k++){
            if(k!=i && k!=j){
              total++;
              printf("%-8s%-8s%-8s\n", getColorString(i),
                        getColorString(j), getColorString(k));
             }
           }
       }
     }
   }
   printf("Total:%d cases.\n", total);
   return 0;
}
```

9.7 用typedef声明新类型名

struct Student, union Result, enum Color
是数据类型的名字，而不是**Student, Result, Color**

int *p1, *p2;

缺点

(1) 名字长，不好记，也不容易理解

(2) 容易出错

解决方法：

使用**typedef**为一个已经存在的数据类型起一个新的名字**(**同义词**synonym)**，从而将复杂的类型简单化，晦涩的类型清晰化，模糊的类型明确化。

用法：

typedef ExistedType NewType;

9.7 用typedef声明新类型名

typedef int Integer;
int a, b; 等价于**Integer a, b;**
typedef float Real;
float x, y; 等价于**Real x, y;**
typedef struct Student STU;
struct Student s1, s2;等价于**STU s1, s2;**
typedef int Num[10];
int a[10];等价于**Num a;**
typedef char* STRING;
char *s; 等价于**STRING s;**
typedef T (*Pointer)(T1, T2, ..., Tn);
T (*p)(T1, T2, ..., Tn);等价于**Pointer p;**

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（如将**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