- **常量符号化**
  用符号而不是具体的数字来表示程序中的数字
  用枚举而不是定义独立的const int 变量

```
enum COLOR {RED,YELLOW,GREEN};
```

枚举是一种用户定义的数据类型，它用关键字enum以如下语法来声明

```
enum 枚举类型名字{名字0,....名字n};
```

枚举类型名字通常并不真正使用，要用的是大括号里的名字，因为它们就是常量符号，它们的类型是int，值依次是从零到n
创建三个常量red是0，yellow是1，green是2
当我们需要一些可以排列起来的常量值时，定义枚举的意义就是给这些常量以名字

- **自动计数的枚举**

```
enum color{red,yellow,green,numColor};
int main(){
    int color = -1;
    char *colorNames[numColor] = {
        "red","yellow","green",
    };
    char *colorName = NULL
    printf("请输入你喜欢的颜色的代码");
    scanf("%d",&color);
    if(color>=0&&color<numColor){
        colorName = colorNames[color];
    }else{
        colorName = "unknown";
    }

    printf("你喜欢的颜色是%s\n",colorName);

    return 0;
}
```

在枚举后面添加一个计数，这样需要遍历所有的枚举量，或者需要一个用枚举量做下标的数组的时候会很方便

- **枚举量指定值**

```
enum color{red=1,yellow,green=5};
```

- **枚举只是int**
  枚举只是int，即使给枚举类型的变量赋不存在的整数值也没有任何的warning或者error
  虽然枚举类型可以当作类型来使用，但是不好用，如果有意义上排比的名字，用枚举比const int方便，枚举比宏（macro）好，因为枚举有int类型
- **结构类型**

```
#include <stdio.h>
int main(int argc,char *const argv[]){
    struct date{
        int month;
        int day;
        int year;
    }; //注意这儿是有分号的

    struct date today;
    today.month = 07;
    today.day = 31;
    today.year = 2014;

    printf("today's date is %i-%i-
%i.\n",today.month,today.day,today.year);
}
```

%i和%d都是表示有符号十进制整数，但%i可以自动将输入的八进制（或者十六进制）转换为十进制，而%d则不会进行转换。

- **在函数内外?**
  struct date{} 和本地变量一样，如果在函数内部声明的结构类型，只能在函数内部使用，所以通常在函数外部声明结构类型，这样就可以被多个函数使用了
- **声明结构的形式**

```
struct point{
    int x;
    int y;
```

```
};

struct point p1,p2p;
p1和p2都是point，里面有x，y值


struct{
    int x;
    int y;
} p1,p2;
p1和p2都是一种无名结构，里面有x和y


struct point{
    int x;
    int y;
} p1,p2;
```

- **结构变量初始化**

```
struct today = {07,31,2014};
struct thisdate = {.month = 07,.year = 2014};
给的值会填进去，没给的填0
```

- **结构成员**
  结构和数组有点像
  数字用[]运算符和下标访问其成员
  结构用.运算符和名字访问其成员
  today.day
  student.firstName
  注意区分结构类型和结构变量
- **结构运算**
  要访问整个结构，直接用结构变量的名字
  对于整个结构，可以做赋值、取地址，也可以传递给函数参数

```
p1 = (struct point){1,5}; //相当于p1.x=1;p1.y=5;
//前面括号中是强制类型转换
p1 = p2;//相当于p1.x=p2.x;p1.y=p2.y;
```

- **结构指针**

和数组不同，结构变量的变量名并不是结构变量的地址，必须用&运算符

```
struct date *pDate = &today;
```

- **结构作为函数的参数**

```
int numberOfDays(struct date c){

}
```

整个结构可以作为参数的值传入函数
这个时候我们是在函数内新建了一个结构变量，并复制调用者的结构的值，也可以返回一个结构

- **输入结构**
  没有直接的方式可以一次性scanf一个结构，
  如果我们打算写一个函数，这个函数可以读入一个结构
  但是读入的结构如何送回来，记住c在函数调用时是传值的
  记住c在函数调用时是传值的，
  解决办法：
  一个是在这个输入函数中，完全可以创建一个临时的结构变量，然后把这个结构返回给调用者

```
struct point{
    int x;
    int y;
};

struct point getStruct(void){
    struct point p;
    scanf("%d",&p.x);
    scanf("%d",&p.y);
    printf("%d,%d",p.x,p.y);
    return p;
}

在调用时
y=getStruct();
```

- **结构指针作为参数**

if a large structure is to be passed to a function,it is generally more efficient to pass a pointer than to copy the whole structure.

- **指向结构的指针**

```
struct date{
    int month;
    int day;
    int year;
} myday;
struct date *p = &myday;

*p.month=12;//要敲的比较多，可以用下面的
 p->month=12;
```

用->表示指针所指的结构变量中的成员

- **结构数组**
  结构数组

```
struct date dates[100];
struct date dates[] = {
    {1,2,2005},{2,4,2006}}
```

- **结构中的结构**

```
struct timeandDate{
    struct date sdate;
    struct time stime;
};
```

嵌套的结构

```
struct point{
    int x;
    int y;
};
struct rectangle{
    struct point pt1;
    struct point pt2;
};
```

```
如果有变量struct rectangle r 就有
r.pt1.x;r.pt1.y;r.pt2.x;r.pt2.t;

如果有变量定义
struct rectangle r,*rp;
rp = &r;
则一下四个式子是等价的
r.pt1.x
rp->pt1.x
(r.pt1).x
(rp->pt1).x
但没有rp->pt1->x,因为pt1不是指针
```

- **typedef自定义数据类型**
  typedef声明一个已有数据类型的新名字
  typedef int length
  使得length成为int的别名
  这样length这个名字就可以代替int出现在变量定义和参数声明的地方了

```
typedef long int64_t;
typedef struct aDate{
    int month;
    int day;
    int year;
} Date;
int64_t t = 1000000000000;
Date d = {9,1,2005};
```

```
typedef char* Strings[10];
Strings是十个字符串的数组的类型
```

- **联合union**

```
union AnElt{
    int i;
    char c;
} elt1,elt2;


elt1.i=4;
```

```
elt1.c='a';
elt2.t=0xEDADBEEF;
```

所有成员共同占用一份内存空间，所以叫联合
选择：成员是一个int i，还是char c
sizeof(union...)=sizeof(每个成员)的最大值

```
#include <stdio.h>

typedef union {
    int i;
    char ch[sizeof(int)];
} CHI;

int main(){
    CHI chi;
    int i;
    chi.i =1234;
    for(i=0;i<sizeof(int);i++){
        printf("%02hhx",chi.ch[i]);
    }
    printf("\n");

    return 0;
}
```

得到的结果是d2040000
可是1234的十六进制储存应该是000004d2
原因是x86处理器是小端，低位靠前
这是常见的用union的场合，得到int，double内部的字节，当我们要做文件操作的时候，把一个整数以二进制的形式写到一个文件里的时候，这就是我们可以做读写的中间的媒介i