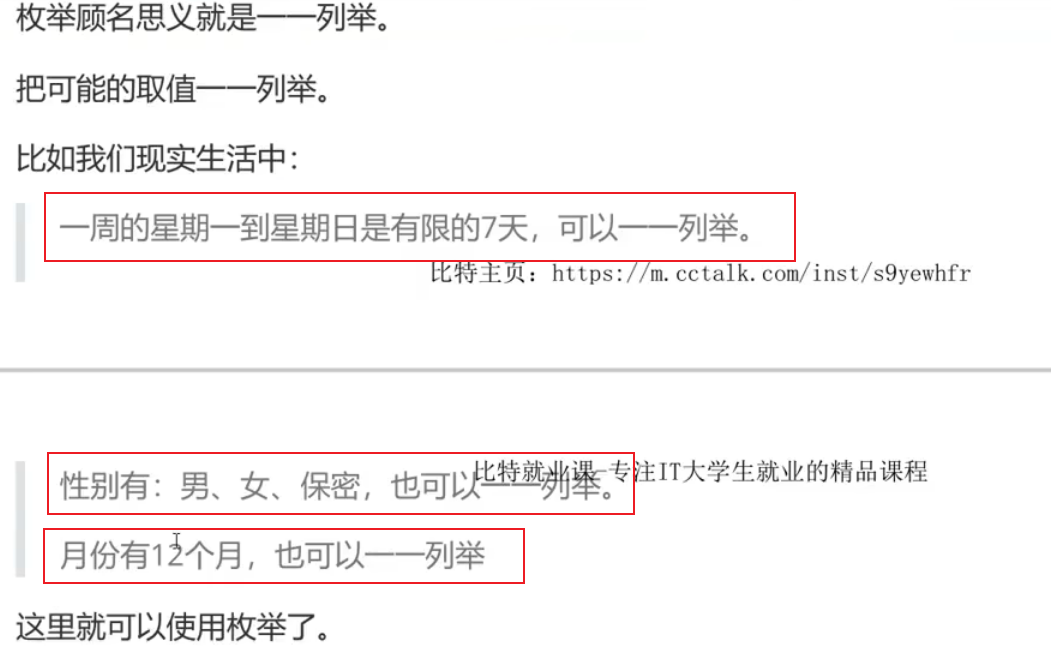
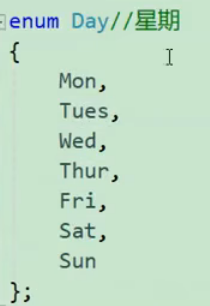
# 枚举

enumerate

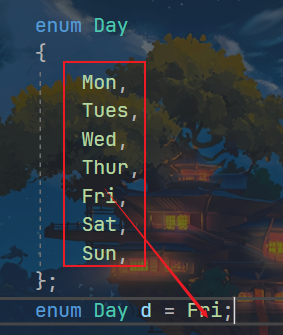
顾名思义：列举



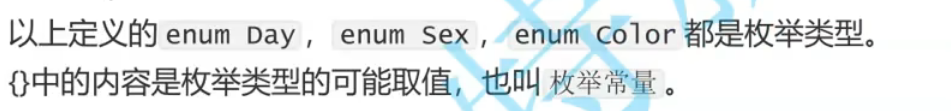
关键词：enum



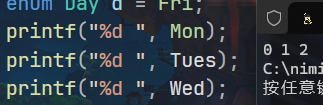
枚举里的成员是用逗号,分隔开的



我们未来再给枚举变量赋值时，只能从自定义枚举内的可能值进行赋值。



里面的值是常量。



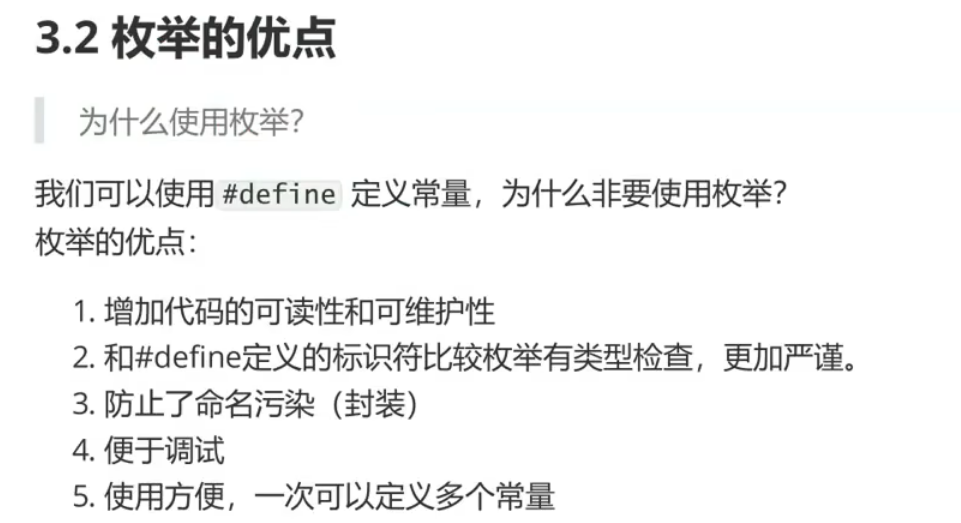
默认情况下，枚举常量的值是从0开始一次递增；如果需要修改，则



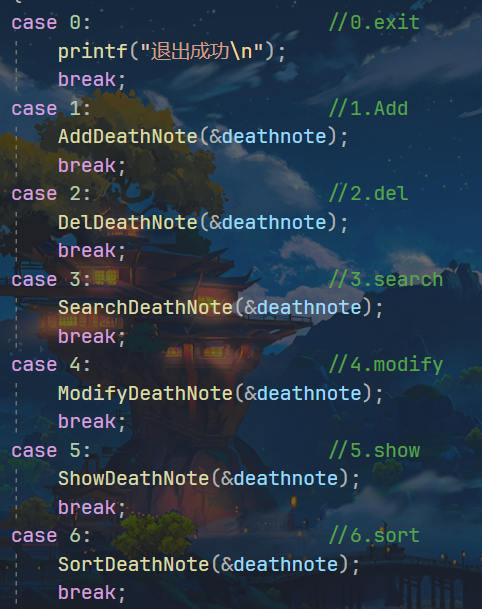
可将第一个赋值后，也是按照递增顺序。

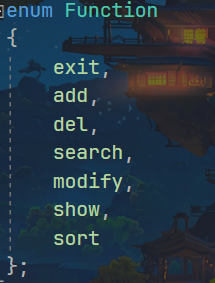
自定义类型相当于图纸，是放在内存中的代码储存区。

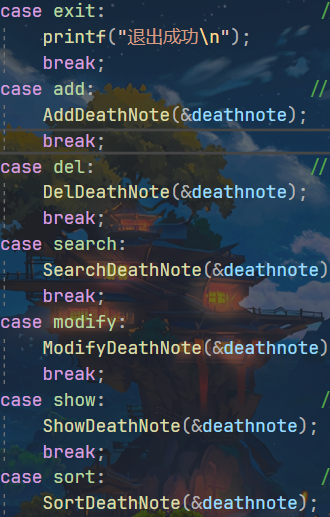
## 枚举的优点



eg

当我在实现这些函数的时候，我总会忘记case 1,2,3,4要去实现什么，然后我又要翻回去看我的菜单……但是如果有枚举常量的话



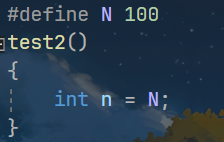
这样就变的一目了然了

同时，和define定义的常量相比，枚举具有类型，能进行严谨的类型检查。

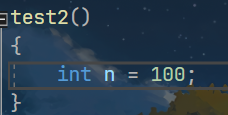
从代码走向程序的历程

test.c----🡪(预处理) ----🡪编译----🡪链接----🡪.exe

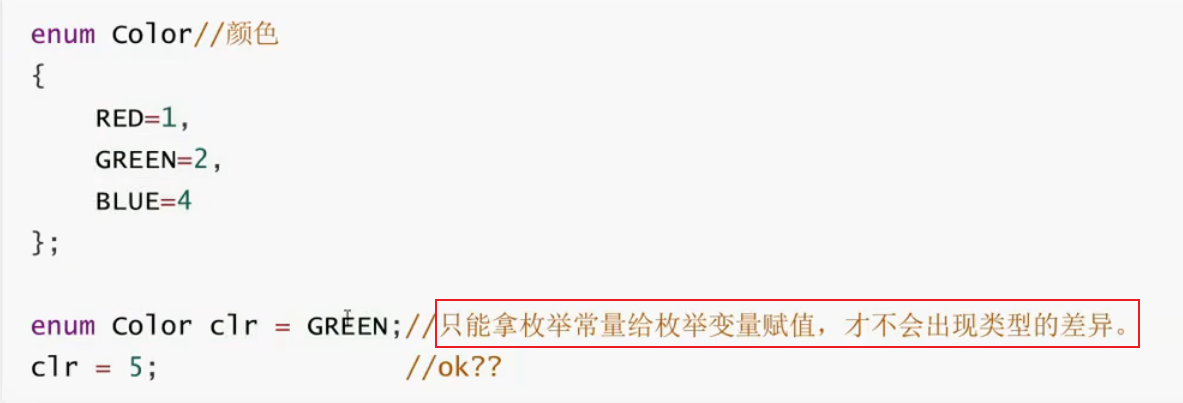
假设在预处理前，我们的代码为：



在预处理之后，我们用define定义的常量就会发生替换为：

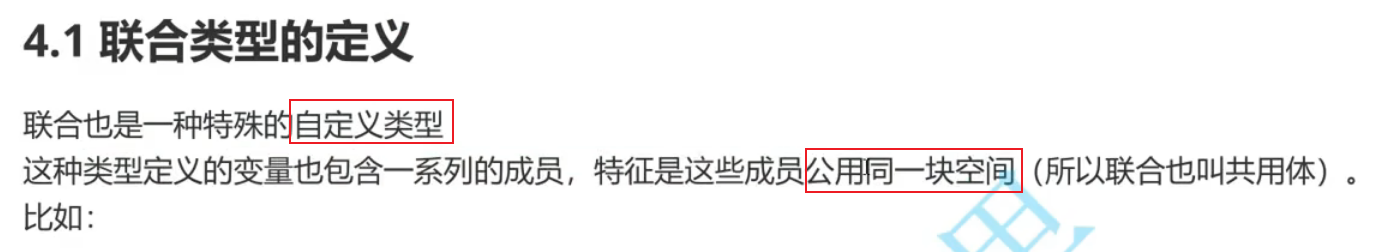
并且以二进制的形式保存在代码区。而当我们在调试的时候，我们虽然肉眼看到的是N，但是实际上N已经被替换成100并转换成二进制保存在代码区了，这样我们调试的时候不方便观测N

而枚举是实实在在有类型的

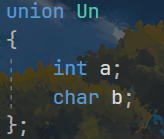


当然，只能拿枚举常量来给枚举变量。

# 联合（共用体）



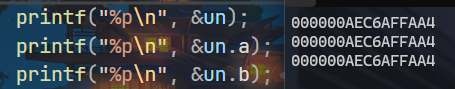
关键词 union



之所以叫共用体，是因为他们共同使用一块空间

sizeof(union Un)的大小为4

点操作符、指针操作符也和结构体相同

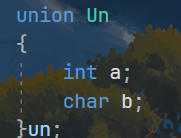


打印出来发现他们的地址竟然一样！

所以说明他们有存在共用的内存空间，相当于合租

而结构体的成员变量是每个成员都有自己独立的空间，相当于独立房间。

共用体成员变量在使用的时候并不会同时使用，



使用a的时候不会使用b；使用b的时候不会使用a

（共享单车 我用好的时候还回去，等下个人用）

所以我更改a的时候b也会被改；改b的时候a也会改





0x表示以16进制进行赋值

两个十六进制位为一个字节

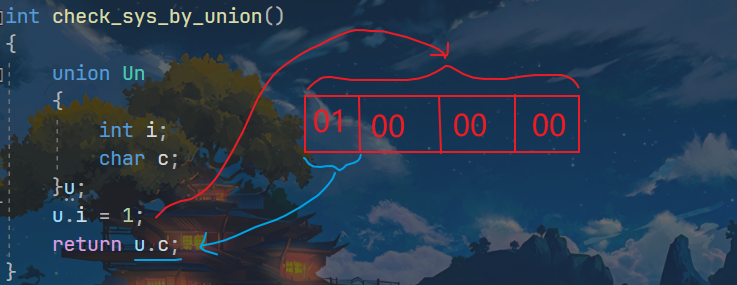
int a = 0x00000001;

00 00 00 01

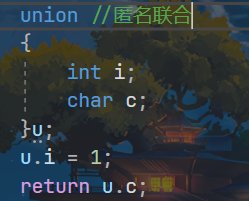
低----------高 （内存地址）

01 00 00 00 （小端）

00 00 00 01 （大端）



用联合进行判断大小端



匿名联合 以后不用了