**位运算**

1. 位的概念
   1. 定义
      1. 一个 int占4个字节内存，一个 char 占 1个字节内存
      2. 一个字节由 8 个二进制位组成
   2. 位运算符
      1. & 按位与 【都为 1 结果才为 1】
      2. | 按位或 【有 1 的话就为 1】
      3. ^ 按位异或 【相同为 0 ，不同为 1】 {翻转某些位，可以与 1 进行异或，不变则与 0 进行异或}
      4. ~ 取反 【1 变 0， 0 变成 1】
      5. << 左移 【将一个数的二进制位向做移动若干个位，右侧补零】{相当于原来数字乘以 2 的若干次方}
      6. >> 右移 【与左移相反】
   3. 位运算符具体运算
      1. 例：假如我们正在开发网络游戏，现在有一个功能模块记录十个每日任务的状态
         1. 我们常规的做法是不是用一个 int task[10] 来记录每一个人物的状态 0 代表没完成， 1 代表完成
         2. 但是我们可以发现，是不是还得用 40 个字节来存储状态
         3. 但是我们现在学习了位运算，我们位二进制位刚好是 0 和 1而且我们的一个 unsigned的整型 4 个字节，是可以有32位的，也就是我们用一个整型就可以存储 32 个每日任务，想想是不是很划算呢？？
         4. 代码
            1. #define BIT(x) (1<<x) // 是什么意思呢？？

BIT(0) 1

BIT(1) 10

BIT(2) 100

……

BIT(9) 1000000000

* + - * 1. 那么我们定义unsigned int task = 0; // 一天刚开始，所有位都为0
        2. 现在我们来定义一个枚举类型的

代码

enum Task{

ETask1 = BIT(0);

ETask2 = BIT(1);

ETask3 = BIT(2);

ETask4 = BIT(3);

ETask5 = BIT(4);

ETask6 = BIT(5);

ETask7 = BIT(6);

ETask8 = BIT(7);

ETask9 = BIT(8);

ETask10 = BIT(9);

};

* + - * 1. 拿我们怎么判断某天的内务是否做过了

那我们的ETask7 = 0000001000000

我们初始的task = 0000000000000

那我们怎么进行判断是否做过了呢

那我们就进行按位与， 如果不等于 0 代表做过了

那我们进行按位或，是不是等于做任务？？？？

总结：

大家可以输出一些东西来玩，比如运行程序，会显示你好，欢迎进入某某游戏，您的每个任务已经发送，请您及时完成并领取官方奖励，利用sleep 来模拟进行任务，利用random 来随机做哪个任务，如果全部完成，发放奖励

struct birth{

int year;

int month;

int day;

};

struct Student{

int num;

char name[10];

char ads[100];

struct birth birthday;

};

1. 结构体的引用
   1. 用 . 符号
      1. S1.num= 1001;
      2. S1.birthday.year = 1996
2. 结构体初始化

struct Student s = {007, "王五", "北京",2019, 2, 2};