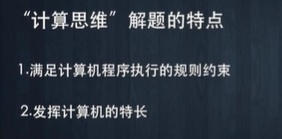
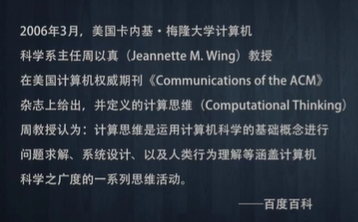
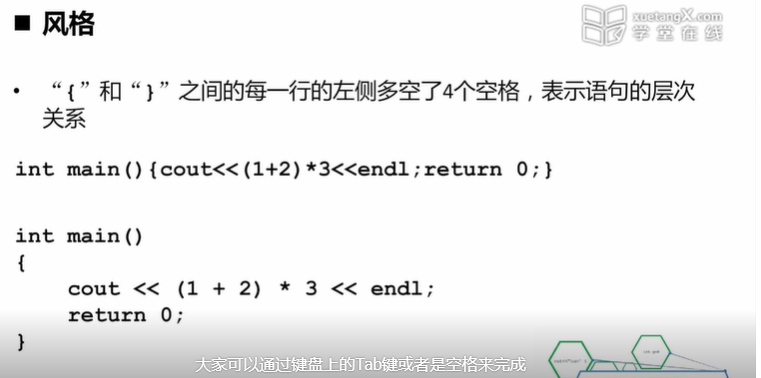
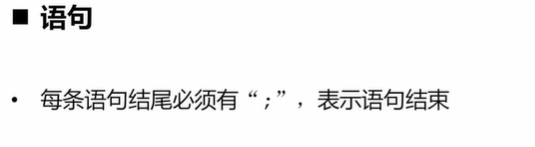
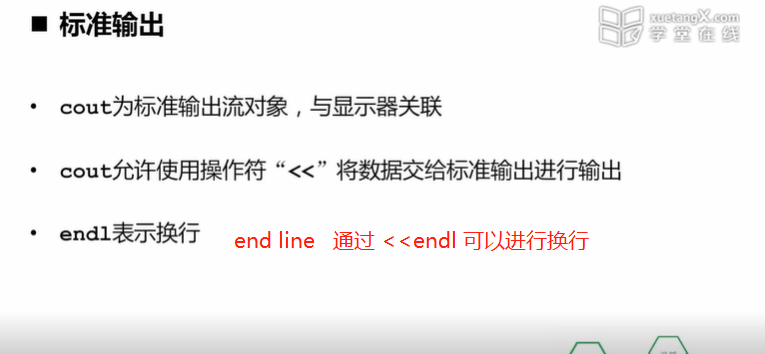
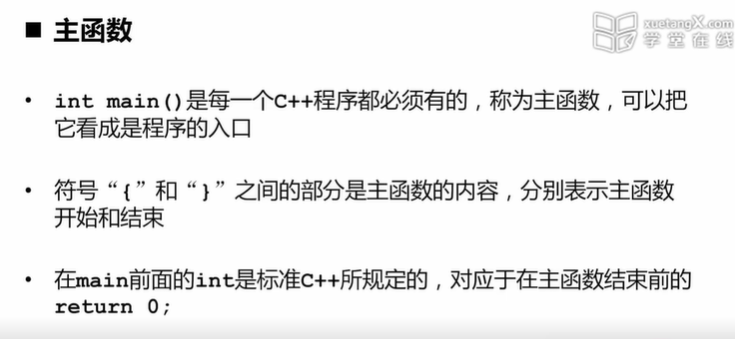
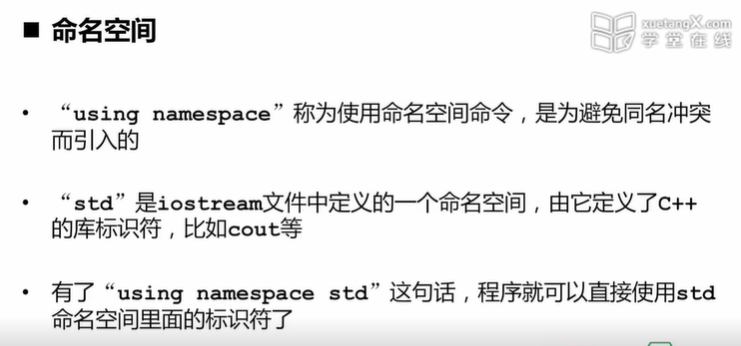
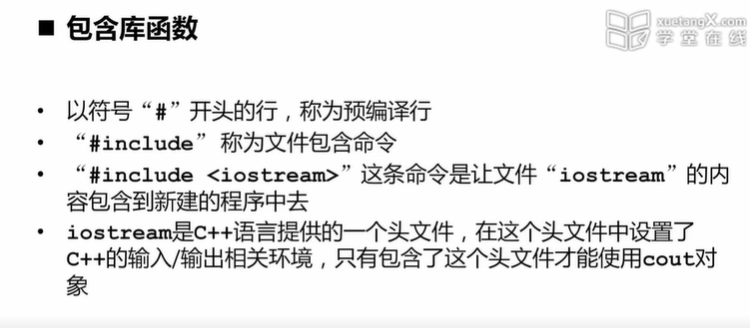
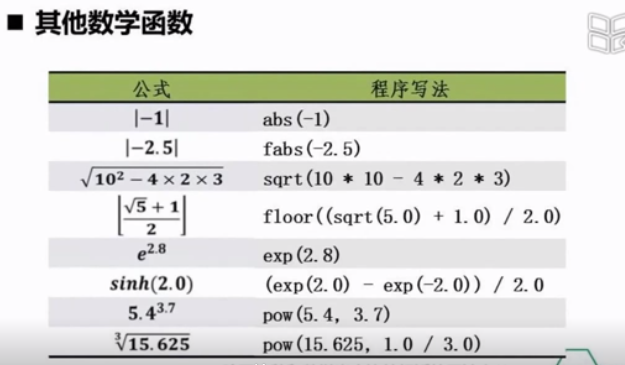
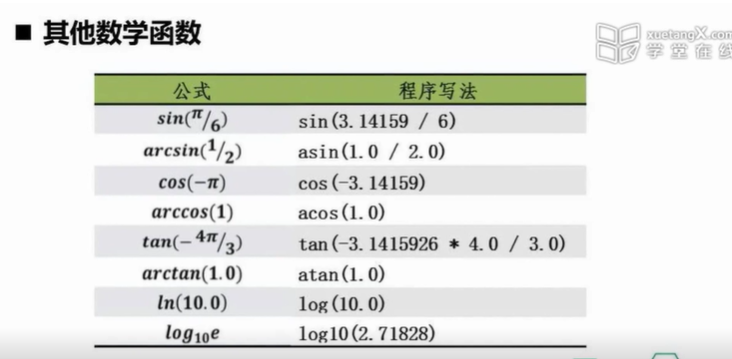
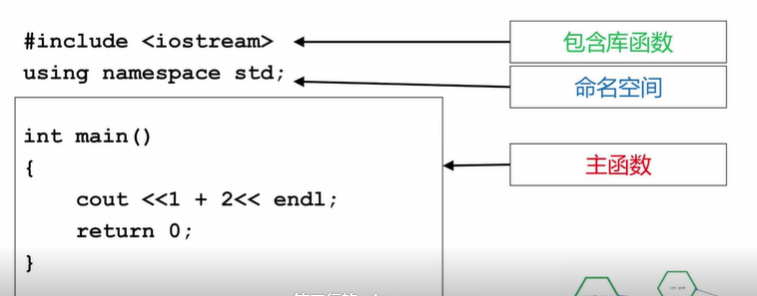
程序：为得到某种结果，通过计算机语言表达的指令序列

c++ 和 java 属同一种类型语言。

计算思维：计算思维是运用计算机科学的基础概念进行问题求解，系统设计，以及人类行为理解等涵盖计算机科学之广度的一些列思维活动。



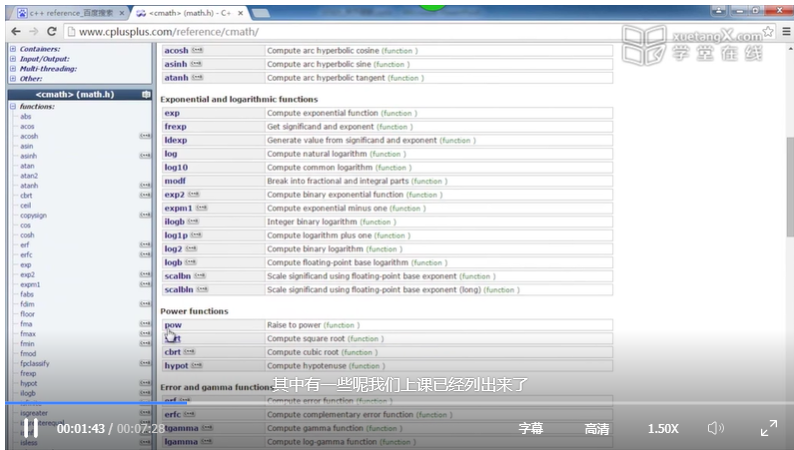


层次：

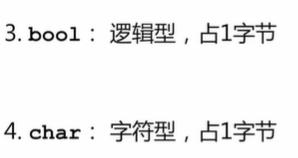
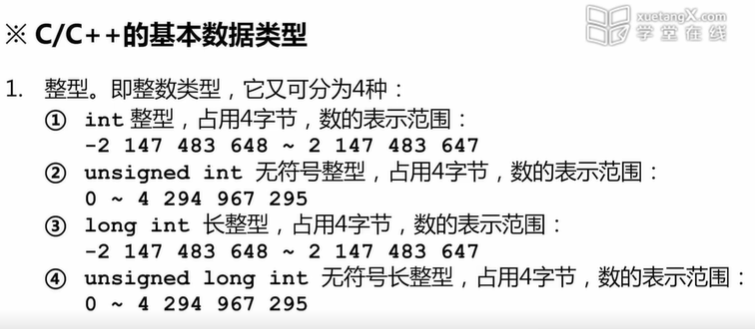
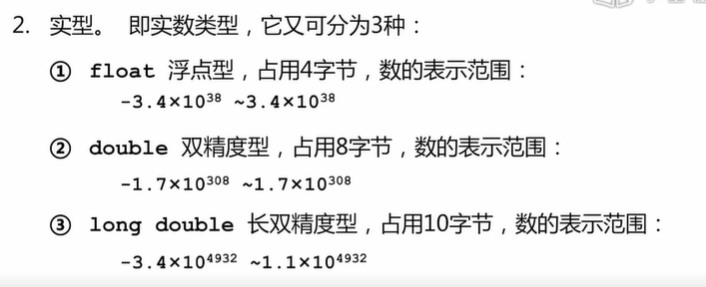
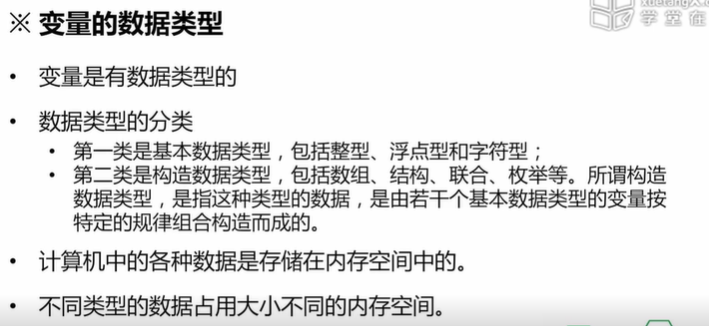
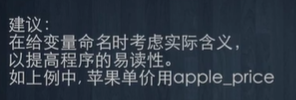
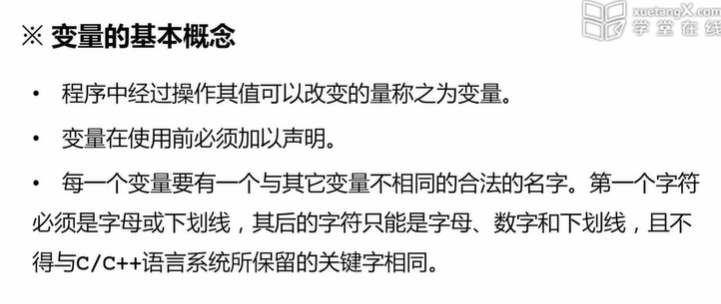


培养计算思维 激发学术志趣 引领计算人生

思考 练习 提问 阅读

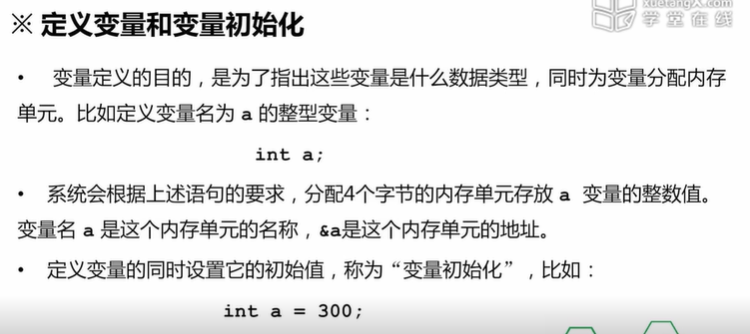


在电子秤模拟中，我们用到了用变量来表达的一个计算的式子，我们把使用变量来思考问题，用变量之间的来表达一定的关系，表达一定的规律，这样的一种思维的方法，我们说这就叫做代数思维。



Int a；

a这个变量所在的存储空间里面的二进制位，要把它解释为为整数（言下之意，我是不是也可以解释成为别的呀）

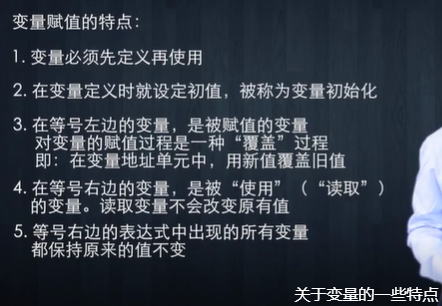
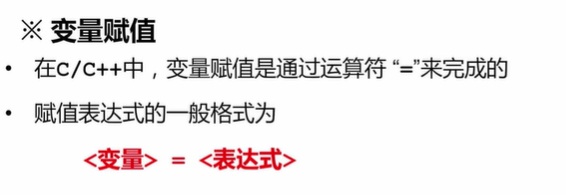


Int a =300；

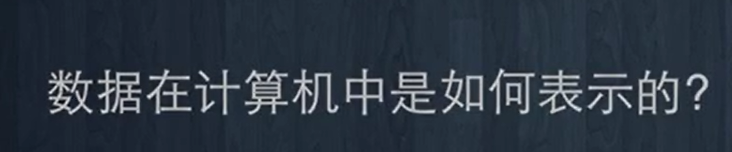
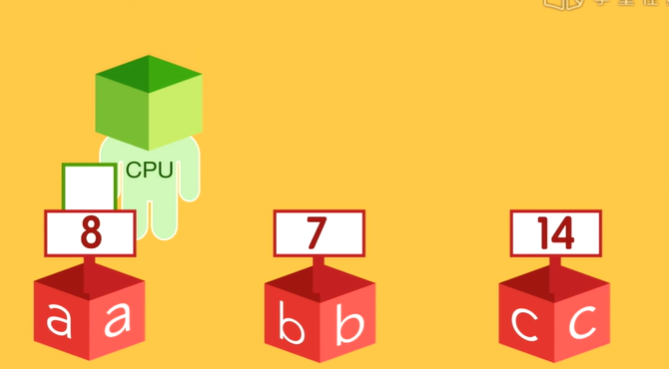
解释：让计算机去分配一个4字节的内存空间，把300这个值存储进去。

这就是一个变量在定义的同时赋了初值，这就叫做变量的初始化。

这个时候就可以使用它（a）；



Cpu 操作变量 赋值

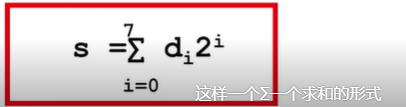


一个8个二进制位所组成的整数

00000000 11111111 每一位都可能是 1 也都可能是0

一个8个二进制位所组成的整数，它去表示的时候就是 8个 01 01 这样的序列

西格玛求和（∑）



di 的值为 0 或者 1 类似于 真或者假

∑ 顶部的数字表示 上界，底部的数字表示 下界， 右侧的数的数字 是一个变量，表示从下界到上界。

运算下来：

例如di，i取值为 1，2，3，4时 di的值为 1

0\*2^0 = 0

1\*2^1 = 2

1\*2^2 = 4

1\*2^3 = 8

1\*2^4 = 16

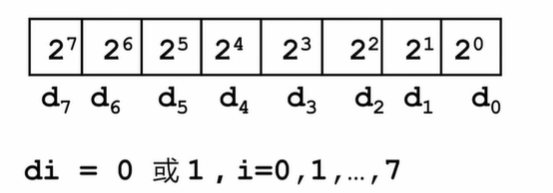
0\*2^5 = 0

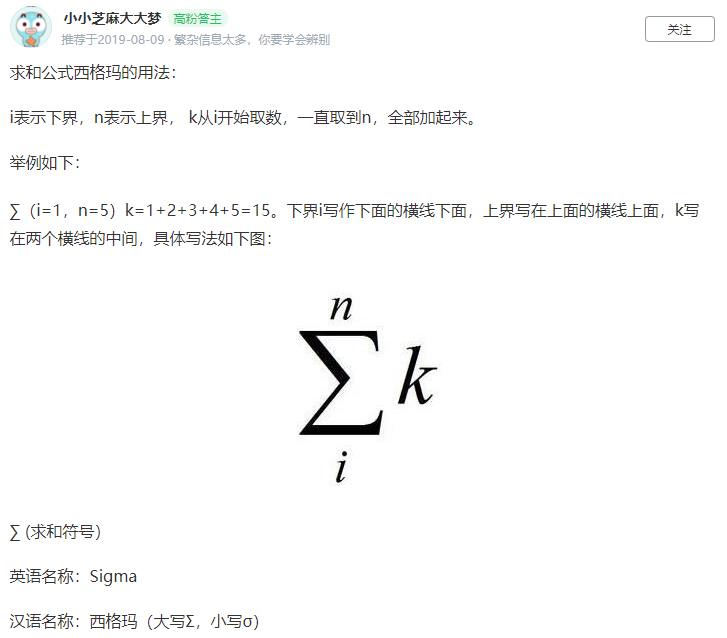
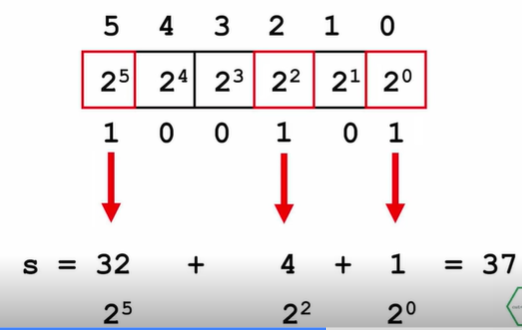
0\*2^6 = 0

0\*2^7 = 0

相加 下来就是 0+2+4+8+16+0+0+0 = 30

也就是说二进制数 转 十进制 数 可以直接通过 西格玛求和的方式 来计算



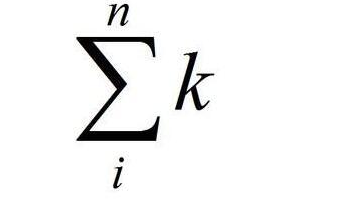


求和公式西格bai玛的用du法：

i表示下界，zhin表示上界， k从i开始取数，一直dao取到zhuann，全部加起来。shu

举例如下：

∑（i=1，n=5）k=1+2+3+4+5=15。下界i写作下面的横线下面，上界写在上面的横线上面，k写在两个横线的中间，具体写法如下图：



∑ (求和符号）

英语名称：Sigma

汉语名称：西格玛（大写Σ，小写σ）



这是 十进制数字 1 到7 的byte形式， 8个二进制位的表示形式 所组成的一个表



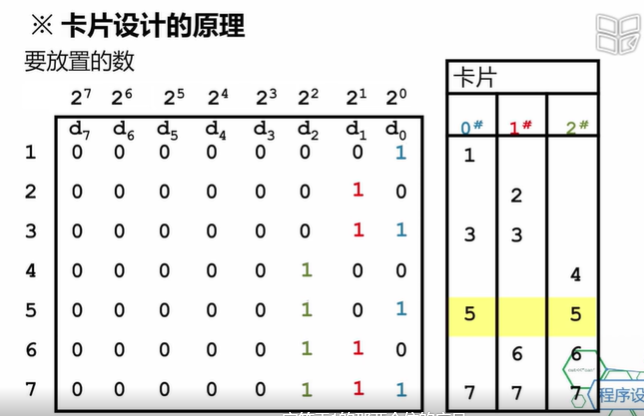
数字 1到 7 在8个二进制位的 0/1值



通过 二进制位 计算 十进制数 的 西格玛 相加所需的元素



整体（第0张，第1张，第2张共三张卡片）



一到50的每一个数字 ，都用二进制来表达，看看它们各自的1出现在什么位置，然后就在对应的卡片写上那个数字，6张卡片就做成了。

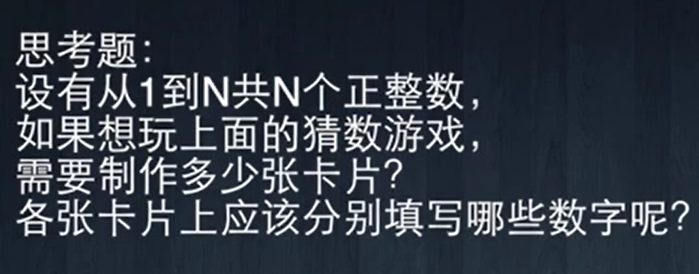
发现了一个规律 ： 当二进制 数 的 最后一位为 1时 为 10进制 1 ， 紧邻的两位数都为1时，左侧数转化为10进制 总是 右侧数字 乘以 2.

比如一个数字： 00000111 最后一个为 十进制1 倒数第二位 为 十进制2 第三位为 十进制 4 相加得到 十进制数字 7

另一个规律 8位 的二进制数 以最后一位 为 下标0，第一位为 下标 7，当每一位为1时，转化为 十进制 就是 2的 下标数 次方 ， 当每一位为0时，就是 2的 下标数 次方 乘以 0 ，也就是 十进制的0

例如 十进制 6 转化为 二进制就是 00000110

老师引申出来一个问题：（这6张卡片能够通过 程序生成出来呢？ 看来是能的）



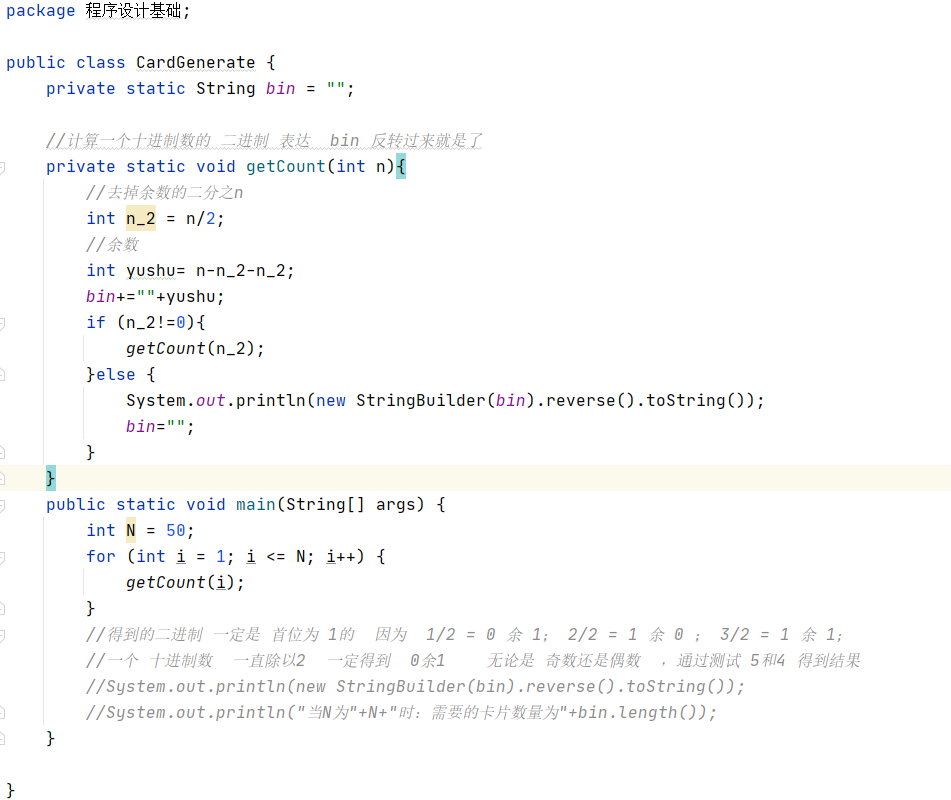
思路：

多少张卡片：卡牌的数量就是 二进制为 1的 最高位，也就是 把 N转化为 二进制数，数一下最高位所在的 位数。

难点： 10进制数 转换为 二进制数

分别填写的数字：1.通过 二进制数 的递增规律，可以推算出 每一个 数字对应的 二进制数字，再通过步骤 1 计算出 每张卡片上填写的数字。

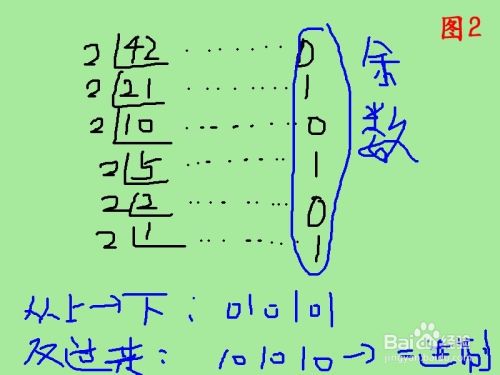
半成品：

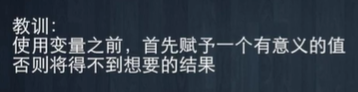
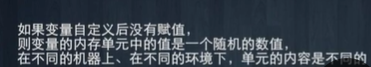
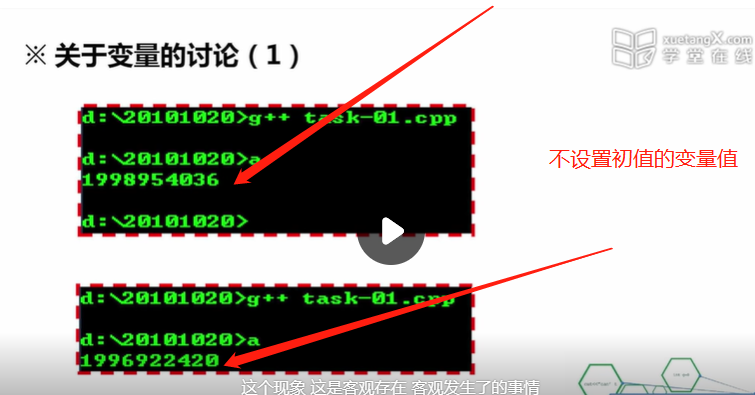


 正整数转成二进制。要点一定一定要记住哈：除二取余，然后倒序排列，高位补零。

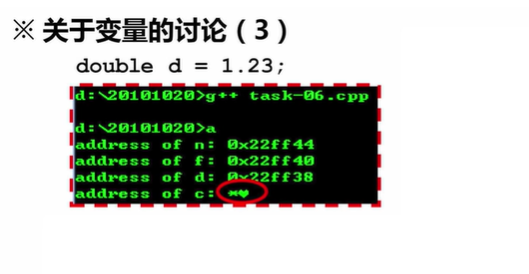
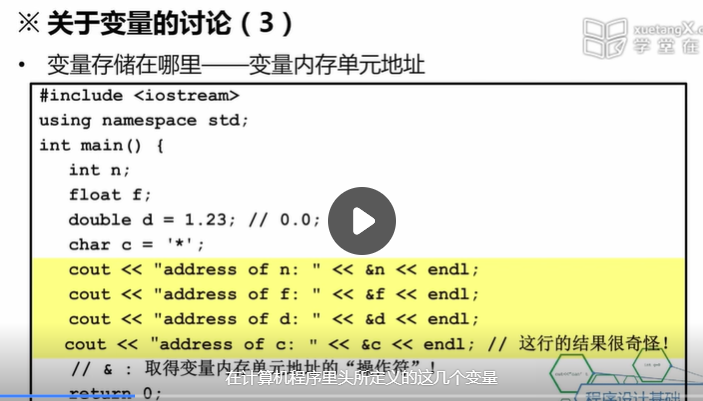
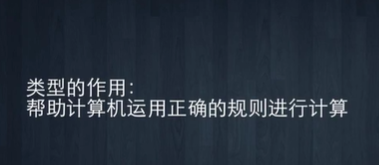
        也就是说，将正的十进制数除以二，得到的商再除以二，依次类推知道商为零或一时为止，然后在旁边标出各步的余数，最后倒着写出来，高位补零就OK咧。哎呀，还是举例说明吧，比如42转换为二进制，如图1所示操作。

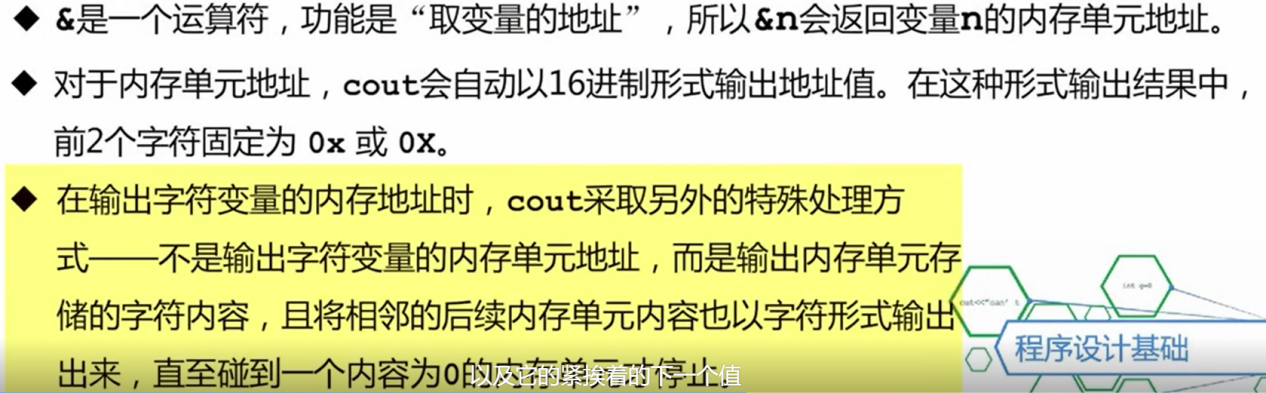
42除以2得到的余数分别为010101，然后咱们倒着排一下，42所对应二进制就是101010.如图2所示更直观的表达。

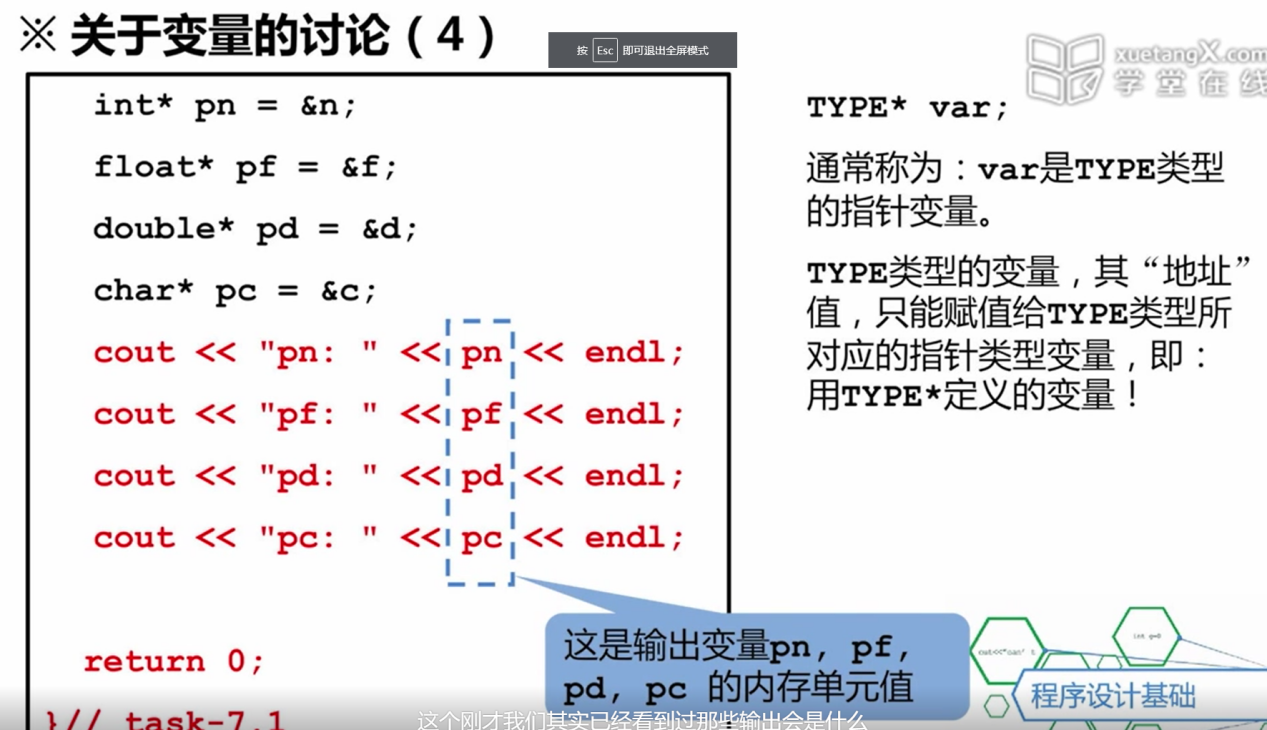
、

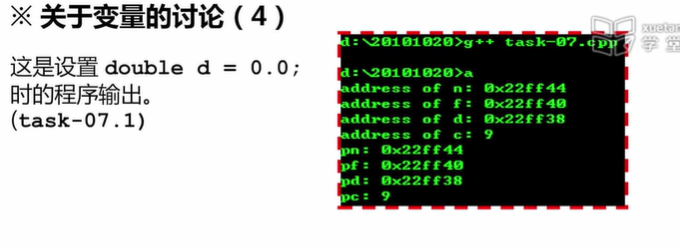


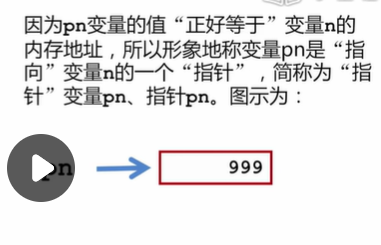
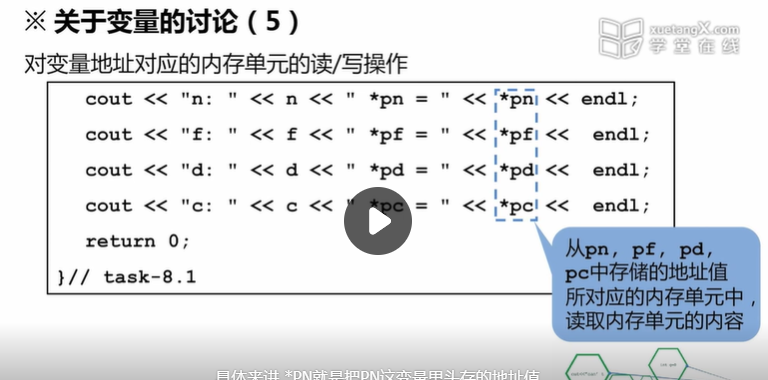
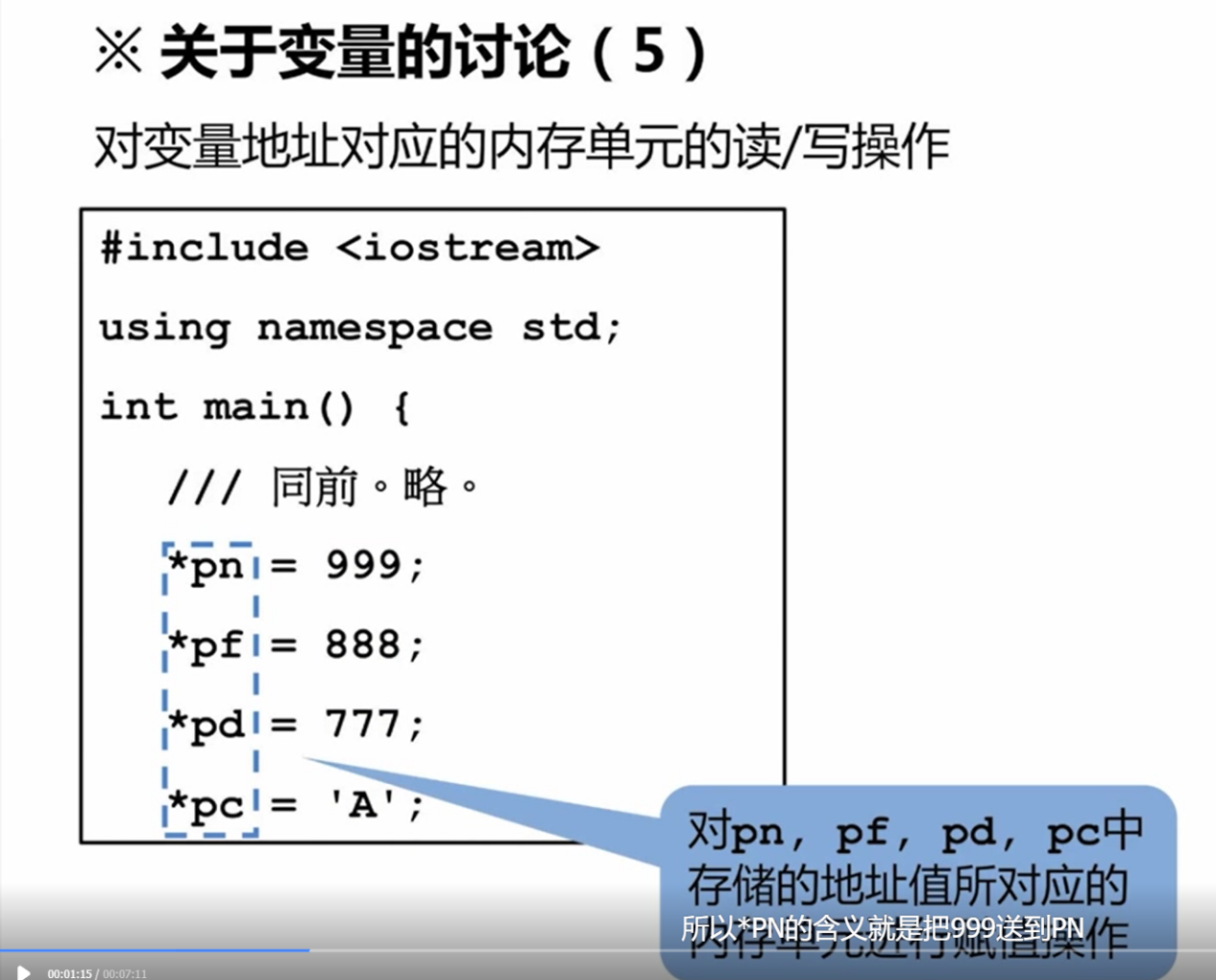
Java对此进行了优化。









放变量地址的变量，在内存里面也是占用一定空间的,是有址可寻的。#include <iostream>

using namespace std;

int main(){

int i = 2;

long l = 3;

float f = 5.0;

char c = 'cao';

// type\* var = &一个type类型的变量 ;

//type表示某种类型，type\* 表示这种类型的指针类型，

//var 时指针类型的变量名 &一个type类型的变量 返回的是一个type类型的指针类型

// &变量名 可以返回变量的 内存地址

int\* zi = &i;

long\* zl = &l;

float\* zf = &f;

char\* zc = &c;

cout<<\*zi<<endl; //2

cout<<zi<<endl; //0x6ffdfc

cout<<i; //2

cout<<&zi<<endl;

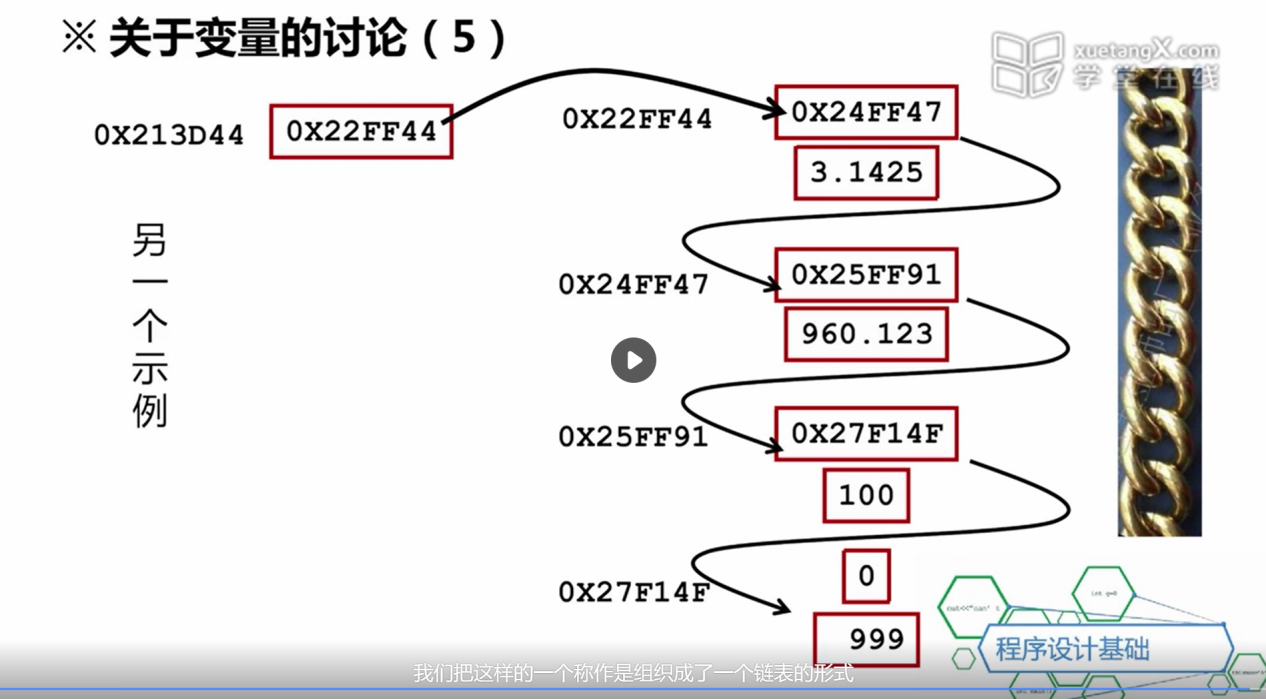
//可以通过 \*指针变量

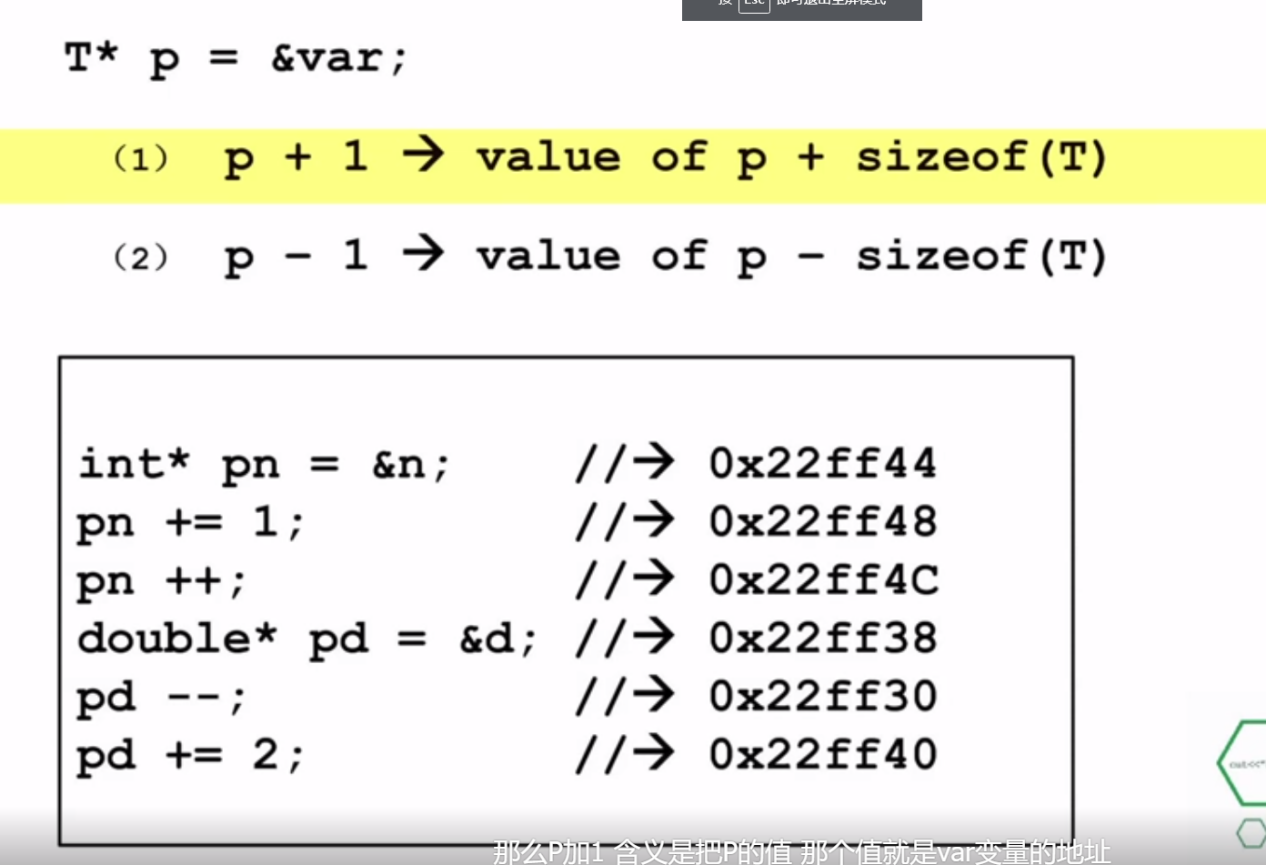
//操作 指针变量（所在的内存地址 所存储的 内存地址（int i所在的内存地址） 的 值

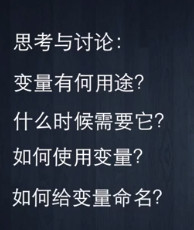
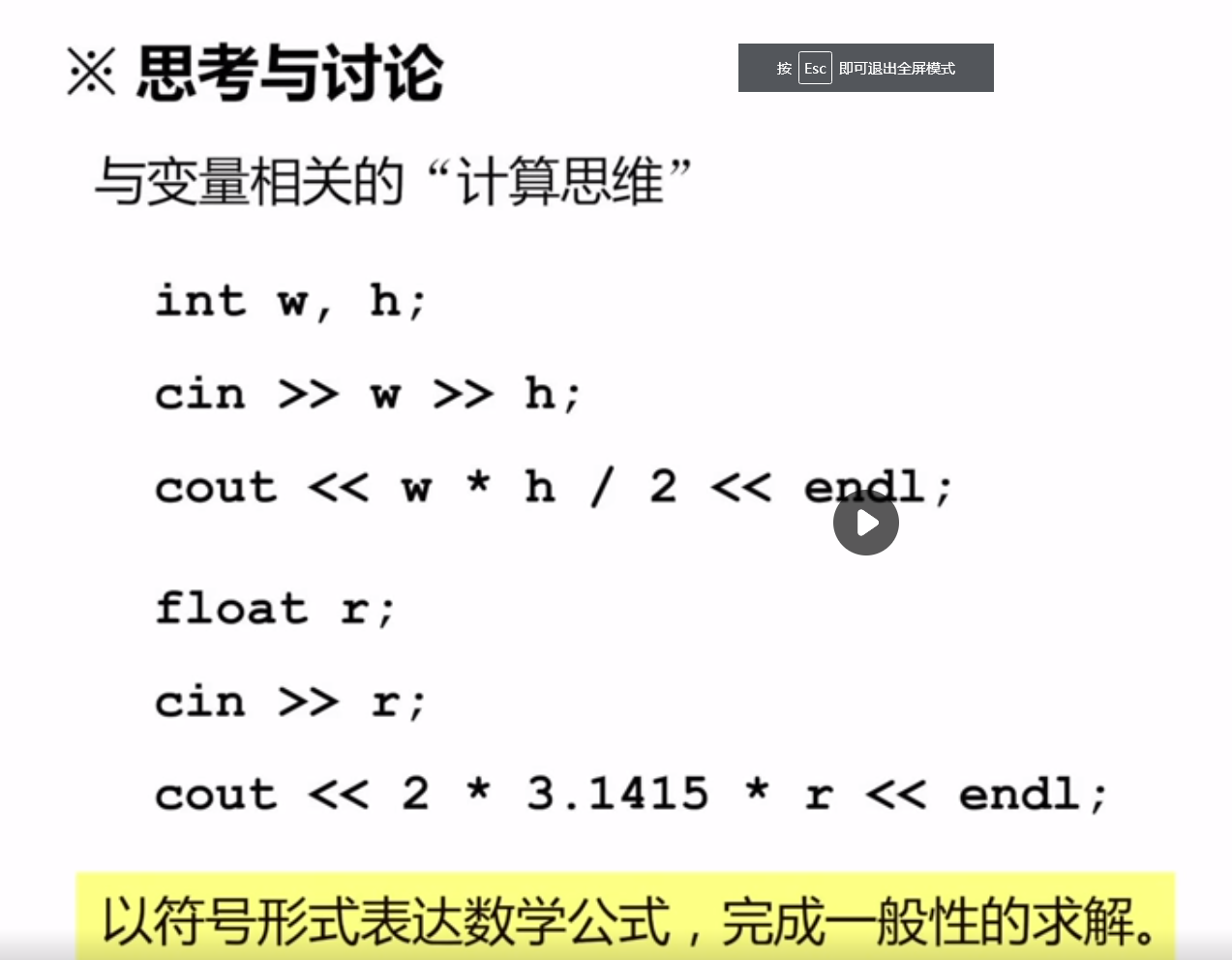
\*zi = 587545;

cout<<i;//587545

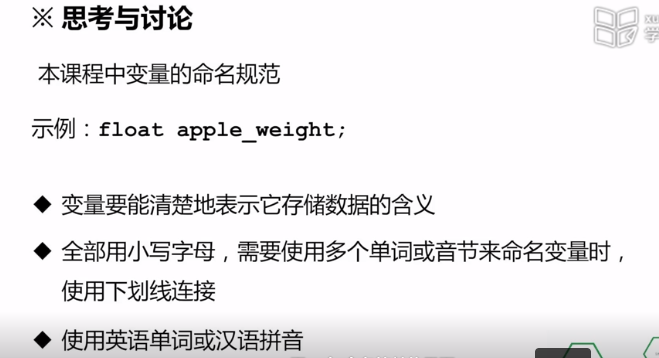
return 0;

}



p+1，加的不是表象的 1 ，而是一个 当前类型的 单位 ，也就是 类型所占的字节数 

保存 值 ，用于 后续计算。



排列组合（中学知识）

每一个元素 有两种 结果，共有 N 种元素，那么总体的 结果共有 2的N次方

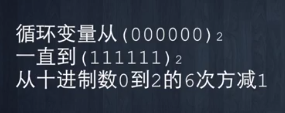
如果有 M 种结果 ，那么 总体的结果为 M的N次方

A01 2

A01B01 4

A01B01C01 8 000 001 010 011 100 101 111 1

0至2的6次方-1，总体的数量也就是 2的6次方



怎么让程序得到一个数 二进位 的每一位数？

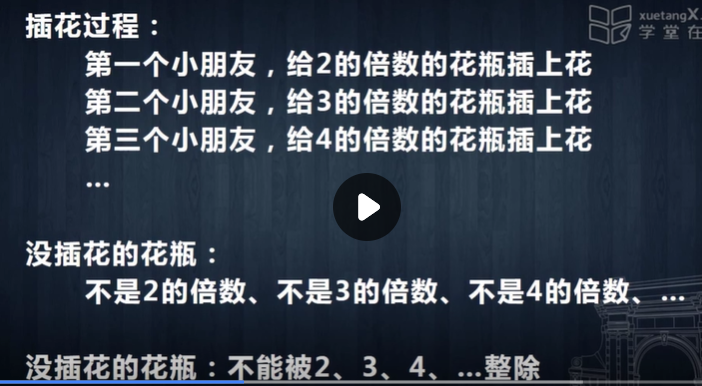
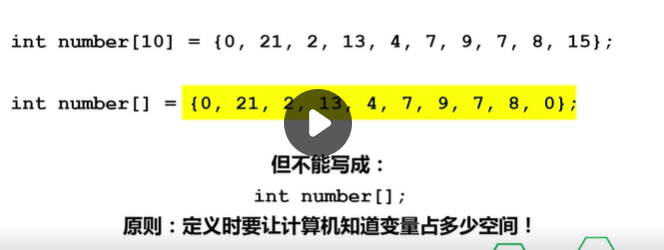
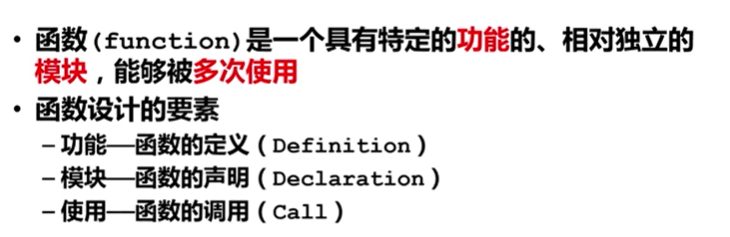
此时需要 位运算。

位运算：数的二进制表示的意义下，所做的操作。 位运算又包括以下几种运算。

按位与：

有限时间内，明确的包含几个步骤，并且每个步骤都能实现的，能够解决特定问题的，具备这样几个特点的。称之为算法

1



#include <iostream>

using namespace std;

//查找 N 中所有的 素数

const int N = 10000;

//这个放在 方法内部 是作为 局部变量 的，局部变量 被分配的内存 可能没有 N个 bool（1个字节）

//放在外面 作为全局变量， 默认初始内存 大一些

//筛法用的 bool数组

// +1 是为了不用 做 +1的偏移转换了 例如 可以用seive[100] 表示数字100，而不是99

bool seive[N+1];

//筛法 筛 插花 我可以 应用这种计算方式 在其他的地方

int main() {

for(int i=2; i<=N;i++)

seive[i] = true;

for(int d = 2;d\*d<=N;d++)

if(seive[d])

//两倍的 d 是 2乘以 2 三倍的d是 3乘以3 九倍的d是9乘以9

//第9个 2 就是 第 2个 9 第9个3 就是 第3个 9

// 在d == 9 之前，d == 8 会筛出掉 第 9个 8 ，也就是 第8个 9

// 挺厉害的 计算方式

for(int n = d\*d;n<=N;n +=d)

//false 代表被筛掉

seive[n] = false;

int sum = 0;

for(int n = 2;n<=N;n++)

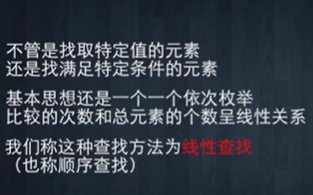
if(seive[n])

sum+=n;

cout<<sum;

return 0;

}



通过 if（条件）的方式找到想要的元素，查找的次数随着被查找的元素的多少呈现线性的递增。万级以下的元素简单查找不进行逻辑加工，速度尚可，如果是百万，千万个元素，速度会明显的下降。