零散知识

如何看懂一个程序：

①、流程；②、每个语句的功能；③、试数

小算法的程序：

①、判断一个数字是否为素数；
②、判断一个数字是否为回文数；
③、编程实现求一个十进制数字的二进制形式；
④、求一个数字的每位是奇数的数字，取出来的数字组合成新的数字；
⑤、求一个数字倒过来的数字。

对一些小算法的程序：

尝试着自己去解决他；
如果解决不了，就看答案；
关键是把答案看懂，这个是要花很大的精力，也是学习的重点；
看懂之后尝试自己修改程序，并且知道修改之后程序的不同输出结果的含义
照着答案去敲，调试错误；
不看答案自己独立把答案敲出；
如果程序无法彻底理解，就把答案背会。

强制类型转换：

格式:(数据类型)(表达式)
功能:将表达式的值强制转化为前面所执行的数据类型
例子:(int)(4.5+2.2) 最终值是 6
(float)(5) 最终值是 5.000000

```c
/*
    2020年12月14日
    强制类型转换
*/
#include <stdio.h>
int main(void)
{
    int i;
    float sum = 0;

    for (i=1; i<=100; i++)
        {
            sum = sum + 1.0/i;//是ok的，推荐使用
            //sum = sum + (float)(1/i);这样写是不对的：因为(1/i)的值一直是0(两个整型)
            //也可以这样写：sum = sum + 1/(float)(i); (float)(...)将后面括号内的值...强制转化为浮点型;   不推荐
        }

    printf("分数相加的值为：%f\n", sum);//float必须用%f输出

    return 0;
}
```

浮点数的存储所带来的问题

Float 和 double 都不能保证可以精确的存储一个小数；//存储的都是近似值
有一个浮点型变量 X，如何判断 X 是 0；

```
If(|X - 0.000001| < 0.000001)
    Printf(“是 0”);
Else
    Printf(“不是 0”);
```

为什么循环中更新的变量不能定义成浮点型

（i=1；是更新的变量。++i 是更新部分，更新部分的值一定是个整数）

因为浮点型是一个非准确存储。

进制转化

1、什么是进制

逢 n 进一

在 C 语言中：

八进制前加 0(零)，十六进制前加 0x 或 0X（也是零），十进制前什么也不加。

在汇编中：

在数字后面加字母 B 表示二进制；

在数字后面加字母 O 表示八进制；

在数字后面加字母 D 表示十进制；

在数字后面加字母 H 表示十六进制。

小数除大数，则商使零，余数是小数本身。

eg:

1/2=0； 余数为 1

2/2=1； 余数为 0

3/2=1； 余数为 1

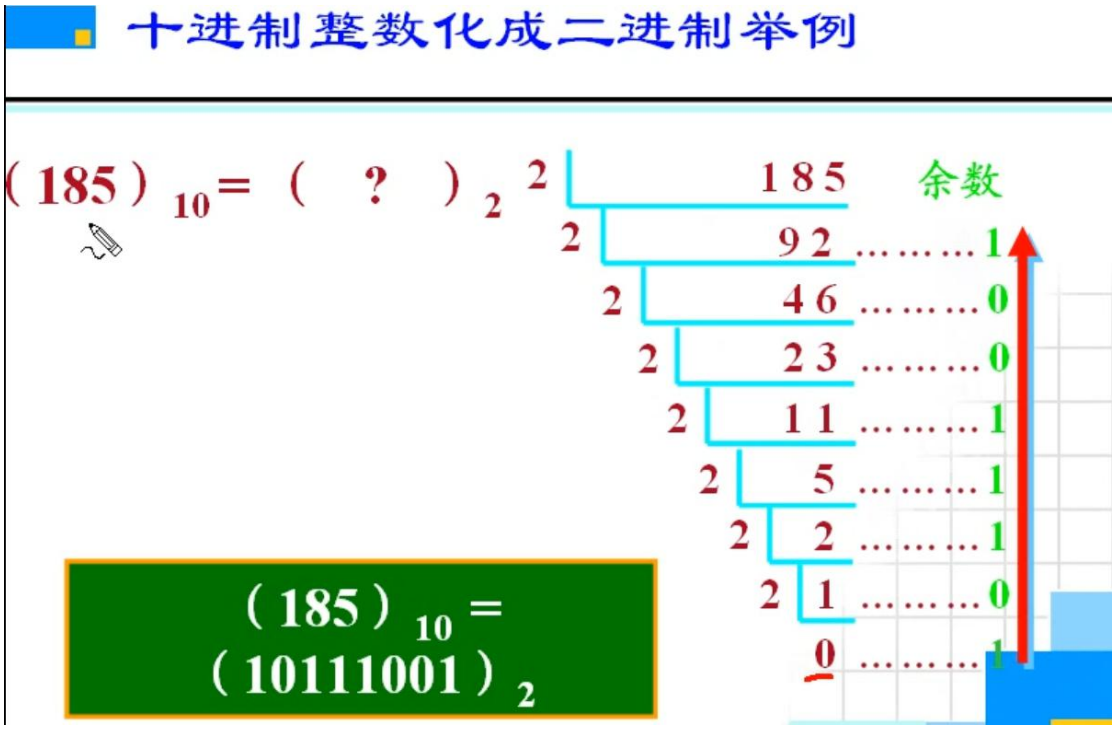
2、把 r 进制转换成十进制

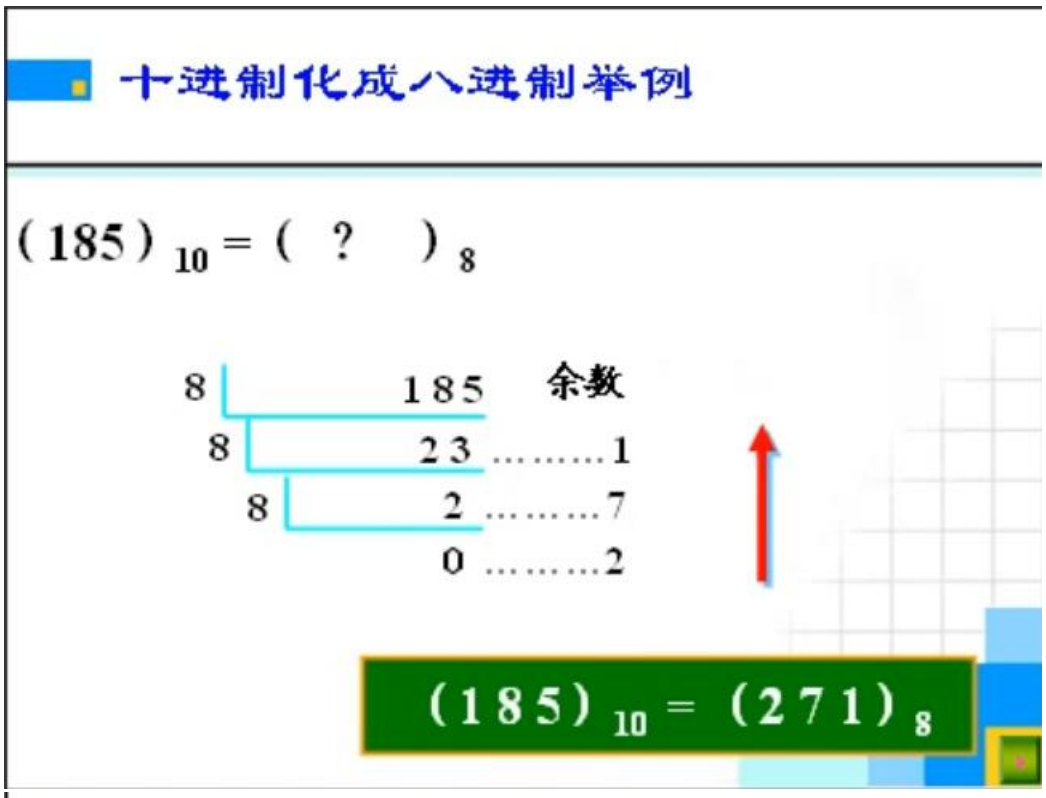
十进制的 1234：4*10 的 0 次幂+3*10 的 1 次幂+2*10 的 2 次幂+1*10 的 3 次幂

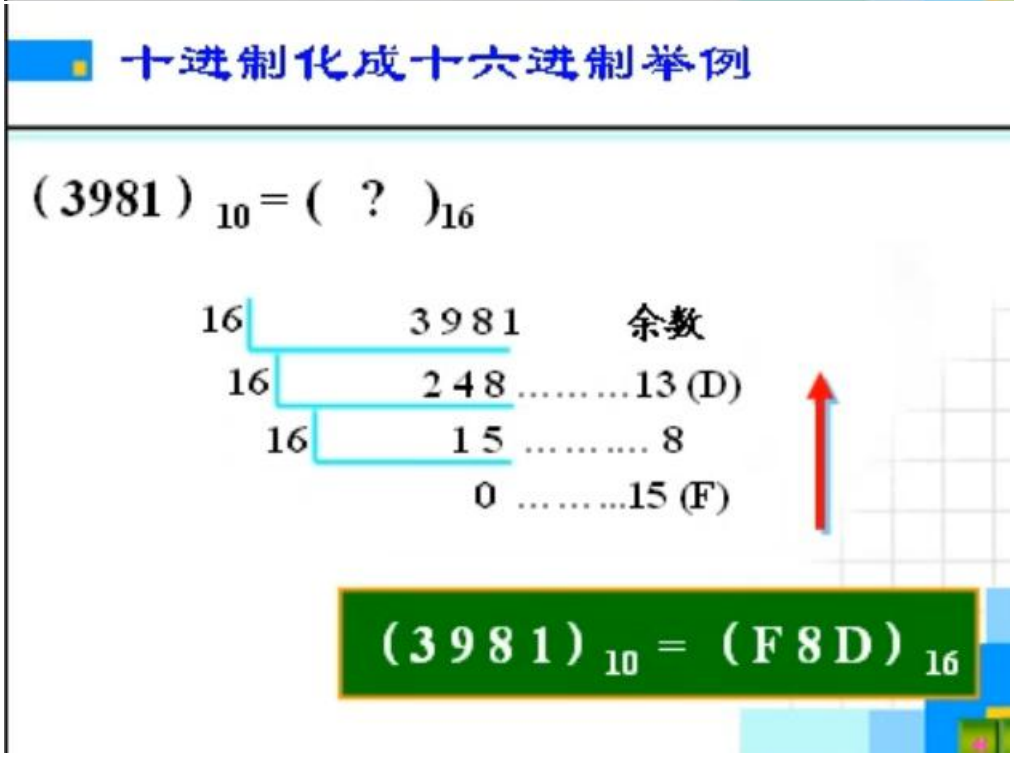
二进制的 1234：3*2 的 0 次幂+3*2 的 1 次幂+2*2 的 2 次幂的+1*2 的 3 次幂

3、把十进制转化成 r 进制

方法：除 r 取余，余数倒序排列







4、不同进制所代表的数值之间的关系

十进制的 3981 转化成 十六进制的 F8D

十进制的的 3981 和十六进制的 F8D 所代表的本质都是同一个数字，只不过按照不同的进位换算所得到的不同的结论。

常用计数制对照表

十进制(D)	二进制(B)	八进制(O)	十六进制(H)
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	a
11	1011	13	b
12	1100	14	c
13	1101	15	d
14	1110	16	e
15	1111	17	f

一些琐碎的运算符知识

自增 自减 三目运算符 逗号表达式【三个越来越不重要，越来越琐碎】

1、自增[或自减]

分类：

前自增： ——++i

后自增： ——i++

前自增后自增的异同：

相同点：最终都使 i 的值加 1；

不同点：前自增整体表达式的值是 i 加 1 之后的值

后自增整体表达式的值是 i 加 1 之前的值

```c
/*
    2020年12月15日
    前自增与后自增的比较
*/
#include <stdio.h>
int main(void)
{
    int i;
    int j;
    int k;
    int m;

    i = j = 3;//等价于：i = 3; j = 3;
    k = i++;//后自增整体的表达式的值是i加1之前的值
    m = ++j;//前自增整体的表达式的值是i加1之后的值
    printf("i = %d, j = %d, k = %d, m = %d\n", i, j, k, m);
    return 0;
}
/*
在dev c++中的输出结果是：
------------------------
i = 4, j = 4, k = 3, m = 4
------------------------
总结：    后自增整体的表达式的值是i加1之前的值
          前自增整体的表达式的值是i加1之后的值
*/
```

为什么会出现自增：

代码更精炼；

自增的速度更快。

学习自增要明白的几个问题：

1、我们编程是应尽量屏蔽掉前自增和后自增的差别；

2、自增表达式最好不要作为一个更大的表达式的一部分来使用。

或：//i++和++i 单独成一个语句，不要把它作为一个完整复合语句的一部分来使用

如：int m = i++ + ++i + i + i++;//这样写不但是不规范的代码，而且是不可移植的代码。

Printf(“%d, %d, %d\n”, i++, ++i, i);//同上。

2、三目运算符

A ? B : C;(其中? : 合称三目运算符)

等价于

If (A)

B;

Esle

C;

3、逗号表达式

格式：(A,B,C,D)

功能：从左到右执行；

最终表达式的值是最后一项的值。

在内存中，一个字节一个编号。(一个字节 8 位，8 个 01 是一个字节，一个字节有个地址)

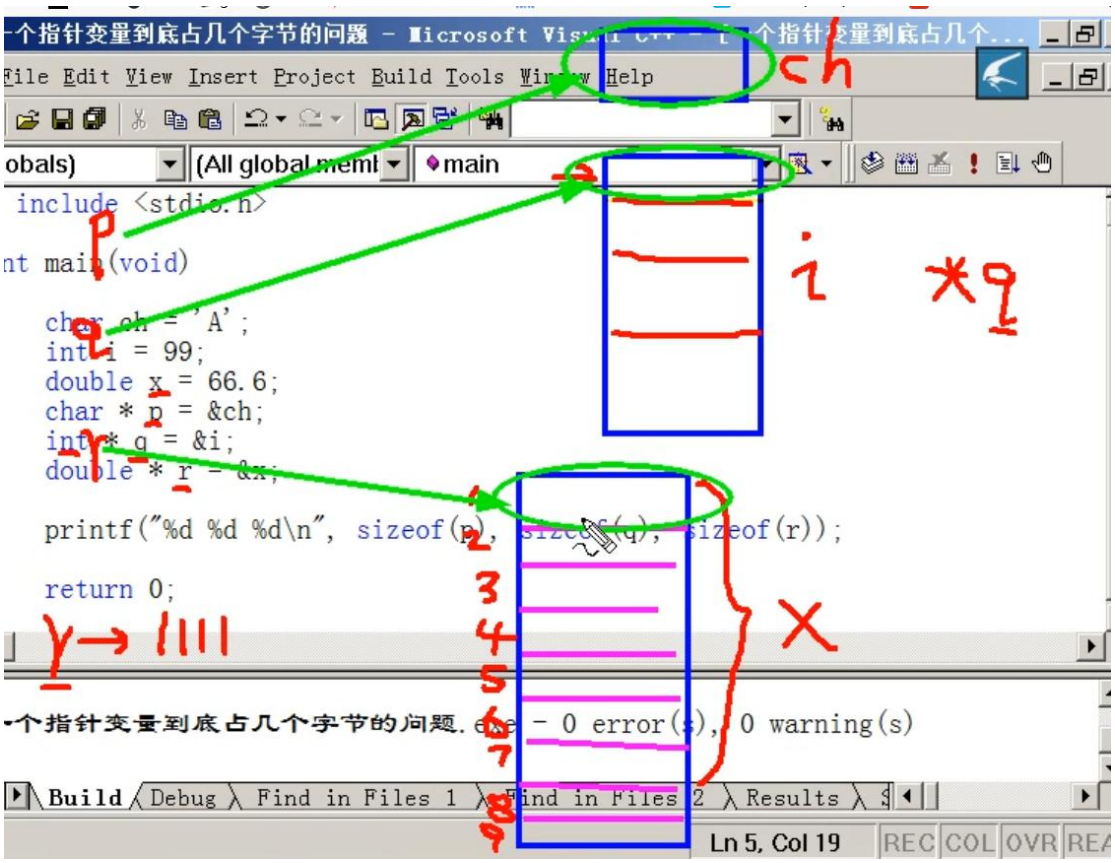
例如：double x = 66.6;

int * p = &x;

a 占 8 个字节，用首字节的编号(地址)当作 a 的编号(地址)。

用 x 占的八个字节中的第一个字节的地址当为 x 地址。

然后 p 指向 x，读取 x 的内容是根据定义 p 是的 int *;



位运算符(其中用到补码的知识)

& -- 按位与(有 0 则 0)

&&逻辑与，也叫做并且

&&和&的含义完全不同

1&1 = 1

1&0 = 0

0&1 = 0

0&0 = 0

5&7 = 5　　21&7 = 5

5&1 = 1　　5&10 = 0

[位运算符.cpp]

```
#include<stdio.h>
int main (void){
    int i = 5;
    int j = 7;
    int k;
    k = i & j;
    printf("%d\n", k);
    k = i && j;//
    printf("%d\n", k);
    return 0;
}
/*
&&是逻辑运算符,结果之能使真或假,真用1表示,假用0表示.
&使位运算符,运算过程是将5和7的二进制代码每一位进行与.
    在dev c++中的输出结果是:
    ------------------------
    5
    1
    ------------------------
*/
```

| -- 按位或(有 1 则 1)

||逻辑或

1|0 = 1

0|1 = 1

1|1 = 1

0|0 = 0

[按位或.cpp]

```
#include<stdio.h>
int main(void)
{
    int i = 3;
    int j = 5;
    int k;
    k = i|j;
    printf("%d\n", k);
    return 0;
 }
 /*
    在dev c++中的输出结果是:
    ------------------------
    7
    ------------------------
 */
```

~ -- 按位取反

~i 就是把 i 变量所有的二进制位取反

[按位取反.cpp]

```
#include<stdio.h>
int main(void){
    int i = 3;
    int k;
    k = ~i;
    printf("%d\n", k);//3是0011,按位取反是1100,变成一个负数.
    return 0;
}
```

^ -- 按位异或(相同为 0,不同为 1)

1^0 = 1

0^1 = 1

1^1 = 1

0^0 = 0

<< -- 按位左移

i<<1 表示把 i 的所有二进制位左移 1 位，右边补 0

左移 n 位相当于乘以 2 的 n 次方

面试题:

A、i = i*8;

B、i = i<<3;

请问上面两个语句,哪个语句执行的速度快

答案:B 快

>> -- 按位右移

i>>1 表示把 i 的所有二进制位右移 1 位，左边一般是补 0

第一种情况:不管空出的是什么,都补 0;

第二种情况:最高位是 1 就补 1,最高位是 0 就补 0.

右移 n 位相当于除以 2 的 n 次方,前提是数据不能丢失

面试题:

A、i = i/8;

B、i = i>>3;

请问上面两个语句,哪个语句执行的速度快

答案:B 快

位运算符的显示意义:

通过位运算符我们可以对数据的操作精确到某一位。