NOI(全国青少年信息学奥林匹克竞赛)

C++的运行

在 NOI 比赛中使用 Dev C++（window 系统）软件进行编译。
Dev c++ 安装地址：https://bloodshed-dev-c.en.softonic.com/
打开 dev c++就可以开始编写我们的程序了。

第一个程序 Hello，world!

```
#include<iostream> //调用 cout 函数

using namespace std;

int main(){

cout<<" hello,world";

    return 0;// 程序结束语

}
```

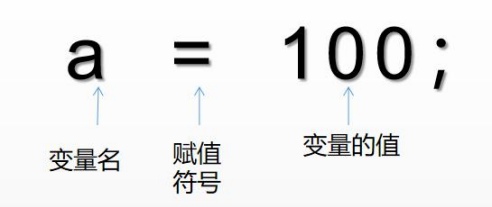
cout 语句使用总结
cout 语句的一般格式：
cout<<语句 1<<语句 2<<....<<语句 N<<endl; // 其中 endl 表示换行
功能:
如果程序是表达式（算式），则输出表达式的值；
如果语句加引号，则输出引号内的内容
当带双引号的语句内出现了' /n' 将换行。

变量的使用

```
#include <iostream>

using namespace std;
```

```
int main(){
    int a;    //变量的声明
    a = 100; //变量的赋值
    cout<<a
    return 0;
}
```



int a; 此行代码就表示创建了一个整数类型的数字。

变量名命名规则：只能是字母(a-z A-Z)，数字(0-9)，下划线(_)的组合，并且之间不能包含空格，数字不能放在变量名首位。

常用的运算符号

+、-、*、/、%（取余）、（）、
注意：c++没有大括号中括号都用小括号表示。
%: 求余运算符可以用来拆分数字
    123456%100 = 56
    123456/100 = 1234

二进制

我们平时计数都是以 10 为进制,我们平时数数:1、2、3、4、5、6、7、8、9,数到 10 的时候就要往前进一位，个位就要变成 0，十位变成 1。
二进制就是满 2 进 1,例如 0、1、10、11、100。

二进制与十进制的转换

我们先来了解一下十进制数的拆分
$(341)_{10} = 3\times10^2 + 4\times10^1 + 1\times10^0$
二进制数转换十进制

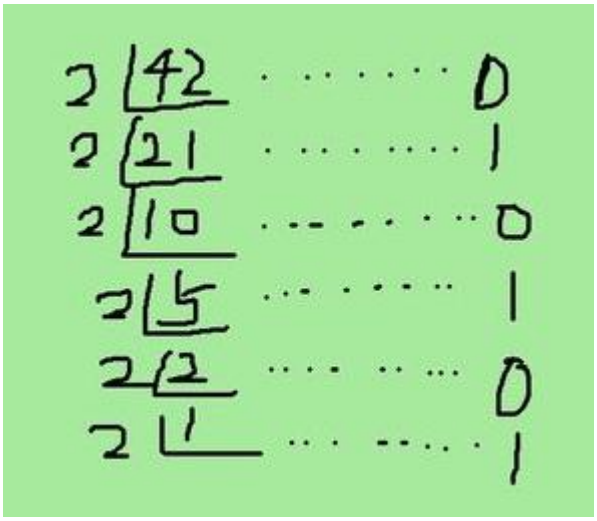
$$(1001)_2 = 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 = 8+0+0+1 = 9$$

想一想：二进制数 101.11 是多少?

十进制转二进制

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。

如图：42 的二进制数为 101010



计算机中的基本单位

位（B）：计算机最小的数据单位，每一位只能是 0 或 1

字节（Byte）：8 个二进制构成一个字节，是存储空间的基本构成单位，一个英文字母占一个字节，一个汉字占俩个字节

KB：1KB 代表 1024 个字节
注:题目中常出现字节与 KB 的转化，牢记：1KB=1024 字节

MB：也就是我们日常生活中所说的兆，通常在计算过程中粗略的记为

1MB=10^6（10 的 6 次方） 字节；实际上是 1024^2

1 Byte = 8 bit;
1 KB = 1024 Byte = 2^10 Byte;
1 MB = 1024 KB = 2^20 Byte ;
1 GB = 1024 MB = 2^30 Byte;
1 TB = 1024 GB = 2^40 Byte;

例如 8 内存一共有多少字节
8G = 8*1024 MB = 8*1024*1024 KB = 8*1024*1024*1024 bit

但如果你有 8G 的 U 盘,你要注意了,硬盘制造商的进制是 1000,电脑的是 1024
8G = 8*1000*1000*1000÷1024÷1024÷1024 ≈ 7.45G

```
#include <iostream>
using namespace std;
int main(){
    int a;    //变量的声明
    a = 100; //变量的赋值
    cout<<a
    return 0;
}
```

原码反码补码

原码

原码：是最简单的机器数表示法。用最高位表示符号位，‘1’表示负号，‘0’表示正号。其他位存放该数的二进制的绝对值。

数字在自然界中抽象出来的时候，一棵树，两只猪，是没有正数和负数的概念的
计算机保存最原始的数字，也是没有正和负的数字，叫没符号数字。
如果我们在内存分配 4 位（bit）去存放无符号数字，是下面这样子的。

十进制	二进制(4 位)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101

为了表示正与负，人们发明了"原码"，把生活应该有的正负概念，原原本本的表示出来。

	正数
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

	负数
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

但使用“原码”储存的方式，方便了看的人类，却苦了计算机。
我们希望 （+1）和（-1）相加是 0，但计算机只能算出 0001+1001=1010 (-2)
另外一个问题，这里有一个（+0）和（-0）
为了解决“正负相加等于 0”的问题，在“原码”的基础上，人们发明了“反码”。

反码

反码：正数的反码还是等于原码，负数的反码就是他的原码除符号位外，按位取反。

反码”表示方式是用来处理负数的，符号位置不变，其余位置相反

反码　　原码

	正数
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

	负数
-0	1111
-1	1110
-2	1101
-3	1100
-4	1011
-5	1010
-6	1001
-7	1000

	负数
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

当“原码”变成“反码”时，完美的解决了“正负相加等于 0”的问题过去的（+1）和（-1）相加，变成了 0001+1101=1111，刚好反码表示方式中，1111 象征 -0 人们总是进益求精，历史遗留下来的问题——　有两个零存在，+0 和 -0 我们希望只有一个 0，所以发明了"补码"。

补码

补码：正数的补码等于他的原码
负数的补码等于反码+1。
（这只是一种算补码的方式，多数书对于补码就是这句话）

补码　　反码　　原码

	正数
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

	负数
~~0~~	~~0000~~
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

	负数
-0	1111
-1	1110
-2	1101
-3	1100
-4	1011
-5	1010
-6	1001
-7	1000

	负数
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

有得必有失，在补一位 1 的时候，要丢掉最高位我们要处理"反码"中的"-0",当 1111 再补上一个 1 之后，变成了 10000，丢掉最高位就是 0000，刚好和左边正数的 0，完美融合掉了这样就解决了+0 和-0 同时存在的问题另外"正负数相加等于 0"的问题，同样得到满足举例，3 和（-3）相加，0011 + 1101 =10000，丢掉最高位，就是 0000（0）同样有失必有得，我们失去了(-0)，收获了（-8）以上就是"补码"的存在方式。

习题：

1、请问 -84 的补码是多少?

2、二进制数 11.11 转换成十进制数为多少?

3、完成程序输入一个整数 N（范围是 int）输出这个数的后 5 位。

4、int 占 4 个字节，并且是用补码的形式在计算机存储请计算出，它的取值范围。