第三章 初等数论一

1. 整除的概念

C++中的求余数运算符 %
符号 | ，如果b $\neq$ 0，若a/b的商为整数并且余数为0，就说b整除a，或者a能被b整除，记作b|a，我们称b叫做a的因数，把a叫做b的倍数。

1.1 整除的性质(理解即可)

1. 如果b|a，c|b，则c|a 例如：8能被4整除，4能被2整除，8%2=，即8也能被2整除；
2. 如果b| a，那么 cb| ca 例如：6能被3整除，若C=2，2×6=122×3=6，12%6=0，即12能被6整除；
3. 如果 c|a，那么对于任何整数d，c|da； 例如：12能被4整除，若d=3，3×12=36，36%4=0，即36也能被4整除；
4. 如果 c|a，c|b，则对任何整数m，n，有c|ma+nb； 例如：4能被2整除，6也能被2整除，若m=3，n=4，3×4+4×6=36，36%2 = 0，即36也能被2整除；
5. 如果a| b，b| a，则a=±b。例如：5能被-5整除，-5也能被5整除。

2. 最大公因数和最小公倍数

gcd（a，b）：表示a和b的最小公倍数
特别的，如果gcd（a，b）=1，我们就说a，b互质

2.1 最大公因数的性质

1. 当b|a时，gcd（a，b）=b
   例如：6能被3整除，3和6的最大公因数为3；
2. a b的所有公因数都是 gcd(a，b）的因数
3. 若a，b是正整数，m为任一正整数，则有 gcd(am，bm）= gcd（a，b）×m
4. 若gcd（a，b）=1，c为任一正整数，则有gcd(ac，b）= gcd（c，b）
   例如：3和8的最大公因数为1，若C=2，2×3=6，6和8的最大公因数为2，2和8的最大公因数也为2
5. 若gcd（a，b）=1，b | ac，则有b|c

2.2 辗转相除法

算法原理：
如果两个整数a和b，另有整数c=a%b；
那么a和b的最大公约数就是b和c的最大公约数；即gcd(a，b）=gcd（b，C)。
辗转相除法的步骤：
1，用a去除b，得到余数r；
2. 如果余数为0，则此时的b即为最大公因数
3. 如果余数不为0，则将b作为新的被除数，r作为新的除数，继续执行第1步
4. 不断重复执行第1步和第3步，直到余数为。
示例
例如：a=46，b=32，利用转相除法求解 gcd（a，b）
46%32=14，令a=32，b=14；
32%14=4，令a=14，b= 4；
14%4=2；令a=4，b=2；
4%2=0；
当余数为0时，此时的除数即为最大公因数，所以gcd（46，32）=2
代码示例
下面的代码要求理解辗转相除法后熟练在心！！

```
#include<iostream>
using namespace std;
int gcd(int a, int b)
{
    int r;
    do
    {
        r = a % b;
        a = b;
        b = r;
    }
    while(r!=0);
    return a;
}

int main()
{
    int a, b;
    cin >> a >>b;
    cout << gcd(a,b);
    return 0;
}
```

2.3最小公倍数的概念

lcm（a，b）记作a和b的最小公倍数

3. 质数和合数

质数又称素数。一个大于1的自然数，除了1和它自身外，不能被其他自然数整除的数叫做质数，否则称为合数
注意：1既不是质数也不是合数

3.1 朴素算法

判断一个整数n是否是质数，一般情况下，只需要将2到n-1逐一进行试除，如果都不能整除，那么n就是一个质数

```
boo isPrime(int n)
{
    if(n < 2 )
        return false;
    for(int i=2; i < n; ++i)
    {
        if( n % i==0)
            return false;
    }
    return true;
}
```

优化思路
如果n是一个合数，则n可以表示a×b的形式，其中a小于等于b，那么 $a^2 \leq a \times b = n$;
所以 $a \leq \sqrt{n}$;
只需要将2到 $\sqrt{n}$ 逐一进行试除，如果都不能整除，那么n就是一个质数。

```
boo isPrime(int n)
{
    if(n < 2 )
        return false;
    for(int i=2; i < sqrt(n); ++i)
    {
        if( n % i==0)
            return false;
    }
    return true;
}
```

4. 质数筛

质数筛的定义
质数筛 （Prime Sieve） 是一种用于找出一定范围内所有质数的算法。 它通过逐步排除非质数的方 法，最终得到质数的列表。 质数筛可以高效地找出较小范围内的质数，并被广泛应用于数论、 密码学等领域。 学习质数筛的原因是他相对于其他传统的质数判断方法具有更高的效率 常用的质数筛包括：

1. 埃氏筛
2. 线性筛，又称欧拉筛

4.1 埃氏筛

埃氏筛的原理 要得到自然数n以内的全部质数，只需要将不大于 $\sqrt{n}$ 的所有质数的倍数都删除，剩下的就是质数。
示例： 请使用埃氏筛查找2到100（包括2和10）以内的所有质数并输出。
埃氏筛步骤

1. 设置一个 bool 类型的数组，用于标记数字是否是质数，将数组初始化为 false；
2. 从下标2开始，2是质数，接下来将2的倍数（自身除外） 下标位置都设置为 true，表示将这些数删除；
3. 继续向后找，3是质数，接下来将3的倍数（自身除外） 下标位置都设置为 true，表 示将这些数删除；
4. 以此类推，将 $\sqrt{100}$ 以内所有质数的倍数位置（除它自身以外） 都设置为 true
5. 最后从2开始遍历数组，如果Prime[i]为false，则i是质数，输出1。

埃氏筛的缺点

埃氏筛在筛选的过程中，某些数字会被重复筛选，降低算法的效率，
例如：
24是2的倍数，同时也是3的倍数，所以24会被重复筛选两次
30是2的倍数，也是3的倍数，同时还是5的倍数，所以30 会被重复筛选三次
有没有更便捷的求素数的方法呢

4.2线性筛

线性筛 （又称欧拉筛）算法，用于找到给定范围内的所有质数。其主要思想是每个合数只会被它的 最小质因数筛去，这样保证了每个数最多只会被处理一次，从而达到线性时间复杂度

线性筛步骤

1. 创建一个布尔数组 Prime，将所有的元素初始化为 true
2. 创建一个数组a，用于存储筛选出的质数
3. 从2开始遍历到指定的上限数n
   如果Prime[i]为true，表示i是质数，将其加入质数数组a
   对于素数数组中的每个素数p，将i * p标记为非素数，即将 isPrime[i*p]设为 false。
   如果i能够整除素数p，则停正标记过程

4. 完成遍历后，素数列表a中存储的即为从2到n之间的所有素数

```
#include<iostream>
using namespacestd;
boolPrime[101];
inta[101],n; //n表示数组a的下标
void isPrime()
{
    for (int i=2; i<=100; i++)
    {
        if (!Prime[i])
            a[n++]=i;
        for(intj=0; j<n&&i*a[j]<=100; j++)
        {
            Prime[i * a[j]] = true;
            if(i % a[j] == 0)
                break;
        }
    }
}

int main()
{
    isPrime();
    for(int i = 0; i< n;++i)
    {
        cout << a[i] << ' ';
    }
    return 0;
}
```