第 1 次编程练习报告

姓名：武桐西　　学号：2112515　　班级：信安一班

一、编程练习 1——Eratosthenes 筛法

➢ 源码部分：

```
#include<iostream>
#include<math.h>
using namespace std;

void Eratosthenes(int n, bool*& A) {
    //n为代求范围[2,n]，A保存整数表
    A = new bool[n + 1]{ false };
    for (int i = 2; i <= n; i++)
        A[i] = true;//初始化为true
    if (n < 4)
        return;//递归结束
    int n_ = (int)(sqrt(1.0 * n));
    bool* A_ = nullptr;
    Eratosthenes(n_, A_);//递归
    for (int i = 2; i <= n_; i++) {
        if (A_[i]) {//i为素数
            for (int j = 2; j <= n; j++) {
                if (j % i == 0 && j != i)
                    A[j] = false;
            }
        }
    }
}

int main() {
    int n;
    cout << "Please Input the integer n (n>1):\n";
    cin >> n;
    if (n <= 1) {
        cout << "Wrong Input!\n";
    }
    else {
        bool* A = nullptr;
```

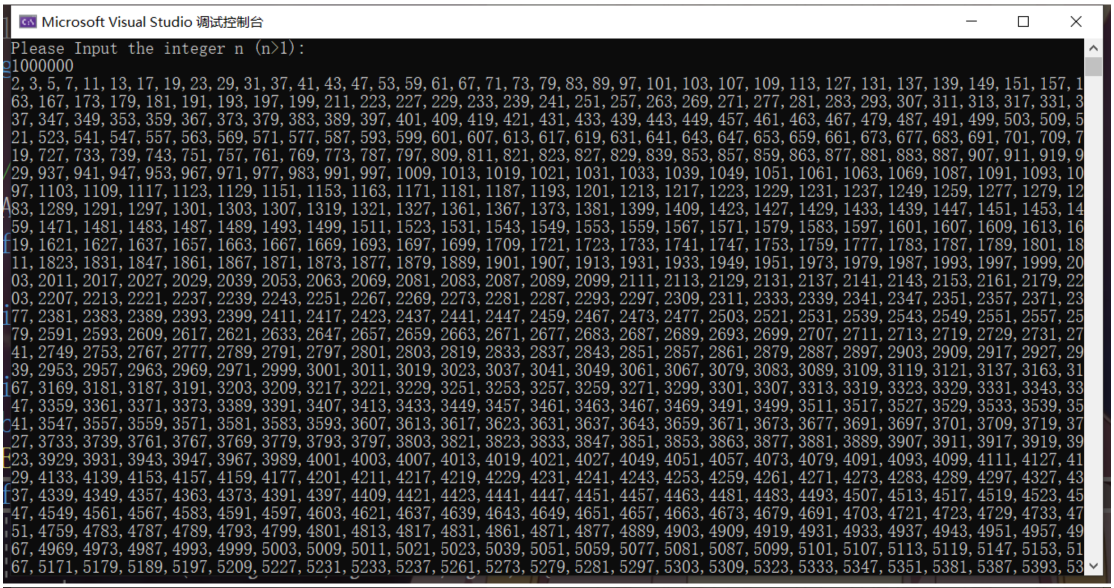
```
        Eratosthenes(n, A);
        int count = 0;//用于统计素数个数
        for (int i = 2; i <= n; i++)
            if (A[i]) {//i为素数
                count++;
                cout << i << ",";
            }
        cout << endl << "Total: " << count << endl;
    }
    return 0;
}
```

- **说明部分：**

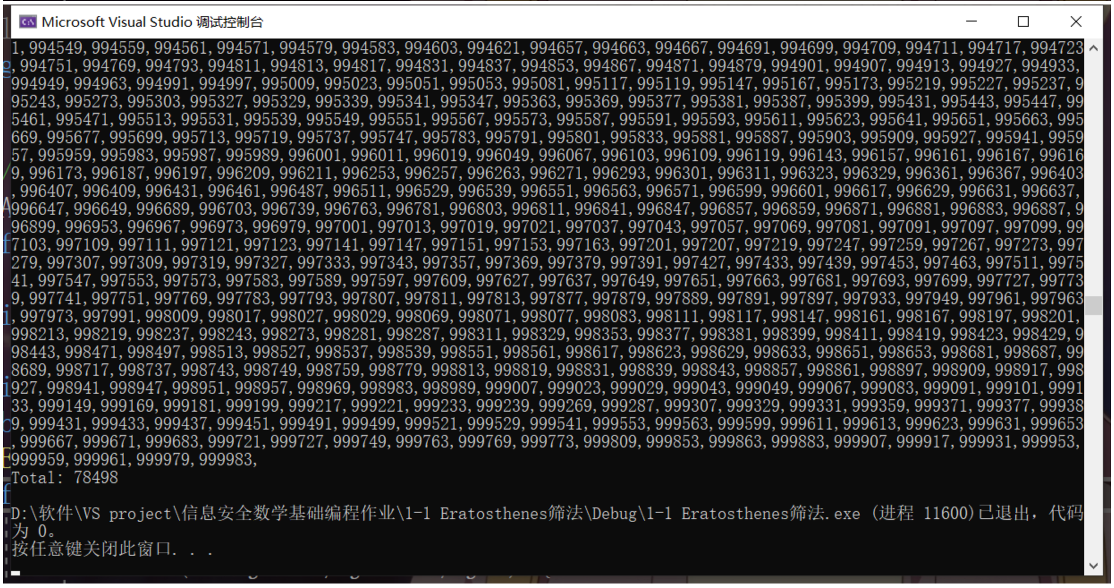
递归地调用 Eratosthenes 筛法。每次调用时，首先得到所有小于或等于$\sqrt{n}$的素数组成的表（该表可由 Eratosthenes 筛法递归得到），然后依据得到的素数表，将[2,n]的范围内的所有正整数进行筛选，去除该范围内所有素数表中的素数的倍数（该数自身除外），剩下的即为所求的[2,n]内的所有素数；递归调用的终止条件可以设置在比较容易求得[2,n]内所有素数的时候（即当 n 较小时），本程序中设置递归终止条件为$n < 4$。最终的素数个数，可以在输出时遍历素数表并同时计数即可得到。

本程序中，用 bool 类型的数组来记录素数表，其中，数组的下标i（索引）代表正整数i，其值为 true，则表示当前正整数i是素数，若为 false，则正整数i不是素数。

- ➤ 运行示例：



```
Microsoft Visual Studio 调试控制台
Please Input the integer n (n>1):
1000000
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 1
63, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 3
37, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 5
21, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 7
19, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 9
29, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 10
97, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 12
83, 1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453, 14
59, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 16
19, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 18
11, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999, 20
03, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131, 2137, 2141, 2143, 2153, 2161, 2179, 22
03, 2207, 2213, 2221, 2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293, 2297, 2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371, 23
77, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437, 2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543, 2549, 2551, 2557, 25
79, 2591, 2593, 2609, 2617, 2621, 2633, 2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713, 2719, 2729, 2731, 27
41, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819, 2833, 2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909, 2917, 2927, 29
39, 2953, 2957, 2963, 2969, 2971, 2999, 3001, 3011, 3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083, 3089, 3109, 3119, 3121, 3137, 3163, 31
67, 3169, 3181, 3187, 3191, 3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259, 3271, 3299, 3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343, 33
47, 3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413, 3433, 3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517, 3527, 3529, 3533, 3539, 35
41, 3547, 3557, 3559, 3571, 3581, 3583, 3593, 3607, 3613, 3617, 3623, 3631, 3637, 3643, 3659, 3671, 3673, 3677, 3691, 3697, 3701, 3709, 3719, 37
27, 3733, 3739, 3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911, 3917, 3919, 39
23, 3929, 3931, 3943, 3947, 3967, 3989, 4001, 4003, 4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079, 4091, 4093, 4099, 4111, 4127, 41
29, 4133, 4139, 4153, 4157, 4159, 4177, 4201, 4211, 4217, 4219, 4229, 4231, 4241, 4243, 4253, 4259, 4261, 4271, 4273, 4283, 4289, 4297, 4327, 43
37, 4339, 4349, 4357, 4363, 4373, 4391, 4397, 4409, 4421, 4423, 4441, 4447, 4451, 4457, 4463, 4481, 4483, 4493, 4507, 4513, 4517, 4519, 4523, 45
47, 4549, 4561, 4567, 4583, 4591, 4597, 4603, 4621, 4637, 4639, 4643, 4649, 4651, 4657, 4663, 4673, 4679, 4691, 4703, 4721, 4723, 4729, 4733, 47
51, 4759, 4783, 4787, 4789, 4793, 4799, 4801, 4813, 4817, 4831, 4861, 4871, 4877, 4889, 4903, 4909, 4919, 4931, 4933, 4937, 4943, 4951, 4957, 49
67, 4969, 4973, 4987, 4993, 4999, 5003, 5009, 5011, 5021, 5023, 5039, 5051, 5059, 5077, 5081, 5087, 5099, 5101, 5107, 5113, 5119, 5147, 5153, 51
67, 5171, 5179, 5189, 5197, 5209, 5227, 5231, 5233, 5237, 5261, 5273, 5279, 5281, 5297, 5303, 5309, 5323, 5333, 5347, 5351, 5381, 5387, 5393, 53
```

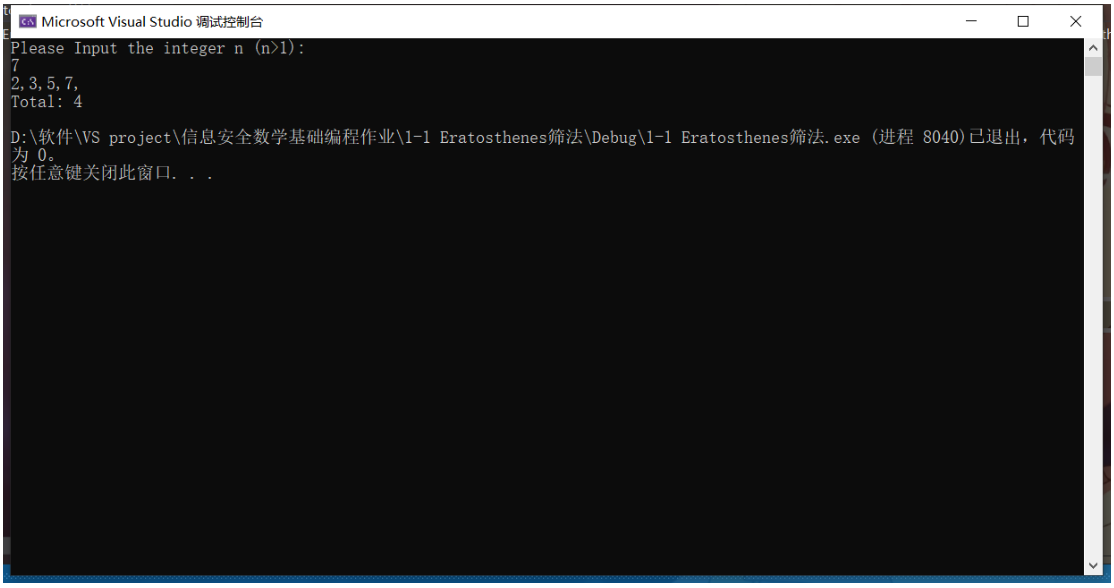


```
Microsoft Visual Studio 调试控制台
1, 994549, 994559, 994561, 994571, 994579, 994583, 994603, 994621, 994657, 994663, 994667, 994691, 994699, 994709, 994711, 994717, 994723
, 994751, 994769, 994793, 994811, 994813, 994817, 994831, 994837, 994853, 994867, 994871, 994879, 994901, 994907, 994913, 994927, 994933,
994949, 994963, 994991, 994997, 995009, 995023, 995051, 995053, 995081, 995117, 995119, 995147, 995167, 995173, 995219, 995227, 995237, 9
95243, 995273, 995303, 995327, 995329, 995339, 995341, 995347, 995363, 995369, 995377, 995381, 995387, 995399, 995431, 995443, 995447, 99
5461, 995471, 995513, 995531, 995539, 995549, 995551, 995567, 995573, 995587, 995591, 995593, 995611, 995623, 995641, 995651, 995663, 995
669, 995677, 995699, 995713, 995719, 995737, 995747, 995783, 995791, 995801, 995833, 995881, 995887, 995903, 995909, 995927, 995941, 9959
57, 995959, 995983, 995987, 995989, 996001, 996011, 996019, 996049, 996067, 996103, 996109, 996119, 996143, 996157, 996161, 996167, 99616
9, 996173, 996187, 996197, 996209, 996211, 996253, 996257, 996263, 996271, 996293, 996301, 996311, 996323, 996329, 996361, 996367, 996403
, 996407, 996409, 996431, 996461, 996487, 996511, 996529, 996539, 996551, 996563, 996571, 996599, 996601, 996617, 996629, 996631, 996637,
996647, 996649, 996689, 996703, 996739, 996763, 996781, 996803, 996811, 996841, 996847, 996857, 996859, 996871, 996881, 996883, 996887, 9
96899, 996953, 996967, 996973, 996979, 997001, 997013, 997019, 997021, 997037, 997043, 997057, 997069, 997081, 997091, 997097, 997099, 99
7103, 997109, 997111, 997121, 997123, 997141, 997147, 997151, 997153, 997163, 997201, 997207, 997219, 997247, 997259, 997267, 997273, 997
279, 997307, 997309, 997319, 997327, 997333, 997343, 997357, 997369, 997379, 997391, 997427, 997433, 997439, 997453, 997463, 997511, 9975
41, 997547, 997553, 997573, 997583, 997589, 997597, 997609, 997627, 997637, 997649, 997651, 997663, 997681, 997693, 997699, 997727, 99773
9, 997741, 997751, 997769, 997783, 997793, 997807, 997811, 997813, 997877, 997879, 997889, 997891, 997897, 997933, 997949, 997961, 997963
, 997973, 997991, 998009, 998017, 998027, 998029, 998069, 998071, 998077, 998083, 998111, 998117, 998147, 998161, 998167, 998197, 998201,
998213, 998219, 998237, 998243, 998273, 998281, 998287, 998311, 998329, 998353, 998377, 998381, 998399, 998411, 998419, 998423, 998429, 9
98443, 998471, 998497, 998513, 998527, 998537, 998539, 998551, 998561, 998617, 998623, 998629, 998633, 998651, 998653, 998681, 998687, 99
8689, 998717, 998737, 998743, 998749, 998759, 998779, 998813, 998819, 998831, 998839, 998843, 998857, 998861, 998897, 998909, 998917, 998
927, 998941, 998947, 998951, 998957, 998969, 998983, 998989, 999007, 999023, 999029, 999043, 999049, 999067, 999083, 999091, 999101, 9991
33, 999149, 999169, 999181, 999199, 999217, 999221, 999233, 999239, 999269, 999287, 999307, 999329, 999331, 999359, 999371, 999377, 99938
9, 999431, 999433, 999437, 999451, 999491, 999499, 999521, 999529, 999541, 999553, 999563, 999599, 999611, 999613, 999623, 999631, 999653
, 999667, 999671, 999683, 999721, 999727, 999749, 999763, 999769, 999773, 999809, 999853, 999863, 999883, 999907, 999917, 999931, 999953,
999959, 999961, 999979, 999983,
Total: 78498

D:\软件\VS project\信息安全数学基础编程作业\1-1 Eratosthenes筛法\Debug\1-1 Eratosthenes筛法.exe (进程 11600)已退出，代码
为 0。
按任意键关闭此窗口. . .
```


Microsoft Visual Studio 调试控制台
Please Input the integer n (n>1):
2
2,
Total: 1
D:\软件\VS project\信息安全数学基础编程作业\1-1 Eratosthenes筛法\Debug\1-1 Eratosthenes筛法.exe (进程 22840)已退出，代码为 0。
按任意键关闭此窗口. . .


Microsoft Visual Studio 调试控制台
Please Input the integer n (n>1):
4
2,3,
Total: 2
D:\软件\VS project\信息安全数学基础编程作业\1-1 Eratosthenes筛法\Debug\1-1 Eratosthenes筛法.exe (进程 13148)已退出，代码为 0。
按任意键关闭此窗口. . .


Microsoft Visual Studio 调试控制台
Please Input the integer n (n>1):
7
2,3,5,7,
Total: 4
D:\软件\VS project\信息安全数学基础编程作业\1-1 Eratosthenes筛法\Debug\1-1 Eratosthenes筛法.exe (进程 8040)已退出，代码为 0。
按任意键关闭此窗口. . .

- **其他：**

a.对比筛法与普通算法的性能差异.

普通算法需要对[2,n]内的每一个正整数根据素数定义判断其是否为素数，粗略估计其时间复杂度为$O(N^2)$，N即为前面的 n。

筛法依据数论中的定理“若n为合数，则n必有素因子 p 满足$p \leq \sqrt{n}.$”，仅需计算小于或等于$\sqrt{n}$的所有素数，并筛去除素数自身外的所有的倍数，粗略估计其时间复杂度为$O(N\log N)$，工作量大大降低，性能得到很大提升。

b.递归调用该算法求更大范围素数进行优化.

当根据 Eratosthenes 筛法求[2,n]内的所有素数时，要先求出$\left[2,\sqrt{n}\right]$内的所有素数，而当 n 很大时，$\sqrt{n}$也会很大，这时可以再次调用 Eratosthenes 筛法求$\left[2,\sqrt{n}\right]$内的所有素数，即递归调用。这样对于较大的 n，算法依然 practically effective。

c.求更大的素数（如2^{512}数量级）该方法是否适用？会引入哪些新的问题？

求更大的素数（如2^{512}数量级）该方法并不适用。首先是因为 int 类型的范围是$-2^{31}\sim 2^{31}-1$，更大的 n 会使 int 型溢出，更大的 n 会导致即使使用 unsigned long long int 也无济于事；其次是因为在递归调用的时候，由于需要递归的次数过多，会导致系统栈的溢出；再次，可能会使程序的运行时间过长。

二、编程练习 2——最大公因数和最小公倍数

➢ 源码部分：

```
#include<iostream>
using namespace std;

long long int gcd(long long int a, long long int b) {
    //辗转相除法求最大公因数
    a = a > 0 ? a : -a;//转为正数
    b = b > 0 ? b : -b;//转为正数
    if (a < b) {//令a为较大者，b为较小者
        long long int tmp = a;
        a = b;
        b = tmp;
    }
    while (a % b) {//余数不为0，继续循环
        long long int r = a % b;
        a = b;
        b = r;
    }
    return b;
}

long long int lcm(long long int a, long long int b) {
    a = a > 0 ? a : -a;//转为正数
    b = b > 0 ? b : -b;//转为正数
    return (a * b) / gcd(a, b);
}

int main() {
    long long int a, b;
    cout << "Please Input two non-zero integers:\n";
    cin >> a >> b;
    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    cout << "gcd(a,b)=" << gcd(a, b) << endl;
    cout << "lcm(a,b)=" << lcm(a, b) << endl;
    return 0;
}
```

➢ 说明部分：

求两个非零整数 a 和 b 的最大公因数和最小公倍数。

首先将 a 与 b 均取其绝对值，不妨令较大者为 a，然后利用辗转相除法，最终得到的最后一个不为零的余数即为所求的 a 与 b 的最大公因数。

得到最大公因数后，可利用$lcm(a,b)=\frac{|ab|}{\gcd(a,b)}$求 a 与 b 的最小公倍数。

➢ **运行示例：**

三、编程练习 3——算术基本定理

➢ 源码部分：

```
#include<iostream>
#include<math.h>
using namespace std;

void Eratosthenes(int n, bool*& A) {
    //n为代求范围[2,n]，A保存整数表
    A = new bool[n + 1]{ false };
    for (int i = 2; i <= n; i++)
        A[i] = true;//初始化为true
    if (n < 4)
        return;//递归结束
    int n_ = (int)(sqrt(1.0 * n));
    bool* A_ = nullptr;
    Eratosthenes(n_, A_);
    for (int i = 2; i <= n_; i++) {
        if (A_[i]) {//i为素数
            for (int j = 2; j <= n; j++) {
                if (j % i == 0 && j != i)
                    A[j] = false;
            }
        }
    }
}

void PrimeFactor(int n) {
    if (n == 1) {
        cout << n << "=2^0\n";
        return;
    }
    //先用Eratosthenes筛法求[2,n]内所有素数
    bool* A = nullptr;
    Eratosthenes(n, A);
    int a = n;
    cout << n << "=";
    bool tag = true;//控制*的输出
    for (int i = 2; i <= n; i++) {
        if (A[i]) {//i为素数
            int count = 0;//记录素因子指数
            while (a % i == 0) {
```

```
                count++;
                a /= i;
            }
            if (count) {//count非零
                if (tag) {//第一次输出，前面不需要输出*
                    cout << i << "^" << count;
                    tag = false;
                }
                else {//非第一次输出，前面需输出*
                    cout << "*";
                    cout << i << "^" << count;
                }
            }
        }
    }
}

int main() {
    cout << "Please Input n (n>0):\n";
    int n;
    cin >> n;
    if (n <= 0)
        cout << "Wrong Input!\n";
    else
        PrimeFactor(n);
    return 0;
}
```

- **说明部分：**

欲找出一个正整数 n 的标准分解式，首先可以用 Eratosthenes 筛法求[2,n]内所有素数，然后按照素数**从小到大**的顺序，求出 n 中包含多少个当前素数，此即该素数对应的指数（若为 0，则不输出），按照一定规律输出（可用一个 tag 来控制*的输出），最终可得 n 的标准分解式。

与此同时，要注意$n = 1$的特殊情况。若$n = 1$，则本程序输出$1 = 2^0$。

➢ 运行示例：


Microsoft Visual Studio 调试控制台
Please Input n (n>0):
888
888=2^3*3^1*37^1
D:\软件\VS project\信息安全数学基础编程作业\1-3 算术基本定理\Debug\1-3 算术基本定理.exe (进程 14864)已退出，代码为 0。
按任意键关闭此窗口. . .


Microsoft Visual Studio 调试控制台
Please Input n (n>0):
22345680
22345680=2^4*3^1*5^1*7^1*47^1*283^1
D:\软件\VS project\信息安全数学基础编程作业\1-3 算术基本定理\Debug\1-3 算术基本定理.exe (进程 16832)已退出，代码为 0。
按任意键关闭此窗口. . .