```cpp
//日期: 2018/ 时间:
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//素数的判断。要判断一个数n是否为素数，需要判断n能否被2,3,4.....n-1整除。这样复杂度为O(n)
//而事实上，只需要判断n能否被2,3......sqrt(n) 整除即可。这样复杂度就是O(sqrt(n))
bool isPrime(int n){
    if(n <= 1)  return false;
    int sqr = (int)sqrt(1.0*n);          //根号
    for(int i=2;i<=sqr;i++){
        if(n%i == 0)    return false;
    }
    return true;
}

//或者
bool isPrime1(int n){
    if(n <= 1)  return false;
    for(long long i = 2; i*i <= n;i++){     //防止i越界，使用long long型变量
        if(n % i == 0)  return false;
    }
    return true;
}

//素数表的获取，打印1~n范围内素数表的方法，即从1~n进行枚举，判断每个数是否是素数。枚举的复杂度是O(n),判断的复杂度是O(sqrt(n))
//因此总的复杂度是O(n*sqrt(n)),这一方法对于n在100000范围内是可以承受的。
const int maxn = 3;
int prime[maxn],pNum=0;     //prime数组存放所有的素数，pNum为素数的个数
bool p[maxn] ={0};          //p[i] == true 表示i是素数

void Find_Prime(){
    for(int i=1;i<maxn;i++){
        if(isPrime(i)==true){
            prime[pNum++]=i;
            p[i] = true;
        }
    }
}

//如果出现更大范围的素数表，以下代码复杂度为O(nloglogn).埃氏筛法。
//算法从小到大枚举所有的数，对每一个素数，筛去它的所有倍数，剩下的就是素数了。从小到大达到某个数a时
//如果a没有被前面的数筛去，那么a一定是素数。
void eFind_Prime(){
    for(int i=2;i<maxn;i++){
        if(p[i]==false){
            prime[pNum++]=i;
            for(int j=i+i;j<maxn;j+=i){
                //筛去所有的i的倍数，循环条件不能写成j<=maxn
                p[j] = true;
            }
        }
    }
```

```
54  }
55
56  int main(){
57      Find_Prime();
58      for(int i=0;i<pNum;i++)
59          printf("%d ",prime[i]);
60
61      return 0;
62  }
63
64
```