Author: **Brandon Lin**, *October 9, 2023*, Shanghai

求给定范围中的所有素数

定义

- 范围 `max_num`
- 被检验是否是素数的数 `num2judge
- 被除数 `div`
- 是素数旗帜 `is_prime`
- 素数列表 `prime_list`

算法 1：原始遍历

1. 创建素数列表，项数为 `max_num
2. 若 `max_num` = 2，直接输出 `prime_list = 2`
3. 对于每一个数 `num2judge` 从 3 到 `max_num` 执行
    1. 初始化：`is_prime` = 1
    2. 对于被除数 `div` 从 2 到 `num2judge`-1 执行
       2. 若 `num2judge` 能够被 `div` 整除：跳出循环，`is_prime = 0`
       3. 反之，继续
    3. 若仍然 `is_prime = 1`，将其添加至 `prime_list` 末端
4. 输出 `prime_list`

算法 1 的分析

1. 在第 3-2 “对于被除数 `div` 从 2 到 `num2judge`-1 执行” 步执行时，对于一个被判断的数 n，我们并不需要将其与 2 至 $n-1$ 全部除一遍，执行至 $E(\sqrt{n})$ 即可。
2. 在第 3-2 “对于被除数 `div` 从 2 到 `num2judge`-1 执行” 步执行时，对于一个被判断的数 n，我们发现，除了 2 之外的素数都不是偶数，因此只用判断所有奇数
3. 与 2 同理，所有 3 的倍数都不用考虑……
4. 事实上在持续进行运算的时候，我们将每一个被判断的数与之前确认的所有素数相除即可。

算法 2——基于 1 的改进

1. 创建素数列表，项数为 `max_num
2. 若 `max_num` = 2，直接输出 `prime_list = 2`
3. 对于每一个数 `num2judge` 从 3 到 `max_num` 执行
    1. 初始化：`is_prime` = 1
    2. 对于被除数 `div` 从 2 到 `sqrt(num2judge)` 执行
       2. 若 `num2judge` 能够被 `div` 整除：跳出循环，`is_prime = 0`
       3. 反之，继续
    3. 若仍然 `is_prime = 1`，将其添加至 `prime_list` 末端

4. 输出 `prime_list`

> **Note**
>
> 基于算法 1 实现，修改了被除数的上限：`num2judge` -1 修改为 `sqrt(num2judge)`，同时运用强制类型转换以保险。

算法 3——基于 2 的改进【最终版】

1. 创建素数列表，项数为 `max_num
2. 若 `max_num` = 2，直接输出 `prime_list = 2`
3. 对于每一个数 `num2judge` 从 3 到 `max_num` 执行
    1. 初始化：`is_prime` = 1
    2. 对于被除数 `div` 从 `prime_list[0]` 到 `prime_list[-1]` 执行，同时当 `prime_list[i]` 的值大于 `sqrt(num2judge)` 随即终止
    2. 若 `num2judge` 能够被 `div` 整除：跳出循环，`is_prime = 0`
    3. 反之，继续
    3. 若仍然 `is_prime = 1`，将其添加至 `prime_list` 末端
4. 输出 `prime_list`

> **Note**
>
> 结合算法 1 以及 1，4两点改进。我是用指针来实现的。（其实也可以加个计数器，同理）

其他实现方法

1. 读取含有从 2 到 `MAX_NUM` 的素数文件
2. 对于用户输入 `n`，将文件中小于 `n` 的全部输出

> *简单粗暴的实现！效率遥遥领先*

算法比较

	算法 1	算法 2	算法 3
时间复杂度	$O(n^2)$	$O(n\sqrt{n})$	难以计算（毕竟无法判断有多少素数），比算法 2 好

附录

算法 1 实现

```c
#include <stdio.h>
#include <assert.h>

int main() {
    int max_num;
    int num2judge;
    int div;
    int is_prime;

    // Receive the input
    printf("Input the range: (Integer > 1) ");
    scanf("%d", &max_num);
    assert ( max_num > 1 );

    // Create the list of primes, initialisation
    int prime_list[max_num];
    for ( int index = 0; index < max_num; index++ ) {
        prime_list[index] = 0;
    }

    prime_list[0] = 2;
    int * prime_pos = prime_list + 1;

    // Operation for each numer in the range;
    for( num2judge = 3; num2judge <= max_num; num2judge++ ) {
        /* Initialisation */
        is_prime = 1;

        // Try for each div
        for( div = 2; div <= num2judge - 1; div++ ) {
            if ( num2judge % div == 0 ) {
                is_prime = 0;
                break;
            }
        }

        if( is_prime ) {
            *prime_pos = num2judge;
            prime_pos++;
        }
    }

    // Print the prime list
    printf("The list of primes: ");
    for( int index = 0; index < max_num; index++ ) {
        if ( prime_list[index] ) {
            printf("%d ", prime_list[index]);
        }
        else { break; }
    }
    return 0;
}
```

算法 2 实现

```c
#include <stdio.h>
#include <assert.h>
#include <math.h>

int main() {
    int max_num; // 范围
    int num2judge; // 被检验是否是素数的数
    int div; // 被除数
    int is_prime; // 是素数旗帜

    // Receive the input
    printf("Input the range: (Integer > 1) ");
    scanf("%d", &max_num);
    assert ( max_num > 1 );

    // Create the list of primes, initialisation
    int prime_list[max_num]; // 素数列表
    for ( int index = 0; index < max_num; index++ ) {
        prime_list[index] = 0;
    }

    prime_list[0] = 2;
    int * prime_pos = prime_list + 1;

    // Operation for each numer in the range;
    for( num2judge = 3; num2judge <= max_num; num2judge++ ) {
        /* Initialisation */
        is_prime = 1;

        // Try for each div
        for( div = 2; div <= (int)(sqrt(num2judge)); div++ ) {
            if ( num2judge % div == 0 ) {
                is_prime = 0;
                break;
            }
        }

        if( is_prime ) {
            *prime_pos = num2judge;
            prime_pos++;
        }
    }

    // Print the prime list
    printf("The list of primes: ");
    for( int index = 0; index < max_num; index++ ) {
        if ( prime_list[index] ) {
            printf("%d ", prime_list[index]);
        }
        else { break; }
```

```
    }
    return 0;
}
```

算法 3 【最终版】的实现

```
#include <stdio.h>
#include <assert.h>
#include <math.h>

int main() {
    int max_num; // 范围
    int num2judge; // 被检验是否是素数的数
    int div; // 被除数
    int is_prime; // 是素数旗帜

    // Receive the input
    printf("Input the range: (Integer > 1) ");
    scanf("%d", &max_num);
    assert ( max_num > 1 );

    // Create the list of primes, initialisation
    int prime_list[max_num]; // 素数列表
    for ( int index = 0; index < max_num; index++ ) {
        prime_list[index] = 0;
    }

    prime_list[0] = 2;
    int * prime_pos = prime_list; // Pointers to the prime_list

    // Operation for each nubmer in the range;
    for( num2judge = 3; num2judge <= max_num; num2judge++ ) {
        /* Initialisation */
        is_prime = 1;

        // Try for each div
        for ( int * prime_list_search = prime_list;
              prime_list_search <= prime_pos &&
              *prime_list_search <= (int)(sqrt(max_num)); // 2 judgements
              prime_list_search++ ) {
                if ( num2judge % *prime_list_search == 0 ) {
                    is_prime = 0;
                    break;
                }
              }

        if( is_prime ) {
            *++prime_pos = num2judge;
        }
    }
```

```
    // Print the prime list
    printf("The list of primes: ");
    for ( int * prime_list_search = prime_list;
          prime_list_search <= prime_pos;
          prime_list_search++ ) {
            printf("%d ", *prime_list_search);
          }

    return 0;
}
```