```c
/*
Author:  Pate Williams (c) 1997 - 2022
*/
#include "Number_Theory.h"

int Jacobi(long long a, long long n) {
    if (a == 0)
        return 0;
    if (a == 1)
        return 1;
    int e = 0;
    long long a1 = a;
    long long quotient = a1 / 2;
    long long remainder = a1 % 2;
    while (remainder == 0) {
        e++;
        a1 = quotient;
        quotient = a1 / 2;
        remainder = a1 % 2;
    }
    int s = 0;
    if (e % 2 == 0)
        s = 1;
    else {
        long long mod8 = n % 8;
        if (mod8 == 1 || mod8 == 7)
            s = +1;
        if (mod8 == 3 || mod8 == 5)
            s = -1;
    }
    long long mod4 = n % 4;
    long long  a14 = a1 % 4;
    if (mod4 == 3 && a14 == 3)
        s = -s;
    long long n1 = n % a1;
    return s * Jacobi(n1, a1);
}

int Miller_Rabin(long long n, int t) {
    long i, j, s = 0;
    long long a, m, n1 = n - 1, r = n1, y;

    if (n == 2 || n == 3)
        return 1;
    m = n % 2;
    if (m == 0)
        return 0;
    m = r % 2;
    while (m == 0)
    {
        r = r / 2;
        m = r % 2;
```

```
53          s++;
54      }
55      for (i = 0; i < t; i++)
56      {
57          a = random_range(2, n - 2);
58          y = pow_mod_M(a, r, n);
59          if (y < 0)
60              y += n;
61          if (y != 1 && y != n1) {
62              j = 1;
63              while (j <= s - 1 && y != n1) {
64                  y = (y * y) % n;
65                  if (y == 1)
66                      return 0;
67                  j++;
68              }
69              if (y != n1)
70                  return 0;
71          }
72      }
73      return 1;
74  }
75
76  long long GCD(long long a, long long b) {
77      long long r;
78
79      while (b != 0) {
80          r = a % b;
81          a = b;
82          b = r;
83      }
84      return a;
85  }
86
87  long get_bit(long i, long *sieve) {
88      long b = i % BITS_PER_LONG;
89      long c = i / BITS_PER_LONG;
90
91      return (sieve[c] >> (BITS_PER_LONG_1 - b)) & 1;
92  }
93
94  void set_bit(long i, long v, long *sieve) {
95      long b = i % BITS_PER_LONG;
96      long c = i / BITS_PER_LONG;
97      long mask = 1 << (BITS_PER_LONG_1 - b);
98
99      if (v == 1)
100         sieve[c] |= mask;
101     else
102         sieve[c] &= ~mask;
103 }
104
```

```c
void get_bits(long long b, int *number, int *bits) {
    int i = 0;
    do {
        bits[i++] = b % 2;
        b = b / 2;
    } while (b > 0);
    *number = i;
}

long long pow_mod_S(long long x, long long b, long long n) {
    int bits[64], i, l;
    long long z = 1;

    get_bits(b, &l, bits);
    for (i = l - 1; i >= 0; i--) {
        z = (z * z) % n;
        if (z < 0)
            z += n;
        if (bits[i] == 1) {
            z = (z * x) % n;
            if (z < 0)
                z += n;
        }
    }
    return z;
}

long long pow_mod_M(long long a, long long k, long long n) {
    int t, bits[64];
    long long A = a, b = 1;

    if (k == 0)
        return b;
    get_bits(k, &t, bits);
    if (bits[0] == 1)
        b = a;
    for (int i = 1; i < t; i++) {
        A = (A * A) % n;
        if (bits[i] == 1)
            b = (A * b) % n;
    }
    return b;
}

long long random_range(
    long long lower, long long upper)
{
    long long r;

    while (1) {
        r = lower + (long long)((double)(upper - lower)
            * ((double)rand() / RAND_MAX));
```

```
157         if (r >= lower && r <= upper)
158             return r;
159     }
160 }
161
162 void Greek_sieve(
163     long bound,
164     long psize,
165     long *sieve,
166     long *prime) {
167     long c, i, inc, p = 0;
168
169     set_bit(0, 0, sieve);
170     set_bit(1, 0, sieve);
171     set_bit(2, 1, sieve);
172     for (i = 3; i <= bound; i++)
173         set_bit(i, i & 1, sieve);
174     c = 3;
175     do {
176         i = c * c, inc = c + c;
177         while (i <= bound) {
178             set_bit(i, 0, sieve);
179             i += inc;
180         }
181         c += 2;
182         while (!get_bit(c, sieve)) c++;
183     } while (c * c <= bound);
184     for (i = 0; i < psize; i++) {
185         while (!get_bit(p, sieve)) p++;
186         prime[i] = p++;
187     }
188 }
189
190 void Greek_sieve_1(long *sieve, long *prime) {
191     Greek_sieve(
192         MAX_SIEVE_1,
193         PRIME_SIZE_1,
194         sieve, prime);
195 }
196 void Greek_sieve_2(long *sieve, long *prime) {
197     Greek_sieve(
198         MAX_SIEVE_2,
199         PRIME_SIZE_2,
200         sieve, prime);
201 }
```