```
/*
Author:  Pate Williams (c) 1997 - 2022
*/
#pragma once
#include <stdlib.h>

#define BITS_PER_LONG   32
#define BITS_PER_LONG_1 31
#define SIEVE_SIZE_1 (MAX_SIEVE_1 / BITS_PER_LONG + 1)
#define MAX_SIEVE_1 2000000l
#define PRIME_SIZE_1 148933l
#define SIEVE_SIZE_2 (MAX_SIEVE_2 / BITS_PER_LONG + 1)
#define MAX_SIEVE_2 100000000l
#define PRIME_SIZE_2 5084753l

int Jacobi(long long a, long long n);
int Miller_Rabin(long long n, int t);
/*
Probabilistic Prime Number Test
returns 1 for prime
returns 0 for composite number
t is the security parameter t = 20 is good
"Handbook of Applied Cryptography"
Alfred J. Menezes among others
4.24 Algorithm page 139
*/
void Greek_sieve(
    long bound,
    long psize,
    long *sieve,
    long *prime);
void Greek_sieve_1(
    long *sieve,
    long *prime);
void Greek_sieve_2(
    long *sieve,
    long *prime);
/*
Sieves of Eratosthenes
100,000,000 5,000,000 9,000,000 maximum sizes
*/
long long GCD(long long a, long long b);
/*
Greatest Common Divisor
*/
void get_bits(long long b, int *number, int *bits);
long long pow_mod_S(long long x, long long b, long long n);
long long pow_mod_M(long long x, long long b, long long n);
/*
Raising a large number to a possibly large power taking
the modulus as you go get_bits helper function S = Stinson,
M = Menezes
```

```
53  */
54  long long random_range(long long lower, long long upper);
55  /*
56  returns a random number, r, in the range lower <= r << upper
57  */
```