```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <inttypes.h>
#include <gmp.h>

void set_n(mpz_t n, mpz_t k, unsigned long m)
//sets n to 2 * k * 7^m - 1
{
  unsigned int i;

  mpz_mul_ui(n, k, 2); // n = 2k
  mpz_sub_ui(n, n, 1); // n = 2k - 1
  // n = 2k * (7^m) - 1
  for (i=0; i < m; ++i) {
    // n = (n + 1) * 7 - 1
    mpz_add_ui(n, n, 1);
    mpz_mul_ui(n, n, 7);
    mpz_sub_ui(n, n, 1);
  }
}

void luke_k(mpz_t k, mpz_t ru, mpz_t rv, mpz_t n)
//finds U_k and V_k
{
  mpz_t temp, qs, rvv, ruv;
  size_t offs;
  unsigned int d;

  mpz_inits(temp, qs, rvv, ruv, NULL);
  mpz_set_ui(rv, 42); //
  mpz_mod(rv, rv, n);
  mpz_set_ui(ru, 1);
  mpz_set_si(qs, -7);
  mpz_mod(qs, qs, n);
  offs = mpz_sizeinbase(k, 2) - 1;
  if (offs == 0) return; //k is either 0 or 1

  for (; offs > 0; --offs) {
    --offs;
    d = mpz_tstbit(k, offs);
    mpz_mul(rvv, rv, rv);
    mpz_mul(ruv, ru, rv);
    if (d == 0) {
      //rv = rv ^ 2 - 2 * Q ^ s
      mpz_mul_ui(rv, qs, 2);
```

```
      mpz_sub(rv, rvv, rv);
      //ru = ru * rv
      mpz_set(ru, ruv);
    }
    else {
      //rv = 21 * (rv ^ 2 - 2 * Q ^ s) + 896 * rv * ru
      mpz_mul_ui(rv, qs, 2); // rv = q^s * 2
      mpz_sub(rv, rvv, rv); // rv = rv * rv - q^s * 2
      mpz_mul_ui(rv, rv, 21); // rv = (rv*rv - 2q^s) * 21
      mpz_mul_ui(temp, ruv, 896); // temp = 896 * ru * rv
      mpz_add(rv, rv, temp); // rv = (rv*rv - 2q^s) * 21 + 896 * ru * rv
      //ru = ru * rv + (rv ^ 2) / 2 + 20 * rv * ru - Q ^ s
      if (mpz_odd_p(rvv)) {
        mpz_add(rvv, rvv, n);
      }
      mpz_divexact_ui(ru, rvv, 2);
      mpz_sub(ru, ru, qs);
      mpz_mul_ui(temp, ruv, 20);
      mpz_add(ru, ru, temp);
      mpz_add(ru, ru, ruv);
    }
    mpz_mod(rv, rv, n);
    mpz_mod(ru, ru, n);
    mpz_mul(qs, qs, qs);
    if (d != 0)
      mpz_mul_si(qs, qs, -7);
    ++offs;
  }
  mpz_clears(temp, qs, rvv, ruv, NULL);
}

void luke_x2(mpz_t v, mpz_t u, mpz_t n, mpz_t N)
gets U_2k and V_2k for U_k and V_k
{
  mpz_t temp;

  mpz_init(temp);
  mpz_set_si(temp, -7);

  //compute U_2n
  mpz_mul(u, u, v);
  mpz_mod(u, u, N);
  //compute V_2n
  mpz_powm(temp, temp, n, N);
  mpz_mul_ui(temp, temp, 2);
  mpz_mul(v, v, v);
```

```
  mpz_sub(v, v, temp);
  mpz_mod(v, v, N);
  mpz_clear(temp);
}

void luke_x7(mpz_t u, mpz_t v, mpz_t n, mpz_t N)
gets U_7k and V_7k for U_k and V_k
{
  mpz_t v2, qn, v_3n, v_4n, u_4n, temp;

  mpz_inits(v2, qn, v_3n, v_4n, u_4n, temp, NULL);
  mpz_pow_ui(v2, v, 2);
  mpz_set_si(qn, -7);
  mpz_powm(qn, qn, n, N);

  //V3n = Vn(Vn ^ 2 - 3 * Q ^ n)
  mpz_mul_ui(v_3n, qn, 3);
  mpz_sub(v_3n, v2, v_3n);
  mpz_mul(v_3n, v, v_3n);
  mpz_mod(v_3n, v_3n, N);
  //V4n = (Vn ^ 2 - 2 * Q ^ n) ^ 2 - 2 * Q ^ 2n
  mpz_mul_ui(u_4n, qn, 2);
  mpz_sub(u_4n, v2, u_4n);
  mpz_pow_ui(v_4n, u_4n, 2);
  mpz_pow_ui(temp, qn, 2);
  mpz_mul_ui(temp, temp, 2);
  mpz_sub(v_4n, v_4n, temp);
  mpz_mod(v_4n, v_4n, N);
  //U4n = Un * Vn * V2n
  mpz_mul_ui(u_4n, qn, 2);
  mpz_sub(u_4n, v2, u_4n);
  mpz_mul(u_4n, v, u_4n);
  mpz_mul(u_4n, u, u_4n);
  mpz_mod(u_4n, u_4n, N);
  //V7n = V3n * V4n - q ^ n * Vn
  mpz_pow_ui(temp, qn, 3);
  mpz_mul(temp, temp, v);
  mpz_mul(v, v_3n, v_4n);
  mpz_sub(v, v, temp);
  mpz_mod(v, v, N);
  //U7n = V3n * U4n - Q ^ n * Un
  mpz_pow_ui(temp, qn, 3);
  mpz_mul(temp, temp, u);
  mpz_mul(u, v_3n, u_4n);
  mpz_sub(u, u, temp);
  mpz_mod(u, u, N);
```

```
  mpz_clears(v2, qn, v_3n, v_4n, u_4n, temp, NULL);
}

int is_prime(mpz_t k, unsigned long m)
{
  mpz_t n, u, v, l, g;
  char *mesg;
  unsigned int i, t;
  int res;
  time_t t1, t2, temp_time;

  mpz_inits(n, u, v, l, g, NULL);
  t1 = time(NULL);
  set_n(n, k, m);

  // get Vk and Uk
  luke_k(k, u, v, n);
  temp_time = t2 - t1;

  // get V(N/2p)
  mpz_set(l, k);
  for (i = 1; i < m ; i++) {
    luke_x7(u, v, l, n);
    mpz_mul_ui(l, l, 7);
  }

  if (mpz_cmp_ui(v, 0) != 0){
    luke_x2(v, u, l, n); //V(N/p), U(N/p)
    mpz_mul_ui(l, l, 2);
    mpz_gcd(g, u, n);
    if (mpz_cmp_ui(g, 1) == 0) {
      // (U(N/p), N) = 0
      luke_x7(u, v, l, n); //V(N), U(N)
      mpz_mul_ui(l, l, 7);
      if (mpz_divisible_p(u, n))
        // N is prime
        res = 1;
      else
        // N is not prime
        res = 0;
    } else {
      // N is not prime
      res = 0;
    }
  } else {
    // result is undefined
```

```
    res = 2;
  }
  t2 = time(NULL);
  mesg = (res == 1) ? "prime" : "undefined";
  if (res != 0) {
    printf("(%s, %d) %d %s\n", mpz_get_str(NULL, 10, k), m,
                               mpz_sizeinbase(n, 10), mpz_get_str(NULL, 10, n
    printf("%s %d\n", mesg, (int)(t2 - t1));
    fflush(stdout);
  }

  mpz_clears(n, u, v, l, g, NULL);
  return res;
}

int main(int argc, char **argv)
//args: m_lower_bound m_higher_bound k_higher_bound k_lower_bound
//all args are optional, default values for them are:
//m_lower_bound 1; m_higher_bound infinity; k_lower_bound 1; k_higher_bound i
{
  mpz_t k, p, temp;
  unsigned long m, res, lower_bound = 1, upper_bound = 10, k_bound = 0, k_l_bo

  if (argc >= 2)
    lower_bound = atol(argv[1]);
  if (argc >= 3)
    upper_bound = atol(argv[2]);
  if (argc >= 4)
    k_bound = atol(argv[3]);
  if (argc >= 5)
    k_l_bound = atol(argv[4]);
  mpz_inits(k, p, temp, NULL);

  if (k_bound == 0) {
  for (m = lower_bound; m <= upper_bound; ++m)
    for (mpz_set_ui(k, 1); mpz_mul_ui(temp, k, 2), mpz_ui_pow_ui(p, 7, m), mp
      is_prime(k,m);
  }
  else {
  for (m = lower_bound; m <= upper_bound; ++m)
    for (mpz_set_ui(k, k_l_bound); mpz_cmp_ui(k, k_bound) < 0 && (mpz_mul_ui(
      is_prime(k,m);
  }

  mpz_clears(k, p, temp, NULL);
  return 0;
```

}