```c
/*
** Author: Pate Williams (c) 1997 - 2024
** Single precision Goldwasser-Kilian primality test.
** See "A Course in Computational Algebraic Number
** Theory" by Henri Cohen Algorithm 9.2.3 page 470.
 */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MaxB0 10000000
#define ssize (MaxB0 / 32 + 1)
#define BITS_PER_LONG  32
#define BITS_PER_LONG_1 31
#define SIEVE_LIMIT 9999889
#define SIEVE_COUNT 664579
#define SIEVE_SIZE ssize

struct point { long long x, y; };
long sieve[SIEVE_SIZE];

long get_bit(long i, long* sieve)
{
    long b = (int)(i % BITS_PER_LONG);
    long c = (int)(i / BITS_PER_LONG);

    return (sieve[c] >> (BITS_PER_LONG_1 - b)) & 1L;
}

void set_bit(long i, long v, long* sieve)
{
    long b = (int)(i % BITS_PER_LONG);
    long c = (int)(i / BITS_PER_LONG);
    long mask = 1L << (BITS_PER_LONG_1 - b);

    if (v == 1)
        sieve[c] |= mask;
    else
        sieve[c] &= ~mask;
}

void Sieve(long n)
{
    long c, i, inc;

    set_bit(0, 0, sieve);
    set_bit(1, 0, sieve);
    set_bit(2, 1, sieve);
    for (i = 3; i <= n; i++)
```

```
53          set_bit(i, i & 1, sieve);
54      c = 3;
55      do {
56          i = c * c, inc = c + c;
57          while (i <= n) {
58              set_bit(i, 0, sieve);
59              i += inc;
60          }
61          c += 2;
62          while (!get_bit(c, sieve)) c++;
63      } while (c * c <= n);
64  }
65
66  long long gcd(long long a, long long b)
67  /* returns greatest common divisor of a and b */
68  {
69      long long r;
70
71      if (a < 0) a = -a;
72      if (b < 0) b = -b;
73      while (b > 0)
74          r = a % b, a = b, b = r;
75      return a;
76  }
77
78  void Euclid_extended(
79      long long a, long long b,
80      long long* u, long long* v,
81      long long* d)
82  {
83      long long q, t1, t3, v1, v3;
84
85      *u = 1;
86      *d = a;
87
88      if (b == 0) {
89          *v = 0;
90          return;
91      }
92
93      v1 = 0, v3 = b;
94  #ifdef DEBUG
95      printf_s("---------------------------------\n");
96      printf_s(" q    t3   *u   *d   t1   v1   v3\n");
97      printf_s("---------------------------------\n");
98  #endif
99      while (v3 != 0) {
100         q = *d / v3;
101         t3 = *d - q * v3;
102         t1 = *u - q * v1;
103         *u = v1, * d = v3;
104 #ifdef DEBUG
```

```
105         printf_s("%4ld %4ld %4ld ", q, t3, *u);
106         printf_s("%4ld %4ld %4ld %4ld\n", *d, t1, v1, v3);
107 #endif
108         v1 = t1, v3 = t3;
109     }
110     *v = (*d - a * *u) / b;
111 #ifdef DEBUG
112     printf_s("---------------------------------\n");
113 #endif
114 }
115
116 long long inverse(long long a, long long b)
117 /* returns inverse of a modulo b or 0 if it does not exist */
118 {
119     long long d, u, v;
120
121     Euclid_extended(a, b, &u, &v, &d);
122     if (d == 1) return u;
123     return 0;
124 }
125
126 int addition_1(long long n,
127     struct point P1,
128     struct point P2,
129     struct point* P3)
130 /* returns 1 if P1 = - P2 therefore P1 + P2 = O, */
131 /* 0 otherwise P1 != P2 */
132 {
133     long long delta_x = (P2.x - P1.x) % n;
134     long long delta_y = (P2.y - P1.y) % n;
135     long long m;
136
137     if (P1.x == P2.x && P1.y == -P2.y) {
138         P3->x = 0, P3->y = 1;
139         return 1;
140     }
141     /* calculate m = (y2 - y1)(x2 - x1) ^ - 1 mod n */
142     if (delta_x < 0) delta_x += n;
143     if (delta_y < 0) delta_y += n;
144     m = delta_y * inverse(delta_x, n) % n;
145     /* calculate x3 = m ^ 2 - (x1 + x2) mod n */
146     P3->x = (m * m - (P1.x + P2.x)) % n;
147     /* calculate y3 = m(x1 - x3) - y1 mod n */
148     P3->y = (m * (P1.x - P3->x) - P1.y) % n;
149     return 0;
150 }
151
152 void addition_2(
153     long long a,
154     long long n,
155     struct point P1,
156     struct point* P3)
```

```c
/* P1 == P2 */
{
    long long m;
    /* calculate m = (3x1 ^ 2 + a)(2y1) ^ - 1 mod n */
    m = (3 * P1.x * P1.x + a) * inverse(2 * P1.y, n) % n;
    /* calculate x3 = m ^ 2 - 2x1 mod n */
    P3->x = (m * m - 2 * P1.x) % n;
    /* calculate y3 = m(x1 - x3) - y1 mod n */
    P3->y = (m * (P1.x - P3->x) - P1.y) % n;
}

int multiply(
    long long a,
    long long k,
    long long n,
    struct point P,
    struct point* R,
    long long* d)
/* returns - 1 if O encountered, 0 if divisor not found, */
/* 1 otherwise */
{
    int value = 1;
    struct point A, B, C;

    /*  A = P */
    A = P;
    /* B = O = (0, 1) the point at infinity */
    B.x = 0, B.y = 1;
    while (value && k > 0) {
        if (k & 1) {
            *d = gcd((B.x - A.x) % n, n);
            k--;
            value = *d == 1 || *d == n;
            if (A.x == 0 && A.y == 1);
            else if (B.x == 0 && B.y == 1) B = A;
            else if (value) {
                addition_1(n, A, B, &C);
                B = C;
            }
        }
        else {
            *d = gcd((2 * A.y) % n, n);
            k >>= 1;
            value = *d == 1 || *d == n;
            if (value) {
                addition_2(a, n, A, &C);
                A = C;
            }
        }
    }
    *R = B;
    if (R->x < 0) R->x += n;
```

```
209     if (R->y < 0) R->y += n;
210     if (R->x == 0 && R->y == 1) return -1;
211     return !value;
212 }
213
214 long long JACOBI(long long a, long long n)
215 {
216     long long a1, b = a, e = 0, m = 0, n1 = 0, s = 0;
217
218     if (a == 0) return 0;
219     if (a == 1) return 1;
220     while ((b & 1) == 0)
221         b >>= 1, e++;
222     a1 = b;
223     m = n % 8;
224     if (!(e & 1)) s = 1;
225     else if (m == 1 || m == 7) s = +1;
226     else if (m == 3 || m == 5) s = -1;
227     if (n % 4 == 3 && a1 % 4 == 3) s = -s;
228     if (a1 != 1) n1 = n % a1; else n1 = 1;
229     return s * JACOBI(n1, a1);
230 }
231
232 long long exp_mod(long long x, long long b, long long n)
233 /* returns x ^ b mod n */
234 {
235     long long a = 1l, s = x;
236
237     while (b != 0) {
238         if (b & 1l) a = (a * s) % n;
239         b >>= 1;
240         if (b != 0) s = (s * s) % n;
241     }
242     return a;
243 }
244
245 long long rand64()
246 {
247     long long hi = ((long long)rand()) << 31;
248     long long lo = (long long)rand();
249     return hi | lo;
250 }
251
252 long long Rabin_Miller(long long N)
253 /* given an integer N >= 3 returns 0 composite */
254 /* 1 probably prime */
255 {
256     long long N1 = N - 1, a, b, c = 20, e, q = N1, t = 0;
257
258     if (N == 2 || N == 3) return 1;
259     /* N - 1 = 2 ^ t * q with q odd */
260     while (!(q & 1)) q >>= 1, t++;
```

```
L2:
    do a = rand64() % N; while (a == 0);
    e = 0;
    b = exp_mod(a, q, N);
    if (b == 1) goto L4;
    while (b != 1 && b != N1 && e <= t - 2) {
        b = (b * b) % N;
        e++;
    }
    if (b != N1) return 0LL;
L4:
    c--;
    if (c > 0) goto L2;
    return 1LL;
}

long long cardinality(long long a, long long b, long long p)
/* returns the cardinality of the elliptic curve
** E(F_p): y ^ 2 = x ^ 3 + ax + b using
** |E(F_p)| = p + 1 - a_p
** a_p = - sum(0 <= x < p, (y ^ 2 / p))
 */
{
    long long a_p = 0, x;

    for (x = 0; x < p; x++)
        a_p += JACOBI((((x * x) % p * x) % p + a * x + b) % p, p);
    return p + 1 + a_p;
}

long long square_root_mod(long long a, long long p)
/* returns square root of a modulo p if it exists 0 otherwise */
{
    long long b, e = 0, m, n, q = p - 1, r, t, x, y, z;

    /* p - 1 = 2 ^ e * q with q odd */
    while (!(q & 1)) q >>= 1, e++;
    /* find generator */
    do
        do n = rand() % p; while (p == 0);
    while (JACOBI(n, p) != -1);
    z = exp_mod(n, q, p);
    y = z, r = e;
    x = exp_mod(a, (q - 1) / 2, p);
    b = ((a * x % p) * x) % p;
    x = (a * x) % p;
L3:
    if (b == 1) return x;
    m = 1;
    while (exp_mod(b, (long long)pow(
        2.0, (double)m), p) != 1) m++;
    if (m == r) return 0;
```

```
313     t = exp_mod(y, (long long)pow(
314         2.0, (double)r - m - 1), p);
315     y = t * t % p;
316     r = m;
317     x = x * t % p;
318     b = b * y % p;
319     goto L3;
320 }
321
322 long long Goldwasser_Kilian(long long N, long B0)
323 /* returns 0 if N is composite 1 if prime */
324 {
325     double runtime = 0.0;
326     long long value;
327     long long Ni = N, a = 0, b = 0, d = 0, i = 0;
328     long long m = 0, p = 0, q = 0, x = 0, y = 0;
329     struct point P = { 0 }, P1 = { 0 }, P2 = { 0 };
330     clock_t time0 = clock(), time1 = 0;
331
332     Sieve(SIEVE_LIMIT);
333     time1 = clock();
334     runtime = ((double)time1 - time0) / CLOCKS_PER_SEC;
335     printf_s("Prime sieving time = %lf\n", runtime);
336 L2:
337 L3:
338     if (Ni <= B0) {
339         p = 2;
340         while (p <= (long long)sqrt((double)Ni)) {
341             if (Ni % p == 0) {
342                 printf_s("1 factor = %lld\n", p);
343                 return 0;
344             }
345             p++;
346             while (!get_bit((long)p, sieve)) p++;
347         }
348         return 1;
349     }
350     if (!Rabin_Miller(Ni)) goto L9;
351     for (;;) {
352         do a = rand() % Ni; while (a == 0);
353         do b = rand() % Ni; while (b == 0);
354         m = (4 * a * a * a + 27 * b * b) % Ni;
355         if (m != 0) break;
356     }
357     m = cardinality(a, b, Ni);
358     q = m / 2;
359     x = (long long)sqrt(sqrt((double)Ni)) + 1;
360     if (m & 1 || !Rabin_Miller(q) || q <= x * x) goto L3;
361 L6:
362     do {
363         do x = rand() % Ni; while (x == 0);
364         y = (((x * x) % Ni * x) % Ni + a * x + b) % Ni;
```

```
365         if (y < 0) y += p;
366     } while (y == 0 || JACOBI(y, Ni) == -1);
367     y = square_root_mod(y, Ni);
368     if (y == 0) goto L9;
369     P.x = x, P.y = y;
370     value = multiply(a, m, Ni, P, &P1, &d);
371     if (value == 1) {
372         printf_s("2 factor = %lld\n", d);
373         goto L9;
374     }
375     if (value == 0) goto L6;
376     value = multiply(a, 2, Ni, P, &P2, &d);
377     if (value == 1) {
378         printf_s("3 factor = %lld\n", d);
379         goto L9;
380     }
381     if (value == -1) goto L6;
382     printf_s("N[%lld] = %lld\n", i, Ni);
383     printf_s("a = %lld\n", a);
384     printf_s("b = %lld\n", b);
385     printf_s("m = %lld\n", m);
386     printf_s("q = %lld\n", q);
387     printf_s("P = (%lld, %lld)\n", P.x, P.y);
388     printf_s("P1 = (%lld, %lld)\n", P1.x, P1.y);
389     printf_s("P2 = (%lld, %lld)\n", P2.x, P2.y);
390     i++;
391     Ni = q;
392     goto L2;
393 L9:
394     if (i == 0) return 0;
395     i--;
396     goto L3;
397 }
398
399 void test_1(void)
400 {
401   long long a = 1, b = 6, d, p = 11;
402   struct point P = {2LL, 7LL}, R;
403
404   multiply(a, 11, p, P, &R, &d);
405   printf_s("%lld(%lld, %lld) = (%lld, %lld)\n",
406       11LL, P.x, P.y, R.x, R.y);
407   multiply(a, 12, p, P, &R, &d);
408   printf_s("%lld(%lld, %lld) = (%lld, %lld)\n",
409       12LL, P.x, P.y, R.x, R.y);
410   printf_s("|E(F_%lld)| = %lld\n", p, cardinality(a, b, p));
411   printf_s("Rabin_Miller(5) = %lld\n", Rabin_Miller(5LL));
412 }
413
414 void test_2(void)
415 {
416   long long a = 1, b = 2, d, p = 5;
```

```
417    struct point P = {1, 3}, R;
418
419    if (multiply(a, 2, p, P, &R, &d))
420      printf_s("d = %lld\n", d);
421    printf_s("%lld(%lld, %lld) = (%lld, %lld)\n",
422        2LL, P.x, P.y, R.x, R.y);
423    if (multiply(a, 4, p, P, &R, &d))
424      printf_s("d = %lld\n", d);
425    printf_s("%lld(%lld, %lld) = (%lld, %lld)\n",
426        4LL, P.x, P.y, R.x, R.y);
427    printf_s("|E(F_%lld)| = %lld\n",
428        p, cardinality(a, b, p));
429  }
430
431  long long ParseNumber(char numStr[], long long* n) {
432      char c = 0, expStr[32] = { 0 };
433      char addStr[32] = { 0 }, radStr[32] = { 0 };
434      char* karet = NULL, * nsign = NULL, * psign = NULL;
435      long long radix = 0, expon = 0, addme = 0;
436      long long error = 0, i = 0, j = 0, l = 0;
437
438      karet = strchr(numStr, '^');
439      nsign = strchr(numStr, '-');
440      psign = strchr(numStr, '+');
441      l = strlen(numStr);
442
443      if (karet == NULL && nsign == NULL && psign == NULL) {
444          c = numStr[i++];
445
446          while (c >= '0' && c <= '9' && i < l) {
447              radStr[j++] = c;
448              c = numStr[i++];
449          }
450
451          if (i == l)
452          {
453              *n = atoll(numStr);
454              return 1;
455          }
456
457          else
458              return -1;
459
460          if (j == 0)
461              return -2;
462      }
463
464      else if (karet) {
465          c = numStr[0];
466          i = j = 0;
467
468          while ((c >= '0' && c <= '9') && (i < l) || (c == '^')) {
```

```
            c = numStr[i++];

            if (c != '^')
                radStr[j++] = c;
            else if (c == '^')
                break;
        }

        radStr[j] = '\0';
        radix = atol(radStr);

        if (psign && !nsign)
            c = *(psign + 1);
        else if (!psign && nsign)
            c = *(nsign + 1);
        else
            return -3;

        j = 0;

        while ((c >= '0' && c <= '9') && (i < l)) {
            c = numStr[i++];

            if (c >= '0' && c <= '9')
                expStr[j++] = c;
        }

        if (j == 0)
            return -4;

        expStr[j++] = '\0';
        expon = atol(expStr);

        if (i == l) {
            *n = (long long)pow(
                (double)radix, (double)expon) + addme;
            return 1;
        }

        j = 0;
        c = numStr[i];

        while (
            (c >= '0' && c <= '9') && (i < l)) {
            if (c >= '0' && c <= '9')
                addStr[j++] = c;
            c = numStr[i++];
        }

        if (j != 0)
        {
            addStr[j] = '\0';
```

```
521
522             if (psign != NULL)
523                 addme = atol(addStr);
524             else if (nsign != NULL)
525                 addme = -atol(addStr);
526             else
527                 return -5;
528
529             if (error == 0) {
530                 *n = (long long)pow(
531                     (double)radix, (double)expon) + addme;
532                 return 1;
533             }
534         }
535
536         return -6;
537     }
538
539     return 0;
540 }
541
542 int main(void)
543 {
544     double runtime = 0;
545     long long N;
546     clock_t time0 = 0, time1 = 0;
547 #ifdef DEBUG
548     test_1();
549     test_2();
550 #endif
551     for (;;) {
552         char numStr[32] = { 0 };
553         long B0 = 0;
554         printf_s("number to be tested or 0 to quit:\n");
555         scanf_s("%s", numStr, sizeof(numStr));
556         if (!ParseNumber(numStr, &N)) {
557             printf_s("Number parsing error\n");
558             continue;
559         }
560         if (N == 0) break;
561         printf_s("number of primes in factor base:\n");
562         scanf_s("%s", numStr, sizeof(numStr));
563         B0 = atol(numStr);
564         time0 = clock();
565         if (Goldwasser_Kilian(N, B0))
566             printf_s("proven prime\n");
567         else
568             printf_s("composite\n");
569         time1 = clock();
570         runtime = ((double)time1 - time0) / CLOCKS_PER_SEC;
571         printf_s("Runtime in seconds = %lf\n", runtime);
572     }
```

```
573     return 0;
574 }
```