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

long long Brent(long long *N) {
    int i, k = 1, l = 1, c = 0;
    long long P = 1, g, y = 2, x = 2, x1 = 2;
L2:
    x = (x * x + 1) % *N;
    P *= (x1 - x) % *N;
    c++;
    if (c == 20) {
        g = GCD(P, *N);
        if (g == 1)
            goto L4;
        else {
            y = x;
            c = 0;
            goto L2;
        }
    }
    k--;
    if (k != 0)
        goto L2;
    g = GCD(P, *N);
    if (g > 1)
        goto L4;
    x1 = x;
    k = l;
    l *= 2;
    for (i = 0; i < k; i++) {
        x = (x * x + 1) % *N;
    }
    y = x;
    c = 0;
    goto L2;
L4:
    do {
        y = (y * y + 1) % *N;
        g = GCD(x1 - y, *N);
    } while (g <= 1);
    if (g < *N)
        return g;
    return -1;
}

int do_Brent_Method(long long *N, struct factor *f, long *count) {
    long long brent;

    *count = 0;
```

```
53      while (1) {
54          brent = Brent(N);
55          if (brent != -1) {
56              int expon = 0;
57              while (*N % brent == 0) {
58                  expon++;
59                  *N /= brent;
60              }
61              f[*count].expon = expon;
62              f[*count].prime = brent;
63              *count = *count + 1;
64              if (*N == 1)
65                  return 0;
66              if (Miller_Rabin(*N, 20) == 1) {
67                  f[*count].expon = 1;
68                  f[*count].prime = *N;
69                  *count = *count + 1;
70                  return 1;
71              }
72          }
73          else
74              return -1;
75      }
76  }
77
78  long long first_stage(long long *N, int B1,
79      int *failure, long long *x, int number1,
80      long *primes1)
81  {
82      int c = 0, i = -1, j = i;
83      long l = 0, q = 0, q1 = 0;
84      long long g, xp, x1, y;
85
86      *failure = 0;
87      y = *x;
88  Step_2:
89      i++;
90      if (i >= number1) {
91          x1 = *x - 1;
92          g = GCD(x1, *N);
93          if (g == 1) {
94              *failure = 1;
95              return -1;
96          }
97          i = j;
98          *x = y;
99          goto Step_5;
100     }
101     if (i < number1) {
102         q = primes1[i];
103         q1 = q;
104         l = B1 / q;
```

```
105     }
106     goto Step_3;
107 Step_3:
108     while (q1 <= l)
109         q1 *= q;
110     xp = pow_mod_M(*x, q1, *N);
111     *x = xp;
112     if (++c < 20)
113         goto Step_2;
114     goto Step_4;
115 Step_4:
116     x1 = *x - 1;
117     g = GCD(x1, *N);
118     if (g == 1) {
119         c = 0;
120         j = i;
121         y = *x;
122         goto Step_2;
123     }
124     i = j;
125     *x = y;
126     goto Step_5;
127 Step_5:
128     q = primes1[++i];
129     q1 = q;
130     goto Step_6;
131 Step_6:
132     xp = pow_mod_M(*x, q, *N);
133     x1 = *x - 1;
134     *x = xp;
135     g = GCD(x1, *N);
136     if (g == 1) {
137         q1 *= q;
138         if (q1 <= B1)
139             goto Step_6;
140         else
141             goto Step_5;
142     }
143     if (g == *N) {
144         *failure = 1;
145         return -1;
146     }
147     failure = 0;
148     return g;
149 }
150
151 long long second_stage(
152     long long *N, int B1,
153     int B2, int *failure,
154     long numdiff, long *differs,
155     long number1, long *primes1,
156     long number2, long *primes2) {
```

```
157     int c = 0, i = 0, j = 0, nd = 0;
158     long long b = 0, g = 0, x, y = 0, P = 0, PP;
159     long long bd, x1, xb, xp;
160
161     for (int k = 0; k < number2 - 1; k++) {
162         long p1 = primes2[k];
163
164         if (p1 >= B1) {
165             long p2 = primes2[k + 1];
166             differs[nd++] = p2 - p1;
167         }
168     }
169     *failure = 0;
170     x = 2;
171     g = first_stage(
172         N, B1, failure, &x,
173         number1, primes1);
174     if (!(*failure))
175         return g;
176     b = x;
177     c = 0;
178     P = 1;
179     i = -1;
180     j = i;
181     y = x;
182     goto Step_2;
183 Step_2:
184     xp = (long long)pow((double)x, primes1[0]);
185     x = xp;
186     goto Step_3;
187 Step_3:
188     i++;
189     if (i == numdiff)
190         goto Step_6;
191     bd = (long long)pow((double)b, differs[i]);
192     x1 = x * bd;
193     x = x1;
194     x1 = x - 1;
195     PP = P * x1;
196     P = PP;
197     c++;
198     if (c < 20)
199         goto Step_3;
200     goto Step_4;
201 Step_4:
202     g = GCD(P, *N);
203     if (g == 1) {
204         c = 0;
205         j = i;
206         y = x;
207         goto Step_3;
208     }
```

```
209     goto Step_5;
210 Step_5:
211     i = j;
212     x = y;
213     while (i < nd) {
214         if (i < 0)
215             i++;
216         bd = (long long)pow((double)b, differs[i++]);
217         xb = x * bd;
218         x1 = b - 1;
219         x = xb;
220         g = GCD(x1, *N);
221         if (g != 1)
222             break;
223     }
224     if (g == *N) {
225         x = 3;
226         g = first_stage(N, B1, failure, &x,
227             number1, primes1);
228         if (!failure)
229             return g;
230         return -1;
231     }
232     return g;
233 Step_6:
234     g = GCD(P, *N);
235     if (g == 1) {
236         *failure = 1;
237         return -1;
238     }
239     goto Step_5;
240     return -1;
241 }
242
243 int trial_division(long long *N,
244     struct factor *f, long *count,
245     long psize, long *prime) {
246     int found;
247     long B, d, e, i, j = 0, k, m, n;
248     long t[8] = { 6, 4, 2, 4, 2, 4, 6, 2 };
249     long table[8] = { 1, 7, 11, 13, 17, 19, 23, 29 };
250     long long l, r;
251
252     if (*N <= 5) {
253         *count = 1;
254         f[0].expon = 1;
255         switch (*N) {
256         case 1: f[0].prime = 1; break;
257         case 2: f[0].prime = 2; break;
258         case 3: f[0].prime = 3; break;
259         case 4: f[0].expon = 2; f[0].prime = 2; break;
260         case 5: f[0].prime = 5; break;
```

```
261         }
262         return 1;
263     }
264     *count = 0;
265     B = prime[psize - 1];
266     i = -1, l = (long long)sqrt((double)*N), m = -1;
267 L2:
268     m++;
269     if (i == psize) {
270         i = j - 1;
271         goto L5;
272     }
273     d = prime[m];
274     k = d % 30;
275     found = 0;
276     for (n = 0; !found && n < 8; n++) {
277         found = k == table[n];
278         if (found) j = n;
279     }
280 L3:
281     r = *N % d;
282     if (r == 0) {
283         e = 0;
284         do {
285             e++;
286             *N /= d;
287         } while (*N % d == 0);
288         f[*count].expon = e;
289         f[*count].prime = d;
290         *count = *count + 1;
291         if (*N == 1) return 1;
292         goto L2;
293     }
294     if (d >= l) {
295         if (*N > 1) {
296             f[*count].expon = 1;
297             f[*count].prime = *N;
298             *count = *count + 1;
299             return 1;
300         }
301     }
302     else if (i < 0) goto L2;
303 L5:
304     i = (i + 1) % 8;
305     d = d + t[i];
306     if (d > B) return 0;
307     goto L3;
308 }
309
310 int Partition(struct factor *f, int n, int lo, int hi) {
311     int pivotIndex = lo + (hi - lo) / 2;
312     struct factor x = f[pivotIndex];
```

```
313     struct factor t = x;
314
315     f[pivotIndex] = f[hi];
316     f[hi] = t;
317
318     int storeIndex = lo;
319
320     for (int i = lo; i < hi; i++)
321     {
322         if (f[i].prime < x.prime)
323         {
324             t = f[i];
325             f[i] = f[storeIndex];
326             f[storeIndex++] = t;
327         }
328     }
329
330     t = f[storeIndex];
331     f[storeIndex] = f[hi];
332     f[hi] = t;
333     return storeIndex;
334 }
335
336 void DoQuickSort(struct factor *f, int n, int p, int r) {
337     if (p < r)
338     {
339         int q = Partition(f, n, p, r);
340
341         DoQuickSort(f, n, p, q - 1);
342         DoQuickSort(f, n, q + 1, r);
343     }
344 }
345
346 void QuickSort(struct factor *f, int n) {
347     DoQuickSort(f, n, 0, n - 1);
348 }
```