```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;
```

```
    while (1) {
        brent = Brent(N);
        if (brent != -1) {
            int expon = 0;
            while (*N % brent == 0) {
                expon++;
                *N /= brent;
            }
            f[*count].expon = expon;
            f[*count].prime = brent;
            *count = *count + 1;
            if (*N == 1)
                return 0;
            if (Miller_Rabin(*N, 20) == 1) {
                f[*count].expon = 1;
                f[*count].prime = *N;
                *count = *count + 1;
                return 1;
            }
        }
        else
            return -1;
    }
}

long long first_stage(long long *N, int B1,
    int *failure, long long *x, int number1,
    long *primes1)
{
    int c = 0, i = -1, j = i;
    long l = 0, q = 0, q1 = 0;
    long long g, xp, x1, y;

    *failure = 0;
    y = *x;
Step_2:
    i++;
    if (i >= number1) {
        x1 = *x - 1;
        g = GCD(x1, *N);
        if (g == 1) {
            *failure = 1;
            return -1;
        }
        i = j;
        *x = y;
        goto Step_5;
    }
    if (i < number1) {
        q = primes1[i];
        q1 = q;
        l = B1 / q;
```

```c
    }
    goto Step_3;
Step_3:
    while (q1 <= l)
        q1 *= q;
    xp = pow_mod_M(*x, q1, *N);
    *x = xp;
    if (++c < 20)
        goto Step_2;
    goto Step_4;
Step_4:
    x1 = *x - 1;
    g = GCD(x1, *N);
    if (g == 1) {
        c = 0;
        j = i;
        y = *x;
        goto Step_2;
    }
    i = j;
    *x = y;
    goto Step_5;
Step_5:
    q = primes1[++i];
    q1 = q;
    goto Step_6;
Step_6:
    xp = pow_mod_M(*x, q, *N);
    x1 = *x - 1;
    *x = xp;
    g = GCD(x1, *N);
    if (g == 1) {
        q1 *= q;
        if (q1 <= B1)
            goto Step_6;
        else
            goto Step_5;
    }
    if (g == *N) {
        *failure = 1;
        return -1;
    }
    failure = 0;
    return g;
}

long long second_stage(
    long long *N, int B1,
    int B2, int *failure,
    long numdiff, long *differs,
    long number1, long *primes1,
    long number2, long *primes2) {
```

```
    int c = 0, i = 0, j = 0, nd = 0;
    long long b = 0, g = 0, x, y = 0, P = 0, PP;
    long long bd, x1, xb, xp;

    for (int k = 0; k < number2 - 1; k++) {
        long p1 = primes2[k];

        if (p1 >= B1) {
            long p2 = primes2[k + 1];
            differs[nd++] = p2 - p1;
        }
    }
    *failure = 0;
    x = 2;
    g = first_stage(
        N, B1, failure, &x,
        number1, primes1);
    if (!(*failure))
        return g;
    b = x;
    c = 0;
    P = 1;
    i = -1;
    j = i;
    y = x;
    goto Step_2;
Step_2:
    xp = (long long)pow((double)x, primes1[0]);
    x = xp;
    goto Step_3;
Step_3:
    i++;
    if (i == numdiff)
        goto Step_6;
    bd = (long long)pow((double)b, differs[i]);
    x1 = x * bd;
    x = x1;
    x1 = x - 1;
    PP = P * x1;
    P = PP;
    c++;
    if (c < 20)
        goto Step_3;
    goto Step_4;
Step_4:
    g = GCD(P, *N);
    if (g == 1) {
        c = 0;
        j = i;
        y = x;
        goto Step_3;
    }
```

```
    goto Step_5;
Step_5:
    i = j;
    x = y;
    while (i < nd) {
        if (i < 0)
            i++;
        bd = (long long)pow((double)b, differs[i++]);
        xb = x * bd;
        x1 = b - 1;
        x = xb;
        g = GCD(x1, *N);
        if (g != 1)
            break;
    }
    if (g == *N) {
        x = 3;
        g = first_stage(N, B1, failure, &x,
            number1, primes1);
        if (!failure)
            return g;
        return -1;
    }
    return g;
Step_6:
    g = GCD(P, *N);
    if (g == 1) {
        *failure = 1;
        return -1;
    }
    goto Step_5;
    return -1;
}

int trial_division(long long *N,
    struct factor *f, long *count,
    long psize, long *prime) {
    int found;
    long B, d, e, i, j = 0, k, m, n;
    long t[8] = { 6, 4, 2, 4, 2, 4, 6, 2 };
    long table[8] = { 1, 7, 11, 13, 17, 19, 23, 29 };
    long long l, r;

    if (*N <= 5) {
        *count = 1;
        f[0].expon = 1;
        switch (*N) {
        case 1: f[0].prime = 1; break;
        case 2: f[0].prime = 2; break;
        case 3: f[0].prime = 3; break;
        case 4: f[0].expon = 2; f[0].prime = 2; break;
        case 5: f[0].prime = 5; break;
```

```
        }
        return 1;
    }
    *count = 0;
    B = prime[psize - 1];
    i = -1, l = (long long)sqrt((double)*N), m = -1;
L2:
    m++;
    if (i == psize) {
        i = j - 1;
        goto L5;
    }
    d = prime[m];
    k = d % 30;
    found = 0;
    for (n = 0; !found && n < 8; n++) {
        found = k == table[n];
        if (found) j = n;
    }
L3:
    r = *N % d;
    if (r == 0) {
        e = 0;
        do {
            e++;
            *N /= d;
        } while (*N % d == 0);
        f[*count].expon = e;
        f[*count].prime = d;
        *count = *count + 1;
        if (*N == 1) return 1;
        goto L2;
    }
    if (d >= l) {
        if (*N > 1) {
            f[*count].expon = 1;
            f[*count].prime = *N;
            *count = *count + 1;
            return 1;
        }
    }
    else if (i < 0) goto L2;
L5:
    i = (i + 1) % 8;
    d = d + t[i];
    if (d > B) return 0;
    goto L3;
}

int Partition(struct factor *f, int n, int lo, int hi) {
    int pivotIndex = lo + (hi - lo) / 2;
    struct factor x = f[pivotIndex];
```

```c
    struct factor t = x;

    f[pivotIndex] = f[hi];
    f[hi] = t;

    int storeIndex = lo;

    for (int i = lo; i < hi; i++)
    {
        if (f[i].prime < x.prime)
        {
            t = f[i];
            f[i] = f[storeIndex];
            f[storeIndex++] = t;
        }
    }

    t = f[storeIndex];
    f[storeIndex] = f[hi];
    f[hi] = t;
    return storeIndex;
}

void DoQuickSort(struct factor *f, int n, int p, int r) {
    if (p < r)
    {
        int q = Partition(f, n, p, r);

        DoQuickSort(f, n, p, q - 1);
        DoQuickSort(f, n, q + 1, r);
    }
}

void QuickSort(struct factor *f, int n) {
    DoQuickSort(f, n, 0, n - 1);
}
```