```c
/*
Author:  Pate Williams (c) 1997 - 2022
*/

#include <math.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "Factoring .h"
#include "Number_Theory.h"

long prime1[PRIME_SIZE_1], sieve1[SIEVE_SIZE_1];
long prime2[PRIME_SIZE_2], sieve2[SIEVE_SIZE_2];
long differ[PRIME_SIZE_2 - PRIME_SIZE_1 + 2];

int main(void) {
    char nStr[128], option[128];
    int d = 0, bm, mr, numdiff = PRIME_SIZE_2 - PRIME_SIZE_1 + 1;
    double sieve_time;
    long count, i, kount, p = 2;
    long long largest_prime, N, sqrtN;
    struct factor f[32], g[32];
    clock_t begin, end;

    begin = clock();
    Greek_sieve(
        MAX_SIEVE_1,
        PRIME_SIZE_1,
        sieve1,
        prime1);
    Greek_sieve(
        MAX_SIEVE_2,
        PRIME_SIZE_2,
        sieve2,
        prime2);
    end = clock();
    sieve_time = (double)(end - begin) / CLOCKS_PER_SEC;
    printf("Time to create prime number ");
    printf("sieves in seconds %lf\n", sieve_time);
    while (1) {
        printf("Menu\n");
        printf("1 Brent's Algorithm\n");
        printf("2 p - 1 Two Stages\n");
        printf("3 Trial Division\n");
        printf("4 Test get_bits\n");
        printf("5 Test pow_mod_M and pow_mod_S\n");
        printf("6 Exit\n");
        printf("Option = ");
        scanf_s("%s", option, 128);
        if (option[0] < '1' || option[0] > '6') {
            printf("Illegal option, try again\n");
            continue;
```

```
53          }
54          if (option[0] == '6')
55              break;
56          printf("N = ");
57          scanf_s("%s", nStr, 128);
58          N = atoll(nStr);
59          if (N <= 3) {
60              printf("Number is too small\n");
61              continue;
62          }
63          if (N <= 1234567890) {
64              mr = Miller_Rabin(N, 20);
65              if (N <= 0) break;
66              if (mr == 1) {
67                  printf("Number is probably prime\n");
68                  continue;
69              }
70          }
71          if (N == 1) {
72              break;
73          }
74          begin = clock();
75          if (option[0] == '1') {
76              bm = do_Brent_Method(&N, f, &count);
77              if (bm == -1) {
78                  printf("Brent's method failed\n");
79                  continue;
80              }
81              QuickSort(f, count);
82              printf("Factors:\n");
83              for (i = 0; i < count; i++) {
84                  printf("%lld", f[i].prime);
85                  if (f[i].expon > 1)
86                      printf(" ^ %ld\n", f[i].expon);
87                  else
88                      printf("\n");
89              }
90              mr = Miller_Rabin(f[count - 1].prime, 20);
91              if (f[count - 1].prime <= 1234567890 && mr == 1)
92                  printf("Last factor is probably prime\n");
93              else if (f[count - 1].prime <= 1234567890 && mr == 0)
94                  printf("Last factor is composite\n");
95          }
96          else if (option[0] == '2') {
97              int failure = 0;
98              count = 0;
99              while (failure == 0) {
100                 long long d = second_stage(
101                     &N, MAX_SIEVE_1, MAX_SIEVE_2,
102                     &failure,
103                     numdiff, differ,
104                     PRIME_SIZE_1, prime1,
```

```
105                     PRIME_SIZE_2, prime2);
106                 if (d > 1) {
107                     int expon = 0;
108                     while (N % d == 0) {
109                         expon++;
110                         N /= d;
111                     }
112                     f[count].expon = expon;
113                     f[count].prime = d;
114                     count++;
115                 }
116                 else if (d == -1) {
117                     printf("p - 1 Method failed\n");
118                     break;
119                 }
120             }
121             QuickSort(f, count);
122             for (i = 0; i < count; i++) {
123                 printf("%lld", f[i].prime);
124                 if (f[i].expon > 1)
125                     printf(" ^ %ld\n", f[i].expon);
126                 else
127                     printf("\n");
128             }
129             mr = Miller_Rabin(f[count - 1].prime, 20);
130             if (f[count - 1].prime <= 1234567890 && mr == 1)
131                 printf("Last factor is probably prime\n");
132             else if (f[count - 1].prime <= 1234567890 && mr == 0)
133                 printf("Last factor is composite\n");
134         }
135         else if (option[0] == '3') {
136             largest_prime = prime1[PRIME_SIZE_1 - 1];
137             printf("Largest prime = %lld\n", largest_prime);
138             sqrtN = (long long)sqrt((double)N);
139             if (sqrtN > largest_prime) {
140                 printf("Number is too large!\n");
141                 printf("Square root %lld must be < %lld\n", sqrtN,
                      largest_prime);
142                 continue;
143             }
144             trial_division(&N, g, &kount,
145                 PRIME_SIZE_1, prime1);
146             printf("Factors:\n");
147             for (i = 0; i < kount; i++) {
148                 printf("%lld", g[i].prime);
149                 if (g[i].expon > 1)
150                     printf(" ^ %ld\n", g[i].expon);
151                 else
152                     printf("\n");
153             }
154             QuickSort(g, kount);
155             mr = Miller_Rabin(g[kount - 1].prime, 20);
```

```
156             if (g[kount - 1].prime <= 1234567890 && mr == 1)
157                 printf("Last factor is probably prime\n");
158             else if (g[kount - 1].prime <= 1234567890 && mr == 0)
159                 printf("Last factor is composite\n");
160         }
161         if (option[0] >= '1' && option[0] <= '3') {
162             end = clock();
163             sieve_time = (double)(end - begin) / CLOCKS_PER_SEC;
164             printf("Time to do factoring in seconds = %lf\n", sieve_time);
165         }
166         if (option[0] == '4') {
167             int l, length, bits[64];
168             get_bits(N, &length, bits);
169             for (l = 0; l < length; l++)
170                 printf("%d", bits[l]);
171             printf("\n");
172         }
173         else if (option[0] == '5') {
174             long long radix, power;
175             printf("Enter radix = ");
176             scanf_s("%s", nStr, 128);
177             radix = atoll(nStr);
178             printf("Enter power = ");
179             scanf_s("%s", nStr, 128);
180             power = atoll(nStr);
181             printf("pow_mod_M = %lld\n", pow_mod_M(
182                 radix, power, N));
183             printf("pow_mod_S = %lld\n", pow_mod_S(
184                 radix, power, N));
185         }
186     }
187     return 0;
188 }
```