lab09

```c
// EE2310 Lab09. GCD and LCM
// 108061213, 劉奕緯
// Nov. 19, 2019

#include<stdio.h>                       // standard I/O
#include <stdio.h>                       // standard I/O
#include<math.h>                        // to use power function
#include <math.h>                        // to use power function
#define S 20                            // size of arrays

void factorize(int N, int factors[S], int power[S]);
                                        // factorizes N into prime factors
                                        // storing in (factors array) and
                                        // associated powers (power array).
void GCD(int Afactors[S], int Apower[S], int Bfactors[S], int Bpower[S],
         int Cfactors[S], int Cpower[S]);
                                        // takes two factors arrays & two power
                                        // arrays to produce two output arrays:
                                        // one for GCD factors
                                        // and the other for GCD power.
void LCM(int Afactors[S], int Apower[S], int Bfactors[S], int Bpower[S],
         int Cfactors[S], int Cpower[S]);
                                        // takes two factors arrays & two power
                                        // arrays to produce two output arrays:
                                        // one for LCM factors
                                        // and the other for LCM power.
    int ia, ib, ic;                     // index of a's, b's, c's arrays
int ia, ib, ic;                      // index of a's, b's, c's arrays
No global variables for this lab.
void write(int factors[S], int power[S]);
                                        //  prints out the factors and power
                                        // arrays in product of prime
                                        // and the integer calculated
                                        // using this product from.

int main(void)
{
    int a, afactor[S] = {0}, apower[S] = {0};
                                        // a & a's factors & power of factors
    int b, bfactor[S] = {0}, bpower[S] = {0};
                                        // b & b's factors & power of factors
    int cfactor[S] = {0}, cpower[S] = {0};
                                        // factors & its power of GCD or LCM
                                        // since LCM must longer then GCD, i
                                        // use an arry to store it

    printf("input A: ");                // prompt for a
    scanf("%d", &a);                    // input a
```

```
45      printf("input B: ");                // prompt for b
46      scanf("%d", &b);                    // input b
47      factorize(a, afactor, apower);      // factorizing a
48      printf("  A = ");                   // and output
49      write(afactor, apower);
50      factorize(b, bfactor, bpower);      // factorizing b
51      printf("  B = ");                   // and output
52      write(bfactor, bpower);
53      GCD(afactor, apower, bfactor,bpower, cfactor, cpower);
        GCD(afactor, apower, bfactor, bpower, cfactor, cpower);
54      printf("  GCD(A,B) = ");            // get GCD of a and b
55      write(cfactor, cpower);             // and ouput
56      LCM(afactor, apower, bfactor,bpower, cfactor, cpower);
        LCM(afactor, apower, bfactor, bpower, cfactor, cpower);
57      printf("  LCM(A,B) = ");            // get LCM of a and b
58      write(cfactor, cpower);             // and output
59      return 0;
60  }
61
62
63  void factorize(int N, int factors[S], int power[S])
64  {                                       // factorizes N into prime factors
65                                          // storing in (factors array) and
66                                          // associated powers (power array).
67      int i, j;                           // index
68
69      i = j = 0;
70      if(N % 2 == 0) {                    // get all the two
        if (N % 2 == 0) {                    // get all the two
71          factors[j] = 2;
72          do {
73              power[j]++;
74              N /= 2;
75          } while (N % 2 == 0);
76          j++;                            // next factor
77      }
78      for (i = 3; i <= N; i += 2){        // since N now must be odd
        for (i = 3; i <= N; i += 2) {        // since N now must be odd
79                                          // its factors must be odds
80          if(N % i == 0) {                // get all the i
            if (N % i == 0) {                // get all the i
81              factors[j] = i;
82              do {
83                  power[j]++;
84                  N /= i;
85              } while (N % i == 0);
86              j++;                        // next factor
87          }
88      }
89      factors[j] = power[j] = 1;          // the final element to be (1,1)
```

```
}
Need a blank line here.
void GCD(int Afactors[S], int Apower[S], int Bfactors[S], int Bpower[S], int Cfa
ctors[S], int Cpower[S])
This line has more than 80 characters
{                                       // takes two factors arrays & two power
                                        // arrays to produce two output arrays:
                                        // one for GCD factors
                                        // and the other for GCD power.
    int ia, ib, ic;                     // index of a's, b's, c's arrays

    ia = ib = ic = 0;
    for ( ; Afactors[ia] != 1; ia++) {
                                        // we use a loop manange to
                                        // gain common factor
                                        // by checking all Afactors
        for ( ; Bfactors[ib] < Afactors[ia]; ib++) ;
                                        // while this loop end
                                        // Afactors[ia] is equal or smaller
                                        // than Bfactors[ib]
        if (Afactors[ia] == Bfactors[ib]) {
                                        // if equal we put factor into c's array
s
This line has more than 80 characters
            Cfactors[ic] = Afactors[ia];
            if(Apower[ia] > Bpower[ib]) // and put the smaller power
            if (Apower[ia] > Bpower[ib]) // and put the smaller power
                Cpower[ic] = Bpower[ib];
            else
                Cpower[ic] = Apower[ia];
            ic++;
        }
        else ib--;                      // else let ib back, since we don't
                                        // know next afactor is eqal to
                                        // previos bfactor
    }
    Cpower[ic] = Cfactors[ic] = 1;      // final element be (1,1)
}
Need a blank line here.
void LCM(int Afactors[S], int Apower[S], int Bfactors[S], int Bpower[S], int Cfa
ctors[S], int Cpower[S])
This line has more than 80 characters
{                                       // takes two factors arrays & two power
                                        // arrays to produce two output arrays:
                                        // one for LCM factors
                                        // and the other for LCM power.
    int ia, ib, ic;                     // index of a's, b's, c's arrays

    ia = ib = ic = 0;
    for ( ; Afactors[ia] != 1; ia++) {
```

```
131         for ( ; Bfactors[ib] < Afactors[ia] && Bfactors[ib] != 1; ib++) {
132                                         // get all factors is smaller
133                                         // than Afactors[ia]
134             Cfactors[ic] = Bfactors[ib];
135             Cpower[ic] = Bpower[ib];
136             ic++;
137         }
138                                         // than get Afactors[ia]
139         if (Afactors[ia] == Bfactors[ib]) {
140                                         // if a and b have same factor
141                                         // get the bigger power
142             Cfactors[ic] = Afactors[ia];
143             if(Apower[ia] < Bpower[ib])
                if (Apower[ia] < Bpower[ib])
144                 Cpower[ic] = Bpower[ib];
145             else
146                 Cpower[ic] = Apower[ia];
147             ic++;
148             ib++;
149         }
150         else {                          // easily we cant get Afactors[ia]
151                                         // and Apower[ia]
152                                         // when only "a" have
153             Cfactors[ic] = Afactors[ia];
154             Cpower[ic] = Apower[ia];
155             ic++;
156         }
157     }                                   // end when we get all a's factors
158     while ( Bfactors[ib] != 1) {        // get all b's factors
        while (Bfactors[ib] != 1) {        // get all b's factors
159         Cfactors[ic] = Bfactors[ib];
160         Cpower[ic] = Bpower[ib];
161         ib++;
162         ic++;
163     }
164     Cpower[ic] = Cfactors[ic] = 1;
165 }
166 void write(int factors[S], int power[S])
167 {                                       //  prints out the factors and power
168                                         // arrays in product of prime
169                                         // and the integer calculated
170                                         // using this product from.
171     int i = 0;                          // index
172     int product = 1;                    // total
173
174     printf("%d", factors[0]);           // output the first factor
175     if (power[0] != 1)                  // we couldn't output ^1
176         printf("^%d", power[0]);
177     product *= pow(factors[0], power[0]);
178     if(factors[0] != 1) {               // if factors[0] == 1 we know it ends
```

```
        if (factors[0] != 1) {              // if factors[0] == 1 we know it ends
179                                         // output else factors
180         for (i = 1; factors[i] != 1; i++) {
181             printf(" * %d", factors[i]);
182             if (power[i] != 1)          // we wouldn't like outputing ^1
183                 printf("^%d", power[i]);
184             product *= pow(factors[i], power[i]);
185         }
186     }
187     printf(" = %d\n", product);
188 }
```

[Format] can be improved.
[Coding] lab09.c spelling errors: arry(1), eqal(1), manange(1), ouput(1), previos(1)
[GCD] can be more efficient.
[LCM] can be more efficient.
[Global] variables are not needed for this lab.

Score: 65