lab09

```
$ gcc lab09.c

$ ./a.out
input A: 100
input B: 225
  A = 2^2 * 5^2 = 100
  B = 3^2 * 5^2 = 225
  GCD(A, B) = 5^2 = 25
  LCM(A, B) = 2^2 * 3^2 * 5^2 = 900
$ ./a.out
input A: 91
input B: 121
  A = 7 * 13 = 91
  B = 11^2 = 121
  GCD(A, B) = 1 = 1
  LCM(A, B) = 7 * 11^2 * 13 = 11011
$ ./a.out
input A: 19
input B: 37
  A = 19 = 19
  B = 37 = 37
  GCD(A, B) = 1 = 1
  LCM(A, B) = 19 * 37 = 703
$ ./a.out
input A: 360
input B: 24
  A = 2^3 * 3^2 * 5 = 360
  B = 2^3 * 3 = 24
  GCD(A, B) = 2^3 * 3 = 24
  LCM(A, B) = 2^3 * 3^2 * 5 = 360
```

```
score: 90.0
o. [Output] Program output is correct, good.
o. [Coding] lab09.c spelling errors: devide(1), facctor(1)
o. [Format] Program format can be improved.
o. [GCD] function can be more efficient.
```

lab09.c

```c
// EE231002 Lab09. GCD and LCM
// 111060023, 黃柏霖
// Date: 2022/11/21

#include <stdio.h>

#define S 20

void factorize(int N, int factors[S], int powers[S]);   // to factorize
void GCD(int Afactors[S], int Apower[S], int Bfactors[S], int Bpowers[S],
         int Cfactors[S], int Cpowers[S]);              // to find GCD
void LCM(int Afactors[S], int Apower[S], int Bfactors[S], int Bpowers[S],
         int Cfactors[S], int Cpowers[S]);              // to find LCM
void write(int factors[S], int powers[S]);              // to print answer
void clean(int array[S]);                               // clean array

int main(void)
{
    int N1, N2;                                 // two numbers
    int fac1[S] = {0}, fac2[S] = {0},
        facans[S] = {0};                        // factors for N1, N2, answer
    int pow1[S] = {0}, pow2[S] = {0},
        powans[S] = {0};                        // powers for N1, N2, answer

    printf("input A: ");                            // prompt for A
    scanf("%d", &N1);                               // get A
    printf("input B: ");                            // prompt for B
    scanf("%d", &N2);                               // get B
    factorize(N1, fac1, pow1);                      // factorize A
    factorize(N2, fac2, pow2);                      // factorize B
    printf("  A = ");
    write(fac1, pow1);                              // print factorized A
    printf("  B = ");
    write(fac2, pow2);                              // print factorized B
    GCD(fac1, pow1, fac2, pow2, facans, powans);    // get GCD of A and B
    printf("  GCD(A, B) = ");
    write(facans, powans);                          // print factorized (A, B)
    clean(facans);                                  // clean factors for answer
    clean(powans);                                  // clean powers for answer
    LCM(fac1, pow1, fac2, pow2, facans, powans);    // get LCM of A and B
```

```
41      printf("  LCM(A, B) = ");
42      write(facans, powans);                          // print factorized [A, B]
43      return 0;
44  }
45
46
47  // To factorize an integer
48  // input: int N is the integer to be factorized
49  //        int factors[S] stores the factors
50  //        int power[S] stores the power of each factor
51  //        return: no return
52  void factorize(int N, int factors[S], int powers[S])
53  {
54      int fac = 2;                                    // factors
55      int k = 0;                                      // index for arrays
56
57      while (N > 1) {                                 // searching when N > 1
58          while (N % fac == 0) {                      // when fac can devide N
59              factors[k] = fac;                       // store fac
60              powers[k]++;                            // add one to the power
61              N /= fac;                               // eliminate fac from N
62          }
63          fac++;                                      // find next factor
64          if (powers[k] != 0) k++;                    // find the next facctor
65      }
66      factors[k] = 1;                                 // the final factor is 1
67      powers[k] = 1;                                  // the power for 1 is 1
68  }
69
70  // To compute GCD of given two numbers
71  // input: Afactors[s] and Apowers[S] are factors and powers for integer A
72  //        Bfactors[s] and Bpowers[S] are factors and powers for integer B
73  //        Cfactors[s] and Cpowers[S] are factors and powers for GCD(A, B)
74  // return: no return
75  void GCD(int Afactors[S], int Apowers[S], int Bfactors[S], int Bpowers[S],
76           int Cfactors[S], int Cpowers[S])
77  {
78      int i, j;                                       // loop control
79      int k = 0;                                      // index for array
80
81      for(i = 0; Afactors[i] != 1; i++) {             // searching until 1
```

```
    for (i = 0; Afactors[i] != 1; i++) {            // searching until 1
        for (j = 0; Bfactors[j] != 1; j++) {        // searching until 1
            if (Afactors[i] == Bfactors[j]) {       // have the same factor
                Cfactors[k] = Bfactors[j];          // store the factor
                Cpowers[k] = Apowers[i] < Bpowers[j] ?
                    Apowers[i] : Bpowers[j];        // find the bigger power
                k++;                                // store the next
            }
        }
    }
    Cfactors[k] = 1;                                // the final factor is 1
    Cpowers[k] = 1;                                 // the power for 1 is 1
}

// To compute LCM of given two numbers
// input: Afactors[s] and Apowers[S] are factors and powers for integer A
//        Bfactors[s] and Bpowers[S] are factors and powers for integer B
//        Cfactors[s] and Cpowers[S] are factors and powers for LCM(A, B)
// return: no return
void LCM(int Afactors[S], int Apowers[S], int Bfactors[S], int Bpowers[S],
         int Cfactors[S], int Cpowers[])
{
    int i = 0;                                      // index for A
    int j = 0;                                      // index for B
    int k = 0;                                      // index for C

    while (Afactors[i] > 1 || Bfactors[j] > 1) {    // search all factors > 1
        // Store A to C if 1. A's factor is not 1 but smaller than B's factor
        //                 2. all factors of B is found
        if ((Afactors[i] < Bfactors[j] && Afactors[i] > 1)
            || (Afactors[i] > 1 && Bfactors[j] == 1)) {
            Cfactors[k] = Afactors[i];
            Cpowers[k] = Apowers[i];
            i++;                                    // go to next A
            k++;                                    // store the next
        }
        // Store B to C if 1. B's factor is not 1 but smaller than A's factor
        //                 2. all factors of A is found
        else if ((Afactors[i] > Bfactors[j] && Bfactors[j] > 1)
                || (Bfactors[j] > 1 && Afactors[i] == 1)){
                || (Bfactors[j] > 1 && Afactors[i] == 1)) {
```

```
121             Cfactors[k] = Bfactors[j];
122             Cpowers[k] = Bpowers[j];
123             j++;                                    // go to next B
124             k++;                                    // store the next
125         }
126         // if A's factor is as big as B's factor, store A factor
127         // and store the one which has bigger power
128         else {
129             Cfactors[k] = Afactors[i];
130             Cpowers[k] = Apowers[i] > Bpowers[j] ?
131                 Apowers[i] : Bpowers[j];
132             i++;                                    // go to next A
133             j++;                                    // go to next B
134             k++;                                    // store the next
135         }
136     }
137     Cfactors[k] = 1;                                // the final factor is 1
138     Cpowers[k] = 1;                                 // the power for 1 is 1
139 }
140
141 // To print factors and power of an integer
142 // input: int factors[S] are the factors of the integer
143 //        int power[S] are the power of each factors
144 // return: no return
145 // output: factors and powers of an integer and the integer itself
146 void write(int factors[S], int powers[S])
147 {
148     int i, j;                                       // loop control
149     int parts = 1;                                  // store factor^power
150     int product = 1;                                // multiple of all parts
151
152     // print 1 directly if it's one
153     if (factors[0] == 1) {
154         printf("1 = 1\n");
155         return;
156     }
157     // print factors and power in a form of factor^power
158     for (i = 0; factors[i] != 1; i++) {
159         if (i == 0) printf("%d", factors[i]);
160         else printf(" * %d", factors[i]);
161         if (powers[i] > 1) printf("^%d", powers[i]);    // print power if > 1
```

```
162         for (j = 0; j < powers[i]; j++) {               // compute parts
163             parts *= factors[i];
164         }
165         product *= parts;                               // compute product
166         parts = 1;                                      // initialize it
167     }
168     printf(" = %d\n", product);
169 }
170
171 // Clean all elements in given array to 0
172 // input: int factors[S] is the array should be cleaned
173 // return: no return
174 void clean(int array[S])
175 {
176     int i;
177
178     for(i = 0; i < S; i++) array[i] = 0;                // turn elements to 0
        for (i = 0; i < S; i++) array[i] = 0;                // turn elements to 0
179 }
```