lab12

```
$ gcc lab12.c

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

```
score: 89.0
o. [Output] Program output is correct, good.
o. [Coding] lab12.c spelling errors: devided(1), noed(1), priem(1), toching(1)
o. [Format] Program format can be improved.
o. [Codes] have memory leakage problem.
```

lab12.c

```c
// EE231002 Lab12. Linked Lists
// 111060023, 黃柏霖
// Date: 2022/12/16

#include <stdio.h>                          // i/o header
#include <stdlib.h>                         // memory control header

typedef struct factor {                     // node for a prime factor
    int prime;                              // prime factor
    int power;                              // associated power
    struct factor *next;                    // pointer for the next prime factor
} FACTOR;

FACTOR *factorize(int N);                   // to factorize N
FACTOR *GCD(FACTOR *A, FACTOR *B);          // to find GCD of two factorized int
FACTOR *LCM(FACTOR *A, FACTOR *B);          // to find LCM of two factorized int
void write(FACTOR *A);                      // to write a factorized int

int main(int argc, char *argv[])            // get string while executing
{
    int A = atoi(argv[1]);                  // get the first int A
    int B = atoi(argv[2]);                  // get the second int B
    FACTOR *Afactor, *Bfactor;              // linked list for factorized A and B

    Afactor = factorize(A);                 // factorize A
    Bfactor = factorize(B);                 // factorize B
    printf("A = ");
    write(Afactor);                         // print A's factors
    printf("B = ");
    write(Bfactor);                         // print B's factors
    printf("GCD = ");
    write(GCD(Afactor, Bfactor));           // print factors of GCD of A and B
    printf("LCM = ");
    write(LCM(Afactor, Bfactor));           // print factors of LCM of A and B
    return 0;                               // end of main
}

// To factorize the input N into its prime factors and their associated powers
// input: int N: the int to be factorized
// return: FACTOR head: the head of the linked list
```

```
FACTOR *factorize(int N)
{
    int fac = 2;                        // factor
    FACTOR* head = NULL;                // the head of link list
    FACTOR* curr = NULL;                // the current node
    FACTOR* new = NULL;                 // new node

    while (N > 1) {                     // while N can be factorized
        if (N % fac == 0) {             // if a fac is found
            new = (FACTOR *) malloc(sizeof(FACTOR));    // get a new node
            new->prime = fac;           // store fac to the new noed
            new->power = 0;
            new->next = NULL;           // no node after the new node yet
        }
        while (N % fac == 0) {          // while N still can be devided by fac
            new->power++;               // power
            N /= fac;                   // remove fac from N
        }
        if (head == NULL) {             // if no head yet
            head = new;                 // set head to new node
            curr = new;                 // set current node to new node
        }
        else {
            curr->next = new;           // the next node is the new node
            curr = curr->next;          // go to next node
        }
        fac++;                          // find next fac
    }
    return head;                        // return the head of the linked list
}

// To find the Greatest Common Divisor of two given linked lists
// input: FACTOR *A, FACTOR *B: the given linked lists
// return: FACTOR Ghead: the head of factorized linked lists of GCD
FACTOR *GCD(FACTOR *A, FACTOR *B)
{
    FACTOR *Acurr = A;                  // the current node for A
    FACTOR *Bcurr = B;                  // the current node for B
    FACTOR *Ghead = NULL;               // the head for GCD
    FACTOR *Gnew = NULL;                // the new node for GCD
    FACTOR *Gcurr = NULL;               // the current node for GCD

    while (Acurr != NULL && Bcurr != NULL) {    // stop when one touch the end
        if (Acurr->prime == Bcurr -> prime) {   // if primes are the same
        if (Acurr->prime == Bcurr->prime) {   // if primes are the same
            Gnew = (FACTOR *) malloc(sizeof(FACTOR));   // get a GCD's new node
            Gnew->prime = Acurr->prime;         // store prime to GCD
            Gnew->power = Acurr->power < Bcurr->power ?
                Acurr->power : Bcurr->power;    // store smaller power
            Acurr = Acurr->next;                // find next A node
            Bcurr = Bcurr->next;                // find next B node
            if (Ghead == NULL) {                // if no GCD has head yet
                Ghead = Gnew;                   // let head be new node
                Gcurr = Gnew;                   // let current be new node
            } else {
                Gcurr->next = Gnew;             // the next node is the new node
                Gcurr = Gcurr->next;            // go to next node
            }
        } else if(Acurr->prime > Bcurr->prime) {    // when A prime > B prime
        } else if (Acurr->prime > Bcurr->prime) {    // when A prime > B prime
            Bcurr = Bcurr->next;                // find next prime of B
        } else Acurr = Acurr->next;             // find next prime of A
    }
    return Ghead;                               // return head of GCD
}

// To find the Least Common Multiple of two given linked lists
// input: FACTOR *A, FACTOR *B: the given linked lists
// return: FACTOR Lhead: the head of factorized linked lists of LCM
FACTOR *LCM(FACTOR *A, FACTOR *B)
{
    FACTOR *Acurr = A;                  // the current node for A
    FACTOR *Bcurr = B;                  // the current node for B
    FACTOR *Lhead = NULL;               // the head for LCM
    FACTOR *Lnew = NULL;                // the new node for LCM
    FACTOR *Lcurr = NULL;               // the current node for LCM

    while (Acurr != NULL || Bcurr != NULL) {    // stop when both touch the end
        Lnew = (FACTOR *) malloc(sizeof(FACTOR));   // get a LCM's new node
        if (Acurr == NULL) {                    // if A is at the end
            Lnew->prime = Bcurr->prime;         // store prime of B to LCM
            Lnew->power = Bcurr->power;         // store power of B to LCM
            Bcurr = Bcurr->next;                // find next B node
        } else if (Bcurr == NULL) {             // if B is at the end
            Lnew->prime = Acurr->prime;         // store prime of A to LCM
            Lnew->power = Acurr->power;         // store power of A to LCM
            Acurr = Acurr->next;                // find next A node
        } else if (Acurr->prime < Bcurr->prime) {   // if A's prime < B's prime
            Lnew->prime = Acurr->prime;         // store A's priem to LCM
            Lnew->power = Acurr->power;         // find A's power to LCM
            Acurr = Acurr->next;                // find next A node
        } else if (Acurr->prime > Bcurr->prime) {   // if A's prime > B's prime
            Lnew->prime = Bcurr->prime;         // store B's prime to LCM
            Lnew->power = Bcurr->power;         // store B's power to LCM
            Bcurr = Bcurr->next;                // find next B node
        } else {
            Lnew->prime = Acurr->prime;         // store prime to LCM
            Lnew->power = Acurr->power > Bcurr->power ?
                Acurr->power : Bcurr->power;    // store the bigger power
            Acurr = Acurr->next;                // find next A node
            Bcurr = Bcurr->next;                // find next B node
        }
        if (Lhead == NULL) {                    // if no head yet
            Lhead = Lnew;                       // let head be new node
            Lcurr = Lnew;                       // let current be new node
        } else {
            Lcurr->next = Lnew;                 // the next node is new node
            Lcurr = Lcurr->next;                // go to next node
        }
    }
    return Lhead;                               // return head of LCM
}

// To print out all primes and their associated powers and compute the products
// input: FACTOR *A: the linked of primes and associated powers
// return: no return
// output: the primes and their associated powers and the products
void write(FACTOR *A)
{
    int products = 1;                           // the product of all factors
    int j;                                      // loop control
    FACTOR *curr = A;                           // the current node

```

```c
    if (A == NULL) {                            // if head is NULL
        printf("1 = 1\n");                      // print 1 = 1
        return;                                 // leave the function
    }
    printf("%d", A->prime);                     // print the first prime
    if (A->power > 1) printf("^%d", A->power);  // print power if it > 1
    for (j = 0; j < A->power; j++) {
        products *= A->prime;                   // compute product
    }
    curr = curr->next;                          // go to next node
    while (curr != NULL) {                      // stop while toching the end
        printf(" * %d", curr->prime);           // print the prime
        if (curr->power > 1)
            printf("^%d", curr->power);         // print power if it > 1
        for (j = 0; j < curr->power; j++) {
            products *= curr->prime;            // compute product
        }
        curr = curr->next;                      // go to next node
    }
    printf(" = %d\n", products);                // print the products
}
```