lab11

```
$ gcc lab11.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
```

```
score: 96.0
o. [Output] Program output is correct, good.
o. [Format] Program format can be improved.
o. [Program] has memory leak problem.
```

lab11.c

```c
// EE231002 Lab11. Linked Lists
// 110060007, 黃俊穎
// 2021/12/20

#include <stdio.h>
#include <stdlib.h>

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;

// factorize num, N = num
FACTOR *factorize(int N);
// set GCD link, input *A and *B links
FACTOR *GCD(FACTOR *A, FACTOR *B);
// set LCM link, input *A and *B links
FACTOR *LCM(FACTOR *A, FACTOR *B);
// print out prime factors and powers, input *A link
void write(FACTOR *A);

int main(int argc, char *argv[])
{
    int num1, num2;                // input numbers
    FACTOR *A, *B;                 // save links

    num1 = atoi(argv[1]);
    num2 = atoi(argv[2]);          // convert input numbers to num1 & num2

    // print out factorized results
    printf("A = ");
    A = factorize(num1);
    write(A);
    printf("B = ");
    B = factorize(num2);
    write(B);

    // print out GCD
    printf("GCD = ");
    write(GCD(A, B));

    // print out LCM
    printf("LCM = ");
    write(LCM(A, B));

    return 0;
}

// factorize num, N = num
FACTOR *factorize(int N)
{
    int i, power; // i for prime in loop detecting, power for power number
    FACTOR *first = NULL, *temp, *new_factor;   // linked list

    // factorize num by each prime, until num = 1
    // increment power each time if num is divided by the same prime
    for (i = 2; N != 1; i++) {
        power = 0;
        while (N % i == 0) {
            N /= i;
            power++;
        }
        // set linked list
        if (power != 0) {
            new_factor = malloc(sizeof(FACTOR));
            new_factor->prime = i;
            new_factor->power = power;
            new_factor->next = NULL;
            if (first == NULL)
                first = new_factor;
            else
                temp->next = new_factor;
            temp = new_factor;
        }
    }
    return first;
}

// set GCD link, input *A and *B links
FACTOR *GCD(FACTOR *A, FACTOR *B)
```

```
{
    // linked list pointers declaration
    FACTOR *first = NULL, *temp, *new_factor = NULL;

    // search for the common prime num, exit until a or b end
    while (A != NULL && B != NULL) {
        if (A->prime < B->prime)
            A = A->next;           // shift A
        else if (A->prime > B->prime)
            B = B->next;           // shift B
        else {
            // when find out common prime num, store prime num
            new_factor = malloc(sizeof(FACTOR));
            new_factor->prime = A->prime;

            // store smaller power
            if (A->power >= B->power)
                new_factor->power = B->power;
            else
                new_factor->power = A->power;

            // connect nodes
            new_factor->next = NULL;
            if (first == NULL)
                first = new_factor;
            else
                temp->next = new_factor;
            temp = new_factor;

            // shift A and B
            A = A->next;
            B = B->next;
        }
    }

    // if 2 numbers relatively prime, return 1 * 1, else return linked list
    if (new_factor == NULL) {
        new_factor = malloc(sizeof(FACTOR));
        new_factor->prime = 1;
        new_factor->power = 1;
        new_factor->next = NULL;
        return new_factor;
    }
    else
        return first;
}

// set LCM link. input *A and *B links
FACTOR *LCM(FACTOR *A, FACTOR *B)
{
    // linked list pointers declaration
    FACTOR *first = NULL, *temp, *new_factor;

    // store all prime num, exit until a or b ended
    while (A != NULL && B != NULL) {
        // if A and B have different prime num, store it.
        // or store bigger power one
        if (A->prime < B->prime) {
            new_factor = malloc(sizeof(FACTOR));
            new_factor->prime = A->prime;
            new_factor->power = A->power;
            new_factor->next = NULL;
            if (first == NULL)
                first = new_factor;
            else
                temp->next = new_factor;
            temp = new_factor;
            A = A->next;            // shift A
        } else if (A->prime > B->prime) {
            new_factor = malloc(sizeof(FACTOR));
            new_factor->prime = B->prime;
            new_factor->power = B->power;
            new_factor->next = NULL;
            if (first == NULL)
                first = new_factor;
            else
                temp->next = new_factor;
            temp = new_factor;
            B = B->next;            // shift B
        } else {
            // store larger power when same prime
            new_factor = malloc(sizeof(FACTOR));
            new_factor->prime = A->prime;

            if (A->power >= B->power)
                new_factor->power = A->power;
            else
                new_factor->power = B->power;

            // connect nodes
            new_factor->next = NULL;
            if (first == NULL)
                first = new_factor;
            else
                temp->next = new_factor;
            temp = new_factor;

            // shift A and B
            A = A->next;
            B = B->next;
        }
    }
    // A or B has come to NULL, need to finish linking the remaining primes
    if (A == NULL && B != NULL) { // B is the one remaining
        while (B != NULL) {
            new_factor = malloc(sizeof(FACTOR));
            new_factor->prime = B->prime;
            new_factor->power = B->power;
            new_factor->next = NULL;
            temp->next = new_factor;
            temp = new_factor;
            B = B->next;            // shift B
        }
    } else { // A is the one remaining
        while (A != NULL){
        while (A != NULL) {
            new_factor = malloc(sizeof(FACTOR));
            new_factor->prime = A->prime;
            new_factor->power = A->power;
            new_factor->next = NULL;
            temp->next = new_factor;
            temp = new_factor;
            A = A->next;            // shift A
        }
    }
    // return linked list
    return first;
}

// print out prime factors and powers, input *A
void write(FACTOR *A)
{
    int product = 1, power;             // initialize variables

    // print out linked list
    for (; A != NULL; A = A->next) {
        // if power = 1, don't print power, otherwise, print it out
        if (A->power == 1)
            printf("%d", A->prime);
        else
            printf("%d^%d", A->prime, A->power);

        // if isn't last prime, print '*'
        if (A->next != NULL)
            printf(" * ");

        // save power product
        power = A->power;

        // calculate total product
        for (; power != 0; power--) {
            product *= A->prime;
        }
    }
    // print product
    printf(" = %d\n", product);
}
```