# lab12

```c
// EE231002 Lab12. Linked Lists
// 108061112, 林靖
// Date: Dec. 14, 2019

#include <stdio.h>                  // Standard input and output library
#include <stdlib.h>                 // Defined malloc(), atoi(), EXIT_FAILURE

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;                           // Linked list for each prime factor

FACTOR *factorize(int N);
// This function factorizes the input N into its prime factors and their
//      associated powers, and returns a linked list that contains all these
//      prime factors.

FACTOR *GCD(FACTOR *A, FACTOR *B);
// This function takes two linked lists of prime factors as input, and finds the
//      Greatest Common Divisor of these two inputs. Note that it returns a
//      linked list of prime factors.

FACTOR *LCM(FACTOR *A, FACTOR *B);
// This function takes two linked lists of prime factors as input, and finds the
//      Least Common Multiple of these two inputs. Note that it also returns a
//      linked list of prime factors.

void write(FACTOR *A);
// This function prints out all the prime factors and their associated powers.
//      In addition, it recalculates the product of all the factors and prints
//      out at the end.

int main(int argc, char *argv[])            // Called at program startup
{
    FACTOR *A;                              // Point to 1st node of factorized A
    FACTOR *B;                              // Point to 1st node of factorized B

    A = factorize(atoi(argv[argc - 2]));    // Take two command line arguments
                       Why  argc - 2?
    B = factorize(atoi(argv[argc - 1]));    //      as the inputs to the program

    printf("A =");
    write(A);                               // Print factorized A out
    printf("B =");
    write(B);                               // Print factorized B out
    printf("GCD =");
    write(GCD(A, B));                       // Print the GCD of A and B out
    Memory leak!
    printf("LCM =");
    write(LCM(A, B));                       // Print the LCM of A and B out

    return 0;                               // Normal program termination
}

// This function factorizes the input N into its prime factors and their
//      associated powers, and returns a linked list that contains all these
//      prime factors.
FACTOR *factorize(int N)
{
    int divisor = 2;                // For testing whether N is divisible
    FACTOR *head = NULL;            // Point to the 1st node of factorized N
    FACTOR *tail;                   // Point to the last node of factorized N
    FACTOR *new;                    // Point to the newly created node

    while(N > 1) {                  // Keep trying division until N becomes 1
    while (N > 1) {                  // Keep trying division until N becomes 1
        if (N % divisor == 0) {     // If N is divisible by the divisor
            new = (FACTOR *)malloc(sizeof(FACTOR)); // Allocate a block of
            if (new == NULL)                        //    memory and test to see
                exit(EXIT_FAILURE);                 //    if it's a null pointer
            new->prime = divisor;   // Save this factor into newly created node
            for (new->power = 0;    // Computing the power of this factor
                 N % divisor == 0;  //      While N is divisible by this divisor
                 N /= divisor,      //      This divisor is one of the factor
                 new->power++) ;    //      Update the power of this factor
            new->next = NULL;       // Mark the end of this linked list
            if (head == NULL)       // If this new node is the 1st node created
                head = new;         //      Make head point to this new node
            else                    // Otherwise
                tail->next = new;   //      Make tail->next point to new node
            tail = new;             // Make tail point to the newly created node
        }
        if (divisor == 2)           // If divisor is 2
            divisor = 3;            //      Make it 3
        else                        // Otherwise
            divisor += 2;           //      N must be odd so skip even divisors
    }

    return head;                    // Return the 1st node of the factorized N
}

// This function takes two linked lists of prime factors as input, and finds the
//      Greatest Common Divisor of these two inputs. Note that it returns a
//      linked list of prime factors.
FACTOR *GCD(FACTOR *A, FACTOR *B)
{
    FACTOR *head = NULL;            // Point to the 1st node of factorized N
    FACTOR *tail;                   // Point to the last node of factorized N
    FACTOR *new;                    // Point to the newly created node

    while (A != NULL && B != NULL) {    // Until one of A and B points to NULL
        if (A->prime < B->prime) {          // If this factor of A is smaller
            A = A->next;                    //      Look for larger factor of A
        } else if (A->prime > B->prime) {   // If this factor of B is smaller
            B = B->next;                    //      Look for larger factor of B
        } else {                            // If these two factors are equal
            new = (FACTOR *)malloc(sizeof(FACTOR)); // Allocate a block of
            if (new == NULL)                        //    memory and test to see
                exit(EXIT_FAILURE);                 //    if it's a null pointer
            new->prime = A->prime;      // Save this factor into the new node
            if (A->power < B->power)    // Find the smaller power to save
                new->power = A->power;  //      Power of A is smaller so save it
            else
                new->power = B->power;  //      Power of B is smaller so save it
            new->next = NULL;       // Mark the end of this linked list
            if (head == NULL)       // If this new node is the 1st node created
                head = new;         //      Make head point to this new node
            else                    // Otherwise
                tail->next = new;   //      Make tail->next point to new node
            tail = new;             // Make tail point to the newly created node
            A = A->next;            // Look for larger factor of A
            B = B->next;            // Look for larger factor of B
        }
    }

    return head;                    // Return the 1st node of the GCD of A and B
}

// This function takes two linked lists of prime factors as input, and finds the
//      Least Common Multiple of these two inputs. Note that it also returns a
//      linked list of prime factors.
FACTOR *LCM(FACTOR *A, FACTOR *B)
{
    FACTOR *head = NULL;            // Point to the 1st node of factorized N
    FACTOR *tail;                   // Point to the last node of factorized N
    FACTOR *new;                    // Point to the newly created node

    while (A != NULL || B != NULL) {    // Until A and B both point to NULL
        new = (FACTOR *)malloc(sizeof(FACTOR)); // Allocate a block of
        if (new == NULL)                        //    memory and test to see
            exit(EXIT_FAILURE);                 //    if it's a null pointer
        if (B == NULL ||                        // If the factor of A is smaller
            A != NULL && A->prime < B->prime) { //      than B, save the factor
            new->prime = A->prime;              //      and the power of A into
            new->power = A->power;              //      the new node, and look
            A = A->next;                        //      for larger factor of A.
        } else if (A == NULL ||                 // If the factor of B is smaller
            B != NULL && A->prime > B->prime) { //      than A, save the factor
            new->prime = B->prime;              //      and the power of B into
            new->power = B->power;              //      the new node, and look
            B = B->next;                        //      for larger factor of B.
        } else {                        // If these two factors are equal
            new->prime = A->prime;      // Save this factor into the new node
            if (A->power > B->power)    // Find the larger power to save
                new->power = A->power;  //      Power of A is larger so save it
            else
                new->power = B->power;  //      Power of B os larger so save it
            A = A->next;            // Look for larger factor of A
            B = B->next;            // Look for larger factor if B
        }
        new->next = NULL;           // Mark the end of this linked list
        if (head == NULL)           // If this new node is the 1st node created
            head = new;             //      Make head point to this new node
        else                        // Otherwise
            tail->next = new;       //      Make tail->next point to new node
        tail = new;                 // Make tail point to the newly created node
    }

    return head;                    // Return the 1st node of the LCM of A and B
}

// This function prints out all the prime factors and their associated powers.
//      In addition, it recalculates the product of all the factors and prints
//      out at the end.
void write(FACTOR *A)
{
    int i;                              // The index fpr looping
    int product = 1;                    // The product of A

    if (A == NULL)                      // Exception handling
        printf(" 1");

    for ( ; A != NULL; A = A->next) {   // For each nodes
        printf(" %d", A->prime);        //      The base of this term
        if (A->power > 1)
            printf("^%d", A->power);    //      The exponent of this term
        if (A->next != NULL)
            printf(" *");               //      Multiplication between two terms
        for (i = 0; i < A->power; i++)
            product *= A->prime;        //      Computing the exponentiation
    }

    printf(" = %d\n", product);         // Print the final product of A

    return;                             // Function termination
}
```

[Format] can be improved.
[Coding] lab12.c spelling errors: fpr(1), os(1)
[Efficiency] can be improved.

Score: 95