

lab12

```
1 // EE231002 Lab12. Linked Lists
2 // 108061112, 林靖
3 // Date: Dec. 14, 2019
4
5 #include <stdio.h>           // Standard input and output library
6 #include <stdlib.h>         // Defined malloc(), atoi(), EXIT_FAILURE
7
8 typedef struct factor {      // node for a prime factor
9     int prime;               //     prime factor
10    int power;               //     associated power
11    struct factor *next;     //     pointer for the next prime factor
12 } FACTOR;                   // Linked list for each prime factor
13
14 FACTOR *factorize(int N);
15 // This function factorizes the input N into its prime factors and their
16 //     associated powers, and returns a linked list that contains all these
17 //     prime factors.
18
19 FACTOR *GCD(FACTOR *A, FACTOR *B);
20 // This function takes two linked lists of prime factors as input, and finds the
21 //     Greatest Common Divisor of these two inputs. Note that it returns a
22 //     linked list of prime factors.
23
24 FACTOR *LCM(FACTOR *A, FACTOR *B);
25 // This function takes two linked lists of prime factors as input, and finds the
26 //     Least Common Multiple of these two inputs. Note that it also returns a
27 //     linked list of prime factors.
28
29 void write(FACTOR *A);
30 // This function prints out all the prime factors and their associated powers.
31 //     In addition, it recalculates the product of all the factors and prints
32 //     out at the end.
33
34 int main(int argc, char *argv[]) // Called at program startup
35 {
36     FACTOR *A;                   // Point to 1st node of factorized A
37     FACTOR *B;                   // Point to 1st node of factorized B
38
39     A = factorize(atoi(argv[argc - 2])); // Take two command line arguments
40     B = factorize(atoi(argv[argc - 1])); //     as the inputs to the program
41
42     printf("A =");
43     write(A);                     // Print factorized A out
44     printf("B =");
45     write(B);                     // Print factorized B out
46     printf("GCD =");
47     write(GCD(A, B));             // Print the GCD of A and B out
```

```

Memory leak!
48 printf("LCM =");
49 write(LCM(A, B)); // Print the LCM of A and B out
50
51 return 0; // Normal program termination
52 }
53
54 // This function factorizes the input N into its prime factors and their
55 // associated powers, and returns a linked list that contains all these
56 // prime factors.
57 FACTOR *factorize(int N)
58 {
59     int divisor = 2; // For testing whether N is divisible
60     FACTOR *head = NULL; // Point to the 1st node of factorized N
61     FACTOR *tail; // Point to the last node of factorized N
62     FACTOR *new; // Point to the newly created node
63
64     while(N > 1) { // Keep trying division until N becomes 1
65         while (N > 1) { // Keep trying division until N becomes 1
66             if (N % divisor == 0) { // If N is divisible by the divisor
67                 new = (FACTOR *)malloc(sizeof(FACTOR)); // Allocate a block of
68                 if (new == NULL) // memory and test to see
69                     exit(EXIT_FAILURE); // if it's a null pointer
70                 new->prime = divisor; // Save this factor into newly created node
71                 for (new->power = 0; // Computing the power of this factor
72                     N % divisor == 0; // While N is divisible by this divisor
73                     N /= divisor, // This divisor is one of the factor
74                     new->power++); // Update the power of this factor
75                 new->next = NULL; // Mark the end of this linked list
76                 if (head == NULL) // If this new node is the 1st node created
77                     head = new; // Make head point to this new node
78                 else // Otherwise
79                     tail->next = new; // Make tail->next point to new node
80                 tail = new; // Make tail point to the newly created node
81             }
82             if (divisor == 2) // If divisor is 2
83                 divisor = 3; // Make it 3
84             else // Otherwise
85                 divisor += 2; // N must be odd so skip even divisors
86         }
87     }
88     return head; // Return the 1st node of the factorized N
89 }
90 // This function takes two linked lists of prime factors as input, and finds the
91 // Greatest Common Divisor of these two inputs. Note that it returns a
92 // linked list of prime factors.
93 FACTOR *GCD(FACTOR *A, FACTOR *B)
94 {
95     FACTOR *head = NULL; // Point to the 1st node of factorized N

```

```

96     FACTOR *tail;                // Point to the last node of factorized N
97     FACTOR *new;                // Point to the newly created node
98
99     while (A != NULL && B != NULL) { // Until one of A and B points to NULL
100         if (A->prime < B->prime) { // If this factor of A is smaller
101             A = A->next;           // Look for larger factor of A
102         } else if (A->prime > B->prime) { // If this factor of B is smaller
103             B = B->next;           // Look for larger factor of B
104         } else {                  // If these two factors are equal
105             new = (FACTOR *)malloc(sizeof(FACTOR)); // Allocate a block of
106             if (new == NULL)       // memory and test to see
107                 exit(EXIT_FAILURE); // if it's a null pointer
108             new->prime = A->prime; // Save this factor into the new node
109             if (A->power < B->power) // Find the smaller power to save
110                 new->power = A->power; // Power of A is smaller so save it
111             else
112                 new->power = B->power; // Power of B is smaller so save it
113             new->next = NULL; // Mark the end of this linked list
114             if (head == NULL) // If this new node is the 1st node created
115                 head = new; // Make head point to this new node
116             else // Otherwise
117                 tail->next = new; // Make tail->next point to new node
118             tail = new; // Make tail point to the newly created node
119             A = A->next; // Look for larger factor of A
120             B = B->next; // Look for larger factor of B
121         }
122     }
123
124     return head; // Return the 1st node of the GCD of A and B
125 }
126
127 // This function takes two linked lists of prime factors as input, and finds the
128 // Least Common Multiple of these two inputs. Note that it also returns a
129 // linked list of prime factors.
130 FACTOR *LCM(FACTOR *A, FACTOR *B)
131 {
132     FACTOR *head = NULL; // Point to the 1st node of factorized N
133     FACTOR *tail; // Point to the last node of factorized N
134     FACTOR *new; // Point to the newly created node
135
136     while (A != NULL || B != NULL) { // Until A and B both point to NULL
137         new = (FACTOR *)malloc(sizeof(FACTOR)); // Allocate a block of
138         if (new == NULL) // memory and test to see
139             exit(EXIT_FAILURE); // if it's a null pointer
140         if (B == NULL || // If the factor of A is smaller
141             A != NULL && A->prime < B->prime) { // than B, save the factor
142             new->prime = A->prime; // and the power of A into
143             new->power = A->power; // the new node, and look
144             A = A->next; // for larger factor of A.
145         } else if (A == NULL || // If the factor of B is smaller

```

```

146         B != NULL && A->prime > B->prime) { //      than A, save the factor
147         new->prime = B->prime;                //      and the power of B into
148         new->power = B->power;                //      the new node, and look
149         B = B->next;                          //      for larger factor of B.
150     } else {                                  // If these two factors are equal
151         new->prime = A->prime;                // Save this factor into the new node
152         if (A->power > B->power)            // Find the larger power to save
153             new->power = A->power; //      Power of A is larger so save it
154         else
155             new->power = B->power; //      Power of B os larger so save it
156         A = A->next;                          // Look for larger factor of A
157         B = B->next;                          // Look for larger factor if B
158     }
159     new->next = NULL;                          // Mark the end of this linked list
160     if (head == NULL)                        // If this new node is the 1st node created
161         head = new;                          //      Make head point to this new node
162     else                                     // Otherwise
163         tail->next = new;                    //      Make tail->next point to new node
164     tail = new;                              // Make tail point to the newly created node
165 }
166
167 return head;                                // Return the 1st node of the LCM of A and B
168 }
169
170 // This function prints out all the prime factors and their associated powers.
171 //      In addition, it recalculates the product of all the factors and prints
172 //      out at the end.
173 void write(FACTOR *A)
174 {
175     int i;                                    // The index fpr looping
176     int product = 1;                          // The product of A
177
178     if (A == NULL)                            // Exception handling
179         printf(" 1");
180
181     for ( ; A != NULL; A = A->next) { // For each nodes
182         printf(" %d", A->prime);           //      The base of this term
183         if (A->power > 1)
184             printf("^%d", A->power); //      The exponent of this term
185         if (A->next != NULL)
186             printf(" *"); //      Multiplication between two terms
187         for (i = 0; i < A->power; i++)
188             product *= A->prime; //      Computing the exponentiation
189     }
190
191     printf(" = %d\n", product); // Print the final product of A
192
193     return; // Function termination
194 }

```

[Format] can be improved.

[Coding] lab12.c spelling errors: fpr(1), os(1)

[Efficiency] can be improved.

Score: 95