lab09

1 // EE231002 Lab09. GCD and LCM

2 // 108061112, 林靖

3 // Nov. 23, 2019

4

5 #include <stdio.h> // Standard input and output library 6

7 #define S 20 // Size of array

8 // Convert N into its prime factors 9 void factorize(int N, int factors[S], int power[S]);

10 void GCD(int Afactors[S], int Apower[S], // Takes two factors arrays and two 11 int Bfactors[S], int Bpower[S], // power arrays to produce the arrays 12 int Cfactors[S], int Cpower[S]); // of the Greatest Common Divisor. 13 void LCM(int Afactors[S], int Apower[S],

14 int Bfactors[S], int Bpower[S], // To produce the arrays of the Least 15 int Cfactors[S], int Cpower[S]); // Common Multiple. 16 void write(int factors[S], int power[S]); // To print out the factors and power 17 // arrays using the product form. 18 int main(void)

19 { // Called at program startup 20 int inputA, inputB; // Two integers to be factorized 21 int Afactors[S], Apower[S] = {0}; // Factorized inputA 22 int Bfactors[S], Bpower[S] = {0}; // Factorized inputB 23 int Cfactors[S], Cpower[S]; // GCD or LCM of inputA & inputB 24 printf("input A: "); // Prompt user to input an integer 25 scanf("%d", &inputA); // Read in inputA 26 printf("input B: "); // Prompt user to input an integer 27 scanf("%d", &inputB); // Read in inputB 28 factorize(inputA, Afactors, Apower); // Convert inputA into its factors 29 printf(" A = "); // Print out factors and powers of 30 write(Afactors, Apower); // inputA using the product form. 31 factorize(inputB, Bfactors, Bpower); // Convert inputB into its factors 32 printf(" B = "); // Print out factors and powers of 33 write(Bfactors, Bpower); // inputB using the product form. 34 GCD(Afactors, Apower, Bfactors, Bpower, Cfactors, Cpower); // Compute GCD 35 printf(" GCD(A,B) = "); // Print out GCD of inputA & inputB 36 write(Cfactors, Cpower); // using the product form. 37 LCM(Afactors, Apower, Bfactors, Bpower, Cfactors, Cpower); // Compute LCM 38 printf(" LCM(A,B) = "); // Print out LCM of inputA & inputB 39 write(Cfactors, Cpower); // using the product form. 40 return 0; // Normal program termination 41 }

42

43 void factorize(int N, int factors[S], int power[S])

44 { // Convert N into its prime factors 45 int divisor = 2; // Candidate factor, trial division 46 int i = 0; // Common index for arrays 47

48 while (N % divisor == 0) { // Keep dividing by 2 untill N become odd 1

49 N /= divisor; // Update N if divisible 50 power[i]++; // Update power if divisible 51 }

52 if (power[i]) { // If power != 0 (divisible) 53 factors[i] = divisor; // Write divisor into array 54 i++; // Update common index for arrays 55 }

56 for (divisor = 3; divisor \* divisor <= N; divisor += 2) { // N must be odd 57 while (N % divisor == 0) { // Keep dividing if divisible 58 N /= divisor; // Update N if divisible 59 power[i]++; // Update power if divisible 60 }

61 if (power[i]) { // If power != 0 (divisible) 62 factors[i] = divisor; // Write divisor into array 63 i++; // Update common index for arrays 64 }

65 }

66 factors[i] = N; // N must be 1 or a prime number now 67 power[i] = 1;

68 if (factors[i] != 1) { // If N is a prime number 69 i++;

70 factors[i] = 1; // Ensure arrays are terminated by 1 71 power[i] = 1;

72 }

73 return; // Function termination

74 }

75

76 void GCD(int Afactors[S], int Apower[S], // Takes two factors arrays and two 77 int Bfactors[S], int Bpower[S], // power arrays to produce the arrays 78 int Cfactors[S], int Cpower[S]) // of the Greatest Common Divisor. 79 {

80 int a, b, c = 0; // Indice for arrays of A, B, and GCD 81

82 for (a = 0; Afactors[a] != 1; a++) { // To find common prime factors 83 for (b = c; Bfactors[b] <= Afactors[a] && Bfactors[b] != 1; b++) { 84 if (Afactors[a] == Bfactors[b]) { // Common prime factor 85 Cfactors[c] = Afactors[a]; // Write the factor into array 86 if (Apower[a] < Bpower[b]) { // Determine the smaller power 87 Cpower[c] = Apower[a]; // Write smaller one into array 88 } else {

89 Cpower[c] = Bpower[b];

90 }

91 c++; // Update index for GCD arrays 92 }

93 }

94 }

95 Cfactors[c] = 1; // Ensure arrays are terminated by 1 96 Cpower[c] = 1;

97 return; // Function termination

98 }

2

99

100 void LCM(int Afactors[S], int Apower[S],

101 int Bfactors[S], int Bpower[S], // To produce the arrays of the Least 102 int Cfactors[S], int Cpower[S]) // Common Multiple. 103 {

104 int a = 0, b = 0, c = 0; // Indice for arrays of A, B, and LCM 105

106 while (Afactors[a] != 1 && Bfactors[b] != 1) { // Sort from small to large 107 if (Afactors[a] < Bfactors[b]) { // If A is smaller than B 108 Cfactors[c] = Afactors[a]; // Write the smaller ones 109 Cpower[c] = Apower[a]; // into the arrays. 110 a++; // Update the index of A 111 } else if (Afactors[a] > Bfactors[b]) { // If B is smaller than A 112 Cfactors[c] = Bfactors[b]; // Write the smaller ones into 113 Cpower[c] = Bpower[b]; // the arrays. 114 b++; // Update the index of A 115 } else { // If A equal to B 116 Cfactors[c] = Afactors[a]; // Write arbitrarily A or B factor 117 if (Apower[a] > Bpower[b]) { // Determine the larger power 118 Cpower[c] = Apower[a]; // Write the larger power into array 119 } else {

120 Cpower[c] = Bpower[b];

121 }

122 a++; // Update both A's & B's indice 123 b++;

124 }

125 c++; // Update the index of LCM arrays 126 }

127 for ( ; Afactors[a] != 1; a++) {

128 Cfactors[c] = Afactors[a]; // Copy the remaining elements in A into 129 Cpower[c] = Apower[a];; // LCM arrays

Cpower[c] = Apower[a]; ; // LCM arrays

130 c++; // Update indice of LCM arrays 131 }

132 for ( ; Bfactors[b] != 1; b++) {

133 Cfactors[c] = Bfactors[b]; // Copy the remaining elements in B into 134 Cpower[c] = Bpower[b]; // LCM arrays

135 c++; // Update indice of LCM arrays 136 }

137 Cfactors[c] = 1; // Ensure arrays are terminated by 1 138 Cpower[c] = 1;

139 return; // Function termination 140 }

141

142 void write(int factors[S], int power[S]) // To print out the factors and power 143 { // arrays using the product form. 144 int i, j; // Indice for looping 145 int product = 1; // Product of factors and powers arrays 146

147 for (i = 0; factors[i] != 1; i++) { // Loop through all elements 3

148 for (j = 0; j < power[i]; j++)

149 product \*= factors[i]; // Compute the factor to its power 150 if (i) // Condition to print multiplicate sign 151 printf(" \* "); // The multiplicate sign 152 printf("%d", factors[i]); // The prime factor 153 if (power[i] != 1) // Condition to print power of factor 154 printf("^%d", power[i]); // The power of the factor 155 }

156 if (product == 1) // Handle in case coprime 157 printf("1"); // GCD = 1 if coprime 158 printf(" = %d\n", product); // Print out the product 159 return; // Function termination 160 }

[Return] is provided.

[Coding] lab09.c spelling errors: Indice(3), indice(3), multiplicate(2), untill(1) [factorize] can be more efficient.

[GCD] can be more efficient.

Score: 82

4