lab04

```
// EE231002 Lab04. Pythagorean Triples
// 108061112, 林靖
// Oct. 11, 2019

#include <stdio.h>          // standard input and output library
#define BOUND 20000         // 0 < a <= b <= c <= upper bound == 20000
#define LIMIT 141           // sqrt(20000) == 141 == upper limit > m > n > 0

#define CALCULATE_gcdMN                             \
    for (u = m, v = n; (u %= v) && (v %= u); ) ;   \
    gcdMN = u + v;          // Compute GCD (Greatest Common Divisor) of m and n
                            // using Euclidean algorithm.
                            // Note that this for loop has an empty updation
                            // and an empty statement.
#define CALCULATE_a_b_c     \
    ak = a = m * m - n * n; \
    bk = b = 2 * m * n;     \
    ck = c = m * m + n * n; // Here we're using Euclid's formula to generate
                            // primitive triples:
                            // a = m^2 - n^2
                            // b = 2 * m * n
                            // c = m^2 + n^2
                            // For avoidance of repetition, here we take
                            // m > n, where m and n are both positive integers.
                            // To ensure the primitivity, m and n have to be
                            // coprime and (m - n) has to be odd, which
                            // mathematically means that we could discuss them
                            // in 2 cases to improve the for loop efficiency.
                            // Note that using this formula does not guarantee
                            // a <= b, so we will need to use if-else to adjust
                            // the printing order later.
#define CALCULATE_ak_bk_ck  \
    ak += a;                \
    bk += b;                \
    ck += c;                // Euclid's formula does not generate non-primitive
                            // triples.
                            // This can be remedied by inserting an additional
                            // parameter k, where k is a positive integer.
                            // So the formula we're using would be:
                            // ak = a * k = (m^2 - n^2) * k
                            // bk = b * k = (2 * m * n) * k
                            // ck = c * k = (m^2 + n^2) * k
                            // In one loop, starting with k = 1, for each k,
                            // we must print out the corresponded triples, so
                            // here we use += operator to iterate the process.
#define OUTPUT_SORT_1       \
    printf("Pythagorean Triple #%hu is (%hu,%hu,%hu)\n", ++ttl, ak, bk, ck);
                            // This line failed to fit into the 80-charactor-
                            // wide layout in a neat and clean way.
                            // Therefore, we separate and redefine it to a
                            // shorter identifier for concise and tidy
                            // typesetting purposes.
#define OUTPUT_SORT_2       \
    printf("Pythagorean Triple #%hu is (%hu,%hu,%hu)\n", ++ttl, bk, ak, ck);
                            // Ditto. But in case a >= b, swap the positional
                            // parameters of ak and bk.


int main()                              // Here is the main function.
int main(void)                              // Here is the main function.
{
    unsigned short m, n;                // m and n are coprime, (m - n) is odd.
    unsigned short u, v;                // For macro to compute GCD of m and n.
    unsigned short gcdMN;               // Greatest Common Divisor of m and n.
    unsigned short a, b, c;             // Primitive triples, a <= b <= c.
    unsigned short ak, bk, ck;          // To generate non-primitive triples.
    unsigned short ttl = 0;             // Total number of triples found.


    for (m = 2; m <= LIMIT; m += 2) {       // Case#1: m is even and
        for (n = 1; n < m; n += 2) {        //         n is odd.
            CALCULATE_gcdMN;                // Evaluating GCD of m and n.
            if (1 == gcdMN) {               // We need m and n to be coprime.
                CALCULATE_a_b_c;            // Evaluate a, b, c from m, n given

                if (ak < bk) {                  // Sort#1: no need transpose.
                    while (ck <= BOUND) {           // Upper bound of triples.
                        OUTPUT_SORT_1;              // Printing out results.
                        CALCULATE_ak_bk_ck;         // Evaluating the next
                    }                               // non-primitive triple.
                } else {                        // Sort#2: do need transpose.
                    while (ck <= BOUND) {           // Upper bound of triples.
                        OUTPUT_SORT_2;              // Printing out results.
                        CALCULATE_ak_bk_ck;         // Evaluating the next
                    }                               // non-primitive triple.
                }
            }
        }
    }


    for (m = 1; m <= LIMIT; m += 2) {       // Case#2: m is odd and
        for (n = 2; n < m; n += 2) {        //         n is even.
            CALCULATE_gcdMN;                // Evaluating GCD of m and n.
            if (1 == gcdMN) {               // We need m and n to be coprime.
                CALCULATE_a_b_c;            // Evaluate a, b, c from m, n given

                if (ak < bk) {                  // Sort#1: no need transpose.
```

```
                    while (ck <= BOUND) {           // Upper bound of triples.
                        OUTPUT_SORT_1;              // Printing out results.
                        CALCULATE_ak_bk_ck;         // Evaluating the next
                    }                               // non-primitive triple.
                } else {                        // Sort#2: do need transpose.
                    while (ck <= BOUND) {           // Upper bound of triples.
                        OUTPUT_SORT_2;              // Printing out results.
                        CALCULATE_ak_bk_ck;         // Evaluating the next
                    }                               // non-primitive triple.
                }
            }
        }
    }

                                // Prining out the number of triples found.
    printf("Total number of Pythagorean triples found is %hu\n", ttl);
    return 0;                   // Indicates normal termination.
}

```

[Format] can be improved.
[Coding] is very difficult to read!
[Coding] lab04.c spelling errors: Prining(1), charactor(1), primitivity(1), updation(1)
[Coding] can be more concise.
[CPU time] 0.014199 sec

Score: 80