lab04

```
$ gcc lab04.c
lab04.c:84:24: warning: format specifies type 'long' but the argument has type 'int'
 [-Wformat]
        printf("n <= %ld.\n", max);                         /* use ld if max is
                      ~~~      ^~~
                      %d
lab04.c:22:13: note: expanded from macro 'max'
#define max 5000_
            ^~~~
1 warning generated.
$ ./a.out
1: 2 ^ 3 = 2 ^ 2 + 2 ^ 2
2: 8 ^ 3 = 16 ^ 2 + 16 ^ 2
3: 32 ^ 3 = 128 ^ 2 + 128 ^ 2
...
...
1399: 4985 ^ 3 = 94715 ^ 2 + 338980 ^ 2
1400: 4986 ^ 3 = 74790 ^ 2 + 344034 ^ 2
1401: 4993 ^ 3 = 159776 ^ 2 + 314559 ^ 2
1401 solutions found for n <= 5000.

CPU time: 0.0100579 sec
score: 70
o. Compilation warnings.
o. [Format] Program format can be improved
o. [Coding] lab04.c spelling errors: caculate(1), increse(1)
o. [Efficiency] can be improved.
```

lab04.c

```c
// EE2310 lab04 Solving Integer Equation
// 109061217, 林峻霆
// Date: 2020/10/19
/*
 I will explain my algorithm here first because i use a different method.

 I use a theorem called "Sum of two squares". In this theorem, it said that
 n = a^2 + b^2 has solutions if and only if its prime decomposition contains no term p^k where p % 4 = 3 and k is odd. According to this theorem , we
 can notice that the solution set of n^3 = a^2 + b^2 is equivalent to
 n = a^2 + b^2.Therefore, we may replace the question with solving
 n = a^2 + b^2, and print n^3 = (an)^2 + (bn)^2 at the end.

 Dealing with every n <= max, approaching it with modulo is a good way.For
 any integer a, a^2 % 4 = 0 or 1, so n % 4 = 0,1 or 2.Therefore, we dont
 need to test n % 4 = 3 case.Additionally, we may find out that n % 4 = 0
 case can be derived from n % 4 = 1 or 2 cases.To sum up, we just only need  to test n % 4 = 1 or 2 case and I also use some properties of odd and even
 while calculating.
*/

#include <stdio.h>

#define max 5000

int main(void)
{
    int n;                                          // n in the equation
    int a;                                          // a in the equation
    int b;                                          // b in the equation
    int i = 1;                                      // scalar i
    int j = 1;                                      // scalar j
    int total = 0;                                  // amount of n

    for (n=2; n <= max; n++) {                      // initialize a loop
    for (n = 2; n <= max; n++) {                      // initialize a loop
        if (n % 4 == 1) {                           // judge cases
            for (a = 1; a * a <= n/2; a++) {
```

```
            for (a = 1; a * a <= n / 2; a++) {
36              for (b = a + 1; b * b + a * a <= n; b = b + 2) {
37                  if (n == a * a + b * b) {          // judge cases
38                      while(n * i <= max){
                        while (n * i <= max) {
39                          int n_output = n * i;      // the output n
Do not mix declarations with statements
40                          int a_output = a * n * j;  // the output a
41                          int b_output = b * n * j;  // the output b
42                          total += 1;                // calculate amount
43                          printf("%d: ", total);     // print the result
44                          printf("%d ^ 3 = ", n_output);
45                          printf("%d ^ 2 + ", a_output);
46                          printf("%d ^ 2\n", b_output);
47                          i = i * 4;                 // increase scalar
48                          j = j * 8;                 // increase scalar
49                      }
50                      i = 1;                         // change back to 1
51                      j = 1;                         // change back to 1
52                      b = n;                         // exit the loop
53                      a = b;                         // exit the loop
54                  }
55              }
56          }
57      }
58      else if(n % 4 ==2){                            // judge cases
        else if (n % 4 == 2) {                           // judge cases
59          for(a=1; a * a <= n/2; a = a + 1){         // initialize loop
            for (a = 1; a * a <= n / 2; a = a + 1) {         // initialize loop
60              for(b = a; b * b + a * a <= n; b = b + 1) {
                for (b = a; b * b + a * a <= n; b = b + 1) {
61                  if(n == a * a + b * b){            // judge
                    if (n == a * a + b * b) {            // judge
62                      while(n * i <= max){
                        while (n * i <= max) {
63                          int n_output = n * i;      // the output n
Do not mix declarations with statements
64                          int a_output = a * n * j;  // the output a
65                          int b_output = b * n * j;  // the output b
66                          total += 1;                // caculate total
67                          printf("%d: ", total);     // print the result
```

```
68                              printf("%d ^ 3 = ", n_output);
69                              printf("%d ^ 2 + ", a_output);
70                              printf("%d ^ 2\n", b_output);
71                              i = i * 4;                  // increse scalar
72                              j = j * 8;                  // increase scalar
73                          }
74                          i = 1;                          // change back to 1
75                          j = 1;                          // change back to 1
76                          b = n;                          // exit the loop
77                          a = b;                          // exit the loop
78                      }
79                  }
80              }
81          }
82      }
83      printf("%d solutions found for ", total);          // print the amount
84      printf("n <= %ld.\n", max);                        /* use ld if max is
85                                                            big*/
86      return 0;                                          // end the program
87  }
88
```