Mass center of particles moving at constant velocity X30485_en

`Preliminaries`
In this exercise we will implement a program that prints real numbers on the output. Due to format and efficiency issues of `cin` and `cout` when they manage real numbers, it is convenient to do some things that we explain as follows.
At the beginning of function `main` you should place these instructions:

```
ios::sync_with_stdio(false);
cin.tie(0);
cout.setf(ios::fixed);
cout.precision(5);
```

In order to print real numbers on the output, it is convenient to include and use this function:

```
void printDouble(double d)
{
if (abs(d) < 1e-7)
cout << 0.0;
else
cout << d;
}
```

Last but not least, do not use `endl` to print break lines. Use `'\n'` instead, that is:

```
// This line has been replaced with next one:
// cout << endl;
cout << '\n';
```

Exercise
Note: In this exercise we will speak about positions of particles in a 3-dimensional system, and about their velocities and masses. The reference and units system is not too relevant, as long as the chosen one is more or less standard and reasonable. Just in case someone feels the need, you can imagine that we speak about meters, seconds, meters per second and kilograms.
As input, we will have the position, velocity and mass of n particles $(p_1, v_1, m_1), \ldots, (p_n, v_n, m_n)$. In particular, the first particle is at position p_1 at time 0, moves at constant velocity v_1, and its mass is m_1. Everything is supposed to be represented in a cartesian coordinate system with three dimensions.
Furthermore, we will have k values of time $t_1, \ldots, t_k$ as input.
We will have to determine the mass center of the n particles after t_1 time units, after $t_1 + t_2$ time units, after $t_1 + t_2 + t_3$, ..., after $t_1 + \cdots + t_k$ time units.
It is compulsory to coherently use the following declarations of types, and implement and coherently use the following functions. Otherwise, the delivery will be invalidated. You can declare more data types and implement more functions if you feel like doing so. That's actually advisable in this exercise.

```
struct Point {
double x, y, z;
};

struct Particle {
Point p,v;
double m;
};

// Pre:
// Post: returns the sum of p1 and p2.
Point sum(Point p1, Point p2)
{
//...
}

// Pre:
// Post: returns a times p.
Point mul(double a, Point p)
{
//...
}
```

Note: We strongly advise you to start doing a simple implementation in order to overcome the public tests, and to try to optimise it later on, in order to overcome the private tests as well.

Input

The input has several cases. Each one starts with two positive natural numbers n, k on a first line. Next, n lines come, each describing the position (three integers), velocity (three integers) and mass (a positive natural) of a particle. Finally, k lines come, each with a positive natural that represents an elapsed time.

Output

For each case, at first k lines must be printed, where i'th line contains the mass center (three reals rounded to 5 digits after the decimal point) of all the particles after the sum of the first i elapsed times. Secondly, n lines must be printed, with the positions (three reals rounded to 5 digits after the decimal point) of the particles after the sum of all the elapsed times. Each case is followed by a blank line.

Sample input 1

```
2 2
0 0 0 0 0 1 1
1 0 0 0 0 1 1
1
1
2 2
0 0 0 1 0 0 1
1 0 0 1 0 0 1
```

```
1
1
2 2
0 0 0 1 0 0 1
1 0 0 0 0 1 1
1
1
```

Sample output 1

```
0.50000 0.00000 1.00000
0.50000 0.00000 2.00000
0.00000 0.00000 2.00000
1.00000 0.00000 2.00000

1.50000 0.00000 0.00000
2.50000 0.00000 0.00000
2.00000 0.00000 0.00000
3.00000 0.00000 0.00000

1.00000 0.00000 0.50000
1.50000 0.00000 1.00000
2.00000 0.00000 0.00000
1.00000 0.00000 2.00000
```

Sample input 2

```
4 2
1 -3 -4 -1 -5 1 5
-4 3 2 0 -2 2 2
4 5 -3 -5 5 3 3
-5 -1 1 -5 5 -2 1
5
3
3 4
2 4 2 3 -3 5 5
5 -1 -4 -5 5 -1 5
2 1 3 1 -3 -3 2
4
3
1
2
2 1
-1 -1 -2 1 4 5 5
-1 -4 -5 5 4 -1 5
3
5 5
1 4 2 -4 -1 4 4
4 -3 5 -2 2 1 1
4 3 4 -1 0 4 5
4 3 -2 3 2 0 2
-1 2 2 3 0 -4 2
1
2
2
1
5
5 5
-1 -3 -1 -2 5 -2 2
-3 5 3 5 0 -5 4
5 1 5 -2 -5 2 4
0 1 4 5 -1 4 4
-2 2 -3 3 2 -1 4
1
2
2
2
1
5 4
3 3 -5 2 -2 4 5
-4 0 3 -1 -5 1 2
-3 -3 4 1 2 4 3
-2 -2 3 3 3 4 1
-5 3 3 3 -1 1 4
2
4
5
2
5 4
-3 1 -2 -1 3 1 4
-4 -5 5 -5 -1 -5 5
5 2 2 -4 2 -3 1
-3 1 -3 -1 -4 0 5
3 5 -1 3 -2 1 5
4
2
3
4
5 4
-4 -4 -4 2 -5 3 1
-4 -3 -1 3 -1 5 2
3 4 -2 -3 1 -5 3
-3 -1 -5 -1 -2 3 2
-2 2 0 -3 4 -1 4
2
2
1
4
5 1
-1 0 -1 -3 -4 -5 4
-4 2 -1 1 4 1 5
-5 4 -4 0 -4 2 3
-3 -2 -3 -4 5 -4 4
1 -4 1 -5 -1 3 3
2
2 4
0 -3 -1 4 4 5 5
4 3 -3 -4 2 2 3
1
5
4
1
5 3
5 -2 3 -2 -4 -3 3
-3 0 1 -1 5 2 1
4 0 0 5 -3 -3 1
-4 2 2 4 1 -1 5
-4 -4 -4 -5 0 4 1
1
3
5
5 4
1 3 -2 -3 -2 5 5
3 4 2 3 2 2 5
-2 0 4 0 5 5 4
-5 4 5 4 5 -2 4
5 5 -4 -2 2 -3 2
```

```
4
5
1
2
4 3
-1 2 5 4 3 -3 4
-2 -4 -1 3 5 -5 4
2 -5 5 -1 3 2 1
-5 -3 0 -2 -5 2 3
5
3
5
4 5
-5 1 4 -2 -5 5 4
-2 0 -3 -2 -4 4 1
4 -5 4 1 -2 2 5
1 5 1 -2 4 -4 5
5
3
4
3
1
4 5
-1 -3 -4 0 2 -4 2
1 5 3 0 3 -1 2
-5 -4 -1 2 4 5 3
2 4 -1 0 3 -5 4
1
4
3
2
1
2 2
5 -4 -2 -2 4 4 1
2 -2 -1 0 -3 1 3
3
3
2 5
-5 4 4 3 3 3 5
5 1 5 -4 0 0 3
4
4
2
1
5
2 3
1 -5 -5 3 -1 4 3
3 -2 -5 3 -3 4 3
5
4
3
4 3
3 1 -1 -4 0 4 4
2 5 -5 -3 0 2 4
-2 -3 2 -4 2 -3 4
2 3 0 -2 -5 -3 3
1
3
2
3 1
-5 1 0 2 -4 5 2
```

```
-4 -5 2 1 -3 5 1
2 -4 -2 2 -1 -5 2
5
```

Sample output 2

```
-11.00000 -3.63636 5.09091
-17.81818 -6.09091 9.45455
-7.00000 -43.00000 4.00000
-4.00000 -13.00000 18.00000
-36.00000 45.00000 21.00000
-45.00000 39.00000 -15.00000

0.58333 2.75000 4.33333
-1.41667 3.75000 7.83333
-2.08333 4.08333 9.00000
-3.41667 4.75000 11.33333
32.00000 -26.00000 52.00000
-45.00000 49.00000 -14.00000
12.00000 -29.00000 -27.00000

8.00000 9.50000 2.50000
2.00000 11.00000 13.00000
14.00000 8.00000 -8.00000

1.64286 2.85714 4.42857
0.07143 3.14286 8.57143
-1.50000 3.42857 12.71429
-2.28571 3.57143 14.78571
-6.21429 4.28571 25.14286
-43.00000 -7.00000 46.00000
-18.00000 19.00000 16.00000
-7.00000 3.00000 48.00000
37.00000 25.00000 -2.00000
32.00000 2.00000 -42.00000

2.11111 1.33333 1.66667
6.55556 0.66667 1.22222
11.00000 0.00000 0.77778
15.44444 -0.66667 0.33333
17.66667 -1.00000 0.11111
-17.00000 37.00000 -17.00000
37.00000 5.00000 -37.00000
-11.00000 -39.00000 21.00000
40.00000 -7.00000 36.00000
22.00000 18.00000 -11.00000

1.86667 -0.93333 6.13333
8.80000 -4.93333 17.33333
17.46667 -9.93333 31.33333
20.93333 -11.93333 36.93333
29.00000 -23.00000 47.00000
-17.00000 -65.00000 16.00000
10.00000 23.00000 56.00000
37.00000 37.00000 55.00000
34.00000 -10.00000 16.00000

-5.95000 -3.65000 -3.85000
-8.25000 -5.75000 -5.75000
-11.70000 -8.90000 -8.60000
-16.30000 -13.10000 -12.40000
-16.00000 40.00000 11.00000
-69.00000 -18.00000 -60.00000
-47.00000 28.00000 -37.00000
-16.00000 -51.00000 -3.00000
42.00000 -21.00000 12.00000

-3.91667 2.00000 -1.83333
-6.41667 3.33333 -1.83333
-7.66667 4.00000 -1.83333
-12.66667 6.66667 -1.83333
14.00000 -49.00000 23.00000
23.00000 -12.00000 44.00000
-24.00000 13.00000 -47.00000
-12.00000 -19.00000 22.00000
-29.00000 38.00000 -9.00000

-6.52632 1.05263 -3.26316
-7.00000 -8.00000 -11.00000
-2.00000 10.00000 1.00000
-5.00000 -4.00000 0.00000
-11.00000 8.00000 -11.00000
-9.00000 -6.00000 7.00000

2.50000 2.50000 2.12500
7.50000 18.75000 21.50000
11.50000 31.75000 37.00000
12.50000 35.00000 40.87500
44.00000 41.00000 54.00000
-40.00000 25.00000 19.00000

0.45455 -0.45455 0.45455
4.00000 -1.81818 -2.54545
9.90909 -4.09091 -7.54545
-13.00000 -38.00000 -24.00000
-12.00000 45.00000 19.00000
49.00000 -27.00000 -27.00000
32.00000 11.00000 -7.00000
-49.00000 -4.00000 32.00000

2.50000 11.85000 9.60000
5.50000 22.85000 19.85000
6.10000 25.05000 21.90000
7.30000 29.45000 26.00000
-35.00000 -21.00000 58.00000
39.00000 28.00000 26.00000
-2.00000 60.00000 64.00000
43.00000 64.00000 -19.00000
-19.00000 29.00000 -40.00000

6.66667 6.50000 -8.25000
11.91667 11.50000 -14.25000
20.66667 19.83333 -24.25000
51.00000 41.00000 -34.00000
37.00000 61.00000 -66.00000
-11.00000 34.00000 31.00000
-31.00000 -68.00000 26.00000

-4.80000 -4.40000 7.20000
-7.80000 -7.20000 10.00000
-11.80000 -10.93333 13.73333
-14.80000 -13.73333 16.53333
-15.80000 -14.66667 17.46667
-37.00000 -79.00000 84.00000
-34.00000 -64.00000 61.00000
20.00000 -37.00000 36.00000
```

```
-31.00000 69.00000 -63.00000

-0.09091 3.81818 -2.18182
2.09091 16.18182 -7.63636
3.72727 25.45455 -11.72727
4.81818 31.63636 -14.45455
5.36364 34.72727 -15.81818
-1.00000 19.00000 -48.00000
1.00000 38.00000 -8.00000
17.00000 40.00000 54.00000
2.00000 37.00000 -56.00000

1.25000 -6.25000 4.00000
-0.25000 -10.00000 9.25000
-7.00000 20.00000 22.00000
2.00000 -20.00000 5.00000

0.25000 10.37500 11.87500
1.75000 17.87500 19.37500
2.50000 21.62500 23.12500
2.87500 23.50000 25.00000
4.75000 32.87500 34.37500
```

```
43.00000 52.00000 52.00000
-59.00000 1.00000 5.00000

17.00000 -13.50000 15.00000
29.00000 -21.50000 31.00000
38.00000 -27.50000 43.00000
37.00000 -17.00000 43.00000
39.00000 -38.00000 43.00000

-2.13333 0.93333 -0.86667
-12.13333 -0.46667 -0.26667
-18.80000 -1.40000 0.13333
-21.00000 1.00000 23.00000
-16.00000 5.00000 7.00000
-26.00000 9.00000 -16.00000
-10.00000 -27.00000 -18.00000

7.00000 -15.20000 4.60000
5.00000 -19.00000 25.00000
1.00000 -20.00000 27.00000
12.00000 -9.00000 -27.00000
```

Observation

Grading up to 10 points:

- Slow solution: 5 points.
- Fast solution: 10 points.

We understand as a fast solution one which is correct, with linear cost and which passes the public and private tests. We understand as slow solution one which is not fast, but it is correct and passes the public tests.

Problem information

Author: PRO1

Generation: 2026-01-25T15:02:21.951Z