1 Beginning — Pythagorean Triples

A Pythagorean Triple is a sequence of 3 integers $a$, $b$, and $c$, satisfying the following properties:

1. $0 < a < b < c$; and
2. $a^2 + b^2 = c^2$.

Write a program that takes 3 integers as input and reports whether they form a Pythagorean Triple.

Example 1:

Enter $a$: 3
Enter $b$: 4
Enter $c$: 5

A Pythagorean Triple.

Example 2:

Enter $a$: -3
Enter $b$: 4
Enter $c$: 5

Not a Pythagorean Triple.

Example 3:

Enter $a$: 4
Enter $b$: 3
Enter $c$: 5

Not a Pythagorean Triple.

Example 4:

Enter $a$: 4
Enter $b$: 5
Enter $c$: 6

Not a Pythagorean Triple.
Driving safety experts often claim that you need to leave “two seconds” of space between you and the car in front of you. This distance is the distance your car goes in two seconds.

You are developing a sensor for a “two second space” detector for an automobile. Given the distance (in feet) you are from the car in front of you and your speed (in mph), the sensor must warn the driver if there is less than a “two second space” between you and the car in front. If there is less than the two second space, printout the length required.

Note that all speeds are in miles per hour; all distances in feet. There are 5280 feet per mile.

Example 1:

Enter distance: 200
Enter your car’s speed: 60

Output:
No danger

Example 2:

Enter your car’s speed: 100
Enter distance: 40

Output:
Danger- driving too close. Two second distance is 292 feet.
A second-order homogeneous recurrence is a function defined as follows:

\[
f(n) = \begin{cases} 
  n_0 & \text{if } n = 0 \\
  n_1 & \text{if } n = 1 \\
  af(n-1) + bf(n-2) & \text{if } n > 1 
\end{cases}
\]

for some given values \(n_0, n_1, a,\) and \(b\). For example, if \(a = 3, b = -4, n_0 = 1,\) and \(n_1 = 2\), we have

- \(f(0) = n_0 = 1\)
- \(f(1) = n_1 = 2\)
- \(f(2) = 3 \times f(1) - 4 \times f(0) = 2\)
- \(f(3) = 3 \times f(2) - 4 \times f(1) = -2\)

and so on. Write a program that takes values for \(a, b, n_0, n_1,\) and \(n\), and prints the value of \(f(n)\). You may assume that all input values are integers.

**Example 1:**

Enter a: 3
Enter b: -4
Enter n0: 1
Enter n1: 2
Enter n: 3

\(f(n) = -2\)

**Example 2:**

Enter a: 1
Enter b: 1
Enter n0: 0
Enter n1: 1
Enter n: 6

\(f(n) = 8\)
The input will be a sequence of N coins (penny, nickel, dime, quarter, fifty-cents). You must put them into two piles are equally as possible. The sequence will be in decreasing order with duplicates.

Each coin in the sequence must be placed in the pile with the smaller value of the coins in the pile. For each coin you must output the value of the coin and in which pile it is placed. After the last coin is read, you must output the total of each pile.

The first number will be the number of coins.

Example 1:

Enter N: 6
Enter coin: 50 output: 50 on pile 1
Enter coin: 25 output: 25 on pile 2
Enter coin: 25 output: 25 on pile 2
Enter coin: 5 output: 5 on pile 1
Enter coin: 1 output: 1 on pile 2
Enter coin: 1 output: 1 on pile 2

Pile 1 is 55
Pile 2 is 52
This problem involves the step-by-step simulation of two particles in a 1-dimensional universe with rather unusual physical properties. At any given step, each particle occupies a position, which is an integer. Both particles may occupy the same position. Each particle travels at a given speed of $1/s$, where $s$ is a positive integer that may be different for the two particles. Thus, a particle with speed $1/s$ will move one position up or down (depending on its current direction) after every $s$ steps; it will maintain the same position for the intervening $s-1$ steps. At any given time, a particle has a current direction of either 1 (indicating up) or -1 (indicating down). After each step, if the two particles occupy the same position, both of their directions are reversed. The following table shows a simulation of 2 particles for 4 steps:

<table>
<thead>
<tr>
<th>step</th>
<th>pos.</th>
<th>dir.</th>
<th>speed</th>
<th>pos.</th>
<th>dir.</th>
<th>speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1/2</td>
<td>2</td>
<td>-1</td>
<td>1/3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1/2</td>
<td>2</td>
<td>-1</td>
<td>1/3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1/2</td>
<td>2</td>
<td>-1</td>
<td>1/3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-1</td>
<td>1/2</td>
<td>1</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-1</td>
<td>1/2</td>
<td>1</td>
<td>1</td>
<td>1/3</td>
</tr>
</tbody>
</table>

Write a program that takes as input the initial position, the initial direction, and the reciprocal of the speed of both particles, plus the number of steps of the simulation. It should then produce as output the final positions of both particles. You may assume that all inputs are integers, and that reciprocals of the speeds are positive.

Example:

Enter initial position of p1: 0
Enter initial position of p2: 2
Enter initial direction of p1: 1
Enter initial direction of p2: -1
Enter reciprocal of speed of p1: 2
Enter reciprocal of speed of p2: 3
Enter number of steps: 4

Final position of p1: 0
Final position of p2: 1
Beginning 6 Average Temperature

Usually the average temperature for a day is calculated by just averaging the max and the min temperature. It is more accurate to use additional data to get an average that reflects more of the hourly temperatures that occurred during the day.

Your program will accept the temperature at midnight at the start of the day, the minimum temperature and the hour at which it occurred, the maximum temperature and the hour at which it occurred and the temperature at midnight at the end of the day. Assume that the minimum temperature occurs before the maximum temperature.

All hours are on the 24-hour clock. No minutes are used in the input. Assume that the temperature does a linear change between readings. For example, the average temp between the midnight temp and the low is equal to the sum of the midnight temp and the min temp divided by two. The average for the whole day is not just the average of the values. It must be weighted by the length of the intervals. Thus if the average of the first 16 hours was 60 and the average of the last 8 hours was 30, the average for the whole day would be \((16/24)*60 + (8/24)*30 = 50\).

Example 1:

Enter midnight temp: 50
Enter the low temp: 30
Enter the time of the low temp: 8
Enter the max temp: 70
Enter the time of the max temp: 16
Enter the last midnight temp: 50

Average temperature is 50

Example 2:

Enter midnight temp: 50
Enter the low temp: 30
Enter the time of the low temp: 7
Enter the max temp: 70
Enter the time of the max temp: 15
Enter the last midnight temp: 50

Average temperature is 50.83
1 Beginning — Pythagorean Triples

Test Cases

Test Case 1:

Enter a: 20
Enter b: 21
Enter c: 29

A Pythagorean Triple.

Test Case 2:

Enter a: 21
Enter b: 20
Enter c: 29

Not a Pythagorean Triple.

Test Case 3:

Enter a: -21
Enter b: -20
Enter c: 29

Not a Pythagorean Triple.

Retest: Do the above tests, plus the following:

Test Case 4:

Enter a: 9
Enter b: 40
Enter c: 41

A Pythagorean Triple.

Test Case 5:

Enter a: 0
Enter b: 2
Enter c: 2

Not a Pythagorean Triple.
Beginning 2 Collision Sensor

First set of test cases. The numbers should be within .1 of those given

Test Case 1:
Enter your car’s speed: 100
Enter distance: 100

Output:
Danger – driving too close. You need 292 feet.

Test Case 2:
Enter your car’s speed: 50
Enter distance: 150

Output:
No Danger

Test Case 3:
Enter your car’s speed: 10
Enter distance: 50

Output:
No Danger

For retesting, repeat the above and do the additional tests below.

Test Case 4:
Enter your car’s speed: 60
Enter distance: 200

Output:
No danger

Test Case 5:
Enter your car’s speed: 60
Enter distance: 140

Output:
Danger – driving too close. You need 176 feet.
3 Beginning — Recurrences
Test Cases

Test Case 1:

Enter a: 2
Enter b: -1
Enter n0: 2
Enter n1: 4
Enter n: 6

f(n) = 14

Test Case 2:

Enter a: 3
Enter b: 2
Enter n0: 1
Enter n1: 2
Enter n: 7

f(n) = 4516

Retest: Do the above tests. plus the following:

Test Case 3:

Enter a: 0
Enter b: 2
Enter n0: 1
Enter n1: 2
Enter n: 8

f(n) = 16
Tests for Beginning 4 Coin Piles

Test 1:
Enter N: 6
Enter coin: 50 output: 50 on pile 1
Enter coin: 25 output: 25 on pile 2
Enter coin: 25 output: 25 on pile 2
Enter coin: 5 output: 5 on pile 1
Enter coin: 1 output: 1 on pile 2
Enter coin: 1 output: 1 on pile 2
Pile 1 is 55 Pile 2 is 52

Test 2:
Enter N: 7
Enter coin: 50 output: 50 on pile 1
Enter coin: 25 output: 25 on pile 2
Enter coin: 10 output: 10 on pile 2
Enter coin: 10 output: 10 on pile 2
Enter coin: 10 output: 10 on pile 2
Enter coin: 1 output: 1 on pile 1
Enter coin: 1 output: 1 on pile 1
Pile 1 is 52 Pile 2 is 55

For retesting, redo the tests above and do the test below:
Test 3:
Enter N: 3
Enter coin: 1 output: 1 on pile 1
Enter coin: 1 output: 1 on pile 2
Enter coin: 1 output: 1 on pile 1
Pile 1 is 2 Pile 2 is 1
Test Case 1:

Enter initial position of p1: 1000
Enter initial position of p2: 1001
Enter initial direction of p1: 1
Enter initial direction of p2: -1
Enter reciprocal of speed of p1: 1
Enter reciprocal of speed of p2: 2
Enter number of steps: 5

Final position of p1: 997
Final position of p2: 1003

Retest: Do the above test, plus the following:

Enter initial position of p1: -10000
Enter initial position of p2: -10003
Enter initial direction of p1: -1
Enter initial direction of p2: 1
Enter reciprocal of speed of p1: 2
Enter reciprocal of speed of p2: 1
Enter number of steps: 6

Final position of p1: -9999
Final position of p2: -10005
Tests for Beginning 6 - Average Temperature

Test 1:
Enter midnight temp: 50
Enter the low temp: 30
Enter the time of the low temp: 8
Enter the max temp: 70
Enter the time of the max temp: 16
Enter the last midnight temp: 50
Average temperature is 50

Test 2:
Enter midnight temp: 10
Enter the low temp: -5
Enter the time of the low temp: 2
Enter the max temp: 8
Enter the time of the max temp: 8
Enter the last midnight temp: 10
Average temperature is 6.58

For retesting, repeat the tests above and do the additional tests below

Test 3:
Enter midnight temp: -10
Enter the low temp: -20
Enter the time of the low temp: 6
Enter the max temp: -5
Enter the time of the max temp: 21
Enter the last midnight temp: -8
Average temperature is -12.37

Test 4:
Enter midnight temp: 50
Enter the low temp: 30
Enter the time of the low temp: 7
Enter the max temp: 70
Enter the time of the max temp: 15
Enter the last midnight temp: 50
Average temperature is 50.83

Test 5:
Enter midnight temp: 0
Enter the low temp: 0
Enter the time of the low temp: 6
Enter the max temp: 0
Enter the time of the max temp: 18
Enter the last midnight temp: 0
Average temperature is 0
/* 1 Beginning - Pythagorean Triples */
* Pythagorean1.java */
import java.io.*;

public class Pythagorean1 {
    public static void main(String[] args) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter a:");
        int a = Integer.parseInt(in.readLine());
        System.out.print("Enter b:");
        int b = Integer.parseInt(in.readLine());
        System.out.print("Enter c:");
        int c = Integer.parseInt(in.readLine());
        System.out.println();
        if (0 < a && a < b && b < c && a * a + b * b == c * c)
            System.out.println("A Pythagorean Triple.");
        else System.out.println("Not a Pythagorean Triple.");
    }
}
#include <iostream>

using std::cin;
using std::cout;
using std::endl;

int _tmain(int argc, TCHAR* argv[])
{
    float yourspeed;
    float otherspeed;
    float actualdistance;
    float distancebeforeimpact;
    bool closingtoofast;
    bool drivingtooclose;
    float impacttime;
    char q;

    closingtoofast=false;
    drivingtooclose=false;

    cout << "Enter your car's speed: ";
    cin >> yourspeed;
    cout << "Enter other car's speed: ";
    cin >> otherspeed;
    cout << "Enter distance: ";
    cin >> actualdistance;
    cout << yourspeed << " " << otherspeed << " " << actualdistance;
    if(yourspeed > otherspeed)
    {
        impacttime = actualdistance/((yourspeed - otherspeed)*5260*3600);
        cout << "\n impacttime " << impacttime;
        if(impacttime <= 120)
        {
            cout << "\n Danger - impact time is " << impacttime;
            closingtoofast=true;
        }
    }
    if(actualdistance <= 2*(5260*yourspeed)/3600)
    {
        cout << "\n Danger - driving too close \n You need ";
        cout << 2*5260/(3600*yourspeed) << " feet";
        drivingtooclose=true;
    }
    if(drivingtooclose==false & closingtoofast==false)
    {
        cout << "\n No Danger";
    }
    return 0;
}
import java.io.*;

public class Recurrence2 {

   public static void main(String[] args) throws Exception {
      BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
      System.out.print("Enter a:");
      int a = Integer.parseInt(in.readLine());
      System.out.print("Enter b:" );
      int b = Integer.parseInt(in.readLine());
      System.out.print("Enter n0:");
      int fprev = Integer.parseInt(in.readLine());
      System.out.print("Enter n:");
      int fn = Integer.parseInt(in.readLine());
      System.out.println();
      if (n == 0) System.out.println("fn= " + fprev);
      else {
         for (int i = 2; i <= n; i++) {
            int fnext = a*fn + b*fprev;
            fprev = fn;
            fn = fnext;
         }
         System.out.println("fn= " + fn);
      }
   }
}
B4 code – coin piles

#include <iostream>
#define MAX 10

using std::cin;
using std::cout;
using std::endl;

int _tmain(int argc, _TCHAR* argv[])
{
    int n;
    int sum1;
    int sum2;
    int coin;

    sum1=0; sum2=0;
    cout << "\n enter the number of coins: ";
    cin >> n;
    int i; i=0;
    while (i<n)
    {
        cout << "\n enter the value of the coin: ";
        cin >> coin;
        cout<< "\n coin "<< coin << " and is added to 
 stack ";
        if(sum1 <= sum2)
        {
            cout<< "1":
            sum1 = sum1 + coin;
        }
        else
        {
            cout<< "2":
            sum2 = sum2 + coin;
        }
        i=i+1;
    }
    cout << "\n the total of pile 1 is " << sum1;
    cout << "\n the total of pile 2 is " << sum2;

    return 0;
}
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Particles2 {

    public static void main(String[] args) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter initial position of p1: ");
        int position1 = Integer.parseInt(in.readLine());
        System.out.print("Enter initial position of p2: ");
        int position2 = Integer.parseInt(in.readLine());
        System.out.print("Enter initial direction of p1: ");
        int direction1 = Integer.parseInt(in.readLine());
        System.out.print("Enter initial direction of p2: ");
        int direction2 = Integer.parseInt(in.readLine());
        System.out.print("Enter reciprocal of speed of p1: ");
        int speed1 = Integer.parseInt(in.readLine());
        int timeLeft1 = speed1;
        System.out.print("Enter reciprocal of speed of p2: ");
        int speed2 = Integer.parseInt(in.readLine());
        int timeLeft2 = speed2;
        System.out.print("Enter number of steps: ");
        int time = Integer.parseInt(in.readLine());
        for (int i = 0; i < time; i++) {
            if (--timeLeft1 == 0) {
                timeLeft1 = speed1;
                position1 += direction1;
            }
            if (--timeLeft2 == 0) {
                timeLeft2 = speed2;
                position2 += direction2;
            }
            if (position1 == position2) {
                direction1 *= -1;
                direction2 *= -1;
            }
        }
        System.out.println();
        System.out.println("Final position of p1: "+ position1);
        System.out.println("Final position of p2: "+ position2);
    }
}
Code for B6

#include <iostream>

using std::cin;
using std::cout;
using std::endl;

int _tmain(int argc, _TCHAR* argv[])
{
    int midnightprior;
    int timeoflow;
    int low;
    int timeofhigh;
    int high;
    int midnightpost;
    float ave;

    cout<< "Enter temp at midnight before:"
    cin>>midnightprior;
    cout<< "Enter temp at low temperature:"
    cin>>low;
    cout<< "Enter time of low temperature:"
    cin*timeoflow;
    cout<< "Enter temp at high temperature:"
    cin>>high;
    cout<< "Enter time of high temperature:"
    cin*timeofhigh;
    cout<< "Enter temp at midnight after:"
    cin>>midnightpost;

    ave = (timeoflow/24.0)*(midnightprior+low)/2.0 + (timeofhigh-
    timeoflow)/24.0* (low+high)/2.0 + (24-
    timeofhigh)/24.0*(high+midnightpost)/2.0;

    cout<<"The average temperature is " << ave;

    return 0;
}
1 Advanced — Vector Multiplication

Given an $m$-element column vector $a$ and an $n$-element row vector $b$, the product $ab$ is the $m \times n$ matrix $c$ such that the element in row $i$ and column $j$ of $c$ is the product of the $i$th element of $a$ and the $j$th element of $b$: i.e.,

$$c_{ij} = a_i b_j.$$

Write a program that takes positive integer values $m$ and $n$, followed by the $m$ integer values of $a$ and the $n$ integer values of $b$, and prints the product $ab$. Your matrix must be printed by rows. The the columns do not need to be lined up, but the elements in each row must be separated by spaces. You may assume that $1 \leq m \leq 10$ and $1 \leq n \leq 10$.

**Example 1:**

Enter $m$: 5
Enter $n$: 4
Enter $a[1]$: 1
Enter $a[2]$: 2
Enter $a[3]$: 3
Enter $a[4]$: 4
Enter $a[5]$: 5
Enter $b[1]$: 4
Enter $b[2]$: 3
Enter $b[3]$: 2
Enter $b[4]$: 1

The product is:

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>12</td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>15</td>
<td>10</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Example 2:**

Enter $m$: 2
Enter $n$: 5
Enter $a[1]$: -3
Enter $a[2]$: 4
Enter $b[1]$: 5
Enter $b[2]$: -6
Enter $b[3]$: 0
Enter $b[4]$: 7
Enter $b[5]$: 2

The product is:

-15 18 0 -21 -6
20 -24 0 28 8
You are developing a sensor for an on-board collision detector for an automobile. Given the distance (in feet) you are behind the car in front of you, your speed (in mph) and the speed of the car ahead (in mph), the sensor must warn the driver about two conditions: 1) is there less than "two seconds of distance" between you and the car in front of you and 2) is there less than 30 seconds before a potential impact with the car in front based on the difference in speeds between you and the car in front of you. Print out the values if either danger condition exists.

"Two second of distance" is the distance a car goes in two seconds.

All speeds are in miles per hour and the distances are in feet. There are 5280 feet per mile.

Example 1:

Enter your car's speed: 100
Enter speed of car in front: 40
Enter distance: 100

Output:
Danger – Potential impact time is 1.13 sec
Danger – driving too close. You need 292 feet.

Example 2:

Enter your car’s speed: 60
Enter speed of car in front: 60
Enter distance: 200

Output:
No danger
Suppose we have a square piece of metal divided into a $10 \times 10$ grid. We apply a different constant temperature to each side of the square -- specifically, to the 8 grid cells of each side, excluding the corner cells (see figure below). We can then find the temperature of each internal cell by initializing all internal cells to some arbitrary value and repeatedly setting each cell's temperature to the average of the temperatures of its four neighbors (above, below, left, and right). Eventually, each cell's temperature will converge to the correct value. Note that the corners of the grid are ignored, and that the value for each iteration is computed using values from the previous iteration (thus, 2 arrays are required — one to store the values computed in the previous iteration, and one to store the new values computed in the current iteration).

Write a program that takes as input the temperatures of the four edges, an initial internal temperature, a number of iterations, and the coordinates of some interior grid cell, and produces as output the temperature of the specified cell after the given number of iterations. You may assume that all temperatures are floating-point numbers, and that the remaining inputs are positive integers. The coordinates of the cells are specified such that the upper-left corner is $(0,0)$, and the lower-right corner is $(9,9)$.

Example 1:

Enter left edge temperature: 61.73
Enter top edge temperature: 43.29
Enter right edge temperature: 87.37
Enter bottom edge temperature: 72.32
Enter initial interior value: 0
Enter number of iterations: 1000
Enter x-coordinate: 1
Enter y-coordinate: 2

The final temperature is: 50.411148

Example 2:

Enter left edge temperature: 51.2
Enter top edge temperature: 78.34
Enter right edge temperature: 64.9
Enter bottom edge temperature: 48.7
Enter initial interior value: 32.0
Enter number of iterations: 1
Enter x-coordinate: 4
Enter y-coordinate: 5

The final temperature is: 32.0
Advanced 4

Number Bins

The input will be a decreasing sequence of N numbers (< 100). You must put them into M bins are equally as possible. The sequence will be in decreasing order with duplicates.

Each number in the sequence must be placed in the bin containing the smallest sum (<10 bins). For each number you must output the number and in which bin it is placed. After the last number is read, you must output the total of each bin.

The first number is N. The second number is M.

Example 1:

Enter N: 7
Enter M: 3
Enter number: 20 output: 20 in bin 1
Enter number: 20 output: 20 in bin 2
Enter number: 15 output: 15 in bin 3
Enter number: 10 output: 10 in bin 3
Enter number: 9 output: 9 in bin 1
Enter number: 6 output: 6 in bin 2
Enter number: 5 output: 5 in bin 3

Bin 1 is 29
Bin 2 is 26
Bin 3 is 30
This problem involves the step-by-step simulation of a small number of particles in a 1-dimensional universe with rather unusual physical properties. At any given step, each particle occupies a position, which is an integer. More than one particle may occupy the same position. Each particle travels at a given speed of $1/s$, where $s$ is a positive integer that may be different for different particles. Thus, a particle with speed $1/s$ will move one position up or down (depending on its current direction) after every $s$ steps; it will maintain the same position for the intervening $s - 1$ steps. At any given time, a particle has a current direction of either $1$ (indicating up) or $-1$ (indicating down). After each step, each particle that shares a position with one or more other particles has its direction reversed. The following table shows a simulation of 3 particles for 4 steps:

<table>
<thead>
<tr>
<th>step</th>
<th>pos.</th>
<th>dir.</th>
<th>speed</th>
<th>pos.</th>
<th>dir.</th>
<th>speed</th>
<th>pos.</th>
<th>dir.</th>
<th>speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1/2</td>
<td>1</td>
<td>1</td>
<td>1/3</td>
<td>5</td>
<td>-1</td>
<td>1/1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1/2</td>
<td>1</td>
<td>1</td>
<td>1/3</td>
<td>4</td>
<td>-1</td>
<td>1/1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>1/2</td>
<td>1</td>
<td>-1</td>
<td>1/3</td>
<td>3</td>
<td>-1</td>
<td>1/1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-1</td>
<td>1/2</td>
<td>0</td>
<td>-1</td>
<td>1/3</td>
<td>2</td>
<td>-1</td>
<td>1/1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1/2</td>
<td>0</td>
<td>1</td>
<td>1/3</td>
<td>1</td>
<td>-1</td>
<td>1/1</td>
</tr>
</tbody>
</table>

Write a program that takes as input:

- a number of particles;
- for each particle:
  - its initial position;
  - its initial direction ($1$ or $-1$);
  - and the reciprocal of its initial speed;
- and the number of steps of the simulation.

It should then produce as output the final positions of all of the particles. You may assume that all inputs are integers, and that reciprocals of the speeds are positive. You may also assume that the number of particles is at least 1 and at most 10.

Example:

Enter number of particles: 3
Enter initial position of p0: 0
Enter initial direction of p0: 1
Enter reciprocal of speed of p0: 2
Enter initial position of p1: 1
Enter initial direction of p1: 1
Enter reciprocal of speed of p1: 3
Enter initial position of p2: 5
Enter initial direction of p2: -1
Enter reciprocal of speed of p2: 5
Enter number of steps: 4

Final position of p0: 0
Final position of p1: 0
Final position of p2: 1
Usually the average temperature for a day is calculated by just averaging the max and the min temperature. It is more accurate to use additional data to get an average that reflects the hourly temperatures that occurred during the day.

Your program will accept the temperature at midnight at the start of the day. Next it will accept the number of temperatures recorded (not including the two midnight temperatures), the temperatures and the hours and minutes at which they occurred, and finally the temperature at midnight at the end of the day.

All hours are on the 24-hour clock. Minutes are used in the input. Assume that the temperature does a linear change between readings. Thus, the average temp between the midnight temp and the next temperature is equal to the sum of the midnight temp and the next temperature divided by two. The average for the whole day is not just the average of the values. It must be weighted by the length of the intervals. Thus if the average of the first 16 hours was 60 and the average of the last 8 hours was 30, the average for the whole day would be \((16/24)*60 + (8/24)*30 = 50\).

Example 1:

Enter midnight temp: 50
Enter the number of temperatures: 1
Enter a temp: 30
Enter the hour of the temp: 12
Enter the minute of the temp: 0
Enter the last midnight temp: 50

Average temperature is 40

Example 2:

Enter midnight temp: 50
Enter the number of temperatures: 2
Enter a temp: 30
Enter the hour of the temp: 7
Enter the minute of the temp: 30
Enter a temp: 70
Enter the hour of the temp: 15
Enter the minute of the temp: 50
Enter the last midnight temp: 50

Average temperature is 58.69
1 Advanced — Vector Multiplication
Test Cases

Test Case 1:

Enter m: 3
Enter n: 10
Enter a[1]: 12
Enter a[2]: -2
Enter a[3]: 7
Enter b[1]: 6
Enter b[2]: -5
Enter b[3]: 1
Enter b[4]: 14
Enter b[5]: -8
Enter b[6]: -9
Enter b[7]: 3
Enter b[8]: 10
Enter b[9]: 4
Enter b[10]: -2

The product is:

72 -60 12 168 -96 -108 36 120 48 -24
-12 10 -2 -28 16 18 -6 -20 -8 4
42 -35 7 98 -56 -63 21 70 28 -14

Test Case 2:

Enter m: 1
Enter n: 1
Enter a[1]: 4
Enter b[1]: -3

The product is:

-12

Retest: Do the above tests, plus the following:

Test Case 3:

Enter m: 3
Enter n: 2
Enter a[1]: -2
Enter a[2]: 0
Enter a[3]: 7
Enter b[1]: 5
Enter b[2]: -6

The product is:

-10 12
0 0
35 -42
First set of test cases. The numbers should be within .1 of those given

**Test Case 1:**
Enter your car's speed: 100
Enter speed of car in front: 40
Enter distance: 100

Output:
Danger – Potential impact time is 1.13 sec
Danger – driving too close. You need 292 feet.

**Test Case 2:**
Enter your car's speed: 50
Enter speed of car in front: 45
Enter distance: 150

Output:
Danger – Potential impact time is 20.5 sec

**Test Case 3:**
Enter your car's speed: 10
Enter speed of car in front: 10
Enter distance: 50

Output:
No Danger

For retesting, repeat the above and do the additional tests below.

**Test Case 4:**
Enter your car's speed: 60
Enter speed of car in front: 60
Enter distance: 200

Output:
No danger

**Test Case 5:**
Enter your car's speed: 60
Enter speed of car in front: 59
Enter distance: 140

Output:
Danger – driving too close. You need 176 feet.
Test Cases

Note: Results should be accurate to 3 significant digits.

Test Case 1:

Enter left edge temperature: 55.5
Enter top edge temperature: 55.5
Enter right edge temperature: 55.5
Enter bottom edge temperature: 55.5
Enter initial interior value: 0
Enter number of iterations: 1
Enter x-coordinate: 2
Enter y-coordinate: 2

The final temperature is: 0.0

Test Case 2:

Enter left edge temperature: 40.0
Enter top edge temperature: 40.0
Enter right edge temperature: 80.0
Enter bottom edge temperature: 80.0
Enter initial interior value: 60.0
Enter number of iterations: 100
Enter x-coordinate: 8
Enter y-coordinate: 8

The final temperature is: 78.90326

Retest: Do the above tests, plus the following:

Test Case 3:

Enter left edge temperature: 12.3
Enter top edge temperature: 45.6
Enter right edge temperature: 78.9
Enter bottom edge temperature: 98.7
Enter initial interior value: 65.4
Enter number of iterations: 100
Enter x-coordinate: 5
Enter y-coordinate: 6

The final temperature is: 69.0748
Tests for Advanced 4 - Number Bins

**Test 1:**
Enter N: 7  Enter M: 3
Enter number: 20  output: 20 in bin 1
Enter number: 20  output: 20 in bin 2
Enter number: 15  output: 15 in bin 3
Enter number: 10  output: 10 in bin 3
Enter number: 9   output: 9 in bin 1
Enter number: 6   output: 6 in bin 2
Enter number: 5   output: 5 in bin 3
Bin 1 is 29  Bin 2 is 26  Bin 3 is 30

**Test 2:**
Enter N: 5  Enter M: 3
Enter number: 10  output: 10 in bin 1
Enter number: 8   output: 8 in bin 2
Enter number: 5   output: 5 in bin 3
Enter number: 5   output: 5 in bin 3
Enter number: 1   output: 1 in bin 2
Bin 1 is 10  Bin 2 is 9  Bin 3 is 10

**Test 3:**
Enter N: 9  Enter M: 7
Enter number: 5   output: 5 in bin 1
Enter number: 5   output: 5 in bin 2
Enter number: 5   output: 5 in bin 3
Enter number: 5   output: 5 in bin 4
Enter number: 5   output: 5 in bin 5
Enter number: 4   output: 4 in bin 6
Enter number: 3   output: 3 in bin 7
Enter number: 3   output: 3 in bin 7
Enter number: 3   output: 3 in bin 6
Bin 1 is 5  Bin 2 is 5  Bin 3 is 5  Bin 1 is 5  Bin 2 is 5  Bin 3 is 7  Bin 7 is 6

For retesting, repeat the above tests and do the additional test below

**Test 4:**
Enter N: 4  Enter M: 1
Enter number: 3   output: 3 in bin 1
Enter number: 3   output: 3 in bin 1
Enter number: 3   output: 3 in bin 1
Enter number: 3   output: 3 in bin 1
Bin 1 is 12
5 Advanced — Particle Simulation
Test Cases

Test Case 1:

Enter number of particles: 4
Enter initial position of p0: 1000
Enter initial direction of p0: 1
Enter reciprocal of speed of p0: 2
Enter initial position of p1: 1003
Enter initial direction of p1: -1
Enter reciprocal of speed of p1: 4
Enter initial position of p2: 999
Enter initial direction of p2: -1
Enter reciprocal of speed of p2: 1
Enter initial position of p3: 1005
Enter initial direction of p3: -1
Enter reciprocal of speed of p3: 5
Enter number of steps: 25

Final position of p0: 1012
Final position of p1: 999
Final position of p2: 995
Final position of p3: 1004

Retest: Do the above test, plus the following.

Test Case 2:

Enter number of particles: 3
Enter initial position of p0: 0
Enter initial direction of p0: -1
Enter reciprocal of speed of p0: 1
Enter initial position of p1: -5
Enter initial direction of p1: 1
Enter reciprocal of speed of p1: 2
Enter initial position of p2: 1
Enter initial direction of p2: -1
Enter reciprocal of speed of p2: 2
Enter number of steps: 10

Final position of p0: -10
Final position of p1: 0
Final position of p2: -4
Tests for Advanced 6

**Test 1:**
Enter midnight temp: 50
Enter the number of temperatures: 1
Enter a temp: 30
Enter the hour of the temp: 12
Enter the minute of the temp: 0
Enter the last midnight temp: 50
Average temperature is 40

**Test 2:**
Enter midnight temp: 32
Enter the number of temperatures: 0
Enter the last midnight temp: 52
Average temperature is 42

**Test 3:**
Enter midnight temp: 32
Enter the number of temperatures: 5

**For retesting, redo the tests above and do the additional tests below:**

**Test 4:**
Enter midnight temp: -10
Enter the number of temperatures: 1
Enter a temp: -20
Enter the hour of the temp: 4
Enter the minute of the temp: 10
Enter the last midnight temp: -30
Average temperature is -23.27

**Test 5:**
Enter midnight temp: 50
Enter the number of temperatures: 2
Enter a temp: 30
Enter the hour of the temp: 7
Enter the minute of the temp: 30
Enter a temp: 70
Enter the hour of the temp: 15
Enter the minute of the temp: 50
Enter the last midnight temp: 50
Average temperature is 58.69
import java.io.*;

public class VectorMult {

    public static void main(String[] args) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter m:");
        int m = Integer.parseInt(in.readLine());
        System.out.print("Enter n: ");
        int n = Integer.parseInt(in.readLine());
        int[] a = new int[m];
        for (int i = 0; i < m; i++) {
            System.out.print("Enter a[" + (i+1) + "]:");
            a[i] = Integer.parseInt(in.readLine());
        }
        int[] b = new int[n];
        for (int j = 0; j < n; j++) {
            System.out.print("Enter b[" + (j+1) + "]:");
            b[j] = Integer.parseInt(in.readLine());
        }
        System.out.println();
        System.out.println("The product is:");
        System.out.println();
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print((a[i]*b[j]) + " ");
            }
            System.out.println();
        }
    }
}
A2 – Driving Problem

#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int _tmain(int argc, _TCHAR* argv[])
{
    float yourspeed;
    float otherspeed;
    float actualdistance;
    bool closingtoofast;
    bool drivingtooclose;
    float impacttime;

    closingtoofast=false;
    drivingtooclose=false;

    cout << "Enter your car's speed: ";
    cin >> yourspeed;
    cout << "Enter other car's speed: ";
    cin >> otherspeed;
    cout << "Enter distance: ";
    cin >> actualdistance;
    if(yourspeed > otherspeed)
    {
        impacttime = (actualdistance*3600) / ((yourspeed - otherspeed)*5280);
        cout << "impacttime " << impacttime;
        if (impacttime <= 30)
        {
            cout << " Danger - impact time is " << impacttime;
            closingtoofast=true;
        }
    }

    if(actualdistance <= 2*(5280*yourspeed)/3600)
    {
        cout << " Danger - driving too close You need ";
        cout << 2*5280*yourspeed/(3600) << " feet";
        drivingtooclose=true;
    }

    if(drivingtooclose==false && closingtoofast==false)
    {
        cout << " No Danger";
    }

    return 0;
}
/* Advanced - Temperature Simulation
 * Grid.java Oct 26, 2006
 */

import java.io.*;

public class Grid {

    public static void main(String[] args) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter left edge temperature: ");
        float left = Float.parseFloat(in.readLine());
        System.out.print("Enter top edge temperature: ");
        float top = Float.parseFloat(in.readLine());
        System.out.print("Enter right edge temperature: ");
        float right = Float.parseFloat(in.readLine());
        System.out.print("Enter bottom edge temperature: ");
        float bot = Float.parseFloat(in.readLine());
        System.out.print("Enter initial interior value: ");
        float init = Float.parseFloat(in.readLine());
        System.out.print("Enter number of iterations: ");
        int iter = Integer.parseInt(in.readLine());
        float[][] grid = new float[10][10];
        for (int i = 1; i < 9; i++) {
            grid[0][i] = top;
            grid[i][0] = left;
            grid[9][i] = bot;
            grid[i][9] = right;
            for (int j = 1; j < 9; j++) {
                grid[i][j] = init;
            }
        }
        for (int k = 0; k < iter; k++) {
            float[][] newgrid = new float[10][10];
            for (int i = 1; i < 9; i++) {
                newgrid[0][i] = top;
                newgrid[i][0] = left;
                newgrid[9][i] = bot;
                newgrid[i][9] = right;
                for (int j = 1; j < 9; j++) {
                    newgrid[i][j] =
                        (grid[i-1][j] + grid[i+1][j] + grid[i][j-1] + grid[i][j+1])/4;
                }
            }
            grid = newgrid;
        }
        System.out.print("Enter x-coordinate: ");
        int x = Integer.parseInt(in.readLine());
        System.out.print("Enter y-coordinate: ");
        int y = Integer.parseInt(in.readLine());
        System.out.println();
        System.out.println("The final temperature is: "+ grid[x][y]);
    }
}
A4code – coin bin problem

#include <iostream>

using std::cin;
using std::cout;
using std::endl;
#define MAXPILES 10
#define MAXVALUE 20

int _tmain(int argc, _TCHAR* argv[])
{
    int numcoins;
    int numpiles;
    int sums[MAXPILES];
    int i;
    for (i=0;i<MAXPILES;i++) {sums[i]=0;}
    int minnumber;
    int minvalue;
    int coin;

    cout << endl << "enter the number of coins: ";
    cin >> numcoins;
    cout << endl << "enter the number of piles: ";
    cin >> numpiles;
    i=0;
    while (i<numcoins)
    {
        cout << endl << "enter the value of the coin: ";
        cin >> coin;
        cout << endl << "coin " << i << " is " << coin << " and is added to stack ";
        minvalue=MAXVALUE;
        int j;
        for(j=0;j<numpiles;j++)
        {
            if(sums[j]<minvalue)
            {
                minnumber=j;
                minvalue=sums[j];
            }
        }
        sums[minnumber]=sums[minnumber]+coin;
        cout<<minnumber+1;
        i=i+1;
    }
    int h;
    for(h=0;h<numpiles;h++)
    {
        cout << endl << "the total of pile " << h+1 << " is " << sums[h];
    }
    return 0;
}
/* 5 Advanced - Particle Simulation
  *
  * Particles.java Oct 25, 2006
  *
  */

import java.io.*;

public class Particles {

  public static void main(String[] args) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("Enter number of particles: ");
    int n = Integer.parseInt(in.readLine());
    int[] positions = new int[n];
    int[] directions = new int[n];
    int[] speeds = new int[n];
    int[] timeLeft = new int[n];
    for (int i = 0; i < n; i++) {
      System.out.print("Enter initial position of p" + i + ": ");
      positions[i] = Integer.parseInt(in.readLine());
      System.out.print("Enter initial direction of p" + i + ": ");
      directions[i] = Integer.parseInt(in.readLine());
      speeds[i] = Integer.parseInt(in.readLine());
    }
    System.out.print("Enter number of steps: ");
    int time = Integer.parseInt(in.readLine());
    for (int i = 0; i < time; i++) {
      for (int j = 0; j < n; j++) {
        if (--timeLeft[j] == 0) {
          timeLeft[j] = speeds[j];
          positions[j] += directions[j];
        }
      }
      for (int j = 0; j < n; j++) {
        boolean collision = false;
        for (int k = 0; k < n; k++) {
          if (j != k && positions[j] == positions[k]) {
            collision = true;
          }
        }
        if (collision) {
          directions[j] *= -1;
        }
      }
    }
    System.out.println();
    for (int i = 0; i < n; i++) {
      System.out.println("Final position of p" + i + ": " + positions[i]);
    }
  }
}
A6 code

#include <iostream>
#define MAX 10

using std::cin;
using std::cout;
using std::endl;

int _tmain(int argc, _TCHAR* argv[])
{
    int numoftemps;
    float newhour;
    float newmin;
    float newtemp;
    float oldtime;
    float oldtemp;
    float midnightpost;
    float ave;

    ave = 0;
    cout << "Enter the number of temps: (excluding midnights)";
    cin >> numoftemps;

    cout << "Enter temp at midnight before:"
    cin >> oldtemp;
    oldtime = 0;

    int i;
    for (i = 0; i < numoftemps; i++)
    {
        cout << "Enter hour of new temperature:"
        cin >> newhour;
        cout << "Enter minutes of new temperature:"
        cin >> newmin;
        cout << "Enter value of new temperature:"
        cin >> newtemp;
        ave = ave + ((newhour*60 + newmin - oldtime)/(24*60)) * (oldtemp + newtemp)/2;
        cout << ave;
        oldtime = newhour*60 + newmin;
        oldtemp = newtemp;
    }

    cout << "Enter temp at midnight after:"
    cin >> midnightpost;

    ave = ave + ((24*60 - oldtime)/(24*60)) * (oldtemp + midnightpost)/2;

    cout << "The average temperature is " << ave;

    return 0;
}