# Практически изпит (4 септември 2017 г.)

Практически упражнения към курса [**"Programming Fundamentals" за ученици**](https://github.com/BG-IT-Edu/School-Programming/tree/main/Courses/Applied-Programmer/Programming-Fundamentals).

Тествайте задачата в judge: [https://judge.softuni.bg/Contests/2673](https://judge.softuni.bg/Contests/2673/Практически-изпит-04-09-2017)

# Problem 1. Resurrection

You ever heard of Phoenixes? Magical Fire Birds that are practically immortal – they reincarnate from an egg when they die. Naturally, it takes time for them to reincarnate. You will play the role of a scientist who calculates the time to reincarnate for each phoenix, based on its body parameters.

You will receive **N**, an **integer** – the **amount** of **phoenixes**.   
For each **phoenix**, you will **receive 3 input lines**:

* On the **first input line** you will receive an **integer** – the **total length** of the **body** of the phoenix.
* On the **second input line** you will receive a **floating-point number** – the **total width** of the **body** of the phoenix.
* On the **third input line** you will receive an **integer** – the **length** of **1 wing** of the phoenix.

For each phoenix, you must **print** the **years** it will take for it to **reincarnate**, which is **calculated** by the following formula:

The totalLength **powered** by 2, **multiplied** by the **sum of the** totalWidth and the totalWingLength (2 \* wingLength).

totalYears = {totalLength} ^ 2 \* ({totalWidth} + 2 \* {wingLength})

### Input

* On the **first input line** you will receive **N**, an **integer** – the **amount** of **phoenixes**.
* On the **next** **N \* 3 input lines** you will be receiving **data** for **each phoenix**.

### Output

* As output, you must print the **total years needed for reincarnation** for each phoenix.
* Print each phoenix’s years **when you’ve calculated** them.
* Print each phoenix’s years **on a new line**.

### Constrains

* The **amount** of **phoenixes** will be an **integer** in **range [0, 1000]**.
* The **total length** of the **body** of the **phoenix** will be an **integer** in **range [-231, 231]**.
* The **total width** of the **body** of the **phoenix** will be a **floating-point number** in **range [-231, 231]**.
* The **total width** of the **body** of the **phoenix** will have up to **20 digits** after the **decimal point**.
* The **total length** of the **wing** of the **phoenix** will be an **integer** in **range [-231, 231 – 1]**.
* The **total years** is a **product** of **integers** and **floating-point numbers**, thus it is a **floating-point number**.
* The **total years** should have the **same accuracy** as the **total width**.
* Allowed working time / memory: **100ms / 16MB**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2  100  50  30  150  25  10 | 1100000  1012500 | **2 phoenixes:**  **P1**:  Body length: 100  Body width: 50  Length of 1 wing: 30  Total years: 100 ^ 2 \* (50 + 2 \* 30) = 1100000  **P2**:  Body length: 150  Body width: 25  Length of 1 wing: 10  Total years: 150 ^ 2 \* (25 + 2 \* 10) = 1012500 |
| 2  100  50.243  31  154  23.132  11 | 1122430.000  1070350.512 | **2 phoenixes:**  **P1**:  Body length: 100  Body width: 50.243  Length of 1 wing: 31  Total years: 100 ^ 2 \* (50.243 + 2 \* 31) = 1122430.000  **P2**:  Body length: 154  Body width: 23.132  Length of 1 wing: 11  Total years: 154 ^ 2 \* (23.132 + 2 \* 11) = 1070350.512 |

# Problem 2. Icarus

Icarus is the majestic phoenix who has been alive from the beginning of creation. Icarus travels through different planes. When Icarus travels through a plane, he damages Reality itself with his overwhelming, beyond godlike flames.

You will receive a **sequence** of **integers** – the **plane**. After that you will receive **1** **integer** – an **index** in that **sequence**, which is Icarus’s **starting position**. Icarus’s **INITIAL DAMAGE** is **1**.

You will then begin **receiving** **commands** in the following format: “{direction} {steps}”. The direction will be either “left” or “right”, and the **steps** will be an **integer**. Depending on the direction, Icarus must **step** through the sequence of **integers to the left** or **right**. Each time he **steps** on a **NEW** **position**, he **damages** it. In other words, he **SUBTRACTS** his **current damage** **from** the **integer** at **that** **position**. Walking left and right has its conditions though:

* If Icarus **passes beyond** the **start** of the **sequence** (index: -1) while going **left**, he must go at the **end** of the **sequence** (index: length – 1).
* If Icarus **passes beyond** the **end** of the **sequence** (index: length - 1) while going **right**, he must go at the **start** of the **sequence** (index: 0).

If **1** of the **2 cases** **stated above** happens, Icarus **increments** his **damage** by **1**.

The input ends when you receive the command “Supernova”. When that happens you must print what is **left** of the **sequence**.

### Input

* On the **first input line** you will get the **sequence** of **integers**, **separated** by **spaces**.
* On the **second input line** you will get Icarus’s **starting position**.
* On the **next several input lines** you will get the **commands**.

### Output

* As output you must print a **single line** containing the **remaining elements** of the **sequence**, **separated** by **spaces**.

### Constrains

* The **integers** in the **sequence** will be in **range [0, 1000]**.
* The **initial position** of Icarus will **always** be **valid** and **inside** the **sequence’s indexes**.
* The **direction** will always be either “left” or “right”.
* The **steps** will be in **range [0, 1000]**.
* There will be **NO invalid** input lines.
* Allowed working time / memory: **100ms / 16MB**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 50 50 25 50 50  3  left 2  right 2  left 2  right 2  Supernova | 50 48 21 48 50 | Initial index: 3  Initial state:  50 50 25 50 50  Go left 2 steps:  50 50 24 50 50  50 49 24 50 50  Go right 2 steps:  50 49 23 50 50  50 49 23 49 50  Go left 2 steps:  50 49 22 49 50  50 48 22 49 50  Go right 2 steps:  50 48 21 49 50  50 48 21 48 50  Final state:  50 48 21 48 50 |
| 5 3 5 5 5  2  left 5  left 5  Supernova | 2 0 0 0 0 | Initial index: 2  Initial state:  5 3 5 5 5  Go left 5 steps:  5 2 5 5 5  4 2 5 5 5  4 2 5 5 3  4 2 5 3 3  4 2 3 3 3  Go left 5 steps:  4 0 3 3 3  2 0 3 3 3  2 0 3 3 0  2 0 3 0 0  2 0 0 0 0  Final state:  2 0 0 0 0 |

# Problem 3. Phoenix Grid

The Phoenix Grid is an ancient artifact created by the Linguistics miracle – Mozilla, The “Fire Bird”. It is used to translate Phoenix language. You are the newest scientist, researching the Grid and as the research team was almost out of hope, you came up with the genius idea to use Regular Expressions! You saved the day! You are a Hero!

You will begin **receiving encoded messages**. You must **CHECK** each **one** of **them** and if it’s a **VALID**.

A **valid encoded message** consists of **one** **phrase** or **more phrases**, separated by **DOTS** (‘.’).

* A **phrase** consists of exactly **3 characters**.
* A **phrase CANNOT** contain **whitespace** characters or the ‘\_’ (underscore) character.

**Valid** messages: “asd.dsa”, “123.312”, “[3@a.231](mailto:3@a.231)”, “111”, “@sd”, “132.31$.ddd” . . .

**Invalid** messages: “123asdasd.dsa”, “\_@a. sd”, “a.s.d” . . .

When you have found a valid message, you must **check** if it a **PALINDROME** – if it reads the same backward as forward.

**Palindrome** messages: “asd.dsa”, “123.321”, “cat.php.tac” . . .

If the **message** is **VALID** and is a **PALINDROME** print “**YES**”. In any other case, print “**NO**”.

The input ends when you receive the command “ReadMe”.

### Input

* As input you will receive several input lines containing encoded messages.

### Output

* As output you must print **for each** **message** “**YES**” or “**NO**” if its **valid** or **not**.

### Constrains

* The input lines may contain **any ASCII character**.
* There will be no more than **1000 input lines**.
* Allowed working time / memory: **100ms / 16MB**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| asd  asd.asd  asd.dsa  123.323.321  \_ds.\_sad.sds  jss.csh.php.hsc.ssj  ReadMe | NO  NO  YES  YES  NO  YES |
| asa  igi.igi  \_\_\_.\_\_\_  .  sds.dsd.sds.dsd.sds.dsd.sds  xha.ahx  ReadMe | YES  YES  NO  NO  YES  YES |

# Problem 4. CODE: Phoenix Oscar Romeo November

The fire creatures are assembling in squads to fight The Evil Phoenix God. You have been tasked to determine which squad is the strongest, so it will be sent as The Vanguard.

You will begin receiving input lines containing information about fire creatures in the following format:

{creature} -> {squadMate}

The creature and the squadMate are **strings**. You should store every **creature**, and his **squad mates**. If the **creature** already **exists**, you should **add** the **new squad mate** to it.

* If there is **already** a **squad mate** with the **given name** in the **given creature’s squad**, **IGNORE** that **line** of **input**.
* If the **given squad mate name** is the **same** as the **given** **creature**, **IGNORE** that **line** of **input**.

The **input sequence ends** when you receive the command “Blaze it!”.

When that happens you must **print** the **creatures ordered** in **descending** order by **count** of **squad mates**. Sounds simple right? But there is one little **DETAIL**.

If a particular creature has a squadMate, and that squadMate has that creature in his squadMates, you **should NOT consider** them as **part** of the **count** of **squad mates**.

**Example**:

Creature 1: **Mozilla** -> {Tony, Dony, Mony}

Creature 2: **Tony** -> {Mozilla, Franzilla, Godzilla}

**Mozilla** has **2 squad mates** in total, because **Tony** also has **Mozilla** in his **squad mates**.

**Tony** has **2 squad mates** in total, because **Mozilla** also has **Tony** in his **squad mates**.

### Input

* As input you will receive several input lines containing information about the fire creatures.
* The input sequence ends when you receive the command “Blaze it!”.

### Output

* As output you must print each of the creatures the following information:
  + {creature} : {countOfSquadMates}
* As it was stated above, mind the **count** of **squad mates**. If **2 creatures** have themselves in their **squad mates**, they should **NOT** be **counted**.

### Constrains

* The creature and the squadMate will be **strings** which may contain **any ASCII character**.
* There will be **NO invalid** input lines.
* Allowed time / memory: **100ms / 16MB**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Mozilla -> Tony  Tony -> Godzilla  Mozilla -> Dony  Tony -> Franzilla  Mozilla -> Mony  Tony -> Mozilla  Blaze it! | Mozilla : 2  Tony : 2 |
| FireBird -> FireMane  Phoenix -> FireVoid  FireVoid -> FireMane  FireSnow -> FireMane  Phoenix -> FireBird  FireMane -> FireBird  FireMane -> FireVoid  Phoenix -> FireSnow  FireMane -> FireSnow  FireMane -> FireMane  Phoenix -> FireMane  Phoenix -> FireVoid  Blaze it! | Phoenix : 4  FireBird : 0  FireVoid : 0  FireSnow : 0  FireMane : 0 |

## Министерство на образованието и науката (МОН)

* Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист".



* Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под **свободен** **лиценз CC-BY-NC-SA** (Creative Commons Attribution-Non-Commercial-Share-Alike 4.0 International).

