# Programming Fundamentals Mid Exam Retake 07 April 2020

## Problem 1. Counter Strike

Write a program that **keeps track of every won** battle against an **enemy**. You will receive **initial energy**. Afterwards you will start receiving the **distance** you need to **go to reach an enemy** until the **"End of battle"** command is given, or until you **run out of energy.**

The **energy** you need for reaching an enemy is **equal to the distance you receive**. Each time you reach an enemy, your **energy is reduced.** This is considered a successful battle (**win**). If you don't have **enough energy** to reach an the enemy, print:

**"Not enough energy! Game ends with {count} won battles and {energy} energy"**

and **end the program.**

Every **third won battle** increases **your energy with the value of your current count of won battles**.

Upon receiving the **"End of battle"** command**,** print the **count of won battles** in the following format:

### "Won battles: {count}. Energy left: {energy}"

### Input / Constraints

* On the **first line** you will receive **initial energy** – an **integer [1-10000]**.
* On the **next lines,** you will be receiving **distance** of the enemy – an **integer** **[1-10000]**

### Output

* The description contains the proper output messages for each case and the format in which they   
  should be print.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 100  10  10  10  1  2  3  73  10 | Not enough energy! Game ends with 7 won battles and 0 energy | Initial energy is 100. The first distance is 10, so we subtract 10 from 100 and we consider this a **won** battle. We are left with 90 energy. Next distance – 10, and 80 energy left.  Next distance – 10, 3 won battles and 70 energy, but since we have 3 won battles, we increase the energy with the current count of won battle, in this case – **3 and it becomes 73**.  The last distance we receive – **10** is unreachalble since we have **0** energy, so we print the appropriate message and the program ends. |
| 200  54  14  28  13  End of battle | Won battles: 4. Energy left: 94 |  |

## Problem 2. Shoot for the Win

Write a program that helps you keep track of your **shot targets**. You will receive a **sequence with integers**, separated by single space, representing targets and their value. Afterwards, you will be receiving indices until the **"End"** command is given and you need to print the **targets** and the **count of shot targets**.

Every time you receive an **index**, you need to shoot the target on that index, **if it is possiblie**.

Everytime you **shoot a target**, its value becomes **-1 and it is considered shot**. Along with that you also need to:

* **Reduce** all the other **targets**, which have **greater values** than your **current** target, **with its value**.
* All the **targets**, which **have less than or equal** value to the **shot target**, you need to **increase** **with its value.**

**Keep in mind that you can't shoot a target, which is already shot.** **You also can't increase or reduce a target, which is considered shot.**

When you receive the **"End"** command, print the targets in their current state and the **count of shot targets** in the following format:

**"Shot targets: {count} -> {target1} {target2}… {targetn}"**

### Input / Constraints

* On the **first line** of input you will receive a **sequence** of **integers**, **separated** by **a single space – the targets sequence**.
* On the **next lines**, until the **"End"** command you be receiving **integers** each on a single line – **the index of the target to be shot.**

### Output

* The format of the output is described above in the problem description.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 24 50 36 70  0  4  3  1  End | Shot targets 3 -> -1 -1 130 -1 | First we shoot target on index 0. It becomes equal to -1 and we start going through the rest of the targets. Since 50 is more than 24, we reduce it to 26 and 36 to 12 and 70 to 46. The sequence looks like that:  **-1 26 12 46**  The next index is invalid, so we don't do anything. Index 3 is valid and after the operations our sequence should look like that:  **-1 72 58 -1**  Then we take the first index with value 72 and our sequence looks like that:  **-1 -1 130 -1**  Then we print the result after the **"End"** command. |
| 30 30 12 60 54 66  5  2  4  0  End | Shot targets: 4 -> -1 120 -1 66 -1 -1 |  |

## Problem 3. Moving Target

You are at the shooting gallery again and you need a program that helps you keep track of moving targets. On the first line, you will receive a **sequence of targets with their integer values**, split by a **single space**. Then, you will start receiving **commands for manipulating the targets**, until the **"End"** command. The commands are the following:

* **Shoot {index} {power}**
  + Shoot the target at the index, **if it exists** by **reducing** its **value** by the **given** **power** (**integer value**).A target is considered **shot** when **its value reaches 0**.
  + Remove the target, **if it is shot**.
* **Add {index} {value}**
  + Insert a target with the received value at the received **index, if it exist**. If not, print: **"Invalid placement!"**
* **Strike {index} {radius}**
  + Remove the **target at the given index** and the **ones before and after it depending on the radius, if such exist. If any of the indices in the range is invalid print:**

**"Strike missed!" and skip this command.**

**Example:** **Strike 2 2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | {radius} | {radius} | {strikeIndex} | {radius} | {radius} |  |  |

* **End**
  + Print the sequence with targets in the following format:

**{target1}|{target2}…|{targetn}**

### Input / Constraints

* On the **first line** you will receive **the sequence of targets** – **integer values [1-10000]**.
* On the **next lines,** until the **"End"** will be receiving the command described above – **strings**.
* There will never be a case when **"Strike"** command would empty the whole sequence.

### Output

* Print the appropriate message in case of **"Strike"** command if necessary.
* In the end, print the sequence of targets in the format described above.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 52 74 23 44 96 110  Shoot 5 10  Shoot 1 80  Strike 2 1  Add 22 3  End | Invalid placement!  52|100 | The first command is "**Shoot**", so we reduce the target on **index** **5**, which is valid, with the given **power** – **10**.  Then we receive the same command but we need to reduce the target on the 1st index, with power 80. The value of this target is 74, so it is considered shot and we **remove** it.  Then we receive the "**Strike**" command on the 2nd index and we need to check if the range with radius 1 is valid:  **52 23 44 96 100**  And it is, so we **remove** the targets.  At last we receive the "**Add**" command, but the index is **invalid** so we print the appropriate **message** and in the end we have the following result:  **52|100** |
| 47 55 85 78 99 20  Shoot 1 55  Shoot 8 15  Strike 2 3  Add 0 22  Add 2 40  Add 2 50  End | Strike missed!  22|47|50|40|85|78|99|20 |  |

# [Programming Fundamentals Mid Exam - 29 February 2020 Group 2](https://judge.softuni.bg/Contests/2031/Programming-Fundamentals-Mid-Exam-29-February-2020-Group-2)

## Problem 1. National Court

*Every day thousands of people pass by the reception at "National Court" with various questions to ask and the employees have to help everyone by providing correct information and to answer all questions.*

There are **3 employees** working on the reception all day long. Each of them can handle different number of **people** **per hour**. Your task is to calculate **how much time** it will take **to** **answer all the questions** of a given number **of people**.

First you will receive 3 lines with integers, representing the **count of people** that each of the **employee can help per hour.** On the next line you will receive the **total** **people count** as a single integer.

Every **fourth hour** all the employees **have a one-hour break** before they start working again. This is the only break they get because they don`t need rest and have no personal life. Calculate the time needed to answer all people`s questions and print it in the following format: "Time needed: {time}h."

#### Input / Constraints

* On first three lines - **each employee`s efficiency** - an integer in the range **[1 - 100]**
* On the fourth line - **people count** – an integer in the range **[0 – 10000]**
* Input will always be valid and in the range specified

#### Output

* Print a single line: "Time needed: {time}h."
* Allowed working **time** / **memory**: **100ms** / **16MB**

#### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 5  6  4  20 | Time needed: 2h. | All employees can answer 15 people per hour. After the first hour there are 5 people left to be answered.  All people will be answered in the second hour. |
| 1  2  3  45 | Time needed: 10h. | All employess can answer 6 people per hour. In the first 3 hours they have answered 6 \* 3 = 18 people. Then they have a break for an hour.  After the next 3 hours there are  18 + 6 \* 3 = 36 answered people.  After the break for an hour, there are only 9 people to answer.  So in the 10th hour all of the people questions would be answered. |
| 3  2  5  40 | Time needed: 5h. |  |

## Problem 2. Shopping List

*It’s the end of the week and it is time for you to go shopping, so you need to create a shopping list first.*

#### Input

You will receive an **initial list** with groceries separated by **"!"**.

After that you will be receiving **4 types** of commands, until you receive **"Go Shopping!"**

* **Urgent {item} -** **add** the item at the **start** of the list. If the item **already exists,** skip this command.
* **Unnecessary {item} - remove** the item with the given name, only **if it exists** in the list. Otherwise skip this command.
* **Correct {oldItem} {newItem} –** if the item with the given **old name** exists, **change** its name with the **new** one. If it **doesn't exist**, skip this command.
* **Rearrange {item} -** if the grocery exists in the list, **remove** it from its **current position** and **add** it at the **end** of the list.

#### Constraints

* There won`t be any duplicate items in the initial list

#### Output

Print the **list** with all the groceries, joined by **", ".**

* **"{firstGrocery}, {secondGrocery}, …{nthGrocery}"**

#### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Tomatoes!Potatoes!Bread  Unnecessary Milk  Urgent Tomatoes  Go Shopping! | Tomatoes, Potatoes, Bread |
| **Input** | **Output** |
| Milk!Pepper!Salt!Water!Banana  Urgent Salt  Unnecessary Grapes  Correct Pepper Onion  Rearrange Grapes  Correct Tomatoes Potatoes  Go Shopping! | Milk, Onion, Salt, Water, Banana |

## Problem 3. Heart Delivery

*Valentine’s Day is coming, and Cupid has very limited time to spread some love across the neighborhood. Help him with his mission!*

You will receive a **string** with **even integers,** separated by a **"@".** This is our neighborhood. After that a series of **Jump** commands will follow, until you receive **"Love!"** Every house in the neighborhood needs a certain number of **hearts** delivered by Cupid, in order to be able to celebrate Valentine’s Day. Those needed hearts are indicated by the integers in the neighborhood.

Cupid starts at the position of the **first** **house** (**index 0) and must jump by a given length. The jump commands will be in this** format: **"Jump {length}"**.

Every time he jumps from one house to another, the needed hearts for the visited house are **decreased by 2**. If the needed hearts for a certain house become **equal to 0** , print on the console **"Place {houseIndex} has Valentine's day."** If **Cupid** jumps to a house where the needed hearts are **already** **0,** print on the console"**Place {houseIndex} already had Valentine's day.**".

Keep in mind that **Cupid** can have a **bigger jump length** than the **size of the neighborhood** and if he does jump **outside** of it, he should **start** from the **first house** again**.**

*For example, we are given this neighborhood: 6@6@6. Cupid is at the start and jumps with a length of 2. He will end up at index 2 and decrease the needed hearts there by 2: [6, 6, 4]. Next he jumps again with a length of 2 and goes outside the neighborhood, so he goes back to the first house (index 0) and again decreases the needed hearts there: [4, 6, 4].*

#### Input

* On the first line you will receive a **string** with **even integers** separated by **"@"** –the neighborhood and the number of hearts for each house.
* On the next lines, until "**Love!**" is received, you will be getting jump commands in this format: "**Jump {length}**".

#### Output

At the end print **Cupid's** **last position** and whether his mission was successful or not:

* "**Cupid's last position was {lastPositionIndex}.**"
* If **each house** has had a Valentine's day, print:
  + "**Mission was successful.**"
* If **not,** print the **count** of all houses that **didn`t** celebrate a Valentine's Day:
  + **"Cupid has failed {houseCount} places."**

#### Constraints

* The **neighborhood`s** size will be in the range [1…20]
* Each **house** will need an **even number** of hearts in the range [2 … 10]
* Each **jump length** will be an integer in the range [1 … 20]

#### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 10@10@10@2  Jump 1  Jump 2  Love! | Place 3 has Valentine's day.  Cupid's last position was 3.  Cupid has failed 3 places. | Jump 1 ->> [10, 8, 10, 2]  Jump 2 ->> [10, 8, 10, 0] so we print "Place 3 has Valentine's day."  Next command is "Love!", so we print Cupid`s last position and the outcome of his mission. |
| 2@4@2  Jump 2  Jump 2  Jump 8  Jump 3  Jump 1  Love! | Place 2 has Valentine's day.  Place 0 has Valentine's day.  Place 0 already had Valentine's day.  Place 0 already had Valentine's day.  Cupid's last position was 1.  Cupid has failed 1 places. |  |

# Programming Fundamentals Mid Exam - 29 February 2020 Group 1

## Problem 1. Bonus Scoring System

Create a program that calculates **bonus points** for each **student**, for a certain **course**. On the first line, you are going to receive **the count of the students** for this course. **On the second line**, you will receive **the count of the lectures** in the course. Every course has **an additional bonus**. You are going to receive it **on the third line**. On the next lines, you will be receiving the **count of attendances** **for each student**.

The bonus is calculated with the following **formula**:

**{total bonus} = {student attendances} / {course lectures} \* (5 + {additional bonus})**

Find the student with the **maximum bonus** and print him/her, along with **his attendances** in the following format:

**"Max Bonus: {maxBonusPoints}."**

**"The student has attended {studentAttendances} lectures."**

Round the bonus points at the end to **the nearest bigger number**.

### Input / Constrains

* On the **first line** you are going to receive the count of the students – an integer number in the range [0…50]
* On the **second line** you are going to receive the count of the lectures – an integer number in the range [0...50].
* On the **third line** you are going to receive **the initial bonus** – an integer number in the range [0….100].
* **On the next lines**, you will be receiving the **attendances of each student**.
* There will **never** be **students with equal bonuses**.

### Output

* Print the maximum bonus points along with the attendances of the given student, **rounded** to the nearest **bigger** number, scored by a student in this course in the format described above.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  25  30  12  19  24  16  20 | Max Bonus: 34.  The student has attended 24 lectures. |
| **Comments** | |
| First, we receive the **number of students** enrolled in the course – **5**. The total count of the lectures is **25** and the initial bonus is **30**. Then we calculate the bonus of the student with 12 attendances, which is **16.8**. We continue calculating **each of the student’s bonuses**. The one **with 24 attendances** has the **highest bonus – 33.6 (34 rounded)**, so we print the appropriate message on the console. | |
| 10  30  14  8  23  27  28  15  17  25  26  5  18 | Max Bonus: 18.  The student has attended 28 lectures. |

## Problem 2. Mu Online

You have **initial health 100 and initial bitcoins 0**. You will be given **a string, representing the dungeons rooms**. Each room is separated with **'|'** (vertical bar): **"room1|room2|room3…"**

Each room contains **a command** and a **number**, separated by space. The command can be:

* **"potion"**
  + You are healed with the number in the second part. But your health **cannot exceed** your **initial health (100)**.
  + First print: **"You healed for {amount} hp."**.
  + After that, print your current health: **"Current health: {health} hp."**.
* **"chest"**
  + You've found some bitcoins, the number in the second part.
  + Print: **"You found {amount} bitcoins."**
* In any other case you are facing a monster, you are going to fight. The second part of the room, contains the attack of the monster. You should remove the monster's attack from your health.
  + If you are not dead (health <= 0) you've slain the monster, and you should print (**"You slayed {monster}."**)
  + If you've died, print **"You died! Killed by {monster}."** and your quest is over. Print the best room you`ve manage to reach: **"Best room: {room}"**.

If you managed to go through all the rooms in the dungeon, print on the next three lines:

**"You've made it!"**, **"Bitcoins: {bitcoins}"**, **"Health: {health}"**.

#### Input / Constraints

You receive a string, representing the dungeons rooms, separated with **'|'** (vertical bar): **"room1|room2|room3…"**.

#### Output

Print the corresponding messages, described above.

#### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| rat 10|bat 20|potion 10|rat 10|chest 100|boss 70|chest 1000 | You slayed rat.  You slayed bat.  You healed for 10 hp.  Current health: 80 hp.  You slayed rat.  You found 100 bitcoins.  You died! Killed by boss.  Best room: 6 |
| **Input** | **Output** |
| cat 10|potion 30|orc 10|chest 10|snake 25|chest 110 | You slayed cat.  You healed for 10 hp.  Current health: 100 hp.  You slayed orc.  You found 10 bitcoins.  You slayed snake.  You found 110 bitcoins.  You've made it!  Bitcoins: 120  Health: 65 |

## Problem 3. Inventory

*As a young traveler, you gather items and craft new items.*

#### Input / Constraints

You will receive a journal with some Collecting items, separated with **', '** (comma and space). After that, until receiving "Craft!" you will be receiving different commands.

Commands (split by **" - "**):

* "Collect - {item}" – Receiving this command, you should add the given item in your inventory. If the item already **exists**, you should **skip** this line.
* "Drop - {item}" – You should remove the item from your inventory, **if it exists**.
* "Combine Items - {oldItem}:{newItem}" – You should check if the **old item exists**, if so, **add** the new item **after** the old one. Otherwise, **ignore** the command.
* "Renew – {item}" – If the given item exists, you should change its position and **put it last** in your inventory.

#### Output

After receiving "Craft!" print the items in your inventory, separated by **", "** (comma and space).

#### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Iron, Wood, Sword  Collect - Gold  Drop - Wood  Craft! | Iron, Sword, Gold |
| **Input** | **Output** |
| Iron, Sword  Drop - Bronze  Combine Items - Iron: Sword  Renew - Iron  Craft! | Sword, Bow, Iron |

# Programming Fundamentals Mid Exam Retake - 10 December 2019

## Disneyland Journey



*You are planning a trip to Disneyland in the next year. Are you able to save money for it?*

Create a program that checks if you can **save the money** for the Disneyland’s journey. You have a **certain number of months** for which you can collect the money.

**At the end of each month,** you save **25% of the cost of the journey.**

**At the beginning of** every **odd** month (**except** the **first**one) you **spent 16%** of the money collected so far for clothes and shoes.

Every **4th (fourth)** month **at the beginning of the month** your bossgives you а bonus. It is **25%** of the money collected so far.

If you save enough money for the journey, calculate how much money will be left for the souvenirs. Then print the following:

**"Bravo! You can go to Disneyland and you will have {money}lv. for souvenirs."**

If the saved money is less you should calculate how much money you need more. Then print the following:

**"Sorry. You need {money}lv. more."**

Both numbers should be **formatted** to the **2nd decimal place**.

### Input

* On the **1st line** you will receive how much the journey will cost – a **real number** in the range [500.0…10000.0]
* On the **2nd line** you will receive the **number of months** for which you have to collect money – an **integer number** in the range [1…12]

### Output

* Print the output in the **format** **described** **above**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1000 4 | Bravo! You can go to Disneyland and you will have 87.50lv. for souvenirs. |
| **Comments** | |
| You need 1000 leva for the journey and you have 4 months to collect them. Every month you can save 1000 \* 25% => 250 lv. So, to the end of the **1st** month you have **250 lv**. To the **end** of the **2nd** month - 250 + 250 -> **500** **lv**. To the **begging** of the **3th** month (**odd month**) you spent **80 lv.** (500 \* 16%) for clothes and shoes, then you save **250 lv**, so now you have **670 lv**. Last month is the **fourth month** and you save 670 + 167.5 (670 \* 25%) + 25 = **1087.5 lv**. You have 1087.5 – 1000 = **87.5 lv.**, so you can go to the trip. | |
|  | |
| 3265  3 | Sorry. You need 1077.45lv. more. |

## Archery Tournament

*Our hero Iskren is going to take part in an archery tournament. Your task is to help him calculate his points.*

On the **first line** you will receive integers separated by **"|"**, representing the targets in the archery field.  
On the next lines until you receive "**Game over**" command, you will receive commands by the Judge of the tournament:

* **"Shoot Left@{start index}@{length}":**
* Iskren starts traversing the archery field to the **left** from **{start index}** with given **{length}**.
* If he goes **out of the field**, he will continue from the **end of the field**.
* **"Shoot Right@{start index}@{length}":**
* Iskren starts traversing the archery field to the **right** from **{start index}** with given **{length}**.
* If he goes **out of the field**, he will continue from the **start of the field.**
* **"Reverse":**

**Reverse** the archery field.

* **"Game Over"**

**Print** the archery **field and collected points**.

When **he arrives at the target**, he will shoot at it and **increase his points by 5** and **decrease the target by 5 points**, if the **target points are less than 5**, he takes **all of them and decreases it to 0**. If the **start index is out of range** of the field Iskren will have to **ignore the command**.

### Input

* On the **first line**, you will receive integers **separated** by **"|"** representing the **targets in the archery field**.
* On the **next lines**, until the "**Game over**" command you will receive **commands** in the **format described** **above**.

### Output

* Print the field in following format: **"{target} - {target} - {target} ….. - {target}"**.

**"Iskren finished the archery tournament with {points}!"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 10|10|10|10|10 Shoot Left@0@2  Shoot Right@4@5  Shoot Right@6@5  Reverse  Game over | 5 - 5 - 10 - 10 - 10  Iskren finished the archery tournament with 10 points! |
| **Comments** | |
| First, Iskren receives the **"Shoot Left@0@2"** command, so he starts traversing the archery field from **index 0** with **length 2** and he stops at **index 3**. He shoots and the archery filed looks like this: 10 - 10 - 10 - 5 – 10 ,  and his points are 5.  Then he receives **"Shoot Right@4@5"** -> **"10 - 10 - 10 - 5 – 5"**  **"Shoot Right@6@5"** -> Index 6 is **out of range** of the field , so Iskren ignores the command.  **“Reverse”**-> After that command the field looks like this: 5 - 5 - 10 - 10 - 10 | |
|  | |
| 20|30|40|50|60 Shoot Left@0@12  Shoot Right@4@15  Shoot Left@6@5  Reverse  Game over | 55 - 45 - 40 - 30 - 20  Iskren finished the archery tournament with 10 points! |

## School Library

*As a young developer Iskren is a part from software development school team. His task is to do online book library, but he needs your help for that.*

On the first line you will receive a String, representing shelf with books in library. Every book is separated with "**&**".

On the next lines until you receive "**Done**" command, you will receive following commands:

* **Add Book | {book name}**
* **Add** a book at **first place** in the shelf.
* If the book **already is present** on the shelf, **ignore the command**.
* **Take Book | {book name}**
* **Remove** the book with the given name only **if the book is on the shelf**, otherwise **ignore this command**.
* **Swap Books | {book1} | {book2}**
* If both books **are on the shelf**, swap their places.
* **Insert Book | {book name}**
* **Add** a book at the **end of the book collection**.
* **Check Book | {index}**
* **Print** the name of the book on the given index the book.
* If the index is **invalid**, **ignore the command**.

### Input

* **On the 1st line**, you will receive a string, representing book names, separated with "**&**".
* On the **next lines**, until you receive "**Done**", you will receive **commands** in the **format described** above.

### Output

* Print the **collection** of books joined by **", ".**

**"{firstBook}, {secondBook}, …{lastBook}"**

### Constraints

* You won't receive duplicate book names in the initial list of books.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Don Quixote&The Great Gatsby&Moby Dick&Hamlet  Add Book | The Odyssey  Take Book | Don Quixote  Insert Book | Alice's Adventures in Wonderland  Check Book | 3  Done | Hamlet  The Odyssey, The Great Gatsby, Moby Dick, Hamlet, Alice's Adventures in Wonderland |
| **Input** | **Output** |
| Anna Karenina&Heart of Darkness&Catch-22& The Stranger  Add Book | David Copperfield  Add Book | One Thousand and One Nights  Swap Books | One Thousand and One Nights | Catch-22  Take Book | David Copperfield  Insert Book | The Stories of Anton Chekhov  Check Book | 17  Done | Catch-22, Anna Karenina, Heart of Darkness, One Thousand and One Nights, The Stranger, The Stories of Anton Chekhov |

# Programming Fundamentals Mid Exam - 2 November 2019 Group 2

## Problem 1. Experience Gaining

*Write a program, that helps a player figure how many battles he will need to play in a battle video game, in order to unlock the next tank in the line.*

On **the first line** you will **receive the amount of experience** that is needed to unlock the tank. On **the second line** you will **receive the count of battles**. On **the** **next lines,** you will receive the experience the player can gain in every battle.

Calculate if he **can unlock** the tank. Keep in mind that he **gets 15%** more experience for every **third battle** and **10% less** for every **fifth battle**. You also need to **stop the program** as soon as he **collects the needed experience**.

If he managed to gather the experience, **print** **how many battles it took him** in the following format:

* "**Player successfully collected his needed experience for {battleCount} battles."**

If he was not able to do it, **print** the following message:

* **"Player was not able to collect the needed experience, {neededExperience} more needed."**

**Format** the needed experience to **the second decimal place**.

### Input

* On the **first line** you will receive the **needed experience** amount – a **real number** in the range [0.0….400000.0]
* On the **second line** you will receive the **count of battles** – an **integer number** in the range

[0….500]

* On the **next lines** you will receive the **experience earned per battle** – a **real number** in the range

[0.0….5000.0]

#### Output

* If he **managed to gather** the experience**:**
  + **"Player successfully collected his needed experience for {battleCount} battles."**
* If he was **not able** to do it**:**
  + **"****Player was not able to collect the needed experience, {neededExperience} more needed."**

**NOTE: Format** the needed experience to **the second decimal place**.

#### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 500  5  50  100  200  100  30 | Player successfully collected his needed experience for 5 battles. |
| **Comments** | |
| The first line is the amount of the wanted experience. – **"500"**  The second line is the expected battles for which he has to collect the experience. – **"5"**  After that is the experience received for every battle:  **50 + 100 + (200 + (200 \* 15 %)) + 100 + (30 – (30 \* 10 %)) = 507**  So on the console is printed :  **"Player successfully collected his needed experience for 5 battles."** | |
| **Input** | **Output** |
| 500  5  50  100  200  100  20 | Player was not able to collect the needed experience, 2.00 more needed. |
| **Input** | **Output** |
| 400  5  50  100  200  100  20 | Player successfully collected his needed experience for 4 battles. |

## Problem 2. Friend List Maintenance

*Our player is having trouble with his friend list and some guys are disappearing without a reason so he asks you to create a program that will figure out what is going on and at the end will bring him a report.*

On **the first line** you will **receive** all his friends separated by **", "**.On **the next lines** until the **"Report"** command you will receive **commands**. The commands could be:

* **Blacklist {name}**
  + **Find** the name in the friend list and **change it** to **"Blacklisted"** and **print on the console**:
    - **"{name} was blacklisted."**
  + If the **name** is **not** in **the friend list** **print**:
    - **"{name} was not found."**
* **Error {index}**
  + **Check** if the **username** at the given index is **not "Blacklisted"** or **"Lost".** If it isn't, **change** the username to **"Lost"** and **print** on the console:
    - **"{name} was lost due to an error."**
* **Change {index} {newName}**
  + **Check** if the user at **index** position is in **range of the array.** If he is**,** **change** the **current** username with the **new one** and **print** on console:
    - **"****{currentName} changed his username to {newName}."**

After you **receive "Report"** print on the console the **count of blacklisted names**, the **count of lost names,** and the friend list separated by a **single space**.

#### Input

* The **first input line** will contain the usernames that need to be stored.
* On the **next input** lines until **"Report"** you will **receive** commands.

#### Output

* The output should be in the following format:
  + **"Blacklisted names: {blacklistedNamesCount}"**
  + **"Lost names: {lostNamesCount}"**
  + **"{name1} {name2} .. {nameN}"**

#### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Mike, John, Eddie  Blacklist Mike  Error 0  Error 1  Change 2 Mike123  Report | Mike was blacklisted.  John was lost due to an error.  Eddie changed his username to Mike123.  Blacklisted names: 1  Lost names: 1  Blacklisted Lost Mike123 |
| **Comments** | |
| On the first line are the names from the friendlist that need to be stored in an array.  After that the commands start to flow in. The first command finds Mike and blacklists him: **"Mike was blacklisted."**  After that **"Error 0"** failed because the name is already blacklisted and we do nothing.  **"Error 1":** John is replaced with **"Lost"** and the messange is sent to the console: **"John was lost due to an error."**  After that Mike changes his username to Mike123: **"Eddie changed his username to Mike123".**  And the report is asked for so the program ends with the shown output. | |
|  | |
| Mike, John, Eddie, William  Error 3  Error 3  Change 0 Mike123  Blacklist Eddie  Report | William was lost due to an error.  Mike changed his username to Mike123.  Eddie was blacklisted.  Blacklisted names: 1  Lost names: 1  Mike123 John Blacklisted Lost |

## Problem 3. Tanks Collector

Tom is a world of tanks player and he likes to collect premium tanks. You will **receive a list** of Tom's already owned premium vehicles **on a single line separated by ", ".** On the next **n** lines you will receive commands that could be:

* **Add, {tankName}**: Check if he already owns the wanted tank.
  + If he owns it, **print on console: "Tank is already bought"**
  + If not, **print on console: "Tank successfully bought"** and **add it** to the tank list.
* **Remove, {tankName}**: Check if he owns the tank.
  + If he owns it **print on console: "Tank successfully sold"** and **remove it** from the tank list.
  + If not **print on console: "Tank not found"**
* **Remove At, {index}**: Check if the **index** is in the range of the list.
  + If not **print on console: "****Index out of range"** and continue.
  + If it’s in range, **remove at the given index** and **print on console: "Tank successfully sold"**
* **Insert, {index}, {tankName}**: Check if the **index** is in range of the list and **check if he already owns the tank**.
  + If not **print on console: "Index out of range"** and continue.
  + If it's in range and he doesn't own the tank then **add the tank at the given index** and **print on** **console:**
    - **"Tank successfully bought"**
  + If the tank is in range and he owns it already than **print on console:**
    - **"Tank is already bought"**

After you go through all the commands you need to **print** the list **on a single line separated by ", ".**

#### Input

* The **first input line** will contain the **list** of **already owned tanks**.
* The **second** **input line** will be the **number of commands** – an **integer number** in range [0…50].
* On the **next input lines** you will be receiving commands.

#### Output

* As output you must print a single line containing the elements of the list, **joined** by **", "**.

#### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| T-34-85 Rudy, SU-100Y, STG  3  Add, King Tiger(C)  Insert, 2, IS-2M  Remove, T-34-85 Rudy | Tank successfully bought  Tank successfully bought  Tank successfully sold  SU-100Y, IS-2M, STG, King Tiger(C) |
| **Comments** | |
| The first command gives the tank list so its splitted and added into the list.  **"T-34-85 Rudy, SU-100Y, STG"**  The second commands gives the number of commands that will be received.  **"3"**  The Add command adds the tank to the list after the necessary checks.  **"Add, King Tiger(C)" – "Tank successfully bought"**  The Insert commands also adds the tank at the given spot after the necessary checks. **"Insert, 2, IS-2M" – "Tank successfully bought"**  The Remove command is the last one and after the checks the tank is sold.  **"Remove, T-34-85 Rudy" – "Tank successfully sold"**  After that we print the list on the console.  **"SU-100Y, IS-2M, STG, King Tiger(C)"** | |
| T 34, T 34 B, T92, AMX 13 57  4  Add, T 34  Remove, AMX CDC  Insert, 10, M60  Remove At, 1 | Tank is already bought  Tank not found  Index out of range  Tank successfully sold  T 34, T92, AMX 13 57 |

# Programming Fundamentals Mid Exam - 2 November 2019 Group 1

## Biscuits Factory

Create a program that **calculates** how many biscuits your factory can make for a month (**30 days**) and the **percentage** of production compared to another **factory** production.

**First**, you will **receive** the biscuits produced **per day** (**per worker**). After that, you will **receive** the count of the **workers** in your factory. Last, you will receive the **number of biscuits** that the **competing factory produces for 30 days**.

You need to **calculate** the production of your factory for **30 days**. Then you have to **calculate how much more or fewer** biscuits you produce **compared** to the other factory (**in percentage**). There will be no case where the factories will produce **the same amount** of biscuits.

Every **third** **day** the workers produce only **75%** of the usual production. Keep in mind that there can be only a **whole biscuit** after making calculations **for each day** – format them to the **lower number**.

In the end, print the amount of **biscuits** **produced** for **30** days in the following format:

**"You have produced {countBiscuits} biscuits for the past month."**

Then print the percentage of the difference, **formatted** to the **2nd decimal place**, in the following format:

If your production is **bigger** than the other factory:

**"You produce {percentage} percent more biscuits."**

If not:

**"You produce {percentage} percent less biscuits."**

### Input

* On the **first line** you will receive the **amount of biscuits** a worker produce a day – an integer number in the range [**1…200**]
* On the **second line** you will receive the **count of the workers** in your factory – an integer number in the range [**1…1000**]
* On the **third line** you will receive the **amount of biscuits** that the competing factory produces for **30** days – an integer number in the range[**1…2000**]

**NOTE**: The input will always be in the right format.

### Output

* In the end print the amount of biscuits produced for 30 days and the **percentage** of **the difference formatted** to **the 2nd** decimal place in the format described above.

### Constraints

* Percentage **can be** **over** **100**%.
* There will be no case where the factories will produce **the same amount** of biscuits.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 78  8  16000 | You have produced 17160 biscuits for the past month.  You produce 7.25 percent more biscuits. |
| **Comments** | |
| -78 biscuits a day  -8 employees  -17160 biscuit production your factory (keep in mind every **third** **day** the workers produce only **75**% of the usual production)  -17160 – 16000 = 1160 - difference between your and the other factory production  -1160/16000 \* 100 = 7.25% more biscuits. | |
|  | |
| 65  12  26000 | You have produced 21450 biscuits for the past month.  You produce 17.50 percent less biscuits. |
| **Comments** | |
| -65 biscuits a day  -12 employees  -21450 biscuit production your factory  -26000 – 21450 = 4550 - difference between your and the other factory production  -4550/26000 \* 100 = 17.50% more biscuits. | |

## Problem 2. Weaponsmith

*You are a legendary weaponsmith. Heroes from all over the world come to you for the greatest weapons so they can rid the world of the greatest threats. In order for you to craft a weapon you need to assemble its particles.*

You will receive a line with **string** **particles**, separated by "**|**", representing parts of the **name** of a **weapon**. The particles will be in **mixed** order and you can **align** them through the commands, which you will receive on the next **lines,** until you receive the "**Done"** command. They will come the form of strings, separated by **space**. There are **five supported commands:**

* **"Move Left {index}":**
  + Moves the value at **{index}** position to the **Left,** if the **index exist** and the move **is possible.**
  + If movement is not possible, **do nothing.**
* **"Move Right {index}":**
  + Moves the value at **{index}** position to the **Right,** if the **index exists** and the move **is possible.**
  + If movement is not possible, **do nothing.**
* **"Check Even":**
  + **Print** the elements at **even** index positions, **separated by a single space**.
* **"Check Odd":**
  + **Print** the elements at **odd** index positions, **separated by a single space**.

After the "**Done**" command, the weapon name is considered **correct** and you should **print the particles** in their current order **joined together** in the following format**:** "**You crafted {WeaponName}!**"

### Input

* On the **first line**, you will receive **parts** of the given weapon name in a **mixed** order **separated** by **"|"**.
* On the **next lines**, until the **"Done"** command, you will receive **commands** in the **format described** above.

### Output

* Print the needed output upon the **"Done"** command as a string.
* Output should be in the format "**You crafted {Weapon name}!**"

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ha|Do|mm|om|er Move Right 0  Move Left 3  Check Odd  Move Left 2  Move Left 10  Move Left 0  Done | ha mm  You crafted Doomhammer! |
| **Comments** | |
| First, we receive the **"Move Right 0"** command, so we move "**ha"** one position to the right and we get "**Do ha mm om er**".  Then we receive **"Move Left 3"** -> **"Do ha om mm er"**  Checking the current state of the array on odd index positons.  **"Move Left 2"** -> **"Do om ha mm er"**  Invalid index, so we **do nothing**  We can't move the value at zero position to the left, so we **do nothing** Lastly, we have to **print** the collection as one word, so our output is: "**You crafted {Weapon name}!**" | |
|  | |
| ri|As|er|hb|ng  Move Left 1  Move Right 2  Move Right 3  Move Left 2  Done | You crafted Ashbringer! |

## Problem 3. Wizard Poker

*The world is threatened by an enemy never seen before. Your hero's weapon seems to be useless against the enemy. But your hero has a super strong arsenal full of powerful magic cards and will challenge the enemy to a card duel to the death and he needs your help to create a deck.*

Create a program that **adds**, **inserts**, **removes** and **swaps** cards in a new **deck**. On the first line, you will receive all cards in the form of strings **separated** by "**:**"**.** Until you receive the **"Ready"** command, you will get commands in the format:

* **Add {card name}**
  + Adds the card to the **end** of the **deck**.
  + If the card **doesn't exist** in print **"Card not found."**
* **Insert {card name} {index}**
  + Insert the card at the **given** index of the **deck.**
  + If the card **doesn't exist** or the index is invalid print **"Error!"**
* **Remove {card name}** 
  + Remove the card from the **deck**.
  + If the card **doesn't exist** in print **"Card not found."**
* **Swap {card name 1} {card name 2}** 
  + **Swap** the **positions** of the cards.
  + Input will **always be valid**
* **Shuffle deck** 
  + Reverse the **deck**

When you receive the "**Ready"** command print the cards in the deck **separated by space**.

### Input

* **On the 1st line,** you will receive the arsenal of all cards available separated by **":"**.
* **On the next lines,** until you receive the **"Ready" command** you will receive commands to arrange your deck.

### Output

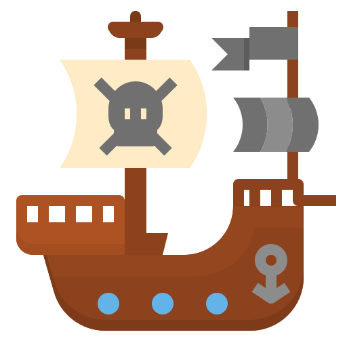
* Print the cards in your deck on a single line, separated by a single space.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Innervate:Moonfire:Pounce:Claw:Wrath:Bite  Add Moonfire  Add Pounce  Add Bite  Add Wrath  Insert Claw 0  Swap Claw Moonfire  Remove Bite  Shuffle deck  Ready | Wrath Pounce Claw Moonfire |
| **Comments** | |
| First command is Add Moofire and now our deck has one card in it.  1. Moonfire Pounce  2. Moonfire Pounce Bite  3. Moonfire Pounce Bite Wrath  4. Claw Moonfire Pounce Bite Wrath  5. Moonfire Claw Pounce Bite Wrath  6. Moonfire Claw Pounce Wrath  7. Wrath Pounce Claw Moonfire | |
|  | |
| Wrath:Pounce:Lifeweaver:Exodia:Aso:Pop  Add Pop  Add Exodia  Add Aso  Remove Wrath  Add SineokBqlDrakon  Shuffle deck  Insert Claw 0  Ready | Card not found.  Card not found.  Error!  Aso Exodia Pop |

# Programming Fundamentals Mid Exam Retake - 6 August 2019

## Black Flag



*Pirates are invading the sea and you're tasked to help them plunder*

Create a program that checks if a **target plunder** is **reached**. First you will receive how many **days** the pirating lasts. Then you will receive how much the pirates **plunder for a day**. Last you will receive the **expected plunder** at the end.

Calculate how much **plunder** the pirates manage to **gather**. Each **day** they gather **plunder**. Keep in mind that every **third day** they attack more ships and they **add additional plunder** to their total gain which is **50% of the daily plunder**. Every **fifth day** the pirates encounter a warship and after the battle they **lose 30%** of their **total plunder**.

If the gained plunder is **more or equal** to the target print the following:

**"Ahoy! {totalPlunder} plunder gained."**

If the gained plunder is **less** than the target. Calculate the **percentage left** and print the following:

**"Collected only {percentage}% of the plunder."**

Both numbers should be **formatted** to the **2nd decimal place**.

### Input

* On the **1st line** you will receive the **days** of the plunder – an **integer number** in the range [0…100000]
* On the **2nd line** you will receive the **daily plunder** – an **integer number** in the range [0…50]
* On the **3rd line** you will receive the **expected plunder** – a **real number** in the range [0.0…10000.0]

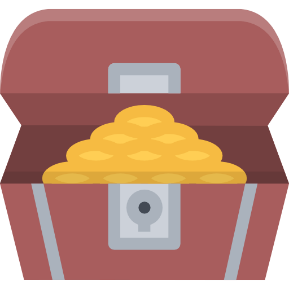
### Output

* In the end print whether the plunder **was successful** or **not** following the format **described above**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 40 100 | Ahoy! 154.00 plunder gained. |
| **Comments** | |
| The days are 5 and the daily plunder is 40. On the third day the total plunder is 120 and since it is a third day, they gain an additional 50% from the daily plunder which adds up to 140. On the fifth day the plunder is 220, but they battle with a warship and lose 30% of the collected cargo and the total becomes 154. That is more than the expected. | |
|  | |
| 10  20  380 | Collected only 36.29% of the plunder. |

## Treasure Hunt



*The pirates need to carry a treasure chest safely back to the ship. Looting along the way.*

Create a program that **manages** the **state** of the **treasure chest** along the way. On the **first line** you will receive the **initial loot** of the treasure chest, which is a **string** of **items** separated by a **'|'.**

**"{loot1}|{loot2}|{loot3}… {lootn}"**

The following lines represent commands **until** **"Yohoho!"** which ends the treasure hunt:

* **Loot {item1} {item2}…{itemn} –** pick up treasure loot along the way. Insert the items at the **beginning** of the chest. If an item is **already** contained **don't** insert it.
* **Drop {index} – remove** the loot at the given **position** and **add** it at the **end** of the treasure chest. If the index is **invalid** skip the command.
* **Steal {count} –** someone steals the **last count** loot items. If there are **less items** than the given count **remove as much** as there are. Print the stolen items separated by **', '**:

**{item1}, {item2}, {item3} … {itemcount}**

In the end output the **average treasure gain** which is the **sum** of all treasure items **length** divided by the **count** of all items inside the chest **formatted** to the **second decimal** point:

**"Average treasure gain: {averageGain} pirate credits."**

If the chest is **empty** print the following message:

**"Failed treasure hunt."**

### Input

* On the **1st line** you are going to receive the **initial treasure chest** (**loot** separated by **'|'**)
* On the next **lines**, until **"Yohoho!"**, you will be receiving commands.

### Output

* Print the output in the **format** **described** **above**.

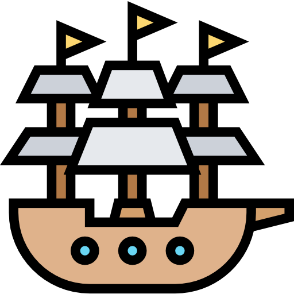
### Constraints

* The **loot items** will be strings containing any ASCII code.
* The **indexes** will be integers in the range [**-200**…**200**]
* The **count** will be an integer in the range [**1**….**100**]

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Gold|Silver|Bronze|Medallion|Cup  Loot Wood Gold Coins  Loot Silver Pistol  Drop 3  Steal 3  Yohoho! | Medallion, Cup, Gold  Average treasure gain: 5.40 pirate credits. |
| **Comments** | |
| The first command **"Loot Wood Gold Coins"** adds **Wood** and **Coins** to the chest but **omits** Gold since it is already contained. The chest now has the following items:  **Coins Wood Gold Silver Bronze Medallion Cup**  The **second** command adds **only Pistol** to the chest  The **third** command **"Drop 3"** removes the **Gold** from the chest, but immediately adds it at the **end**:  **Pistol Coins Wood Silver Bronze Medallion Cup Gold**  The **fourth** command **"Steal 3"** removes the **last 3** items **Medallion**, **Cup**, **Gold** from the chest and prints them.  In the end calculate the average treasure gain which is the sum of all items length Pistol(**6**) + Coins(**5**) + Wood(**4**) + Silver(**6**) + Bronze(**6**) = **27** and **divide** it by the count 27 / 5 = **5.4** and format it to the **second decimal** point. | |
| **Input Output** | |
| Diamonds|Silver|Shotgun|Gold  Loot Silver Medals Coal  Drop -1  Drop 1  Steal 6  Yohoho! | Coal, Diamonds, Silver, Shotgun, Gold, Medals  Failed treasure hunt. |

## Man-O-War



*The pirates encounter a huge Man-O-War at sea.*

Create a program that **tracks** the **battle** and either chooses a **winner** or prints a **stalemate**. On the **first line** you will receive the **status** of the **pirate ship**, which is a **string** representing **integer sections** separated by **'>'**. On **the second line** you will receive the **same** type of status, but for the **warship**:

**"{section1}>{section2}>{section3}… {sectionn}"**

On the **third line** you will receive the **maximum health capacity** a section of the ship can reach.

The following lines represent commands **until** **"Retire"**:

* **Fire {index} {damage} –** the pirate ship **attacks** the warship with the **given damage** at that section. Check if the **index is valid** and if not **skip** the command. If the section **breaks** (health <= 0) the warship **sinks**, print the following and **stop** the program:

**"You won! The enemy ship has sunken."**

* **Defend {startIndex} {endIndex} {damage} -** the warship **attacks** the pirate ship with the **given damage** at that **range** (**indexes are inclusive)**. Check if both **indexes are valid** and if not **skip** the command. If the section **breaks** (health <= 0) the pirate ship **sinks**, print the following and **stop** the program:

**"You lost! The pirate ship has sunken."**

* **Repair {index} {health} -** the crew **repairs** a section of the **pirate ship** with the **given health**. Check if the **index is valid** and if not **skip** the command. The health of the section **cannot** exceed the **maximum health capacity**.
* **Status –** prints the **count** of all sections of the **pirate ship** that need repair soon, which are all sections that are **lower than 20%** of the **maximum** **health capacity**. Print the following:

**"{count} sections need repair."**

In the end if a **stalemate** occurs print the **status** of **both** ships, which is the **sum** of their individual sections in the following format:

**"Pirate ship status: {pirateShipSum}"**

**"Warship status: {warshipSum}"**

### Input

* On the **1st line** you are going to receive the **status** of the **pirate ship** (**integers** separated by **'>'**)
* On the **2nd line** you are going to receive the **status** of the **warship**
* On the **3rd line** you are going receive the **maximum health** a section of a ship can reach.
* On the next **lines**, until **"Retire"**, you will be receiving commands.

### Output

* Print the output in the **format** **described** **above**.

### Constraints

* The **section numbers** will be integers in the range [**1**….**1000**]
* The **indexes** will be integers [**-200**….**200**]
* The **damage** will be an integer in the range [**1**….**1000**]
* The **health** will be an integer in the range [**1**….**1000**]

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 12>13>11>20>66  12>22>33>44>55>32>18  70  Fire 2 11  Fire 8 100  Defend 3 6 11  Defend 0 3 5  Repair 1 33  Status  Retire | 2 sections need repair.  Pirate ship status: 135  Warship status: 205 |
| **Comments** | |
| First, we receive the command "**Fire 2 11**" and damage the warship at section index 2 which is currently 33 and after reduction the status of the warship is the following:  **12 22 22 44 55 32 18**  The **second** and **third** command have **invalid indexes**, so we skip them.  The **fourth** command **"Defend 0 3 5"** damages **4 sections** of the pirate ship with **5** which results in the following status:  12>13>11>20>66  **7 8 6 15 66**  The **fifth** command **"Repair 1 33"** repairs the pirate ship section and adds **33 health** to the current **8** which results in **41**  Only **2 sections** of the pirate ship (**7** and **6**) need repair soon.  In the end there is a **stalemate,** so we print both ship statuses (**sum** of all sections). | |
| **Input Output** | |
| 2>3>4>5>2  6>7>8>9>10>11  20  Status  Fire 2 3  Defend 0 4 11  Repair 3 18  Retire | 3 sections need repair.  You lost! The pirate ship has sunken. |

# Programming Fundamentals Mid Exam - 30 June 2019 Group 2

## Gift Box Coverage

**

Create a program that **calculates** what **percentage** you can cover of a **6-sided gift box (all sides are equal and square)**. **First**, you will **receive** the size of a side. Also, you will **receive** the **sheets** of paper you have. Last, you will receive how much **area** covers a **single sheet** of paper.

First, you need to **calculate** the **area** of the **gift** **box**. Then you have to **calculate how much area** you can cover with the **paper available**. Keep in mind that every **third** **sheet** covers only 25% of the usual area. You have to calculate what **percentage** **of the gift box you’ve covered**. **Percentage can exceed 100%!**

In the end, print the percentage of the area covered, **formatted** to the **2nd decimal place**, in the following format:

**"You can cover {percentage}% of the box."**

### Input

* On the **1st line** you will receive the **size of a side** – a **real number** in the range [0.0…50.0]
* On the **2rd line** you will receive the **number of** **sheets of paper** – an **integer number** in the range [0…1000]
* On the **3th line** you will receive the **area** a **single sheet** of paper **covers** – a **real number** in the range[0.0…50.0]
* The input will always be in the right format.

### Output

* In the end print the **percentage** of **the area covered** **formatted** to **the 2nd** decimal place in the format described above.

### Constraints

* Percentage **can** **be** **over** **100**%.
* All numbers are **centimeters**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 30 4 | You can cover 60.00% of the box. |
| **Comments** | |
| The size of a side is 5. We have 6 sides, so the area is 5 \* 5 \* 6 = 150. 20 of sheets will cover 4 centimeters and 10 – 1 cm. The total area covered is 90, which is 60% of the total area. | |
|  | |
| 2.5 32 4.25 | You can cover 277.67% of the box. |

## Tasks Planner



Create a program that helps you organize your daily tasks. First, you are going to **receive** **the hours each task takes** оn a **single line,** **separated** **by** **space**, in the following **format**:

**"{task1} {task2} {task3}… {taskn}"**

Each task takes **from 1 to 5 hours**. If its time is **set to 0** – it is **completed**. If its time is **set to a negative number** – the task is **dropped**.

Then you will start receiving **commands** until you read the "**End**" message. There are **six** possible commands:

* **"Complete {index}"**
  + Find **the task** on this **index** in your collection and complete it, if the **index** **exists**.
* "**Change {index} {time}**"
  + **Replace** the time needed of the **task** on the given index **with the time given,** if the **index** **exists**.
* "**Drop {index}"**
  + **Drop** the taskon the given **index, setting its hour to -1,** if the **index** **exists**.
* "**Count Completed"**
  + Print the number of **completed** tasks.
* "**Count Incomplete"**
  + Print the number of **incomplete** tasks (this **doesn’t** **include** the **dropped** tasks).
* "**Count Dropped"**
  + Print the number of **dropped** tasks (this **doesn’t** **include** the **incomplete** tasks).

In the end, print the **incomplete** **tasks** on a **single** **line,** separated by a **single** **space** in the following format:

**"{task1} {task2} {task3}… {taskn}"**

### Input

* On the **1st line** you are going to receive the **time of each task**, separated by a single space.
* On the next **lines**, until the **"End"** command is received, you will be receiving commands.

### Output

* Print the tasks in the **format** **described** **above**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 -1 2 3 4 5 Complete 4 Change 0 4 Drop 3  Count Dropped End | 2 4 2 5 |
| **Comments** | |
| First, we receive the command "**Complete 4**" and we to complete the task on index 4. After this command, the task collection looks like this:  **1 -1 2 3 0 5**  1 -1 2 3 4 5  Afterwards, we receive the "**Change 0 4**" command and we need to change the time of the task on index 0. The collection looks like this now:  **4 -1 2 3 0 5**  After, we receive the "**Drop 3**" command, which means we need to drop the task on index 3. The collection looks like this:  **4 -1 2 -1 0 5**  Then, we receive the "**Count Dropped**" command. The result is 2 as we have only 2 dropped tasks.  In the end, we print all of the **incomplete** tasks. This is the result collection:  **4 2 5** | |
|  | |
| 1 2 3 4 5 4 0 3 2 1 Complete 0 Complete 1 Complete 2 Drop 3 Change 4 1  Count Completed End | 4 1 4 3 2 1 |

## Froggy Squad



Create a program that helps you keep track of the **frogs** that are on the riverside. Because you are an extreme animal lover, you also name them. You will **receive** the **names of the frogs** that are already on the riverside on a **single line,** **separated** by a **single** **space** in the following format:

**"{frog1} {frog2} {frog3}… {frogn}"**

Then you will receive commands that describe their action. There are **five** **possible** **commands**:

* **"Join {name}":**
  + A frog **comes** on the riverside and you need to **add** it in the **end** of your **collection**. Frog names will **never repeat**.
* **"Jump {name} {index}"**
  + A frog **jumps** **out** of the water and **joins** the other frogs. You need to **add** it in your **collection** **on the given index, if** the **index** **exists**.
* **"Dive {index}":**
  + The **frog on the given index** has decided to **jump into the water**. You have to **remove** it from your **collection, if** the **index** **exists**.
* **"First/Last {count}":**
  + Print the **first**/**last** **{count}** frogs separated by a **single** **space**. If the **count** requested is **more than the frogs**- just **print them to the end**.

**"{frog} {frog} {frog}"**

* **"Print Normal/Reversed"**
  + **Print** the **names** of the frogs in your **collection** in **normal** (in the order they have been added) or **reversed** **order** in the **format** described **below**, then **stop** **the** **program**:

**"Frogs: {frog1} {frog2}… {frogn}"**

### Input

* On the **1st line**, you will receive the **starting list** with the **names of the frogs** **separated** by a **single space**.
* On the **next lines**, you will receive commands in the **format** **described** above**.**

### Output

* Print the **list after** the **manipulations upon the "Print" command** in the **format** **described** above.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Blake Muggy Kishko Join Kvachko Dive 0  First 10 Print Reversed | Muggy Kishko Kvachko Frogs: Kvachko Kishko Muggy |
| **Comments** | |
| First, we receive the "**Join Kvachko**" command, so we add the frog in the end of the collection. Then, we receive the command "**Dive 0"**, so we remove the frog on index 0. Also, we receive the command "**First 10**", which is more than the frogs we have, so we print all frogs instead. Lastly, we have to **print** the collection in **reversed**, so our output is: "**Frogs: Kvachko Kishko Muggy**"**.** | |
|  | |
| A B C D E F Join G Jump Q 3  Last 3 Dive 2 Print Normal | E F G Frogs: A B Q D E F G |

# Programming Fundamentals Mid Exam - 30 June 2019 Group 1

## Distance Calculator

**

Create a program that **calculates** what **percentage** you’ve travelled. **First**, you will **receive** how many steps you’ve made. Then, you will **receive** how **long** **1 step i**

**s in centimeters**. Last, you will **receive** the **distance** you need to travel in **meters**.

Then you have to **calculate what distance** you have travelled with the **steps given**. Have in mind that every **fifth** **step** is **30% shorter** than usual. You have to calculate what **percentage** **of the distance you’ve travelled**.

In the end, print the percentage of the distance travelled, **formatted** to the **2nd decimal place**, in the following format:

**"You travelled {percentage}% of the distance!"**

### Input

* On the **1st line** you will receive the **steps made** – an **integer number** in the range [0…100000]
* On the **2nd line** you will receive the **length of 1 step** – a **real number** in the range [0.0…50.0]
* On the **3rd line** you will receive the **distance you need to travel** – an **integer number** in the range [0…100000]

### Output

* In the end print the **percentage** of **the distance travelled** **formatted** to **the 2nd** decimal place in the format described above.

### Constraints

* The input will always be in the right format.
* Percentage can be over 100%.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 100 2 1 | You travelled 188.00% of the distance! |
| **Comments** | |
| The length of a step is 2 centimeters. Every fifth step will be 1.4 centimeters long. 20 shorter steps are made. The distance that has to be travelled is 1 meter. The distance travelled is 1.88 meters which is 188% of the distance that had to be travelled. | |
|  | |
| 5000  7.5  500 | You travelled 70.50% of the distance! |

## Number Array



Create a program that helps you keep track of a number array. First, you are going to **receive** **the numbers** оn a **single line,** **separated** **by** **space**, in the following **format**:

**"{number1} {number2} {number3}… {numbern}"**

Then you will start receiving **commands** until you read the "**End**" message. There are **five** possible commands:

* **"Switch {index}"**
  + Find **the number** on this **index** in your collection, if the **index** **exists**, and **switch** its **sign** (negative <-> positive).
* "**Change {index} {value}**"
  + **Replace** the **number** on the given index **with the number given,** if the **index** **exists**.
* "**Sum Negative"**
  + Print the **sum** of **all** **negative** **numbers**.
* "**Sum Positive"**
  + Print the **sum** of **all** **positive** **numbers**.
* "**Sum All"**
  + Print the **sum** of **all** **numbers**.

In the end, print the **positive** **numbers** on a **single** **line, keeping in mind that 0 is positive,** separated by a **single** **space** in the following format:

**"{number1} {number2} {number3}… {numbern}"**

### Input

* On the **1st line** you are going to receive the **numbers of the array** (always **integers**), separated by a single space.
* On the next **lines**, until the **"End"** command is received, you will be receiving commands.

### Output

* Print the tasks in the **format** **described** **above**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 4 5 Switch 4 Change 0 -3 Sum Negative End | -8 2 3 4 |
| **Comments** | |
| First, we receive the command "**Switch 4**" and we make the number on index 4 **negative** (because it is **positive** **before** the **command**). After this command, the task collection looks like this:  **1 2 3 4 -5**  Afterwards, we receive the "**Change 0 -3**" command and we need to change the number on index 0 with the number -3. The collection looks like this now:  **-3 2 3 4 -5**  After that, we receive the "**Sum Negative**" command, which means we need to print the sum of all negative numbers and it is **-8**.  In the end, we print all of the **positive numbers**. This is the result collection:  **2 3 4** | |
|  | |
| 1 2 3 4 5 4 3 2 1 0 Switch -4 Change 13 0 Switch 0 Sum All End | 23 2 3 4 5 4 3 2 1 0 |

## Contact List



Create a program that helps you keep track of the **contacts** that you have. You will **receive** the **list** of **contacts** you already have on a **single line,** **separated** by a **single** **space** in the following format:

**"{contact1} {contact2} {contact3}… {contactn}"**

Then you will receive commands that you need to execute over your list. There are **four** **possible** **commands**:

* **"Add {contact} {index}":**
  + If {**contact**} **isn’t** already **contained** – add it in the **end** of the collection.
  + If {**contact**} **is** already **contained** – add it on the **given** **index, if** the **index exists**.
* **"Remove {index}"**
  + **Remove** the contact on the given index, **if** the **index** **exists**.
* **"Export {startIndex} {count}":**
  + Print the **next {count} contacts starting from the given {startIndex} (including)**, separated by a **single** **space.** If the **count** requested is **more than the contacts**- just **print them to the end**.   
    **"{contact} {contact} {contact}"**
* **"Print Normal/Reversed"**
  + **Print** the contact **list** in **normal** (in the order they have been added) or **reversed order** and then **stop** **the** **program**:

**"Contacts: {contact1} {contact2}… {contactn}"**

### Input

* On the **1st line**, you will receive the **starting list** with the **names of the contacts** **separated** by a **single space**.
* On the **next lines**, you will receive commands in the **format** **described** above**.**

### Output

* Print the needed output upon the **"Export"** command.
* Print the **list after** the **manipulations upon the "Print" command** in the **format** **described** above.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Alisson Bellamy Candace Tristan Remove 3 Add Bellamy 2 Print Normal | Contacts: Alisson Bellamy Bellamy Candace |
| **Comments** | |
| First, we receive the “**Remove 3**” command, so we remove the contact at **index 3** (“**Tristan**”). Then, we receive the command “**Add Bellamy 2**” but we **already have** “**Bellamy**” in our collection, so we **add it on** **index 2**. Lastly, we have to **print** the collection in **normal** order, so our output is: “**Contacts: Alisson Bellamy Bellamy Candace**” | |
|  | |
| Zayn Katy Ariana Avril Nick Mikolas Remove 3 Add Jacob 0 Export 0 3 Export 3 8 Print Reversed | Zayn Katy Ariana Nick Mikolas Jacob  Contacts: Jacob Mikolas Nick Ariana Katy Zayn |

# Technology Fundamentals Retake Mid Exam - 16 April 2019

## Problem 1. Easter Cozonacs

*Since it’s Easter you have decided to make some cozonacs and exchange them for eggs.*

Create a program that **calculates** how much **cozonacs** you can make with the **budget** you **have**. **First**, you will **receive** your **budget**. Then, you will **receive** the **price** for **1 kg flour**. Here is the **recipe** for **one** cozonac:

|  |  |
| --- | --- |
| **Eggs** | **1 pack** |
| **Flour** | **1 kg** |
| **Milk** | **0.250 l** |

The **price for 1 pack of eggs** is **75%** of the **price** **for 1 kg flour**. The **price** for **1l** **milk** is **25%** **more** than price for **1 kg flour**. Notice, that you need **0.250l milk** for **one** cozonac and the calculated price is for **1l**.

**Start** cooking the cozonacs and **keep making** them until you have **enough budget**. Keep in mind that:

* For **every** cozonac that you make, you will receive **3 colored eggs**.
* For **every** **3rd** cozonac that you make, you will lose some of your **colored** eggs **after** you have **received** the usual **3 colored eggs** for your cozonac. The count of eggs you will lose is calculated when you **subtract** **2** from your **current** **count** of **cozonacs** – **({currentCozonacsCount} – 2)**

In the end, print the cozonacs you made, the eggs you have gathered and the money you have **left**, **formatted** to the **2nd decimal place**, in the following format:

**"You made {countOfCozonacs} cozonacs! Now you have {coloredEggs} eggs and {moneyLeft}BGN left."**

### Input / Constraints

* On the **1st line** you will receive the budget – a **real number** in the range [0.0…100000.0]
* On the **2nd line** you will receive the price for **1 kg floor** – a **real number** in the range [0.0…100000.0]
* The input will always be in the right format.
* You will **always** have a **remaining** **budget**.
* There will **not** be a case in which the **eggs** become a **negative** **count**.

### Output

* In the end print the **count** of **cozonacs** you have made, the colored **eggs** you have gathered and the **money** **formatted** to **the 2nd** decimal place in the format described above.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 20.50  1.25 | You made 7 cozonacs! Now you have 16 eggs and 2.45BGN left. |
| **Comments** | |
| We start by calculating the price for a **pack of eggs**, which is **75%** of the price for **1 kg** floor, which in this case is **1.25**. The pack of eggs price is **0.9375**. The price for **1l milk** is **25%** more than the price for **1kg** floor and in this case it is – **1.5625**, but we need the price for 0.250ml, which is - **0.390625**. The total price for one cozonac is:  **1.25** + **0.9375 + 0.390625** = **2.578125**.  And we start subtracting the **price** for a **single** cozonac **from the budget**, and **for every cozonac** we receive **3** eggs. So after the first **subtraction** we will have **17.921875** budget, **1** cozonac and **3** eggs.After the second **- 15.34375** budget, **6** eggs, and on the **third** - **12.765625 budget** and **9 eggs** and since it’s the **third**, we need to **subtract** the **lost eggs**, which will be 3 – 2 = **1**, so we subtract 1 from 9 and our **eggs** become **8**. We continue **subtracting** money from the **budget** until the money **aren't enough** for us to make a cozonac. In the end we have 2.45BGN left. | |
|  | |
| 15.75  1.4 | You made 5 cozonacs! Now you have 14 eggs and 1.31BGN left. |

## Problem 2. Easter Gifts

*As a good friend, you decide to buy presents for your friends.*

Create a program that helps you plan the gifts for your friends and family. First, you are going to **receive** **the gifts** you plan on buying оn a **single line,** **separated** **by** **space**, in the following **format**:

**"{gift1} {gift2} {gift3}… {giftn}"**

Then you will start receiving **commands** until you read the "**No Money**" message. There are **three** possible commands:

* **"OutOfStock {gift}"**
  + Find **the gifts** with **this name** in your collection, **if there are any**, and change their values to "**None**".
* "**Required {gift} {index}**"
  + **Replace** the value of the **current gift** on the given index **with this** **gift,** if the **index** is **valid**.
* "**JustInCase {gift}"**
  + **Replace** the value of your **last** gift **with this** **one**.

In the end, print the **gifts** on a **single** **line**, **except the ones** with value **"None",** separated by a **single** **space** in the following format:

**"{gift1} {gift2} {gift3}… {giftn}"**

### Input / Constraints

* On the **1st line** you are going to receive the **names of the gifts**, separated by a single space.
* On the next **lines**, until the **"No Money"** command is received, you will be receiving commands.
* The **input** will **always** be **valid**.

### Output

* Print the gifts in the **format** **described** **above**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Eggs StuffedAnimal Cozonac Sweets EasterBunny Eggs Clothes  OutOfStock Eggs  Required Spoon 2  JustInCase ChocolateEgg  No Money | StuffedAnimal Spoon Sweets EasterBunny ChocolateEgg |
| **Comments** | |
| First, we receive the command "**OutOfStock**" and we need to replace the values of "**Eggs**" with "**None**". After this command the list should look like this:  **None StuffedAnimal Cozonac Sweets EasterBunny None Clothes**.  Afterwards, we receive the "**Required**" command and we need to replace the value on the 2nd index of our list with the value "**Spoon**". The list should look like this:  **None StuffedAnimal Spoon Sweets EasterBunny None Clothes**  After, we receive the "**JustInCase**" command, which means we need to replace the last value in our list with "**ChocolateEggs**". The list should look like this:  **None StuffedAnimal Spoon Sweets EasterBunny None ChocolateEggs**  In the end, we print all of the gifts, except the ones with values **"None"**. This is the result list:  **StuffedAnimal Spoon Sweets EasterBunny ChocolateEggs** | |
|  | |
| Sweets Cozonac Clothes Flowers Wine Clothes Eggs Clothes  Required Paper 8  OutOfStock Clothes  Required Chocolate 2  JustInCase Hat  OutOfStock Cable  No Money | Sweets Cozonac Chocolate Flowers Wine Eggs Hat |

## Problem 3. Easter Shopping

*You have decided to go on an Easter shopping spree to take advantage of the promotions.*

Create a program that helps you keep track of the **shops** that you want to visit. You will **receive** the **list** of **shops** you have planned on checking out on a **single line,** **separated** by a **single** **space** in the following format:

**"{shop1} {shop2} {shop3}… {shopn}"**

Then you will receive a number – **n** - a **count** of **commands** you need to execute over your list. There are **four** **possible** **commands**:

* **"Include {shop}":**
  + **Add** the shop **at the end of your list.**
* **"Visit {first/last} {numberOfShops}"**
  + **Remove** either the "**first"** or the "**last"** **number of shops from your list**, **depending** on the **input**. If you have **less** **shops** on your list than the **given** **number**, **skip** this command.
* **"Prefer {shopIndex1} {shopIndex2}":**
  + **If** **both** of the **shop indexes** **exist** in your list, take the shops that are on them and **change** **their places**.
* **"Place {shop} {shopIndex}"**
  + **Insert** the **shop** **after** the given **index**, only **if** the **resulted index** **exists**.

In the end **print** the **manipulated list** in the following format:

**"Shops left:**

**{shop1} {shop2}… {shopn}"**

### Input / Constraints

* On the **1st line**, you will receive the **starting list** with the **names of the shops** **separated** by a **single space**.
* On the **2nd line**, you will receive the number of commands - **n – an integer in range [1…100]**
* On the next **n** lines you will be receiving commands in the **format** **described** above.

### Output

* Print the **list after** the **manipulations** in the **format** **described** above.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Bershka CandyStore ThriftShop Armani Groceries ToyStore PeakStore  5  Include HM  Visit first 2  Visit last 1  Prefer 3 1  Place Library 2 | Shops left:  ThriftShop ToyStore Groceries Library Armani PeakStore |
| **Comments** | |
| First we receive the "**Include**" and the name of the store and we **add** the store to our **list**. The list should look like this: **Bershka CandyStore ThriftShop Armani Groceries ToyStore PeakStore HM**  After, we receive the "**Visit**" command and "**first**", which means we have to visit **the first 2 stores**, so we **remove** them from our list and the collection should look like this: **ThriftShop Armani Groceries ToyStore PeakStore HM**. After that, we receive the "**Visit**" command again, but this time we need to visit the "**last**" 1 store, so we **remove** it and the collection should look like this: **ThriftShop Armani Groceries ToyStore PeakStore**. After that we receive the "**Prefer**" command, which means we need to find the shop on the first given index – **3** and change it with the one that is on index – **1**, and the collection should look like this: **ThriftShop ToyStore Groceries Armani PeakStore**. At last, we receive the "**Place**" command and we need to **insert** the shop at the **next** index **after** **2**. And our final list looks like this:  **ThriftShop ToyStore Groceries Library Armani PeakStore** | |
|  | |
| Boutique Flowers CandyStore ThriftShop Versace Groceries ToyStore PeakStore  6  Visit first 9  Visit last 4  Prefer 3 8  Prefer 0 1  Place Store 7  Place ShoeAquarium 2 | Shops left:  Flowers Boutique CandyStore ShoeAquarium ThriftShop |

# Technology Fundamentals Mid Exam - 10 March 2019 Group 2

## The Hunting Games

*A group of friends have decided to participate in a game called "The Hunting Games". The first stage of the game is to gather some supplies. They have a list and your job is to help them follow it and make the needed calculations.*

Write a program that calculates the needed provisions for a quest in the woods.

First you will receive **the days of the adventure**, **the count of the players** and the **group’s energy**. Afterwards, you will receive the following **provisions per day for one person**:

* **Water**
* **Food**

The group calculates how many supplies they’d need for the adventure and take that much water and food.

**Every day** they chop wood and **lose** **a certain amount of energy**. For each of the days, you are going to receive the energy loss from chopping wood. The program should **end** If the energy reaches **0** or **less**.

**Every second day** they **drink water**, which **boosts** their **energy** with **5% of their current energy** and at the same **time drops their water** **supplies** by **30% of their current water.**

**Every** **third** day they **eat**, which **reduces their food supplies by the following amount:**

{currentFood} / {countOfPeople} and at the same time **raises** their group’s **energy** by **10%**.

The chopping of wood, the drinking of water, and the eating happen in the order above.

If they have **enough** **energy** to finish the quest, print the following message:

"You are ready for the quest. You will be left with - {energyLevel} energy!"

If they **run out of energy** print the following message and the **food** and **water** they were left with **before** they ran out of energy:

"You will run out of energy. You will be left with {food} food and {water} water."

### Input / Constraints

* **On the 1st line**, you are going to receive a number **N** - the days of the adventure – **an integer** in the range **[1…100]**
* **On the 2nd line** – the count of players – **an integer** in the range **[0 – 1000]**
* **On the 3rd line** - the group’s energy – **a real number** in the range **[1 - 50000]**
* **On the 4th line** – water per day for one person – **a real number** **[0.00 – 1000.00]**
* **On the 5th line** – food per day for one person – **a real number** **[0.00 – 1000.00]**
* On the next **N** lines – one for each of the days – the amount of **energy loss**– **a real number** in the range **[0.00 - 1000]**
* You will **always** have **enough** **food** and **water**.

### Output

* "You are ready for the quest. You will be left with - {energyLevel} energy!" –   
  if they have enough energy

"You will run out of energy. You will be left with {food} food and {water} water."

* All of the real numbers should be **formatted** to the **second** **digit** after the decimal separator

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 10  7  5035.5  11.3  7.2  942.3  500.57  520.68  540.87  505.99  630.3  784.20  321.21  456.8  330 | You are ready for the quest. You will be left with - 658.72 energy! |
| **Comments** | |
| The **days** are **10** and the **players** are **7**. The **energy** of the whole **group** is **5035.5**. We receive the **water** and **food** and we can **calculate** the needed amount of both for the whole quest:  **10 \* 7 \* 11.3 – total water = 791**  **10 \* 7 \* 7.2 – total food = 504**  Afterwards, for **each** of the **days** you have to calculate the energy loss. On each day you receive energy loss and you have to **subtract** it. On the first day it is:  **5035.5 – 942.3 = 4093.2**  On **every second day** we **add** the **energy boost** from the drank water, which is **5% of the current energy** and **subtract** the amount **from the total water**. The first time we reach a second day, the energy will become **3772.26** and the water will become **553.7**. The first time we reach a **third** **day**, we have to **boost the energy with 10%** and **reduce** the food supplies and the energy will become - **3576.74** and the food **432**. Make all of the calculations and in the end, you must have **658.77** energy left and **132.94** water and **317.39** food left. | |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 12  6  4430  9.8  5.5  620.3  840.2  960.1  220  340  674  365  345.5  212  412.12  258  496 | You will run out of energy. You will be left with 229.17 food and 118.59 water. |

## Seize the Fire

*The group of adventurists have gone on their first task. Now they have to walk through fire - literally. They have to use all of the water they have left. Your task is to help them survive.*

Create a program that calculates the water that is needed to put out a "fire cell", based on the given information about its "fire level" and how much it gets affected by water.

First, you will be given **the level of fire** inside the cell with the **integer** **value** of the **cell**, which represents the needed water to put out the fire. They will be given in the following format:

**"{typeOfFire} = {valueOfCell}#{typeOfFire} = {valueOfCell}#{typeOfFire} = {valueOfCell}……**"

Afterwards you will receive the **amount of water** you have for putting out the fires. There is a **range** of fire for each of the fire types, and if a cell’s value is below or exceeds it, it is invalid and you don’t need to put it out.

|  |  |
| --- | --- |
| **Type of Fire** | **Range** |
| High | 81 - 125 |
| Medium | 51 - 80 |
| Low | 1 - 50 |

If a cell is valid, you have to put it out by reducing the water with its value. Putting out fire also takes **effort** and you need to **calculate it**. Its value is equal to **25% of the cell’s value**. In the end you will have to print the **total effort**. Keep putting out cells until you run out of water. If you **don’t have enough** **water** to put out a given cell – **skip it** and **try the next one**. In the end, **print the cells you have put out** in the following format:

**"Cells:**

**- {cell1}**

**- {cell2}**

**- {cell3}**

**……**

**- {cellN}"**

**"Effort: {effort}"**

In the end, print the total fire you have put out from all of the cells in the following format: "Total Fire: {totalFire}"

### Input / Constraints

* **On the 1st line** you are going to receive the **fires with their cells** in the format described above **– integer numbers in the range [1…500]**
* **On the 2nd line**, you are going to be given the **water** – **an integer number** in the range **[0….100000]**

### Output

* Print the cells, which you have put out in the following format:

"Cells:

- {cell}

- {cell2}

- {cell3}

- {cell5}

……

- {cellN}"

* Print the effort, rounded 2 digits after the decimal separator in the following format:

"Effort: {effort}"

* Print the total fire put out

"Total Fire: {totalFire}"

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| High = 89#Low = 28#Medium = 77#Low = 23  1250 | Cells:  - 89  - 28  - 77  - 23  Effort: 54.25  Total Fire: 217 |
| **Comments** | |
| After reading the output, we start **checking** the **level of the fire** and its validity. The first is valid, so we **subtract the 89** from the amount of **water** – 1250, and the water becomes 1161. We need to calculate the **effort**, which is **25%** of 89. We will **add 89 to the total fire** we have put out. In the end the effort is 54.22 and the total fire: 217 | |

|  |  |
| --- | --- |
| **Input** | **Output** |
| High = 150#Low = 55#Medium = 86#Low = 40#High = 110#Medium = 77  220 | Cells:  - 40  - 110  Effort: 37.50  Total Fire: 150 |

## The Final Quest

*After walking through fire, the group has reached the final step of the quest. They have received a list with instructions that will help them resolve the last riddle that will lead them to the truth about the Hunting Games.*

Create a program that **follows** given **instructions**. You will receive a **collection of words on a single line**, split by a **single space**. They are not what they are supposed to be, so you have to **follow the instructions** in order to find the **real message**. You will be receiving commands. Here are the possible ones:

* Delete {index} – removes the word **after** the given index **if it is valid**.
* Swap {word1} {word2} – find the given words in the collections **if they exist** and **swap** their places.
* Put {word} {index} – **add** a word at the **previous place {index} before** the   
  given one, **if it is valid**. Note: putting at the last index simply appends the word to the end of the list.
* Sort – you must **sort** the words in descending order.
* Replace {word1} {word2}– find the **second word** **{word2}** in the collection (**if it exists**) and **replace** it with the **first word – {word1}**.

Follow them until you receive the "**Stop**" command. After you have successfully **followed the instructions**, you must print the words on a single line, split by a space.

### Input / Constraints

* **On the 1st line**, you are going to receive the collection of words, split by a single space – **strings**
* **On the next lines**, you are going to receive **commands**, until you receive the "**Stop**" command

### Output

* Print the words you have gathered on a single line, split by a single space

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Congratulations! You last also through the have challenge!  Swap have last  Replace made have  Delete 2  Put it 4  Stop | Congratulations! You made it through the last challenge! |
| **Comments** | |
| First, we receive the command **“Swap”,** so we change the positions of the words **have** and **last**. The text at this point should look like this:  **Congratulations! You have also through the last challenge!**  After that, we receive **“Replace”** and we have to replace the **second word** – “have” **with the first** – “made”. Afterwards we have to **delete** the **word**, which is **after the second index**. And finally, we have to put a word on the **previous** position **before** **4**. | |

|  |  |
| --- | --- |
| **Input** | **Output** |
| This the my quest! final  Put is 2  Swap final quest!  Delete 2  Stop | This is the final quest! |

# Technology Fundamentals Mid Exam - 10 March 2019 Group 1

## Spring Vacation Trip

*A group of friends decide to go to a trip for a few days during spring vacation. They have a certain budget. Your task is to calculate their expenses during the trip and find out if they are going to have enough money to finish the vacation.*

Create a program that calculates travelling expenses by entering the following information:

* **Days of the vacation**
* **Budget** - itsfor the whole group
* **The count of people**
* **Fuel per kilometer – the price for fuel** that their car consumes **per kilometer**
* **Food expenses per person**
* **Hotel room price for one night** – again, for **one person**

If the group **is bigger than 10**, they receive a **25% discount** from the **total hotel expenses**.

**Every day**, they **travel** some **distance** and you have to **calculate the expenses** for the **travelled kilometers**.

Every **third** and **fifth** day, they have some additional expenses, which are **40% of the current value of the expenses**.

Every **seventh** day, their expenses are reduced, because they **withdraw (receive)** a small amount of money – you can calculate it by **dividing the amount of the current expenses** by the **group of people**.

If the **expenses** **exceed the budget at some point**, stop calculating and print the following message:

"Not enough money to continue the trip"

If the **budget is enough**:

"You have reached the destination. You have {money}$ budget left."

**Print** the result formatted **2 digits** after the decimal separator.

### Input / Constraints

* **On the 1st line**, you are going to receive the days of the trip – **an integer** in the range **[1…100]**
* **On the 2nd line** – the budget – **a real number** in the range **[0.00 – 1000000.00]**
* **On the 3rd line** - the group of people – **an integer** in the rang **[1 - 50]**
* **On the 4th line** – the price for fuel per kilometer – **a real number** **[0.00 – 20.00]**
* **On the 5th line** – food expenses per person for a day – **a real number** **[0.00 – 50.00]**
* **On the 6th line** – the price for a room for one night per person – **a real number** **[0.00 – 1000.00]**
* On the next **n** lines – one for each of the days – the **travelled** **distance** in kilometers per day– **a real number** in the range **[0.00 - 1000]**

### Output

* "Not enough money to continue the trip. You need {money}$ more." –   
  if the budget is not enough
* "You have reached the destination. You have {money}$ budget left." – if it’s enough.

Print the result formatted **2 digits** after the decimal separator.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| **7**  **12000**  **5**  **1.5**  **10**  **20**  512  318  202  154  222  108  123 | You have reached the destination. You have 5083.48$ budget left. | We receive the days of the vacation, the budget, the group, the consumed fuel per kilometer, the food expenses and the price for a hotel room for one night. We must calclate the food expenses: **10 \* 5 \* 7 = 350**  And the price for the hotel for all of the nights:  **20 \* 5 \* 7 = 700**  The curent expenses are **1050**. For each day, we have to calculate the consumed fuel – **{travelledDistance} \* 1.5**  On every **3rd**  and **5th**  day we add the additional expenses:  **0.4 \* {currentExpenses}**  On every **7th** day, they **receive** some **money**:  **{currentExpense} / {groupOfPeople}**  After **we have added** the **fuel expenses for each day** and made the **other calculations**, the **expenses** have reached **8645.652**. When we divide them by **the group (5),** the result is **1729.1304**, so we have to **reduce the expenses** by that sum. The expenses become **6916.5216**. The budget is **enough**, so the result is:  "You have reached the destination. You have 5083.48$ budget left." |
| **10**  **20500**  **11**  **1.2**  **8**  **13**  100  150  500  400  600  130  300  350  200  300 | Not enough money to continue the trip. You need 465.79$ more. |  |

## Hello, France

*The budget was enough to get them to Frankfurt and they have some money left, but their final aim is to go to France, which means that they will need more finances. They’ve decided to make profit by buying items on discount from the Thrift Shop and selling them for a higher price. You must help them.*

Create a program that calculates the profit after buying some items and selling them on a higher price. In order to fulfil that, you are going to need certain data - you will receive a **collection of items** and a **budget** in the following format:

{type->price|type->price|type->price……|type->price}

{budget}

**The prices** for each of the types **cannot** **exceed** a certain **price**, which is given bellow:

|  |  |
| --- | --- |
| **Type** | **Maximum Price** |
| Clothes | 50.00 |
| Shoes | 35.00 |
| Accessories | 20.50 |

If a **price** for a certain **item** is **higher than** the **maximum** price, **don’t buy it**. Every time you **buy an item**, you have to **reduce the budget** with the value of **its** **price**. If you don’t have enough money for it, you **can’t buy it**. Buy **as much** items **as you can**.

You have to **increase** the price of **each of the items you have successfully bought with 40%.** Print the list with **the new prices** and **the profit** you will gain **from selling the items**. They need exactly **150$** for tickets for the train, so if their budget after selling the products is enough – print – "Hello, France!" and if not – "Time to go."

### Input / Constraints

* **On the 1st line** you are going to receive the **items with their prices** in the format described above **– real numbers in the range [0.00……1000.00]**
* **On the 2nd line**, you are going to be given the **budget** – **a real number** in the range **[0.0….1000.0]**

### Output

* Print the list with the bought item’s new prices, rounded 2 digits after the decimal separator in the following format:

"{price1} {price2} {price3} {price5}………{priceN}"

* Print the profit, **rounded 2 digits** after the decimal separator in the following format:

"Profit: {profit}"

* If the money for tickets are enough, print: "Hello, France!" and if not – "Time to go."

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| Clothes->43.30|Shoes->25.25|Clothes->36.52|Clothes->20.90|Accessories->15.60  120 | 60.62 35.35 51.13  Profit: 42.03  Hello, France! | We start subtracting the valid prices from the budget:  120 – 43.40 = **76.7.**  76.7 – 25.25 = **51.45**  51.45 – 36.52 = **14.93**  14.93 is **less** than **20.90** and **15.60**, so we can’t buy either of the last two. We must increase **each price** with 40% and the new prices are: **60.62 35.35 51.13.** The profit is **42.03** and their new budget will be – what is left of the budget - **14.93 + {sum of all newPrices}.** It is enough, so we print: **Hello, France!** |
| Shoes->41.20|Clothes->20.30|Accessories->40|Shoes->15.60|Shoes->33.30|Clothes->48.60  90 | 28.42 21.84 46.62  Profit: 27.68  Time to go. |  |

## Last Stop

*The group has reached Paris and went to visit "La Louvre". They accidently found a map behind "The Wedding at Canna" painting. It had some instructions, so they have decided to follow them and see where they will lead them. Your job is to help them.*

Create a program that follows instructions in order to fulfil a quest. First, you will receive a collection of numbers – each **representing** a **painting number**. After that, you are going to be receiving **instructions**, until the "**END**" command is given.

* Change {paintingNumber} {changedNumber} – find the painting with the first number in the collection (**if it exists**) and **change** its **number** with the **second number –** {changedNumber}.
* Hide {paintingNumber} – find the painting with this value and **if it exists** and hide it (**remove** it).
* Switch {paintingNumber} {paintingNumber2} – find the given paintings in the collections **if they exist** and **switch** their places.
* Insert {place} {paintingNumber} – **insert** the painting (**paintingNumber**) **on the next place after** the givenone, **if it exists**.
* Reverse – you must **reverse** the **order** of the paintings.

Once you complete the instructions, print the numbers of the paintings on a single line, split by a space.

### Input / Constraints

* **On the 1st line**, you are going to receive the numbers of the paintings, split by a single space – **integer numbers** in the range **[1…1000]**
* **On the next lines**, you are going to receive **commands**, until you receive the "**END**" command

### Output

* Print the message you have received after the conversion of all numbers on a single line

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 115 115 101 114 73 111 116 75  Insert 5 114  Switch 116 73  Hide 75  Reverse  Change 73 70  Insert 10 85  END | 70 114 111 116 114 101 115 115 | The first command is "**Insert**". You have to insert **painting number 114** at the next index after the 5th:  115 115 101 114 73 111 **114** 116 75  The "**Switch**" will switch number **116** with **73** and the collection should look like this:  115 115 101 114 **116** 111 114 **73**  75  After receiving the"**Hide**" **command**, you must remove **75**. After that you receive "**Reverse**" and you have to reverse the whole collection. By receiving "**Change**" you have to exchange the value **73** with the value – **70.** The next "**Insert**"command is **invalid**, because there is **no 11th index** in the collection. |
| 77 120 115 101 101 97 78 88 112 111 108 101 111 110  Insert 5 32  Switch 97 78  Hide 88  Change 120 117  END | 77 117 115 101 101 78 32 97 112 111 108 101 111 110 |  |