# CoE 164

Computing Platforms

## Assessments Week 02

Academic Period: 2nd Semester AY 2022-2023
Workload: 3 hours
Synopsis: Rust functions and data structures

## SE Week 02A

This assessment will let you be familiar with functions and strings in Rust.

**This is worth 40% of your grade for this week**.

### Problem Statement


In internet speech, a mocking sentence is written by alternately or randomly using uppercase and lowercase letters throughout the sentence. It has been popularized through a meme from the Spongebob cartoon series, and is used to mimic somebody's statement in a mocking fashion. However, several years before, the Philippines used a similar writing style as a text speech in a sociolect attributed to the *jejemon* phenomenon. Its usage is more of marking oneself as part of the phenomenon and does not carry any mocking tone that we know of today.

You are working as an intern for a PG-rated video sharing site, and the company seniors wanted to post-process the comments in a way that makes it PG-friendly. You are tasked to "demock" paragraphs by making sure that each sentence (i.e. sequence of words that ends with a period) starts with an uppercase letter and the remaining letters are all lowercase letters.

### Input

The input to the program starts with a number $T$ denoting the number of testcases. $T$ lines then follow, with each line denoting a paragraph of comments $S$ to demock.

For this problem, a *word* is a sequence of contiguous characters without a space in between. A *sentence* is a sequence of space-separated words with the last word's last character being a period. A *paragraph* is a sequence of sentences on a single line.

## Output

The output of the program consists of $T$ lines. Each line should be of the format `Case #T_i: S_{democked}` starting with $T_i = 1$. $S_{democked}$ should not have any leading or trailing spaces.

## Constraints

*Input Constraints*

$S$ consists *only* of the following characters:

- uppercase letters A-Z
- lowercase letters a-z
- numbers 0-9
- space
- comma, period, semicolon, colon, apostrophe, hyphen, underscore

$0 < |S| \leq 1000$

You can assume that all of the inputs are well-formed and are always provided within these constraints. You are not required to handle any errors.

*Functional Constraints*

You are **required** to write a function named **`democker(in_text: &String) -> String`** which will return the democked version of `in_text` once the function finishes running. You are also **not allowed** to add other functions in the code aside from `democker(in_text: &String) -> String` and `main()`.

Failure to follow these functional constraints will mark your code with a score of zero.

## Sample Input/Output

*Sample Input 1:*

| **Input** | **Output** |
|---|---|
| 3<br>ThE qUICK BROwN FOX jUMPs OVeR THE lazy DoG. The qUick BrOwn fOX JuMPs oveR the LaZy DOg.<br>In IntERNEt speEcH, A MocKINg senteNCE IS WrItTen By aLtErnatEly Or RAnDomlY USIng UppercAse and LoweRCAsE LEttErs thRoughoUt ThE sEntENCe. It Has BEen populARIzEd THROugh a Meme | Case #1: The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.<br>Case #2: In internet speech, a mocking sentence is written by alternately or randomly using uppercase and lowercase letters throughout the sentence. It has been popularized through a meme from the spongebob cartoon |

| | |
|---|---|
| frOM thE SPOngeBOb caRtOon SERiES, AnD Is USed tO MIMiC SOMEBOdY's STatEMeNt In A mockIng faSHIon. HoWevEr, sevEral YeArS BefORE, the PHilippinEs uSed a sIMilar wRITINg StYlE As a TEXt sPEech in a sOcIolECt atTRiBuTeD to The _jejeMon_ pHenomeNON. its usaGe IS MORE of MArkiNg oneSElF AS paRt of tHE pheNOmeNOn ANd DoeS NOt Carry any mockIng Tone tHAt we knoW Of TODaY.<br>feicHANg GaOXIng DI xIANG ZHOngguO gUOjIA ZHUXi xIJInpIng ZHONgGUO ZHu FeILvBiN dAShi HuAnGXILiAn HE quAnTI zhoNggUoREnmiN ZHIYi ReLIE zhUHE QiNGzHu zhONGhUARenmINGoNgHeGuO chENGLI qIShIsAn ZHOuNIan ZHU qUAntI zhongGUORENMiN GUoQING KUAIlE. | series, and is used to mimic somebody's statement in a mocking fashion. However, several years before, the philippines used a similar writing style as a text speech in a sociolect attributed to the _jejemon_ phenomenon. Its usage is more of marking oneself as part of the phenomenon and does not carry any mocking tone that we know of today.<br>Case #3: Feichang gaoxing di xiang zhongguo guojia zhuxi xijinping zhongguo zhu feilvbin dashi huangxilian he quanti zhongguorenmin zhiyi relie zhuhe qingzhu zhonghuarenmingongheguo chengli qishisan zhounian zhu quanti zhongguorenmin guoqing kuaile. |

## Steps

1. Write your program in Rust. Make sure to accept input via standard input and print your output via standard output.
2. Submit a copy of the source code to the Week 02A submission bin. Make sure that you attach one (1) file in the bin containing the Rust source code with a .rs extension or .rs.txt extension (if UVLe doesn't accept .rs files).

## SE Week 02B

This assessment will let you be familiar with structs and hashmaps in Rust.

**This is worth 30% of your grade for this week**.

### Problem Statement

You are working as a records assistant in the university tasked with safekeeping the student, faculty, and staff "rolls". With the university existing for a very long time, there are some records that are still kept in record books, which you need to digitize. In the time being, you will only encode the names, degrees, and student numbers of the students written in these books.

The first module of the digital record books is the record finder. A user will enter the student number of a person and the module will return the name and degree of the student. It is also capable of updating some information in case any of these may change. Right now, only the degree of the student can be changed.

### Input

The input starts with a number $S$ on a single line denoting the number of students to save into the module. $S$ lines then follow, with each block of three lines denoting a student. The first line of the block contains the 9-digit student number, the second line contains the name of the student, and the third line contains the degree.

After the block of student information is a list of commands. The first line of this command block contains a number $C$ denoting the number of commands that will follow. $C$ lines then follow, denoting a command. A command can be written in any of the two formats:

- `g <sn>` - fetch a record with the given student number `sn`.
- `u <sn> <update_cmd>` - update a record with the given student number `sn`.
    - `update_cmd` can only be from one of the following command:
        - `d <degree>` - update the degree of the student in the record with `degree`

### Output

The output consists of $C$ blocks of lines, with each block $R_i$ denoting the result of executing the $C_i$th command. The $R_i$th block should contain the following depending on what the $C_i$th command is. Please see the sample input(s) for reference.

- `g <sn>`

    - Two or three lines containing information of the student with the student number `sn`. The information is indented four spaces to the right.
    - If `sn` exists in the records:
        - `Get <sn>`
          `    Name: <name>`
          `    Degree: <degree>`
        - `name` and `degree` should be printed *as is* (i.e. how it was entered in the records)
    - If `sn` *does not exist* in the records:
        - `Get <sn>`
          `    Does not exist`
- `u <sn> <update_cmd>`
    - Two lines denoting whether the update was successful or not. The update status is indented four spaces to the right.
    - If a student with student number `sn` exists in the records:
        - `Update <sn> <update_cmd>`
          `    Success`
    - If `sn` does not exist in the records:
        - `Update <sn> <update_cmd>`
          `    Does not exist`
    - `update_cmd` should be one of the following depending on the update command received:
        - `d` - `Degree`

## Constraints

*Input Constraints*

$0 < S, C \leq 100$

`sn` always consists of nine digits.

`name` and `degree` will contain only alphabetical characters and spaces. There will be no leading and trailing spaces in them

You can assume that all of the inputs are well-formed and are always provided within these constraints. You are not required to handle any errors.

*Functional Constraints*

You are **required** to create and use a struct named `StudentRecords` with fields `name: String`, `degree: String`, and `sn: u32`.

Failure to follow these functional constraints will mark your code with a score of zero.

## Sample Input/Output

*Sample Input 1:*

| **Input** | **Output** |
| --- | --- |
| 3<br>201003243<br>Mark Reyes<br>BS Computer Engineering<br>200594938<br>John Asis<br>BS Computer Engineering<br>200648392<br>Milo Garcia<br>BS Computer Engineering<br>8<br>g 201003243<br>g 201129343<br>u 200648392 d BS Electronics Engineering<br>u 201129343 d BS Electrical Engineering<br>u 201003243 d MS Electrical Engineering<br>g 201003243<br>g 200594938<br>g 200648392 | Get 201003243<br>    Name: Mark Reyes<br>    Degree: BS Computer Engineering<br>Get 201129343<br>    Does not exist<br>Update 200648392 Degree<br>    Success<br>Update 201129343 Degree<br>    Does not exist<br>Update 201003243 Degree<br>    Success<br>Get 201003243<br>    Name: Mark Reyes<br>    Degree: MS Electrical Engineering<br>Get 200594938<br>    Name: John Asis<br>    Degree: BS Electronics Engineering<br>Get 200648392<br>    Name: Milo Garcia<br>    Degree: BS Electrical Engineering |

*Sample Input 2:*

| **Input** | **Output** |
| --- | --- |
| 1<br>293849200<br>abcdeF<br>sSDed ndies<br>3<br>g 293849200<br>u 293849200 d abcdeF<br>g 293849200 | Get 293849200<br>    Name: abcdeF<br>    Degree: sSDed ndies<br>Update 293849200 Degree<br>    Success<br>Get 293849200<br>    Name: abcdeF<br>    Degree: abcdeF |

## Steps

1. Write your program in Rust. Make sure to accept input via standard input and print your output via standard output.
2. Submit a copy of the source code to the Week 02B submission bin. Make sure that you attach one (1) file in the bin containing the Rust source code with a .rs extension or .rs.txt extension (if UVLe doesn't accept .rs files).

## SE Week 02C

This assessment will let you be familiar with enums and vectors in Rust.

**This is worth 30% of your grade for this week**.

### Problem Statement

There is an increase in the use of smartwatches and fitness trackers in the past five years. Mainly due to the increase of health-conscious users, smartphone companies have jumped ship to create a new line of "wearables" designed to track biometrics and offer data-driven fitness advice to users aside from still being the quintessential wristwatch. One of the perils of using such devices is that data is usually stored in the "cloud" - i.e. the servers of the company who manufactured the watch. As more people are also being conscious of how and where their personal data is stored, you have decided to create an open-source fitness tracker that lets users locally store their fitness data on their smartphone without transmitting it via the internet.

A fitness tracker cannot be one without a way to keep a "diary" on which fitness-related activity the user did during the course of a day. This is the first functionality you have decided to work on for your open-source project. Most trackers have a preset number of exercises it can track, but you have decided to track only three activities in the meantime - walking, running, and biking. You have also decided to track their sleep, eat, and idle times, which you would collectively call "rest" times.

In the meantime, the tracker will keep track of whether the user has achieved their sleep and exercise goals and the maximum contiguous duration of their "exercise" and "rest" activities throughout the day. The user sets the "goals" as the minimum cumulative time for them to exercise or sleep during the day and it is said that they have surpassed it if they have a cumulative time of at least that goal. On the other hand, an exercise or rest duration is said to be contiguous if two or more consecutive activities are exercise- or rest-related activities.

### Input

The input starts with a number $T$ on a single line denoting the number of days. $T$ blocks then follow containing a list of activities done during the day. The first line of each block contains two numbers $E$ $S$ denoting the exercise and sleep goals in minutes, respectively. Then, the next line will contain a number $A$ denoting the number of activities. It will then be followed by $A$ lines, with each line written in the format $c$ $T$. $c$ is a lowercase letter denoting the activity the user did (with the corresponding valid values shown in the table below) and $T$ the duration of the activity.

| Prefix | Activity | Type |
|---|---|---|

| | | |
|---|---|---|
| r | running | exercise |
| w | walking | exercise |
| b | biking | exercise |
| s | sleeping | rest |
| e | eating | rest |
| i | idle | rest |

## Output

The output consists of $T$ blocks of lines, with each block containing five lines. The first line of a block should contain a string of the format $Case\ \#t$: where $t$ is the serial of the test case starting from 1. The next two lines show whether the exercise or sleep goals have been reached, respectively. The last two lines show the maximum contiguous duration of their exercise and rest periods, respectively.

## Constraints

*Input Constraints*

$0 < T, A \leq 100$

$0 \leq E, S \leq 1440$

$c \in$ {r, w, b, s, e, i}

You can assume that all of the inputs are well-formed and are always provided within these constraints. You are not required to handle any errors.

*Functional Constraints*

You are **required** to create and use an enum named `FitnessActivity` containing the six possible activities as subtypes: `Walk(t: i64)`, `Run(t: i64)`, `Bike(t: i64)`, `Eat(t: i64)`, `Sleep(t: i64)`, and `Idle(t: i64)`.

Failure to follow these functional constraints will mark your code with a score of zero.

## Sample Input/Output

*Sample Input 1:*

| **Input** | **Output** |
|---|---|
| `3`<br>`60 480`<br>`10`<br>`s 480` | `Case #1:`<br>`Exercise goal reached`<br>`Sleep goal reached`<br>`Max exercise duration: 25m` |

| | |
|---|---|
| w 15<br>i 30<br>r 10<br>i 180<br>e 60<br>i 300<br>w 25<br>i 30<br>w 15<br>0 0<br>1<br>i 180<br>60 480<br>3<br>s 60<br>w 15<br>s 60 | Max rest duration: 480m<br>Case #2:<br>Exercise goal reached<br>Sleep goal reached<br>Max exercise duration: 0m<br>Max rest duration: 180m<br>Case #3:<br>Exercise goal not reached<br>Sleep goal not reached<br>Max exercise duration: 15m<br>Max rest duration: 60m |

## Steps

1. Write your program in Rust. Make sure to accept input via standard input and print your output via standard output.
2. Submit a copy of the source code to the Week 02C submission bin. Make sure that you attach one (1) file in the bin containing the Rust source code with a .rs extension or .rs.txt extension (if UVLe doesn't accept .rs files).