CoE 164

Computing Platforms

Assessments Week 03

Academic Period: 2nd Semester AY 2023-2024
Workload: 3 hours
Synopsis: Rust collections and generics

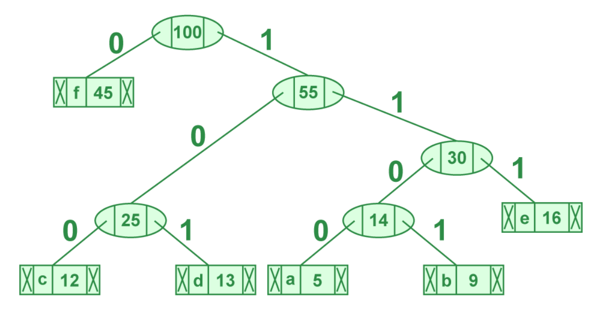
SE Week 03A

This assessment will help you become familiar with vectors and hashmaps.

This is worth 40% of your grade for this week

Problem Statement

A Teaser to Huffman Coding.



Huffman Coding is a widely used compression algorithm in information theory and computer science. It is a variable-length coding technique that efficiently represents data with shorter codes assigned to more frequent symbols and longer codes to less frequent ones. Thus, reducing the overall size of data, making it an essential tool in data compression applications, such as file compression and transmission of digital information.

Interested in building your Huffman coding function, you start with creating an algorithm that extracts the occurrences of each letter in a given text while being case-insensitive using hashmaps.

Input

The input to the program will start with a number T denoting the number of testcases. It will then be followed by T lines containing a sequence of contiguous characters W with a space in between each “word,” which is not necessarily found in the English dictionary.

Output

The output of the program will consist of T_i blocks which indicates the frequency of a given letter and the space character. Each block should be of the format:

`---LETTER FREQUENCY of CASE #T_{i}---` (surrounded by 3 dashes)

```
a: <frequency>
b: <frequency>
c: <frequency>
...
z: <frequency>
` `: <frequency>
```

`---------------------------------` (33 dashes in total)

If `<frequency>` is 0, then the corresponding character **must be skipped and not printed.**

Constraints

Input Constraints

$T \leq 10$

W consists *only* of the following characters:

- uppercase letters A-Z
- lowercase letters a-z
- space

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 the following `struct` and `impl`:

- `LetterFreq` - `struct` representing the number of times a character has been seen.
  - `dictionary: HashMap<char, u64>`

You are **required** to have the following function signatures, their arguments in order, and their return values:

- `LetterFreq::new()` - initialize a new hashmap with default values
  - Arguments
    - None
  - Return value - `LetterFreq` representing an empty HashMap.
- `LetterFreq::count()` - count the occurrence of a given value
  - Arguments
    - `&mut self` - the "self" instance of the struct
    - `input: char` - the character to be counted or updated
- `LetterFreq::current_counter()` - prints the frequencies of each letter
  - Arguments
    - `&mut self` - the "self" instance of the struct
    - `input: char` - the character of interest

    - Return value - `u64` representing the number of occurrences of the input character.
- `main()` - entry point to the program
    - Arguments
        - None
    - Return value
        - None
    - Additional constraints
        - Input and output parsing should be done here.

Failure to follow these functional constraints will result in a score of zero in this assessment.

Sample Input/Output[1]

Sample Input 1:

```
3
Oh wow Hev Abi u a listener of them din pala haha I too ay isang
listener ng Hev Abi haha Napakinggan mo na ba whole album niya
Kung alam mo lang Im in awe na we both like Hev abi haha\n
What HAPPEN bella whY are you crying again I know vampire right
vamFIRE WILL FEYT TO ME i trusted to protect bella but you did
not I will sure yoU die EDWARD\n
Lorem ipsum dolor amet consectetur adipiscing elit sed eiusmod
tempor incididunt ut labore et dolore magna aliqua Never gonna
give you up ut enim minim veniam quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat Never
gonna give you up duis aute irure dolor reprehenderit voluptate
velit esse cillum dolore eu fugiat nulla pariatur Never gonna
give you up excepteur sint occaecat cupidatat non proident sunt
culpa qui officia deserunt mollit anim id est laborum\n
```

Sample Output 1:

```
---LETTER FREQUENCY of CASE #1---
a: 26
b: 6
d: 1
e: 12
f: 1
g: 6
h: 13
i: 13
k: 3
l: 8
m: 6
n: 14
o: 9
p: 2
r: 2
s: 3
t: 5
u: 3
```

[1] “\n” indicates a new input.

```
v: 3
w: 5
y: 2
 : 41
---------------------------------
---LETTER FREQUENCY of CASE #2---
a: 10
b: 3
c: 2
d: 6
e: 13
f: 2
g: 3
h: 4
i: 12
k: 1
l: 8
m: 3
n: 5
o: 8
p: 4
r: 9
s: 2
t: 11
u: 6
v: 2
w: 6
y: 6
 : 31
---------------------------------
---LETTER FREQUENCY of CASE #3---
a: 31
b: 3
c: 16
d: 17
e: 47
f: 3
g: 9
h: 1
i: 41
l: 22
m: 17
n: 30
o: 34
p: 14
q: 5
r: 25
s: 17
t: 31
u: 35
v: 9
x: 3
y: 3
 : 77
---------------------------------
```

Steps

1. Write your program in Rust. Compile and make sure that there are no syntax errors.
2. Make sure to accept input via standard input and print your output via standard output. For example, you can write your inputs into a text file named `in_pub.txt` and the expected and correct outputs into another text file named `out_pub_ans.txt`. If the compiled program is named `wa`, and you want the printed output to be saved into a file named `out_pub.txt`, you can execute the following command from the following terminals to run it:

   Windows (Powershell): `cat in_pub.txt | ./wa.exe | Out-File out_pub.txt`
   Linux/macOS (bash, zsh): `./wa < in_pub.txt > out_pub.txt`

   Then, compare the program output with the reference output by executing the following commands:

   Windows (Powershell): `Compare-Object (gc out_pub.txt) (gc out_pub_ans.txt)`
   Linux/macOS (bash, zsh): `diff out_pub.txt out_pub_ans.txt`
3. Submit a copy of the source code to the Week 03A submission bin. Make sure that you attach one (1) file in the bin containing the Rust source code with a `.rs` extension (preferably named `w03a.rs`). **Please do not send compressed files!**

SE Week 03B

This assessment will help you become familiar with generics and traits in Rust.

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

Problem Statement

mEEEdia: A Media Discord Bot.

Discord has emerged as a powerful platform for communities to connect, collaborate, and share experiences. Among the myriad features that enhance the Discord experience, media bots stand out as invaluable tools for making servers not just interactive, but also highly productive.

As an admin of the EEEI discord server, namely dEEEscord, you propose to create a media discord bot using Rust to boost the server's productivity. Like any other media discord bot, the bot has the functionality to `play`, `add`, and `show` the current queue list. The bot, however, can only queue songs as of the moment and is limited to 12 items only. In order to expand the bot capabilities, you will need to implement a generic queue should the next developer want to expand the capabilities of the bot to accommodate another media type, like a movie or an image.

Input

The input starts with a number T on a single line denoting the number of commands to be executed. T lines then follow which could be one of the following commands:

- `play`
  - Play the first media (if available) on the list and remove it from the queue.
- `add <media_name>`
  - Adds a media to the queue. The added media, `media_name`, could be any of the shown in the table below. Note that the `media_artist` is not part of the input.

`media_name`	`media_artist`
OMG	New Jeans
Perfect Night	LE SSERAFIM
Raining in Manila	Lola Amour

Never Gonna Give You Up	Rick Astley
Mananatili	Cup of Joe
Aphrodite	The Ridleys
Hanggang sa Buwan	Kenaniah
Dumaloy	SUD

- `show_queue`
  - Shows the current content of the queue.

Output

The output will consist of T lines indicating the response of the bot to the command. The bot has the following response for each command:

- `play`
  - If the queue is empty: `Queue is empty! No media to play...`
  - Otherwise: `Now playing: <media_name> by <media_artist>`
- `add <media_name>`
  - If the queue size is less than or equal to 12: `Successfully added <media_name> by <media_artist> to the queue!`
  - Otherwise: `Queue is full! <media_name> by <media_artist> is dropped.`
- `show_queue`
  - If the queue size is non-empty:

    ```
    -----mEEEdia bot-----                    (5 dashes each side)
    1. <media_name> by <media_artist>
    2. <media_name> by <media_artist>
    3. <media_name> by <media_artist>
    ...
    n. <media_name> by <media_artist>
    ---------------------                    (21 dashes in total)
    ```
  - Otherwise: `No media in queue.`

Please see the sample output for more details.

Constraints

Input Constraints

$T \leq 20$

$\mathrm{cmd} \in \{\mathrm{play}, \mathrm{add}, \mathrm{show}_{\mathrm{q}}\mathrm{ueue}\}$

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 the following `trait`s, `struct`s, and `impl`s.

- `trait Media` - the custom trait to be implemented.
    - `play(&self)`
    - `title(&self) -> String`
    - `artist (&self) -> String`
- `struct Song`
    - `title` - the title of the media
    - `artist` - the artist of the media
- `impl Media for Song` - method signature for Song
    - `play` - plays the first media on the list
        - Arguments
            - `&self` - the “self” instance of the struct
        - Return value
            - None
    - `title` - returns the media title
        - Arguments
            - `&self` - the “self” instance of the struct
        - Return value
            - `String` - the media title
    - `artist` - returns the media artist
        - Arguments
            - `&self` - the “self” instance of the struct
        - Return value
            - `String` - the media artist
- `struct Queue<T>` - the generic struct for queue
    - `list: Vec<T>` - container for the queued media
- `impl <T: Media> Queue <T>` - method signature for Queue. It **must** be generic.
    - `Queue::new()` - initialize a new `Queue` struct with empty values
        - Arguments
            - None
        - Return value
            - An empty `Queue` struct
    - `play` - plays the first item on the list
        - Arguments
            - `&mut self` - the “self” instance of the struct
        - Return value
            - None
    - `add` - adds a media to the queue

        - Arguments
            - `&mut self` - the “self” instance of the struct
            - `media: T` - the media to be added to the queue
        - Return value
            - None
    - `show_queue` - prints the current contents of the queue.
        - Arguments
            - `&self` - the “self” instance of the struct
        - Return value
            - None
    - `is_empty` - check if the queue is empty.
        - Arguments
            - `&self` - the “self” instance of the struct
        - Return value
            - `bool` - indicates if the queue is empty or not.

Failure to follow these functional constraints will result in a score of zero in this assessment.

Sample Input/Output

Sample Input 1:

```
10
show_queue
add OMG
add Mananatili
add Aphrodite
play
play
add Dumaloy
show_queue
add Raining in Manila
play
```

Sample Output 1:

```
No media in queue.
Successfully added OMG by New Jeans to the queue!
Successfully added Mananatili by Cup of Joe to the queue!
Successfully added Aphrodite by The Ridleys to the queue!
Now playing: OMG by New Jeans
Now playing: Mananatili by Cup of Joe
Successfully added Dumaloy by SUD to the queue!
-----mEEEdia bot-----
1. Aphrodite by The Ridleys
2. Dumaloy by SUD
---------------------
Successfully added Raining in Manila by Lola Amour to the queue!
Now playing: Aphrodite by The Ridleys
```

Sample Input 2:

```
10
play
add Hanggang sa Buwan
add Never Gonna Give You Up
add Perfect Night
play
add Never Gonna Give You Up
play
play
play
show_queue
```

Sample Output 2:

```
Queue is empty! No media to play...
Successfully added Hanggang sa Buwan by Kenaniah to the queue!
Successfully added Never Gonna Give You Up by Rick Astley to the
queue!
Successfully added Perfect Night by LE SSERAFIM to the queue!
Now playing: Hanggang sa Buwan by Kenaniah
Successfully added Never Gonna Give You Up by Rick Astley to the
queue!
Now playing: Never Gonna Give You Up by Rick Astley
Now playing: Perfect Night by LE SSERAFIM
Now playing: Never Gonna Give You Up by Rick Astley
No media in queue.
```

Steps

1. Write your program in Rust. Compile and make sure that there are no syntax errors.
2. Make sure to accept input via standard input and print your output via standard output. For example, you can write your inputs into a text file named `in_pub.txt` and the expected and correct outputs into another text file named `out_pub_ans.txt`. If the compiled program is named `wa`, and you want the printed output to be saved into a file named `out_pub.txt`, you can execute the following command from the following terminals to run it:

   Windows (Powershell):
   ```
   cat in_pub.txt | ./wa.exe | Out-File out_pub.txt
   ```
   Linux/macOS (bash, zsh):
   ```
   ./wa < in_pub.txt > out_pub.txt
   ```

   Then, compare the program output with the reference output by executing the following commands:

   Windows (Powershell):
   ```
   Compare-Object (gc out_pub.txt) (gc out_pub_ans.txt)
   ```
   Linux/macOS (bash, zsh):
   ```
   diff out_pub.txt out_pub_ans.txt
   ```

3. Submit a copy of the source code to the Week 03B submission bin. Make sure that you attach one (1) file in the bin containing the Rust source code with a `.rs` extension (preferably named `w03b.rs`). **Please do not send compressed files!**

SE Week 03C

This assessment will help you become familiar with hashmaps, traits, and lifetime annotations in Rust.

This is worth 30% of your grade for this week

Problem Statement

You are tasked to write an inventory system that keeps track of the current items in a warehouse and people that borrow them. A person can log in to the system and look at the items available for loan, look at the list of registered borrowers, borrow or return ("unborrow") an item, or transfer a currently borrowed item to another person.

For this system, there is strictly one borrower per item and items do not necessarily have to have a borrower. An item can only be borrowed if there is nobody that currently borrows it. On the other hand, an item can only be unborrowed if there is somebody that currently borrows it. Finally, an item can be transferred to another borrower only if, for verification purposes, the entered borrower has the same name as the recorded current borrower, and that the two borrowers are different. Of course the item and the borrowers should exist in the system for all of these cases for them to be processed. For simplicity of implementation, all items and borrowers have unique names.

Input

The first line of the input starts with the number B denoting the number of borrowers. B lines then follow, with each line written in the format `YYYY-MM-DD b_name` denoting the registration date in ISO 8601 format and name of a borrower `b_name`, respectively. After these lines, the next line then contains the number I denoting the number of items available for lending. I lines then follow, with each line written in the format `YYYY-MM-DD i_name` denoting the acquisition date in ISO 8601 format and name of an item `i_name`, respectively. After these lines, the next line then contains the number C denoting the number of commands that follow. C lines then follow, with each line written in the general format `cmd args` containing the command `cmd` and its respective space-separated arguments `args` respectively. Specifically, each of these lines can be from one of the following:

- `bi b_name`
  - print information about borrower `b_name`
- `ii i_name`
  - print information about item `i_name`
- `t i_name from_b to_b`

    - lend an item `i_name`, currently borrowed by `from_b`, to a new borrower `to_b`
- `b i_name b_name`
    - set item `i_name` to be borrowed by borrower `b_name`
    - both item `i_name` and borrower should exist for the command to work
- `u i_name`
    - unborrow item `i_name` if somebody currently borrows it

Output

The output contains C blocks of lines corresponding to each command in the input. Each block should output one of the following depending on its respective command:

- `bi` command
    - If borrower with name `b_name` exists
        - `[BINFO] Borrower(b_name) [Registered YYYY-MM-DD]`
        - Above line is followed by the following lines indented four spaces from the leftmost side:
            - `-----BORROWED ITEMS-----`
            - List of items borrowed by this borrower of the format `LentItem(i_name) [Acquired YYYY-MM-DD] [Borrowed by b_name]`, or `NONE` otherwise. Items are printed by order of ascending lexicographical order or ASCII value of `i_name`s.
    - Otherwise
        - `[BINFO] Borrower "b_name" not found!`
- `ii` command
    - If item with name `i_name` exists
        - If item is borrowed by borrower with name `b_name`
            - `[IINFO] LentItem(b_name) [Acquired YYYY-MM-DD] [Borrowed by b_name]`
        - Otherwise
            - `[IINFO] LentItem(b_name) [Acquired YYYY-MM-DD]`
    - Otherwise
        - `[IINFO] Item "i_name" not found!`
- `t` command
    - If item with name `i_name` and borrowers `from_name` and `to_name` exist
        - If item `i_name` is borrowed by borrower `from_name` and `from_name` is not the same as `to_name`
            - `[TRANSFER] Item "i_name" transferred from "from_name" to "to_name"!`
        - Otherwise, if item `i_name` is borrowed by borrower `from_name` and `from_name` is the same as `to_name`

      - `[TRANSFER] Item "i_name" is already borrowed by requester "to_name"!`
    - Otherwise, if item does not have a borrower
      - `[TRANSFER] Item "i_name" does not have a borrower!`
    - Otherwise, if item has a borrower with name `x_name`, which is not the same as `from_name`
      - `[TRANSFER] Item "i_name" cannot be transferred as it is currently borrowed by "x_name", not "from_name"!`
  - Otherwise, if item `i_name` does not exist
    - `[TRANSFER] Item "i_name" not found!`
  - Otherwise, if borrower `from_b` does not exist
    - `[TRANSFER] FROM borrower "from_b" not found!`
  - Otherwise, if borrower `to_b` does not exist
    - `[TRANSFER] TO borrower "to_b" not found!`
- `b` command
  - If item with name `i_name` and borrower with name `b_name` exists
    - If item does not currently have a borrower
      - `[BORROW] Item "i_name" borrowed by "b_name"!`
    - Otherwise, if item has a borrower with name `b_name`
      - `[BORROW] Item "i_name" already borrowed by requester and current borrower "b_name"!`
    - Otherwise, if item has a borrower with name `x_name`
      - `[BORROW] Item "i_name" cannot be borrowed as it is currently borrowed by "x_name"!`
  - Otherwise, if item `i_name` does not exist
    - `[BORROW] Item "i_name" not found!`
  - Otherwise, if borrower `b_name` does not exist
    - `[BORROW] Borrower "b_name" not found!`
- `u` command
  - If item with name `i_name` exists
    - If item currently has a borrowed with name `b_name`
      - `[UNBORROW] Item "i_name" unborrowed from "b_name"!`
    - Otherwise
      - `[UNBORROW] Item "i_name" has no borrower!`
  - Otherwise
    - `[UNBORROW] Item "i_name" not found!`

Constraints

Input Constraints

$B, I \in Z; B, I \in (0, 100]$

$C \leq 100$

Names `b_name` and `i_name` and contain characters from the set [a-z0-9_] only.

All names `b_name` and `i_name` are unique in each test case.

Functional Constraints

You are **required** to create and use the following `struct`s. Note that lifetime annotations may be required:

- `SplitDate` - represents a date
  - `year: u64` - 4-digit year
  - `month: u8`- two-digit month with values between 1 and 12 inclusive
  - `day: u8`- two-digit day with values between 1 and 31 inclusive
- `LentItem` - represents a lent item
  - `name: String` - item name
  - `acquire_date: SplitDate` - date of acquisition
  - `borrowed_by: Option <Borrower>` - current borrower; `NONE` if it does not currently have a borrower
- `Borrower` - represents a
  - `name: String` - borrower name
  - `reg_date: SplitDate` - date of registration

You are **required** to have the following function and method signatures, their arguments in order, and their return values. Note that lifetime annotations may be required:

- `LentItem::new()` - initialize a new `LentItem` struct with the provided values
  - Arguments
    - `name: String` - item name
    - `year: u64` - year component of the date of acquisition
    - `month: u8` - month component of the date of acquisition
    - `day: u8` - day component of the date of acquisition
  - Return value - `LentItem` with item name `name` and provided acquisition date
- `LentItem::borrow()` - lend an item to a borrower assuming that the item itself is open
  - Arguments
    - `&mut self: &mut Self` - the "self" instance of the struct
    - `to: &Borrower` - the borrower interested in borrowing the item
  - Return value - `Option <&Borrower>` representing the previous borrower; `None` if the item did not previously have a borrower
  - Additional constraints
    - `self.borrowed_by` should be updated if the item can be borrowed by borrower `to`
- `LentItem::ubborrow()` - unborrow an item assuming that the item itself currently has a borrower
  - Arguments

    - `&mut self: &mut Self` - the "self" instance of the struct
  - Return value - `Option <&Borrower>` representing the previous borrower; `None` if the item did not previously have a borrower
  - Additional constraints
    - `self.borrowed_by` should be updated if the item can be unborrowed
- `LentItem::transfer()` - transfer current borrower of the item to a new borrower
  - Arguments
    - `&mut self: &mut Self` - the "self" instance of the struct
    - `from: &Borrower` - the borrower that currently holds the item
    - `to: &Borrower` - the borrower interested in borrowing the item from borrower `from`
  - Return value - tuple `(Option <&Borrower>, Option <&Borrower>)` representing the previous and new borrower of the item, respectively; second element is `None` if transfer was not successful due to 1) previous and current borrowers are the same, or 2) borrower `from` is not the same as current item borrower
  - Additional constraints
    - `self.borrowed_by` should be updated if the item can be transferred
- `Borrower::new()` - initialize a new `Borrower` struct with the provided values
  - Arguments
    - `name: String` - borrower name
    - `year: u64` - year component of the date of acquisition
    - `month: u8` - month component of the date of acquisition
    - `day: u8` - day component of the date of acquisition
  - Return value - `Borrower` with borrower name `name` and provided registration date
- `Borrower::borrowed_items()` - get a list of items currently borrowed by this borrower
  - Arguments
    - `&self: &Self` - the "self" instance of the struct
    - `items: &Vec <&LentItem>` - the list of all items in the inventory regardless of whether they are borrowed
  - Return value - `Vec <&LentItem>` representing the list of items borrowed by this borrower
  - Additional constraints
    - `self.borrowed_by` should be updated if the item can be borrowed by borrower `to`
    - Returned list of items are sorted by ascending lexicographical order or ASCII value of their names.
- `main()` - entry point to the program
  - Arguments
    - None
  - Return value

        - None
    - Additional constraints
        - Input and output parsing should be done here

You are **required** to implement the following traits for the given `struct`s, their arguments in order, and their return values. Note that lifetime annotations may be required:

- `std::fmt::Display for SplitDate` - return a string representation of this struct
    - Arguments
        - `&self: &Self` - the "self" instance of the struct
        - `&fmt: &mut std::fmt::Formatter` - the formatter object
    - Return value - `std::fmt::Result` containing the string representation of this `struct`
    - Additional constraints
        - Should display the string `YYYY-MM-DD` corresponding to the four-digit year, two-digit month, and two-digit day respectively.
- `std::fmt::Display for LentItem` - return a string representation of this struct
    - Arguments
        - `&self: &Self` - the "self" instance of the struct
        - `&fmt: &mut std::fmt::Formatter` - the formatter object
    - Return value - `std::fmt::Result` containing the string representation of this `struct`
    - Additional constraints
        - Should display one of the following:
            - `LentItem(i_name) [Acquired YYYY-MM-DD]`
                - For items that do not have a borrower - corresponding to the name of the item `i_name` and the string representation of the acquisition date.
            - `LentItem(i_name) [Acquired YYYY-MM-DD] [Borrowed by b_name]`
                - For items that have a borrower - corresponding to the name of the item `i_name`, the string representation of the acquisition date, and current borrower `b_name`.
- `std::fmt::Display for Borrower` - return a string representation of this struct
    - Arguments
        - `&self: &Self` - the "self" instance of the struct
        - `&fmt: &mut std::fmt::Formatter` - the formatter object
    - Return value - `std::fmt::Result` containing the string representation of this `struct`
    - Additional constraints

- Should display the string `Borrower(b_name) [Registered YYYY-MM-DD]` corresponding to the name of the borrower `b_name` and the string representation of the registration date.

Failure to follow these functional constraints will result in a score of zero in this assessment.

Sample Input/Output

Sample Input 1:

```
3
2024-01-01 carl_a
2024-03-02 carl_b
2022-04-05 carl_c
5
2020-03-04 book
2021-05-20 computer
2020-08-06 lamp
2020-12-24 phone
2020-01-05 bedsheet
5
bi carl_a
bi lester
ii bedsheet
b phone carl_b
bi carl_b
```

Sample Output 1:

```
[BINFO] Borrower(carl_a) [Registered 2024-01-01]
    Borrowed Items:
    NONE
[BINFO] Borrower "lester" not found!
[IINFO] LentItem(bedsheet) [Acquired 2020-01-05]
[BORROW] Item "phone" borrowed by borrower "carl_b"!
[BINFO] Borrower(carl_b) [Registered 2024-03-02]
    -----BORROWED ITEMS-----
    LentItem(phone) [Acquired 2020-12-24] [Borrowed by carl_b]
```

Sample Input 2:

```
3
2019-03-04 alice
2019-03-05 bob
2019-12-31 trudy
2
2020-03-04 key
2021-05-20 lock
8
b key alice
```

```
b lock bob
bi alice
bi bob
b lock alice
b lock bob
u lock
bi bob
```

Sample Output 2:

```
[BORROW] Item "key" borrowed by "alice"!
[BORROW] Item "lock" borrowed by "bob"!
[BINFO] Borrower(alice) [Registered 2019-03-04]
    -----BORROWED ITEMS-----
    LentItem(key) [Acquired 2020-03-04] [Borrowed by alice]
[BINFO] Borrower(bob) [Registered 2019-03-05]
     -----BORROWED ITEMS-----
     LentItem(lock) [Acquired 2021-05-20] [Borrowed by bob]
[BORROW] Item "lock" cannot be borrowed by "alice" as it is
currently borrowed by "bob"!
[BORROW] Item "lock" already borrowed by requester and current
borrower "bob"!
[UNBORROW] Item "lock" unborrowed from "bob"!
[BINFO] Borrower(bob) [Registered 2019-03-05]
    -----BORROWED ITEMS-----
    NONE
```

Steps

1. Write your program in Rust. Compile and make sure that there are no syntax errors.
2. Make sure to accept input via standard input and print your output via standard output. For example, you can write your inputs into a text file named `in_pub.txt` and the expected and correct outputs into another text file named `out_pub_ans.txt`. If the compiled program is named `wa`, and you want the printed output to be saved into a file named `out_pub.txt`, you can execute the following command from the following terminals to run it:

   Windows (Powershell): `cat in_pub.txt | ./wa.exe | Out-File out_pub.txt`
   Linux/macOS (bash, zsh): `./wa < in_pub.txt > out_pub.txt`

   Then, compare the program output with the reference output by executing the following commands:

   Windows (Powershell): `Compare-Object (gc out_pub.txt) (gc out_pub_ans.txt)`
   Linux/macOS (bash, zsh): `diff out_pub.txt out_pub_ans.txt`

3. Submit a copy of the source code to the Week 03C submission bin. Make sure that you attach one (1) file in the bin containing the Rust source code with a `.rs` extension (preferably named `w03c.rs`). **Please do not send compressed files!**