Lab 2: Address Book

Professor Patt has a bad memory. He often cannot remember one's room number. To prevent forgetting this information, he usually takes a note on his address book. As a computer man, he would like to implement the address book by using a linked list. But a few days ago the searching program for this linked list broke down, and he is now quite urgent to have a meeting with Professor Jiang and his TAs. Please help him!

Your job is to write a program to search on the provided linked list. The user will enter a name on the console, and your program is expected to print the corresponding full names, together with the room numbers, based on the information stored in the linked list.

> This lab is associated with the following chapters and sections. If you have no idea how to do this lab, please review the corresponding parts of the book.
>
> - 8.5 Character Strings
> - 16.2 Pointers
> - 19.5 Linked Lists
> - Usage of trap service routines, in Table A.3 on page 675

Implementation Details

- You are required to write in **LC-3 assembly language**.
- Your program should start at x3000, which means the first instruction of your program is located in position x3000.

Procedure

1. Prompt the user to type a name. Print the string `Enter a name:` on the computer screen. Then wait for the user to input a string and press the Enter key.
    - There should be a space character after the colon `:`. See the samples.
    - When someone is typing, echo the characters on the console.
    - You do not need to handle with Backspace, Delete, or any other non-printable characters.
2. Search the information matching the input name. For each person whose first name or last name is exactly the same as the input name, then print his or her first name, last name, and then room number in one line, separated by a space.
    - The matching is case sensitive.
    - If there are more than one matches, then print all of them in order, each in one line. See the samples.
    - Specially, if someone's first name and last name are the same, and both of them match the input name, you should only print this entry once.
3. If there is no matching information, print `Not found`.
4. Halt.

Structure of the address book

The address book is a linked list. The **head pointer** pointing to the first node has been stored in address x4000. All other nodes and strings are stored between address x4001 and xEFFF. Each node contains 4 locations in memory:

1. The first location is a pointer pointing to the next node.
2. The second location is a pointer pointing to an ASCII string representing the room number.
3. The third location is a pointer pointing to an ASCII string representing the first name.
4. The last location is a pointer pointing to an ASCII string representing the last name.

Note: If a pointer is pointing to the location x0000, it means that there is no node after it.

Location	Node structure
p	The pointer for the next node
p+1	The pointer to an ASCII string representing the room number
p+2	The pointer to an ASCII string representing the first name
p+3	The pointer to an ASCII string representing the last name

How to Run?

1. Reinitialize or randomize the machine.
2. Load the address book (i.e. `sample.obj`) into the simulator.
3. Load your program into the simulator. Now PC should be x3000, and both the address book and your program are loaded into the memory.
4. Run the machine.
5. Enter a name on the console.
6. The program prints the result and then halts the machine.
7. If you want to test another case, just move the PC to x3000 and then start a new run (step 4).

Samples

```
Enter a name: Patt
Yale Patt 101

--- Halting the LC-3 ---
```

```
Enter a name: Yang
Yang Zhiquan 410
Yang Bochun 411

--- Halting the LC-3 ---
```

```
Enter a name: Qiu
Not found

--- Halting the LC-3 ---
```

The sample address book is attached to this document.

> Of course, we will use different address books to check your program :)

Limitations

- The length of the linked list l: $0 \leq l < 100$.
- The length of the input name n: $0 \leq n < 16$.

Grading

Lab 2 takes 5% of the total score, consisting of Check part (50%) and Report part (50%).

Check part (50%)

- Find a TA to check your code in person. TAs may ask you questions when grading your lab assignment. You will get 100%, 80% or 60% of the checking score according to your response.
- You can try again if you fail in checking, but there will be a penalty of -10% (of checking part) for each try.
- We suggest you to run your program on PTA to check by yourself before you find a TA. The link to this lab on PTA will be available later.
- We suggest you to write enough comments in your code so that you will be aware of what's going on in your program and confident to answer TA's questions.

Report part (50%)

- English report should be concise and carrying main ideas. Try to use the report to convince TAs that you complete the task by yourself.
- Your lab report should *at least* contains the following contents:
    - Your algorithm. To make it clear, you can use figures, tables or any other easy-to-understand appearance.
    - Essential parts of your code with sufficient comments. Please only select the most important code phases and explain them.
    - The questions that TA asked you, and answers.
- No more than 3 A4 pages. No template provided. Be sure to make it readable.

Penalty

- **Wrong Answer**: -10% of Check part each time.
- **Delay**: -20% of the corresponding part per day.
- **Cheating**: -100% of this lab. Additionally, -10% of the final score of this course.