Assignmet Overview

In this project you will practice strings and control statements.

It is worth 10% of the course grade and must be completed and turned in no later than 11:59 on Sunday, Nov 8, 2015.

You can do this project in groups of maximum two or you can do it individually.

This project will also be graded on a one on one presentation where the students will have answer questions pertaining to their process. Group members will present their project together.

Background

Finite State Machine (also called Finite State Automata)

A Finite State Machine (FSM) is a popular tool that is widely used in computer science. Why it is so popular? Because it is simple, and easy to understand and implement. Here is a brief introduction: http://cs.union.edu/~striegnk/courses/nlp-with-prolog/html/node2.html

Your task is to implement an FSM that can recognize a simple patterns of strings such as:

ha!
haha!
ahahaha!
ho!
hoho!
hohoho!
hahohoho!
hahahahaha!

Basically, the pattern to be recognized is a sequence of “ha” or “ho”, or a mixture of both. The sequence can be of any length (at least greater than three, i.e., the shortest sequence is “ha!” or “ho!”). There is no restriction on how many “ha”s or “ho”s the sequence can contain, and the “ha”s and “ho”s can appear in any order if the sequence is a mixture of both. The string must end with one “!”.

The FSM that recognizes those exact string patterns can be drawn as follows:
This FSM contains 4 states and 5 arcs. State 1 is the start state and state 4 is the end state. The arcs show how it transits from one state to another. For example, if we are at state 1 and observe a “h”, then we transit to state 2, and so forth.

When using this FSM to recognize a string pattern (i.e., whether or not an input string conforms to the pattern described by the FSM), we scan the input string from left to right, and make transitions based on the observed character and the FSM.

For example, if the input is “ha!”, we scan it from left to right, reading in one character at each time. We start the FSM transition from state 1, and the first character we read in is “h”, thus we can transit to state 2 according to the FSM. Then we read in the second character “a”, and we can transit to state 3 since we are already at state 2. The last character is “!” which brings us from state 3 to state 4, the end state. If we reach the end state and there is no more character left in the input string, we say that the FSM recognizes/accepts the input string (i.e., the input string conforms to the pattern encoded by this FSM).

Requirements and Notes:

- Implement the above FSM to recognize whether a given string conforms to the specific pattern or not.
- Keep prompting the user for typing in a string, scan through the user’s input string and transit the FSM accordingly. Report to the user whether the input string conforms to the FSM pattern or not.
- Quit the program when the user types in “bye”.
  (See the sample output at the end of this document)

- To clarify the project specifications, sample output is provided at the end of this document.
- The inputs that a user provides to your program can be either integers or float numbers.
- It is important and mandatory that you put as many comments as possible in your code. For each line or a block of code you should put comments (you will loose credits for not having comments).
- The variables that you are defining should be readable and meaningful. Do not use variable names such as ‘a’, ‘b’ or ‘c’; pick meaningful names. For example, name the length of one side of the garden variable
- Your code should follow the coding standard on the course website: http://www.cse.msu.edu/~cse220/coding/default.html
  Remember that it is important to start with a good habit.
Hints:

(1) FSM recognizes/accepts a string only when it reaches the final state (i.e., state 4) and there is no unread character left over in the input string. For any other cases, such as it cannot reach the final state, or it can reach the final state but there are still unread characters left over, the FSM fails to accept the string (i.e., the string does not conform to the FSM’s pattern).

Besides, if the observed character cannot make the current state transit to any other state, then the recognition already fails and we don’t need to continue the process. For example, if the FSM is currently at state 2 and it read in a “h” from the input string, the recognition fails since no transition can be made given the state and the observed character.

Here are more examples:

- Input “ha!”: starting in state 1, the “h” transitions to state 2, the “a” then transitions to state 3 and the “!” transitions to state 4. Since that was the last character we finish in state 4 which indicates success.
- Input “hax”: starting in state 1, the “h” transitions to state 2, the “a” to state 3, but the “x” cannot transit to any state, which indicates failure.
- Input “hahohax”: starting in state 1, the “h” transitions to state 2, the first “a” to state 3, the next “h” sends us back to state 2, the “o” to state 3, the next “h” back again to state 2, the next “a” to state 3, and finally the “!” to state 4. Since that is the last character we finish in state 4 which is success.
- Input “yha!”: starting in state 1, the “y” transits to nowhere, which is failure.
- Input “ha!x”: starting in state 1, the “h” transitions to state 2, the “a” to state 3, the “!” to state 4, but there is one more character “x” left after we have already reached the final state 4, which is failure.

(2) Since we haven’t talked about string yet, a sample code (proj2-sample.c) is provided to you. It demonstrates how to input a string, scan through the string character-by-character, etc. You can just use the provided code as the starting point of your own project implementation.

Assignment Deliverable

The deliverable for this assignment is the following file:

proj2.c – the source code for your C program

Be sure to use the specified file name (“proj2.c”) and to submit it for grading via the handin system before the project deadline.
**Sample Output**

```plaintext
<arctic:~ >a.out
I can recognize if you are laughing or not, please type in a string: ha!
Your input is: ha!
    You are laughing!
I can recognize if you are laughing or not, please type in a string: haha!
Your input is: haha!
    You are laughing!
I can recognize if you are laughing or not, please type in a string: ho!
Your input is: ho!
    You are laughing!
I can recognize if you are laughing or not, please type in a string: hohohaha!
Your input is: hohohaha!
    You are laughing!
I can recognize if you are laughing or not, please type in a string: haa!
Your input is: haa!
    You are not laughing...
I can recognize if you are laughing or not, please type in a string: ha!!
Your input is: ha!!
    You are not laughing...
I can recognize if you are laughing or not, please type in a string: ha!ha
Your input is: ha!ha
    You are not laughing...
I can recognize if you are laughing or not, please type in a string: bye
Bye now!
```