CSCI 431 Project 5: A Turing Machine Simulator

For this project you will create a Turing Machine simulator that has a tape that is infinite in one direction. The first tape square will be tape square 0, the next tape square 1, and so on. You will create a Turing Machine simulator as described below.

The Software

Create software that takes in four command line parameters, as follows:

The first parameter: This is the name of an TM definition file

the second parameter: This is the name of a file holding strings, one string per line.

The third parameter: This is the output file.

The fourth parameter: This is the maximum number of steps the machine can run before it is shut down. (This prevents the program from going on forever in a non-terminating loop.)

TM definition file

The file is formatted as follows:

Line 1: This holds a string. The string represents the input alphabet. It can only include ASCII characters with decimal codes 33-126. The space character (code 32) represents the blank, and *cannot* be part of the input alphabet.

Line 2: This holds a string. The string represents the tape alphabet. It must include the blank (space) character *as the first character* and all characters of the input alphabet. It can only include ASCII characters with decimal codes 32-126.

α represents the number of characters in the tape alphabet.

Line 3: This line holds a single number. The number is the number of states in the machine (not counting the three halting states). In the rest of this description, η is used to represent the number of states, *not* counting the halting states.

State 0 is always taken to be the start state.

The next η lines: Each of these lines represents a row in the table for the transition function (δ). The first line represents the row for state 0, the next line the row for state 1, etc.

Each line has α triplets of the form [ψ,ρ,σ] where ψ is either R or L, ρ is the character to write to the tape, and σ is the number of the next state.

The value of σ must be an integer in the range $0 \leq \sigma < \eta$ or -1 (which is the HALT-NO state) or -2 (which is the HALT-YES state).

The number of triplets is the same as the number of tape characters. When the machine is in a given state reading a character, the triplet used for the next move is the one on the line for the state in the same position as the character on line 2. If the character was the fourth one, then the triplet used is the fourth triplet on the line for the current state.

The ψ part of the triplet indicates the direction that the machine moves. If the value is R and `n` is the index of the current tape square the machine moves to the square `n+1`. If the value is L and `n` is the index of the current tape square the machine moves to the square `n-1`.

If a move would take the machine to an index less than 0 then the machine halts in state -3 (which is the HALT-BADMOVE state).

The ρ part of the triplet is the character that is written to the current tape square (before the move).

The σ is the next state.

The input file

This file has a string on each line. Empty lines are allowed. Empty lines represent the empty string.

Each line represents a run of the machine. The string is the initial string written on the tape. The string is written starting on square 1. Square 0 is given a blank.

The output file

There should output in the output file for each line in the input file.

The output for an input line should formatted thusly:

1. The first line contains 20 equal signs (=)
2. The next line contains the input
3. The next line contains the final state (usually one of: -1, -2, or -3, but it could be a regular state if the program quit because it hit the maximum number of steps given by the fourth command line parameter)
4. The last line contains the contents of the tape when the machine stops

How it works

The program should:

1. Check for the proper number of parameters - quit w/ an error message if wrong.
2. Read in the FSA definition and configure itself internally based on the definition. If the file is erroneous, then output an error message and quit.
3. Process lines from the input file, outputting corresponding lines to the output file as you go.

You should have a makefile for your program.

Teams

You may work in pairs or individually. You may pick your own partner if you work with someone else.