Due 9/22

Write a program to simulate the execution of a restricted Turing machine. Your program can be written in Java, C, or C++, and needs to be able to be compiled and executed on the computers in EB-G7 (or a Linux or Mac computer I have access to). If you do not know Java, C, or C++, you may submit a version in Python (with my permission).

Your program will read the definition of the machine from a file (the first command line argument), with the second command line argument being the string to simulate the machine running on (the input to the automata), and the third command line argument being the maximum number of transitions to simulate. The output of your program will be written to standard output. The output will consist of the list of states the automata transitions through while reading the input string (separated by "->") followed by either "accept" or "reject", depending on wether the automata accepts or rejects the input string.

The states will be numbered between 0 and 1,000 (not necessarily contiguous or in any particular order). There are three special types of states – the start state (exactly one), the accept state(s) (0 or more), and the reject state(s) (0 or more). The format of a "state definition" line is:

"state x y" where x is a number in [0, 1000], and y is either null, start, accept, or reject. No state can be more than one of these (can't be start and accept, start and reject, or accept and reject).

Some examples are:
state 7 start
state 12 accept
state 952 reject

There is no guarantee on the order of the state lines, other than they are at the beginning of the file and there is one line per state. The input file will be tab delimited (should be easily parsed with Java, C, or C++). The only state lines that are required are those for the start, accept, and reject states.

The remainder of the file defines the transitions. For this machine, the transition format is "q,a->r,b,x" where q is the current state that the machine is in, the a is the symbol that the machine reads, the r is the state that the machine transitions to, the b is the value that the machine writes on top of the a, and the x is either an L or an R, which tells you to either move back one symbol (L) or to the next symbol (R).

The format of the transitions in the file will be:

transition q a r b x

Since q and r are states, they will be numbers in the range of [0, 1000]. You can assume that a and b will be digits {0, 1, …, 9}, lower case letters {a, b, …, z} or special characters {$, #, _, %}. And finally x will be in {L, R}. The "_" for a or b is used to represent the blank space character. This is due to problems associated with parsing a blank space from an input line, when white space characters are used to delimit the other values. In my program, when I read a "_" for a or b, I simply replace the "_" with a blank space. You should assume that to the right or left of the input there are blank space characters.

The input will be a string, consisting of digits, lower case letters, and special characters. Initially the machine will be looking at the left most symbol of the input string. The transitions will tell you what symbol to process next. You should assume that there are blank spaces to the left and right of the input. You will need to be able to handle cases where you move to the left or right of the input symbols.

Due 9/22

If the machine ever transitions to an accept state, then it is to immediately stop and output the sequence of states that is transitioned through (with “->” between them), followed by a blank space and “accept”.

If the machine ever transitions to a reject state, then it is to immediately stop and output the sequence of states that is transitioned through (with “->” between them), followed by a blank space and “reject”.

If the machine hasn't entered the accept state or reject state after the number of transitions specified by the third command line argument, then it is to stop and output the sequence of states that it transitioned through (with “->” between them), followed by a blank space and “quit”. The largest value for the maximum number of transitions that I will use is 10,000.

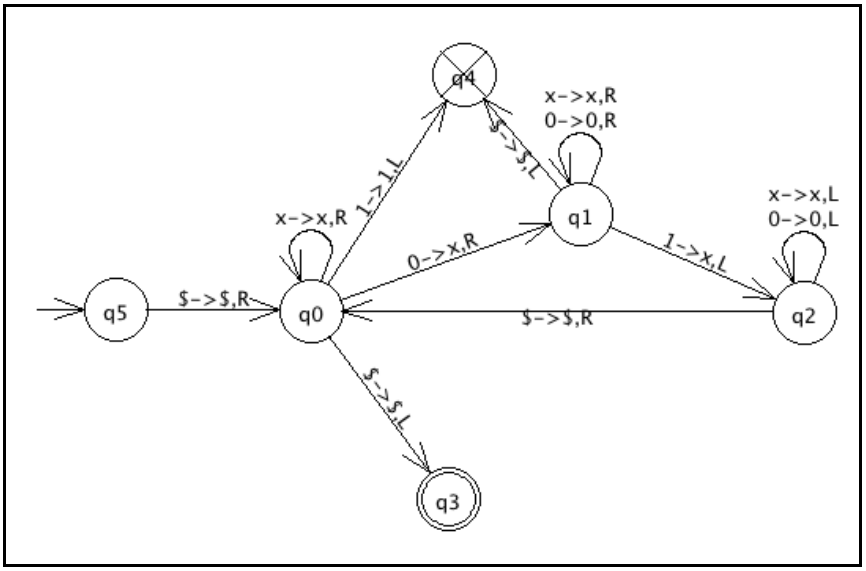
For java, standard input is System.in, standard output is System.out, and standard error is System.err.
For C, standard input is stdin, standard output is stdout, and standard error is stderr.
For C++, standard input is cin, standard output is cout, and standard error is cerr.

E-mail your program (source file(s) and makefile if required attached to the e-mail, I don't want a tar, zip, rar, jar, or any other package file) to me (david.garrison@binghamton.edu) by 11:59:59pm on the date due. Your main filename is to be your last name (lower case) followed by “_p1” (for example, my filename would be “garrison_p1.java”) and the subject of your e-mail is “CS 373 program 1”. If you write C or C++, your executable file is to be named your last name (lower case) followed by “_p1” (for example, my executable filename would be “garrison_p1” if I wrote a C or C++ program).

The grading will be based on the percentage of correct results your program gets for my collection of test automata and test strings and following the directions.

In particular, 20% of the grade will be for following the instructions (filename being your last name in lower case, subject, attaching files to e-mail, correct executable name) and 80% will be correct results (check my out put format). If you are using java, I should be able to execute “javac “your last name in lower case_p1”.java” to compile your program. For C or C++ your should include a makefile so that I can simply execute “make “your last name in lower case_p1”” to compile and ./”your last name in lower case_p1” to execute.

Here's is a sample state transition diagram of a restricted Turning machine.



Here q4 is a reject state (has an “x” through it). Since I said the states were numbers, the text file associated with this diagram will have the “q” removed from the state names.

Here is the text file associated with the diagram.

```
state   3       accept
state   4       reject
state   5       start
transition      0       x       0       x       R
transition      0       1       4       1       L
transition      0       0       1       x       R
transition      1       0       1       0       R
transition      1       x       1       x       R
transition      1       1       2       x       L
transition      2       0       2       0       L
transition      2       x       2       x       L
transition      2       $       0       $       R
transition      0       $       3       $       L
transition      5       $       0       $       R
transition      1       $       4       $       L
```

The language accepted by this machine is { 0^n1^n | n ≥ 0 }.

For my C++ version of the program, I only have a single file, garrison_p1.cpp. So, I can simply execute “make garrison_p1” to compile and create the executable “garrison_p1”.

Below are three samples runs that ideally are correct.

```
./garrison_p1 prog1_sample.txt '$01$' 500                ← command line
5->0->1->2->2->0->0->0->3 accept                         ← output written to standard output
```

The “$” has special meaning in the unix shell(s), so the single quotes around the input string, '01' are so that the unix shell treats the “$” symbol as a normal symbol. We could also enter the string as \$01\$.

Below are examples for accept and reject. For quit, an input string of somewhere around 250 '0's followed by 250 '1's will results in a quit with a limit of 5000 transitions.

Accept example:

```
./garrison_p1 prog1_sample.txt '$000111$' 500
5->0->1->1->1->2->2->2->2->0->0->1->1->1->2->2->2->2->2->0->0->0->1->1->1->2->2->2->2->2->2->
>0->0->0->0->0->0->0->3 accept
```

Reject example:

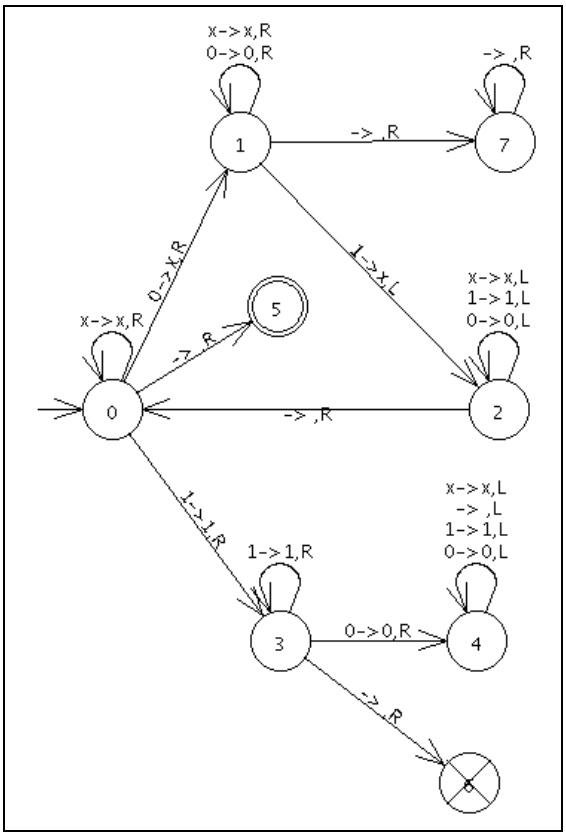
```
./garrison_p1 prog1_sample.txt '$00011$' 500
5->0->1->1->1->2->2->2->2->0->0->1->1->1->2->2->2->2->2->0->0->0->1->1->1->4 reject
```

Quit example:

Due 9/22

./garrison_p1 ../prog1_sample.txt '00011' 10
5->0->1->1->1->2->2->2->2->0->0 quit

Below is the state transition diagram of a second example.



The file associated with it is below.

state	0	start			
state	5	accept			
state	6	reject			
transition	0	1	3	1	R
transition	3	0	4	0	R
transition	1	_	7	_	R
transition	0	_	5	_	R
transition	0	0	1	x	R
transition	1	1	2	x	L
transition	2	0	2	0	L
transition	2	1	2	1	L
transition	2	x	2	x	L
transition	4	0	4	0	L
transition	4	1	4	1	L
transition	4	_	4	_	L
transition	4	x	4	x	L
transition	1	0	1	0	R

transition	1	x	1	x	R
transition	3	1	3	1	R
transition	0	x	0	x	R
transition	7	_	7	_	R
transition	2	_	0	_	R
transition	3	_	6	_	R

Below are examples of accept, reject, quit (one going to the left of the input and one going to the right of the input).

./garrison_p1 ../p1_sample1.txt 0011 100
0->1->1->2->2->2->0->0->1->1->2->2->2->2->0->0->0->0->0->5 accept

./garrison_p1 ../p1_sample1.txt 00111 25
0->1->1->2->2->2->0->0->1->1->2->2->2->2->0->0->0->0->0->3->6 reject

./garrison_p1 ../p1_sample1.txt 001 20
0->1->1->2->2->2->0->0->1->1->7->7->7->7->7->7->7->7->7->7->7 quit

./garrison_p1 ../p1_sample1.txt 0110 24
0->1->2->2->0->0->0->3->4->4->4->4->4->4->4->4->4->4->4->4->4->4->4->4->4->4->4 quit

You should note that case matters on your output (lower case), spaces matter on the output (only one blank space) and the output should have a newline character at the end of the line. When I test your program, I will execute your program on 20 sets of input data (each test is worth 4 percent of the grade). Each test case has two components of the grade, the correct path and the correct result (2 percent for each). If things go well, I will generate all of the test runs, execute them, and grade them automatically. You should also note that there will be test cases with more than one accept state and more than one reject state.