Due 9/7

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 the remote cs computer(s) (or a Linux or Mac computer I have access to). Contact me if you do not know Java, C, or C++.

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 restricted part). The output of your program will be written to standard output. The output will consist of the output generated by the Turing machine followed by a blank space followed by either “accept”, “reject”, “quit”, or “crash”, depending on wether the automata accepts, rejects, performs the maximum number of transitions on the input string, or moves past the left end of the input. If there is no transition defined for the current state and input symbol, this should be considered as equivalent to transitioning to a reject state.

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 988
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 at most 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. You should assume that any state in [0, 1000] that is not the start state or accept/reject state is a valid state to transition to.

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, a is the symbol that the machine reads, r is the state that the machine transitions to, b is the symbol that the machine writes on top of the a, and x is either an L, S, or an R, which tells you to either move back one symbol (L), stay on the current symbol (S), or move 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 {$, #, _, %, -, ., ,} (period and comma are included). And finally x will be in {L, R, S}. 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 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 right of the input. You will need to be able to handle cases where you move to the right of the input symbols. My test machines and strings may move to the left of the input string, resulting in the Turing machine crashing.

If the machine ever transitions to an accept state, then it is to immediately stop and output the Turing machine output, 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 Turing machine output, followed by a blank space and “reject”. Likewise if there is no transition for the current state and input symbol.

The output is to end with a single newline character.

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 Turing machine output, followed by a blank space and “quit”. The largest value for the maximum number of transitions that I will use is 2,000,000. Accepting and rejecting should be evaluated prior to quitting.

If the machine attempts to move past the left end of the input, then it is to immediately stop and output the Turing machine output, followed by a blank space and “crash”. Since the Turning machine has moved past the left end of the input, the output should begin at the left end of the input.

The Turing machine output is all symbols, starting with the one under the Turing machines read/write head and all symbols to the right until a blank space is encountered (the blank space is not part of the output).

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.

Post your program (source file(s) and makefile, if required) on Brightspace for the assignment associated with this programming assignment by 11:59:59pm on the date due. I don't want a tar, zip, rar, jar, or any other package files. Your main filename is to be your last name (lower case) followed by “_p1” (for example, my filename would be “garrison_p1.java”). 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).

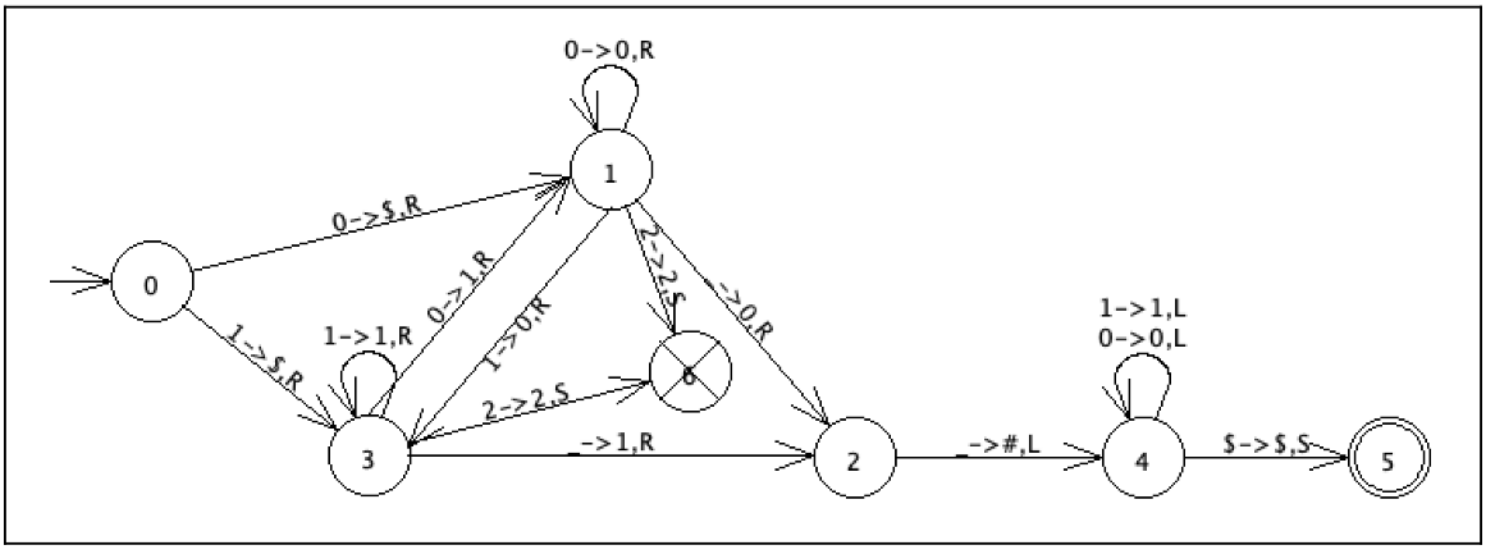
For my C++ version of the program, I only have a single file, garrison_p1.cpp. So, I can simply execute “make garrison_p1” to compile and create the executable “garrison_p1” or “g++ garrison_p1.cpp -o garrison_p1”. If I don't need a makefile to build your program, then your don't need to include one.

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, correct executable name, submitting source files correctly) and 80% will be correct results (check my output 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++ you should include a makefile, if needed, 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.

Below is a sample state transition diagram for Turing machine “insertDollarSignAndAppendPoundSign.txt”.



State 6 is a reject state (has an “x” through it) and state 5 is an accept state (has circle in it).

The associated text file is below.

```
state   0       start
state   1
state   2
state   3
state   4
state   5       accept
state   6       reject
transition      3       2       6       2       S
transition      4       0       4       0       L
transition      4       1       4       1       L
transition      2       _       4       #       L
transition      1       0       1       0       R
transition      3       1       3       1       R
transition      1       _       2       0       R
transition      4       $       5       $       S
transition      0       0       1       $       R
transition      3       _       2       1       R
transition      1       1       3       0       R
transition      3       0       1       1       R
transition      1       2       6       2       S
transition      0       1       3       $       R
```

Due 9/7

This Turing machine inserts a $ and shifts the symbols to the right and appends a # at the right of the input and rewinds the tape, leaving the read/write head on the $. It reject if it reads a 2 after reading a 0 or 1.

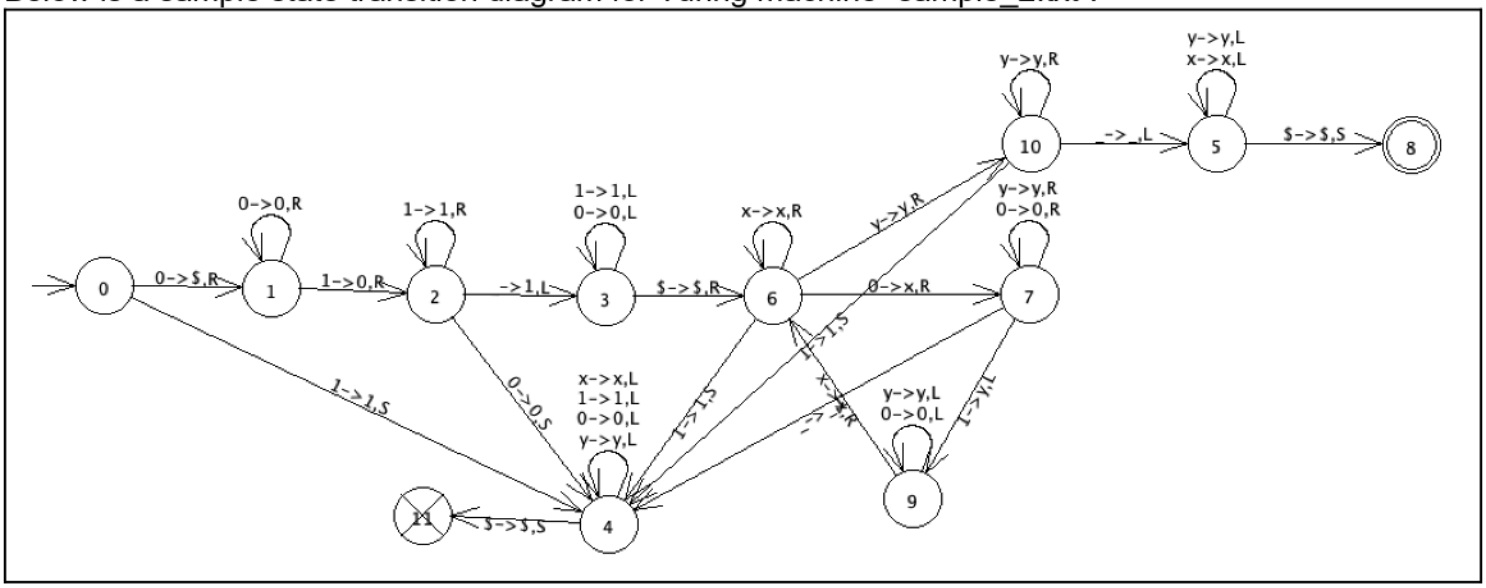
Below are three samples runs that ideally are correct.

```
./garrison_p1 insertDollarSignAndAppendPoundSign.txt 0110 100  ← command line
 $0110# accept                                                 ← output written to standard output

./garrison_p1 insertDollarSignAndAppendPoundSign.txt 0110 9    ← command line
 0110# quit                                                    ← output written to standard output

./garrison_p1 ../insertDollarSignAndAppendPoundSign.txt 021 20 ← command line
21 reject                                                      ← output written to standard output
```

Below is a sample state transition diagram for Turing machine “sample_2.txt”.



The file associated with it is below.

```
state   0       start
state   1
state   2
state   3
state   4
state   5
state   6
state   7
state   8       accept
state   9
state   10
state   11      reject
transition      6       y       10      y       R
```

transition	10	1	4	1	S
transition	5	$	8	$	S
transition	3	$	6	$	R
transition	9	x	6	x	R
transition	6	0	7	x	R
transition	7	_	4	_	L
transition	0	0	1	$	R
transition	0	1	4	1	S
transition	4	$	11	$	S
transition	2	0	4	0	S
transition	2	_	3	1	L
transition	9	0	9	0	L
transition	3	0	3	0	L
transition	3	1	3	1	L
transition	5	x	5	x	L
transition	4	y	4	y	L
transition	5	y	5	y	L
transition	9	y	9	y	L
transition	4	0	4	0	L
transition	4	1	4	1	L
transition	4	x	4	x	L
transition	10	_	5	_	L
transition	6	1	4	1	S
transition	1	0	1	0	R
transition	2	1	2	1	R
transition	6	x	6	x	R
transition	7	0	7	0	R
transition	7	y	7	y	R
transition	10	y	10	y	R
transition	1	1	2	0	R
transition	7	1	9	y	L

This Turing machine accepts strings of the form 0^n1^n.

Below are examples of accept, reject, quit.
garrison dave$ ./garrison_p1 ../sample_2.txt 00001111 100
$xxxxyyyy accept

garrison dave$ ./garrison_p1 ../sample_2.txt 0001111 100
$xxxyyy1 reject

garrison dave$ ./garrison_p1 ../sample_2.txt 0000111 100
$xxxxyyy reject

garrison dave$ ./garrison_p1 ../sample_2.txt 00001111 10
111 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 output string 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, more than one reject state, the start state not being state 0, the start state not being on the first line, gaps in the state numbers listed in the input file, missing state lines for states that are not the start, accept or reject states.