COMP 4400 Automata and Language Design
Program#2 –NPDA
Due Date: 05/08/15

Overview:

You will need to implement a general purpose NPDA in C++ that can read the description of the NPDA from an input file and construct the NPDA. The input file will be assumed to store the NPDA description in a standard format. Two sample files are given – one for the language $L = \{a^n b^n : n \geq 0\}$ and the other for the language $L = \{ww^R : w \in \Sigma^*\}$.

You will modify and extend the DFA class from Program#1 to create the NPDA class. The NPDA class will have the following data: a vector of states where each state will be represented as a string, a vector of alphabet symbols where each symbol will be represented as a string, a vector of stack alphabet symbols where each symbol will be represented as a character, a map (key-value) of transitions $\left(\delta: Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to (Q \times \Gamma^*)\right)$ where the "key" will be a 3-tuple (a string representing the current state of the automata, a string representing the input symbol being processed, and a character representing the stack top symbol) and the "value" will be a vector of pairs (each pair is a string representing the next state and a string representing the sequence of stack symbols that will replace the stack top), and a vector of final states where each final state will be represented as a string.

The NPDA class will have the following member functions: a function for adding a given state to the automata, a function for adding a given alphabet symbol, a function for adding a given stack alphabet symbol, a function to add a transition, a function to set the current state to the initial state, a function to add a final state, a function to display the NPDA description (as shown in the sample execution), a function to make a transition from the current state to the next state for a given input symbol and stack top symbol, and a function to check if a given state is a final state.

Note that you may need other member data and functions to help with the processing of an input string.

The driver code should read a given input file (whose name will be provided from command line) and instantiate the NPDA. Then the program should display the DFA description in the format shown in the sample execution. Then the user will be continually prompted to enter an input string (until Ctrl^C is pressed to terminate the program) and the program will determine using the NPDA functions whether this string is accepted or rejected by the NPDA. The program should also display the sequence of all the instantaneous descriptions in the format shown in the sample execution. In case the input string contains an invalid symbol, the program should stop processing the rest of the input. Note that you will need to use recursive backtracking to process an input string through the NPDA:

```
bool process()
    if no more input symbol to read and the current state is a final state
        return true
    for each possible next state
        if process() = true
            return true
    return false
```

Deliverables:

- You will need to submit the C++ source file(s) in Blackboard by 05/08/15.

Instructions:

- **YOU WILL LOSE 50% OF THE POINTS IF YOUR PROGRAM DOES NOT COMPILE IN LINUX.**
- Use meaningful identifiers, sufficient and helpful comments, and a consistent coding style.
- This will be an individual assignment. However, feel free to contact the instructor (and ONLY the instructor) if you need help. You may discuss general aspects of the assignment with your classmates, but you may not collaborate in any way in producing code. FAILURE TO FOLLOW THESE REQUIREMENTS WILL RESULT IN A GRADE OF F IN THE ASSIGNMENT.

Grading:

This assignment will be graded as follows:

Program Correctness	90%
Constructing the NPDA (60%)	
Processing the Input (30%)	
Program Style	10%

Sample Executions:

```
$ ./prog2
usage: <prog_name> <file_name>

$ ./prog2 foo
Automata file could not be opened!

$ ./prog2 test4.txt
---------N P D A---------
<states>
q0 q1 q2 q3
<alphabet>
a b
<stack alphabet>
0 1
<transitions>
(q0,*,0)->(q3,*)
(q0,a,0)->(q1,10)
(q1,a,1)->(q1,11)
(q1,b,1)->(q2,*)
(q2,*,0)->(q3,*)
(q2,b,1)->(q2,*)
<initial state>
q0
<stack start>
0
<final states>
q3
-------------------------

Enter a string to process (Ctrl^C to end): aabb

(q0,aabb,0)
|- (q1,abb,10)
|- (q1,bb,110)
|- (q2,b,10)
|- (q2,*,0)
|- (q3,*,*)
Accepted

Enter a string to process (Ctrl^C to end): aaa

Rejected

Enter a string (Ctrl^C to end):

(q0,*,0)
|- (q3,*,*)
Accepted
```