Answer the questions in the spaces provided. If you run out of room for an answer, continue on the back of the page.

Question	Points	Score
1	69	
Total:	69	

Name: ____

Section: ____

1. (69 points) **Pushdown Automata**

A point breakdown for this question is as follows:

Functionality	Point value
Quitting	2
Final and start states	3
Removing states	3
Prompts for initial creation of NDPA	4
Adding states	4
Removing transitions	4
User interface quality	5
Display of NDPA after each change	5
Adding transitions	10
Underlying NDPA structure	14
Testing custom strings	15
Total	69

Preliminary definitions:

- ε is the empty string. It contains no characters and has length 0.
- The $\circ$ symbol is what is known as the string concatenation operator. For example, $a \circ b = ab$.
- The $*$ symbol is what is known as the Kleene star. It is a unary operation defined as follows:

$$\begin{aligned} A^0 &= \{\varepsilon\} \\ A^1 &= A \\ A^{i+1} &= \{w \circ v : w \in V^i \wedge v \in V\} \quad \forall i \geq 1 \\ A^* &= \bigcup_{i \geq 0} V^i \end{aligned}$$

Note that A^*, for set of strings A, is necessarily a (countably) infinite set of finite-length strings.

A **nondeterministic pushdown automaton** (NPDA) is a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, s, \perp, F),$$

where

- Q is a finite set (the *states*),
- Σ is a finite set (the *input alphabet*),
- Γ is a finite set (the *stack alphabet*),
- $\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\} \times \Gamma) \times (Q \times \Gamma^*))$ is a finite *transition relation*,
- $s \in Q$ (the *start state*),
- $\perp \in \Gamma$ (the *initial stack symbol*), and
- $F \subseteq Q$ (the *final* or *accept states*).

If

$$((p, a, A), (q, B_1 B_2 \cdots B_k)) \in \delta,$$

this means intuitively that whenever the machine is in state p reading input symbol a on the input tape and A is on top of the stack, the NDPA can pop A off the stack, push $B_1 B_2 \cdots B_k$ onto the stack (B_k first and B_1 last), move its read head right one cell past the a, and enter state q.

We define *acceptance* of a string by a given NPDA in one of two ways. Either the NPDA accepts by *final state* (i.e., it accepts if, when it has finished reading a string, the current state of the machine is a final state), or by empty stack (i.e., if at any time during the evaluation of the string, the stack is emptied via pop operations). For this question, take all NPDAs to accept by final state rather than empty stack.

Your task: Implement a CLI for creation of NPDAs in Python. Your CLI should ask the user for their desired input alphabet, stack alphabet, and initial stack symbol, and should allow users to add and remove states and transitions, as well as declare states to be start states and/or final states. Finally, the user should be able to check their NPDA against a string to see whether it accepts it.

An example session in your CLI should look something like this:

```
Welcome to the NPDA creation CLI.

Please enter an input alphabet, as lowercase characters separated by spaces
(the empty string is denoted as ‘e’ internally, so don’t use that!):
> a b
Please enter a stack alphabet, as uppercase characters separated by spaces:
> A B C
Please choose an initial stack symbol from those you gave above:
> C

Enter a command of the following form:
[add/remove/[test string]/quit]
    [[state name [start] [final]]/[transition in-state out-state on-char]]
> add state A start final
You have added a state: A
Your NPDA now looks like this:

Q = { A }
Sigma = { a b }
Gamma = { A B C }
delta = { }
s = A
stack-final = C
state-final = A

Enter a command of the following form:
[add/remove/[test string]/quit]
    [[state name [start] [final]]/[transition in-state out-state on-char]]
> test aa

Testing...
Your string is accepted by the NPDA!

Enter a command of the following form:
[add/remove/[test string]/quit]
    [[state name [start] [final]]/[transition in-state out-state out-stack on-char on-stack]]
> add transition A A a e C
You have added a transition: A -- a,C --> A,e
Your NPDA now looks like this:

Q = { A }
Sigma = { a b }
Gamma = { A B C }
delta = { (A (a C) -> (A e)) }
s = A
stack-final = C
state-final = A
```

```
Enter a command of the following form:
[add/remove/[test string]/quit]
    [[state name [start] [final]]/[transition in-state out-state on-char]]
> quit
Quitting.
```

Your implementation should use Python's `input` function, in conjunction with `print`. You will likely want to use some scrap paper for this question.

Good luck!