Programming Languages and Paradigms
Spring 2022

Assignment 3

Submission Details

You can submit your solutions to this assignment in the OLAT course of PLP until Wednesday, April 27, 2022, at 11:59pm. Submit a single .zip file that includes all relevant files.

Note the following:

1. Source code files you submit should compile/run without errors.
2. Also include compiled binaries if there are any.
3. For each task, also submit screenshots of the task compiling and then running in your command line, IDE, or programming environment. For small tasks, feel free to include a single screenshot showing multiple tasks running one after another.
4. It's up to you how exactly you implement the programs internally, as long as you're using the assigned programming language. The tasks only describe the expected behavior of the programs.

1. Task: Text-based adventure game

In the previous assignment, you implemented a state machine that can parse and run `.machine` files. In this assignment, you will extend your previous solution to add additional features as outlined below.

a). Clearing the screen before transitioning

Your previous implementation continued printing machine states and transitions one after the other in your terminal output. Improve the implementation by clearing the terminal screen before executing a transition. This way, after each transition, the most recent transition text should be printed at the top of the terminal with the current state following and a prompt at the end.

b). Auto-forwarding states

It should be possible to mark a state as *auto-forwarding*. Such a state can only have one action/transition to leave the state, and when entering this state, the machine should automatically execute the that transition after a fixed timeout. To indicate such a state, there should be an additional flag `!` besides `*` and `+`, which is always followed by a strictly positive integer indicating the number of milliseconds this state will be shown before auto-forwarding to the next state. See the following example:

```
@*Peak{You're at the top of the hill. [slide] start sliding!}
@!Rolling1000{It's a steep slope!}
@!StillRolling1000{You can't stop!}
@+Bottom{You're at the bottom of the hill}
>Peak (slide) Rolling: You start sliding down the hill
>Rolling (slide) StillRolling: You're still sliding down the hill
>StillRolling (slide) Bottom: You reached the bottom!
```

Here, the 'Rolling' states have only one valid transition and they're both indicated as auto-forwarding. When running this machine, the user can `slide` at the top of the hill, but will keep getting automatically forwarded after 1 second until reaching the Bottom state. While in an auto-forwarding state, no input should be accepted from the user.

Note that the starting state can also be auto-forwarding (e.g., `@*!Start{}` or `@!*Start{}`). Your program should ensure that such states only have one transition specified and reject the machine file otherwise.

Contrary to the previous specification, note that state names only contain *alphabetical* characters, rather than alphanumerical ones, making it easy to separate the milliseconds where necessary.

c). Wildcard actions

Previously, entering an invalid action would always produce an error message. Now, it should be possible to use an asterisk (`*`) as a wildcard at the end of an action name to catch any actions that start with the given string but do not match any other actions. This could be used to transition to a different state or to simply customize the error message. An example:

```
> Select (Cancel) Ready: You cancel the transaction
> Select (Choose Cola) Dispense: You select a Cola
> Select (Choose Fanta) Dispense: You select a Fanta
# Custom error message for invalid action following 'Choose '
> Select (Choose *) Select: That's not sold here...
# Custom error message for any invalid action
> Select (*) Select: What are you doing??
# Any action starting with 'Smash with' will transition to Exit
> Select (Smash with *) Exit: You broke it!!!
```

Note that the most specific (longest matching) action should be executed! So for example, in the above example, entering `Choose Pepsi` should print 'That's not sold here' while entering something arbitrary like `Walk away` (which does not match any action) should print 'What are you doing??' You can assume that * will not otherwise appear in action names and that there will only ever be zero or one asterisks at the end of an action name.