Programming Assignment 1: Event Driven String Processing

Due 3 Sep by 23:59 **Points** 200 **Submitting** an external tool
Available after 6 Aug at 18:00

Assessment

Weighting:	20% (200 Marks)
Task description:	In this assignment, you will be writing a simple Regex engine that uses ε-NFAs to match an input sequence.
Academic Integrity Checklist	Do ✓ Discuss/compare high level approaches ✓ Discuss/compare program output/errors ✓ Regularly commit/push your work and add comments/notes on you commits Be careful ? Code snippets from reference pages/guides/Stack Overflow must be attributed/referenced. ? Only use code snippets that do not significantly contribute to the exercise solution. Do NOT ✗ Submit code not solely authored by you. ✗ Post/share code on Piazza/online/etc. ✗ Give/show your code to others ✗ Set your GitHub repository to "Public"

Before you begin

This assignment will be marked using a combination of automated and manual techniques.

You must log your development progress

- During manual marking, we will look look at your development process. If we do not see a clear path to a solution
  (i.e. code changes and regular commits and comments on those commits **reflecting** your

learning to develop your implementation)
you **may forfeit up to 100 marks.**

- *An example case of forfeiting 100 marks would be the sudden appearance of working code with no prior evidence of your development process*.
- It is up to you to provide evidence of your development through regular commits and comments.

This assignment requires thought and planning. You need to start early to allow yourself time to think of what and how to test before modifying any code. Failing to do this is likely to make the assignment take far more time than it should.

Aims

- Apply Event-Driven Techniques to an Input Stream.
- Use FSAs (ε-NFAs, NFAs and/or DFAs) to evaluate the inputs against for a given Regular Expression.

Overview

In this assignment, you will be writing code to simulate a FSA based regular expression engine.

The system you develop will need to generate the FSA structures that match a given Regex dynamically and then evaluate several series' of inputs against that FSA to determine whether they are a match.

This assignment requires you to apply concepts and techniques covered in Chapters 0-4 of the Notes on FSAs (see course readings).

You'll also need to document and thoroughly test your code using unit tests to ensure robust event handling techniques are being used.

Your Task

The system will need to:

1. Parse an basic regular expression from standard input
2. Generate a ε-NFA to evaluate the regular expression
3. Evaluate the subsequent inputs against the regular expression.

Event Driven systems are particularly prone to errors so you'll also need to:

1. Document your planning of the system (See Logging/Documenting your Progress below)
2. Write your own suite of test cases for your code (See Testing your Code below)

Programming Language/Software Requirements

Version Control System

- Your work must be stored and submitted using GitHub
- You will need to log your progress by commenting on your commits

Programming Language

You will need to use Java & JUnit to complete this assignment.

- Your code will be run using JDK 11
- Your test code will be run using JUnit 4
- Your implementation must **not** use any of the programming language's inbuilt regular expression parsing/evaluation libraries/classes.
- Your implementation may use any other libraries/classes available in the standard JDK and JUnit, but **no other external libraries/classes**.
- Your programs will be executed with the commands shown below:
  - Compiled with
    ```
    javac RegexEngine.java
    ```
  - Run with:
    ```
    java RegexEngine
    ```
    or for verbose mode
    ```
    java RegexEngine -v
    ```
  - Tests compiled with:
    ```
    javac *_Test.java
    ```
  - Each Test run with:
    ```
    java org.junit.runner.JUnitCore TestName_Test
    ```

Input Format

Your program will need to read input from the terminal/command line's standard input (`System.in`).

The expected input format is as follows:

```
(ab)*|c+
abc
ccc
```

1. The input begins with **the regular expression to test.**
   - In this assignment, a regular expression can consist of lower and upper case letters, numbers, spaces, the alternation operator (`|`), the Kleene star and Kleene plus operators (`*` and `+`), as well as brackets (`(` and `)`).
   - You are not expected to handle nested brackets.
   - Invalid input in this section should cause the program to print an error message and exit with an exit code of 1.
2. The next section of input contains the input strings to evaluate against the regular expression.
   - Each string is on its own line; when a new line is entered, another string begins (don't forget to reset states)
   - A string can be empty/blank.
   - A string can contain whitespace and other control characters.
3. This repeats until the program is terminated with a SIGTERM (Ctrl-C).

Expected Output Format

The program has 2 output modes; a **normal mode** and **verbose mode**.
Verbose should trigger if the user adds a `-v` when running the program (see Software Requirements above)

Under **normal mode**

After the user has entered the regex your program should print `ready`

Your program should then print `true` or `false` once for each input string provided as they are entered.

- If the input string matches the regular expression, print `true`
- If the input string does not match, print `false`
- The input string should be evaluated for an exact match
  - i.e. the whole string from start to end must exactly match the regular expression to be accepted.
  - A string containing a match, but not exactly matching should print `false`.

Example (user input is grey, output is highlighted pink)

```
(ab)*|c+
ready
abc
false
ccc
true
```

Under **verbose mode**

After the user has entered the regex your program should print a transition table for the ε-NFA generated followed by the word `ready`

- The transition table does not have to exactly match the example below, but does need to reflect the ε-NFA generated by your system;
  this is for manual review and your own debugging.

Your program should begin to print `true` or `false` as the user enters the input strings for each character input (including the initial state),

- If the system is currently in an accepting state, print `true`
- Otherwise, print `false`
- Hint: After a user has entered the input string the most recent output should match the normal output

Example (user input is grey, output is highlighted pink)

```
(ab)*|c+
    epsilon a   b   c   other
>q0 q1,q5
q1  q2,q8
q2          q3
q3              q4
q4  q1
q5  q6
q6                  q7
q7  q5,q8
*q8

ready
true
a
false
b
true
c
false

true
c
true
c
true
c
true
```

Submission, Assessment & Marking

Submission

Before submitting

- You will need a github.com account to submit

- Be sure to set any repository you create for this assignment to **private**.
- Your source files must either be in the root of your repository, or in a `/src` folder

To submit

1. Click the Load Programming Assignment ... in a new window button at the bottom of this page
2. Select the Submit button on that page
3. When prompted, link your Github account
4. Select the repository and branch that you want to submit
5. Submit and wait for the system to check your work.

Your submission will run through some acceptance tests and provide basic feedback. Full testing comes after the assignment deadline and will be done using a combination of automatic and manual tests.

Assessment

The assignment is marked out of 200. These 200 marks are allocated as follows using a combination of automated testing and manual review:

- Basic Implementation **[70 marks]**
  - The system can handle basic regular expressions consisting of single operations
  - These regular expressions generate a reasonable ε-NFA
  - Inputs are correctly evaluated against that ε-NFA
- More complex expressions **[70 marks]**
  - The system can handle regular expressions consisting of multiple operations.
  - Order of operations is correctly handled.
  - These regular expressions generate a reasonable ε-NFA
  - Inputs are correctly evaluated against that ε-NFA
- Brackets and nested expressions **[30 marks]**
  - The system can handle regular expressions containing brackets (max 1 layer deep) and chained operators.
  - Order of operations is correctly handled.
  - These regular expressions generate a reasonable ε-NFA
  - Inputs are correctly evaluated against that ε-NFA
- Edge Cases **[30 marks]**
  - The system is otherwise functional and correctly handles edge cases and erroneous input

However, your marks are **scaled and reviewed** based on the following:

1. Up to 10 marks may be deducted for poor code quality.
   - When in doubt, consider **Google's Java Code Style Guide (https://google.github.io/styleguide/javaguide.html)** .
   - Javadoc comments are always recommended.
2. Up to 100 marks may be deducted for poor or missing testing.

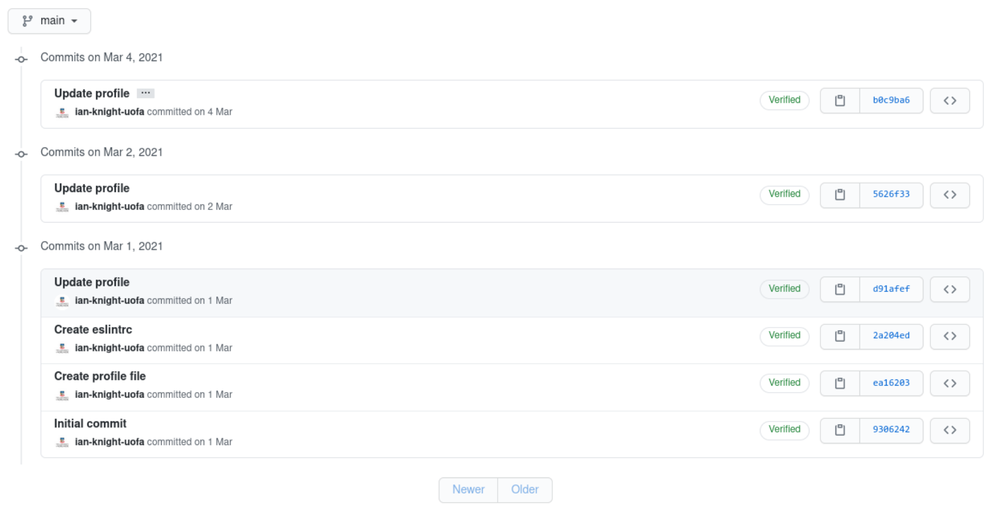
- Your test code will be reviewed against a code coverage checker. Poor test coverage will result in lost marks.

3. Up to 100 marks may be deducted for poor/insufficient/missing evidence of development process.

Logging/Documenting your Progress

We'll be using GitHub to log our development progress this semester.

Each time you make a commit and push those changes, they will be visible in your commit history:



main

Commits on Mar 4, 2021

Update profile ··· — ian-knight-uofa committed on 4 Mar — Verified — b0c9ba6

Commits on Mar 2, 2021

Update profile — ian-knight-uofa committed on 2 Mar — Verified — 5626f33

Commits on Mar 1, 2021

Update profile — ian-knight-uofa committed on 1 Mar — Verified — d91afef

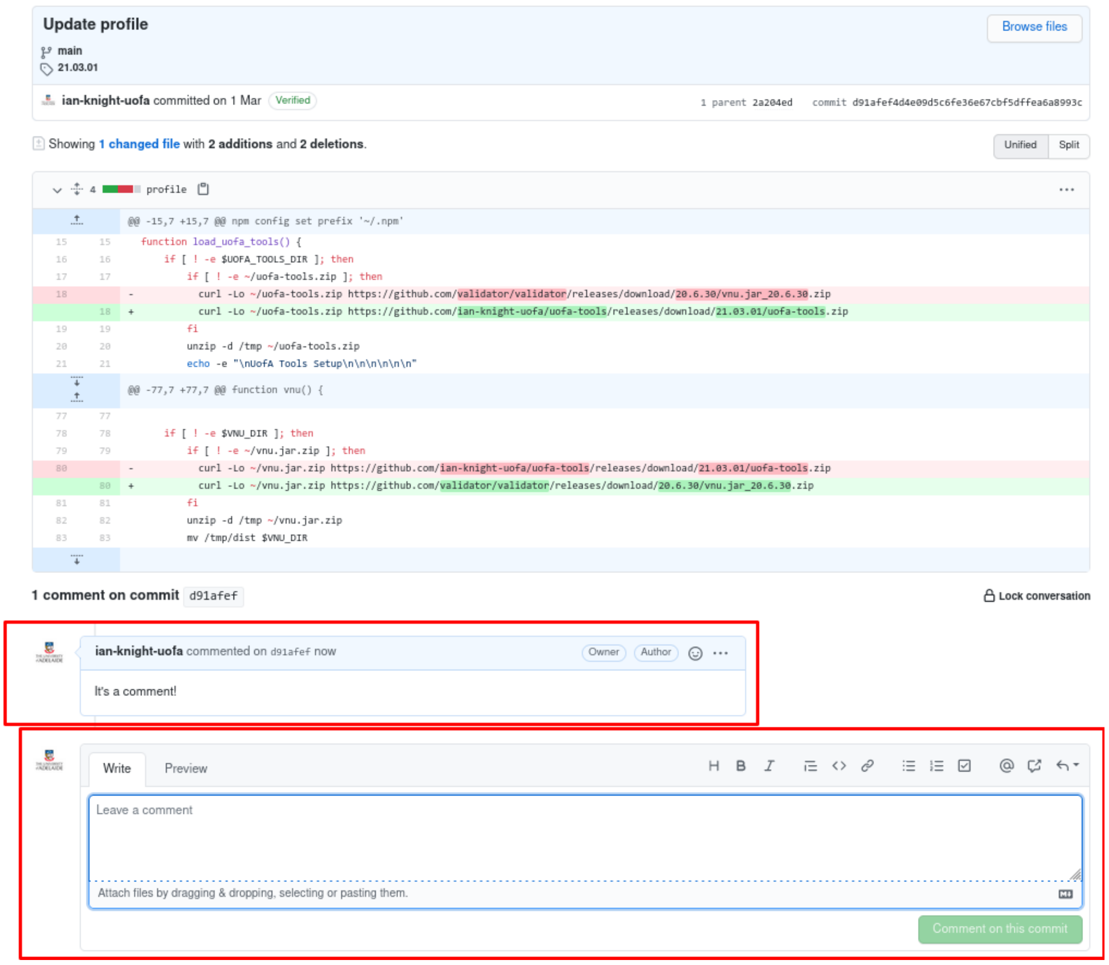
Create eslintrc — ian-knight-uofa committed on 1 Mar — Verified — 2a204ed

Create profile file — ian-knight-uofa committed on 1 Mar — Verified — ea16203

Initial commit — ian-knight-uofa committed on 1 Mar — Verified — 9306242

Newer Older

Each commit can have additional notes and comments added to it:


Update profile

main
21.03.01

ian-knight-uofa committed on 1 Mar Verified

1 parent 2a204ed commit d91afef4d4e09d5c6fe36e67cbf5dffea6a8993c

Showing 1 changed file with 2 additions and 2 deletions.

```
4 profile

@@ -15,7 +15,7 @@ npm config set prefix '~/.npm'
   function load_uofa_tools() {
       if [ ! -e $UOFA_TOOLS_DIR ]; then
           if [ ! -e ~/uofa-tools.zip ]; then
-            curl -Lo ~/uofa-tools.zip https://github.com/validator/validator/releases/download/20.6.30/vnu.jar_20.6.30.zip
+            curl -Lo ~/uofa-tools.zip https://github.com/ian-knight-uofa/uofa-tools/releases/download/21.03.01/uofa-tools.zip
           fi
           unzip -d /tmp ~/uofa-tools.zip
           echo -e "\nUofA Tools Setup\n\n\n\n\n\n"
@@ -77,7 +77,7 @@ function vnu() {

       if [ ! -e $VNU_DIR ]; then
           if [ ! -e ~/vnu.jar.zip ]; then
-            curl -Lo ~/vnu.jar.zip https://github.com/ian-knight-uofa/uofa-tools/releases/download/21.03.01/uofa-tools.zip
+            curl -Lo ~/vnu.jar.zip https://github.com/validator/validator/releases/download/20.6.30/vnu.jar_20.6.30.zip
           fi
           unzip -d /tmp ~/vnu.jar.zip
           mv /tmp/dist $VNU_DIR
```

1 comment on commit d91afef

Lock conversation

ian-knight-uofa commented on d91afef now Owner Author

It's a comment!

Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Comment on this commit

Testing your Code

We'll be using JUnit to test our code

Take a look at their Getting Started guide: https://github.com/junit-team/junit4/wiki/Getting-started (https://github.com/junit-team/junit4/wiki/Getting-started)

You will be reviewed on your test coverage, which will be done using a combination of manual review and automated test coverage reports.

This tool needs to be loaded in a new browser window

Load Programming Assignment 1: Event Driven String Processing in a new window