

Project 2 Readme Onyx

1	Team Name: Onyx																		
2	Team members names and netids Freeman Nkouka jnkouka																		
3	Overall project attempted, with sub-projects: Program 3 (A TM where "Left" causes the heap to move all the way left)																		
4	Overall success of the project: Complete, all working																		
5	Approximately total time (in hours) to complete: 8																		
6	Link to github: https://github.com/Freemankn/Project2-TOC																		
7	List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary). Add more rows as necessary.																		
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="padding: 5px;">File/folder Name</th> <th style="padding: 5px;">File Contents and Use</th> </tr> </thead> <tbody> <tr> <td colspan="2" style="text-align: center; padding: 5px;">Code Files</td></tr> <tr> <td style="padding: 5px;">src/helpers/argument_input_onyx.py</td><td style="padding: 5px;">Gets the TM file and input string from the user</td></tr> <tr> <td style="padding: 5px;">src/helpers/turing_machine_onyx.py</td><td style="padding: 5px;">Contains the TuringMachineSimulator abstract class. This class has a function that parses a TM csv file.</td></tr> <tr> <td style="padding: 5px;">src/reset_tm_onyx.py</td><td style="padding: 5px;">Contains the ResetTuringMachineSimulator abstract. This class has function the runs the TM with such that left direction resets the head to the beginning.</td></tr> <tr> <td style="padding: 5px;">main.py</td><td style="padding: 5px;">Runs the entrypoint file</td></tr> <tr> <td style="padding: 5px;">src/entrypoint_onyx.py</td><td style="padding: 5px;">Runs program 3 using the files provided by the user.</td></tr> <tr> <td colspan="2" style="text-align: center; padding: 5px;">Test Files</td></tr> <tr> <td style="padding: 5px;">input/reset_tm_ab_equal.csv</td><td style="padding: 5px;">Simulates a TM that decided the language, $L = \{a^n b^n \mid n \geq 1\}$. The states are: $\{q1, q2, q3, q4\}$</td></tr> </tbody> </table>		File/folder Name	File Contents and Use	Code Files		src/helpers/argument_input_onyx.py	Gets the TM file and input string from the user	src/helpers/turing_machine_onyx.py	Contains the TuringMachineSimulator abstract class. This class has a function that parses a TM csv file.	src/reset_tm_onyx.py	Contains the ResetTuringMachineSimulator abstract. This class has function the runs the TM with such that left direction resets the head to the beginning.	main.py	Runs the entrypoint file	src/entrypoint_onyx.py	Runs program 3 using the files provided by the user.	Test Files		input/reset_tm_ab_equal.csv	Simulates a TM that decided the language, $L = \{a^n b^n \mid n \geq 1\}$. The states are: $\{q1, q2, q3, q4\}$
File/folder Name	File Contents and Use																		
Code Files																			
src/helpers/argument_input_onyx.py	Gets the TM file and input string from the user																		
src/helpers/turing_machine_onyx.py	Contains the TuringMachineSimulator abstract class. This class has a function that parses a TM csv file.																		
src/reset_tm_onyx.py	Contains the ResetTuringMachineSimulator abstract. This class has function the runs the TM with such that left direction resets the head to the beginning.																		
main.py	Runs the entrypoint file																		
src/entrypoint_onyx.py	Runs program 3 using the files provided by the user.																		
Test Files																			
input/reset_tm_ab_equal.csv	Simulates a TM that decided the language, $L = \{a^n b^n \mid n \geq 1\}$. The states are: $\{q1, q2, q3, q4\}$																		

	<p>qacc,qrej}. The alphabet is: {a,b}. Gamma is: {a,b,X,Y,_}. The start is q1. qacc is the accept state. qrej is the reject state.</p>
Output Files (None because simulation is printed to the screen)	
Plots (as needed)	
8	Programming languages used, and associated libraries: Python
9	Key data structures (for each sub-project): Program 3: Lists
10	General operation of code (for each subproject) Program 3: The first four lines of the program display the machine name, the input string, the start state, and what the symbol the head points to. An infinite while loop is then used to read the current character under the head and iterate through the list of transitions to determine what the Turing machine should do next. After each step, the current state of the tape is printed. If the direction is L, the head is reset to the beginning of the tape (index 0). If the machine reaches either the accept or reject state, the loop terminates and the program prints whether the input string was accepted or rejected. Finally, as a safety check, if the Turing machine has not halted after 75 steps, the loop is terminated to prevent an infinite loop.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I tested my simulator using a Turing machine that decides the language $L = \{ a^n b^n \mid n \geq 1 \}$. To verify correctness, I created three categories of input: <ul style="list-style-type: none"> • More a's than b's (e.g., "aaab"), which the TM should reject. • More b's than a's (e.g., "abbb"), which should also be rejected.

	<ul style="list-style-type: none"> Equal numbers of a's and b's (e.g., "aabb"), which should be accepted. <p>I chose this particular TM because it has 13 transitions and 6 states, providing a long enough execution sequence to observe how my implementation handles state changes, tape updates, and the reset behavior for left moves.</p> <p>These cases confirmed that the simulator correctly identifies when the input string belongs to the language and properly rejects invalid patterns. They also demonstrated that the “reset to index 0” functionality works as intended whenever the TM moves left. Overall, the tests showed that my code can accurately simulate a TM with more complex transition logic.</p>
12	<p>How you managed the code development:</p> <p>I broke the project into small, testable milestones and built the solution incrementally. For example, in the run() function, I first verified the start state and transitions using print statements to confirm that the TM initialized correctly. Once that worked, I tested the behavior for the first input character to make sure the state progression was accurate. After confirming those results, I gradually expanded testing to longer inputs, validating each stage before moving on. This step-by-step approach helped isolate issues early and ensured each component worked before integrating the next.</p>
13	<p>Detailed discussion of results:</p> <p>The Turing Machine correctly determined whether a string should be accepted or rejected. One behavior that stood out during testing was how the machine handled transitions that required moving left. Instead of stepping left one cell at a time, the head returned directly to the beginning of the tape. This demonstrated that the left-direction logic was functioning similarly to a reset operation.</p>
14	<p>How the team was organized:</p> <p>Since I was the only member of the team, I focused on managing my time effectively and seeking clarification early. I made a point to ask the professor questions as soon as uncertainties arose, which helped me avoid getting stuck later. This allowed me to debug issues proactively and keep steady progress throughout the project.</p>
15	<p>What you might do differently if you did the project again</p> <p>If I were to redo the project, I would focus more on optimizing the efficiency of my code. I initially chose to use a list because I was comfortable with its operations, but in retrospect, a dictionary may have been a better fit for the way the data was processed.</p>
16	<p>Any additional material: Screenshots of output</p> <p>Testing aabb using reset_tm_ab_equal.csv</p>

```
PS C:\Users\nkouk\OneDrive\Documents\Discrete 2.0\Project2-TOC> python .\main.py .\input\reset_tm_ab_equal.csv aabb
Start State: q1

Step: 0
State: q1
a a b b _
^

Step: 1
State: q2
X a b b _
^

Step: 2
State: q2
X a b b _
^

Step: 3
State: q3
X a Y b _
^

Step: 4
State: q1
X a Y b _
^

Step: 5
State: q1
X a Y b _
^

Step: 6
State: q2
X X Y b _
^

Step: 7
State: q2
X X Y b _
^

Step: 8
State: q3
X X Y Y _
^

Step: 9
State: q1
X X Y Y _
^

Step: 10
State: q1
X X Y Y _
^

Step: 11
State: q1
X X Y Y _
^

Step: 12
State: q4
X X Y Y _
^

Step: 13
State: q4
X X Y Y _
^

Result: Accepted
```

Testing aab using reset_tm_ab_equal.csv

```
PS C:\Users\nkouk\OneDrive\Documents\Discrete 2.0\Project2-TOC> python .\main.py .\input\reset_tm_ab_equal.cs
● v aab
Machine: reset-a^nb^n-equal
Input: aab
Start State: q1

Step: 0
State: q1
a a b _
^
Step: 1
State: q2
X a b _
^
Step: 2
State: q2
X a b _
^
Step: 3
State: q3
X a Y _
^
Step: 4
State: q1
X a Y _
^
Step: 5
State: q1
X a Y _
^
Step: 6
State: q2
X X Y _
^
Step: 7
State: q2
X X Y _
^
Result: Rejected
```

Testing abb using reset_tm_ab_equal.csv

```
PS C:\Users\nkouk\OneDrive\Documents\Discrete 2.0\Project2> python .\main.py .\input\reset_tm_ab_equal.cs
● v abb
Machine: reset-a^nb^n-equal
Input: abb
Start State: q1

Step: 0
State: q1
a b b _
^
Step: 1
State: q2
X b b _
^
Step: 2
State: q3
X Y b _
^
Step: 3
State: q1
X Y b _
^
Step: 4
State: q1
X Y b _
^
Step: 5
State: q4
X Y b _
^
Result: Rejected
```