**CSC615M - Machine Project 4 Documentation**

Aquino, Kurt Neil

De La Salle University, Manila

254 Sitio Bagong Anyo, Pagsawitan, Santa Cruz, Laguna

(+63) 9273597974

kurt\_aquino@dlsu.edu.ph

1. **INTRODUCTION**

For the fourth learning output for the course of CSC615M – Automata Theory, Computability, Formal Languages, the students were tasked to design a software which would simulate the process of running a Turing Machine based on specified programs given an input.

As discussed in class, Turing machines, as first described by Alan Turing, are simple and abstract computational devices intended to help investigate the extent and limitations of what can be computed. Turing stated that every algorithmic problem solution may be represented by a program of instructions modelled as a Turing Machine (Denning, P.J, Dennis, J.B., & Qualitz, J.E., 1978).

Overall, this project aims is to simulate more complex Turing Machines made up of a combination of simpler Turing Machines which are described in *Section 2.1.*

1. **SOFTWARE DESIGN**
   1. **Inputs**

As the project aims to simulate the process of performing the operations of a specified Turing Machine, the inputs to be accepted will be the machine’s corresponding components, namely:

* The set of elementary instructions, which are denoted by:
* An input tape of integers delimited by ‘#’:

For the user’s ease of access, the inputs will be saved in a text file, which follows the same formatting of the listed components of the machine. This is done in order to avoid the repetitive process of asking the user to type in the necessary components every single time the user would want to run the system.

The descriptions of the defined elementary instructions are as follows:

* shL,k/ shR,k – shifts the tape cursor k numbers to the left/right
* copy,k – copies the kth integer to the left of the tape cursor and positions the integer to the right of the current position of the tape cursor. Positions the tape cursor to the right of the newly copied integer.
* const,k – positions a constant k to the right of the tape cursor. The tape cursor remains in the same index.
* move.m,n – removes the integers m numbers to the left of the tape cursor and moves the integers n numbers from the right of the tape cursor to the left. Positions the tape cursor at the left of the newly moved n set of integers.
* pushL – moves the integers 1 position to the left. The tape cursor remains in the same index.
* inc – integer at the right of the cursor is copied then incremented. Positions the tape cursor at the left of the incremented value.
* dec - integer at the right of the cursor is copied then decremented. Positions the tape cursor at the left of the decremented value.
* add – adds the two integers at the right of the tape cursor, removes the two integers, and replaces them with the sum. The tape cursor remains in the same index.
* mult - multiplies the two integers at the right of the tape cursor, removes the two integers, and replaces them with the product. The tape cursor remains in the same index.
* monus/minus - subtracts the two integers at the right of the tape cursor, removes the two integers, and replaces them with the result. If the first integer is less than the second integer, a 0 is set as the result. The tape cursor remains in the same index.
* swap – swaps the values of the two integers at the right of the tape cursor. The tape cursor remains in the same index.
* goto,k – redirects the tape cursor to the kth command unconditionally.
* gotoEQ,k – checks if the first integer to the right of the tape cursor is equal to the second integer. Proceeds to the kth command/instruction if true.
* gotoNE,k – checks if the first integer to the right of the tape cursor is not equal to the second integer. Proceeds to the kth command/instruction if true.
* gotoGE,k – checks if the first integer to the right of the tape cursor is greater than or equal to the second integer. Proceeds to the kth command/instruction if true.
* gotoGT,k – checks if the first integer to the right of the tape cursor is greater than the second integer. Proceeds to the kth command/instruction if true.
* gotoLE,k – checks if the first integer to the right of the tape cursor is less than or equal to the second integer. Proceeds to the kth command/instruction if true.
* gotoLT,k – checks if the first integer to the right of the tape cursor is less than the second integer. Proceeds to the kth command/instruction if true
  1. **Algorithm and Implementation**

In order to simulate the functionality of a Turing Machine, the corresponding implementation of each of the machine’s components should first be discussed.

* + 1. **File Reader**

This component of the system is mainly involved with the reading of the given text files and then storing their corresponding values to the machine’s components.

The file readers are separated into 2 parts: One for the input program, and another for the input tape. The corresponding notations of the 2 components are defined in *Section 2.1.*

Sample input program components: (Swap Program)

1,shR,2

2,copy,2

3,shL,2

4,move,1,2

5,shL,1

6,halt

Sample input tape components:

4,3

* + 1. **Command, and State Objects**

This component focuses on the storing and manipulation of the states and their corresponding transitions as according to the specified commands of the Turing Machine.

Command Objects are essentially the program instructions to be performed by the Turing Machine and are defined in the system to have the following components:

* commandID – sequence ID or label of the command in the Turing Machine
* commandName – name of the elementary instruction to be performed
* params – set of parameters to be used accordingly depending on the commandName specified

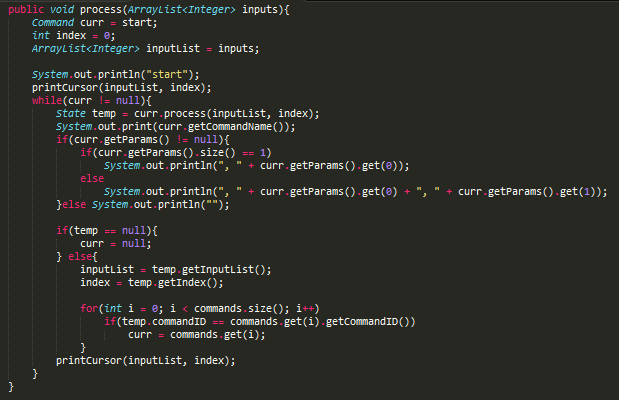
Command Objects also contain the respective functionalities to be executed by the Turing Machine depending on the object’s assigned commandName as defined in *Section 2.1.*

State Objects basically stores the current state of the input tape as well as the sequence ID of the next instruction to be performed by the Turing Machine. These are defined in the system to have the following components:

* inputList – current state of the input tape
* index – current index of the tape cursor
* commandID – sequence ID of the succeeding instruction to be performed
  + 1. **Turing Machine**

This component of the system simulates to process of performing the functionality of a Turing Machine given its corresponding set of instructions and input tape.

The function essentially scans through the specified set of instructions to be performed by the Turing Machine, and applies the corresponding functions accordingly on the input tape. The direction on which the tape cursor proceeds through the input tape depends on the instruction being performed. The system will apply the necessary modifications to the input tape and will then present the resulting tapes as well as the current position of the tape cursor for every instruction followed. The final resulting tape will be presented once all of the instructions have been performed or when the Turing Machine has reached a “HALT” state.

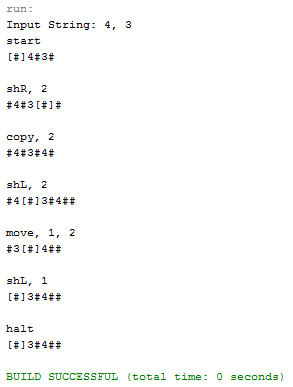


*Figure 2.1. Code for simulating a Turing Machine given an input tape and a set of programs*

* 1. **Output**

The system outputs the resulting tapes after every instruction being performed in the Turing Machine, as well as the final resulting tape after all of the operations have been performed or when the Turing Machine has reached a “HALT” state.

Sample output of the program stated in *Section 2.2.1:*



*Figure 2.2. Output for the Swap Turing Machine on the input 4,3*

1. **TEST CASES**
   1. **List Down**

Given the input tape: #3#, and the input program:

1,shR,1

2,copy,1

3,const,1

4,shL,1

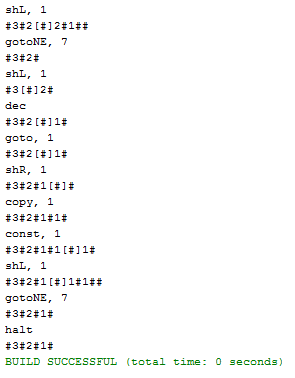
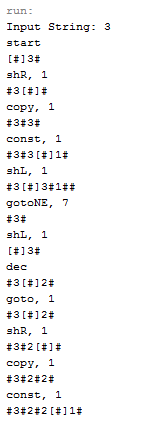
5,gotoNE,7

6,halt

7,shL,1

8,dec

9,goto,1



*Figure 3.1. Output for the List Down Turing Machine on the input tape #3#*

* 1. **Square**

Given the input tape: #25#, and the input program:

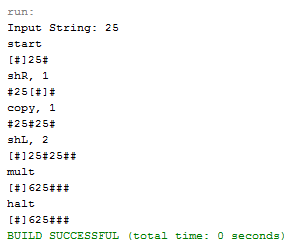
1,shR,1

2,copy,1

3,shL,2

4,mult

5,halt



*Figure 3.2. Output for the Square Turing Machine on the input tape #25#*

* 1. **Square Root**

Given the input tape: #4# and the input program:

1,shR,1 11,gotoNE,16

2,const,1 12,shL,1

3,shR,1 13,pushL

4,copy,1 14,shL,1

5,copy,1 15,halt

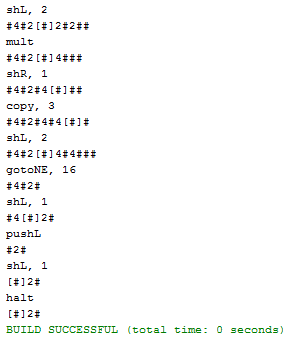
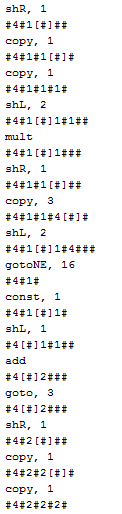
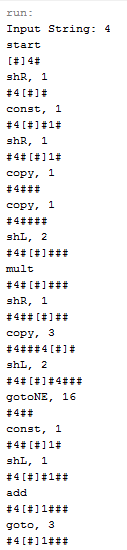
6,shL,2 16,const,1

7,mult 17,shL,1

8,shR,1 18,add

9,copy,3 19,goto,3

10,shL,2



*Figure 3.3. Output for the Square Root Turing Machine on the input tape #4#*

1. **CONCLUSION**

Overall, the system was successfully implemented. As proven by the test cases of varying scenarios, the system can successfully simulate a Turing Machine. It should be noted however that the system assumes that the input tapes as well as the input programs are correct and would produce the expected results. Using incompatible inputs, either tape or program, may cause the Turing Machine to produce inconsistent or unreliable results.

1. **REFERENCES**

[1] Denning, P.J., Dennis, J.B., & Qualitz, J.E. (1978). *Machines, languages, and computation*. New Jersey: Prentice Hall.

1. **ASSESSMENT RUBRIC**

|  |  |
| --- | --- |
| **Criteria** | **Score** |
| Input-Output Modules | **29** |
| Core Process | **30** |
| Quality of testing | **18** |
| Documentation | **20** |