

DESIGN A TURING MACHINE USING JAVA, TO IMPLEMENT BASIC OPERATIONS OF TM

BY

GROUP 6

A 300L RESEARCH PROJECT SUBMITTED TO THE DEPARTMENT OF MATHEMATICS, COMPUTER SCIENCE AND PHYSICS.

IN PARTIAL FUFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF BACHELOR OF SCIENCE B.Sc,(Honours), COMPUTER SCIENCE AND CYBER SECURITY.

JANUARY, 2024

# **GROUP MEMBERS**

|  |  |  |
| --- | --- | --- |
| NAMES | MATRIC NO. | DEPARTMENT |
| TALABI DAMILARE HANNAH  FATOKUN DAMILOLA  DIVINE UKPAI  SANYA DANIEL AYOMIDE  NNEDU REBECCA  OLUMORIN AYOMIDE STEPHEN  OMEREONYE NNAEMEKA WINSTON | 21/7954  21/8666  21/8200  21/8760  21/9153  21/8479  21/9007 | COMPUTER SCIENCE  COMPUTER SCIENCE  COMPUTER SCIENCE  COMPUTER SCIENCE  COMPUTER SCIENCE  COMPUTER SCIENCE  COMPUTER SCIENCE |

# **TABLE OF CONTENTS**

[**GROUP MEMBERS** 2](#_Toc158892911)

[**TABLE OF CONTENTS** 3](#_Toc158892912)

[**OBJECTIVE** 3](#_Toc158892913)

[**INTRODUCTION** 4](#_Toc158892914)

[**BACKGROUND** 6](#_Toc158892915)

[**NOTATIONS AND TERMINOLOGIES** 10](#_Toc158892916)

[**APPLICATIONS** 12](#_Toc158892917)

[**ADVANTAGES OF TURING MACHINES** 13](#_Toc158892918)

[**DESIGN** 14](#_Toc158892919)

[**IMPLEMENTATION** 19](#_Toc158892920)

[**RESULT ANALYSIS** 21](#_Toc158892921)

[**CONCLUSION** 22](#_Toc158892922)

[**BIBLIOGRAPHY** 24](#_Toc158892923)

# 

# **OBJECTIVE**

A Turing machine is a theoretical machine that can perform any calculation that is algorithmically computable. It is often used as a theoretical model of computation and is considered as the theoretical foundation of modern computing.

The project's objective is to create a single-head deterministic Turing machine in Java that can process an input string and generate an output string in addition to deciding whether or not to accept it.

A single head deterministic Turing machine is a kind of Turing computer that follows deterministic rules and has a single reading/writing head. The machine does not have the ability to change its behavior based on random or non-deterministic choices. Numerous computational models in theoretical computer science are based on this kind of Turing machine.

Java was chosen specifically for its OOP structure, which allowed for the definition of classes for State, Transition, Machine, and other concepts, allowing for the encapsulation of different aspects of an entity within an object.

As an illustration, a transition is described as a class consisting of three characters: read, write, and shift. These characters store the index of the subsequent state that the machine should transition to, as well as the read symbol, write symbol, and shift direction, respectively. The same principle applies to State objects, which have possible transitions to other states.

To build a Java Turing machine, define a class that carries out the control logic and transition function of the machine. The machine's method of changing states in response to an input symbol and its current state is defined by the transition function. The tape head's movement and the symbols' reading and writing are carried out by the control logic.

# **INTRODUCTION**

A Turing machine is a mathematical model of computation that describes an abstract machine that can manipulate symbols on a tape according to predefined rules, making it a powerful and general model of computation despite its simplicity.

It was invented by Alan Turing in 1936. Turing developed the machine while studying at Princeton University to define the limits of mechanical computation and answer the “Entscheidungsproblem.” The machine operates on an infinite tape divided into cells containing symbols, reading, writing, and moving the tape based on rules. Turing proved that no single machine could solve the “Entscheidungsproblem”, establishing undecidable problems. The Turing machine laid the foundation for modern computers and the Church-Turing thesis, stating that any effective computation can be performed by a Turing machine.

The Entscheidungsproblem, a 1928 mathematical and computer science challenge, aimed to find an algorithm that could definitively determine the truth of any mathematical statement. The problem, based on a formal language like mathematical logic, could automate theorem proving and answer open questions. However, in 1936, Alan Turing and Alonzo Church independently demonstrated that the problem is unsolvable, highlighting the limitations of mechanical computation and formalization, and leading to further research in computability theory.

Turing machines are a powerful computational model due to their infinite memory, which allows them to handle long input sequences. They can simulate common computers, making them a fundamental model for understanding computation. Turing machines can represent and model all computable functions, allowing researchers to explore computation limits. They can also model recursively enumerable languages, a broader class of languages than regular languages, allowing them to tackle a wide range of problems.

# **BACKGROUND**

A field of computer science and mathematics known as "the theory of computation" examines what issues can be resolved and how effectively they can be resolved by abstract models of computation like Turing machines.

A Turing machine is a theoretical device that follows a set of rules to manipulate symbols on an infinite tape. If given enough time and room, it can execute any computation that a real computer can. A Turing machine is formally defined by a 7-tuple (Q, Σ, Γ, δ, q0, qA, qR), where:

* Q is a finite set of states
* Σ is a finite input alphabet, not containing the blank symbol \_
* Γ is a finite tape alphabet, containing Σ and \_
* δ is a transition function, mapping Q × Γ to Q × Γ × {L, R}
* q0 is the initial state
* qA is the accept state
* qR is the reject state, distinct from qA

One way to think of a Turing machine is as a tape head that travels left or right based on the transition function, reads and writes symbols from the tape, and scans the tape. With the input string on the tape, the machine begins in the starting state and stops when it reaches the accept or refuse states. If the machine stops in the accept state, it accepts the input; if not, it rejects it.

EXAMPLES:

Construct TM for the addition function for the unary number system.

**Solution:**

The unary number is made up of only one character, i.e. The number 5 can be written in unary number system as 11111. In this TM, we are going to perform the addition of two unary numbers.

**For example**

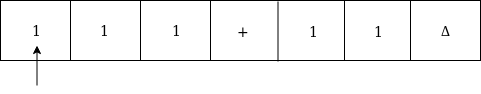
2+3  
i.e. 11 + 111 = 11111

If you observe this process of addition, you will find the resemblance with string concatenation function.

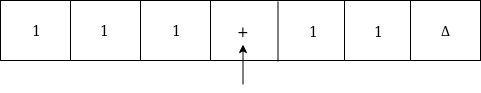
In this case, we simply replace + by 1 and move ahead right for searching end of the string we will convert last 1 to Δ.

**Input:** 3+2

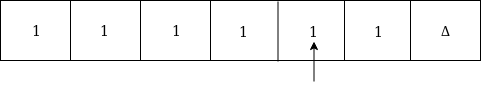
The simulation for 111+11Δ can be shown as below:



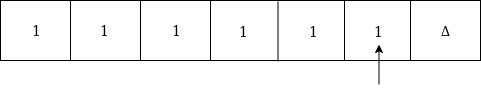
Move right up to + sign as:



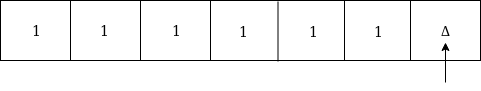
Convert + to 1 and move right as:



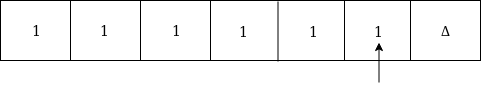
Now, move right



Again move right



Now Δ has encountered, so just move left as:



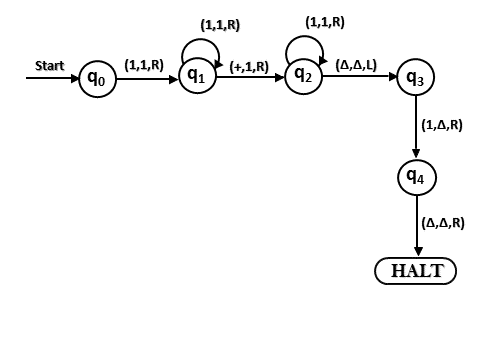
Convert 1 to Δ

Examples of TM

Thus the tape now consists of the addition of two unary numbers.

The TM will look like as follows:

Here, we are implementing the function of f(a + b) = c. We assume a and b both are non zero elements.



Some other examples of Turing machines for different tasks are:

* A Turing machine that recognizes the language L = {0n1n | n ≥ 0}, consisting of strings of equal numbers of 0s and 1s. The machine works by crossing out the first 0 and the last 1, and repeating this process until either the tape is empty (accept) or there is a mismatch (reject).
* A Turing machine that computes the subtraction of two unary numbers, separated by a - symbol, assuming the first number is larger than or equal to the second. The machine works by crossing out the first 1 before the - symbol and the first 1 after the - symbol, and repeating this process until either there are no more 1s after the - symbol (accept) or there are no more 1s before the - symbol (reject). The result is the difference of the two numbers in unary.
* A Turing machine that computes the 1’s complement of a binary number, i.e., flipping every bit. The machine works by scanning the tape from left to right, and replacing every 0 with a 1 and every 1 with a 0. The result is the 1’s complement of the input number
* A Turing machine that computes the 2’s complement of a binary number, i.e., adding 1 to the 1’s complement. The machine works by first computing the 1’s complement as above, and then adding 1 to the result using binary addition. The result is the 2’s complement of the input number.

# **NOTATIONS AND TERMINOLOGIES**

**1. Input alphabet (Σ):** The input alphabet is the set of symbols that can appear on the input tape. For example, in a Turing machine that processes binary input, the input alphabet might consist of the symbols {0, 1}.

**2. Tape alphabet (Γ):** The tape alphabet is the set of symbols that can appear on the tape, including the input alphabet as well as any special symbols such as blanks. For example, in a binary Turing machine, the tape alphabet might consist of the symbols {0, 1, \_}, where "\_" represents a blank symbol.

**3. Start state (q0):** The start state is the initial state of the Turing machine when it begins its computation. It is the state in which the machine starts processing the input.

**4. Accept state (qA):** In some models of Turing machines, there may be an accept state that, when reached, indicates that the Turing machine has successfully completed its computation and accepted the input. If the machine reaches this state, it is considered to have accepted the input.

**5. Reject state (qR):** Similarly, there may be a reject state that, when reached, indicates that the Turing machine has rejected the input. If the machine reaches this state, it is considered to have rejected the input.

**6. Transition function (δ):** The transition function describes how the Turing machine transitions from one state to another based on the current state and the symbol read from the tape. It can be represented as a set of rules or a transition diagram. For example, a transition rule might be represented as δ(q0, 0) = (q1, 1, R), indicating that if the machine is in state q0 and reads a 0 on the tape, it transitions to state q1, writes a 1 to the tape, and moves the head to the right.

**7. Configuration:** A configuration of a Turing machine consists of the current state, the contents of the tape, and the position of the head on the tape. It provides a snapshot of the machine's current state during its computation.

**8. Computation:** The computation of a Turing machine refers to the sequence of configurations that the machine goes through when processing an input. This sequence represents the step-by-step evolution of the machine's state and tape contents as it processes the input.

# **APPLICATIONS**

Turing machines are used to study the properties and limitations of computation, such as what problems can be solved, how efficiently they can be solved, and what resources are required to solve them. Some of the applications of Turing machines are:

**Artificial intelligence**: The Turing test, which gauges a machine's capacity to mimic human speech, can be used to study the behavior and intelligence of machines and agents.

**Cryptography**: By applying the idea of Turing machines with infinite computing capacity to crack encryption algorithms, one can examine the security and complexity of cryptographic systems and protocols.

**Algorithmic complexity theory**: By applying ideas like Turing-completeness, decidability, and NP-completeness, Turing machines can be used to categorize issues and algorithms according to their level of difficulty and resource requirements.

**Compiler Design**: By utilizing the concept of universal Turing machines, which can emulate any other Turing computer, compilers that convert high-level programming languages into executable code can be designed and implemented using Turing machines.

**Computation theory**: By applying the Church-Turing thesis, which argues that a Turing machine is capable of performing any efficient computation technique, Turing machines can be utilized to investigate the underlying ideas and concepts of computation.

# **ADVANTAGES OF TURING MACHINES**

1. **Universal Computation**: Despite its apparent simplicity, a Turing machine has the remarkable ability to simulate any computer algorithm. It can handle any computation that can be expressed algorithmically.
2. **Decidability of Formal Languages**: Turing machines play a pivotal role in the study of formal languages. They can determine whether a given string belongs to a specific language (for instance, verifying the correctness of a program’s syntax).
3. **Solving Mathematical Functions**: By manipulating symbols on an infinite tape according to a set of rules, Turing machines can compute mathematical functions. They evaluate functions and perform calculations.
4. **Exploring Computational Limits**: The Turing machine serves as an abstract framework for exploring the boundaries of computability. It allows us to study algorithmic properties and assess the solvability of problems.
5. **Modeling Computation**: Offering a straightforward yet powerful model of computation, the Turing machine enables simulation of algorithm behavior and analysis of computational complexity.
6. **Theoretical Foundations**: In computer science, Turing machine**s** provide insights into the fundamental limits of computation and aid in analyzing the complexity of computational problems.
7. **Turing Completeness**: These machines are crucial in studying Turing completen**ess**, which refers to a computational model’s ability to simulate a **Turing machine**.
8. **Impact on Computer Science**: Turing machine**s** have significantly influenced the evolution of computer science, serving as a cornerstone concept in the theory of computation.

# **DESIGN**

**DESCRPTION**

The Turing Machine will be represented as class, with attributes and methods for the tape, the position, the state, the transition states.

**Class**

TuringMachine class: A class that represents the abstract model of a Turing machine. The class should have fields for the tape, the head position, the current state and transition states

**Enumerator class**

This is a special class that represents a group of named constants, which are the possible states of the Turing machine: **A, B, E, F**. The machine starts in state **A** and halts in state **F**.

**Fields**

* A variable called **currentState** that stores the current state of the machine.
* An ArrayList called **tape** that represents the tape of the machine. The tape is initially empty and can grow dynamically as the machine writes symbols on it. The **tape** can hold symbols **‘1’, ‘#’**, or **blank.**
* A variable called **headPosition** that stores the position of the tape head. The head can move left or right on the tape, and can read, write, or erase symbols on the tape.

**Methods**

**processSymbol():** The processSymbol method is a private method that simulates the behaviour of the Turing machine on a given symbol. It takes a char parameter called symbol and performs one of the following actions based on the current state and the symbol read:

* Replace the symbol with another symbol on the tape
* Move the head left or right on the tape
* Change the state to another state
* Halt the machine and output the tape contents

The method also prints the current state, the tape head position, and the tape contents after each action.

**processInput()**: The processInput method is a public method that takes a String parameter called input and performs the following steps:

* For each character in the input string, add it to the tape and call the processSymbol method with that character as an argument.
* Add a ‘#’ symbol to the end of the tape to mark the end of the input and call the processSymbol method with ‘#’ as an argument.
* Count the number of '1’s on the tape and print the final tape contents and the count.

The processInput method is used to simulate the input and output of the Turing machine on a given string. It also calls the processSymbol method to execute the state transitions of the machine.

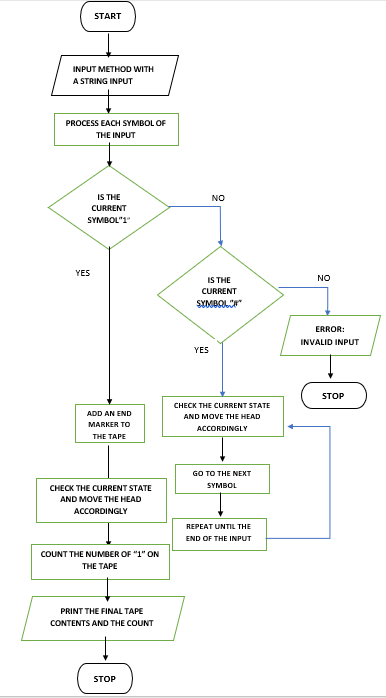
**Main Method()**: In the main method, an instance of the TuringMachine class will be created with a given input string and calls the processInput method to start the computation.

ALGORITHM

Start

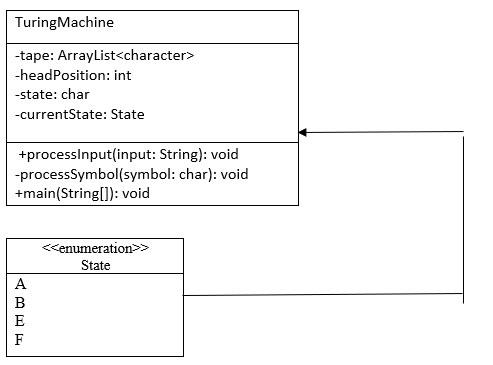
* Input: A string of symbols ‘1’ and ‘#’ with ‘#’ as the process symbol of the addition operation
* Output: The tape and The number of '1’s on the tape after the machine halts
* Steps:
  + Initialize the tape, the head position, and the current state
  + For each symbol in the input string, add it to the tape and process it according to the state transition table
  + Add a ‘#’ symbol to the end of the tape and process it according to the state transition table
  + Count the number of '1’s on the tape and print the final tape contents and the count

End

**FLOWCHART:** The diagrammactic representation of the entire process**. **

**CLASS DIAGRAM:**

This is a static structure diagram that shows the attributes and operations(or methods) of the class.



# **IMPLEMENTATION**

**import** java.util.ArrayList;

**public** **class** TuringMachine {

**private** **enum** State {

***A***, ***B***, ***E***, ***F***,

}

**private** State currentState = State.***A***;

**private** ArrayList<Character> tape = **new** ArrayList<>();

**private** **int** headPosition = 0;

**public** **void** processInput(String input) {

**for** (**char** symbol : input.toCharArray()) {

tape.add(symbol);

processSymbol(symbol);

}

tape.add('#'); // Add end marker

processSymbol('#');

// Count the number of '1's on the tape

**int** countOnes = 0;

**for** (Character symbol : tape) {

**if** (symbol == '1') {

countOnes++;

}

}

System.***out***.println("Final tape contents (sum): " + tape);

System.***out***.println("Final number of '1's on the tape: " + countOnes);

}

**private** **void** processSymbol(**char** symbol) {

**switch** (currentState) {

**case** ***A***:

**if** (symbol == '1') {

tape.set(headPosition, '#');

headPosition++;

currentState = State.***B***;

} **else** **if** (symbol == '#') {

headPosition--;

currentState = State.***F***;

}

**break**;

**case** ***B***:

**if** (symbol == '1') {

headPosition++;

} **else** **if** (symbol == '#') {

tape.set(headPosition, '1'); // Add '1' to the sum

headPosition--;

currentState = State.***E***;

}

**break**;

**case** ***E***:

**if** (symbol == '1') {

headPosition--;

} **else** **if** (symbol == '#') {

headPosition++;

currentState = State.***A***; // Go back to the beginning

}

**break**;

**case** ***F***:

// Halt and output tape contents

**break**;

}

// Print the current process

System.***out***.println("Current state: " + currentState + ", Tape head position: " + headPosition + ", Tape contents: " + tape);

}

**public** **static** **void** main(String[] args) {

TuringMachine TM = **new** TuringMachine();

TM.processInput("111#111111"); // Example input

}

}

**Output:**

Current state: B, Tape head position: 1, Tape contents: [#]

Current state: B, Tape head position: 2, Tape contents: [#, 1]

Current state: B, Tape head position: 3, Tape contents: [#, 1, 1]

Current state: E, Tape head position: 2, Tape contents: [#, 1, 1, 1]

Current state: E, Tape head position: 1, Tape contents: [#, 1, 1, 1, 1]

Current state: E, Tape head position: 0, Tape contents: [#, 1, 1, 1, 1, 1]

Current state: E, Tape head position: -1, Tape contents: [#, 1, 1, 1, 1, 1, 1]

Current state: E, Tape head position: -2, Tape contents: [#, 1, 1, 1, 1, 1, 1, 1]

Current state: E, Tape head position: -3, Tape contents: [#, 1, 1, 1, 1, 1, 1, 1, 1]

Current state: E, Tape head position: -4, Tape contents: [#, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Current state: A, Tape head position: -3, Tape contents: [#, 1, 1, 1, 1, 1, 1, 1, 1, 1, #]

Final tape contents (sum): [#, 1, 1, 1, 1, 1, 1, 1, 1, 1, #]

Final number of '1's on the tape: 9

# **RESULT ANALYSIS**

ASSUMPTIONS

1. Input Format: The input format is assumed to consist of a series of "1's" seperated by a "#." If the input differs from this particular format, the machine might not function properly.
2. Tape Size: It is assumed that the tape has enough capacity to store both the computation and the input. Though this cannot be implemented in a computer program, a Turing machine actually possesses an infinite tape.
3. State Transitions: For states A, B, E, and F, the state transitions are hardcoded. This indicates that the machine is limited to a certain type of calculation. It would be necessary to alter the state transitions in order to carry out a different computation.

LIMITATIONS

1. Limited States: There are just a few states (A, B, E, and F) that the machine can be in. Many more states could exist in a Turing machine with more complexity.
2. No Error Handling: The code does not contain any error handling. The machine may crash or behave strangely if the input is not in the expected format or if an unexpected character appears on the tape.

CHALLENGES

1. Simulating an Infinite Tape: One of the main challenges in implementing a Turing Machine is simulating the infinite tape. This approach simulates the tape using an ArrayList, however it can only offer a limited amount of space.
2. Determining When to Halt: Deciding when to stop the machine is another difficulty. In this implementation, upon reaching state F, the machine breaks. But it can be harder to know when to stop a more complicated equipment.
3. Debugging: Tracing the state transitions and head movement on the tape is a difficult part of debugging a Turing machine. The **System.out.println** statements in the code help with this, but it can still be difficult to understand what is going wrong if the machine does not behave as expected.

# **CONCLUSION**

The project's aim is to bring into practice a Turing machine, an abstract computational model that can modify symbols on a tape by following predetermined guidelines.

The TuringMachine class, which has fields and methods for the tape, head position, current state, and transition stages, is defined in the Java programming language project.

Additionally, the project defines an Enumerator class that represents the A, B, E, and F potential states of the Turing machine. The machine enters state A at startup and stops in state F.

The two methods that the project implements are processSymbol and processInput, which mimic the actions of a Turing computer when given an input string or a symbol, respectively.

The project counts the number of "1s" on the tape as the output and utilizes the symbol "#" to indicate the end of the input and the addition operation.

The project prints the final tape contents and the count after testing the Turing machine with a sample input string.

Some possible extensions or improvements for the future work are:

* Implementing various Turing machine types—such as multi-tape, non-deterministic, and universal Turing machines—to use and contrasting their strengths and weaknesses.
* Adding a graphical user interface (GUI) that enables the user to observe the contents of the tape, enter the input string, and manage the Turing machine's step-by-step execution.
* Using various input strings, testing the Turing machine's accuracy and performance while looking for mistakes or exceptions.
* Exploring the applications and implications of the Turing machine in various fields of computer science, such as computability, complexity, cryptography, artificial intelligence, etc.

# **BIBLIOGRAPHY**

Erik Gregersen Dec 28, 2023, Turing machine | Definition & Facts | Britannica

Hodges, A., 1983, Alan Turing: the Enigma, London: Burnett; New York:

Simon & Schuster; London: Vintage, 1992, 2012; Princeton University

Press, 2012.

Petzold, Charles, 2008, The Annotated Turing: A Guided Tour Through Alan

Turing’s Historic Paper on Computability and Turing Machines,

Indianapolis, IN: Wiley.

Robert Sedgewick and Kevin Wayne. June 30, 2017

[Turing Machines (princeton.edu)](https://introcs.cs.princeton.edu/java/52turing/)

Sipser, Michael, 1996, Introduction to the Theory of Computation, Boston:

PWS Publishing.

The Entscheidungsproblem and Alan Turing:

[Entscheidungsproblem - Simple English Wikipedia, the free encyclopedia](https://simple.wikipedia.org/wiki/Entscheidungsproblem)

Turing Machines. Brilliant.org. Retrieved 00:17, February 13,

2024, from <https://brilliant.org/wiki/turing-machines/>

Turing Machines (Stanford Encyclopedia of Philosophy)

OTHER INTERNET SOURCES:

[Example of Turing Machine - Javatpoint](https://www.javatpoint.com/examples-of-turing-machine)

[Turing Machine in TOC - GeeksforGeeks](https://www.geeksforgeeks.org/turing-machine-in-toc/)