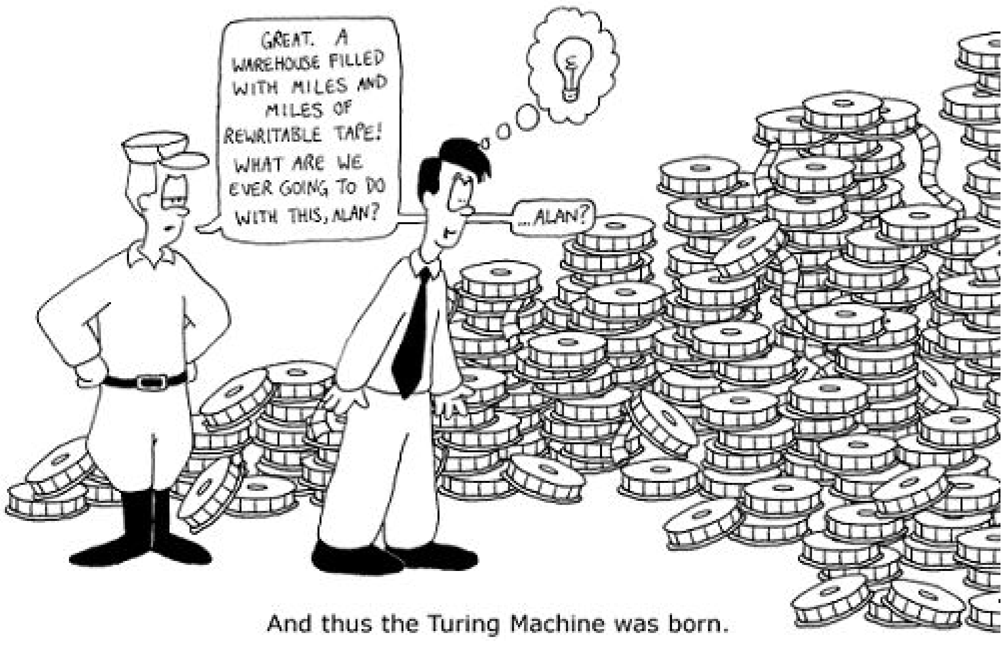
TURING MACHINE IMPLEMENTATION

Sebastián García Acosta

Table of contents:

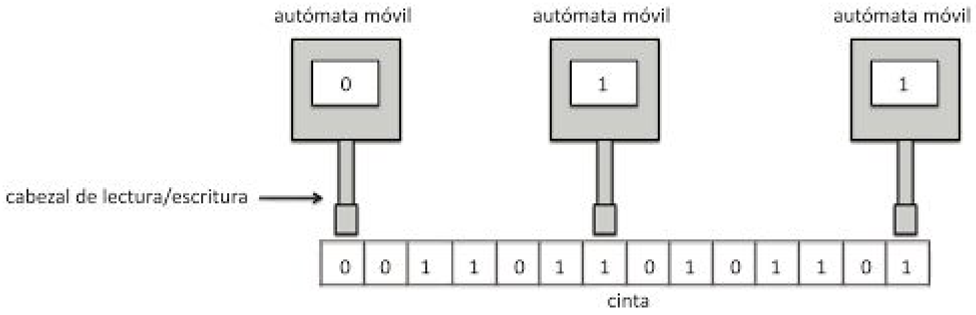
Statement


GREAT. A WAREHOUSE FILLED WITH MILES AND MILES OF REWRITABLE TAPE! WHAT ARE WE EVER GOING TO DO WITH THIS, ALAN?
...ALAN?
And thus the Turing Machine was born.

A Turing machine is a device that manipulates symbols on a strip of tape according to a table of rules. Despite its simplicity, a Turing machine can be adapted to simulate the logic of any computer algorithm and is particularly useful in explaining the functions of a computer's central processing unit.

Computer scientist Srdna Labázitsira has just proposed a finite resizable but not limited size Turing machine with three heads. It is very important for the continuity of your investigation to test its operation and for this reason it has asked you that given a series of instructions that will be defined below, you simulate the machine and execute each one of them.

Note that the positions in the list are 0-index, that is, the indexes start at 0 and therefore the first position is 0, the second position is 1, and so on.



The machine has 3 reading heads: C0, C1 and C2, which are at all times at the beginning of the tape (C0), that is to say on the first element of the tape, in the middle of the tape (C1), that is , on the element in the middle of the tape (if there are two elements in the middle, then the head is on the first of the two), and at the end of the tape (C2), that is, on the last element.

The heads can carry out three operations:

Read the item they are on.
Add an item. Which in each case implies leaving the new element in the same position in which each head will be. If it is inserted by C0, the new element will be first, if it is inserted by C1, it will be in the middle and if it is inserted by C2 it will be last.
Delete an item. The element on which the head is removing the value is removed.

The three operations can be done at any time. The tape of the machine will initially be empty and therefore, the three reading heads will be in the same place, on no element.

If an item is read over a playhead that is over no item, the read value will be #. If an element is deleted on a playhead that is on any element, nothing will happen, that is, there will be no modification on the tape.

Input

There will be one line for each test case. The line of a test case is made up of a series of operations on a three reading head turing machine that starts with an empty tape. Every operation is made up of at least two elements, first the reading head (0 indicates the reading head C0, 1 for C1 and 2 for C2) and then the operation to be performed (0 indicates reading, 1 to add and 2 to remove). If the operation is to add an item then you will additionally have the item to be added to the ribbon. Neither the operations nor the elements within the operations are separated by any character. The elements to add will always be capital letters of the English alphabet (from A to Z -26 letters-). Each test case will have at most 2147486 characters. The input file can weigh about 20MB.

Output

There will be one line for each letter read in each test case present in the entry. For each test case, the elements read by the reading heads will be printed in the order in which these operations were carried out.

Example

Input	**Output**
01A1021N11L1000122002 01P21P11I01G 01T1021U11R2021K1011N01I21G001020	A L A N T U R I N G

You can see the previous examples are detailed step by step graphically.
You should test your solution with the big test case. You should take the time it takes for your solution in milliseconds, from the start of the program, before any other instructions, until all other instructions have been executed.

Reading the data should be done using text files (BufferedReader suggested) and writing the output also in text files (BufferedWriter suggested). You can use the program to compare files found in the same directory as the test files.

The time it takes for your algorithm should be the only value printed in console. The time it takes for any input (example or large) should not exceed 1 second.

The delivered solution will be rated with the third laboratory rubric for the Turing Machine problem.

Functional requirements:

The turing machine must be able to:

1. Add a new character at the respective position to which each of the three heads points maintaining their position, i.e., first head C0 will always point to first node, the C1 head will always point to the node at position 2/n, and C2 will always point to the last node.
2. Read the character value to which each head points.
3. Delete the character value to which each head points.

Test cases design

Scenarios configuration

Name	**Class**	**Scenario**
emptySetup	TuringMachineTest	An initialized object of class TuringMachine.
oneNodeSetup	TuringMachineTest	An object of class TuringMachine with one node.
oddNodesSetup	TuringMachineTest	An object of class TuringMachine whose doubly linked list has has n number of nodes added such that n % 2 != 0.
evenNodesSetup	TuringMachineTest	An object of class TuringMachine whose doubly linked list has n number of nodes added such that n % 2 == 0.
emptyListSetup	DoublyLinkedListTest	An object of class DoublyLinkedList initialized.
nonEmptyListSetup	DoublyLinkedListTest	An object of class DoublyLinkedList that contains <-A-><-B-><-C->D-> nodes.

Test cases:

Input reading:

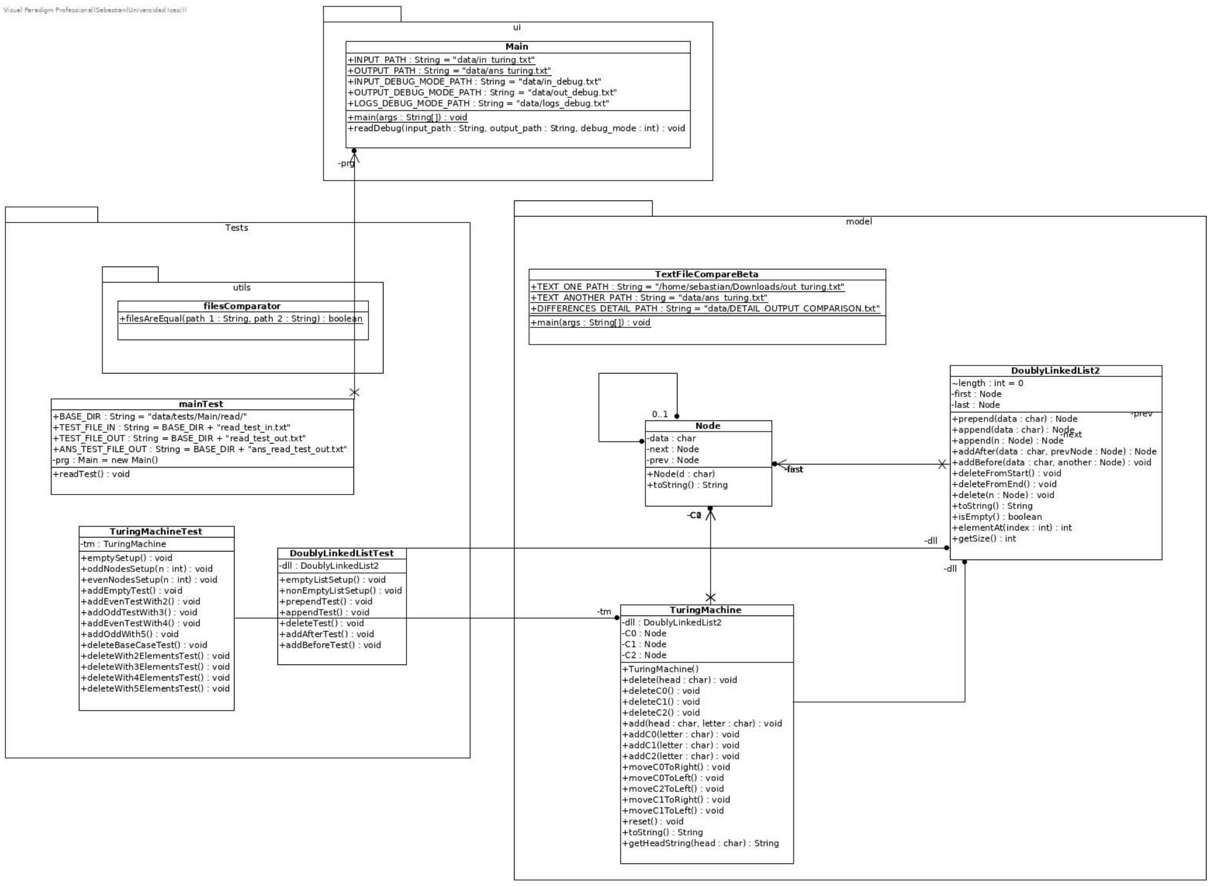
Test objective: Check that the TuringMachine is reading properly the operations.				
Class	**Method**	**Scenario**	**Inputs**	**Result**
Main	main	None		The operations identified by the program are equal to the expected, i.e.. If operation to test was: `01A1021N11L100012200` then, the expected output written in plain text is: “01A\n10\n21N\n11L\n10\n00\n12\n 20\n00”

List operations:

Test objective: Check that operations prepend, append, delete, addbefore and addafter works properly.				
Class	**Method**	**Scenario**	**Inputs**	**Result**
Doubly Linked List	prepend	nonEmptyListSetup	‘A’, ‘B’, ‘C’ and ‘D’	After prepending ‘A’, ‘B’, ‘C’ and ‘D’, dll.toString() is equal to:”<-D->-C->B->A->A->B->C->D-> ” and dll.length is equal to 8
Doubly Linked List	append	nonEmptyListSetup	‘A’, ‘B’, ‘C’ and ‘D’	After appending ‘A’, ‘B’, ‘C’ and ‘D’, dll.toString() is equal to:”<-A-><-B-><-C-><-D-><-A-><-B-><-C->D->” and dll.length is equal to 8
Doubly Linked List	delete	nonEmptyListSetup	‘B’, ‘A’, ‘D’	After deleting character values of ‘B, ‘A’ and ‘D’, dll.toString() is equal to “<-C->”
Doubly Linked List	addAfter	nonEmptyListSetup	‘F’,nodeE	After appending Node(‘E’) by reference and ‘G’ by value, dll.toString() is equal to ”<-A-><-B-><-C-><-D-><-E-><-F-><-G->”
Doubly Linked List	addBefore	nonEmptyListSetup	nodeG,F	After appending (by value) ‘E’ and ‘G’ by reference, dll.toString() is equal to ”<-A-><-B-><-C-><-D-><-E-><-F-><-G->”

Turing machine operations:

Test objective: Check that operations add, delete and read works properly and the functional and non-functional requirements holds true.				
Class	**Method**	**Scenario**	**Inputs**	**Result**
TuringMachine	add	emptySetup	None	After adding ‘A’, the first node is equal to the node pointed by all the heads (C0,C1, and C2)
DoublyLinkedList	add	nonEmptyList Setup	‘A’, ‘B’, ‘C’ and ‘D’	Each nodes is added properly, holding true the functional requirements. The add process is done n times with n number of nodes and randomly chosen operations. Everytime is checked that functional requirements holds true.
DoublyLinkedList	delete	nonEmptyList Setup	‘B’, ‘A’, ‘D’	Each nodes is added properly, holding true the functional requirements.The delete process is done n times with n number of nodes
DoublyLinkedList	addAfter	nonEmptyList Setup	nodeE,F	Each nodes is added properly after the specified Node, holding true the functional requirements
DoublyLinkedList	addBefore	nonEmptyList Setup	nodeG,F	Each nodes is added properly before the specified Node, holding true the functional requirements


ui
Main
+INPUT_PATH : String = "data/in_turing.txt"
+OUTPUT_PATH : String = "data/ans_turing.txt"
+INPUT_DEBUG_MODE_PATH : String = "data/in_debug.txt"
+OUTPUT_DEBUG_MODE_PATH : String = "data/out_debug.txt"
+LOGS_DEBUG_MODE_PATH : String = "data/logs_debug.txt"
+main(args : String[]) : void
+readDebug(input_path : String, output_path : String, debug_mode : int) : void
-prg
Tests
utils
filesComparator
+filesAreEqual(path_1 : String, path_2 : String) : boolean
mainTest
+BASE_DIR : String = "data/tests/Main/read/"
+TEST_FILE_IN : String = BASE_DIR + "read_test_in.txt"
+TEST_FILE_OUT : String = BASE_DIR + "read_test_out.txt"
+ANS_TEST_FILE_OUT : String = BASE_DIR + "ans_read_test_out.txt"
-prg : Main = new Main()
+readTest() : void
TuringMachineTest
-tm : TuringMachine
+emptySetup() : void
+oddNodesSetup(n : int) : void
+evenNodesSetup(n : int) : void
+addEmptyTest() : void
+addEvenTestWith2() : void
+addOddTestWith3() : void
+addEvenTestWith4() : void
+addOddWith5() : void
+deleteBaseCaseTest() : void
+deleteWith2ElementsTest() : void
+deleteWith3ElementsTest() : void
+deleteWith4ElementsTest() : void
+deleteWith5ElementsTest() : void
DoublyLinkedListTest
-dll : DoublyLinkedList2
+emptyListSetup() : void
+nonEmptyListSetup() : void
+prependTest() : void
+appendTest() : void
+deleteTest() : void
+addAfterTest() : void
+addBeforeTest() : void
model
TextFileCompareBeta
+TEXT_ONE_PATH : String = "/home/sebastian/Downloads/out_turing.txt"
+TEXT_ANOTHER_PATH : String = "data/ans_turing.txt"
+DIFFERENCES_DETAIL_PATH : String = "data/DETAIL_OUTPUT_COMPARISON.txt"
+main(args : String[]) : void
0..1
Node
-data : char
-next : Node
-prev : Node
+Node(d : char)
+toString() : String
-first
-C2
DoublyLinkedList2
~length : int = 0
-first : Node
-last : Node
+prepend(data : char) : Node
+append(data : char) : Node
+append(n : Node) : Node
+addAfter(data : char, prevNode : Node) : Node
+addBefore(data : char, another : Node) : void
+deleteFromStart() : void
+deleteFromEnd() : void
+delete(n : Node) : void
+toString() : String
+isEmpty() : boolean
+elementAt(index : int) : int
+getSize() : int
-prev
-next
-dll
-dll
-tm
TuringMachine
-dll : DoublyLinkedList2
-C0 : Node
-C1 : Node
-C2 : Node
+TuringMachine()
+delete(head : char) : void
+deleteC0() : void
+deleteC1() : void
+deleteC2() : void
+add(head : char, letter : char) : void
+addC0(letter : char) : void
+addC1(letter : char) : void
+addC2(letter : char) : void
+moveC0ToRight() : void
+moveC0ToLeft() : void
+moveC2ToLeft() : void
+moveC1ToRight() : void
+moveC1ToLeft() : void
+reset() : void
+toString() : String
+getHeadString(head : char) : String