Riley Campbell

CSC-445
Exam 3

Index

i. How to compile the code

No compiling is needed, this program may be run from the command line so long as Main.py, ComplexTM.py, StringAcceptorTM.py, and TransducerTM.py are all in the same directory.

ii. How to run program

This program may be from the command line with '$python3 Main.py'. All other inputs will be prompted to the user from this point on.

iii. Program description

Once the program has started, a menu will display asking the user to select which category of Turing machine you wish to use. The choices are Language Acceptor Machines, Transducer Machines, and Complex Machines, or the option to quit the program. From each of the machine choices, a submenu will be displayed for each of the machines for selecting which functionality of the machine you wish to utilize. The program will be run in a loop where each main menu option will open a submenu and run a Turing Machine process before returning to the main menu.

iv. Complete function descriptions

def leave():

this method displays the exiting program message and returns the signal to exit
:return: 'q'

def complicatedTM():

this method handles the choice of a complicated Turing machine operation. it constructs an object of the Complex class and calls the class menu. if a '1' is returned and the input for unary multiplication is valid, then the unary multiplication machine is run. if a '2' is returned and the input for unary division is valid, then the machine for unary division is run. then this method returns 'y'
:return: 'y'

def transducerTM():

This method handles the choice of a transducer Turing machine operation. it constructs an object of the Transducer class and calls the class menu. if a '1' is returned and the input for unary addition is valid, then the machine for binary addition is run. then this method returns 'y'

:return: 'y'

def stringAcceptorTM():
This method handles the choice of a String Acceptor Turing machine operation. it constructs an object of the StringAcceptor class and calls the class menu. if a '1' is returned then the a^nb^nc^n machine is run. if a '2' is returned and the input for ww^R is valid, then the machine for ww^R is run. then this method returns 'y'
:return: 'y'

def menu():
This method displays the menu for the program. selecting option 1 will select the Language Acceptor Machine, option 2 will select Transducer Machine, option 3 will select the Complex Machine, and option 4 will end the program.
:return: the selected menu option

def main():
this is the main program method. it builds the dictionary of function calls for this file and runs the program loop.
:return:

class StringAcceptor:
this is the class for the String Acceptor. It has the ability to determine if a string is in the form of symbol1^n + symbol2^n + symbol3^n and if a string is in the form of ww^R, an even palindrome. the methods include a menu, setting up symbol1^n + symbol2^n + symbol3^n, setting up ww^R, running the tape, the constructor, the states for symbol1^n + symbol2^n + symbol3^n, and the states for ww^R.

def __init__(self):
this is the constructor for StringAcceptor. it gives the class a blank, an empty tape, a final state, a start state, the input string, a dead state, a placeholder for symbol 1, a placeholder for symbol 2, a placeholder for symbol 3, a visitedX symbol, a visitedY symbol, a visitedZ symbol, an index pointer, a dictionary handler for the state function calls, and a target symbol placeholder

def menu(self):
this method displays the menu for the StringAcceptor class. selecting option 1 will select, a^nb^nc^n and where a, b, and c are any three user input values. option 2 will select ww^R and where the string (ww^R) is even. option 3 will go back to the main menu.
:return: the selected menu option

def setInput1(self):
this method sets the input for the a^nb^nc^n language. it takes in the input string for the machine to test, sets the final state to 'q5a', the dead state to 'q6a', and the start state to 'q0a'. next it begins making the first part of the tape. it checks that the input string has at least 1 symbol and if so, saves that to the symbol1 holder then transfers all the matching symbols to the tape till a symbol is found that doesn’t match or the end of the input string is found. then this same process is done to store symbol2 and symbol3 and transfer all their matching symbols to the tape. next it adds a blank to the tape. after this it sets up the handler dictionary of keys q0a to q4a with values for function calls of state_q0a to state_q4a.
:return:

def runSelection1(self):

this method runs the a^n+b^n+c^n. it sets the index to -1, the direction to 'R', and the handler to the initial state method. then it begins the Turing machine run by calling the state functions and checking the states that are returned. if the Dead State is returned, display the message that the input resulted in a dead state and the machine stops. if a final state is returned, display the message that the Final State was reached and the language was accepted, and the machine ends. otherwise, the new state is traversed to.

:return:

def state_q0a(self, direction):

this method checks the input direction and if it is a 'R', move the index right and checks the tape at the current index. if it is a symbol1, set the tape at the index to a visitedX then return 'q1a' and 'R'. if the tape at index is a visitedY, set the tape at index to visitedY, then return 'q2a' and 'R'. otherwise return the Dead State and None

:param direction: the direction to move the index

:return: the state to go to next, the direction to travel

def state_q1a(self, direction):

this method checks the input direction and if it is a 'R', move the index right and checks the tape at the current index. if it is a symbol1, set the tape at the index to a symbol1 then return 'q1a' and 'R'. if the tape at index is a visitedY, set the tape at index to visitedY, then return 'q1a' and 'R'. if the tape at index is a symbol2, set the tape at index to visitedY, and return 'q2a' and 'R'. otherwise return the Dead State and None

:param direction: the direction to move the index

:return: the state to go to next, the direction to travel

def state_q2a(self, direction):

this method checks the input direction and if it is a 'R', move the index right then it checks the tape at the current index. if it is a symbol2, set the tape at the index to a symbol2 then return 'q2a' and 'R'. if the tape at index is a visitedZ, set the tape at index to visitedZ, then return 'q2a' and 'R'. if the tape at index is a symbol3, set the tape at index to visitedZ, and return 'q3a' and 'L'. otherwise return the Dead State and None

:param direction: the direction to move the index

:return: the state to go to next, the direction to travel

def state_q3a(self, direction):

this method checks the input direction and if it is a 'L', move the index left then it checks the tape at the current index. if it is a symbol1, set the tape at the index to a symbol1 then return 'q3a' and 'L'. if the tape at index is a symbol2, set the tape at index to symbol2, then return 'q3a' and 'L'. if the tape at index is a visitedY, set the tape at index to visitedY, and return 'q3a' and 'L'. if the tape at index is a visitedZ, set the tape at index to visitedZ, and return 'q3a' and 'L'. if the tape at index is a visitedX, set the tape at index to visitedX, and return 'q0a' and 'R'. otherwise return the Dead State and None

:param direction: the direction to move the index

:return: the state to go to next, the direction to travel

def state_q4a(self, direction):

this method checks the input direction and if it is a 'R', move the index right then it sets the calculating bool to True and checks the tape at the current index. if it is visitedY, set the tape at the index to visitedY then return 'q4a' and 'R'. if the tape at index is a visitedZ, set the tape at index to visitedZ, then return

'q4a' and 'R'. if the tape at index is a blank, return 'q5a' and None. otherwise return the Dead State and None
:param direction: the direction to move the index
:return: the state to go to next, the direction to travel

def setInput2(self):
this method sets the input for the wwR language. it takes in the input string for the machine to test, if the length of the string is zero, display an error message and return False. then set the target symbol to the first symbol in the input string, push the input string into the tape, and append on a blank symbol. then it sets the final state to 'q5b', the dead state to 'q4b', and the start state to 'q0b'. after this it sets up the handler dictionary of keys q0b to q3b with values for function calls of state_q0b to state_q3b. then return True
:return: true or false

def runSelection2(self):
this method runs the wwR machine. it sets the index to -1, the direction to 'R', and the handler to the initial state method. then it begins the Turing machine run by calling the state functions and checking the states that are returned. if the Dead State is returned, display the message that the input resulted in a dead state and the machine stops. if a final state is returned, display the message that the Final State was reached and the language was accepted, and the machine ends. otherwise, the new state is traversed to.
:return:

def state_q0b(self, direction):
this method checks the input direction and if it is a 'R', move the index right, otherwise move the index left. then it checks the tape at the current index. if it is blank, set the tape at the index to blank then return the Final State and None. otherwise, set the target symbol to tape at the index, set the tape at the index a blank and return 'q1b and 'R'.
:param direction: the direction to move the index
:return: the state to go to next, the direction to travel

def state_q1b(self, direction):
this method checks the input direction and if it is a 'R', move the index right, then it checks the tape at the current index. if it is blank, set the tape at the index to blank then return 'q2b' and 'L'. otherwise, set the tape at the index to itself and return 'q1b and 'R'.
:param direction: the direction to move the index
:return: the state to go to next, the direction to travel

def state_q2b(self, direction):
this method checks the input direction and if it is a 'L', move the index left, then it checks the tape at the current index. if it is the target symbol, set the tape at the index to blank then return the 'q3b' and 'L'.
otherwise, return the Dead State and 'L'.
:param direction: the direction to move the index
:return: the state to go to next, the direction to travel

def state_q3b(self, direction):
this method checks the input direction and if it is a 'L', move the index left, then it checks the tape at the current index. if it is blank, set the tape at the index to blank then return the 'q0b' and 'R'.
otherwise, return 'q3b' and 'L'.
:param direction: the direction to move the index

:return: the state to go to next, the direction to travel

class Transducer:

this is the class for handling addition of unary and binary numbers. this includes a constructor, a method to check inputs for errors, methods to set the input state of the machines, a method to run the machines, and methods for all the states

def __init__(self):

this method constructs the class and gives it a blank, 2 tapes with a blank in them, an answer queue, the startState, the finalState, a dictionary for the handles of function calls, and the index point

def menu(self):

this method displays the menu for the Transducer class. selecting option 1 will select unary addition, option 2 will select binary addition, and option 3 will go back to the main menu.

:return: the selected menu option

def checkInput(self, input1, input2, language):

this method takes in two input strings and checks them against the language for the inputs. if anything in the inputs is not found in the language, or if either string is empty, the method returns False. otherwise, it returns True

:param input1: user input 1

:param input2: user input 2

:param language: the language

:return: true or false

def setInput2(self):

this method sets up the machine for binary addition. it asks the user for the two numbers to be added together. it then checks those inputs with checkInput() and if that returns False, the method returns False. next it pads the input strings with '0's on the left till they are of equal length. then it puts input 1 in tape1 with a leading '0', then it puts input 2 in tape 2 with a leading '0'. then it sets the index to the length of the tape1, the start state to 'q0b, the final states to 'q2b', and sets the dictionary handler of keys q0b to q2b with values for function calls of state_q0b to state_q2b. lastly the method returns True

:return: True or False

def setInput1(self):

this method sets up the machine for unary addition. it asks the user for the two numbers to be added together. it then checks those inputs with checkInput() and if that returns False, the method returns False. next it puts the input 1 in the tape, then it puts a 'c' in the tape for a separator, then it puts input 2 in the tape, then it puts the blanks from the infinity list in the tape. after the tape is set up, it sets the index to 1, the start state to 'q0a, the final states to 'q5a', and sets the dictionary handler of keys q0a to q5a with values for function calls of state_q0a to state_q5a. lastly the method returns True

:return: True or False

def run(self):

this method runs the process for unary and binary addition. it sets a handler to the start state function and then begins running the machine by calling functions from the handlers dictionary with the keys returned from the state functions. this loop will end when a final state is reached. once that happens, the result from the run will be displayed from whichever final state is reached.

:return:

def state_q0b(self):

this method moves the index left and combines the values at the index from tape1 and tape2. if there is a blank in the value, return the final state. if the value is '11', insert '0' in the answer queue, and return 'q1b'. if the value is '01' or '10', insert '1' in the answer queue, and return 'q0b'. if the value is '00', insert '0' in the answer queue, and return 'q0b'

:return: the state which to go to next

def state_q1b(self):

this method moves the index left and combines the values at the index from tape1 and tape2. if there is a blank in the value, return the final state. if the value is '00', insert '1' in the answer queue, and return 'q0b'. if the value is '01' or '10', insert '0' in the answer queue, and return 'q1b'. if the value is '11', insert '1' in the answer queue, and return 'q1b'

:return: the state which to go to next

def state_q2b(self):

this method is the final state for the binary calculator. it strips all of the leading '0's from the answer and then converts the queue to a string to return for printing

:return: the answer string

def state_q0a(self):

this method checks the tape at the current index. if it is a '0', set the tape at the index to 'X', move the index right, then returns 'q1a'. if the tape at index is 'c', set the tape at index to blank, move the index right, then return 'q5a'.

:return: the state which to go to next

def state_q1a(self):

this method checks the tape at the current index. if it is a '0', set the tape at the index to '0', move the index right, then returns 'q1a'. if the tape at index is 'c', set the tape at index to 'c', move the index right, then return 'q2a'.

:return: the state which to go to next

def state_q2a(self):

this method checks the tape at the current index. if it is a '0', set the tape at the index to '0', move the index right, then returns 'q2a'. if the tape at index is blank, set the tape at index to '0', move the index left, then return 'q3a'.

:return: the state which to go to next

def state_q3a(self):

this method checks the tape at the current index. if it is a '0', set the tape at the index to '0', move the index left, then returns 'q3a'. if the tape at index is 'c', set the tape at index to 'c', move the index left, then return 'q4a'.

:return: the state which to go to next

def state_q4a(self):

this method checks the tape at the current index. if it is a '0', set the tape at the index to '0', move the index left, then returns 'q4a'. if the tape at index is 'X', set the tape at index to 'X', move the index right, then return 'q0a'.

:return: the state which to go to next

def state_q5a(self):

this is the final state for the unary adder. if formats the output by stripping away all of the 'X's and blanks before returning the string
:return: the answer string

class Complex:

This is the class for handling unary multiplication and division. the methods include a menu, setting up multiplication, setting up division, running the tape, the constructor, the states for multiplication, and the states for division

def __init__(self):

This is the constructor for Complex. it sets blank to '□', tape to a list with 1 blank, infinity to a list with 8 blanks, the start state to '', final states to an empty list, the stack to an empty list, the handlers to an empty dictionary, the index of the tape to 0, the right direction to 1, and the left direction to -1.

def menu(self):

this method displays the menu for the Complex class. selecting option 1 will select unary multiplication, option 2 will select unary division, and option 3 will go back to the main menu.
:return: the selected menu option

def checkInput(self, input1, input2, language):

this method takes in two input strings and checks them against the language for the inputs. if anything in the inputs is not found in the language, the method returns False. otherwise, it returns True
:param input1: user input 1
:param input2: user input 2
:param language: the language
:return: true or false

def setInput1(self):

this method sets up the machine for multiplication. it asks the user for the two numbers to be multiplied together. it then checks those inputs with checkInput() to see if anything isn't a '0'. if so, it returns False. next it puts the input 1 in the tape, then it puts a 'c' in the tape for a separator, then it puts input 2 in the tape, then it puts a blank in the tape. after the tape is set up, it sets the index to 1, the start state to 'q0a, the final states to 'q12a', and sets the dictionary handler of keys q0a to q12a with values for function calls of state_q0a to state_q12a. lastly the method returns True
:return: True or False

def run(self):

this method runs the process for multiplication and division. it sets a handler to the start state function and then begins running the machine by calling functions from the handlers dictionary with the keys returned from the state functions. this loop will end when a final state is reached. once that happens, the result from the run will be displayed from whichever final state is reached.
:return:

def state_q0a(self):

this method checks the tape at the current index. if it is a '0', set the tape at the index to '0' move the index right, then return 'q0a'. if the tape at index is 'c', set the tape at index to 'c', move the tape right, then return 'q1a'
:return: the state which to go to next

def state_q1a(self):
this method checks the tape at the current index. if it is a '0', set the tape at the index to '0' and move the index right, then return 'q1a'. if the tape at index is a blank, set the tape at index to 'c', move the index left, then return 'q2a'
:return: the state which to go to next

def state_q2a(self):
this method checks the tape at the current index. if it is a '0', set the tape at the index to '0', move the index left, then returns 'q2a'. if the tape at index is 'c', set the tape at index to 'c', move the index right, then return 'q3a'
:return: the state which to go to next

def state_q3a(self):
this method checks the tape at the current index. if it is a 'X', set the tape at the index to 'X', move the index right, then returns 'q3a'. if the tape at index is 'c', set the tape at index to blank, move the index right, then return 'q12a'. if it is a '0', set the tape at index to 'X', move the index left, then return 'q4a
:return: the state which to go to next

def state_q4a(self):
this method checks the tape at the current index. if it is a 'X', set the tape at the index to 'X', move the index left, then returns 'q4a'. if the tape at index is 'c', set the tape at index to 'c', move the index left, then return 'q5a'
:return: the state which to go to next

def state_q5a(self):
this method checks the tape at the current index. if it is a 'Y', set the tape at the index to 'Y', move the index left, then returns 'q5a'. if the tape at index is a blank, set the tape at index to blank, move the index right, then return 'q11a'. if it is a '0', set the tape at index to 'Y', move the index right, then return 'q6a
:return: the state which to go to next

def state_q6a(self):
this method checks the tape at the current index. if it is a 'Y', set the tape at the index to 'Y', move the index right, then returns 'q6a'. if the tape at index is 'c', set the tape at index to 'c', move the index right, then return 'q7a'
:return: the state which to go to next

def state_q7a(self):
this method checks the tape at the current index. if it is a 'c', set the tape at the index to 'c', move the index right, then returns 'q8a'. if the tape at index is '0', set the tape at index to '0', move the index right, then return 'q7a'. if it is a 'X', set the tape at index to 'X', move the index right, then return 'q7a
:return: the state which to go to next

def state_q8a(self):
this method checks the tape at the current index. if it is a blank, set the tape at the index to '0', move the index left, then returns 'q9a'. if the tape at index is '0', set the tape at index to '0', move the index right, then return 'q8a'
:return: the state which to go to next

def state_q9a(self):

this method checks the tape at the current index. if it is a '0', set the tape at the index to '0', move the index left, then returns 'q9a'. if the tape at index is 'c', set the tape at index to 'c', move the index left, then return 'q10a'

:return: the state which to go to next

def state_q10a(self):

this method checks the tape at the current index. if it is a 'c', set the tape at the index to 'c', move the index left, then returns 'q5a'. if the tape at index is '0', set the tape at index to '0', move the index left, then return 'q10a'. if it is a 'X', set the tape at index to 'X', move the index left, then return 'q10a

:return: the state which to go to next

def state_q11a(self):

this method checks the tape at the current index. if it is a 'Y', set the tape at the index to '0', move the index right, then returns 'q11a'. if the tape at index is 'c', set the tape at index to 'c', move the index right, then return 'q3a'.

:return: the state which to go to next

def state_q12a(self):

this method removes the blank from the left side of the tape, then its searched for first index that a blank is found in starting from the left. once it fights that index, it strips everything from that index left. next a string is constructed from only the 0s on the tape encased in single quotes before returning it.

:return: the state which to go to next

def setInput2(self):

this method sets up the machine for division. it asks the user for the two numbers to be divided together. it then checks those inputs with checkInput() to see if anything isn't a '0'. if so, it returns False. next it checks that the divisor is equal to at least a one, if not, it returns False. next it puts the input 1 in the tape, then it puts a 'c' in the tape for a separator, then it puts input 2 in the tape, then it puts a 'c' in the tape for a separator, then it adds the infinity list to the tape. after the tape is set up, it pushes a 'X' onto the stack, sets the index to 1, the start state to 'qa, the final states to ''q8b', 'qd', 'qe'', and sets the dictionary handler of keys to 'qa' to 'qe' and 'q0b' to 'q08' with values for function calls of state_qa to state_qe and state_q0b to state_q08. lastly the method returns True

:return: True or False

def state_qa(self):

this method checks the tape at the current index. if it is a '0', set the tape at the index to '0', push a '0' onto the stack, move the index right, then return 'qa'. if the tape at index is 'c', set the tape at index to 'c', move the index right, then return 'qb'.

:return: the state which to go to next

def state_qb(self):

this method checks the tape at the current index. if it is a 'c', set the tape at the index to 'c', move the index right, then return 'qd'. if the tape at index is 'c' and the top of the stack is '0', set the tape at index to '0', move the index right, then return 'qc'.

:return: the state which to go to next

def state_qc(self):

this method checks the tape at the current index. if it is a 'c' and the top of the stack is not 'X, set the tape at the index to 'c', move the index left, then return 'qe'. if the tape at index is '0' and the top of the stack is not 'X', set the tape at index to '0', pop the value off the top of the stack, move the index right, then return 'qc'. otherwise set index to 1 and return 'q0b'

:return: the state which to go to next

def state_qe(self):

this method is one of the final states for the division. it builds the answer string initially with 'Q = , R = ' and then sets the index to 1. next start walking through the tape and appending all '0's to the answer string until a 'c' is found. then return the string.

:return: the string with the answer

def state_qd(self):

this method is one of the final states for the division. it builds the answer string initially with 'Q = , R = ' then return the string.

:return: the string with the answer

def state_q0b(self):

this method checks the tape at the current index. if it is a '0', set the tape at the index to blank, move the index right, then returns 'q1b'. if the tape at index is 'c', set the tape at index to 'c', move the index right, then return 'q4a'.

:return: the state which to go to next

def state_q1b(self):

this method checks the tape at the current index. if it is a '0', set the tape at the index to '0', move the index right, then returns 'q1b'. if the tape at index is 'c', set the tape at index to 'c', move the index right, then return 'q2a'.

:return: the state which to go to next

def state_q2b(self):

this method checks the tape at the current index. if it is a '0', set the tape at the index to 'x', move the index left, then returns 'q3b'. if the tape at index is 'x', set the tape at index to 'x', move the index right, then return 'q2b'. if the tape at index is 'c', set the tape at index to 'c', move the index left, then return 'q6b'.

:return: the state which to go to next

def state_q3b(self):

this method checks the tape at the current index. if it is a 'c', set the tape at the index to 'c', move the index left, then returns 'q3b'. if the tape at index is '0', set the tape at index to '0', move the index left, then return 'q3b'. if the tape at index is 'x', set the tape at index to 'x', move the index left, then return 'q3b'. if the tape at index is blank, set the tape at index to '0', move the index right, then return 'q0b'.

:return: the state which to go to next

def state_q4b(self):

this method checks the tape at the current index. if it is a 'x', set the tape at the index to 'x', move the index right, then returns 'q4b'. if the tape at index is '0', set the tape at index to '0', move the index right, then return 'q4b'. if the tape at index is 'c', set the tape at index to 'c', move the index

right, then return 'q4b'. if the tape at index is blank, set the tape at index to '0', move the index left, then return 'q5b'.
:return: the state which to go to next

def state_q5b(self):
this method checks the tape at the current index. if it is a 'c', set the tape at the index to 'c', move the index left, then returns 'q5b'. if the tape at index is '0', set the tape at index to '0', move the index left, then return 'q5b'. if the tape at index is 'x', set the tape at index to 'x', move the index left, then return 'q5b'. if the tape at index is blank, set the tape at index to blank, move the index right, then return 'q0b'.
:return: the state which to go to next

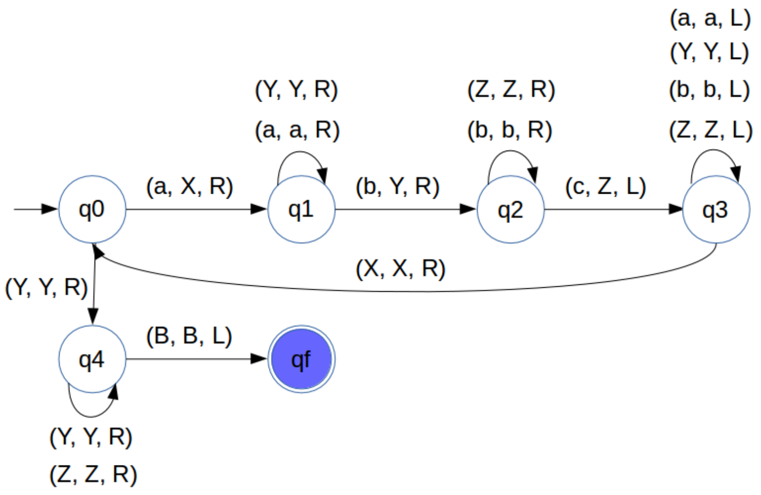
def state_q6b(self):
this method checks the tape at the current index. if it is a 'x', set the tape at the index to blank, move the index left, then returns 'q6b'. if the tape at index is 'c', set the tape at index to blank, move the index left, then return 'q6b'. if the tape at index is '0', set the tape at index to blank, move the index left, then return 'q6b'. if the tape at index is blank, set the tape at index to blank, move the index right, then return 'q7b'.
:return: the state which to go to next

def state_q7b(self):
this method checks the tape at the current index. if it is a blank, set the tape at the index to blank, move the index right, then returns 'q7b'. if the tape at index is '0', set the tape at index to '0', move the index left, then return 'q8b'.
:return: the state which to go to next

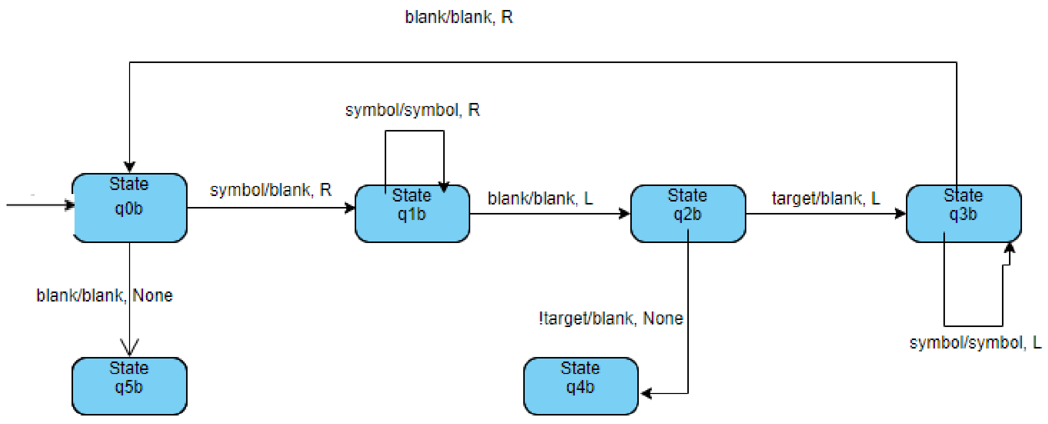
def state_q8b(self):
this method formats the result of the division machine. it strips off the leading and trailing blanks, then uses all the '0's on the left side of the center blanks for the remainder portion of the answer and then uses all the '0's on the right side of the center blanks for the quotient portion of the answer. lastly it returns the formatted answer string
:return: the answer

v. Machine descriptions

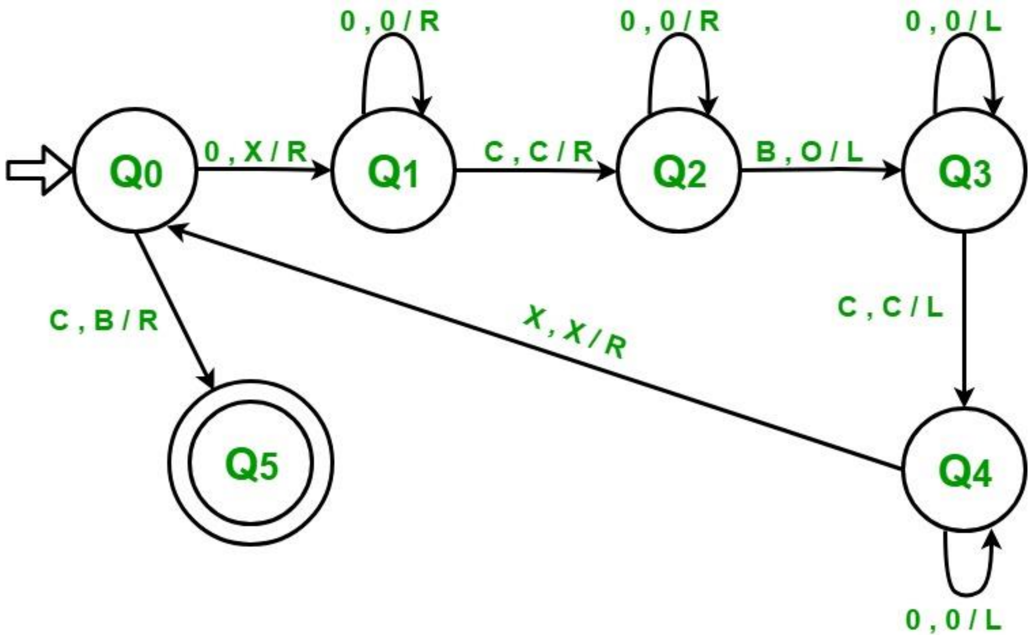
- **a^{n}b^{n}c^n** where a is the first symbol, b is the second symbol and c is the third symbol



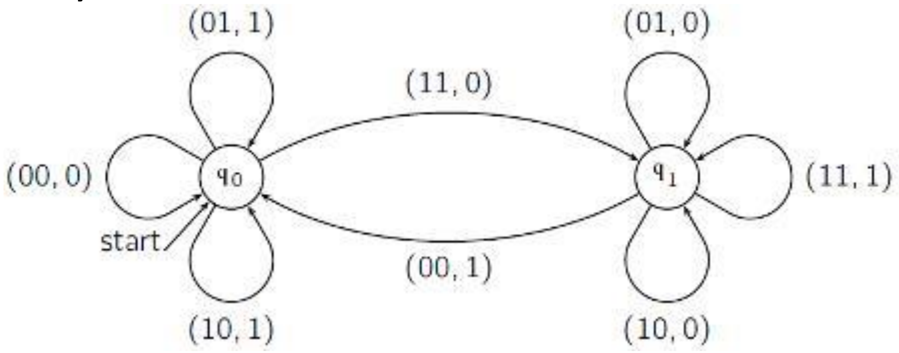
- **ww^R** where q5b is the final state, q4b is the dead state and symbol is any symbol that is not the current target symbol or a blank



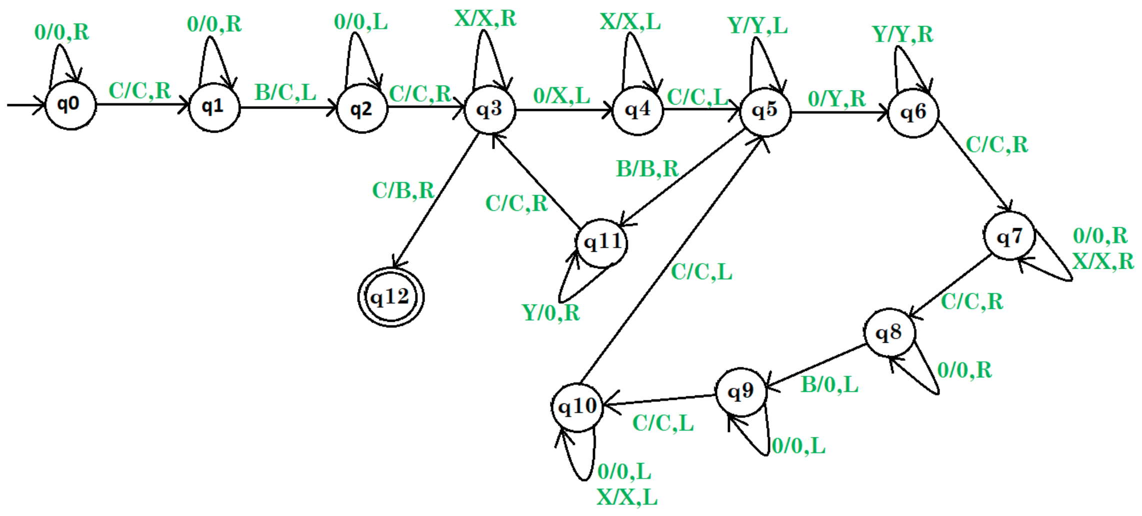
- **Unary addition**



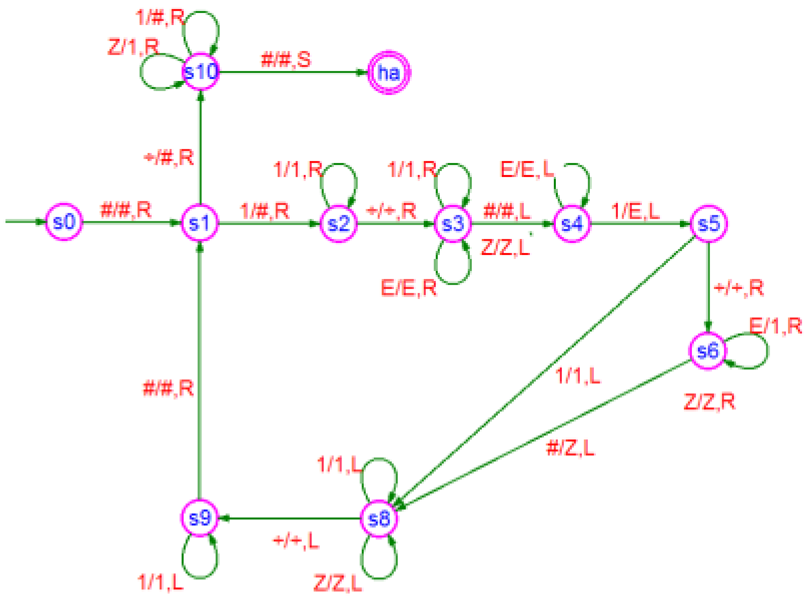
- **Binary addition**



- **Unary multiplication**

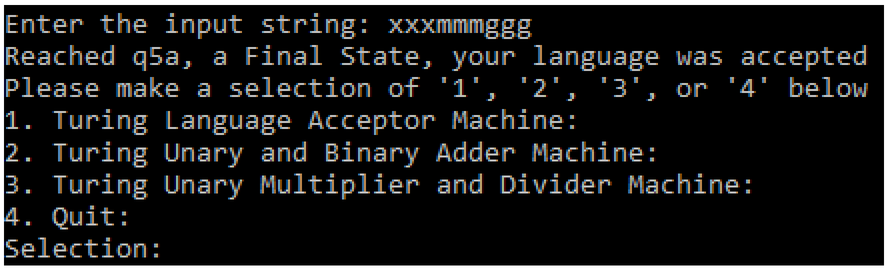


- **Unary division**

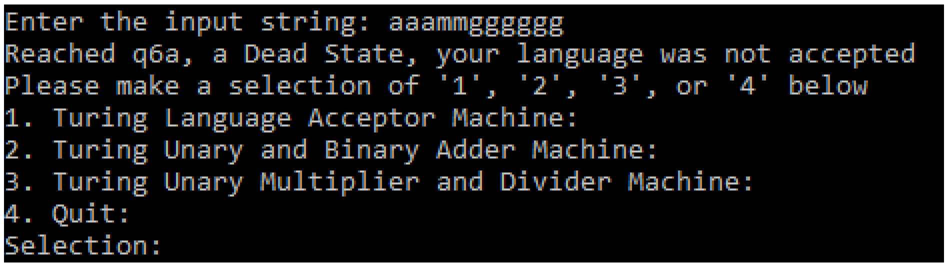


vi. Test words for the machines and expected outputs

a. $a^n b^n c^n$ machine

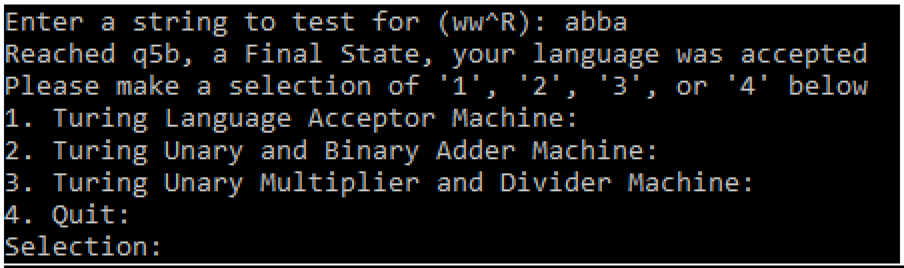


```
Enter the input string: xxxmmmggg
Reached q5a, a Final State, your language was accepted
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```

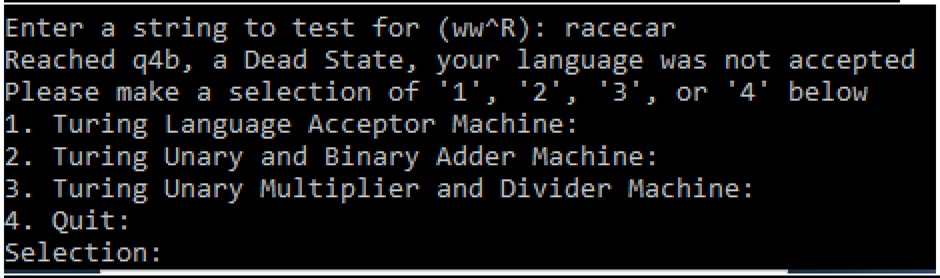


```
Enter the input string: aaammgggggg
Reached q6a, a Dead State, your language was not accepted
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```

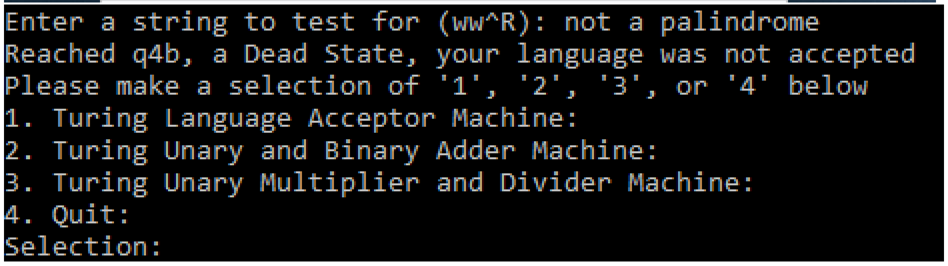
b. **ww^R**



```
Enter a string to test for (ww^R): abba
Reached q5b, a Final State, your language was accepted
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```

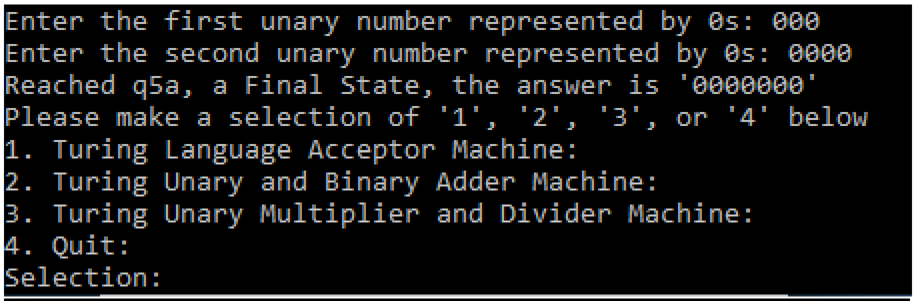


```
Enter a string to test for (ww^R): racecar
Reached q4b, a Dead State, your language was not accepted
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```

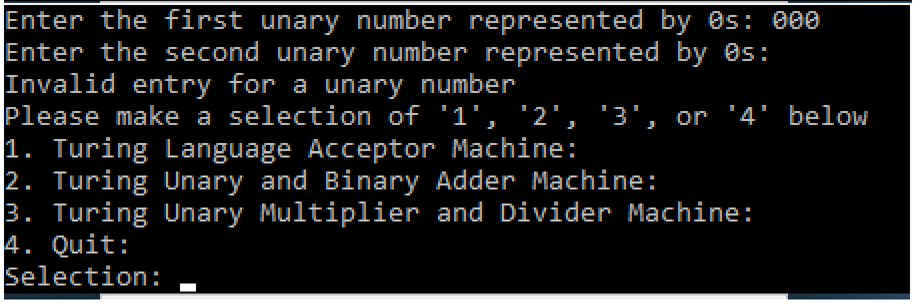


```
Enter a string to test for (ww^R): not a palindrome
Reached q4b, a Dead State, your language was not accepted
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```

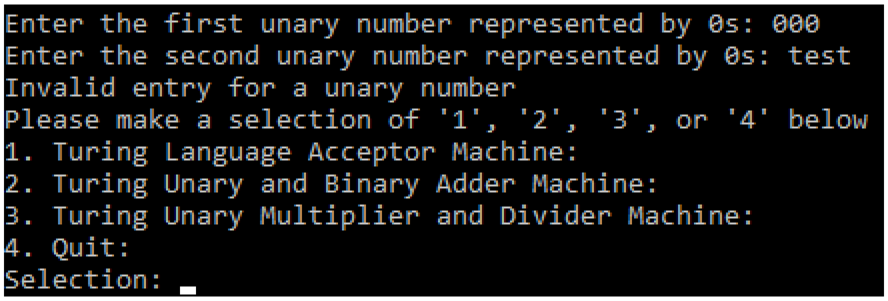
c. **Unary addition**



```
Enter the first unary number represented by 0s: 000
Enter the second unary number represented by 0s: 0000
Reached q5a, a Final State, the answer is '0000000'
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```

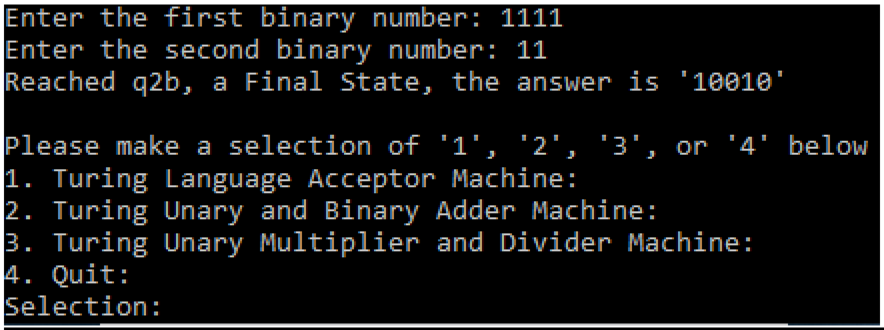


```
Enter the first unary number represented by 0s: 000
Enter the second unary number represented by 0s:
Invalid entry for a unary number
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```



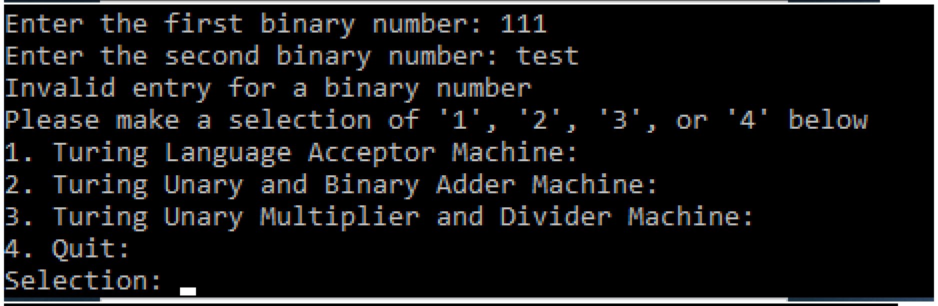
```
Enter the first unary number represented by 0s: 000
Enter the second unary number represented by 0s: test
Invalid entry for a unary number
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```

d. **Binary addition**

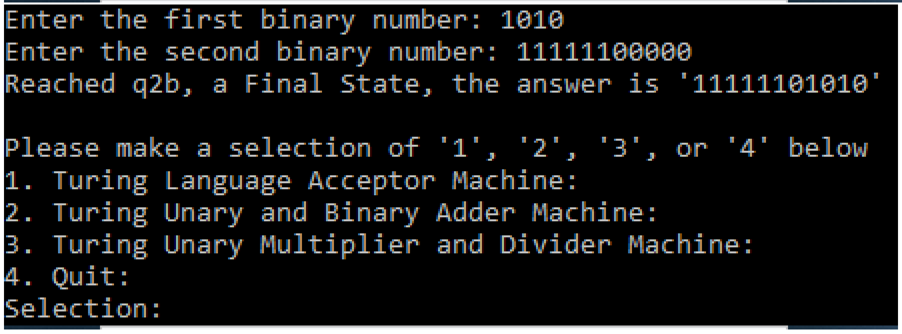


```
Enter the first binary number: 1111
Enter the second binary number: 11
Reached q2b, a Final State, the answer is '10010'

Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```



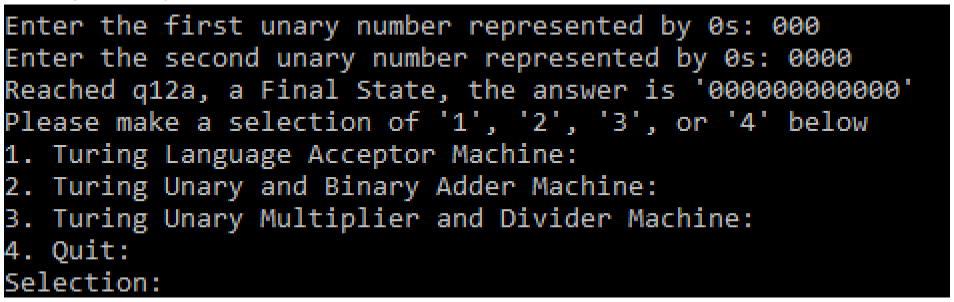
```
Enter the first binary number: 111
Enter the second binary number: test
Invalid entry for a binary number
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```



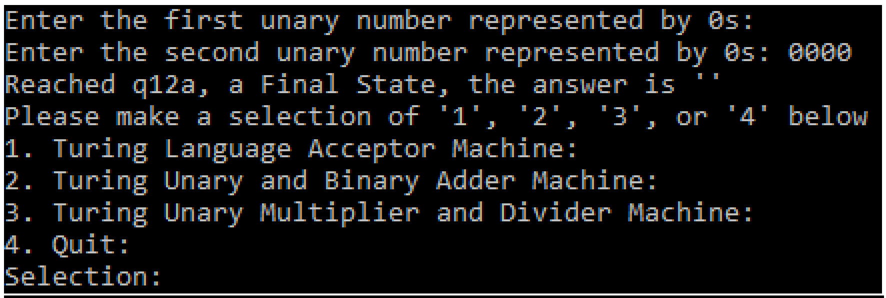
```
Enter the first binary number: 1010
Enter the second binary number: 11111100000
Reached q2b, a Final State, the answer is '11111101010'

Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```

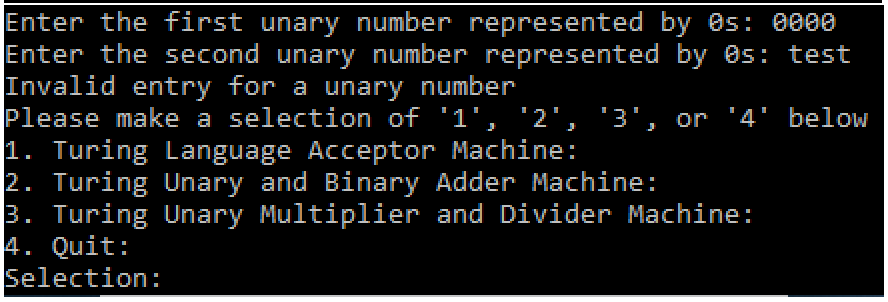
e. **Unary multiplication**



```
Enter the first unary number represented by 0s: 000
Enter the second unary number represented by 0s: 0000
Reached q12a, a Final State, the answer is '000000000000'
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```

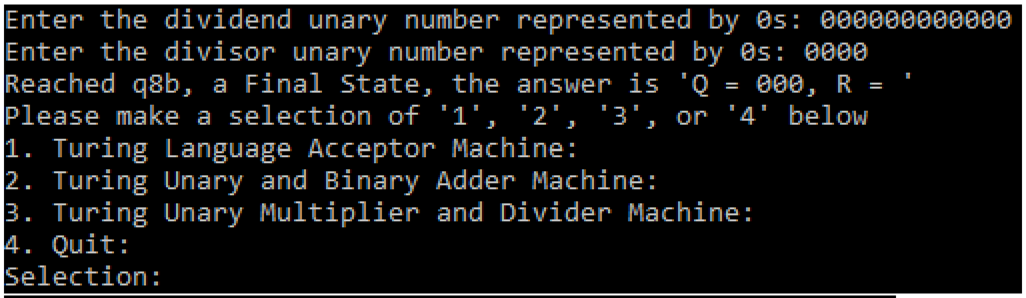


```
Enter the first unary number represented by 0s:
Enter the second unary number represented by 0s: 0000
Reached q12a, a Final State, the answer is ''
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```

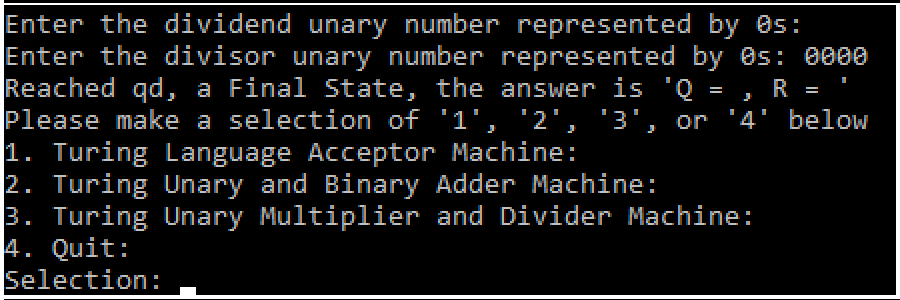


```
Enter the first unary number represented by 0s: 0000
Enter the second unary number represented by 0s: test
Invalid entry for a unary number
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```

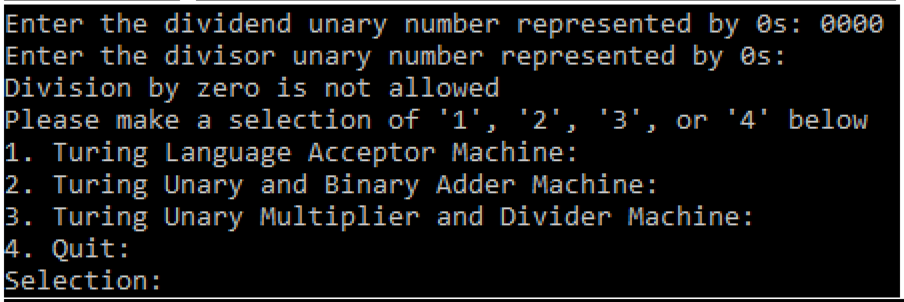
f. **Unary division**



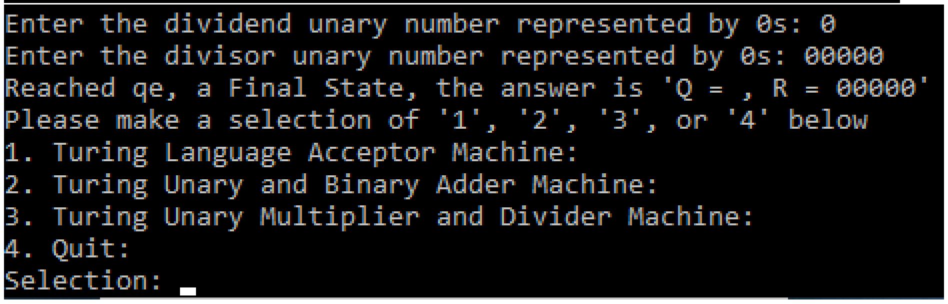
```
Enter the dividend unary number represented by 0s: 000000000000
Enter the divisor unary number represented by 0s: 0000
Reached q8b, a Final State, the answer is 'Q = 000, R = '
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```



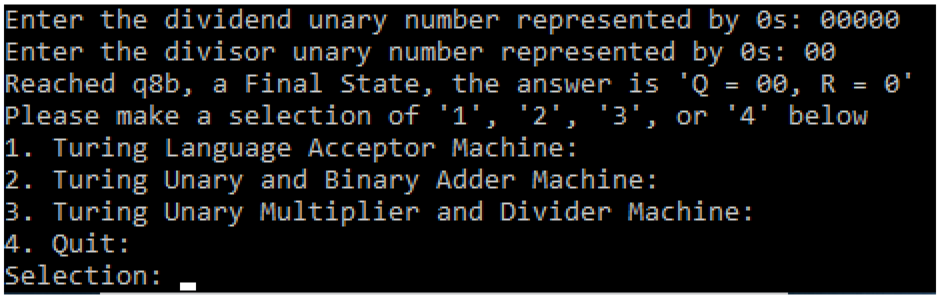
```
Enter the dividend unary number represented by 0s:
Enter the divisor unary number represented by 0s: 0000
Reached qd, a Final State, the answer is 'Q = , R = '
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```



```
Enter the dividend unary number represented by 0s: 0000
Enter the divisor unary number represented by 0s:
Division by zero is not allowed
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```



```
Enter the dividend unary number represented by 0s: 0
Enter the divisor unary number represented by 0s: 00000
Reached qe, a Final State, the answer is 'Q = , R = 00000'
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection:
```



```
Enter the dividend unary number represented by 0s: 00000
Enter the divisor unary number represented by 0s: 00
Reached q8b, a Final State, the answer is 'Q = 00, R = 0'
Please make a selection of '1', '2', '3', or '4' below
1. Turing Language Acceptor Machine:
2. Turing Unary and Binary Adder Machine:
3. Turing Unary Multiplier and Divider Machine:
4. Quit:
Selection: _
```

vii. Limitations of the program and bugs

There are no known bugs or limitations at this time.