Assignment 2

Doubly Text Editor

**DSA Spring 2024**

# Introduction

In this assignment you will implement a text editor using doubly linked lists. This is a non-conventional text editor in which the user enters text through a series of commands *(explained in one of the sections below)*.

You are also given “a2\_starter.zip” which has a single “a2.py” python file. “a2.py” contains bare bones skeletal code which you will complete in this assignment.

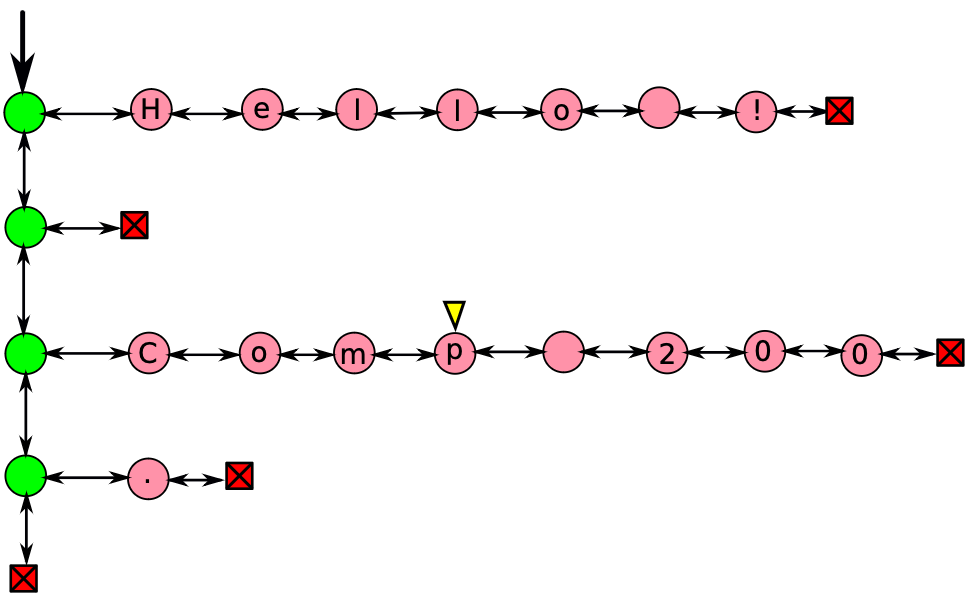
*As before you are encouraged to add any extra helper methods of member variables. But DO NOT modify the names of the classes and methods or the autograder will not work.*

## Internal representation

Suppose that a text document contains the following text:

|  |
| --- |
| Hello ! comp 200  . |

The internal representation of this document is shown by the diagram below:



A doubly linked list *(shown as green above)* is made up of the usual two way nodes. Each green node however contains a reference to a “line” as its data. Each line, shown as a series of pink nodes above, is also a doubly linked list. Each node in the pink row contains a single character as its data. Red squares denote None.

There is also a “cursor” *(shown as yellow triangle above)* which can point to any character in the text. You can freely move it around all over the document. If the user wants to insert or delete text then he first has to move the cursor to the desired position and then perform appropriate operations.

***Important Note: Even though sentinel nodes are convenient as we saw in the class, we are NOT using the header or trailer nodes for doubly linked lists in this assignment. Here just like the singly linked list, the head contains the very first node that actually contains the data. Or head is None if the list is empty. What you see in the diagrams is exactly what you have.***

Also note that the start node of the green linked list is fixed as the name “doc”, short form for document. You are also expected to call this head node the doc and use it likewise. Do not change this name as the autograder will need to read it:



Open up a2.py skeletal file. Following code is already filled in the TextEditor class constructor. DO NOT modify these variables:

|  |
| --- |
| class TextEditor:  def \_\_init\_\_(self):  # WARNING: DO NOT MODIFY THE FOLLOWING VARIABLE NAMES   self.doc = None |

As shown in the figure above, the doc member variable is the ‘head’ of the green linked list. In short if you have the “doc” you have the entire document.

# Project Requirements

## Text Editor Commands

The way a user enters text in this text editor is through a series of commands. Each command and its arguments are entered in a “single line”. There are only a few commands that the user can give as shown in the table below:

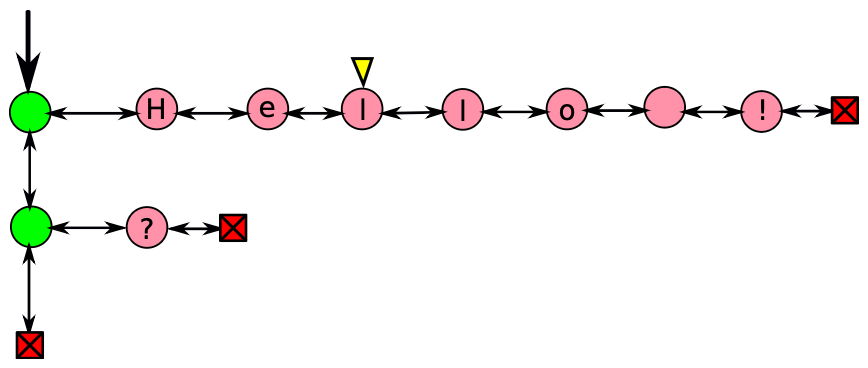
|  |  |
| --- | --- |
| **Command** | **Arguments** |
| goto | row , col |
| forward | - |
| back | - |
| home | - |
| end | - |
| insert | string |
| delete | number of chars to delete |
| countCharacters | - |
| countLines | - |
| printDoc | - |
| quit | - |

## **goto**

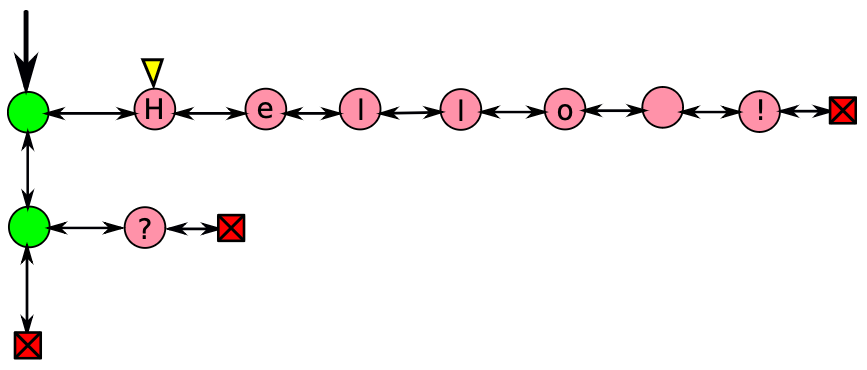
The goto command moves the cursor around. It takes two arguments i.e. *row number* and *column number*, and moves the cursor to that character node. The following command:

|  |
| --- |
| >> goto 0 2 |

Sets the cursor at the character sitting at row 0 and column 2:



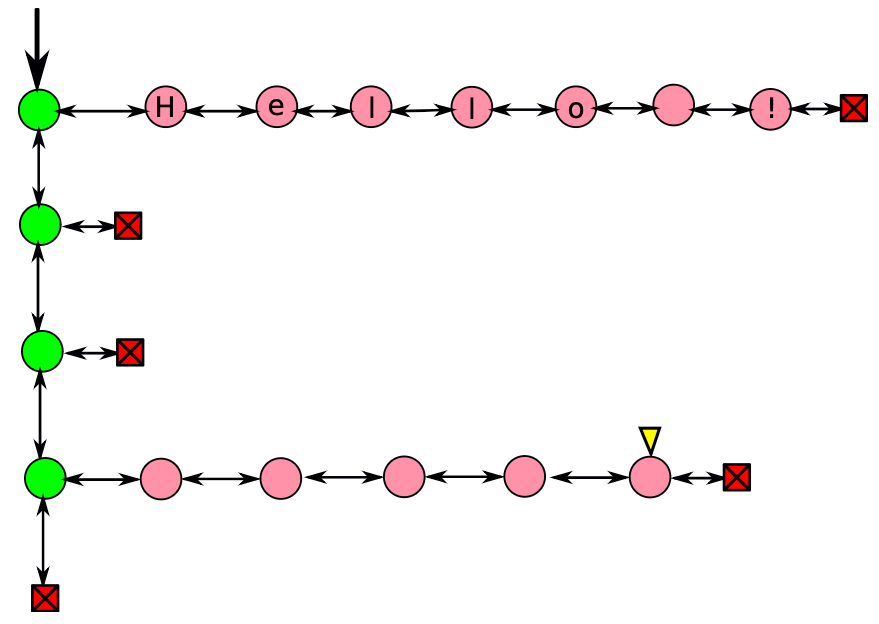
Similarly “goto 0 0” sets the cursor at the character sitting at row 0 and column 0 *(beginning of the document):*



### Special Cases

1. When you create a new document the cursor is invalid i.e it points to (-1,-1) row and column respectively.
2. If you try to send the cursor to any row or a column that does not exist, then it will create one list at that row *(with nodes containing the space character)* and then move the cursor to the desired character.
   1. *It can either create an entire empty row*
   2. *It can append empty nodes after a currently existing line*.

For example “goto 3 4” will create a new list of length “4” at row number 3 *(4th row)* as shown below:



***Note:*** *it skipped row number 1 and 2.*

1. Negative values for the row and column will be ignored. For example suppose if the cursor is at (3,4) then the following commands will have to effect on the position of the cursor:

|  |
| --- |
| >> goto -3 8 *(ignored)* >> goto 10 -3 *(ignored)* >> goto -5 -6 *(ignored)* |

***Note:*** *The cursor can never point to any of the green nodes or even the red squares. It can only point to the pink nodes.*

For this command to work you will have to implement the goto method. There are two parameters but no return value for this method:

|  |
| --- |
| def goto(self, row, col):   raise NotImplementedError |

## **forward**

Moves the cursor one step forward.

### Special Cases

1. If the cursor is already at the last character of the current line then forwarding it will move it to the first character of the next line.
2. If there is no next line *(i.e. the next line is empty or it’s the end of the document)* then it will remain at its current position.
3. forward operation on an invalid cursor shall be ignored.

For the forward command to work you will have to implement the forward method. There are no parameters or any return value for this method:

|  |
| --- |
| def forward(self):  raise NotImplementedError |

## **back**

Moves the cursor one step back.

### Special Cases

1. If the cursor is already at the first character of the current line then backing it will move it to the last character of the previous line.
2. If there is no previous line (i.e. *the previous line is empty or it’s the start of the document*) then the cursor will remain at its current position.
3. back operation on an invalid cursor shall be ignored.

For the back command to work you will have to implement the back method. There are no parameters or any return value for this method:

|  |
| --- |
| def back(self):  raise NotImplementedError |

## **home**

The home command will bring the cursor to the very first character of the current line.

### Special Cases

1. home operation on an invalid cursor shall be ignored.

For the home command to work you will have to implement the home method. There are no parameters or any return value for this method:

|  |
| --- |
| def home(self):  raise NotImplementedError |

## **end**

The end command will bring the cursor to the very last character of the current line.

### Special Cases

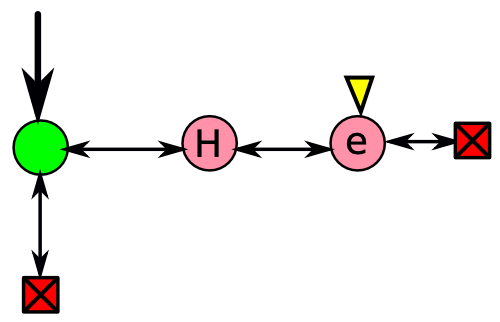
1. end operation on an invalid cursor shall be ignored.

For the end command to work you will have to implement the end method. There are no parameters or any return value for this method:

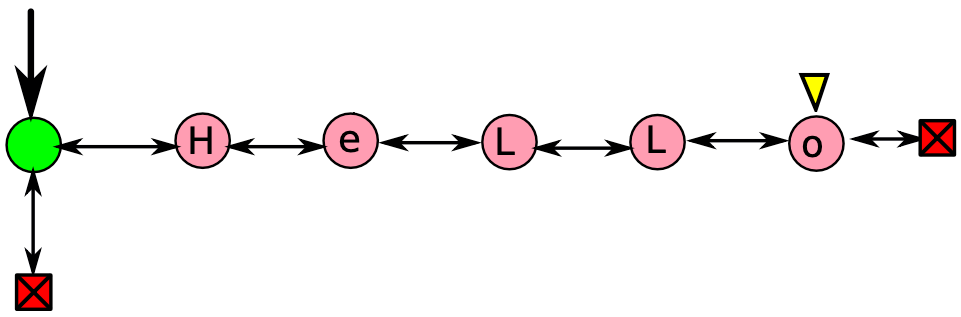
|  |
| --- |
| def end(self):  raise NotImplementedError |

## **insert**

The insert command takes one argument namely a text string to insert. It then inserts the string *“immediately after”* the cursor position. The string will be divided into individual characters before inserting in the document nodes. For example if the document was:



then the command “insert LLo” will result in the string “LLo” to be inserted immediately after the cursor:



The new cursor position will point to the last character of the inserted string, in this case ‘o’.

### Special Cases

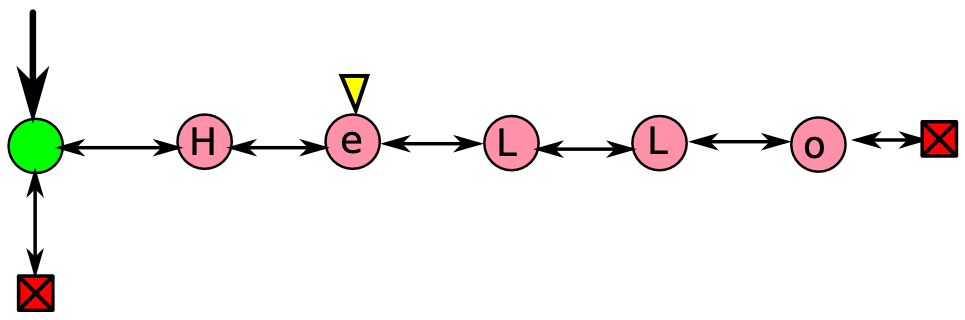
1. As we know that an invalid cursor (-1,-1) corresponds to an empty document, Inserting text at an invalid cursor will first move the cursor to (0,0) and then it will insert the text at the new cursor location.

For the insert command to work you will have to implement the insert method. This method takes a string as a parameter and returns nothing:

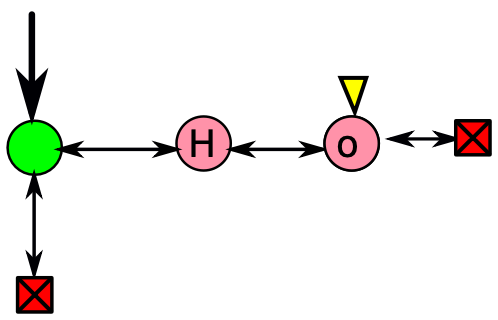
|  |
| --- |
| def insert(self, string):  raise NotImplementedError |

## **delete**

The delete command deletes a number of characters from the current cursor position *(including the character at the cursor position)*. It takes a number as an argument which specifies the number of characters to delete. For example if the initial document was:



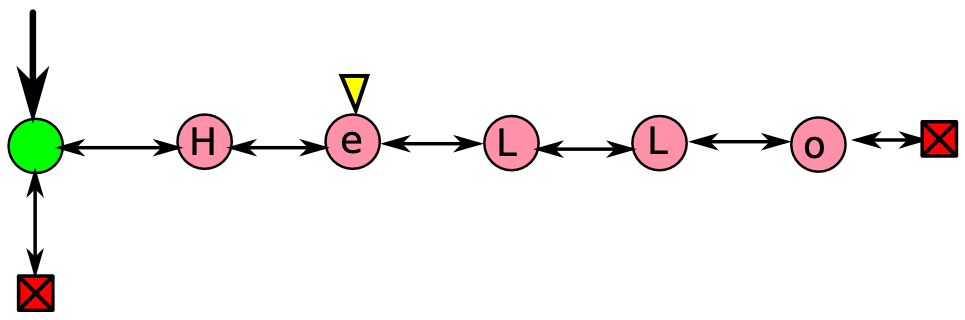
the command “delete 3” deletes three characters at and including the cursor position:



In normal cases the cursor position is not affected due to the delete command.

### Special Cases

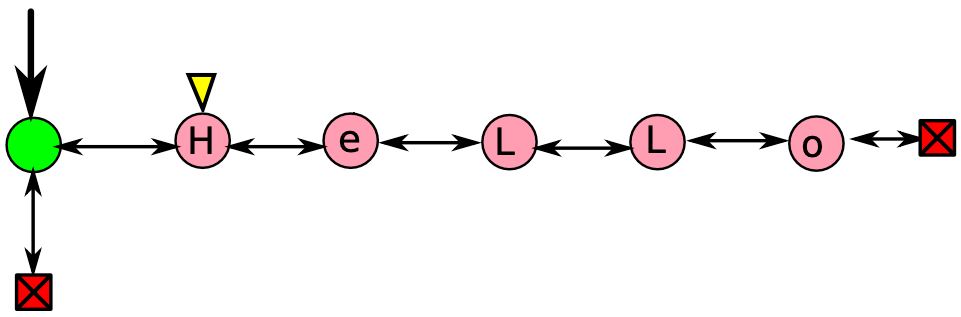
1. Running delete command at an invalid cursor has no effect.
2. Consider the following figure again. What if you delete 10 characters?



|  |
| --- |
| >> delete 10 |

It will delete the 4 available characters in this line and the cursor will point to the immediately previous character. In this example the cursor will point to ‘H’.

1. What if you want to delete the entire line containing “Hello”:



In this case the cursor will point to the first character of the next line.

1. The delete command should ignore negative inputs:

|  |
| --- |
| >> delete -5 *(ignored)* |

For the delete command to work you will have to implement the delete method. This method takes an integer as a parameter and returns nothing:

|  |
| --- |
| def delete(self, num):  raise NotImplementedError |

## **countCharacters**

The countCharacters command returns the total number of characters in the entire document. Since the user can insert anything as a character, not just the alphabet, by *“character”* we mean a *“pink node”*.

For the countCharacters command to work you will have to implement the countCharacters method. This method takes no parameters and but it returns the total number of characters *(or pink nodes)* in the document:

|  |
| --- |
| def countCharacters(self):  raise NotImplementedError |

## **countLines**

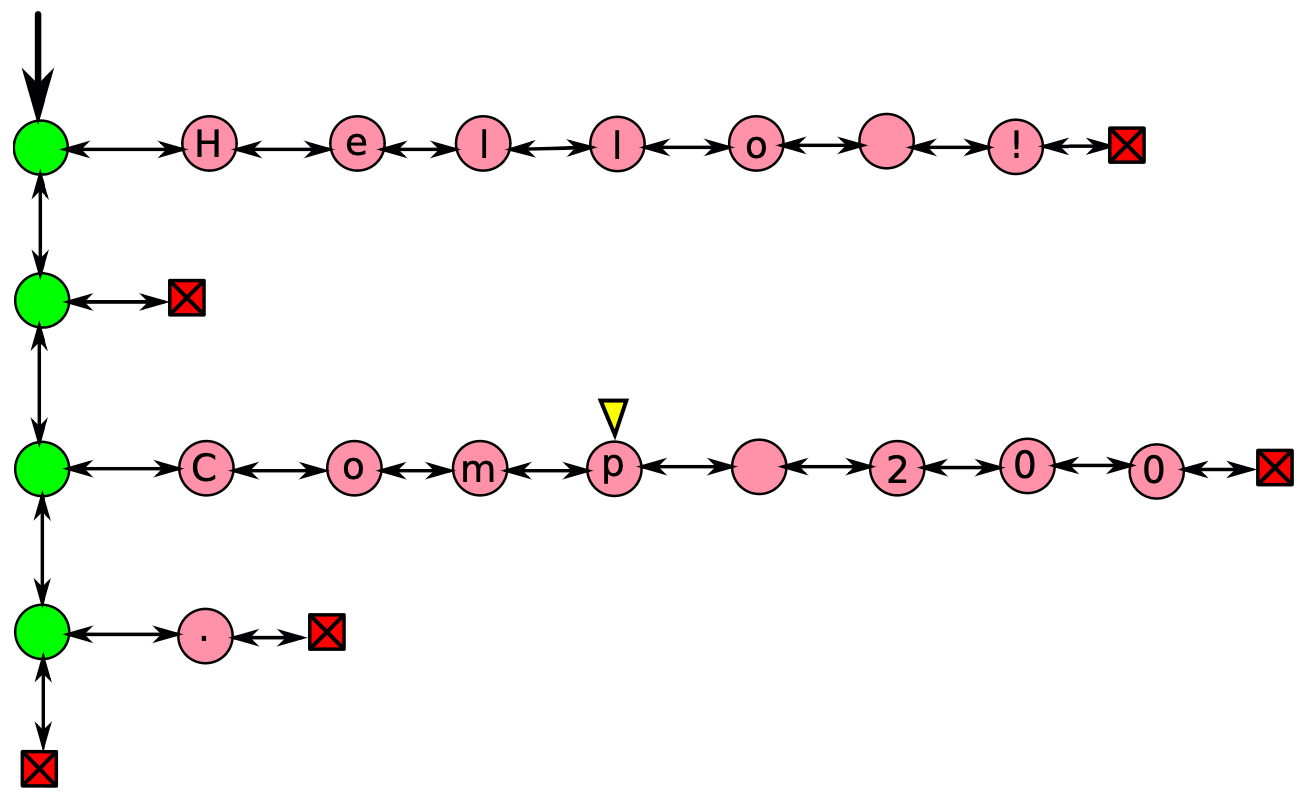
The countLines command returns the total number of non-empty lines in the entire document. It does not count any empty lines.

For the countLines command to work you will have to implement the countLines method. This method takes no parameters and it returns an integer denoting the total number of non-empty lines:

|  |
| --- |
| def countLines(self):  raise NotImplementedError |

## **printDoc**

The printDoc command prints the internal representation of the document in a text form on the screen. For example:



will be printed on the screen as:

|  |
| --- |
| Hello !  com|p 200  . |

Note that the cursor is printed as a “|” pipe character on the screen. The pipe character is only for output, it is not stored anywhere in the actual document.

You have to implement the printDoc method. This function has no parameters nor any return value:

|  |
| --- |
| def printDoc(self):  raise NotImplementedError |

## **quit**

Finally the quit command quits the program.

# Program Execution

Your program will print out a greeting message and then it will wait for the user to enter a series of commands. As soon as the user enters the command it will execute it and then displays the next prompt in a new line:

|  |
| --- |
| Welcome to DS Text Editor Enter commands at the prompt  >> goto 0 0 >> insert Comp 200 >> goto 2 0 >> insert This editor is >> goto 2 21 >> insert -=COOL=- >> goto 3 0 >> insert Bye >> goto 4 0 >> insert . >> countLines 5 >> printDoc Comp 200 This editor is -=COOL=- Bye . >> quit |

# Bonus

Bonus commands are given in the table below:

|  |  |
| --- | --- |
| **Command** | **Arguments** |
| save | *file name* |
| load | *file name* |
| undo | *-* |
| redo | *-* |
| find | *substring* |

The details are as follows:

* **save**

The save command takes the filename as one argument. It saves the document in that file.

* **load**

The load command takes the filename as one argument. It reads the given file and loads the contents of that file into the running program.

* **undo**

undo the most recent command and take the user one step back.

* **redo**

redo the most recent undo command and take the user one step forward again.

* **find**

Finds a given substring within the entire document and returns the reference to the first character of that substring. If no such substring is found then it returns “None”.

# Marks Distribution

Total marks: *[170 marks]*

* *[130 marks]* Commands
  + *[20 marks]* goto
  + *[10 marks]* forward
  + *[10 marks]* back
  + *[10 marks]* home
  + *[10 marks]* end
  + *[20 marks]* insert
  + *[20 marks]* delete
  + *[10 marks]* countCharacters
  + *[5 marks]* countLines
  + *[10 marks]* printDoc
  + *[5 marks]* quit
* *[25 marks]* Driver program, accept and execute commands
* *[15 marks]* Error free, Bug free code, Great design, OOP, Good programming practices etc.
* *[65 marks]* Bonus
  + *[10 marks]* save
  + *[10 marks]* load
  + *[15 marks]* undo
  + *[15 marks]* redo
  + *[15 marks]* find

# Submission

Please make submission on Moodle