Sri Sivasubramaniya Nadar College of Engineering
Kalavakkam – 603 110
(An Autonomous Institution, Affiliated to Anna University, Chennai)

Department of Information Technology

UIT1311 – Programming and Data Structures Lab

Mini Project – 2021-2022

III SEMESTER IT – INSTANCE B

TEXT EDITOR

1. 205002038 – HARISH A -- Batch 1
2. 205002051 – K MUGHILAN -- Batch 2
3. 205002301– AKKASH ANNIYAPPA C S -- Batch 2

1.

2.

3.

Signature of the team member (s)   Signature of the faculty

Project title: TEXT EDITOR

Problem statement:

Create a simple text editor that stores and displays a string of characters. Your editor should support the following operations:
• left: Move pointer left one character (do nothing if at beginning).
• right: Move pointer right one character (do nothing if at end).
• insert c: Insert the character c just after the pointer.
• delete: Delete the character just after the pointer (do nothing at end).

Choice of data structure:

Doubly linked list data structure was used to implement the project proposed.

```
# Node of a doubly linked list

class Node:

    def __init__(self, next=None, prev=None, data=None):

        self.next = next # reference to next node in DLL

        self.prev = prev # reference to previous node in DLL

        self.data = data
```

Detailed technical information:

In computer science, a doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains three fields: two link fields (references to the previous and to the next node in the sequence of nodes) and one data field. The beginning and ending nodes' previous and next links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list. If there is only one sentinel node, then the list is circularly linked via the sentinel node. It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequential orders.

A doubly linked list whose nodes contain three fields: an integer value, the link to the next node, and the link to the previous node.

A doubly linked list whose nodes contain three fields: the link to the previous node, an integer value, and the link to the next node.

The two node links allow traversal of the list in either direction. While adding or removing a node in a doubly linked list requires changing more links than the same operations on a singly linked list, the operations are simpler and potentially more efficient (for nodes other than first nodes) because there is no need to keep track of the previous node during traversal or no need to traverse the list to find the previous node, so that its link can be modified.

The first and last nodes of a doubly linked list are immediately accessible (i.e., accessible without traversal, and usually called head and tail) and therefore allow traversal of the list from the beginning or end of the list, respectively: e.g., traversing the list from beginning to end, or from end to beginning, in a search of the list for a node with specific data value. Any node of a doubly linked list, once obtained, can be used to begin a new traversal of the list, in either direction (towards beginning or end), from the given node.

The link fields of a doubly linked list node are often called next and previous or forward and backward. The references stored in the link fields are usually implemented as pointers, but (as in any linked data structure) they may also be address offsets or indices into an array where the nodes live.

Methodology:

1) Concept:
Using doubly linked list node to perform text editor operations by initializing head as the cursor "|".

2) Operations:
- Inserting a character to the left of the cursor
- Moving the cursor to the left
- Moving the cursor to the right
- Deleting the character to the right of the cursor
- Saving the changes made to the dll into a '.txt' file specified (or "*.txt" in default)
- Opening the requested '.txt' file and initializing it into the dll (or in default opens "*.txt")
- Printing the current contents of the dll
- Displaying legend of the keys corresponding to the operations for better user interface

3) Algorithm:

1. Declare a class Node which is the basic structure for Doubly linked list.

2. Make an ADT to perform operations which a texteditor does.

3. The ADT's constructor has a variable head which initialises Node('|') that is the cursor for our text editor.

4. Declare the methods insert, left, right, delete, printL, save, get, help.

4.1. Insert()

    4.1.1. Gets a character from the user and initialise the character-{data} as Node(data) to the newNode.

    4.2.2. Checks if head value is '|'. If yes, then insert the newNode before the head and give its necessary links and change the head to newNode.

    4.2.3. Else. save the head to temp and traverse it till the Node('|') is found. Then save get the next Node to the temp and save it as temp2. Now insert the newNode in between temp and temp2.

4.2. Left()

    4.2.1. save the head to temp

    4.2.2. Traverse the temp till the Node('|') is found.

    4.2.3. if temp.prev is None then PASS

    4.2.4. else save prev Node to the temp to temp2. Swap the values in temp and temp2

4.3. Right()

    4.2.1. Save the head to temp

    4.2.2. Traverse the temp till the Node('|') is found.

    4.2.3. If temp.next is None then PASS

    4.2.4. Else save next Node to the temp to temp2. Swap the values in temp and temp2

4.4. Delete()

    4.4.1. Save the head to temp

    4.4.2. Traverse the temp till the Node('|') is found.

    4.4.3. If next of temp is None then pass

    4.4.4. Else

        4.4.4.1. if next of next of temp is None then save next of temp to temp2 and remove the links of temp2.

        4.4.4.2. else save the next of temp to temp2 then change next of temp to next of temp2, next of temp2 to temp2, temp to prev of temp2.

4.5. printL()

      4.5.1. Save the head to temp.

      4.5.2. Traverse the temp till it's not None.

      4.5.3. Print the value of temp while traversing.

4.6. Save()

      4.6.1. Open the file with the given file name by user.

      4.6.2. Save the head to temp.

      4.6.3. Traverse the temp till it's not None.

      4.6.4. if value of temp is '|' traverse it.

      4.6.5. else append the value of temp and save it into the file and traverse it.

4.7. Get()

      4.7.1. Assign self.head as Node('|')

      4.7.2. Save the head to temp.

      4.7.3. get the values of string from the file.

      4.7.4. Traverse till the string is not None.

            4.7.4.1. Save the Node with the character as its value to the newNode

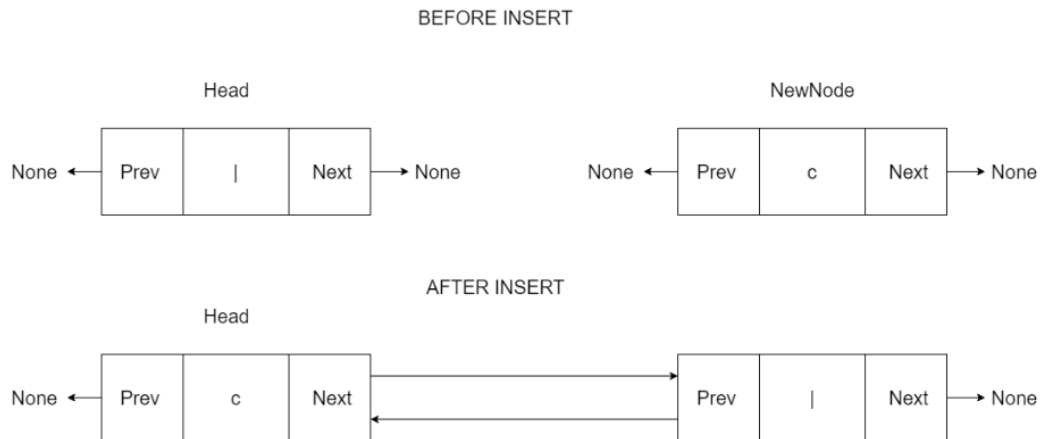   4.7.4.2. Link the newNode next to the temp using dll implmentations.

4.8. Help()

      4.8.1. Print the necessary methods to perform for a better user interface.

5. Define a User Interface to use the texteditor_ADT and perform its operations
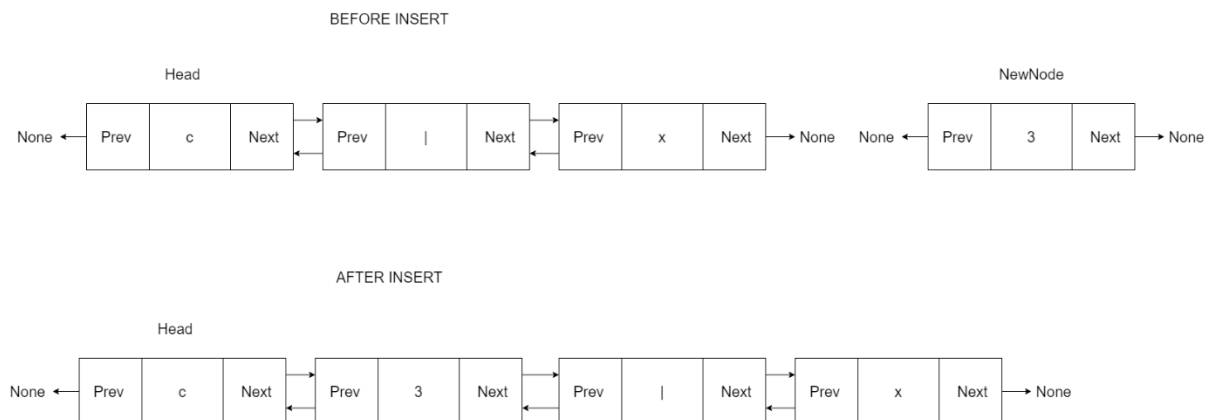
4) Diagrammatic Representation:

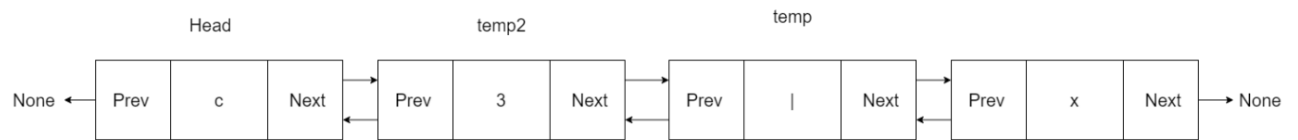1. Inserting character into the dll when the dll has only cursor as the node.

Case 1

BEFORE INSERT

	Head					NewNode		
None ←	Prev			Next	→ None	None ←	Prev	c	Next	→ None

AFTER INSERT

	Head							
None ←	Prev	c	Next			Prev			Next	→ None

2) Inserting into the dll when the dll has entries other than the cursor in it.

Case 2

BEFORE INSERT

Head   NewNode

| None ← | Prev | c | Next | Prev | | | Next | Prev | x | Next | → None   None ← | Prev | 3 | Next | → None |

AFTER INSERT

Head

| None ← | Prev | c | Next | Prev | 3 | Next | Prev | | | Next | Prev | x | Next | → None |

3. Moving the cursor to its left.

BEFORE LEFT

	Head			temp2			temp				
None ←	Prev	c	Next →	Prev	3	Next →	Prev	|	Next →	Prev	x	Next → None

AFTER LEFT

	Head			temp2			temp				
None ←	Prev	c	Next →	Prev	|	Next →	Prev	3	Next →	Prev	x	Next → None
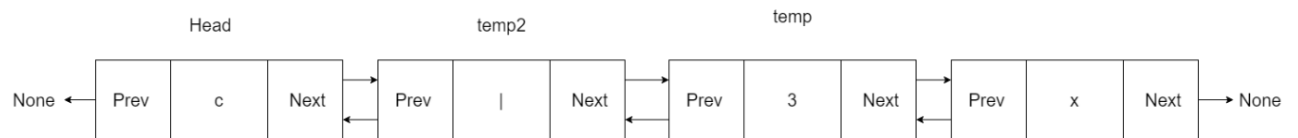
4. Moving the cursor to its right.

BEFORE RIGHT

	Head					temp			temp2		
None ←	Prev	c	Next →	Prev	3	Next →	Prev	|	Next →	Prev	x	Next → None

AFTER RIGHT

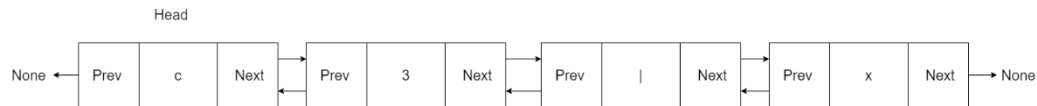|	Head					temp			temp2		
None ←	Prev	c	Next →	Prev	3	Next →	Prev	x	Next →	Prev	|	Next → None
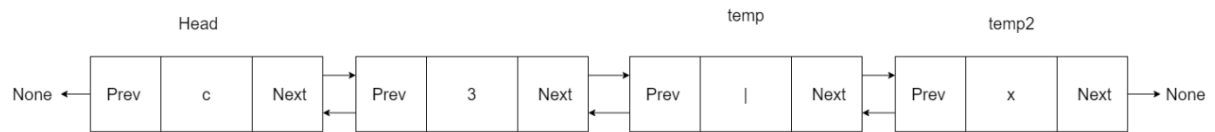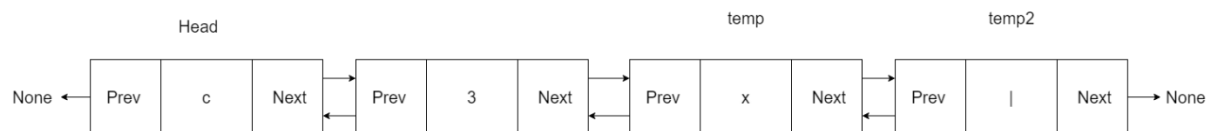
5. Deleting the entry to the right of the cursor where the cursor has only one node to its right.
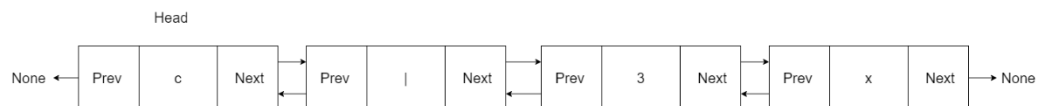

Case 1

BEFORE DELETE

Head

None ← | Prev | c | Next | → | Prev | 3 | Next | → | Prev | \| | Next | → | Prev | x | Next | → None

DURING DELETE

Head

None ← | Prev | c | Next | → | Prev | 3 | Next | → | Prev | \| | Next | → | Prev | x | Next | → None

AFTER DELETE

Head

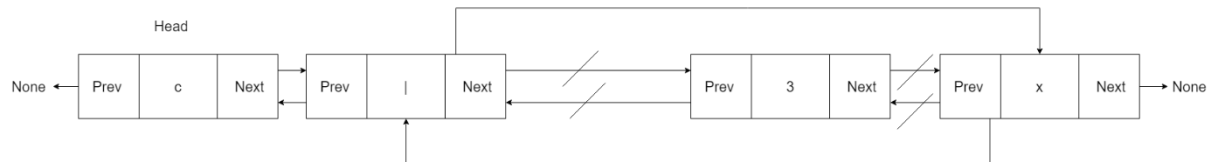None ← | Prev | c | Next | → | Prev | 3 | Next | → | Prev | \| | Next | → None

6. Deleting the entry to the right of the cursor where the cursor has more than one node to its right.
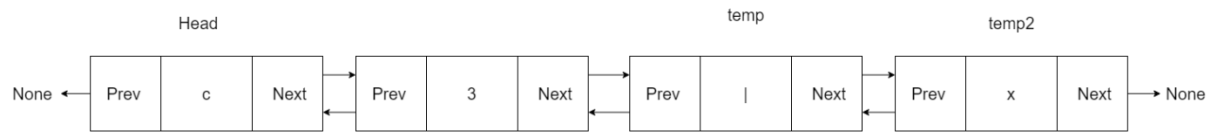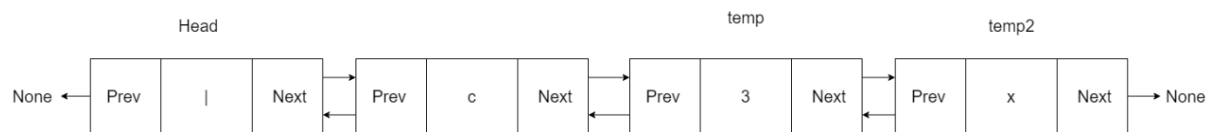

Case 2

BEFORE DELETE

Head

None ← | Prev | c | Next | → | Prev | \| | Next | → | Prev | 3 | Next | → | Prev | x | Next | → None

DURING DELETE

Head

None ← | Prev | c | Next | → | Prev | \| | Next | → | Prev | 3 | Next | → | Prev | x | Next | → None

AFTER DELETE

None ← | Prev | c | Next | → | Prev | \| | Next | → | Prev | x | Next | → None

7. Dll after save and open operation.

SAVE

	Head						temp			temp2	
None ←	Prev	c	Next	Prev	3	Next	Prev	|	Next	Prev	x	Next → None

OPEN

	Head						temp			temp2	
None ←	Prev	|	Next	Prev	c	Next	Prev	3	Next	Prev	x	Next → None

5) Source Code:

Contents of texteditor_ADT.py:

```python
import os
# Node of the doubly linked list
class Node:
  def __init__(self,val,prev=None,next=None):
    self.val=val
    self.prev=prev
    self.next=next

# Body of text editor ADT
class texteditor:
  # initializing cursor
  def __init__(self):
    self.head=Node('|')
  # method insert
  def insert(self,data):
    newNode=Node(data)
    # If the dll is empty
    if self.head.val=='|':
      newNode.next=self.head
      newNode.prev=None
      self.head.prev=newNode
      self.head=newNode
    # if the dll is not empty
    else:
      temp=self.head
      while(temp.next.val!='|'):
        temp=temp.next
      temp2=temp.next
      newNode.next=temp.next
      newNode.prev=temp
      temp2.prev=newNode
      temp.next=newNode
  # function to move the cursor to the left
  def left(self):
    temp=self.head
    while(temp.val!='|'):
      temp=temp.next
    # if there is no entry to the left of the cursor
    if temp.prev==None:
      pass
    # if the left side of the cursor is not empty
    else:
```

```python
      temp2=temp.prev
      temp2.val,temp.val=temp.val,temp2.val
  # function to move the cursor to the right
  def right(self):
    temp=self.head
    while(temp.val!='|'):
      temp=temp.next
    # if there is no entry to the right of the cursor
    if temp.next==None:
      pass
    # if the right side of the cursor is not empty
    else:
      temp2=temp.next
      temp2.val,temp.val=temp.val,temp2.val
  # function to delete the character to the right of the
cursor
  def delete(self):
    temp=self.head
    while(temp.val!='|'):
      temp=temp.next
    # if there is no entry to the right of the cursor
    if temp.next==None:
      pass
    # if there is entry to the right of the cursor
    else:
      # if there is only one entry to the right of the cursor
      if temp.next.next==None:
        temp2=temp.next
        temp2.prev=None
        temp.next=None
      # if there is more than one entry to the right of the
cursor
      else:
        temp2=temp.next
        temp.next=temp2.next
        temp2=temp2.next
        temp2.prev=temp

  # function to print the dll
  def printL(self):
    temp=self.head
    while temp:
      print(temp.val,end='')
      temp=temp.next
    print()
  # function to save the dll in .txt file
  def save(self,file="*.txt"):
```

```python
        f=open(file,"wt")
        s=[]
        strin=""
        temp=self.head
        while temp:
          if temp.val!='|':
            s.append(temp.val)
            temp=temp.next
          else:
            temp=temp.next
        strin=strin.join(s)
        f.write(strin)
        f.close()
    # function to open the requested .txt file
    def get(self,file="*.txt"):
        if os.path.isfile(file):
          print('File Opened...')
          f=open(file,"rt")
          strin=f.read()
          self.head=Node('|')
          temp=self.head
          for s in strin:
            newNode=Node(s)
            temp.next=newNode
            newNode.prev=temp
            temp=temp.next
          f.close()
        else:
          print('File doesnt exists!!!')
    # legend to assist the user with the keys corresponding to
specific operations
    def help(self):
        print('"i <char> or I <char>" to insert a character before
the cursor')
        print('"l or L" to move the cursor to left')
        print('"r or R" to move the cursor to right')
        print('"d or D" to delete the character after the cursor')
        print('"p or P" to print the values in the text editor')
        print('"s <filename> or S <filename>"" to save the text to
the file <filename> or it saves in the file *.txt if
<filename> argument is not present')
        print('"o <filename> or O <filename>"" to open the text in
file <filename> or it opens the file *.txt if <filename>
argument is not present')
        print('"x or X" to close the text editor')
```

Contents of cmd.py:

```python
# importing the ADT
from texteditor_ADT import texteditor
# function to perform the requested operations
def cmd():
  te=texteditor()
  while True:
    cmd=input(">>")
    cmd=cmd.split()
    if(cmd==[]):
      pass
    elif (cmd[0]=='i' or cmd[0]=='i'):
      if (len(cmd[1])==1):
        te.insert(cmd[1])
      else:
        print('Can insert only one character.... ')
    elif (cmd[0]=='p' or cmd[0]=='P'):
      te.printL()
    elif (cmd[0]=='l' or cmd[0]=='L'):
      te.left()
    elif (cmd[0]=='r' or cmd[0]=='R'):
      te.right()
    elif (cmd[0]=='d' or cmd[0]=='D'):
      te.delete()
    elif (cmd[0]=='x' or cmd[0]=='X'):
      exit()
    elif (cmd[0]=='s' or cmd[0]=='S'):
      if len(cmd)==1:
        te.save()
        print('File saved in *.txt')
      elif len(cmd)==2:
        te.save(str(cmd[1]))
    elif (cmd[0]=='o' or cmd[0]=='O'):
      if len(cmd)==1:
        print('Opening file *.txt')
        te.get()
      elif len(cmd)==2:
        print(f'Opening file {str(cmd[1])} .txt')
        te.get(str(cmd[1]))
    elif (cmd[0]=='h' or cmd[0]=='H'):
      if len(cmd)==1:
        te.help()
    else:
      print("Wrong command !!!")
      print('Try using "h or H" for help')
cmd()
```

6) Sample Input/Output:

Test case-1(Doing various text editing operations):

```
>>7                                                              Q ×
Wrong command !!!
Try using "h or H" for help
>>h
"i <char> or I <char>" to insert a character before the cursor
"l or L" to move the cursor to left
"r or R" to move the cursor to right
"d or D" to delete the character after the cursor
"p or P" to print the values in the text editor
"s <filename> or S <filename>"" to save the text to the file <filename>
 or it saves in the file *.txt if <filename> argument is not present
"o <filename> or O <filename>"" to open the text in file <filename> or
it opens the file *.txt if <filename> argument is not present
"x or X" to close the text editor
>>i 1
>>i q
>>i 8
>>p
1q8|
>>l
>>p
1q|8
>>l
>>p
1|q8
>>r
>>p
1q|8
>>d
>>p
1q|
>>s text.txt
>>p
1q|
>>o text.txt
Opening file text.txt
File Opened...
>>p
|1q
>>x
```
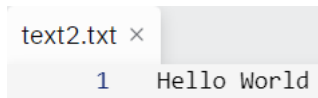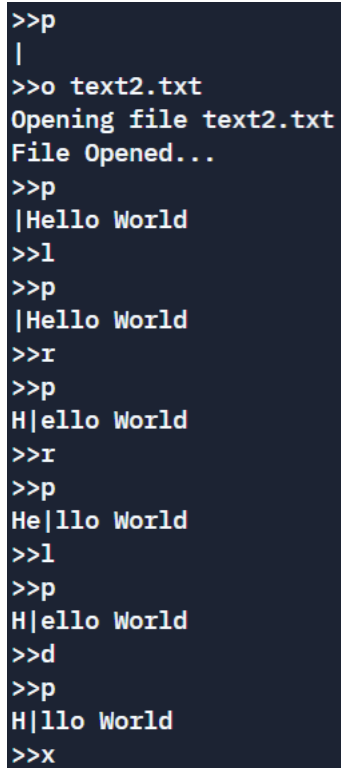
Contents of text.txt:

```
text.txt ×

    1    1q
```

Test case-2(Opening a ".txt" with pre-existing entries and importing it as a dll):

Contents of text2.txt:

text2.txt ×

```
1    Hello World
```

```
>>p
|
>>o text2.txt
Opening file text2.txt
File Opened...
>>p
|Hello World
>>l
>>p
|Hello World
>>r
>>p
H|ello World
>>r
>>p
He|llo World
>>l
>>p
H|ello World
>>d
>>p
H|llo World
>>x
```

Complexity Analysis:

Method	Best Case	Average Case	Worst Case
insert	O(1)	O(n)	O(n)
delete	O(1)	O(n)	O(n)
left	O(1)	O(n)	O(n)
right	O(1)	O(n)	O(n)

Summary:

The given project prototype has been developed successfully. The project can be further developed by incorporating to front-end applications.

References:

- https://stackabuse.com/doubly-linked-list-with-python-examples/
- https://www.geeksforgeeks.org/doubly-linked-list/
- https://www.w3schools.com/python/python_file_handling.asp
- Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser, "Data Structures & Algorithms in Python", John Wiley & Sons Inc., 2013