**Designing a Text Editor using stack implementation:**

**Stack Implementation using Linked List:**

class Node:

    def \_\_init\_\_(self,data):

        self.data=data

        self.next=None

class StackLinkedList:

    def \_\_init\_\_(self):

        self.head=None

        self.cursor=None

        self.prev=None

        self.h=None

        self.temp=None

    def push(self,data):

        newnode=Node(data)

        if self.head is None:

            self.head=newnode

            self.cursor=newnode

        else:

            newnode.next=self.head

            self.head=newnode

    def pop(self):

        if self.head is None:

            return None

        elif self.head==self.cursor:

            self.movecursorRight()

            delval=self.head

            self.head=self.head.next

            delval.next = None

            return delval.data

        else:

            delval=self.head

            self.head=self.head.next

            delval.next = None

            return delval.data

    def movecursorLeft(self):

        if self.head is None:

            return

        elif self.head.next is None:

                self.cursor=self.head

        else:

            self.h=self.head

            while self.h.next != self.cursor.next:

                self.prev=self.h

                self.h=self.h.next

            self.cursor=self.prev

    def movecursorRight(self):

         if self.head is None:

            return

         elif self.cursor.next is None:

            return self.cursor

         else:

            self.cursor=self.cursor.next

    def getText(self):

         printval=self.head

         while printval is not None:

             print(printval.data,end="")

             printval=printval.next

         print("\n the cursor is at",self.cursor.data)

    def insertelement1(self,data):

        if self.cursor and self.head is None:

            self.push(data)

        else:

            self.y=self.cursor

            while self.head != self.cursor:

                self.x=self.pop()

                self.newnode1=Node(self.x)

                if self.temp is None:

                    self.temp=self.newnode1

                else:

                    self.newnode1.next=self.temp

                    self.temp=self.newnode1

            if self.head == self.cursor:

                self.x=self.pop()

                self.newnode1=Node(self.x)

                if self.temp is None:

                    self.temp=self.newnode1

                else:

                    self.newnode1.next=self.temp

                    self.temp=self.newnode1

            self.push(data)

            self.movecursorLeft()

            while self.temp is not None:

                self.z=self.temp.data

                self.push(self.z)

                self.temp=self.temp.next

    def delete1(self):

        if self.head is None:

            return

        else:

            while self.head!=self.cursor:

                self.x=self.pop()

                self.newnode1=Node(self.x)

                if self.temp is None:

                    self.temp=self.newnode1

                else:

                    self.newnode1.next=self.temp

                    self.temp=self.newnode1

            if self.head==self.cursor:

                if self.cursor.next is None and self.head.next is None:

                    self.head=None

                    self.cursor=None

                else:

                    if self.cursor.next is not None:

                        self.movecursorRight()

                        self.pop()

                    else:

                        self.movecursorLeft()

                        self.pop()

            while self.temp is not None:

                if self.head is None:

                    self.z=self.temp.data

                    newnode=Node(self.z)

                    newnode.next=self.head

                    self.head=newnode

                    self.cursor=newnode

                    self.temp=self.temp.next

                else:

                    self.z=self.temp.data

                    newnode=Node(self.z)

                    newnode.next=self.head

                    self.head=newnode

                    self.cursor=newnode

                    self.temp=self.temp.next

l=StackLinkedList()

l.push('a') # insert element a into the stack and cursor is at first element inserted

l.push('b') # insert element b into the stack

l.push('c') # insert element c into the stack

l.push('d') # insert element d into the stack

l.getText() # output is dcba

l.movecursorLeft() # moves the cursor left from a to b

l.movecursorLeft() # moves the cursor left from b to c

l.movecursorLeft() # moves the cursor left from c to d

l.movecursorLeft() # cursor stays at d as it is the top element of stack

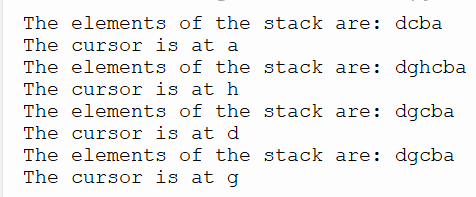
l.insertelement1('g')# insert the element at the current cursor position.

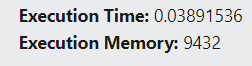
l.insertelement1('h')# insert the element at the current cursor position.

l.delete1()#delete the element at cursor position

l.getText()# prints final text dgcba

**Output:**





**Stack implementation using arrays/vector:**

class Stackarray:

    def \_\_init\_\_(self,):

        self.stack=[]

        self.stack2=[]

        self.cursor=None

        self.top=None

        self.top2=-1

    def pushelement(self,data):

        self.stack.append(data)

        self.top=len(self.stack)-1

        self.cursor=len(self.stack)-1

    def is\_empty(self):

        return len(self.stack)==0

    def peek(self):

        if self.is\_empty():

            return None

        else:

            return self.stack[-1]

    def getText(self):

        print(self.stack)

        if self.cursor is None:

            print("No elements in the stack")

        else:

            print("The cursor is at",self.stack[self.cursor])

    def movecursorLeft(self):

        if len(self.stack)==0:

            self.cursor=None

        elif len(self.stack)==1:

            self.cursor=self.cursor

        elif self.cursor<=0:

            self.cursor=0

        else:

            self.cursor=self.cursor-1

    def movecursorRight(self):

        if len(self.stack)==0:

            self.cursor=-1

        elif len(self.stack)==1:

            self.cursor=self.cursor

        elif self.cursor == len(self.stack)-1:

            self.cursor=len(self.stack)-1

        else:

            self.cursor=self.cursor + 1

    def size1(self):

        return len(self.stack)

    def insertelement(self,data):

        if self.is\_empty():

            self.pushelement(data)

        else:

            while self.top != self.cursor:

                self.stack2.append(self.stack[self.top])

                self.top2=len(self.stack2)-1

                del(self.stack[self.top])

                self.top=len(self.stack)-1

            if self.top==self.cursor:

                self.stack2.append(self.stack[self.top])

                self.top2=len(self.stack2)-1

                del(self.stack[self.top])

                self.pushelement(data)

            while self.top2>=0:

                self.stack.append(self.stack2[self.top2])

                del(self.stack2[self.top2])

                self.top2=len(self.stack2)-1

                self.top=len(self.stack)-1

    def deleteelement(self):

        if self.is\_empty():

            print("stack empty")

        else:

            while self.top != self.cursor:

                self.stack2.append(self.stack[self.top])

                self.top2=len(self.stack2)-1

                del(self.stack[self.top])

                self.top=len(self.stack)-1

            if self.top==self.cursor:

                if self.cursor==0:

                    if len(self.stack)==1:

                         del(self.stack[self.top])

                         self.top=None

                         self.cursor=None

                    else:

                         self.movecursorRight()

                         del(self.stack[self.top])

                         self.top=len(self.stack)-1

                else:

                    if self.cursor==0:

                        self.movecursorRight()

                        del(self.stack[self.top])

                        self.top=len(self.stack)-1

                    else:

                        self.movecursorLeft()

                        del(self.stack[self.top])

                        self.top=len(self.stack)-1

            while self.top2>=0:

                if len(self.stack)==0:

                    self.pushelement(self.stack2[self.top2])

                    del(self.stack2[self.top2])

                    self.top2=len(self.stack2)-1

                else:

                    self.stack.append(self.stack2[self.top2])

                    del(self.stack2[self.top2])

                    self.top2=len(self.stack2)-1

                    self.top=len(self.stack)-1

list=Stackarray()

list.pushelement('a')# pushes element a into the array

list.pushelement('b')# pushes element b into the array

list.pushelement('c')# pushes element c into the array

list.pushelement('d')# pushes element d into the array

list.getText() # prints the elements of the array

list.movecursorLeft() # moves the cursor position from d to c

list.movecursorLeft() # moves the cursor position from c to b

list.movecursorLeft() # moves the cursor position from b to a

list.movecursorLeft() # cursor stays at a

list.insertelement('h') # inserts element h between b and c

list.insertelement('g') # inserts element g between b and h

list.getText() # displays the current array

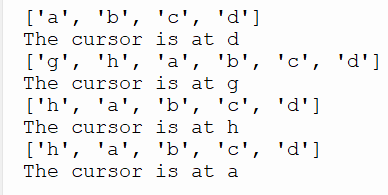
list.deleteelement()# deletes the element at the cursor position

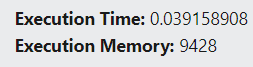
list.getText()

list.movecursorRight() # moves cursor to right position from the current cursor position

list.getText()# prints final text dgcba

**Output:**





**Stack implementation between Arrays and Linked List:**

|  |  |  |
| --- | --- | --- |
| **Time taken for:** | **Arrays** | **Linked List** |
| **Insertion** | **6.67572021484375e-06** | **2.3365020751953125e-05** |
| **Deletion** | **8.821487426757812e-06** | **1.2874603271484375e-05** |
| **moveCusorLeft** | **9.5367431640625e-07** | **2.6226043701171875e-06** |
| **moveCursorRight** | **1.1920928955078125e-06** | **2.384185791015625e-07** |
| **getText** | **4.291534423828125e-06** | **1.1444091796875e-05** |

**Time Complexity:**

|  |  |
| --- | --- |
| **Arrays** | **Linked List** |
| **Insertion: O(n)** | **Insertion: O(n)** |
| **Deletion: O(n)** | **Deletion: O(n)** |
| **moveCursorLeft: O(1)** | **moveCursorLeft:O(1)** |
| **moveCursorRight: O(1)** | **moveCursorRight: O(1)** |
| **getText:O(n)** | **getText:O(n)** |

The time complexity of both arrays and Linked list for stack implementation are almost same.

**Space Complexity:**

The space complexity of arrays is less than the space complexity of the Linked List for stack implementation

Due to additional pointer variables used in Linked List, more memory is utilized, thereby have more space complexity compared to arrays

**Insertion and deletion :**

The insertion and deletion time of data in arrays is less compared to the time taken in the Linked List implementation.

The time complexity for both arrays and Linked list is O(n). Though, arrays is preferred because less pointer variables are required and more efficient to traverse across elements.

Whereas, in linked list implementation more pointer variables are required thereby increasing space complexity.

**Random access of data:**

The random access of data in arrays is easy compared to that of Linked Lists.

In stack random access is not directly possible. It is possible only through push and pop operations.

The time taken by push and pop operations is less in case of arrays as compared to Linked Lists.

Therefore random access is best helpful in case of arrays over linked lists

**Memory Usage:**

The memory consumed by Linked List is more than that of arrays because of the many additional pointers implemented in it. These variables utilize more memory space.

Therefore, memory utilized by Linked List is more than that of arrays.

**Scalability:**

In case of arrays , the memory needs to allocated in continuous manner. Where as, in linked list memory allocation is random in the memory space and are linked through address of the memory location. Thereby making it more effective for large volumes of data.

Overall Linked lists are best preferred for larger volumes of data and arrays are best for smaller volumes of data

**Conclusion:**

**Stack implementation using arrays is more effective than that of linked list for real world applications.**

**Reason:**

The conclusion is based on factors like time and space complexity, memory usage.Arrays utilize less memory, time and space complexity is comparably less . Though Linked lists are more scalable, arrays are preferred for its above factors.