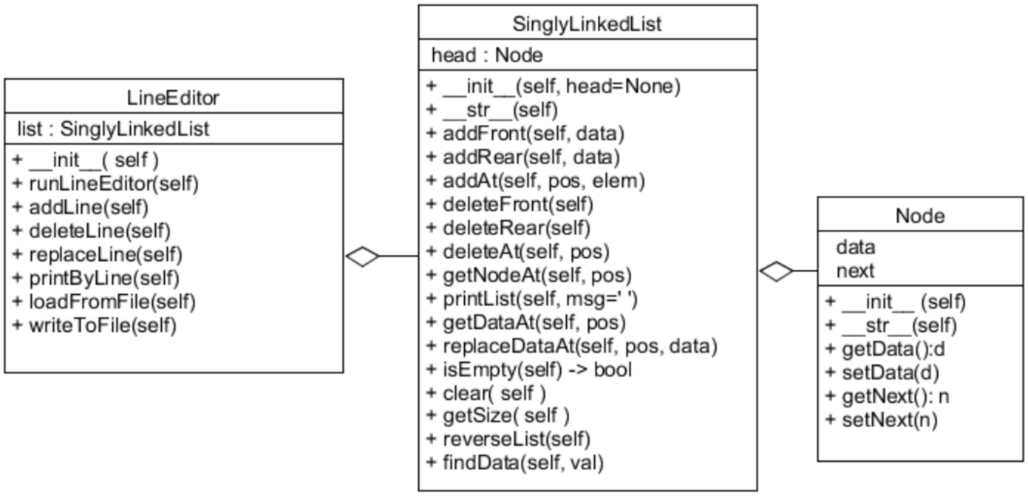
Data Structures 2023-2

Lab 05: Linked List Data Structures

Task-1: Implement Line Editor

Implement a simple text editor, where users can add, delete, and replace text lines. In addition, text lines can be written to a file or a list of text lines can be populated from a text file.



Code

Node.py

```python
class Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

    def __str__(self):
        return f"( {str(self.data)} )"

    def getNext(self):
        return self.next

    def getData(self):
        return self.data

    def setNext(self, n):
        self.next = n

    def setData(self, d):
        self.data = d
```

SinglyLinkedList.py

```python
from node import *

class SinglyLinkedList:
```

```python
def __init__(self):
    self.head = None

def addFront(self, data):
    new_node = Node(data) # create a new node
    if self.head:
        new_node.next = self.head # link new_node to heaf
    self.head = new_node # make new_node as head

def addRear(self, data):
    if self.head is None:
        self.insertFront(data) # if this is first node,

    else:
        temp = self.head
        while temp.next: # traverse to last node
            temp = temp.next
        temp.next = Node(data)

def addAt(self, pos, data):
    before = self.getNodeAt(pos-1)
    if before == None: # if it is the last node
        self.head = Node(data, self.head)
    else:
        node = Node(data, before.next)
        before.next = node
def deleteFront(self): # delete from head
    tmp = self.head
    if self.head:
        self.head = self.head.next
        tmp.next = None
    return tmp

def deleteRear(self): # delete from tail
    tmp = self.head
    if self.head:
        if self.head.next is None:
            self.head = None
        else:
            while tmp.next.next:
                tmp = tmp.next

            second_last = tmp
            tmp = tmp.next
            second_last.next = None
    return tmp

def deleteAt(self, pos):
    tmp = Node()
    if self.isEmpty() or pos > self.getSize():
        tmp = None
    elif pos == 0:
        tmp = self.deleteFront()
    elif pos == self.getSize():
        tmp = self.deleteRear()
    else:
        prev = self.getNodeAt(pos -1)
```

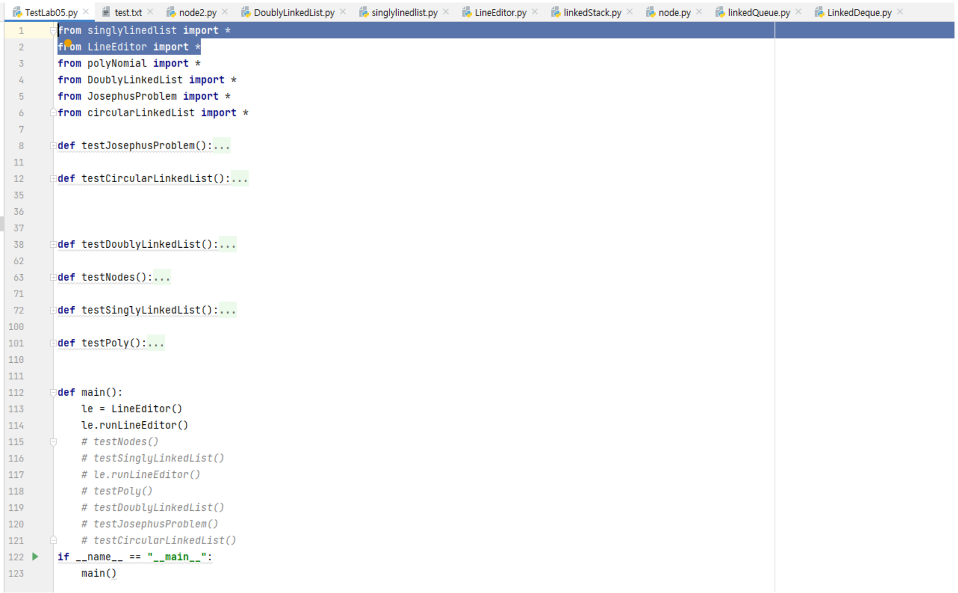
```python
            tmp = prev.next
            prev.next = tmp.next
            tmp.next = None
        return tmp

    def getNodeAt(self, pos):
        if pos < 0: return None
        node = self.head
        while pos > 0 and node != None:
            node = node.next
            pos -= 1
        return node

    # print every node data
    def printList(self, msg = "Singly Linked List : ") -> None:
        print(msg, end='')
        tmp = self.head
        while tmp:
            print(tmp.data, end = "->")
            tmp = tmp.next
        print("END")

    def __str__(self):  # String representataion
        tmp = self.head
        string_repr = ""
        while tmp:
            string_repr += str(tmp) + "->"
            tmp = tmp.next
        return string_repr + "END"

    def getDataAt(self, pos):
        node = self.getNodeAt(pos)
        if node == None:
            return None
        else:
            return node.getData()

    def replaceDataAT(self, pos, data):
        node = self.getNodeAt(pos)
        if node != None:
            node.data =data

    def isEmpty(self) -> bool:
        return self.head == None # return True if head is None

    def clear(self):
        self.head = None

    def getSize(self):
        node = self.head
        count = 0
        while node is not None:
            node = node.getNext()
            count += 1
        return count

    def reverseList(self):
        prev = None
        tmp = self.head

        while tmp:
            next_node = tmp.next # Store the current node's next node
            tmp.next = prev # Make the current node's next point backwards
            prev = tmp # Make the previous node be the current node
            tmp = next_node # Make the current node the next node(to progress iteration)
            self.head = prev # Return prev in order to put the head at the end

    def findData(self, val):
        node = self.head
        while node is not None:
            if node.data == val : return node
            node = node.next
        return node

    def printByLine(self):
        print("Line Editor")
        node = self.head
        line = 0
        while node is not None:
            #print("[%2d] "%line, end='')
            print(f"{line} = {node}")
            # print(node)
            node = node.next
            line += 1
        print()
```

LineEditor.py

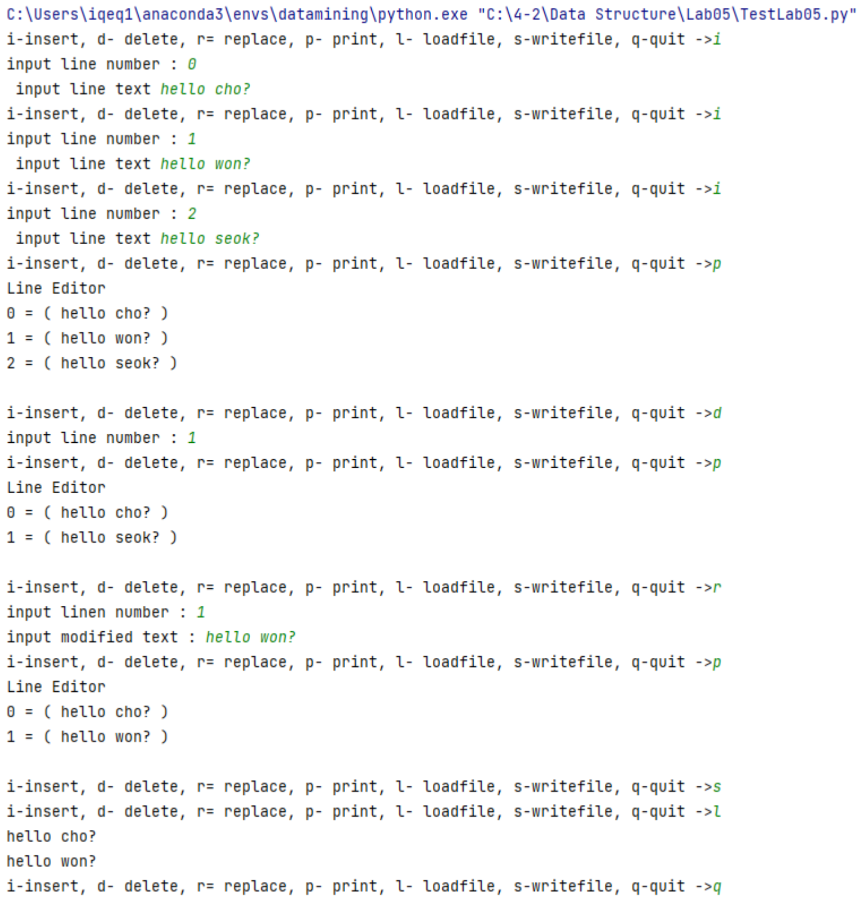
```
from singlylinedlist import *

class LineEditor:
    def __init__(self):
        self.lst = SinglyLinkedList()

    def runLineEditor(self):
        while True:
            command = input(f"i-insert, d- delete, r= replace, p- print, l-
loadfile, s-writefile, q-quit ->")
            if command == 'i' : self.addLine()
            elif command == 'd' : self.deleteLine()
            elif command == 'r' : self.replaceLine()
            elif command == 'p' : self.printByLine()
            elif command == 'l' : self.loadFromFile()
            elif command == 's' : self.writeToFile()
            elif command == 'q' : return

    def addLine(self):
        pos = int(input("input line number : "))
        _str = input(" input line text ")
        self.lst.addAt(pos, _str)

    def deleteLine(self):
        pos = int( input("input line number : "))
        self.lst.deleteAt(pos)

    def replaceLine(self):
        pos = int( input("input linen number : "))
        _str = input("input modified text : ")
        self.lst.replaceDataAT(pos, _str)

    def printByLine(self):
        self.lst.printByLine()

    def loadFromFile(self):
        # filename = input("read from file")
        filename = "test.txt"
        with open(filename, "r") as infile:
            lines = infile.readlines()
            for line in lines:
                self.lst.addAt(self.lst.getSize(), line.rstrip('\n'))
                print(line, end="")

    def writeToFile(self):
        # filename = input("write to file")
        filename = "test.txt"
        with open(filename, "w") as outfile:
            sz = self.lst.getSize()
            # print(sz)
            for i in range(sz):
                outfile.write(self.lst.getDataAt(i) + '\n')
```

Results/Output

Insert pictures for the output of the programs written for this task



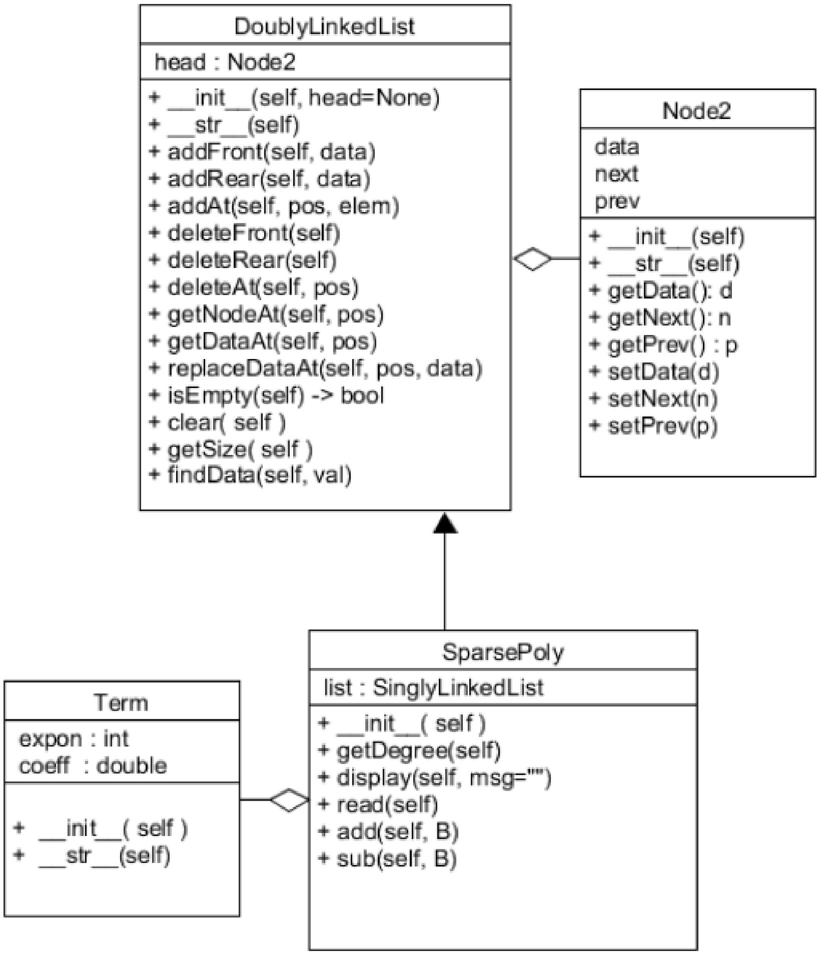
```
from singlylinedlist import *
from LineEditor import *
from polyNomial import *
from DoublyLinkedList import *
from JosephusProblem import *
from circularLinkedList import *

def testJosephusProblem():...

def testCircularLinkedList():...



def testDoublyLinkedList():...

def testNodes():...

def testSinglyLinkedList():...

def testPoly():...


def main():
    le = LineEditor()
    le.runLineEditor()
    # testNodes()
    # testSinglyLinkedList()
    # le.runLineEditor()
    # testPoly()
    # testDoublyLinkedList()
    # testJosephusProblem()
    # testCircularLinkedList()
if __name__ == "__main__":
    main()
```



```
C:\Users\iqeq1\anaconda3\envs\datamining\python.exe "C:\4-2\Data Structure\Lab05\TestLab05.py"
i-insert, d- delete, r= replace, p- print, l- loadfile, s-writefile, q-quit ->i
input line number : 0
 input line text hello cho?
i-insert, d- delete, r= replace, p- print, l- loadfile, s-writefile, q-quit ->i
input line number : 1
 input line text hello won?
i-insert, d- delete, r= replace, p- print, l- loadfile, s-writefile, q-quit ->i
input line number : 2
 input line text hello seok?
i-insert, d- delete, r= replace, p- print, l- loadfile, s-writefile, q-quit ->p
Line Editor
0 = ( hello cho? )
1 = ( hello won? )
2 = ( hello seok? )

i-insert, d- delete, r= replace, p- print, l- loadfile, s-writefile, q-quit ->d
input line number : 1
i-insert, d- delete, r= replace, p- print, l- loadfile, s-writefile, q-quit ->p
Line Editor
0 = ( hello cho? )
1 = ( hello seok? )

i-insert, d- delete, r= replace, p- print, l- loadfile, s-writefile, q-quit ->r
input linen number : 1
input modified text : hello won?
i-insert, d- delete, r= replace, p- print, l- loadfile, s-writefile, q-quit ->p
Line Editor
0 = ( hello cho? )
1 = ( hello won? )

i-insert, d- delete, r= replace, p- print, l- loadfile, s-writefile, q-quit ->s
i-insert, d- delete, r= replace, p- print, l- loadfile, s-writefile, q-quit ->l
hello cho?
hello won?
i-insert, d- delete, r= replace, p- print, l- loadfile, s-writefile, q-quit ->q
```

Task2: Implement Sparse Polynomial

Sparse polynomials have many missing terms. In other words, coefficients for many terms are zero. Linked list data structures are better for this kind of polynomial. Implement the sparse polynomial using a list data structure



Code

Node2.py

```
class Node2:
    def __init__(self, prev = None, data = None,nxt = None):
        self.prev = prev
        self.data = data
        self.next = nxt

    def __str__(self):
        return str(self.data)

    def getNext(self):
        return self.next

    def getPrev(self):
        return self.prev

    def setData(self, d):
        self.data = d

    def setNext(self, nn):
        self.next = nn

    def setPrev(self, pn):
        self.prev = pn
```

DoublyLinkedList.py

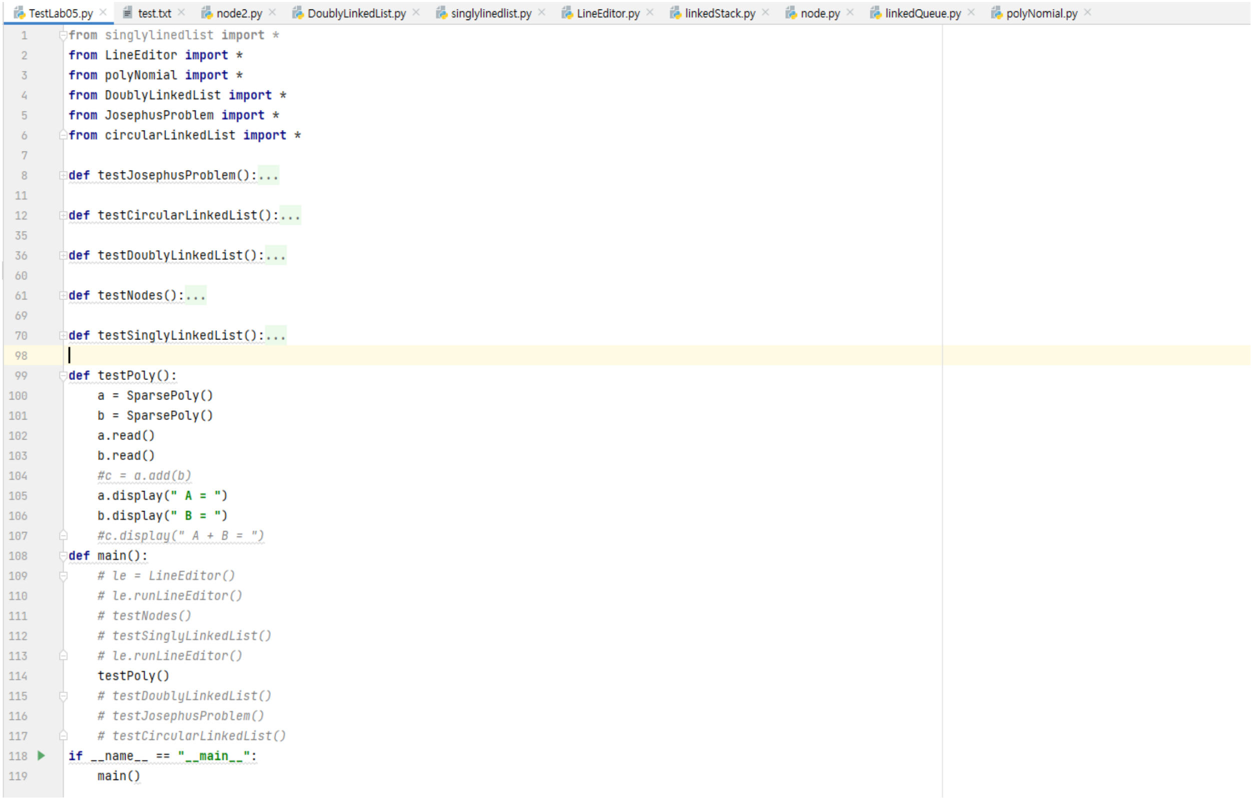
```
from node2 import *

class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def __str__(self): #string representation
        temp = self.head
        string_repr = ""
        while temp:
            string_repr += str(temp) + "->"
            temp = temp.next

        return string_repr + "END"

    def isEmpty(self): return self.head == None
    def clear(self): self.head = None

    def deleteFront(self):
        if self.isEmpty():
            print("List is Empty..")
            return None

        temp = self.head
        if temp.next == temp.prev:
            self.head = None
            return temp
        else:
            self.head = temp.next
            self.head.prev = None
            return temp
    def deleteRear(self):
        if self.isEmpty():
            print("List is Empty..")
            return None

        temp = self.head
        if temp.next == temp.prev:
            self.head = None
            return temp
        else:
            while temp.next:
                temp = temp.next

            temp.prev.next = None
            temp.prev = None
            return temp

    def deleteAt(self, pos):
        temp = Node2()
        if pos == self.getSize():
            temp = self.deleteRear()
        elif pos == 0:
            temp = self.deleteFront()
        else:
            before = self.getNodeAt(pos-1)
            if before is None:
                print("This node doesn't exit in DLL")
                return

            temp = before.next
            before.next = temp.next
            temp.next.prev = before
            temp.next = None
            temp.prev = None
        return temp

    def addFront(self, data):
        newNode = Node2(None, data, None)
        if self.head is None:
            self.head = newNode
        else:
            newNode.next = self.head
            if self.head is not None:
                self.head.prev = newNode

            self.head = newNode

    def addRear(self, data):
        newNode = Node2(None, data, None)
        if self.head is None:
            self.head = newNode
            return
        temp = self.head
        while temp.next is not None:
            temp = temp.next
        temp.next = newNode
        newNode.prev = temp

    def addAt(self, pos, data):
        if pos == self.getSize():
            self.addRear(data)
            return
        if pos == 0:
            self.addFront(data)
            return

        newNode = Node2(None, data, None)
        before = self.getNodeAt(pos-1)
        if before is None:
            print("This node doesn't exist in DLL")
            return

        newNode.next = before.next
        before.next = newNode
        newNode.prev = before
        if newNode.next is not None:
            newNode.next.prev = newNode

    def getNodeAt(self, pos):
        if pos < 0 or pos > self.getSize():
            print("Invalid position")
            return None
        temp = self.head
        while pos > 0 and temp != None:
            temp = temp.next
            pos -= 1
        return temp

    def printList(self, msg = "Doubly Linked List : "): # print every ndoe
data
        print(msg, end = "")
        tmp = self.head
        while tmp:
            print(tmp, end="->")
            tmp = tmp.next
        print("END")

    def getSize(self):
        node = self.head
        count = 0
        while node is not None:
            node = node.next
            count += 1
        return count

    def getDataAt(self, pos):
        node = self.getNodeAt(pos)
        if node == None:
            return None
        else:
            return node.data

    def replaceDataAt(self, pos, data):
        node = self.getNodeAt(pos)
        if node != None:
            node.data = data
```

polyNormal.py : term and SparsePoly class

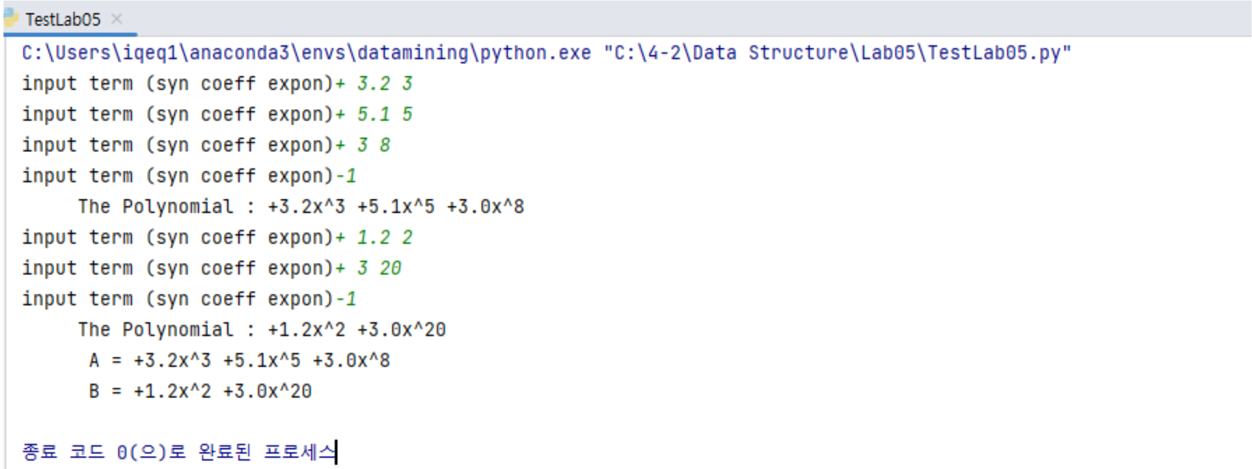
```
from DoublyLinkedList import *
class Term:
    def __init__(self, sgn=None, coeff=None, expon=None):
        self.sgn = sgn
        self.coeff = coeff
        self.expon = expon

    def __str__(self):
        return str(self.sgn) + str(self.coeff) + "x^"+ str(self.expon) + " "

    def getCoeff(self):
        return self.coeff

    def getExpon(self):
        return self.expon

    def getSyn(self):
        return self.sgn

class SparsePoly(DoublyLinkedList):
    def __init__(self):
        super().__init__()

    def display(self, msg=""):
        print("\t", msg, end='')

        node = self.head
        while node is not None:
            print(node, end='')
            node = node.getNext()
        print()

    def read(self):
        self.clear()
        while True:
            token = input("input term (syn coeff expon)").split(" ")
            if token[0] == '-1':
                self.display("The Polynomial : ")
                return
            self.addAt(self.getSize(), Term(token[0], float(token[1]),
int(token[2])))

    def add(self, B):
        C = SparsePoly()
        a = self.head
        b = B.head

    def sub(self, B):
        pass

    def getDegree(self):
        pass
```

Results/Output

Insert pictures for the output of the programs written for this task



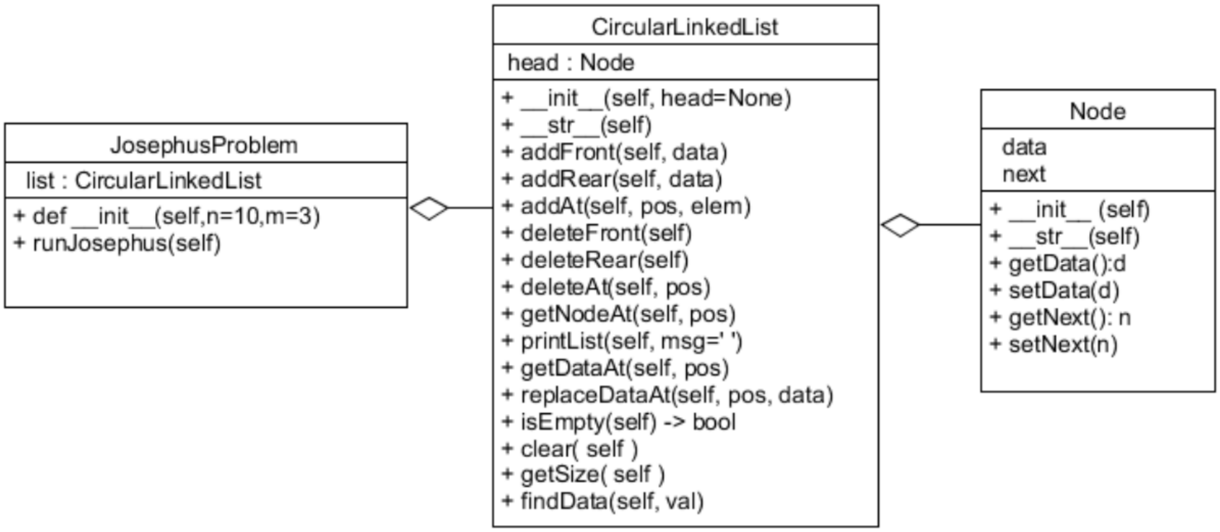
```
from singlylinedlist import *
from LineEditor import *
from polyNomial import *
from DoublyLinkedList import *
from JosephusProblem import *
from circularLinkedList import *

def testJosephusProblem():...

def testCircularLinkedList():...

def testDoublyLinkedList():...

def testNodes():...

def testSinglyLinkedList():...

def testPoly():
    a = SparsePoly()
    b = SparsePoly()
    a.read()
    b.read()
    #c = a.add(b)
    a.display(" A = ")
    b.display(" B = ")
    #c.display(" A + B = ")
def main():
    # le = LineEditor()
    # le.runLineEditor()
    # testNodes()
    # testSinglyLinkedList()
    # le.runLineEditor()
    testPoly()
    # testDoublyLinkedList()
    # testJosephusProblem()
    # testCircularLinkedList()
if __name__ == "__main__":
    main()
```



```
C:\Users\iqeq1\anaconda3\envs\datamining\python.exe "C:\4-2\Data Structure\Lab05\TestLab05.py"
input term (syn coeff expon)+ 3.2 3
input term (syn coeff expon)+ 5.1 5
input term (syn coeff expon)+ 3 8
input term (syn coeff expon)-1
	The Polynomial : +3.2x^3 +5.1x^5 +3.0x^8
input term (syn coeff expon)+ 1.2 2
input term (syn coeff expon)+ 3 20
input term (syn coeff expon)-1
	The Polynomial : +1.2x^2 +3.0x^20
	 A = +3.2x^3 +5.1x^5 +3.0x^8
	 B = +1.2x^2 +3.0x^20

종료 코드 0(으)로 완료된 프로세스
```

Task-3: Solve Josephus Problem

- There are n people (n is an even number) standing in a circle waiting to be executed.
- The counting out begins at some point in the circle and proceeds around the circle in a fixed direction.
- In each step, a certain number of people m (m is an odd number) are skipped, and the next person is executed.
- The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom.



Code

Node.py

```
class Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

    def __str__(self):
        return f"( {str(self.data)} )"

    def getNext(self):
        return self.next

    def getData(self):
        return self.data

    def setNext(self, n):
        self.next = n

    def setData(self, d):
        self.data = d
```

CircularLinkedList.py

```
from node import *
class CircularLinkedList:
    def __init__(self):
        self.head = None

    def isEmpty(self): return self.head == None
    def clear(self): self.head = None

    def addFront(self, data):
        newNode = Node(data)
        if self.head is None:
            self.head = newNode
            newNode.next = self.head
            self.head.next = self.head
        else:
            newNode.next = self.head.next
            self.head.next = newNode

    def addRear(self, data):
        newNode = Node(data)

        if self.head is None:
            self.head = newNode
            newNode.next = self.head
        else:
            newNode.next = self.head.next
            self.head.next = newNode
            self.head= newNode

    def addAt(self, pos, data):
        if pos == self.getSize():
            self.addRear(data)
            return
        newNode = Node(data)
        before = self.getNodeAt(pos-1)
        if before is None:
            print("This node doesn't exist in CLL")
            return

        if before is None:
            print("This node doesn't exist in CLL")
            return

        newNode.next = before.next
        before.next = newNode

    def deleteFront(self):
        if self.isEmpty():
            print("List is Empty..")
            return None

        temp = self.head

        if temp == temp.next:
            self.head = None
            return temp
        else:
            temp=self.head.next
            self.head.next = temp.next
            temp.next = None
            return temp
    def deleteRear(self):
        temp = self.head
        if self.isEmpty():
            print("List is Empty..")
            return None
        if self.head == self.head.next:
            self.head = None
            return temp
        else:
            before = self.getNodeAt(self.getSize() - 2)
            self.head = before
            self.head.next = temp.next
            temp.next = None
            return temp
    def deleteAt(self, pos):
        temp = Node()
        if pos == self.getSize() -1:
            temp = self.deleteRear()
        elif pos == 0:
            temp = self.deleteFront()
        else:
            before = self.getNodeAt(pos-1)
            if before is None:
                print("This node doesn't exist in DLL")
                return

            temp = before.next
            before.next = temp.next
            temp.next = None
        return temp

    def getNodeAt(self, pos):
        if pos < 0 or pos > self.getSize(): return None
        temp = self.head
        if self.head is not None:
            while True:
                temp = temp.next
                pos -= 1
                if pos < 0:
                    break
        return temp

    def printList(self, msg='CircularlySingly Linked List : '):
        # print every node data
        temp = self.head

        if self.head is not None:
            while True:
                print(temp, end="->")
                temp = temp.next
```

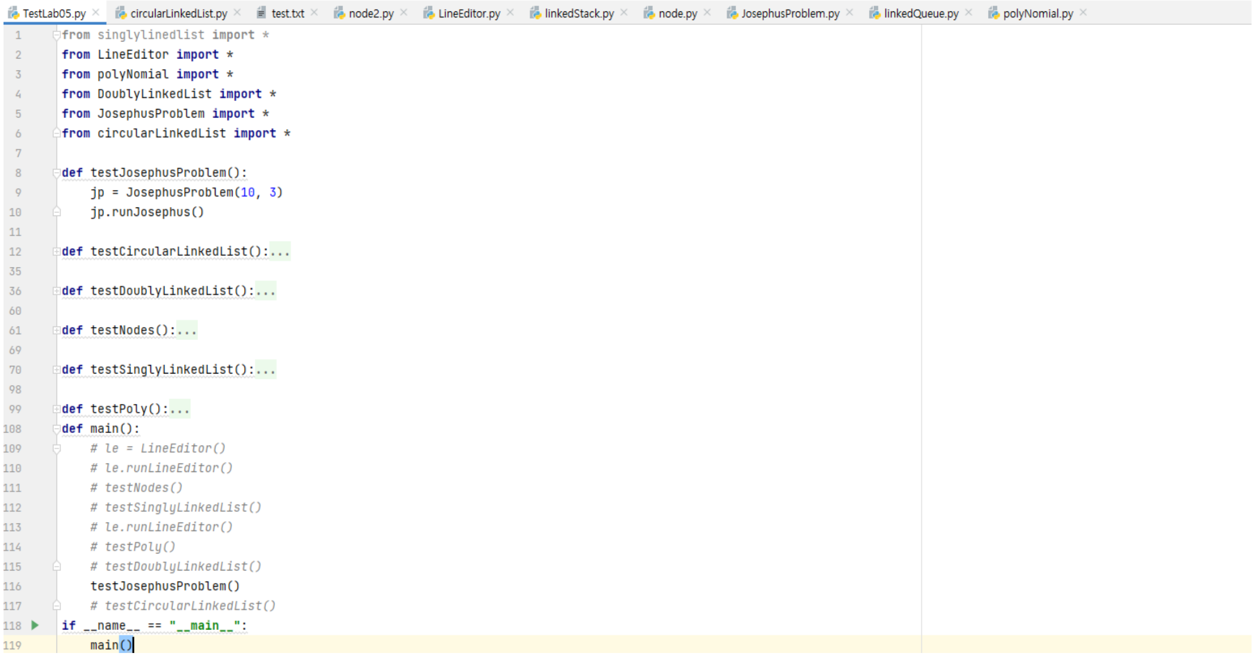
```
def __str__(self):
    temp = self.head
    string_repr = ""
    if self.head is not None:
        while True:
            string_repr += str(temp) + "->"
            temp = temp.next
            if temp == self.head:
                break
    return string_repr

def getSize(self):
    temp = self.head
    count = 0
    if self.head is not None:
        while True:
            count += 1
            temp = temp.next
            if temp == self.head:
                break
    return count

def findPos(self, node):
    temp = self.head.next

    pos = 0
    while True:
        temp = temp.next
        pos += 1
        if temp.data == node.data or pos > self.getSize() +1:
            break
    return pos%self.getSize()
```

JosephusProblem.py

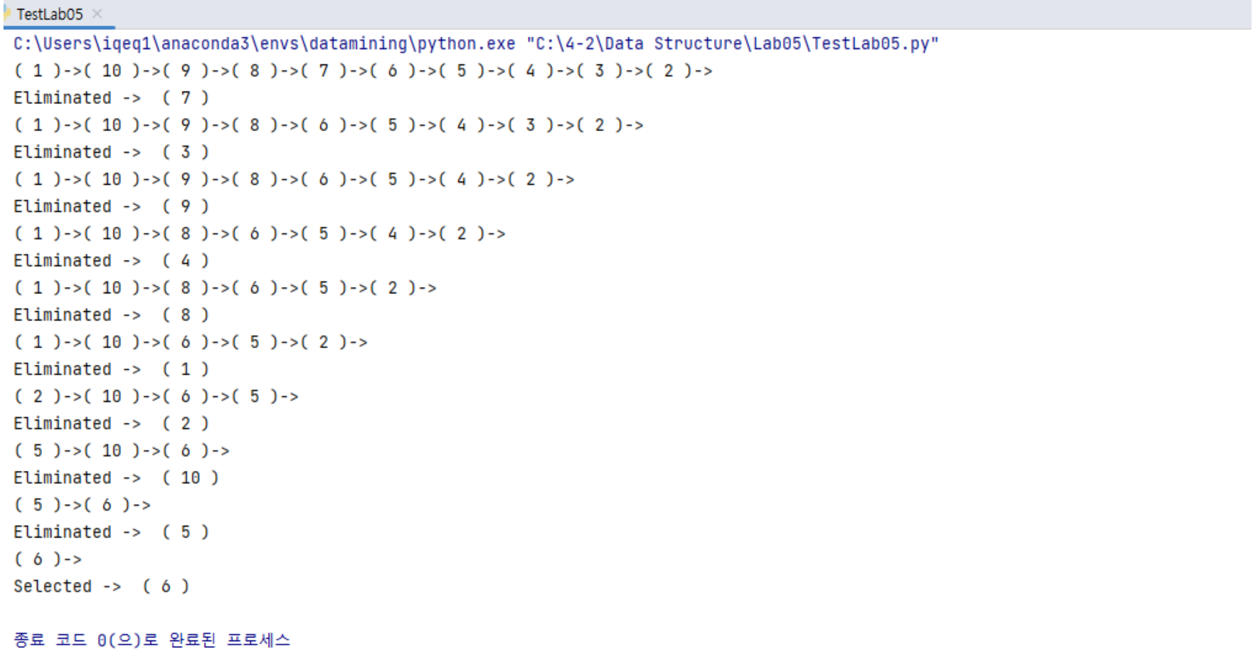
```python
from circularLinkedList import *
class JosephusProblem:
    def __init__(self, n = 10, m = 3):
        self.lst = CircularLinkedList()
        self.n = n
        self.m = m
        for i in range(1, n + 1):
            self.lst.addFront(i)

    def runJosephus(self):
        print(self.lst)
        temp = self.lst.head.next
        count = 0
        while True:
            temp = temp.next
            count += 1

            if count == self.m:
                temp2 = temp.next
                pos = self.lst.findPos(temp)
                print("Eliminated -> ", self.lst.deleteAt(pos))
                temp = temp2
                print(self.lst)

                count = 0
            if temp == temp.next:
                print("Selected -> ", temp)
                break
```

Results/Output

Insert pictures for the output of the programs written for this task



```python
from singlylinedlist import *
from LineEditor import *
from polyNomial import *
from DoublyLinkedList import *
from JosephusProblem import *
from circularLinkedList import *

def testJosephusProblem():
    jp = JosephusProblem(10, 3)
    jp.runJosephus()

def testCircularLinkedList():...

def testDoublyLinkedList():...

def testNodes():...

def testSinglyLinkedList():...

def testPoly():...
def main():
    # le = LineEditor()
    # le.runLineEditor()
    # testNodes()
    # testSinglyLinkedList()
    # le.runLineEditor()
    # testPoly()
    # testDoublyLinkedList()
    testJosephusProblem()
    # testCircularLinkedList()
if __name__ == "__main__":
    main()
```



```
C:\Users\iqeq1\anaconda3\envs\datamining\python.exe "C:\4-2\Data Structure\Lab05\TestLab05.py"
( 1 )->( 10 )->( 9 )->( 8 )->( 7 )->( 6 )->( 5 )->( 4 )->( 3 )->( 2 )->
Eliminated ->  ( 7 )
( 1 )->( 10 )->( 9 )->( 8 )->( 6 )->( 5 )->( 4 )->( 3 )->( 2 )->
Eliminated ->  ( 3 )
( 1 )->( 10 )->( 9 )->( 8 )->( 6 )->( 5 )->( 4 )->( 2 )->
Eliminated ->  ( 9 )
( 1 )->( 10 )->( 8 )->( 6 )->( 5 )->( 4 )->( 2 )->
Eliminated ->  ( 4 )
( 1 )->( 10 )->( 8 )->( 6 )->( 5 )->( 2 )->
Eliminated ->  ( 8 )
( 1 )->( 10 )->( 6 )->( 5 )->( 2 )->
Eliminated ->  ( 1 )
( 2 )->( 10 )->( 6 )->( 5 )->
Eliminated ->  ( 2 )
( 5 )->( 10 )->( 6 )->
Eliminated ->  ( 10 )
( 5 )->( 6 )->
Eliminated ->  ( 5 )
( 6 )->
Selected ->  ( 6 )

종료 코드 0(으)로 완료된 프로세스
```

Conclusion

Thank you