Lab-VIII

Hasan Amin

(374866)

CS-250

Data Structures and Algorithms

School of Natural Sciences

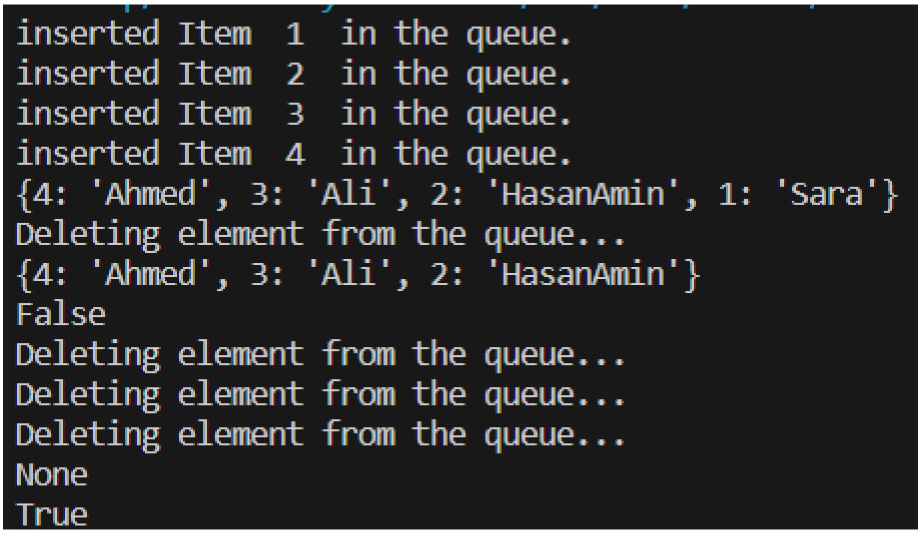
Contents

Task 1:

Code

```
from LQueue import LQueue

class Waiting:
    def __init__(self,MaxSize):
        self.maxsize=MaxSize
        self.queue=LQueue(self.maxsize)
        self.record={}
    def RegisterPatient(self,name):
        if self.queue.isEmpty():
            id=1

        else:
            id=self.queue.peek()+len(self.queue)
        self.queue.insert(id)
        self.record[id]=name
        return id
    def ServePatient(self):
        ServedPatient=self.queue.remove()
        self.record.pop(ServedPatient)
        return self.queue.peek(),self.record[self.queue.peek()] #returning
the id and name of next patient in the queue
    def CancelAll(self):
        while not self.queue.isEmpty():
            self.queue.remove()
    def CanDoctorGoHome(self):
        return self.queue.isEmpty()

    def ShowAllPatients(self):
        sortedDict=dict(sorted(self.record.items(), key=lambda x:x[1]))
        return sortedDict

if __name__=="__main__":
    room= Waiting(5)
    room.RegisterPatient("Sara")
    room.RegisterPatient("HasanAmin")
    room.RegisterPatient("Ali")
    room.RegisterPatient("Ahmed")
    print(room.ShowAllPatients())
    room.ServePatient()
    print(room.ShowAllPatients())
    print(room.CanDoctorGoHome())
    print(room.CancelAll())
    print(room.CanDoctorGoHome())
    # Working Fine
```

Output



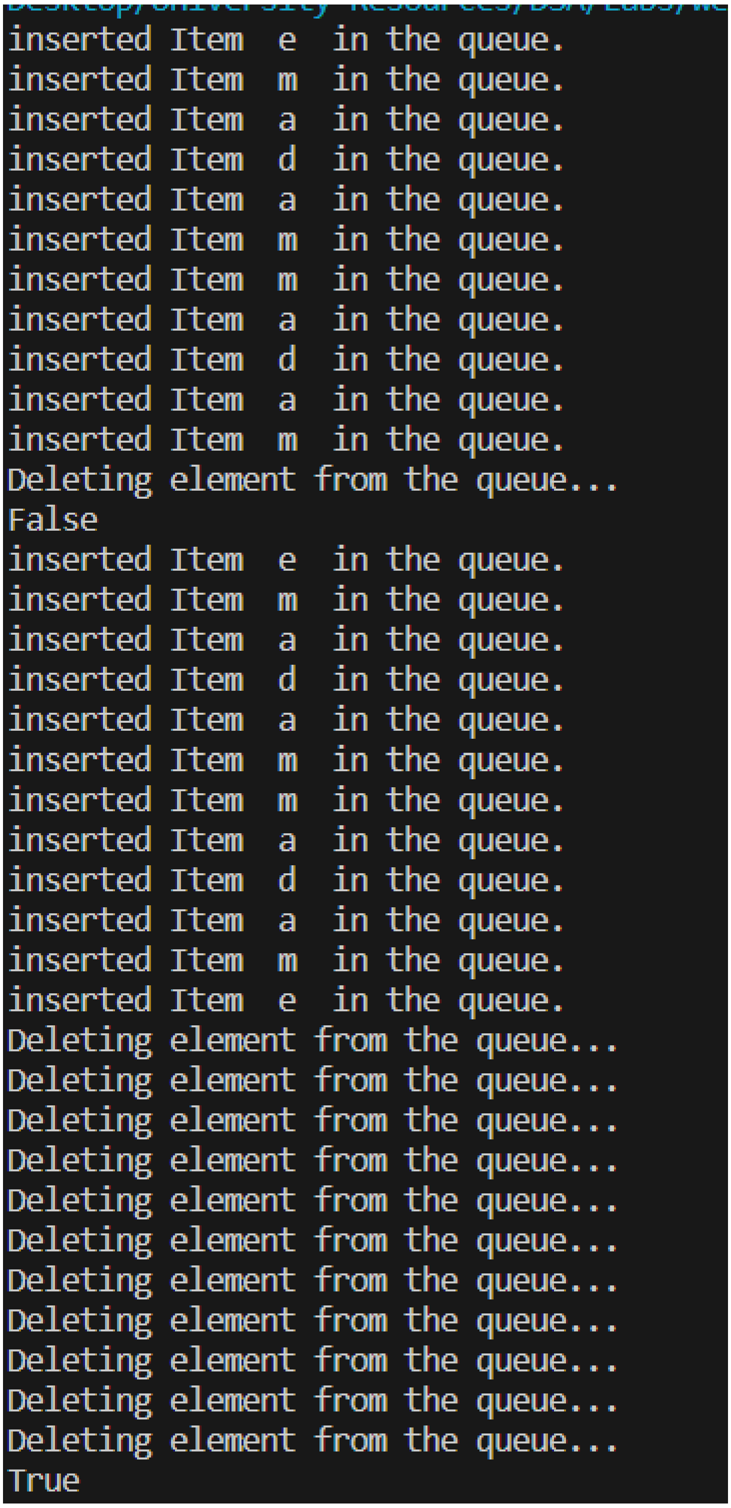
```
inserted Item  1  in the queue.
inserted Item  2  in the queue.
inserted Item  3  in the queue.
inserted Item  4  in the queue.
{4: 'Ahmed', 3: 'Ali', 2: 'HasanAmin', 1: 'Sara'}
Deleting element from the queue...
{4: 'Ahmed', 3: 'Ali', 2: 'HasanAmin'}
False
Deleting element from the queue...
Deleting element from the queue...
Deleting element from the queue...
None
True
```

Task 2:

Code

```python
from LQueue import LQueue
from Stack import Stack
def palindrome_check(string:str):
    myQueue=LQueue(len(string))
    myStack=Stack()
    for i in string:
        myQueue.insert(i.lower())
        myStack.push(i.lower())
    for i in range(len(string)):
        if len(myQueue)>1:
            front=myQueue.remove()
            rear=myStack.pop()
            if front==rear:
                continue
            else:return False
        else:
            break
    return True





print(palindrome_check("eMadammAdam"))
print(palindrome_check("eMadammAdame"))
```

Output

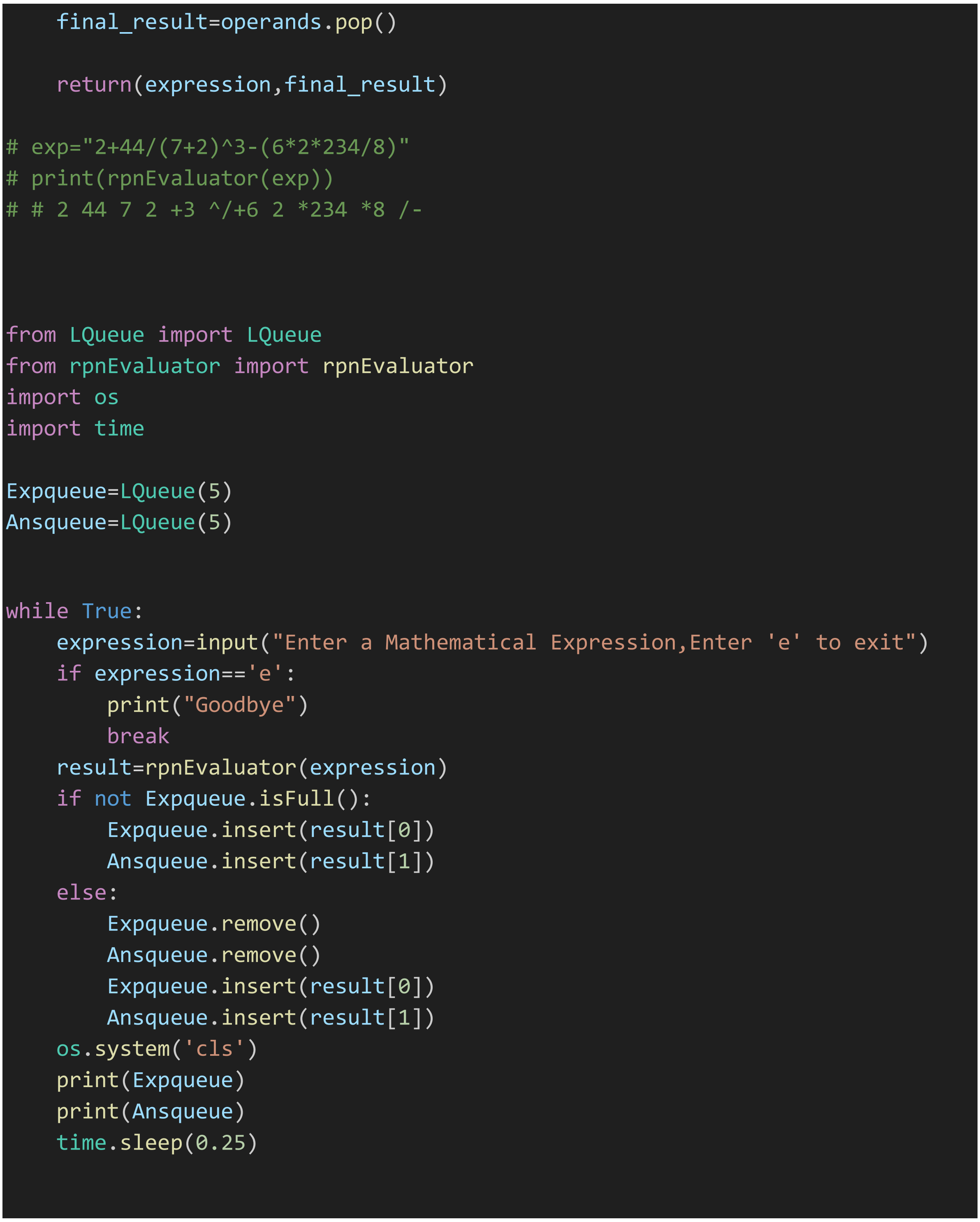


```
Desktop/University Resources/DSA/Labs/We
inserted Item  e  in the queue.
inserted Item  m  in the queue.
inserted Item  a  in the queue.
inserted Item  d  in the queue.
inserted Item  a  in the queue.
inserted Item  m  in the queue.
inserted Item  m  in the queue.
inserted Item  a  in the queue.
inserted Item  d  in the queue.
inserted Item  a  in the queue.
inserted Item  m  in the queue.
Deleting element from the queue...
False
inserted Item  e  in the queue.
inserted Item  m  in the queue.
inserted Item  a  in the queue.
inserted Item  d  in the queue.
inserted Item  a  in the queue.
inserted Item  m  in the queue.
inserted Item  m  in the queue.
inserted Item  a  in the queue.
inserted Item  d  in the queue.
inserted Item  a  in the queue.
inserted Item  m  in the queue.
inserted Item  e  in the queue.
Deleting element from the queue...
Deleting element from the queue...
Deleting element from the queue...
Deleting element from the queue...
Deleting element from the queue...
Deleting element from the queue...
Deleting element from the queue...
Deleting element from the queue...
Deleting element from the queue...
Deleting element from the queue...
Deleting element from the queue...
True
```

Task 3:

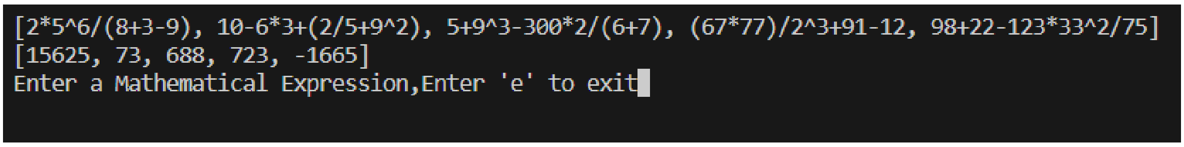
Code

```
from Stack import Stack
from BalancedExpression import is_balanced



def RPN(expression):
    if not is_balanced(expression=expression): #from previous lab
        print("Invalid Expression")
        return False
    stack=Stack()
    precedence = {'^': 3, '*': 2, '/': 2, '+': 1, '-': 1}
    bracket_pairs = {')': '(', ']': '[', '}': '{'}
    RPNstring=''
    for index,i in enumerate(expression):
        if i in '[{(':
            stack.push(i)
        elif i in '^*/+-':
            while (
                stack.peek() in precedence
                and precedence.get(stack.peek(), 0) >= precedence.get(i, 0)
            ):
                RPNstring += stack.pop()

            stack.push(i)
        elif i in ']})':
            while stack.peek() !=  bracket_pairs[i]:
                RPNstring+=stack.pop()
            stack.pop()
        else:
            if not expression[index]==expression[-1]: # only if the
character is not the last character of the expression to avoid index out of
range error when checking for index+1
                if expression[index+1] in '0123456789.':
                    RPNstring+=i
                else:
                    RPNstring=RPNstring  + i+" "
            else:
                RPNstring=RPNstring  + i+" "

    while not stack.is_empty():
        RPNstring+=stack.pop()
    return RPNstring

# print(RPN("(3+5)*(7/2)-4"))


from ReversePolish import RPN
from Stack import Stack

# this function uses floor division inplace of true division
def rpnEvaluator(expression:str):
    rpn_string=RPN(expression=expression)
    print(rpn_string)
    operands=Stack()
    digits="0123456789"
    operators="*^+-/"
    i=0
    while i<=len(rpn_string)-1:
        # print(operands)
        # print(rpn_string[i])
        char=rpn_string[i]
        if char==" ":
            i+=1
            continue
        while rpn_string[i] in digits and rpn_string[i+1] in digits: #
accumulating digits of numerals with number of digits>1
            char+=rpn_string[i+1]
            i+=1
        if char in operators:
            i+=1
            print(operands)
            SecondOperand=operands.pop()
            FirstOperand=operands.pop()
            if char =="+":
                result=int(FirstOperand)+int(SecondOperand)
            elif char =="-":
                result=int(FirstOperand)-int(SecondOperand)
            elif char =="*":
                result=int(FirstOperand)*int(SecondOperand)
            elif char =="^":
                result=int(FirstOperand)**int(SecondOperand)
            else:
                if SecondOperand!="0":
                    result=int(FirstOperand)//int(SecondOperand)
                else:raise ZeroDivisionError("Cannot divide by zero")
            operands.push(result)
        else: # only numerals left
            i+=1
            operands.push(int(char))
```



```
    final_result=operands.pop()

    return(expression,final_result)

# exp="2+44/(7+2)^3-(6*2*234/8)"
# print(rpnEvaluator(exp))
# # 2 44 7 2 +3 ^/+6 2 *234 *8 /-



from LQueue import LQueue
from rpnEvaluator import rpnEvaluator
import os
import time

Expqueue=LQueue(5)
Ansqueue=LQueue(5)


while True:
    expression=input("Enter a Mathematical Expression,Enter 'e' to exit")
    if expression=='e':
        print("Goodbye")
        break
    result=rpnEvaluator(expression)
    if not Expqueue.isFull():
        Expqueue.insert(result[0])
        Ansqueue.insert(result[1])
    else:
        Expqueue.remove()
        Ansqueue.remove()
        Expqueue.insert(result[0])
        Ansqueue.insert(result[1])
    os.system('cls')
    print(Expqueue)
    print(Ansqueue)
    time.sleep(0.25)
```

Output



```
[2*5^6/(8+3-9), 10-6*3+(2/5+9^2), 5+9^3-300*2/(6+7), (67*77)/2^3+91-12, 98+22-123*33^2/75]
[15625, 73, 688, 723, -1665]
Enter a Mathematical Expression,Enter 'e' to exit
```