

Lab-VIII

Hasan Amin

(374866)

CS-250

Data Structures and Algorithms

School of Natural Sciences

Contents

[Task 1: 3](#_Toc150812490)

[Task 2: 5](#_Toc150812491)

[Task 3: 7](#_Toc150812492)

# 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** |
|  |

# Task 2:

|  |
| --- |
| **Code** |
| 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** |
|  |

# 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** |
|  |
|  |