DATA-STRUCTURES: LISTS, STACK AND QUEUE

PROGRAMS

#Evaluating postfix expression

```
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()
    def is_empty(self):
        return (self.items == [])

def eval_postfix(expr):
    import re
    token_list = re.split("([^0-9])", expr)
    stack = Stack()
    for token in token_list:
        if token == "" or token == " ":
            continue
        if token == "+":
            sum = stack.pop() + stack.pop()
            stack.push(sum)
        elif token == "*":
            product = stack.pop() * stack.pop()
            stack.push(product)
        else:
            stack.push(int(token))
    return stack.pop()

res=eval_postfix("5 7 + 2 *")
print(res)
```

PYTHON.MYKVS.IN

#Infix to postfix conversion

```
class Stack:
    def __init__(self):
        self.items = []
        self.length = 0
    def push(self, val):
        self.items.append(val)
        self.length += 1
    def pop(self):
        if self.empty():
            return None
        self.length -= 1
        return self.items.pop()
    def size(self):
        return self.length
    def peek(self):
        if self.empty():
            return None
        return self.items[0]
    def empty(self):
        return self.length == 0
    def __str__(self):
        return str(self.items)
precedence = {}
precedence['*'] = 3
precedence['/'] = 3
precedence['+'] = 2
precedence['-'] = 2
precedence['('] = 1

def convert(expression):
    print(__convert(expression.split()))
def __convert(tokens):
    postfix = []
```

PYTHON.MYKVS.IN

```
    opstack = Stack()

    for token in tokens:
        if token.isidentifier():
            postfix.append(token)
        elif token == '(':
            opstack.push(token)
        elif token == ')':
            while True:
                temp = opstack.pop()
                if temp is None or temp == '(':
                    break
                elif not temp.isidentifier():
                    postfix.append(temp)

        else: # must be operator
            if not opstack.empty():
                temp = opstack.peek()

                while not opstack.empty() and precedence[temp] >= precedence[token] and
token.isidentifier():
                    postfix.append(opstack.pop())
                    temp = opstack.peek()

            opstack.push(token)

    while not opstack.empty():
        postfix.append(opstack.pop())

    return postfix

convert("A + B")
convert("A + B * C")
convert("A * ( B + C ) + D")
```

PYTHON.MYKVS.IN

#Reverse a string using stack

```
def createStack():
    stack=[]
    return stack

def size(stack):
    return len(stack)

def isEmpty(stack):
    if size(stack) == 0:
        return true

def push(stack,item):
    stack.append(item)

def pop(stack):
    if isEmpty(stack): return
    return stack.pop()

def reverse(string):
    n = len(string)
    stack = createStack()
    for i in range(0,n,1):
        push(stack,string[i])
    string=""
    for i in range(0,n,1):
        string+=pop(stack)
    return string

string="python.mykvs.in"
string = reverse(string)
print("Reversed string is " + string)
```

PYTHON.MYKVS.IN

```
# Python3 program to reverse a queue
from queue import Queue


def Print(queue):
    while (not queue.empty()):
        print(queue.queue[0], end = ",")
        queue.get()

def reversequeue(queue):
    Stack = []
    while (not queue.empty()):
        Stack.append(queue.queue[0])
        queue.get()
    while (len(Stack) != 0):
        queue.put(Stack[-1])
        Stack.pop()

if __name__ == '__main__':
    queue = Queue()
    queue.put(10)
    queue.put(20)
    queue.put(30)
    queue.put(40)
    queue.put(50)
    queue.put(60)
    queue.put(70)
    queue.put(80)
    queue.put(90)
    queue.put(100)

    reversequeue(queue)
    Print(queue)
```

PYTHON.MYKVS.IN

#Create a Queue with 2 Stacks

```
class Queue2Stacks(object):
    def __init__(self):
        self.stack1 = []
        self.stack2 = []

    def enqueue(self, item):
        self.stack1.append(item)

    def dequeue(self):
        #check if the stack2 is empty
        if not self.stack2:
            while len(self.stack1) > 0: # = while self.stack1:
                self.stack2.append(self.stack1.pop())
        #once it is not empty then we can return the elements
        return self.stack2.pop()

q = Queue2Stacks()
for i in range(5):
    q.enqueue(i)

for i in range(5):
    print (q.dequeue())
```

PYTHON.MYKVS.IN