#### 1 !date

```
Fri Mar 11 05:10:43 UTC 2022
```

Please run the above line to refresh the date before your submission.

### CSCI-SHU 210 Data Structures

### Recitation 5 Stacks and Queues

Name: Peter Yao

NetID: yy4108

- For students who have recitation on Wednesday, you should submit your solutions by Friday Mar 11 11:59pm.
- For students who have recitation on Thursday, you should submit your solutions by Saturday Mar 12 11:59pm.
- For students who have recitation on Friday, you should submit your solutions by Sunday Mar 13 11:59pm.

No late submission is permitted. All solutions must be from your own work. Total points of the assignment is 100.

## ▼ Bad Queue Example

```
1 class ArrayQueue():
2
3
      DEFAULT CAPACITY = 10
 4
5
      def init (self):
          self. data = [None for i in range(ArrayQueue.DE
 6
          self.first idx=0
 7
           self.n=0
8
9
      def len (self):
10
          # return len(self. data)
11
          return self.n
12
13
14
      def is empty(self):
          # return len(self. data) == 0
15
```

```
16
           return self.n==0
17
18
      def first(self):
          # return self. data[0]
19
          return self. data[self.first idx]
20
21
22
23
      def dequeue(self):
          temp= self. data[self.first idx]
24
25
           self. data[self.first idx]=None
          self.first idx=(self.first idx+1)%ArrayQueue.DE
26
27
           self.n=1
           return temp
28
29
30
31
      def enqueue(self, e):
          # self. data.append(e)
32
33
           self.n+=1
          self. data[(self.first idx+self.n-1)%ArrayQueue
34
35
36
37
      def str (self):
           ''' You can simply print self. data '''
38
39
           return str(self. data)
40
          # print(self. data)
41
42 def main():
      # Empty Queue, size 10.
43
44
      queue = ArrayQueue()
45
46
      # Engueue 0, 1, 2, 3, 4, 5, 6, 7
47
      for i in range(8):
48
           queue.enqueue(i)
      print(queue) # [0, 1, 2, 3, 4, 5, 6, 7, None, Non
49
50
51
      # Dequeue 5 times.
52
      for j in range(5):
53
          queue.dequeue()
54
      print(queue) # [None, None, None, None, 5, 6
55
56
      # Enqueue 8, 9, 10, 11, 12
57
      for k in range(5):
```

```
58          queue.enqueue(k + 8)
59          print(queue) # [10, 11, 12, None, None, 5, 6, 7, 8
60
61
62 if __name__ == '__main__':
63          main()
64

[0, 1, 2, 3, 4, 5, 6, 7, None, None]
[None, None, None, None, None, 5, 6, 7, None, None]
[10, 11, 12, None, None, 5, 6, 7, 8, 9]
```

## → 2. Computing Spans

```
1 class ArrayStack:
 2
           Stack implemented with python list append/pop''
      def init (self):
 3
 4
           self.array = []
 5
      def len (self):
 6
 7
           return len(self.array)
 8
 9
      def is empty(self):
           return len(self.array) == 0
10
11
12
      def push(self, e):
13
           self.array.append(e)
14
15
      def top(self):
16
           if self.is empty():
               raise Exception("Stack is empty!")
17
18
           return self.array[-1]
19
20
      def pop(self):
21
           if self.is empty():
22
               raise Exception("Stack is empty!")
23
           return self.array.pop(-1)
24
25
      def repr__(self):
26
               return str(self.array)
27
```

```
28 def spans1(x):
       1 1 1
29
30
       :param X: List[Int] -- list of integers.
31
32
      No stack allowed. For each index,
33
      we look to the front of array until we find a value
34
35
       @return: list of span values.
       1 1 1
36
37
38
       res=[]
39
      # pointer=0
40
       for idx in range(len(x)):
41
           count=1
42
           ap=idx-1
43
           while ap>=0 and x[idx]>=x[ap]:
44
               count+=1
45
               ap=1
46
           res.append(count)
47
       return res
48
49
      # res=[]
50
      \# ap=0
51
      # count=0
52
      # for pos in range(len(X)):
53
      #
             count=1
54
      #
             ap=pos-1
55
      #
             while ap >= 0:
56
      #
                 while X[pos]>=X[ap]:
57
                      count+=1
      #
58
      #
                      ap = 1
59
      #
                  res.append(count)
60
      #
                 break
61
62
      # return res
63
64 def spans2(x):
65
66
       :param X: List[Int] -- list of integers.
67
68
       Use a stack. We use the stack to compute the span d
```

```
69
70
       If the top of the stack is "Smaller" than the next
71
       top of the stack should be popped.
72
73
       :return: list of span values.
74
75
       res=[1 for i in range(len(x))]
76
       stack=ArrayStack()
77
       for i, v in enumerate(x):
78
           count=1
79
           while stack and stack.top()[1]<v:
80
               temp=stack.pop()
81
               count+=res[temp[0]]
82
           res[i]=count
83
           stack.push((i,v))
84
       return res
85
86
87
88
89 def main():
90
       print(spans1([6,3,4,5,2])) # [1, 1, 2, 3, 1]
91
      print(spans1([6,7,1,3,4,5,2])) # [1, 2, 1, 2, 3, 4
       print(spans2([6,3,4,5,2])) # [1, 1, 2, 3, 1]
92
      print(spans2([6,7,1,3,4,5,2])) # [1, 2, 1, 2, 3, 4
93
94
95 if name == '__main__':
96
      main()
97
98
99
   [1, 1, 2, 3, 1]
   [1, 2, 1, 2, 3, 4, 1]
   [1, 1, 2, 3, 1]
   [1, 2, 1, 2, 3, 4, 1]
```

# → 3. Double ended queue

```
1 class ArrayDeque:
2     DEFAULT_CAPACITY = 10
3
```

```
def init (self):
 4
           self._data = [None] * ArrayDeque.DEFAULT_CAPACI
 5
           self. size = 0
 6
           self. front = 0
 7
 8
      def len_(self):
 9
10
           return self. size
11
12
      def is empty(self):
           return self. size == 0
13
14
15
      def is full(self):
           return self. size == len(self. data)
16
17
18
      def first(self):
19
           return self. data[self. front]
20
21
      def last(self):
           return self. data[(self. front+self. size-1)%Ar
22
23
24
      def delete first(self):
25
           temp=self. data[self. front]
26
           self._data[self._front]=None
27
           self. front=(self. front+1)%ArrayDeque.DEFAULT
28
29
           self. size-=1
30
31
           return temp
32
33
      def add first(self, e):
           self. front=((self. front-1)% ArrayDeque.DEFAUL
34
           self. data[self. front]=e
35
36
           self. size+=1
37
           return
38
39
      def delete last(self):
           temp=self. data[(self. front+self. size-1)%Arra
40
           self._data[(self._front+self._size-1)%ArrayDequ
41
42
           self. size-=1
43
44
           return temp
```

```
45
46
       def add last(self, e):
47
           self. data[(self. front+self. size)%ArrayDeque.
           self. size+=1
48
49
50
      def str (self):
51
52
           # return "Incomplete!! Change this."
           return str(self. data)
53
54
55 def main():
56
      # Empty Queue, size 10.
57
       deque = ArrayDeque()
58
      # Add 0, 1, 2, 3 following FIFO.
59
60
       for i in range(4):
61
           deque.add first(i)
62
      print(deque) # [None, None, None, None, None, None
63
64
      # Add 4, 5, 6, 7 following LIFO.
65
       for j in range(4):
           deque.add last(j + 4)
66
67
      print(deque) # [4, 5, 6, 7, None, None, 3, 2, 1, 0
68
69
      # Remove first one
70
      print(deque.delete first()) # 3
71
72
      # Remove last one
73
      print(deque.delete last()) # 7
74
75
76 if name == ' main ':
77
      main()
78
   [None, None, None, None, None, 3, 2, 1, 0]
   [4, 5, 6, 7, None, None, 3, 2, 1, 0]
   7
```

# 4. Evaluation of arithmetic expressions

```
1 class ArrayStack:
           Stack implemented with python list append/pop''
 2
      def init (self):
 3
 4
           self.array = []
 5
      def len (self):
 6
 7
           return len(self.array)
 8
 9
      def is empty(self):
           return len(self.array) == 0
10
11
12
      def push(self, e):
13
           self.array.append(e)
14
15
      def top(self):
16
           if self.is empty():
17
               raise Exception("Stack is empty!")
18
           return self.array[-1]
19
20
      def pop(self):
21
           if self.is empty():
               raise Exception("Stack is empty!")
22
23
           return self.array.pop(-1)
24
25
      def repr (self):
26
           return str(self.array)
27
28
29 def simple calc(num1, num2, op):
30
      num1=float(num1)
31
      num2=float(num2)
      if op=='+':
32
33
           return num1+num2
34
      elif op=='-':
35
           return num1-num2
36
      elif op=='*':
37
           return num1*num2
38
      elif op=='/':
39
           return num1/num2
40
41 def evaluate(string):
       11 11 11
42
```

```
43
       :param string: Str -- The string arithmetic express
44
45
       :return: Float -- the float answer for the given ar
       11 11 11
46
47
      operator stack = ArrayStack()
48
      operand stack = ArrayStack()
      table = {"+":2, "-":2, "*":3, "/":3, "(":1, ")":1}
49
      for c in string:
50
51
           if c !=' ':
52
               if c in table.keys():
                   if c =='(':
53
54
                       table['+']+=2
                       table['-']+=2
55
                       table['*']+=2
56
                       table['/']+=2
57
58
                   elif c==')':
                       # table = {"+":2, "-":2, "*":3, "/"
59
60
                       table['+']-=2
                       table['-']-=2
61
62
                       table['*']-=2
63
                       table['/']-=2
64
                   elif operator stack and table[c] <= opera
65
                       while operator stack and table[c]<=
66
                            num2=operand stack.pop()
67
                            num1=operand stack.pop()
68
                            res=simple calc(num1, num2, ope
69
                            operand stack.push(res)
                       operator stack.push((c, table[c]))
70
71
                       # continue
72
                   else:
73
                       operator stack.push((c, table[c]))
74
               else:
                   operand stack.push(c)
75
      # if len(operator stack)==len(operand stack)==1:
76
                       # operand stack.push(simple calc(op
77
78
      while operator stack:
79
           temp=operand stack.pop()
80
81
           res=simple calc(operand stack.pop(), temp, oper
           operand stack.push(res)
82
83
84
      return operand stack.top()
```

```
85
86
87
88 if __name__ == '__main__':
89
      # print(evaluate('9+8*4/2+3'))
90
      print(evaluate("9 + 8 * ( 7 - 6 ) / ( 2 / 8 )"))
91
      print(evaluate("9 + 8 * 7 / (6 + 5 ) - (4 + 3 ) *
92
      print(evaluate("9 + 8 * 7 / ( ( 6 + 5 ) - ( 4 + 3 )
93
94
95
  41.0
  0.09090909090908994
  -9.6666666666668
```

X