```
!date
```

```
Fri Mar  1 12:01:48 PM UTC 2024
```

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

CSCI-SHU 210 Data Structures

Recitation 5 Stacks and Queues

Name: Margad Gereltbattar (MJ)

NetID:mg7502

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

```
class ArrayQueue():

    DEFAULT_CAPACITY = 10

    def __init__(self):
        self._data = []

    def __len__(self):
        return len(self._data)

    def is_empty(self):
        return len(self._data) == 0

    def first(self):
        if self.is_empty():
            raise Exception("Queue is empty")
        return self._data[s]

    def dequeue(self):
        return self._data.pop(0)

    def enqueue(self, e):
        self._data.append(e)

    def __str__(self):
        ''' You can simply print self._data '''
        return str(self._data)

def main():
    # Empty Queue, size 10.
    queue = ArrayQueue()

    # Enqueue 0, 1, 2, 3, 4, 5, 6, 7
    for i in range(8):
        queue.enqueue(i)
    print(queue)   # [0, 1, 2, 3, 4, 5, 6, 7, None, None]

    # Dequeue 5 times.
    for j in range(5):
        queue.dequeue()
    print(queue)  # [None, None, None, None, None, 5, 6, 7, None, None]

    # Enqueue 8, 9, 10, 11, 12
    for k in range(5):
        queue.enqueue(k + 8)
    print(queue)  # [10, 11, 12, None, None, 5, 6, 7, 8, 9]


if __name__ == '__main__':
    main()
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
[5, 6, 7]
[5, 6, 7, 8, 9, 10, 11, 12]
```

1. Array Queue

```
class ArrayQueue():

    DEFAULT_CAPACITY = 10

    def __init__(self):
        self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
        self._size = 0
        self._front = 0

    def __len__(self):
        return self._size

    def is_empty(self):
        return self._size == 0

    def first(self):
        if self.is_empty():
            raise Exception('queue is empty')
        return self._data[self._front]

    def dequeue(self):
        if self.is_empty():
            raise("queue is empty")
        result = self._data[self._front]
        self._data[self._front] = None
        self._size -= 1
        self._front = (self._front + 1) % len(self._data)
    def enqueue(self, e):
        if self._size == len(self._data):
            raise Exception("queue is full")
        else:
            back = (self._size + self._front) % len(self._data)
            self._data[back] = e
            self._size += 1

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

def main():
    # Empty Queue, size 10.
    queue = ArrayQueue()

    # Enqueue 0, 1, 2, 3, 4, 5, 6, 7
    for i in range(8):
        queue.enqueue(i)
    print(queue)   # [0, 1, 2, 3, 4, 5, 6, 7, None, None]

    # Dequeue 5 times.
    for j in range(5):
        queue.dequeue()
    print(queue)  # [None, None, None, None, None, 5, 6, 7, None, None]
```

```
    # Enqueue 8, 9, 10, 11, 12
    for k in range(5):
        queue.enqueue(k + 8)
    print(queue)  # [10, 11, 12, None, None, 5, 6, 7, 8, 9]


if __name__ == '__main__':
    main()
```

```
[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

```
class ArrayStack:
    ''' Stack implemented with python list append/pop'''
    def __init__(self):
        self.array = []

    def __len__(self):
        return len(self.array)

    def is_empty(self):
        return len(self.array) == 0

    def push(self, e):
        self.array.append(e)

    def top(self):
        if self.is_empty():
            raise Exception("Stack is empty!")
        return self.array[-1]

    def pop(self):
        if self.is_empty():
            raise Exception("Stack is empty!")
        return self.array.pop(-1)

    def __repr__(self):
            return str(self.array)

def spans1(X):
    rs = [1] * len(X)
    for x in range(len(X)):
        for j in range(x-1,-1,-1):
            if X[x] > X[j]:
                rs[x]+=1
            else: break
    return rs
def spans2(X):
    '''
    :param X: List[Int] -- list of integers.

    Use a stack. We use the stack to compute the span distance.

    If the top of the stack is “Smaller” than the next data,
    top of the stack should be popped.

    :return: list of span values.
    '''
    stk = ArrayStack()
    spans = [0] * len(X)
```

```
def main():
    print(spans1([6,3,4,5,2])) # [1, 1, 2, 3, 1]
    print(spans1([6,7,1,3,4,5,2]))  # [1, 2, 1, 2, 3, 4, 1]
    print(spans2([6,3,4,5,2])) # [1, 1, 2, 3, 1]
    print(spans2([6,7,1,3,4,5,2]))  # [1, 2, 1, 2, 3, 4, 1]

if __name__ == '__main__':
    main()
```

```
 [1, 1, 2, 3, 1]
 [1, 2, 1, 2, 3, 4, 1]
 None
 None
```

3. Double ended queue

```
class ArrayDeque:
    DEFAULT_CAPACITY = 10

    def __init__(self):
        self._data = [None] * ArrayDeque.DEFAULT_CAPACITY
        self._size = 0
        self._front = 0

    def __len__(self):
        return self._size

    def is_empty(self):
        return self._size == 0

    def is_full(self):
        return self._size == len(self._data)

    def first(self):
        if self.is_empty():
            raise Exception('Deque is empty')
        return self._data[self._front]

    def last(self):
        if self.is_empty():
            raise Exception('Deque is empty')
        back = (self._front + self._size - 1) % len(self._data
        return self._data[back]

    def delete_first(self):
        if self.is_empty():
            raise Exception('Deque is empty')
        answer = self._data[self._front]
        self._data[self._front] = None
        self._front = (self._front + 1) % len(self._data)
        self._size -= 1
        return answer

    def add_first(self, e):
        if self.is_full():
            self._resize(2 * len(self._data))
        self._front = (self._front - 1) % len(self._data)
        self._data[self._front] = e
        self._size += 1

    def delete_last(self):
        if self.is_empty():
            raise Exception('Deque is empty')
        back = (self._front + self._size - 1) % len(self._data
        answer = self._data[back]
        self._data[back] = None
        self._size -= 1
```

```
        return answer

    def add_last(self, e):
        if self.is_full():
            self._resize(2 * len(self._data))
        back = (self._front + self._size) % len(self._data)
        self._data[back] = e
        self._size += 1

    def _resize(self, cap):
        old = self._data
        self._data = [None] * cap
        walk = self._front
        for k in range(self._size):
            self._data[k] = old[walk]
            walk = (1 + walk) % len(old)
```

4. Evaluation of arithmetic expressions

```
        if self.is_empty():
class ArrayStack:
    ''' Stack implemented with python list append/pop'''
    def __init__(self):
        self.array = []

    def __len__(self):
        return len(self.array)

    def is_empty(self):
        return len(self.array) == 0

    def push(self, e):
        self.array.append(e)

    def top(self):
        if self.is_empty():
            raise Exception("Stack is empty!")
        return self.array[-1]

    def pop(self):
        if self.is_empty():
            raise Exception("Stack is empty!")
        return self.array.pop(-1)
```