# HW5: Data Structure in Mathematics

Saen-Anan Bunyasiwa (SCIM 6105626)
Monchita Toopsuwan (SCIM 6105731)

February 11, 2020

# R - 6.1

What values are returned during the following series of stack operations, if executed upon an initially empty stack? `push(5)`, `push(3)`, `pop()`, `push(2)`, `push(8)`, `pop()`, `pop()`, `push(9)`, `push(1)`, `pop()`, `push(7)`, `push(6)`, `pop()`, `pop()`, `push(4)`, `pop()`, `pop()`.

## Solution Table [1]

| Operation | Return | Values in the Stack |
|---|---|---|
| `push(5)` | - | `[5]` |
| `push(3)` | - | `[5, 3]` |
| `pop()` | 3 | `[5]` |
| `push(2)` | - | `[5, 2]` |
| `push(8)` | - | `[5, 2, 8]` |
| `pop()` | 8 | `[5, 2]` |
| `pop()` | 2 | `[5]` |
| `push(9)` | - | `[5, 9]` |
| `push(1)` | - | `[5, 9, 1]` |
| `pop()` | 1 | `[5, 9]` |
| `push(7)` | - | `[5, 9, 7]` |
| `push(6)` | - | `[5, 9, 7, 6]` |
| `pop()` | 6 | `[5, 9, 7]` |
| `pop()` | 7 | `[5, 9]` |
| `push(4)` | - | `[5, 9, 4]` |
| `pop()` | 4 | `[5, 9]` |
| `pop()` | 9 | `[5]` |

[1] Note that we have 9 pushes, and 8 pops. Therefore, we have one value left in the stack as expected.

# R - 6.5

Implement a function that reverses a list of elements by pushing them onto a stack in one order, and writing them back to the list in reversed order.

**Solution Source Code**

```
def list_reverse(S):
        t = ArrayStack()
        for e in S:
                t.push(e)

for i in range(len(S)):
        S[i] = t.pop()

lists = [[1,2,3,4,5,6],
        [9,8,7,6,5,4,3,2,1]]

for X in lists:
        print ('Before: ', l)
        reverse_list(l)
        print('After:  ', l)
        print()
```

**Output**

```
Before:  [1, 2, 3, 4, 5, 6]
After:   [6, 5, 4, 3, 2, 1]

Before:  [9, 8, 7, 6, 5, 4, 3, 2, 1]
After:   [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

# R - 6.7

What values are returned during the following sequence of queue operations, if executed on an initially empty queue? `enqueue(5)`, `enqueue(3)`, `dequeue()`, `enqueue(2)`, `enqueue(8)`, `dequeue()`, `dequeue()`, `enqueue(9)`, `enqueue(1)`, `dequeue()`, `enqueue(7)`, `enqueue(6)`, `dequeue()`, `dequeue()`, `enqueue(4)`, `dequeue()`, `dequeue()`.

## Solution Table

| Operation | Return | Values in the Stack |
|---|---|---|
| `enqueue(5)` | - | `[5]` |
| `enqueue(3)` | - | `[5, 3]` |
| `dequeue()` | 5 | `[3]` |
| `enqueue(2)` | - | `[3, 2]` |
| `enqueue(8)` | - | `[3, 2, 8]` |
| `dequeue()` | 3 | `[2, 8]` |
| `dequeue()` | 2 | `[8]` |
| `enqueue(9)` | - | `[8, 9]` |
| `enqueue(1)` | - | `[8, 9, 1]` |
| `dequeue()` | 8 | `[9, 1]` |
| `enqueue(7)` | - | `[9, 1, 7]` |
| `enqueue(6)` | - | `[9, 1, 7, 6]` |
| `dequeue()` | 9 | `[1, 7, 6]` |
| `dequeue()` | 1 | `[7, 6]` |
| `enqueue(4)` | - | `[7, 6, 4]` |
| `dequeue()` | 7 | `[6, 4]` |
| `dequeue()` | 6 | `[4]` |

# R - 6.8

Suppose an initially empty queue $Q$ has executed a total of 32 enqueue operations,10 first operations, and 15 dequeue operations, 5 of which raised Empty errors that were caught and ignored. What is the current size of $Q$?

**Source Code**

```
if __name__ == '__main__':
        Q = ArrayQueue()
        for i in range(4):
                Q.dequeue()

        for i in range(31):
                Q.enqueue(i)

        for i in range(9):
                Q.first()

        for i in range(9):
                Q.dequeue()

        print(Q.__len__())
```

Output:

```
22
```

# Matching Parentheses in Python

```
def is_matched(expr):
        """Return True if all delimiters are properly match; False otherwise."""
        lefty = '({['   # opening delimiters
                righty = ')}]'  # respective closing delims
        S = ArrayStack()
        for c in expr:
                if c in lefty:
                        S.push(c)  # push left delimiter on stack
                elif c in righty:
                        if S.is_empty():
                                return False   # nothing to match with
                        if righty.index(c) != lefty.index(S.pop()):
                                return False    # mismatched
        return S.is_empty()    # were all symbols matched?

print(is_matched('()(()){([()])}'))
```

**If lefty**:

Starting with `S = []`

`expr = ['(']`

Is `'('` lefty? True

Push `'('` into `S`

**If righty**:

`expr = [')']`

Is `')'` lefty? False

Is `')'` righty? True

Is `S` empty? False

Popped `S`

Is `0 != 0` False

Now, `S = []`

**If lefty but not yet righty**:

`is_matched('{{}}')`

`expr = ['{']`

Is `'{'` lefty? True

Push `'{'` into `S`

Next, `expr = ['{']`

Is `'{'` lefty? True

Push `'{'` into `S`

Now, `S = ['{', '{']`

Next, `expr = ['}']`

Is `'}'` lefty? False

Is `'}'` righty? True

Is `S` empty? False

Popped `S`

Is `1 != 1` False

Now, `S = ['{']`

Continue loop ...