Problem Set 2

Justin Ely

615.202.81.FA15 Data Structures

15 September, 2015

1a)

Set the variable i equal to the second element from the top of the stack. Leave the stack unchanged.

A very brute force method, employing each method the required number of times is outlined below. This method does no checking for errors, and is very verbose to indicate the core ideas.

```
s = Stack()  # Empty stack initializer
a = s.pop()
b = s.pop()
i = b

s.push(b)
s.push(a)
```

An alternative method, with some error checking, and a more easily extended signature is below. This verifies that the stack is not empty during each iteration, and could easily have dynamic limits to get the nth item from the top of the stack.

```
s = Stack()
t = Stack()

n_popped = 0
while popped < 2:
    if s.empty():
        raise Exception

    i = s.pop()
    t.push(i)
```

```
    n_popped += 1

# Restore popped values to original stack
while n_popped > 0:
    s.push(t.pop())
    n_popped -= 1
```

1b)

Given an integer n, set the variable i equal to the nth element from the top of the stack. Leave the stack without its top n elements

---

```
s = Stack()

j = 1
while j < n:
    if s.empty():
        raise Exception

    s.pop()
    j += 1

i = s.pop()
```

2a)

Set the variable i equal to the bottom element of the stack. Leave the stack unchanged.

My implementation takes advantage of the fact that the underlying pop mechanism will modify the stack appropriately on each iteration in a while loop. Each time through the stack will be 1 item fewer, until if finally returns True on the empty check. At that point, a simple peek at the top item of the spare stack will give the bottom value of the original stack, followed by pushing all the items back onto the original.

```
s1 = stack()
s2 = stack()

while not s1.empty():
```

```
    s2.push(s1.pop())

i = s2.peek()

while not s2.empty():
    s1.push(s2.pop())
```

2b)

Set the variable i equal to the third element from the bottom of the stack.

```
s1 = stack()
s2 = stack()

while not s1.empty():
    s2.push(s1.pop())

iter = 0
while iter < 3:
    if s2.empty():
        raise Exception

    i = s2.pop()
    iter += 1
```

3a)

Iteration	Stack
0	
1	{
2	{ [
3	{
4	{ [
5	{ [(
6	{ [
7	{

This would raise an error as the left curly brace is never matched and the stack is not empty at the end of the expression.

3b)

Iteration	Stack
0	
1	(
2	((
3	(
4	({
5	({ (
6	({ (
7	({ ([
8	({ (
9	({
10	(
11	

This expression would not raise and error as all delimiters matched and the stack was empty after evaluation.

4)

```
def mirrored_string(string):
    s = Stack()
    second_half = False

    for letter in string:
        if not second_half:
            s.push(letter)

        if letter == 'C':
            s.pop()
            second_half = True

        if (s.empty()) or (letter != s.pop()):
            return False

    return True
```

---

5)

```
def mirrored_string()
    # using function definition from problem 4

def stack_to_string(stack):
    sub_string = ''
    while not s.empty():
        sub_string += stack.pop()

      return sub_string

def pattern_match(string):
    for letter in string:
        s.push(letter)

        if letter == 'D':
            s.pop()

            sub_string = stack_to_string(s)
            if not mirrored_string(sub_string):
                return False

    if not s.empty():
        sub_string = stack_to_string(s)
        if not mirrored_string(sub_string):
            return False

    return True
```

---

6)

For my implementation I chose a stack of stacks. This would allow an arbitrary number of integers to be added to each stack element. I know this has some particulars of various languages - returning more than one value, accepting variable-length arguments, but since this is pseudo-code that seemed alright.

*If I had my choice of languages and datatypes, I'd like something similar to a Python list as each element. However, given the point made on Java vectors, I figured lists might not be viable options.

```
class IntStack()
    this.stack = Stack()

    def push(integers)

        sub_stack = Stack()
        for i in integers:
            sub_stack.push(i)

        this.stack.push(sub_stack)

    def pop()
        if this.stack.empty()
            raise Exception

        sub_stack = this.stack.pop()
        while not sub_stack.empty():
            sub_stack.pop()
```

---

7)

```
class StackArray(array_data)
    left_data = Stack()     #TOP of stack points to N-1 index
    right_data = Stack()    #TOP of stack points to 0 index

    top = 0

    initialize(array_data)  #fill the right stack

    def initialize(array_data)
        for item in array_data:
            right_data.push(item)

    def shift_to(index)
        while top != index:
            if index > top:
                if right_data.empty()
                    raise Exception

                left_data.push(right_data.pop())
```

```
                top += 1

            elif index < top:
                if left_data.empty()
                    raise Exception

                right_data.push(left_data.pop())
                top -= 1

    def setitem(index, value):
        shift_to(index)
        right_data.pop()
        right_data.push(value)

    def getitem(index)
        shift_to(index)
        return right_data.peek()

    def slice(start, stop):
        for (i=start; i<stop; i++):
            shift_to(start)

            #  pass these values out to another built-in like list, vector, etc.
            #  current implementation is yielding an iterable, for demonstration
            yield right_data.peek()
```

8)

```
class ArrayStacks()
    s = array[SPACESIZE]
    top1 = 0
    top2 = SPACESIZE - 1

    def push1(value)
        if stop1 >= top2:
            raise Exception

        s[top1] = value
        top1 += 1

    def push2(value)
        if top2 <= top1:
```

```
            raise Exception

        s[top2] = value
        top2 -= 1

    def pop1()
        if top1 == -1:
            raise Exception

        top1 -= 1
        return s[top1 + 1]

    def pop2()
        if top2 > (SPACESIZE - 1):
            raise EXCEPTION

        top2 += 1
        return s[top2 - 1]
```

9)

9a]

Prefix: - * + A B + $ C - D E F G
Postfix: A B + C D E - $ F + * G -

9b]

Prefix: + A $ / + * - B C - D E F G - H J
Postfix: A B C - D E - * F + G / H J - $ +

10)

10a: A + (((B $ C) * D) - ((E + F) / (G * H))) + I
10b: ((A $ B) - C) + D * E * F * G
10c: (A - B + C) $ (D + E - F)
10d: A * (B $ (C + (D - E))) - (E * F)

11)

11a: ((A+B)-C) - ((B+A) $ C) = -21
11b: (A * (B+C)) * (C+(B-A)) = 20

12)

```
def prefix(in_string):
    ops = Stack()
    out_string = ''

    for c in in_string:
        if isletter(c):
            out_string = letter + out_string
        if (isoperator(c)):
            if (c == '(') or (ops.empty()) or (priority(c) > ops.peek()):
                out_string = ops.pop + out_string
            else:
                ops.push(c)

    return out_string
```