Stacks worksheet (solutions)

Arjun Chandrasekhar

Review of Stack Operations

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. Below are the operations associated with a stack:

- **Push(x):** Add an element x to the top of the stack.
- **Pop():** Remove and return the top element of the stack. (Error if the stack is empty.)
- **Peek():** Return the top element without removing it. (Error if the stack is empty.)
- **isEmpty():** Check if the stack has no elements.
- **getLength():** Return the number of elements in the stack.

Array-based stack

Below is the pseudocode for a series of stack operations on an array-based stack. Assume the stack starts empty and has an initial capacity of 5. Describe what the array contents will look like after each operation. State whenever an operation returns a value, and state what value will be returned.

- `Push(10)`

`[10, _, _, _, _]`

- `Push(20)`

`[10, 20, _, _, _]`

- `Push(30)`

`[10, 20, 30, _, _]`

- `Pop()`

`[10, 20, _, _, _], Pop() returns 30`

- `Push(40)`

`[10, 20, 40, _, _]`

- `Peek()`

`[10, 20, 40, _, _] Peek() returns 40`

- Push(50)

```
[10, 20, 40, 50, _]
```

- Push(60)

```
[10, 20, 40, 50, 60]
```

- Push(80)

```
We must re-size
[10, 20, 40, 50, 60, 80, _, _, _, _]
```

- Push(70)

```
[10, 20, 40, 50, 60, 80, 70, _, _, _]
```

- Pop()

```
[10, 20, 40, 50, 60, 80, _, _, _, _] Pop() returns 70
```

- Push(10)

```
[10, 20, 40, 50, 60, 80, 10, _, _, _]
```

- Pop()

```
[10, 20, 40, 50, 60, 80, _, _, _, _] Pop() returns 10
```

- Pop()

```
[10, 20, 40, 50, 60, _, _, _, _, _] Pop() returns 80
```

- Pop()

```
[10, 20, 40, 50, _, _, _, _, _, _] Pop() returns 60
```

Question: How many Push() operations would be needed before we need to re-size the array?

We can service 6 push operations. We would need to resize after the 7th push operation.

Linked stack

Below is the pseudocode for a series of stack operations on an array-based stack. Assume the stack starts empty and has an initial capacity of 5. Describe what the array contents will look like after each operation. State whenever an operation returns a value, and state what value will be returned.

We will use $ to indicate the pointer to the head of the stack, and x to indicate the pointer to the tail of the stack.

- Push(5)

```
$->5->x
```

- Push(15)

```
$->15->5->x
```

- Push(25)

```
$->25->15->5->x
```

- Pop()

```
$->15->5->x Pop() returns 25
```

- Push(35)

```
$->35->15->5->x
```

- Peek()

```
$->35->15->5->x Peek() returns 35
```

- Push(6)

```
$->6->35->15->5->x
```

- Push(14)

```
$->14->6->35->15->5->x
```

- Pop()

```
$->6->35->15->5->x Pop() returns 14
```

- Pop()

```
$->35->15->5->x Pop() returns 6
```

- Peek()

```
$->35->15->5->x Peek() returns 35
```

Transforming a Stack

You are given the following stack configuration (top of stack is on the left):

Initial Stack: `->[D, C, B, A]`

Use stack operations to transform it into this configuration:

Target Stack: `->[D, A, C, B]`

How many total operations (pushes and pops) are needed? Write the sequence of operations that achieve this.

Operation	Resulting stack
Pop()	->[D, C, B]
Pop()	->[D, C]
Pop()	->[D]
Push(A)	->[D, A]
Push(C)	->[D, A, C]
Push(B)	->[D, A, C, B]

Reversing a String using a stack

Write pseudocode to reverse a string using stack operations. For example:

Input: "STACK"

Output: "KCATS"

Your pseudocode may include the following operations:

- Declaring a new empty stack
- Calling Push(), Pop(), and isEmpty() on an previously declared stack object
- Declaring an empty String
- Concatenating to the end of an existing String
- While loops
- Looping through each character in a String in the forwards direction

```
Create an empty Stack S
For each character ch in the input:
        S.push(ch)
```

```
Create an empty String Str
While S is not empty:
        ch ← S.pop()
        Concatenate the popped character to the end of Str
```

Balancing Parentheses

A stack can be used to check if a string of parentheses is balanced. For example:

Balanced: "(())"
Balanced: "()()"
Unbalanced: "(()"
Unbalanced: "())(()"

Write pseudocode to check if a string of parentheses is balanced using stack operations. Your pseudocode may include the following operations:

- Declaring a new empty stack
- Calling Push(), Pop(), Peek(), and isEmpty() on an previously declared stack object
- If/else-if/else statements
- Looping through each character in a String

```
Create an empty Stack S
For each character in the input:
        If ch == ‘(‘
                S.push(‘(‘)
        Otherwise:
                If S.isEmpty():
                        Output “Unbalanced”
                Else:
                        S.pop()

If S.isEmpty()
        Output “Balanced”
Else:
        Output “Unbalanced””
```

Applying your pseudocode

Run your pseudocode on each of the parenthesis Strings listed above. Describe the state of the stack after each String that you read.

Let’s run through this String: "(())"

- Read ‘(‘, Stack contents ->[‘(‘]
- Read ‘(‘, Stack contents ->[‘(‘, ‘(‘]
- Read ‘)’, Stack contents ->[‘(‘]
- Read ‘)’, Stack contents ->[]
- End of loop, Stack is empty, output Balanced

Let’s run through the String “()()”

- Read ‘(‘, Stack contents ->[‘(‘]
- Read ‘)’, Stack contents ->[]
- Read ‘(‘, Stack contents ->[‘(‘]
- Read ‘)’, Stack contents ->[]
- End of loop, Stack is empty, output Balanced

Let’s run through this String: "(()"

- Read ‘(‘, Stack contents ->[‘(‘]
- Read ‘(‘, Stack contents ->[‘(‘, ‘(‘]
- Read ‘)’, Stack contents ->[‘(‘]
- End of loop, Stack is not empty, output Unbalanced

Let’s run through the String “())(()”

- Read ‘(‘, Stack contents ->[‘(‘]
- Read ‘)’, Stack contents ->[]
- Read ‘)‘, Stack is not empty, output Unbalanced