University of Engineering and Technology Taxila
Computer Engineering Department

Data Structure and Algorithm

Lab Manual # 07

Lab Instructor: Engr. Shahryar Khan

University of Engineering and Technology Taxila
Computer Engineering Department

Lab Title: **Practicing Stacks Operations in Data Structures**

Lab Overview

This lab focuses on implementing and understanding Stacks, a fundamental data structure used in various real-world applications. The lab covers both array-based and linked-list-based implementations of stacks, including push, pop, peek, and isEmpty operations. Students will also explore real-life applications of stacks, such as expression evaluation, undo/redo functionality, browser history, and recursion call stacks.

By the end of this lab, students will have a strong foundation in using stacks efficiently in problem-solving scenarios, including their use in software applications and operating system functionalities.

Lab Objectives

- By completing this lab, students will:
- Understand the concept and importance of stacks in data structures.
- Implement stack operations using arrays and linked lists.
- Explore real-world applications of stacks in software engineering.
- Develop hands-on problem-solving skills through guided tasks and exercises
- Solve scenario-based problems involving stack applications.

Note: For the guided tasks type the code yourself.

Guided Tasks

Task 1: Implementing a Stack using Arrays

Objective: Implement a stack using an array with basic operations: push, pop, peek, and isEmpty.

Implementation:

```
class StackArray:
    def __init__(self, size):
        self.stack = []
        self.size = size

    def push(self, item):
        if len(self.stack) < self.size:
            self.stack.append(item)
        else:
            print("Stack Overflow: Cannot add more elements!")

    def pop(self):
        if self.stack:
            return self.stack.pop()
        else:
            print("Stack Underflow: No elements to pop!")
            return None

    def peek(self):
        return self.stack[-1] if self.stack else None
```

```
    def isEmpty(self):
        return len(self.stack) == 0

    def display(self):
        print("Stack:", self.stack)

# Example Usage
stack = StackArray(5)
stack.push(10)
stack.push(20)
stack.push(30)
stack.display()
stack.pop()
stack.display()
```

Task 2: Implementing a Stack using Linked Lists

Objective: Implement a stack using a linked list and perform stack operations.

Implementation:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class StackLinkedList:
    def __init__(self):
        self.top = None
```

```
def push(self, data):
    new_node = Node(data)
    new_node.next = self.top
    self.top = new_node

def pop(self):
    if self.top is None:
        print("Stack Underflow: No elements to pop!")
        return None
    popped_data = self.top.data
    self.top = self.top.next
    return popped_data

def peek(self):
    return self.top.data if self.top else None

def isEmpty(self):
    return self.top is None

def display(self):
    current = self.top
    print("Stack:", end=" ")
    while current:
        print(current.data, end=" -> ")
        current = current.next
    print("None")
```

```
# Example Usage

stack = StackLinkedList()

stack.push(10)

stack.push(20)

stack.push(30)

stack.display()

stack.pop()

stack.display()
```

Exercise Questions

Easy Problems (5 Questions)

1. **Stack Push & Pop**: Implement a stack where users can push and pop elements interactively.
2. **Check Stack is Empty**: Write a function to check if a stack is empty.
3. **Peek Implementation**: Implement a peek operation to retrieve the topmost element.
4. **Reverse a String using Stack**: Reverse a given string using stack operations.
5. **Check Balanced Parentheses**: Write a function to check if parentheses in an expression are balanced.

Intermediate Problems (5 Questions)

1. **Undo/Redo System**: Implement an undo/redo system using two stacks.
2. **Evaluate Postfix Expression**: Implement a function to evaluate a postfix expression.
3. **Browser Back & Forward Navigation**: Simulate browser history using stacks.
4. **Sort a Stack**: Implement a function to sort a stack using recursion.
5. **Recursive Stack Traversal**: Implement stack traversal using recursion instead of loops.

Advanced Problems (3 Questions)

1. **Call Stack Simulation**: Simulate recursive function calls using a stack.
2. **Stack-Based Expression Evaluator**: Implement an advanced calculator supporting parentheses and operator precedence.
3. **Tower of Hanoi Problem**: Solve the Tower of Hanoi using a stack-based approach.