

**Vellore Institute of Technology, Vellore**

**Course Name: Data Structures & Algorithms**

**Course Code: PMCA501P**

**Student Name: Barsha Routh**

**Instructor’s Name: [Instructor's Name]**

**Submission Date: 19th October 2024**

**Title: Critical Analysis of Data Structures for Postfix Expression Evaluation**

**Table of Contents**

1. Introduction
2. [Background/Context](#backgroundcontext)
3. [Main Body/Content](#main-bodycontent)  
   3.1. [Postfix Evaluation Using Stack](#postfix-evaluation-using-stack)  
   3.2. [Deque as an Alternative for Mixed Expressions](#deque-as-an-alternative-for-mixed-expre)
4. [Case Study/Example](#case-studyexample)
5. [Complete Implementation (C Implementation)](#complete-implementation-c-implementatio)
6. [Diagrams, Tables, and Charts](#diagrams-tables-and-charts)
7. [Conclusion](#conclusion)
8. [References](#references)

**1. Introduction**

The evaluation of postfix expressions is a common problem in computer science.

This assignment critically analyzes the stack as the preferred data structure due to its LIFO nature. We also explore alternative structures such as deques that could perform better in specific contexts.

The purpose is to assess these structures in terms of speed, complexity, and implementation efficiency.

**2. Background/Context**

Postfix notation (Reverse Polish Notation) is widely used in computer science. A stack is the most efficient data structure for this due to its LIFO mechanism, which fits naturally with postfix evaluations. A postfix expression places operands before the operator, and the last operands encountered are operated on first. The deque, a double-ended queue, can be a viable alternative in some advanced scenarios like mixed prefix-postfix expressions.

**3. Main Body/Content**

**3.1. Postfix Evaluation Using Stack**

A stack is ideally suited for postfix evaluation, where operands are pushed onto the stack, and the operator pops operands to compute results.

**Example**: 5 6 2 + \*

**Steps**:

* Push 5, 6, 2 onto the stack.
* Pop 6 and 2 for the + operation.
* Push the result 8 and then evaluate 5 \* 8, resulting in 40.

**Advantages**:

* **LIFO Efficiency**: The stack ensures constant-time O(1) operations for push and pop, making it ideal for postfix evaluations.
* **Simplicity**: Stack operations are simple, well-supported in most languages.

**Diagram 1: Stack Operation for Postfix Evaluation**

plaintext

Copy code

| 5 6 2 + \* | Stack: 5

| 5 6 2 + \* | Stack: 5, 6

| 5 6 2 + \* | Stack: 5, 6, 2

| 5 6 2 + \* | Pop: (6 + 2 = 8) -> Stack: 5, 8

| 5 6 2 + \* | Pop: (5 \* 8 = 40) -> Stack: 40

**3.2. Deque as an Alternative for Mixed Expressions**

While a stack is optimal for postfix, a deque could outperform it when dealing with both prefix and postfix notations. A deque allows operations at both ends, making it possible to evaluate expressions where some parts follow prefix notation and others follow postfix notation.

**Example**:  
Expression: \* 3 + 4 2 (prefix) and - 5 1 (postfix).

Steps:

1. Use the front of the deque for prefix evaluation.
2. Use the back for postfix evaluation.

**Diagram 2: Deque Operations**

plaintext

Copy code

Expression: \* 3 + 4 2 -> Deque: 3, +, 4, 2 (Prefix Evaluation)

Deque -> Postfix: - 5 1 -> Deque: 3, 6 (after prefix evaluation)

**4. Case Study/Example**

In postfix expression evaluation, a stack is best suited for traditional cases. In contrast, a scenario where multiple notations (prefix and postfix) are mixed, a deque offers more flexibility due to its ability to insert and remove from both ends.

**5. Complete Implementation (C Implementation)**

c

Copy code

#include <stdio.h>

#include <stdlib.h>

// Stack Node Structure

struct Node {

int data;

struct Node\* next;

};

// Stack Operations

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

return newNode;

}

int isEmpty(struct Node\* stack) {

return !stack;

}

void push(struct Node\*\* stack, int data) {

struct Node\* newNode = createNode(data);

newNode->next = \*stack;

\*stack = newNode;

}

int pop(struct Node\*\* stack) {

if (isEmpty(\*stack)) return -1;

struct Node\* temp = \*stack;

\*stack = (\*stack)->next;

int popped = temp->data;

free(temp);

return popped;

}

// Postfix Evaluation Function

int evaluatePostfix(char\* exp) {

struct Node\* stack = NULL;

for (int i = 0; exp[i]; ++i) {

if (isdigit(exp[i])) push(&stack, exp[i] - '0');

else {

int val1 = pop(&stack);

int val2 = pop(&stack);

switch (exp[i]) {

case '+': push(&stack, val2 + val1); break;

case '-': push(&stack, val2 - val1); break;

case '\*': push(&stack, val2 \* val1); break;

case '/': push(&stack, val2 / val1); break;

}

}

}

return pop(&stack);

}

int main() {

char exp[] = "562+\*";

printf("Postfix Evaluation: %d\n", evaluatePostfix(exp));

return 0;

}

**6. Diagrams, Tables, and Charts**

* **Diagram 1**: Stack operation for postfix evaluation (as shown in the earlier section).
* **Diagram 2**: Deque operation for mixed expression (prefix + postfix).

**7. Conclusion**

A stack is the most efficient and straightforward structure for postfix expression evaluation, offering constant-time operations and simplicity. However, a deque might be more suitable for mixed notations due to its flexibility. Both structures offer unique benefits depending on the nature of the problem.

**8. References**

* Weiss, M. A. (2014). *Data Structures and Algorithm Analysis in C++*.
* Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*.
* Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed Systems: Principles and Paradigms*. Pearson Education.