"Enhanced Stack Data Structure for Real-Time Minimum Element Tracking"

RA2411030010295-SHAIK AIMAN HUSSAIN,RA2411030010299-GADUPUTI PURUSHOTHAM,RA2411030010301-MOPADA BHARATH KUMAR,RA2411030010308-AVOY DEBNATH
SRM UNIVERSITY-DEPARTMENT OF NWC

ABSTRACT

Stacks are one of the most fundamental data structures used in computing for expression evaluation, memory management, and recursion handling.While traditional stacks support operations like push, pop, and top efficiently in O(1) time, finding the minimum element typically requires O(n) traversal of the stack.This project presents an optimized stack design implemented in the C programming language, where the getMin() operation retrieves the minimum element in constant time O(1).The solution uses an auxiliary stack to track minimum values dynamically, making it ideal for real-time systems such as stock price tracking, gaming applications, and streaming analytics.

OBJECTIVES

1. . To design a special stack that supports push, pop, top, and getMin operations in O(1) time complexity.
2. 2. To implement the stack using C language with clean, modular, and efficient code.
3. 3. To use a dual stack approach with a main stack and an auxiliary minimum stack.
4. 4. To dynamically track the minimum element at every stage of stack operations.
5. 5. To make the stack suitable for real-time applications, like stock market tracking or gaming.
6. 6. To analyze and validate time and space complexity of the implemented solution.
7. 7. To ensure error handling for stack overflow and underflow conditions.
8. 8. To test the solution with various cases, including negative and duplicate values.
9. 9. To document the process, methodology, and results of the project clearly.
10. 10. To present the project through a poster and demonstration for academic evaluation.

MATERIALS AND METHODS

*Materials

>Hardware Requirements:

PC/Laptop with at least 4 GB RAM, 1.5 GHz processor

>Software Requirements:

GCC Compiler

IDE: Code::Blocks, Dev-C++, or VS Code

OperatingSystem:Windows/Linux/Mac

Methods

The approach involves two stacks:

Main Stack (mainStack): Stores all elements.

Min Stack (minStack): Stores the current minimum element at every state.

Algorithm Workflow:

Push(x):

Insert x into mainStack.

If minStack is empty or x <= minStack.top(), also push x into minStack.

Pop():

Remove the top element from mainStack.

If the popped element equals minStack.top(), pop from minStack as well.

Top():

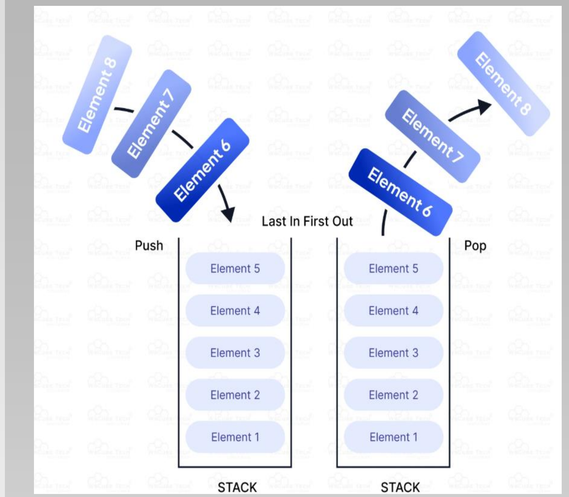
Return mainStack.top().

GetMin():

Return minStack.top().

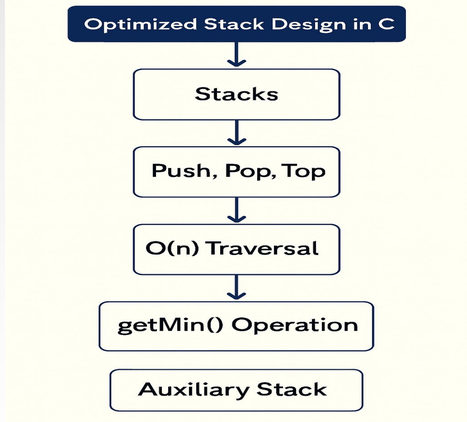
APPLICATIONS

Stock market analysis: Quickly get minimum stock price in a given range.
Gaming systems: Keep track of minimum health/score in real-time.
Data streaming: Maintain running minimum values in real-time systems.



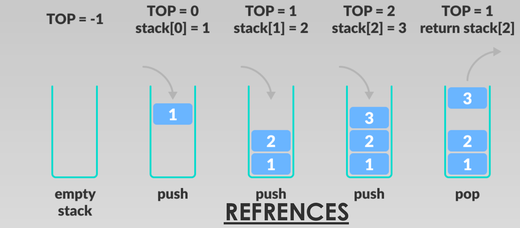
RESULT

The implementation successfully meets the project objectives.When tested with different input sequences, the stack operations, including getMin, consistently executed in O(1) time.Sample Execution:Input Sequence:Push → -2, 0, -3Pop → Remove last element



CONCLUSION

The project successfully demonstrates a C-based implementation of a stack that retrieves the minimum element in O(1) time.By maintaining an auxiliary stack, the solution ensures all four stack operations are efficient and suitable for real-time applications, such as: Stock Market Analysis – tracking minimum stock prices. Gaming Systems – monitoring minimum health or scores. Streaming Data – quickly finding minimum data points.This approach showcases a space-time tradeoff, where extra memory is used to achieve significant performance gains.



REFRENCES

References:1. Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language. Prentice Hall.

2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.

3. Horowitz, E., Sahni, S., & Anderson-Freed, S. (2008). Fundamentals of Data Structures in C. Universities Press.

ACKNOWLEDGMENT AND CONTACT

Guided by: Dr. Prabhu Chakkaravarthy

CODEQR :