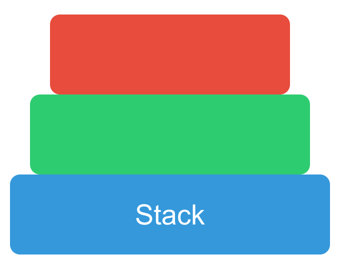
NESTED COMMENT BLOCK VALIDATOR

Implementation Using Stack Data Structure



Course: Data Structures and Algorithms

Domain: Computer Science & Engineering

Author: [Your Name]

Roll No: [Your Roll Number]

Institution: [Your College/University Name]

Date: December 2025

Table of Contents

1. Abstract

This project presents a comprehensive implementation of a Nested Comment Block Validator using Stack data structure. The validator efficiently checks the correctness of nested comment blocks in source code files by ensuring proper opening and closing of comment delimiters across multiple programming languages including C/C++, HTML, and Python. The stack-based approach provides optimal time complexity of O(n) where n is the length of the input text. The system accurately identifies syntax errors such as unclosed comment blocks, mismatched delimiters, and unexpected closing tags, providing detailed error reports with line numbers. This tool serves as a practical application of stack data structure in compiler design and code analysis, demonstrating the fundamental principle of Last-In-First-Out (LIFO) in solving real-world parsing problems.

Keywords: Stack Data Structure, Comment Block Validation, Nested Delimiters, Parser, Syntax Checking, LIFO, Algorithm Design, C Programming

2. Introduction

In modern software development, source code often contains various types of comments for documentation, debugging, and code explanation purposes. These comments use specific delimiter patterns that must be properly opened and closed. When comments are nested (one comment block inside another), the validation becomes more complex.

The Stack data structure, with its Last-In-First-Out (LIFO) principle, provides an elegant solution to this problem. When we encounter an opening delimiter, we push it onto the stack. When we encounter a closing delimiter, we check if it matches the most recent opening delimiter (top of stack). This approach naturally handles nested structures.

2.1 Motivation

- Compilers and interpreters need to validate comment syntax
- IDEs require real-time syntax checking
- Code quality tools need to detect malformed comments
- Understanding stack applications in real-world scenarios

3. Problem Statement

Given a source code file or text input containing various types of comment blocks, determine whether: