Report on
Stack-Based Browser History System

A Comprehensive Project Report

Submitted by:

B Navaneeth	**AP24110011614**
Harsha Srinivas	**AP24110011613**
Satish Sahu	**AP24110011620**
Vivek Verma	**AP24110011638**
Ashok Reddy	**AP24110011640**
A.Naga sree pavan	**AP24110011674**

Date: December 2025

Institution: Computer Science Department

Project Type: Data Structures & Algorithms (DSA) Implementation

Certificate Page

Certificate of Project Completion

This is to certify that the project titled **"Stack-Based Browser History System"** has been successfully completed and submitted as a course requirement for Data Structures and Algorithms.

The project demonstrates a practical understanding of stack data structures and their real-world applications in web browser functionality.

Project Guide: [Faculty Name]

Date: December 2025

Acknowledgement

I would like to express my gratitude to all those who contributed to the successful completion of this project.

First and foremost, I thank my project guide and faculty members for their valuable guidance, encouragement, and constructive feedback throughout the development process. Their insights helped me understand the practical applications of data structures in real-world scenarios.

I also acknowledge the online resources, documentation, and open-source communities that provided valuable reference material and coding examples. Websites like GeeksforGeeks, Stack Overflow, and LeetCode proved instrumental in understanding complex concepts.

Special thanks to my peers who provided feedback and suggestions that helped improve the project documentation and code quality.

Finally, I appreciate the opportunity to work on this project, which enhanced my understanding of stack data structures and their applications in designing efficient software systems.

Abstract

This report presents a comprehensive study and implementation of a **Stack-Based Browser History System** using the C++ programming language. The project demonstrates the practical application of stack data structures—a fundamental concept in computer science—in simulating real browser functionality like going back and forward through pages.

Key Points:

- **Objective:** Implement browser history navigation using two stack data structures
- **Technology Used:** C++ with STL (Standard Template Library)
- **Data Structure:** Two stacks (backStack for history, forwardStack for forward navigation)
- **Time Complexity:** O(1) for all operations (visit, back, forward, current)
- **Space Complexity:** O(n) where n is the number of visited pages
- **Features:** Visit new pages, go back, go forward, view current page, visualize stack states, animated UI

The system efficiently manages browser navigation with real-time visualization of internal stack operations. This project serves as an excellent learning tool for understanding how fundamental data structures solve practical problems in software engineering.

Introduction

What is a Stack?

A stack is a linear data structure that follows the **Last-In-First-Out (LIFO)** principle. In a stack:

- The last element added is the first one to be removed
- Elements are inserted and removed from the same end, called the **top**
- Two main operations: **PUSH** (insert) and **POP** (remove)

Real-World Applications of Stacks

Stacks are used extensively in computer science:

- Web browser back/forward buttons
- Undo/Redo functionality in text editors
- Function call management in programming languages
- Expression evaluation (infix to postfix conversion)
- Backtracking algorithms (like solving mazes)

Browser History: A Perfect Use Case

Every web browser maintains a history of visited pages. Users expect to:

- Move forward through recently visited pages
- Go back to previously visited pages
- Know the current page they are viewing
- Clear forward history when visiting a new page after going back

The stack data structure is perfectly suited for this because of its LIFO property and efficient O(1) operations. This project implements this concept using two stacks to handle both backward and forward navigation.

Problem Statement

The Challenge

Design and implement a browser history system that:

- Allows users to visit new web pages and maintain a history of these visits
- Enables users to navigate backward through their browsing history
- Enables users to navigate forward if they have gone backward
- Prevents invalid operations (e.g., going back when already at the first page)
- Clears the forward history when a new page is visited after navigating backward
- Provides visual feedback and real-time animation of stack operations

Constraints

- Must handle multiple page visits efficiently
- Must display the current page at any time
- Must prevent memory wastage while maintaining history
- Must provide an interactive and user-friendly interface
- Must run on multiple platforms (Windows, Linux, macOS)

Why This Problem?

This problem is an excellent teaching tool because it:

- Demonstrates stack fundamentals in a practical scenario
- Shows how two data structures work together
- Introduces the concept of state management
- Illustrates the importance of boundary checking (preventing errors)

Objectives

Primary Objectives

1. **Implement Stack Data Structure:** Create a working implementation of stacks using C++ STL
2. **Design Browser Logic:** Develop algorithms for back/forward navigation
3. **Ensure Efficient Operations:** Achieve O(1) time complexity for all operations
4. **Create Interactive UI:** Build a user-friendly console interface with visual feedback

Secondary Objectives

1. **Educational Value:** Demonstrate practical application of DSA concepts
2. **Code Quality:** Write clean, well-documented, and maintainable code
3. **Error Handling:** Implement proper validation and error messages
4. **User Experience:** Add animations and visual elements for better engagement
5. **Portability:** Ensure the code works across different platforms and compilers

Learning Outcomes

After completing this project, students will understand:

- How stacks work internally
- How to use C++ STL stack containers
- How to solve real-world problems using fundamental data structures
- How to design systems with multiple interacting components
- How to create interactive console applications

Project Layout / System Architecture

High-Level Architecture

The system consists of two main components:

1. WebPage Structure

- Stores the URL and title of a webpage
- Provides a display method to show webpage information

2. BrowserHistory Class

- Manages two stacks internally
- Handles all operations (visit, back, forward, current)
- Provides visualization and user interaction methods

Data Structure Diagram

```
┌──────────────────────────────────────────┐
│ BrowserHistory Class │
├──────────────────────────────────────────┤
│ │
│ ┌─────────────────┐ ┌─────────────────┐ │
│ │ backStack │ │ forwardStack │ │
│ │ (Navigation) │ │ (Forward Nav) │ │
│ │ │ │ │
│ │ Page 3 (TOP) │ │ Page 1 (TOP) │ │
│ │ Page 2 │ │ │
│ │ Page 1 (Bottom) │ │ │
│ └─────────────────┘ └─────────────────┘ │
│ │
├──────────────────────────────────────────┤
│ Public Methods: │
│ - visit(): Add new page │
│ - goBack(): Navigate to previous page │
│ - goForward(): Navigate to next page │
│ - showCurrent(): Display current page │
│ - showHistory(): Visualize both stacks │
└────────────────────────────────────────
```

System Flow

1. **Initialization:** Browser starts at homepage
2. **User Input:** User selects an operation (visit, back, forward, etc.)
3. **Processing:** Stack operations are performed with validation
4. **Visualization:** Current stack states are displayed with animation
5. **Status Update:** User sees the result and available next operations

Modules Explanation

Module 1: WebPage Structure

Purpose: Store and manage individual webpage information

Members:

- url (string): The web address of the page
- title (string): The display name of the page

Methods:

- toString(): Returns formatted string showing title and URL

Example:

```
WebPage page("https://github.com", "GitHub");
cout << page.toString();
// Output: GitHub (https://github.com)
```

Module 2: BrowserHistory Class (Core Logic)

2.1 Constructor

- Initializes with a homepage
- Creates empty forward stack
- Displays welcome message
- **2.2 Visit Operation (PUSH)**
- **Input:** New URL and title from user
- **Process:**
  - Validate URL is not empty
  - Push page to backStack
  - Clear forwardStack (important for browser behavior)
- **Output:** Confirmation message and animation

2.3 GoBack Operation (POP from back, PUSH to forward)

- **Input:** None (automatic)
- **Validation:** Ensure backStack has more than 1 element
- **Process:**

    - Pop current page from backStack
    - Push it to forwardStack
    - Display new current page
- **Output:** Previous page becomes current

2.4 GoForward Operation (POP from forward, PUSH to back)

- **Input:** None (automatic)
- **Validation:** Ensure forwardStack is not empty
- **Process:**
    - Pop page from forwardStack
    - Push it to backStack
    - Display new current page
- **Output:** Next page becomes current

2.5 ShowCurrent

- **Purpose:** Display the URL and title of current page
- **Process:** Access top of backStack
- **Output:** Current page information

2.6 ShowHistory

- **Purpose:** Visualize both stack states side-by-side
- **Process:** Extract stack contents into vectors and display
- **Output:** Formatted table showing both stacks

Module 3: Visualization & UI Methods

- `drawBeautifulStacks()`: Creates side-by-side stack visualization
- `animateStackOperation()`: Shows POP/PUSH animation
- `showOperation()`: Displays operation details with timing
- `showMenu()`: Displays available options to user

Features Implemented

Core Features

1. Visit New Page

- Users can enter a URL and optional page title
- New page is added to browsing history
- Forward history is automatically cleared
- **Time Complexity:** O(1)

2. Navigate Backward

- Move to the previously visited page
- Current page is saved for forward navigation
- Cannot go back past the first page (homepage)
- **Time Complexity:** O(1)

3. Navigate Forward

- Move to a page that was previously backed out of
- Only available if user has gone back at least once
- **Time Complexity:** O(1)

4. View Current Page

- Displays the URL and title of the current page
- Shows stack sizes at bottom of screen
- **Time Complexity:** O(1)

5. View History (Visualization)

- Shows both backStack and forwardStack side-by-side
- Displays all visited pages in order
- Shows which page is currently at the top (most recent)
- **Time Complexity:** O(n) where n is total pages visited

Advanced Features

6. Real-Time Animations

- POP and PUSH operations show visual animations
- Delays between steps for educational clarity
- Animated emojis showing stack direction

7. Smart Input Handling

- Accepts both numeric (1-6) and text commands
- Command options: "visit", "back", "forward", "current", "history", "quit"
- Error handling for invalid inputs

8. Automatic Title Generation

- If user doesn't provide a title, system generates one
- Smart detection: "google.com" → "Google Search Engine"
- Fallback: "Web Page" for unknown sites

9. Cross-Platform Console

- Clears screen before each operation
- Works on Windows ("cls") and Unix ("clear")
- Uses conditional command execution

10. State Monitoring

- Displays stack sizes at all times
- Shows current page status
- Provides operation confirmations
- Clear error messages for invalid operations

Data Flow Diagram

Operation Flow Diagram

```
START
│
├─→ User Input
│ │
│ ├─→ visit → Validate URL → PUSH(backStack)
│ │ → POP ALL(forwardStack)
│ │ → Display Animation
│ │
│ ├─→ back → Validate backStack.size > 1
│ │ → POP(backStack) → PUSH(forwardStack)
│ │ → Display Animation
│ │
│ ├─→ forward → Validate forwardStack.empty
│ │ → POP(forwardStack) → PUSH(backStack)
│ │ → Display Animation
│ │
│ ├─→ current → Display backStack.top()
│ │
│ ├─→ history → Extract both stacks
│ │ → Display visualization
│ │
│ └─→ quit → Exit program
│
└─→ Display Result & Repeat
```

Stack State Transitions

Example: Visit → Back → Visit → Forward

Step 1: Initial (Homepage)
backStack: [Google]

Step 2: Visit GitHub
backStack: [Google, GitHub]
forwardStack: []

Step 3: Visit LeetCode
backStack: [Google, GitHub, LeetCode]
forwardStack: []

Step 4: Go Back
backStack: [Google, GitHub]
forwardStack: [LeetCode]

Step 5: Go Back Again
backStack: [Google]
forwardStack: [GitHub, LeetCode]

Step 6: Visit Stack Overflow (NEW VISIT)
backStack: [Google, StackOverflow]
forwardStack: [] ← CLEARED!

Step 7: Go Forward
Nothing happens (forwardStack is empty)

Screenshots (Placeholders)

Screenshot 1: Welcome Screen

```
══════════════════════════════════════════════
🕷️ STACK BASED BROWSER 🕷️
══════════════════════════════════════════════
🚀 Started at: Google Search Engine

══════════════════════════════════════════════
STACK BROWSER - LIVE OPERATIONS
══════════════════════════════════════════════

1️⃣ VISIT -> PUSH(backStack)
2️⃣ BACK <- POP(backStack) -> PUSH(forward)
3️⃣ FORWARD -> POP(forward) -> PUSH(backStack)
4️⃣ CURRENT -> backStack.top()
5️⃣ HISTORY -> Visualize Stack States
6️⃣ QUIT -> Exit Program
════════════════════════════════════════════
```

Screenshot 2: Stack Visualization (History)

```
══════════════════════════════════════════════
🔵 BACK STACK 🔵 | 🔴 FORWARD STACK 🔴
══════════════════════════════════════════════
🟢 LeetCode |
GitHub |
Google |
───────────────────────┼──────────────────────

🟢 TOP=Recent | Back: 3 | Forward: 0
```

Screenshot 3: Operation Animation

⬅️ GO BACK

🔧 LIVE STACK OPERATION:
⬅️ POP(backStack) + PUSH(forwardStack)

📤 POPPING ⬆️ ⬆️ ⬆️ ⬆️

✅ STACK OPERATION COMPLETED!

✅ BACK GitHub (https://github.com)

📊 STATUS: Back Stack(2) | Forward Stack(1) 📊

Screenshot 4: Current Page Display

📍 CURRENT GitHub (https://github.com)

📊 STATUS: Back Stack(2) | Forward Stack(1) 📊

Conclusion

Project Summary

The Stack-Based Browser History System successfully demonstrates the practical application of stack data structures in solving real-world problems. Through this implementation, we created a functional browser navigation system with the following achievements:

Technical Achievements:

- Successfully implemented two interacting stacks for managing browser history
- Achieved O(1) time complexity for all core operations
- Created an interactive, animated console user interface
- Implemented proper error handling and boundary validation
- Handled multiple input formats for user convenience

Learning Outcomes:

- Deep understanding of LIFO (Last-In-First-Out) principle
- Experience with C++ STL containers (stack, vector)
- Knowledge of state management and design patterns
- Skills in creating interactive console applications
- Practice in writing clean, documented code

Key Insights

1. **Simplicity with Power:** Stack is a simple data structure (just two operations), yet it solves complex problems elegantly
2. **Two Stacks Strategy:** Using two stacks together creates a powerful tool for handling forward/backward navigation
3. **State Management:** Browser history is essentially managing two different states (past and future)
4. **User Feedback:** Visual animations and clear status messages improve user experience significantly

---

Future Enhancements

Short-Term Improvements

1. Persistent Storage

- Save browsing history to a file
- Load previous sessions
- Implement history expiration (clear after N days)

2. Enhanced Title Management

- Use actual web scraping to fetch page titles
- Cache titles for faster retrieval
- Support for custom title editing

3. Search Functionality

- Search through history by URL or title
- Jump directly to a specific page
- Filter pages by date or domain

4. Statistics and Analytics

- Track most visited pages
- Display time spent on each page
- Generate usage reports

Medium-Term Improvements

5. Advanced Navigation

- Bookmark frequently visited pages
- Organize pages into categories
- Create shortcuts for favorite sites
- Support multiple tabs/windows

6. GUI Interface

- Transition from console to graphical interface
- Use Qt or SFML for cross-platform GUI
- Drag-and-drop page management
- Interactive timeline visualization

7. Web Integration

- Connect to actual web APIs
- Fetch real page metadata
- Support URL validation
- Integration with search engines

Long-Term Enhancements

8. Performance Optimization

- Implement LRU (Least Recently Used) cache for memory management
- Use more efficient data structures (deques instead of stacks)
- Parallel processing for multiple tabs
- Compression of stored page data

9. Advanced Features

- Session recovery and crash protection
- Synchronized history across devices
- Privacy modes and history clearing
- Extension/plugin support
- Page loading progress tracking

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[2] Stroustrup, B. (2013). *The C++ Programming Language* (4th ed.). Addison-Wesley.

[3] GeeksforGeeks. (2024). Stack Data Structure. https://www.geeksforgeeks.org/stack-data-structure/

[4] LeetCode. (2024). Design Browser History. https://leetcode.com/problems/design-browser-history/

[5] Cplusplus.com. (2024). Stack Container Documentation. https://www.cplusplus.com/reference/stack/stack/

[6] Karumanchi, N. (2016). *Data Structures and Algorithms Made Easy* (5th ed.). CareerMonk Publications.

[7] The Algorithms Project. (2024). C++ Stack Implementation. https://github.com/TheAlgorithms/C-Plus-Plus

[8] Stack Overflow. (2024). How to Clear Console in C++. https://stackoverflow.com/questions/6486289

[9] Google Code Archive. (2024). C++ Style Guide. https://google.github.io/styleguide/cppguide.html

[10] Sutter, H., & Alexandrescu, A. (2004). *C++ Coding Standards: 101 Rules, Guidelines, and Best Practices*. Addison-Wesley.

[11] ISOC++ Standards Committee. (2020). C++ Standards Library Reference. https://en.cppreference.com/w/cpp/header

[12] Knuth, D. E. (1997). *The Art of Computer Programming: Fundamental Algorithms* (3rd ed.). Addison-Wesley.