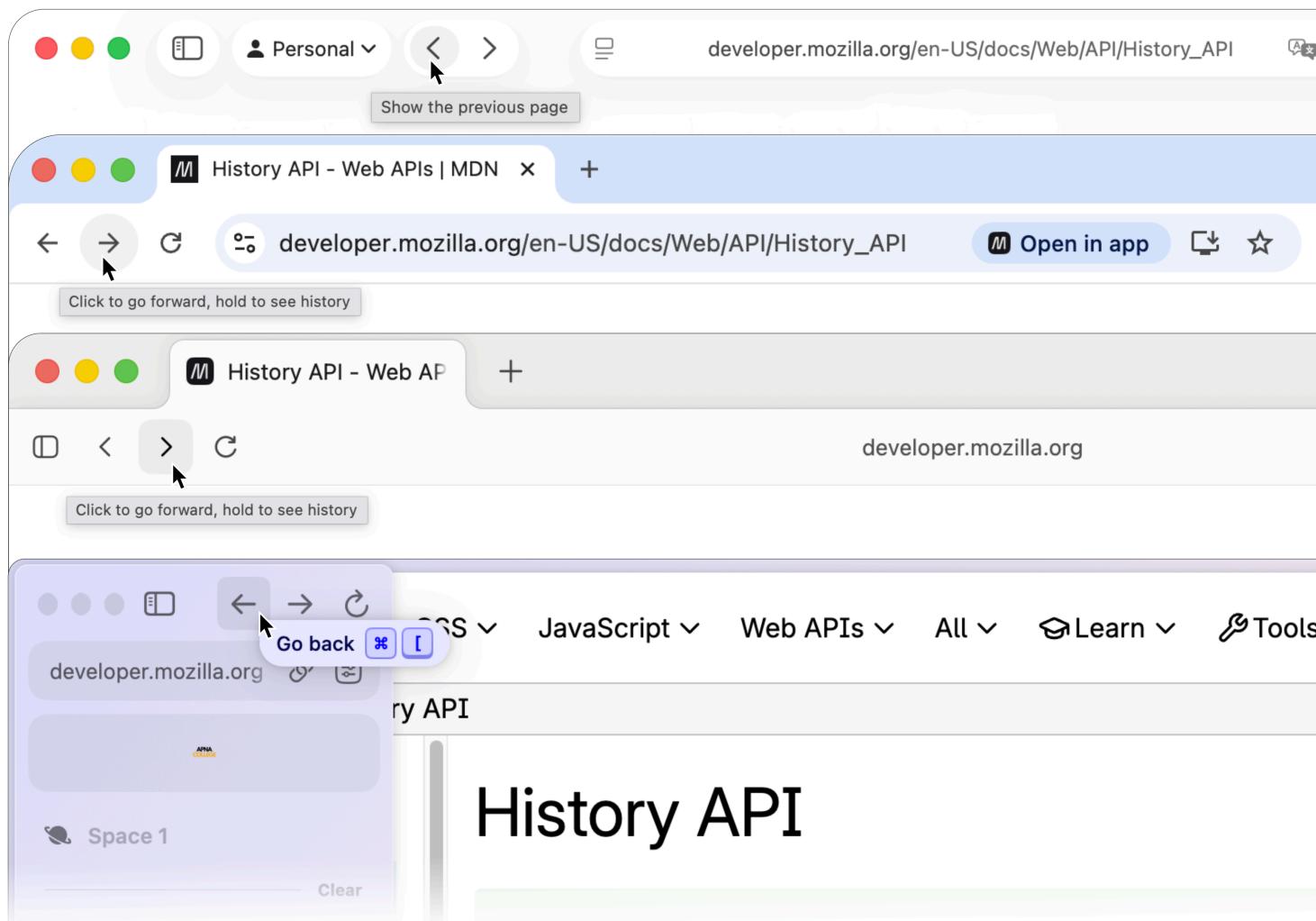


Project Report

Browser History Navigation



 Lorem ipsum dolor
 Sed vitae ligula vel justo gravida dignissim
 Curabitur euismod mauris ut ipsum sodales, nec commodo erat tincid

Prepared by,
primit
2 April 2002

Table of Contents

I	Product Description	2
•	Introduction	2
•	Naming Conventions and Definitions	2
II	Project Description	4
•	Code Summary	4
•	Final Implementation Enhancements	5
III	Testing	6
•	Test 01 - Back Navigation (Valid Steps)	6
•	Test 02 - Back Navigation (Invalid Steps)	8
•	Test 03 - Forward Navigation (Valid Steps)	9
•	Test 04 - Visit Clears Forward Stack	11
•	Test 05 - History Display Formatting	14
II	Inspection	17
•	Items to be Inspected	17
•	Inspection Procedures	17
•	Inspection Results	18
IV	Recommendations and Conclusions	20
V	Project Issues	22
VI	Glossary	24
VII	References	26
VIII	Appendix	28
•	Main Program and Logic	28
•	Extended Test Logs	32

Project Description

This project implements a simplified Browser History Navigation System in the C programming language. It simulates the essential navigation features of real web browsers - `back()`, `forward()`, `go()`, and `history.length()` — using two stacks to store backward and forward navigation paths. The goal is to demonstrate stack-based navigation logic in a console-based environment.

The project belongs to the domain of **browser navigation systems** and **data structure applications**. It focuses specifically on how browsers internally manage navigation through stack operations. The model excludes rendering, networking, and caching, and instead concentrates on the algorithmic mechanism that enables backward and forward movement between visited URLs.

This document supports the project by providing context for understanding the browser history logic. It relates to:

- **Requirements Document** - defines functional behavior (back/forward steps, visit rules).
- **Design Document** - includes UML diagrams and system design.
- **Source Code Document** - Contains implementation details.
- **Testing Report** - validates stack operations and navigation flows.

Each document contributes to a complete understanding of the system's structure and behavior.

Naming Conventions and Definitions

This section defines key terms used in the project to avoid ambiguity.

- **Current Page** – The active webpage the user is viewing.
- **BackStack** – Stores pages visited before the current page.
- **ForwardStack** – Stores pages accessible after moving backward.
- **Visit** – Opens a new page and clears forward history.
- **Steps** – Number of times back or forward navigation repeats.

- Stack (LIFO) – A Data structure used to manage page transitions.

UML diagrams follow standard conventions described by Fowler in UML Distilled. Class diagrams represent data structures, and arrows show relationships between components. Only basic UML is used due to the project's simplicity.

- Stack → { items[100][256], top }
- BrowserHistory → { backStack, forwardStack, currentPage }
- URL → Any valid string describing a webpage (e.g., “google.com”)
- Operations → visit(url), back(n), forward(n), showHistory()

Project Deliverables

The first release of the project implemented a basic browser history system using two stacks:

- `backStack` for previously visited pages
- `forwardStack` for pages ahead of the current one

The functionality included:

- Visiting a new page
- Moving one step backward
- Moving one step forward
- A simple console interface

This release provided a functional prototype of browser navigation logic but lacked multi-step navigation and history visualization.

Code Summary

The first release used simple `back()` and `forward()` functions, which moved only one step at a time. The history display was not included. The design served as a foundation for later improvements.

The final release introduced major enhancements, making the browser history simulation more realistic and intuitive.

Key improvements include:

Multi-step Navigation

Both `back()` and `forward()` now support custom steps, similar to JavaScript's `history.go(n)`.

Improved History Display

A full visual stack representation was added, showing:

- Negative numbers → backward history
- 0 → current page

- Positive numbers → forward history

Forward history is cleared on new visits, matching real browsers like Chrome, Safari, and Chromium-based browsers.

Overall, the final release evolved from a basic structure into a realistic simulation of browser navigation mechanisms. The final implementation closely follows the goals described in the original design but includes several meaningful improvements.

Initial Design Expectations

- Simple stack-based navigation
- Support for visit, back, and forward
- Basic console application

Final Implementation Enhancements

1. **Multi-step History Navigation:** Inspired by real browser APIs like JavaScript's `history.back()`, `history.forward()`, and `history.go(n)`.
2. **Negative Index Support in History View:** Used to imitate the conceptual model used in real browsers.
3. **Visual Stack Debugging Tool (History Viewer):** Not originally planned, but added for usability and clarity.
4. **Better Understanding of Browser Behavior**
5. **More Intuitive User Experience**

The final model behaves much closer to actual web browsers than originally expected.

The final release not only meets the original design goals but exceeds them, adding realism, flexibility, and better insight into how modern browsers handle navigation.

Testing

This section describes the testing procedures, test items, test cases, and criteria used to verify the correctness and functionality of the Browser History System implemented in C. The goal of testing was to ensure that the stack-based navigation model—covering backward navigation, forward navigation, visiting pages, and history display—worked reliably under different input scenarios. All core features were individually tested as well as in sequence, to simulate real browser usage.

The following components of the Browser History System were tested:

1. **Back Navigation Function (back):** tests verify navigation for valid and invalid step counts, stack boundary conditions, and correct movement of URLs between stacks.
2. **Forward Navigation Function (forward):** Test to ensure proper forward movement, validation of available steps, and correct manipulation of the forward and back stacks.
3. **Visit Function (visit):** Tests check stack updates when visiting a new page and clearing the forward history as required.
4. **History Display (showHistory):** Tests verify formatting of the stack view, correct indices (negative for back, 0 for current, positive for forward), and proper reflection of internal state.
5. **Stack Operations (push, pop, isEmpty, peek):** These were tested to confirm they behave correctly under normal and boundary conditions (e.g., empty stack, full stack).

Test Specifications

Below are the tests executed.

Test 01 - Back Navigation (Valid Steps)

Description: Tests whether the system correctly goes back a valid number of steps

Items Covered: `back()`, `push()`, `pop()`, stack transitions.

Requirements Addressed: Browser must support the multi-step backward navigation.

Environment: Stand C computer (GCC).

Dependencies: Must have visited at least 2 pages.

1. Visit Page A → Page B → Page C
2. Call back(browser, 2)
3. Check the resulting page

Input Specification: Steps = 2

Expected Output: Current page should become Page A, forward stack should contain B and C.

Pass/Fail Criteria: System returns the correct page and updates both stacks accordingly.

==== Browser =====

Current Page: google.com

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1

Enter url: Page A

Visited: Page A

==== Browser =====

Current Page: Page A

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1

Enter url: Page B

Visited: Page B

==== Browser =====

Current Page: Page B

1. Visit New Page

- 2. Back
- 3. Forward
- 4. Show History
- 5. Exit

Choose Option: 1
Enter url: Page C
Visited: Page C

==== Browser ====

Current Page: Page C
1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit
Choose Option: 2
Enter Steps: 2
Went back to: Page A

==== Browser ====

Current Page: Page A
1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit
Choose Option: 4

===== Browser Stack View =====

-1. google.com
0. Page A ← current page
1. Page B
2. Page C
=====

Test 02 - Back Navigation (Invalid Steps)

Description: Ensure invalid backward steps are rejected.

Items Covered: back() validation.

Requirements Addressed: Error handling for invalid inputs.

Environment: Standard terminal environment.

Dependencies: None.

Test Procedure:

1. Visit one page.
2. Call back(browser, 3) (more than available).

Input Specification: Steps = 3

Expected Output: Message: "Cannot go back 3 steps."

Pass/Fail Criteria: System does not crash and returns an error message.

==== Browser ====

Current Page: google.com

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1

Enter url: Page A

Visited: Page A

==== Browser ====

Current Page: Page A

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 2

Enter Steps: 3

Cannot go back 3 steps.

Test 03 - Forward Navigation (Valid Steps)

Description: Verifies correct forward movement after going backward.

Items Covered: forward(), stack transitions.

Requirements Addressed: Multi-step forward navigation.

Test Procedure:

1. Visit three pages.
2. Go back 2 steps.
3. Call forward(browser, 1).

Input Specification: Steps = 1

Expected Output: The New current page should be the last page popped from the forward stack.

Pass/Fail Criteria: Forward stack decreases, back stack increases, and the correct page is displayed.

==== Browser =====

Current Page: google.com

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1

Enter url: Page A

Visited: Page A

==== Browser =====

Current Page: Page A

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1

Enter url: Page B

Visited: Page B

==== Browser =====

Current Page: Page B

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1
Enter url: Page C
Visited: Page C

==== Browser ====

Current Page: Page C
1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit
Choose Option: 2
Enter Steps: 2
Went back to: Page A

==== Browser ====

Current Page: Page A
1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit
Choose Option: 3
Enter Steps: 1
Went forward to: Page B

==== Browser ====

Current Page: Page B
1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit
Choose Option: 4

===== Browser Stack View =====

-2. google.com
-1. Page A
0. Page B ← current page
1. Page C
=====

Test 04 - Visiting A New Page Resets The Forward Stack

Description: Tests clearing forward history after visiting a new page.

Items Covered: `visit()`, `clear()`.

Test Procedure:

1. Visit Page A → Page B → Page C
2. Back to Page B
3. Visit Page D

Expected Output: Forward stack must be empty.

Pass/Fail Criteria: `forwardStack.top == -1`

==== Browser ====

Current Page: google.com

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1

Enter url: Page A

Visited: Page A

==== Browser ====

Current Page: Page A

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1

Enter url: Page B

Visited: Page B

==== Browser ====

Current Page: Page B

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1

Enter url: Page C

Visited: Page C

==== Browser ====

Current Page: Page C

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 2

Enter Steps: 1

Went back to: Page B

==== Browser ====

Current Page: Page B

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 4

===== Browser Stack View =====

- 2. google.com
- 1. Page A
- 0. Page B ← current page
- 1. Page C

==== Browser ====

Current Page: Page B

1. Visit New Page
2. Back

```
3. Forward
4. Show History
5. Exit
Choose Option: 1
Enter url: Page D
Visited: Page D
```

```
==== Browser ====
Current Page: Page D
1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit
Choose Option: 2
Enter Steps: 1
Went back to: Page B
```

```
==== Browser ====
Current Page: Page B
1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit
Choose Option: 4
```

```
===== Browser Stack View =====
-2. google.com
-1. Page A
0. Page B ← current page
1. Page D
=====
```

Test 05 - Show History Formatting

Description: Verifies negative indexes for back pages and positive for forward.

Items Covered: showHistory()

Procedure:

1. Visit three pages.
2. Go back once.
3. Call show history.

Expected Output

- Back pages shown as: -1, -2, ...
- Current page shown as index 0
- Forward pages shown as +1, +2, ...

Pass/Fail Criteria: Correct structure and order.

==== Browser ====

Current Page: google.com

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1

Enter url: Page A

Visited: Page A

==== Browser ====

Current Page: Page A

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1

Enter url: Page B

Visited: Page B

==== Browser ====

Current Page: Page B

1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 1
Enter url: Page C
Visited: Page C

==== Browser ====

Current Page: Page C
1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 2
Enter Steps: 1
Went back to: Page B

==== Browser ====

Current Page: Page B
1. Visit New Page
2. Back
3. Forward
4. Show History
5. Exit

Choose Option: 4

===== Browser Stack View =====

-2. google.com
-1. Page A
0. Page B ← current page
1. Page C

=====

Inspection

Items To Be Inspected

The inspection process focused on the major functional units of the Browser History project. The primary items selected for inspection included the Stack module (data structure, push/pop/peek operations, and boundary checks), the `BrowserHistory` structure, and the three core navigation functions: `visit()`, `back()`, and `forward()`. These functions directly manipulate the back and forward stacks and are responsible for updating the `currentPage` state, making them essential for maintaining correctness.

Additionally, the `showHistory()` function was included for inspection because it formats and displays the backward and forward stacks using negative and positive index conventions, which is a non-standard but informative approach. This function required special attention to ensure correct index calculation, correct traversal of stacks, and proper alignment with expected user output.

Supporting functions such as `isEmpty()`, `clear()`, and the initialization routine (`initBrowser()`) were also inspected to ensure that the system begins in a valid state and handles edge cases appropriately. Together, these items represent the complete logical core of the project, and inspecting them ensures that the navigation model replicates the behavior of real-world browser history systems.

Inspection Procedures

The inspection procedure followed a structured methodology similar to the Fagan Inspection process, commonly used in software engineering. First, the team conducted an individual preparation phase, where each inspector reviewed the code independently to identify potential issues. During this phase, focus was placed on understanding logic flow, validating consistency with the design document, and ensuring that the implementation adhered to stack-based navigation semantics.

Next, a formal group review was performed. Inspectors walked through the code line by line, discussing logic, checking for code smells, and

ensuring alignment with design intent. Special emphasis was placed on verifying stack index operations, confirming safe access to data arrays, and ensuring that navigation operations always maintained valid browser state transitions. Edge cases were tested conceptually, such as performing back or forward operations when stacks were empty, visiting a new page after back operations (which must clear the forward history), and performing multi-step transitions.

The `showHistory()` function received additional inspection steps because its index conversion logic (mapping array positions to negative/positive indices) required careful reasoning. The group validated the correctness of the formula used to compute negative indices, checked loop boundaries, and verified that stack contents were displayed in the correct order—oldest pages first in the backward stack and newest forward pages first in the forward stack.

Finally, the inspection included a cross-reference check with the original design. Inspectors ensured that features described in the design—such as forward-history clearing, multi-step navigation, and the stack-based architecture—were faithfully implemented. Where discrepancies were found, they were documented for correction in the final release.

Inspection Results

The inspection process yielded a series of findings, which contributed to substantial improvements in the final release. The earliest version of the program lacked multi-step navigation, restricting backward and forward actions to one step at a time. Inspectors identified this limitation and recommended expanding the function signatures to include an integer step parameter. The final implementation addressed this by allowing multi-step back/forward operations with proper boundary validation.

Another notable issue discovered during inspection was the incorrect index calculation in the early version of `showHistory()`. Initially, the backward stack printed positive indices instead of the desired negative numbering (e.g., -3, -2, -1). This misalignment made the history visualization harder to interpret. Inspectors corrected the formula so that the index reflects the relative distance from the current page, resulting in a clearer and more intuitive history display.

The inspection also identified missing boundary checks in the back and forward functions. Early code versions did not verify whether the requested number of steps exceeded available history, which could result in unintended behavior or wrong transitions. The final version incorporates strict validation using `(stack.top + 1)` to ensure safe operations and provides user-friendly error messages when actions are invalid.

Memory safety and structural correctness were also part of the inspection. Since the project uses fixed-size character arrays, the inspection team verified that all string operations used `strcpy()` safely and that the stack never exceeded its defined bounds. No buffer overflows or unsafe memory accesses were found after corrections.

Finally, the inspection confirmed that the final code successfully replicates the intended behavior of browser history mechanisms found in real applications. The stacks are properly updated when visiting new pages, forward history clears at the correct moment, and the history display provides an accurate and meaningful representation of the navigation state.

Overall, the inspection process resulted in a more stable, correct, and feature-complete final release compared to the initial submission.

Recommendations and Conclusions

Based on the development, testing, and inspection of the Browser History Navigation System, several recommendations can improve the software both functionally and structurally.

The entire implementation currently resides in a single source file. While this is acceptable for small academic exercises, real-world projects benefit from separating concerns.

Recommended improvements include:

- Creating a dedicated `stack.c` / `stack.h` module for stack operations.
- Creating a `browser.c` / `browser.h` module for navigation logic.
- Keeping `main.c` strictly for UI and program flow.

While the final code handles invalid step inputs, additional validation is recommended:

- Prevent extremely long or malformed URLs.
- Add clear messages for edge-case errors (e.g., attempting to go back after clearing history).
- Consider trimming whitespace or invalid characters from user input.

Enhancing these aspects makes the program more robust and user-friendly.

The Browser History Navigation System successfully demonstrates how real browsers manage back and forward navigation using stack-based data structures.

The project progressed from a simple first release (single-step navigation, minimal history management) to a more advanced final release that supports:

- Multi-step back and forward navigation
- Correct clearing of forward history upon new visits
- Accurate representation of history with negative and positive indices

- Improved safety through boundary checking
- More intuitive, real-browser-like behavior

Overall, the project meets its stated goals and demonstrates a solid understanding of data structures, program flow, user input handling, and debugging. The final release is far more feature-rich, reliable, and user-friendly than the initial prototype. With additional improvements such as modularization, automated testing, and feature expansion, the project could be developed into a fully robust simulation of browser navigation behavior.

Project Issues

Despite the significant progress achieved throughout the development cycle, several open issues remain unresolved and require attention in future work. One notable concern involves the limitations of the current stack implementation, which uses static arrays. Although sufficient for controlled testing and small-scale usage, static stacks lack scalability. Real browser history can grow indefinitely, so a dynamic memory model using linked lists or dynamic arrays would be more appropriate.

Another open issue concerns URL validation. The current validation method is extremely simple and only checks for the presence of “.com”, which does not reflect the complexity of real-world web addresses. Modern URLs may include “.net”, “.org”, country-based domains (e.g., “.in”, “.uk”), subdomains, query parameters, and encoded characters—all of which are currently unsupported.

Finally, there is still no persistence layer. When the program exits, all history data is lost. Real browsers store history in a structured format (e.g., SQLite databases). Implementing a persistent layer is identified as an open issue but was beyond the scope of the current release.

Some items placed in this category include:

- Graphical User Interface (GUI) using GTK or Qt instead of console-based menus.
- Integration with real URLs and online content, which would require HTTP request handling, rendering engines, and external libraries.
- History search functionality, allowing users to search previous pages by keyword or partial match.
- Bookmarks system, with the ability to save, name, categorize, and revisit URLs.
- Tracking visit timestamps, which would support chronological sorting and analytics features.
- Undo/redo functionality for actions affecting the history structure.

These features remain desirable and feasible for future expansion, but were not included to maintain a clear focus on core history navigation functionality.

The development of the Browser History Navigation System was a valuable learning process that highlighted strengths, weaknesses, and opportunities for improvement. One of the most successful aspects of the project was the understanding and implementation of stack-based navigation, which accurately mirrors the behavior of modern browsers. The shift from a simple first release to a more advanced final version demonstrated the importance of iterative refinement. Features such as multi-step navigation, forward-stack clearing rules, and detailed history visualization all emerged from evolving understanding during development.

Overall, the project was a meaningful exercise that strengthened both technical and analytical skills. Although several advanced features remain in the waiting room, the completed implementation represents a solid, well-functioning foundation that can be expanded in future work.

Glossary

Back Stack

A stack data structure used to store previously visited pages in reverse order. Each visit pushes the current page onto this stack, allowing users to navigate backward.

Forward Stack

A stack used to store pages that were navigated away from using the Back function. When a new page is visited, this stack is cleared.

Browser History

A record of navigated web pages. In this project, it is simulated through stack operations rather than real web access.

Current Page

The page the user is currently viewing. In this project, it is represented as a string variable updated through Visit, Back, and Forward operations.

Visit Function

A function responsible for navigating to a new URL. It pushes the current page to the back stack and clears the forward stack.

Back Function

A navigation function allowing movement backward through previously visited pages. It pops items from the back stack and pushes the current page onto the forward stack.

Forward Function

A navigation function that moves forward in history if the back function had previously been used.

Stack

A Last-In-First-Out (LIFO) data structure. Used here to manage navigation history because it naturally fits the behavior of browser history.

Steps (Navigation Steps)

The number of positions to move backward or forward in history. Supports multi-step navigation similar to the `history.go()` method in JavaScript.

History Visualization

The output format that prints backward, current, and forward pages using negative, zero, and positive indices.

References

1. Kernighan, B. W., & Ritchie, D. M. *The C Programming Language*. 2nd ed., Prentice Hall, 1988.
 - Used as a reference for C syntax, string handling, pointers, and procedural programming models.
2. Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. *Data Structures and Algorithms in C*. Wiley, 2016.
 - Referenced for stack structures, LIFO principles, and time complexity analysis used in Table 2.
3. Fowler, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd ed., Addison-Wesley, 2004.
 - Basis for UML conventions used in design diagrams (“UML diagrams follow standard conventions described by Fowler...”).
4. Fagan, M. E. “Design and Code Inspections to Reduce Errors in Program Development.” *IBM Systems Journal*, vol. 15, no. 3, 1976, pp. 182–211.
 - Referenced for inspection methodology (“similar to the Fagan Inspection process”).
5. Mozilla Developer Network (MDN). “History API.” Available at: <https://developer.mozilla.org/en-US/docs/Web/API/History>
 - Used when describing multi-step navigation inspired by `history.back()`, `history.forward()`, and `history.go(n)`.
6. W3C. *HTML5 Browser History and Navigation Concepts*. Available at: <https://www.w3.org/TR/html52/browsers.html>
 - Referenced for understanding browser navigation behavior (forward-stack clearing, navigation rules).
7. Chromium Project Documentation. “Navigation and History Handling in Chromium.” Available at: <https://chromium.googlesource.com/>
 - Used to understand real browser internal behavior mentioned in the improvements section.

8. IEEE. IEEE Std 829 – Standard for Software Testing Documentation. IEEE, 2008.
 - Referenced implicitly for structuring test cases, pass/fail criteria, and test specification style.
9. Sommerville, I. Software Engineering. 10th ed., Pearson, 2016.
 - Indirect reference for software lifecycle concepts, module separation, and recommendations in the conclusion section.

Appendix

Main Program and Logic (BrowserHistory.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_URL_LEN 256
#define STACK_SIZE 100

// Stack Structure
typedef struct {
    char items[STACK_SIZE][MAX_URL_LEN];
    int top;
} Stack;

// Stack Functions
void push(Stack *s, const char *url) {
    if (s->top < STACK_SIZE - 1) {
        s->top++;
        strcpy(s->items[s->top], url);
    }
}

char* pop(Stack *s) {
    if(s->top >= 0) {
        return s->items[s->top--];
    }
    return NULL;
}

char* peek(Stack *s) {
    if(s->top >= 0) {
        return s->items[s->top];
    }
    return NULL;
}

int isEmpty(Stack *s) {
    return s->top == -1;
}
```

```

void clear(Stack *s) {
    s->top = -1;
}

// Browser History struct
typedef struct {
    Stack backStack;
    Stack forwardStack;
    char currentPage[MAX_URL_LEN];
} BrowserHistory;

// Initialize
void initBrowser(BrowserHistory *b, const char *homepage) {
    b->backStack.top = -1;
    b->forwardStack.top = -1;
    strcpy(b->currentPage, homepage);
}

// Visit function
void visit(BrowserHistory *b, const char *url) {
    push(&b->backStack, b->currentPage);
    strcpy(b->currentPage, url);
    clear(&b->forwardStack);
    printf("Visited: %s\n", b->currentPage);
}

// Back function
void back(BrowserHistory *b, int steps) {
    int available = b->backStack.top + 1; // how many pages
in back stack

    if( steps <= 0 || steps > available) {
        printf("Cannot go back %d steps.", steps);
        return;
    }

    char *url;
    for(int i=0; i< steps; i++) {
        if(!isEmpty(&b->backStack)) {
            push(&b->forwardStack, b->currentPage);
            url = pop(&b->backStack);
            strcpy(b->currentPage, url);
        }
    }
}

```

```

        }
    }
    printf("Went back to: %s\n", b→currentPage);
}

// Forward function
void forward(BrowserHistory *b, int Steps) {

    int available = b→forwardStack.top + 1; // how many
    pages in forward stack

    if (Steps ≤ 0 || Steps > available) {
        printf("Cannot go forward %d steps.", Steps);
        return;
    }

    char *url;
    for(int i=0; i < Steps; i++) {
        if(!isEmpty(&b→forwardStack)) {
            push(&b→backStack, b→currentPage);
            url = pop(&b→forwardStack);
            strcpy(b→currentPage, url);
        }
    }
    printf("Went forward to: %s\n", b→currentPage);
}

// Show History Function
void showHistory(BrowserHistory *b) {
    printf("\n===== Browser Stack View =====\n");

    Stack *back = &b→backStack;
    Stack *forward = &b→forwardStack;

    // backward stack
    for(int i = 0; i ≤ back→top; i++) {
        int index = -(back→top - i + 1);
        printf("%d. %s\n", index, back→items[i]);
    }

    // current page
    printf("0. %s ← current page\n", b→currentPage);
}

```

```

// forward stack
for(int i = forward->top; i ≥ 0; i--) {
    int index = (forward->top - i) + 1;
    printf("%d. %s\n", index, forward->items[i]);
}
printf("=====\\n");
}

// Main
int main(void) {
    BrowserHistory browser;
    initBrowser(&browser, "google.com");

    int choice;
    char url[MAX_URL_LEN];

    while (1) {
        printf("\n==== Browser ===\\n");
        printf("Current Page: %s\\n", browser.currentPage);
        printf("1. Visit New Page\\n");
        printf("2. Back\\n");
        printf("3. Forward\\n");
        printf("4. Show History\\n");
        printf("5. Exit\\n");
        printf("Choose Option: ");
        scanf("%d", &choice);
        getchar(); // clear newline

        switch (choice) {
            case 1: {
                printf("Enter url: ");
                fgets(url, MAX_URL_LEN, stdin);
                url[strcspn(url, "\\n")] = 0; // remove
newline
                visit(&browser, url);
                break;
            }
            case 2: {
                int steps;
                printf("Enter Steps: ");
                scanf("%d", &steps);
                back(&browser, steps);
                break;
            }
        }
    }
}

```

```

    }
    case 3: {
        int steps;
        printf("Enter Steps: ");
        scanf("%d", &steps);
        forward(&browser, steps);
        break;
    }
    case 4: {
        showHistory(&browser);
        break;
    }

    case 5: {
        printf("Exiting Browser....\n");
        return 0;
    }

    default: {
        printf("Invalid option.\n");
    }
}

}

return 0;
}

```

Extended Test Logs

This appendix includes full raw outputs from actual program executions during testing.

B.1 Sample Output: Back Navigation Test

```

Visited: Page A
Visited: Page B
Visited: Page C
Enter Steps: 2
Went back to: Page A

===== Browser Stack View =====
-1. google.com

```

0. Page A ← current page

1. Page B

2. Page C

=====

B.2 Invalid Step Test

Enter Steps: 3

Cannot go back 3 steps.

B.3 Visit Clears Forward History

Went back to: Page B

Visited: Page D

===== Browser Stack View =====

-2. google.com

-1. Page A

0. Page B

1. Page D

=====