PERIYAR MANIAMMAI
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University)
Established Under Sec. 3 of UGC Act, 1956 • NAAC Accredited
think • innovate • transform

TEAM - 3

PROJECT REPORT

TITLE: DESIGN BROWSER HISTORY

MEMBERS:

S.AFRIDHA SHAHEEN

M.THRISHA

E.RACHEL

1. REQUIREMENT ANALYSIS

PROBLEM STATEMENT :

We are designing and implementing a simple browser history like Chrome, supports the following operations:

1. Initialising with a homepage.
2. Forward the given number of steps in history
3. Backward the given number of steps in history
4. Visit a new URL from the current page [which clears all the forward history]

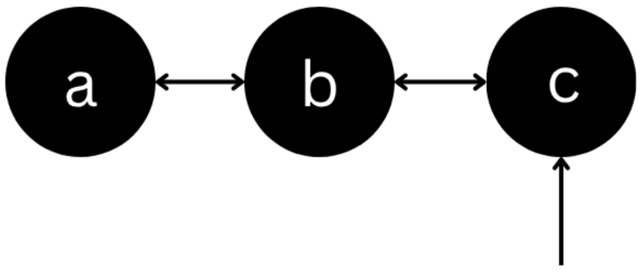
USE CASES :

1. **Initialize Browser History :** The browser starts with a new homepage, which becomes the first and the only page in the history.
2. **Go Back in History :**The user navigates backward by a specified number of steps given by the user. The system moves the current page pointer backward in history, unless the beginning of the history is reached.
3. **Go Forward in History :** The user navigates forward by a specified number of steps. The system moves the current page pointer forward in history, unless the end of the history is reached
4. **Visit a New Page :** The user visits a new website. The system adds this page to the history and clears any forward history that exists from previous navigation.
5. **Visit New Page After Going Back :** If the user goes back and then visits a new page, the system clears all forward history and adds the new page as the latest entry in history.

EDGE CASES:

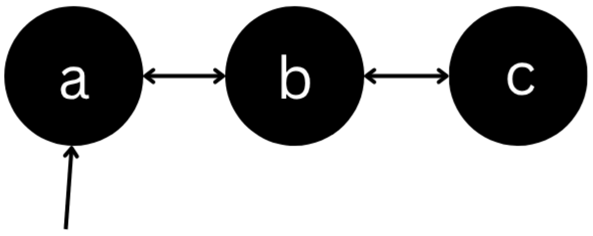
Back beyond the first page:

Since the homepage has no previous node (webpage), any attempt to go back beyond will be ignored. The current page remains the homepage in this case.



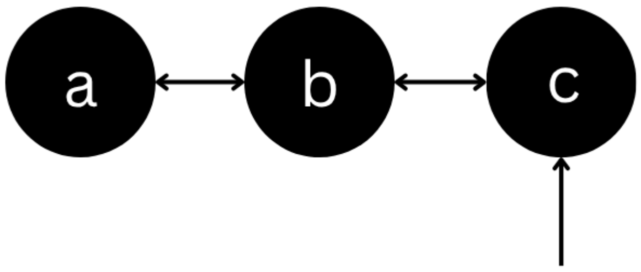
For example,

If we are currently in the webpage “c” and we instruct the system to go 3 steps back, It is beyond the first page “a”. So, the first page (homepage) will be returned.



Going Forward Beyond the Last Visited Page

Since it does not have any new nodes (in this case, webpages) when the user attempts to go forward many times beyond the last visited page, the last visited page is only returned.

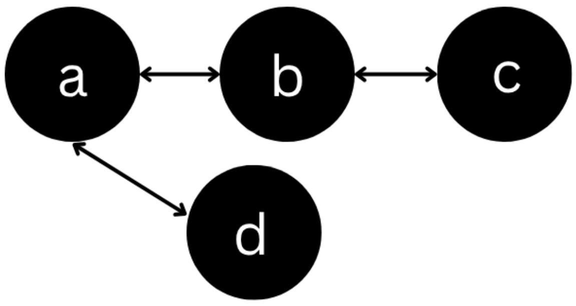


For example,

If we are currently in the webpage “b”, and the user commands to go forward multiple times, it only returns the last visited page, that is, the webpage “c”.

Visiting a new page

When a new page is visited from a current page, the other forward history is removed.



For example,

We are at “a”, we have the forward page’s history “b”,”c”. From here, if we try to move to a different page, we can move backward from webpage “d” to “a” and “a” to “d” but the pages “b” and “c” cannot be accessed.

2. SOLUTION APPROACH

Approach 1 : Using a Doubly Linked List

In this approach, we create a **doubly linked list** where each node represents a webpage. Each node contains:

1. The URL (data)
2. A pointer to the **previous** page
3. A pointer to the **next** page

We maintain a pointer temp to keep track of the page the user is currently

Pros:

- Dynamic size
- Efficient back and forward operations.

Cons:

- More complex to implement: Requires careful pointer handling and memory management.
- Memory overhead: Each node stores extra pointers (prev and next).
- Slower access: No direct indexing — need to traverse node by node.

Approach 2: Array/ Stack

- To solve the browser history problem conceptually using an array or stack, we treat the browsing experience as a sequence of page visits stored in order.
- Each time the user visits a new page, we add it to a history structure—like pushing onto a stack or appending to an array—and discard any forward history beyond the current page.
- The "back" operation is like popping or moving a pointer backward in the sequence, while the "forward" operation moves the pointer ahead, as long as forward pages exist.

Pros:

- Easy indexing using integers.
- Direct access to any history entry by index.
- Everything stays in continuous memory.

Cons:

- Fixed size
- Space reserved for unused future entries if forward history is cleared.

What we chose and Why?

Our Solution Approach is using a doubly linked list as it is:

- Efficient for both forward and backward operations.
- Can use pointers to point to the previous webpage and next webpage easily.
- It's easy to manage dynamic length history with linked nodes.

3. IMPLEMENTATION

The browser history is implemented using a **doubly linked list**. Each page is represented as a node, and the system maintains a pointer to the current page (temp). When a new page is visited, the current page's next link is updated to point to the new page, and any forward history is discarded.

Step by step implementation using doubly linked list

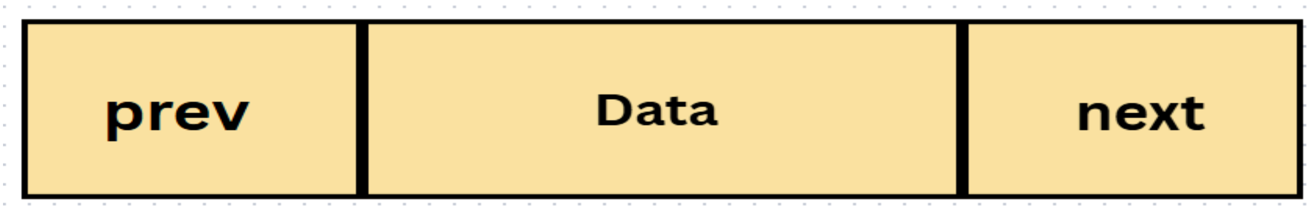
STEP 1: We Create Node Structure that contains

→ The Data part

→ The Pointer to the previous node (prev)

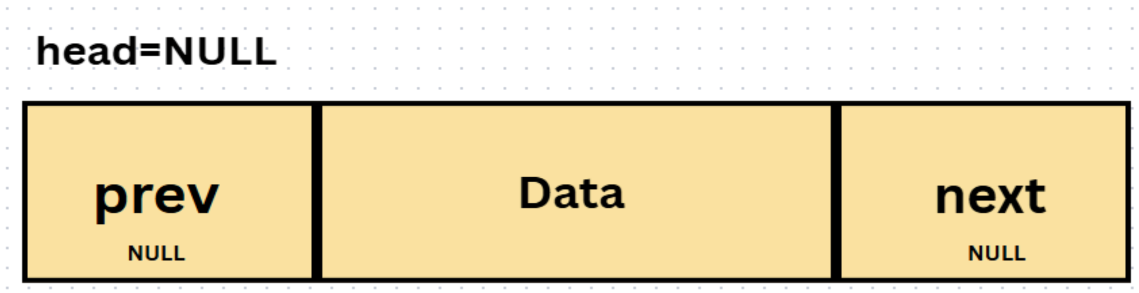
→ The Pointer to the next node (next)

And we also create a pointer called a head.

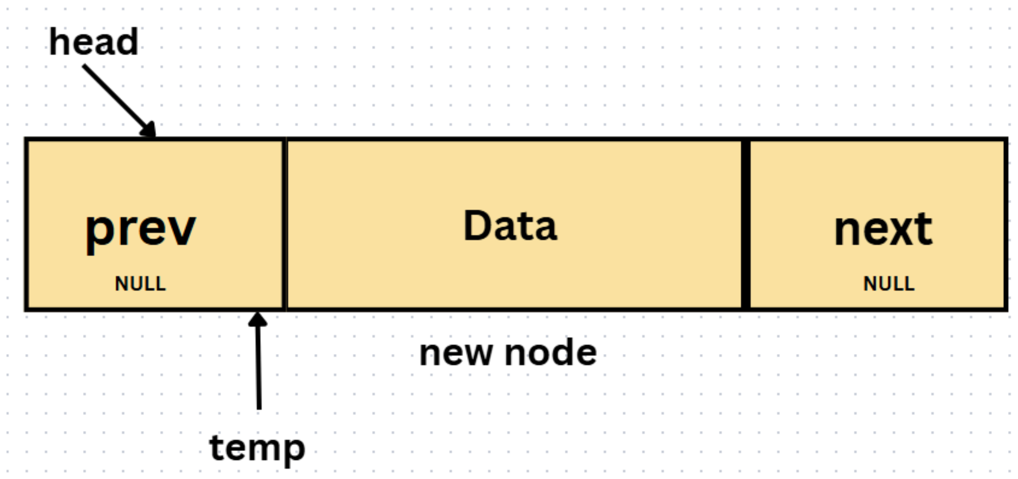


STEP 2 : Creating New Homepage Node

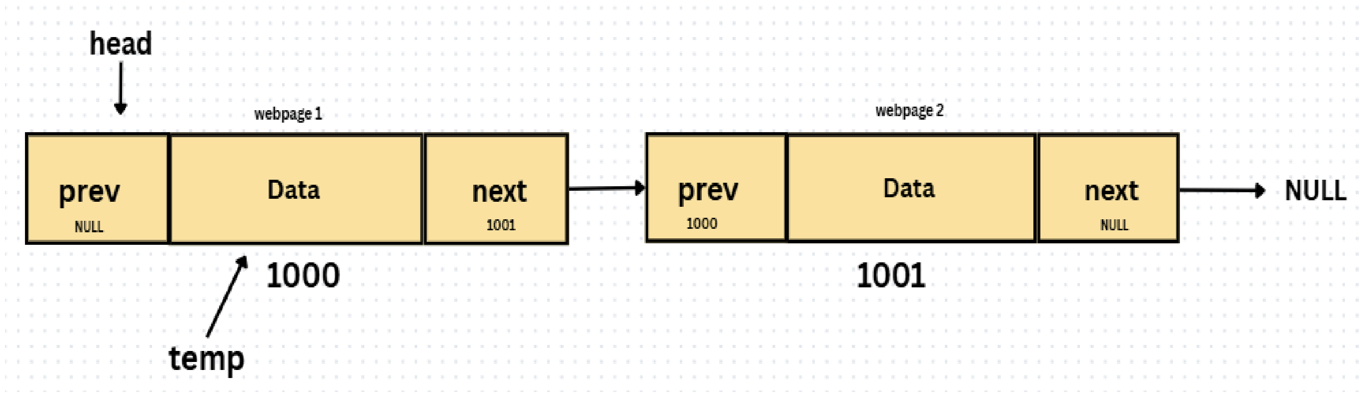
- For the first homepage node, we set the pointers - previous node (prev), head node (head) and the next node (next) as NULL.



- We create a temporary pointer that points to the current node. In this instance it points to the first node. (the new homepage). And the head also points to the first node.

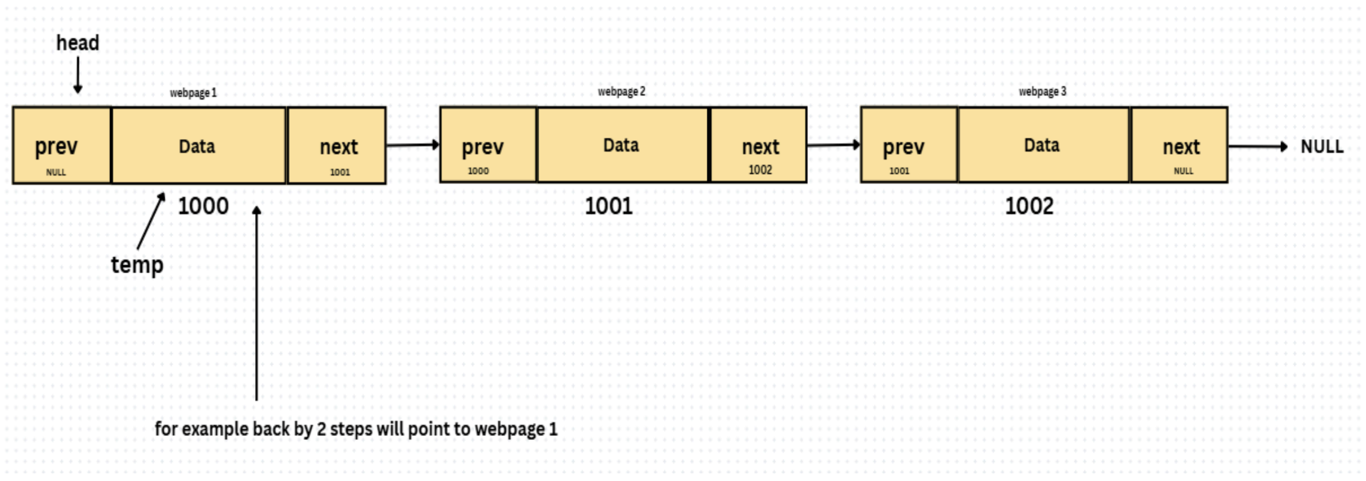


STEP 3 : Visiting a new node
We create a new node and set the temp node's next pointer to point to the new node and we make the previous node (prev) of the new node point to the current node. (temp).



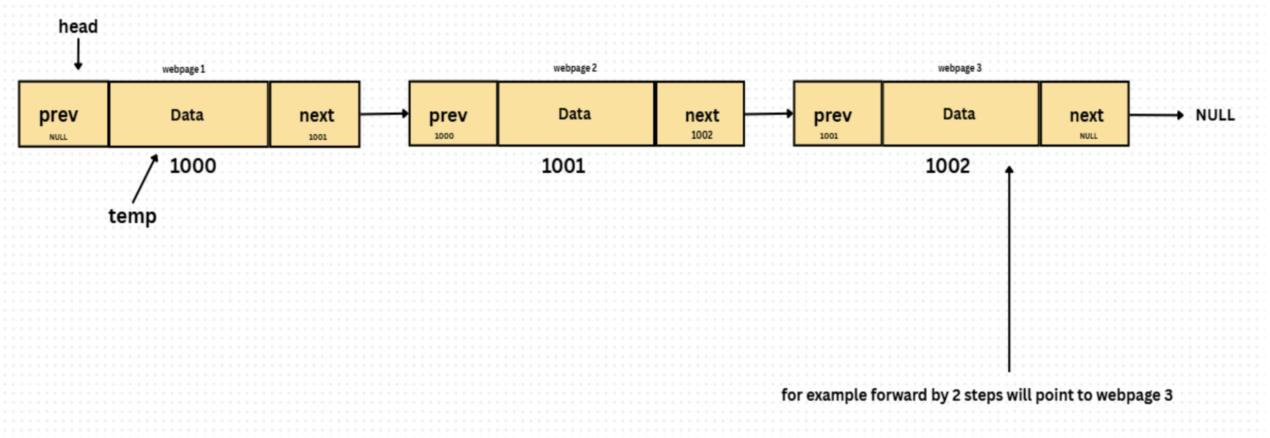
STEP 4 : We are creating a function to move backwards.

While the number of steps is greater than 0 and the previous pointer of the temp is not NULL, we make the temporary node (temp) point to the temporary node's previous node. This repeats until the last node is NULL.



STEP 5 : We are creating a function to move forward.

While the number of steps is greater than 0 and the previous pointer of the temp is not NULL, we make the temporary node (temp) point to the temporary node's next node. This repeats until the last node is NULL.



STEP 6 : Inside the main function

1. The user is first asked to enter the homepage URL
2. The user enters the command (visit,back,forward) along with the webpage URL

3. Based on the user’s commands,

- If the user commands to visit, the browser visits the URL and updates the history.
- If the command is back, it goes back to the specified number of steps in the history.
- If the command is forward, it moves forward in the history by the given number of steps

This continues until the user enters the exit.

Example:

1. Start at homepage → "leetcode.com"

2. Visit → "google.com" [history becomes: leetcode , google]

3. Visit → "facebook.com" [history: leetcode , google ,facebook]

3. Back(2) → current page is "leetcode.com"

4. Visit → "linkedin.com" [history: leetcode, linkedin]

(the previous forward pages "google.com" and "facebook.com" are erased)

4. TESTING AND VALIDATION

To ensure that the browser history works correctly, we manually tested all functions using different inputs.

Test Case 1: Basic Navigation

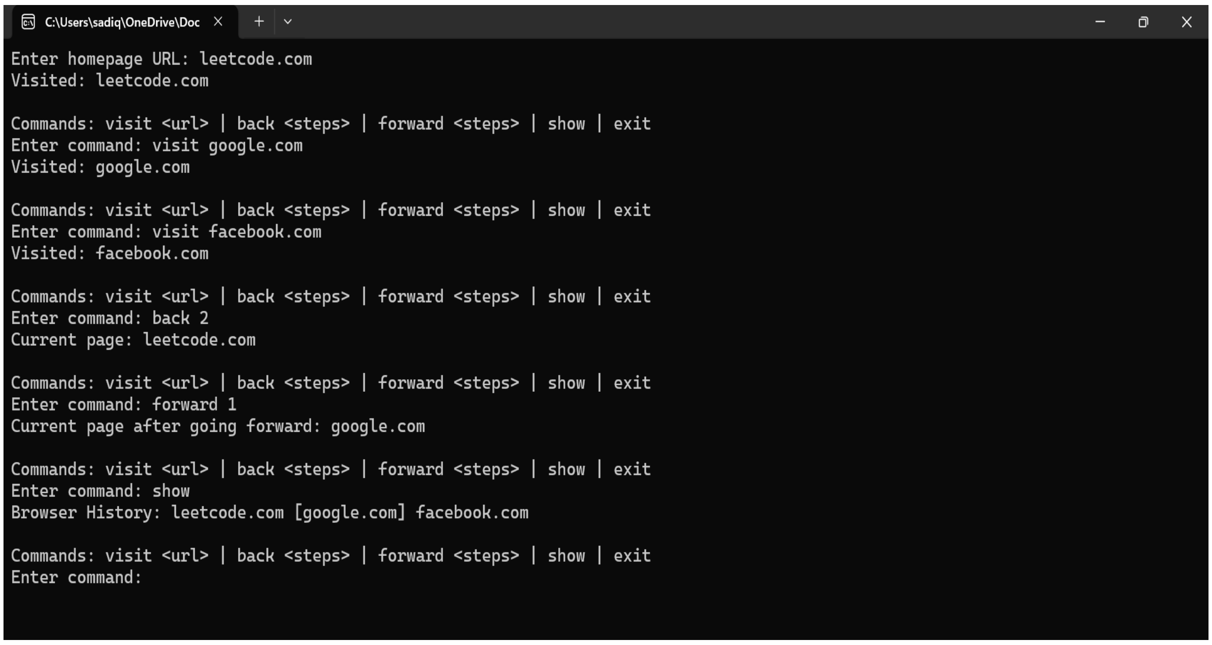
Input :

Enter homepage URL : leetcode.com
visit google.com
visit facebook.com
back 2
forward 1
show

Expected Output :

- Visited : leetcode.com
- Visited : google.com
- Visited : facebook.com
- Current page : leetcode.com (after "**Back 2**")
- Current page **:** google.com (after "**Forward 1**")
- Browser History : leetcode.com [google.com] facebook.com

Actual Output :



```
Enter homepage URL: leetcode.com
Visited: leetcode.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: visit google.com
Visited: google.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: visit facebook.com
Visited: facebook.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: back 2
Current page: leetcode.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: forward 1
Current page after going forward: google.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: show
Browser History: leetcode.com [google.com] facebook.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command:
```

"Tested and it matches the output as expected"

Test Case 2 : Visit After Back (It clears forward history)

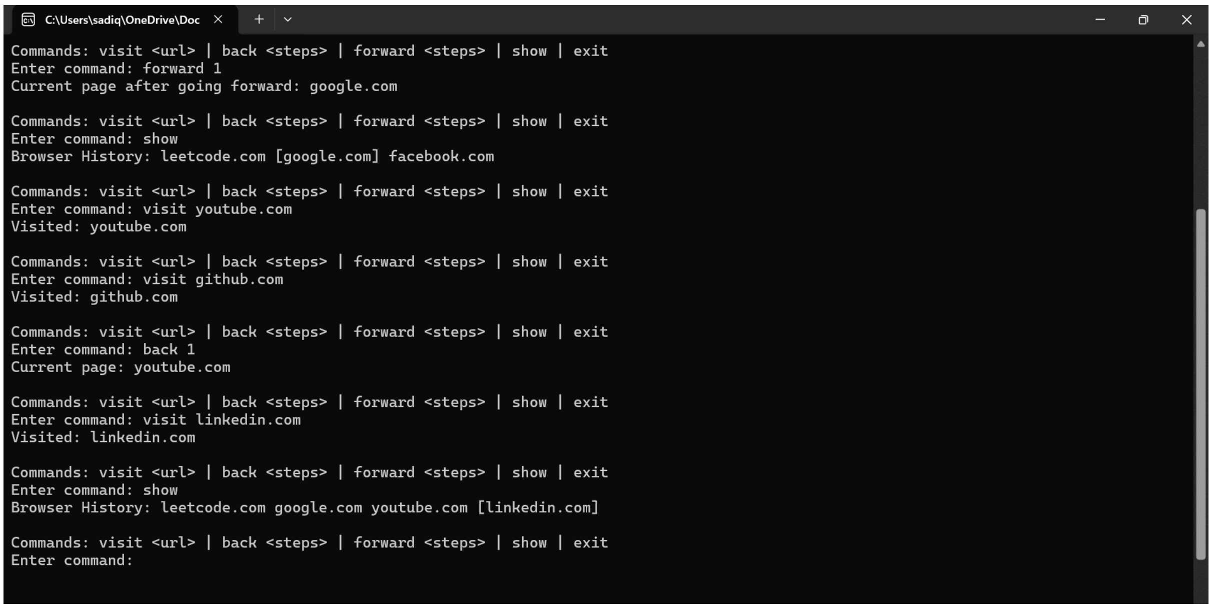
Input :

visit youtube.com
visit github.com
back 1
visit linkedin.com
show

Expected Output :

- Visited : youtube.com
- Visited : github.com
- Current page : youtube.com (after **"Back 1"**)
- Visited : linkedin.com
- Browser History : leetcode.com google.com youtube.com [linkedin.com]

Actual Output :



```
C:\Users\sadiq\OneDrive\Doc
Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: forward 1
Current page after going forward: google.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: show
Browser History: leetcode.com [google.com] facebook.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: visit youtube.com
Visited: youtube.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: visit github.com
Visited: github.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: back 1
Current page: youtube.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: visit linkedin.com
Visited: linkedin.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: show
Browser History: leetcode.com google.com youtube.com [linkedin.com]

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command:
```

"Tested and it matches the output as expected"

Test Case 3 : Back Beyond Homepage

Input :

back 5

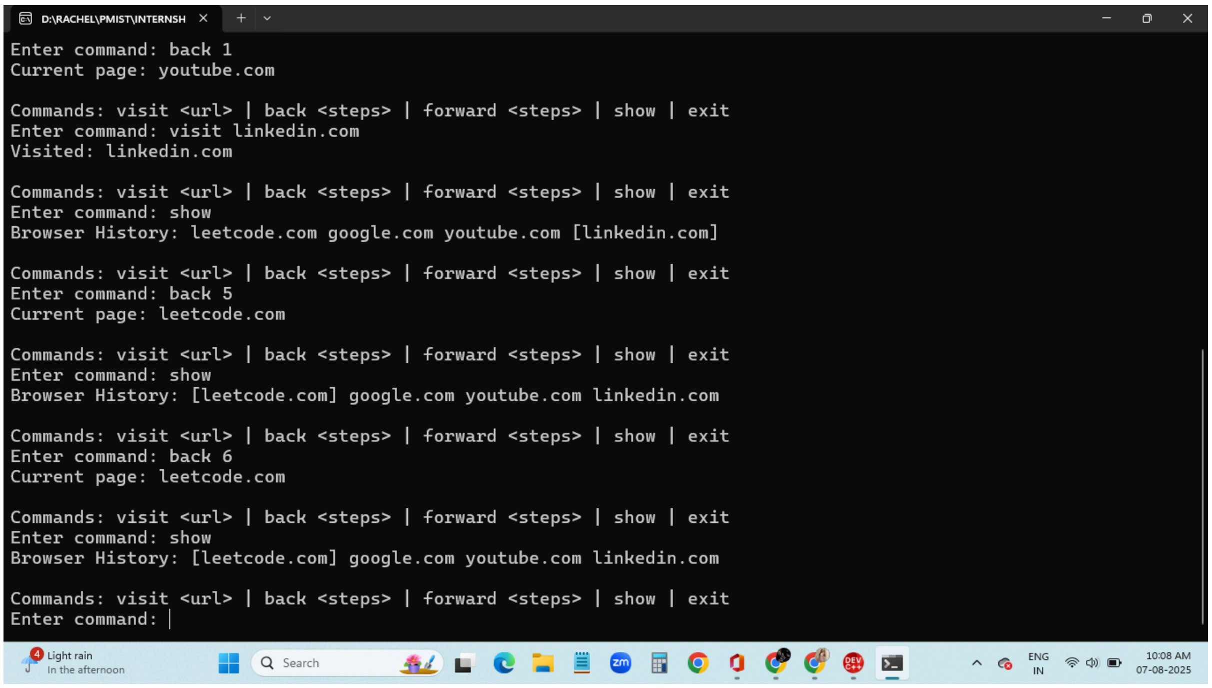
back 6

show

Expected Output :

- Browser History : leetcode.com
- It should stay at the Homepage URL
- Browser History : [leetcode.com] google.com youtube.com linkedin.com

Actual Output :



```
Enter command: back 1
Current page: youtube.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: visit linkedin.com
Visited: linkedin.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: show
Browser History: leetcode.com google.com youtube.com [linkedin.com]

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: back 5
Current page: leetcode.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: show
Browser History: [leetcode.com] google.com youtube.com linkedin.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: back 6
Current page: leetcode.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: show
Browser History: [leetcode.com] google.com youtube.com linkedin.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command:
```

"Tested and it matches the output as expected"

Test Case 4 : Forward Beyond Latest Page

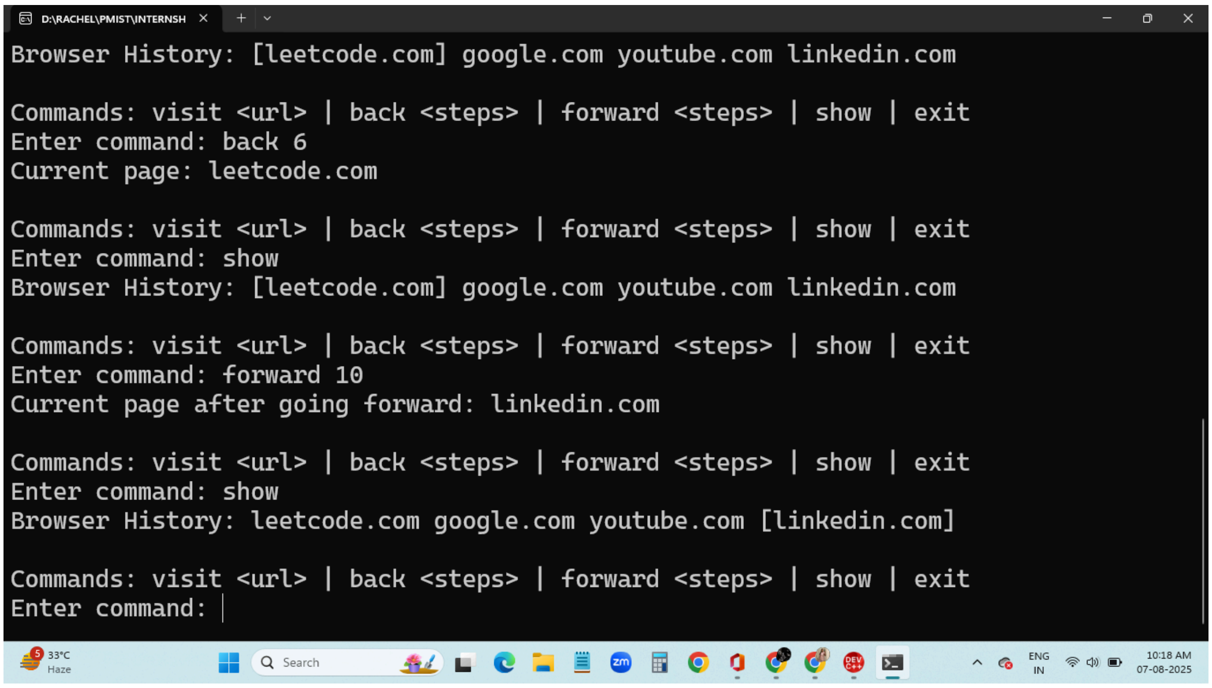
Input :

forward 10
show

Expected Output :

- Current page : linkedin.com
- Browser History : leetcode.com google.com youtube.com [linkedin.com]
- It should stay at last visited URL

Actual Output :



```
Browser History: [leetcode.com] google.com youtube.com linkedin.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: back 6
Current page: leetcode.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: show
Browser History: [leetcode.com] google.com youtube.com linkedin.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: forward 10
Current page after going forward: linkedin.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: show
Browser History: leetcode.com google.com youtube.com [linkedin.com]

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command:
```

"Tested and it matches the output as expected"

Test Case 5 : Back And Forward With 0 Steps

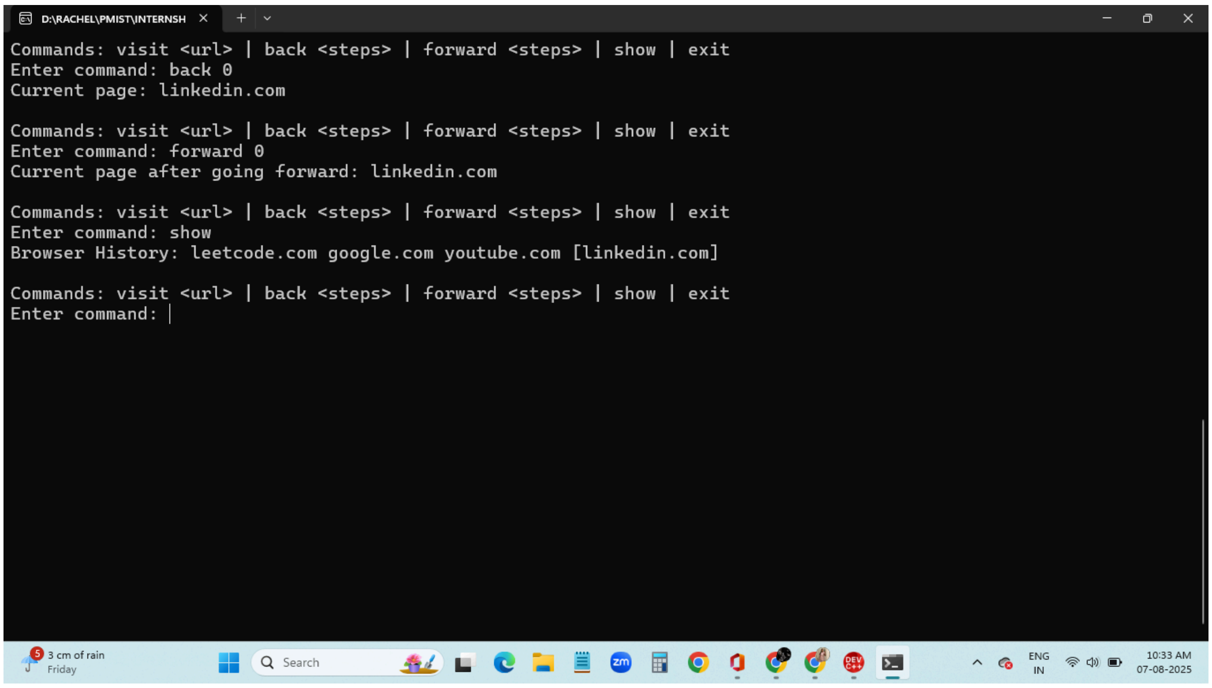
Input :

back 0
forward 0
show

Expected Output :

- Current Page : linkedin.com
- Current Page : linkedin.com
- Browser History : leetcode.com google.com youtube.com [linkedin.com]
- Stays at the same current URL page

Actual Output :



```
Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: back 0
Current page: linkedin.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: forward 0
Current page after going forward: linkedin.com

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command: show
Browser History: leetcode.com google.com youtube.com [linkedin.com]

Commands: visit <url> | back <steps> | forward <steps> | show | exit
Enter command:
```

"Tested and it matches the output as expected"

All test cases have passed successfully and it confirmed that the doubly linked list approach handles dynamic navigation, forward and backward moves and edge cases.

5. RESULTS AND EVALUATION

The implemented Browser History System met all the functional requirements and performed as expected under various testing scenarios. Use of doubly linked list allows efficient backward and forward navigation, and it dynamically manages the various commands involving different history sizes without degrading the performance.

- Backward and Forward navigation correctly followed the history list, with checks in place to avoid moving beyond the last node.
- When visiting a new page after going back, forward history is properly removed from the history.
- Giving commands like to go beyond the first or last page did not create any errors.
- Commands like **visit** , **back** , **forward** and **show** allows smooth and clear interaction with the browser history system.

Performance Analysis

Pros

- **Time Complexity**
  - ☐ We can go backward and forward quickly using **prev** and **next** pointers.
  - ☐ New pages are added easily in constant time without any time delay.
- **Space Complexity**
  - ☐ New pages are added easily without any limit on their size, due to the use of doubly linked lists.
  - ☐ So the program uses memory only when needed, which helps avoid wasting memory space.

Cons / Limitations

- **Time Complexity**
  - ☐ Going back or forward many steps takes more time, because it moves one step at a time using the linked list.
  - ☐ There is no direct way to jump to a specific page like in an array.
- **Space Complexity**
  - ☐ Each page uses more memory because it stores two pointers (prev and next) along with the URL

6. CONCLUSION

In this project, we successfully designed and implemented a simple **Design Browser History** using a doubly linked list. The system allows users to visit new pages, go back, go forward, and can view the full history.

Our implementations handled all the main operations efficiently. We have tested different cases, including the Edge Cases like going back beyond the first page or going forward beyond the last page. The system worked correctly in all scenarios.

Using a doubly linked list gave us the advantages of easy navigation in both directions (forward and backward) and dynamic memory usage. Even though the program has some limitations, it performs well and meets the goals of user needs.

This project helped us understand how real browser history works and how data structures like doubly linked lists can be used to manage dynamic operations.

7. REFERENCE

1. GeeksforGeeks. “Doubly Linked List in C++”
   https://www.geeksforgeeks.org/cpp/doubly-linked-list-in-cpp/
2. GeeksforGeeks. “Traversal in Doubly Linked List”
   https://www.geeksforgeeks.org/dsa/traversal-in-doubly-linked-list/
3. TutorialsPoint. “C++ to Implement Doubly Linked List”
   https://www.tutorialspoint.com/cplusplus-program-to-implement-doubly-linked-list
4. Youtube. “Introduction to Doubly Linked List”
   https://youtu.be/JdQeNxWCguQ?si=0-OR1DtXoA7qpmgZ
5. Youtube. “Doubly Linked List”
   https://youtu.be/1fc8FPm1new?si=U0SS5UWbCToFzWcm

8. APPENDIX

Main_DesignBrowserHistory

```
/*
 * Filename: Design_BrowserHistory.cpp
 *
 * Authors: Rachel, Afridha Shaheen & Thrisha
 *
 * Description: Implements a simple browser history system using a doubly linked list.
 *            The user can visit a new webpage, move backward and forward, and
 *            view the entire history.
 */

#include <iostream>                  // Header file for standard I/O operations
#include <string>                   // Header file for using string data type
using namespace std;                 // Standard namespace
#include "Node-Structure.cpp"                         //include node structure
#include "Visit-Newpage.cpp"          // Include visit function implementation
#include "Move-Backwards.cpp"         // Include backward movement function
#include "Move-Forward.cpp"           // Include forward movement function
#include "Show-History.cpp"                     // Include show function

/*
 * Main function to interact with the user and perform browser history operations.
 */
int main() {
   string homepage;                                       // Variable to store homepage URL
     cout << "Enter homepage URL: ";                                      // Prompt user to enter
homepage
   cin >> homepage;                                        // Read homepage URL
     visit_newWebpage(homepage);                                          // Call function to add
homepage

    string command;                                              // To store the command entered by
user
    while (true) {                                               // Infinite loop to keep the browser
running
         cout << "\nCommands: visit <url> | back <steps> | forward <steps> | show | exit" <<
endl;
       cout << "Enter command: ";                                // Prompt user for a command
       cin >> command;                                      // Read the command
```

```
        if (command == "visit") {                              // If user wants to visit a new page
            string url;                                     // Variable to store new URL
            cin >> url;                                      // Read URL from user
            visit_newWebpage(url);                                 // Call visit function
        }
        else if (command == "back") {                             // If user wants to go back
            int steps;                                      // Variable to store steps
            cin >> steps;                                     // Read number of steps
            move_backward(steps);                                 // Call backward function
        }
        else if (command == "forward") {                           // If user wants to go forward
            int steps;                                      // Variable to store steps
            cin >> steps;                                     // Read number of steps
            move_forward(steps);                                  // Call forward function
        }
        else if (command == "show") {                              // If user wants to see history
            show();                                         // Call show function
        }
        else if (command == "exit") {                             // If user wants to exit
            cout << "Exiting browser history..." << endl;              // Exit the program
            return 0;
        }
        else {                                             // If command is not valid
            cout << "Invalid command. Please try again." << endl;        // Show error message
        }
    }

    return 0;
}
```

Node Structure

```
/*
 * Structure representing a webpage node in the doubly linked list.
 */
struct Webpage {
    string data;                      // URL of the webpage
    Webpage* prevWebpage;             // Pointer to the previous webpage
    Webpage* nextWebpage;             // Pointer to the next webpage
};

Webpage* head = nullptr;              // Pointer to the first page (homepage)
Webpage* NewWebpage = nullptr;        // Pointer to the current page
```

Visit Page

```
/*
 * Function to visit a new webpage and add it to the history.
 *
 * @param - url - The URL of the new webpage to visit.
 */
int visit_newWebpage(string url) {
   if (head == NULL) {
                                        // If the head pointer is NULL
      Webpage* Homepage = new Webpage;
                        // Create a new node for the homepage
      Homepage->data = url;
                              // Store the URL in the data field
      Homepage->prevWebpage = nullptr;
                              // Previous pointer is NULL
      Homepage->nextWebpage = nullptr;
                              // Next pointer is NULL
      head = Homepage;
                              // Head now points to homepage
      NewWebpage = Homepage;
                                        // NewWebpage (current page) also points to
homepage
   } else {
      Webpage* temp = new Webpage;
                        // Create a new webpage node for the new URL
      temp->data = url;
                              // Store the URL in the node's data field
      temp->prevWebpage = NewWebpage;
                              // Set the previous pointer to current page
      temp->nextWebpage = nullptr;
                              // Set next pointer to NULL (end of list)
      NewWebpage->nextWebpage = temp;
                              // Current page's next pointer updated to new node
      NewWebpage = temp;
                                        // Move current page pointer to the new page
   }
```

```
    cout << "Visited: " << NewWebpage->data << endl;
                // Print the visited page
    return 0;

}
```

Move Back

```
/*
 * Function to move backward in the browser history.
 *
 * @param - steps - Number of steps to move back.
 */
int move_backward(int steps) {
    while (steps > 0 && NewWebpage->prevWebpage != nullptr) {
                        // As long as steps > 0 and previous page exists
        NewWebpage = NewWebpage->prevWebpage;
                                // Move back to previous page
        steps--;
                                // Decrease step count
    }
    cout << "Current page: " << NewWebpage->data << endl;
                // Show current page after moving back
    return 0;

}
```

Move Forward

```
/*
 * Function to move forward in the browser history.
 *
 * @param - steps - Number of steps to move forward.
 */
int move_forward(int steps) {
    while (steps > 0 && NewWebpage->nextWebpage != nullptr) {
                                // As long as steps > 0 and next page exists
        NewWebpage = NewWebpage->nextWebpage;
                                        // Move forward to next page
        steps--;
                                             // Decrease step count
    }
    cout << "Current page: " << NewWebpage->data << endl;      //  Show  current
page after moving forward
    return 0;

}
```

Show

```
/*
 * Function to display the full browser history from the beginning.
 */
int show() {
    Webpage* temp = head;
                                             // Start from the head of the list
    cout << "Browser History: ";
                              // Print heading
    while (temp != nullptr) {
                                             // Loop through all nodes
        if (temp == NewWebpage) {
                                    // If the node is the current page
            cout << "[" << temp->data << "] ";
                     // Highlight it with brackets
        } else {
            cout << temp->data << " ";
                              // Print normally
        }
        temp = temp->nextWebpage;
                                    // Move to the next page
    }
    cout << endl;
                                             // End the output line
    return 0;

}
```