|  |
| --- |
| **Experiment No : 4** |

|  |  |
| --- | --- |
| **Name : Manjiri Chavande** | **DSE SY COMPS** |
| **Division: B batch-A** | **UID: 2023301003** |

|  |
| --- |
| * **Problem Statement** |

1. Design Browser History.

|  |
| --- |
| * **Theory** |

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.

* **Link** − Each link of a linked list can store a data called an element.
* **Next** − Each link of a linked list contains a link to the next link called Next.
* **Prev** − Each link of a linked list contains a link to the previous link called Prev.
* **Linked List** − A Linked List contains the connection link to the first (head) link called First and to the last link called Last.

The concept of doubly linked list is widely used while implementing the logic for browser history,since we can traverse back and forth with the help of doubly LL it helps in visiting nodes and forward as well as in backward direction. We traverse the nodes in this case.

|  |
| --- |
| * **Algorithm** |

**Creating Browser History:**

1. Create a new `BrowserHistory` strct variable.

2. Create a homepage node and set it as the current node.

3. Initialize the `prev` and `next` pointers of the homepage node.

4. Set the current node to the homepage node.

**Visiting a URL:**

1. Create a new node for the visited URL.

2. Copy the URL into the new node.

3. Link the new node to the current node:

- Set the `prev` pointer of the new node to the current node.

- Set the `next` pointer of the new node to `NULL`.

4. Update the `next` pointer of the current node (if not `NULL`) to point to the new node.

5. Update the current node to the new node.

**Moving Backward:**

1. Start with the current node.

2. While there are more steps to move backward and a previous node exists:

- Move the current node backward by setting `current = current->prev`.

**Moving Forward:**

1. Start with the current node.

2. While there are more steps to move forward and a next node exists:

- Move the current node forward by setting `current = current->next`.

**Freeing Browser History:**

1. Start with the current node.

2. While there are nodes in the history:

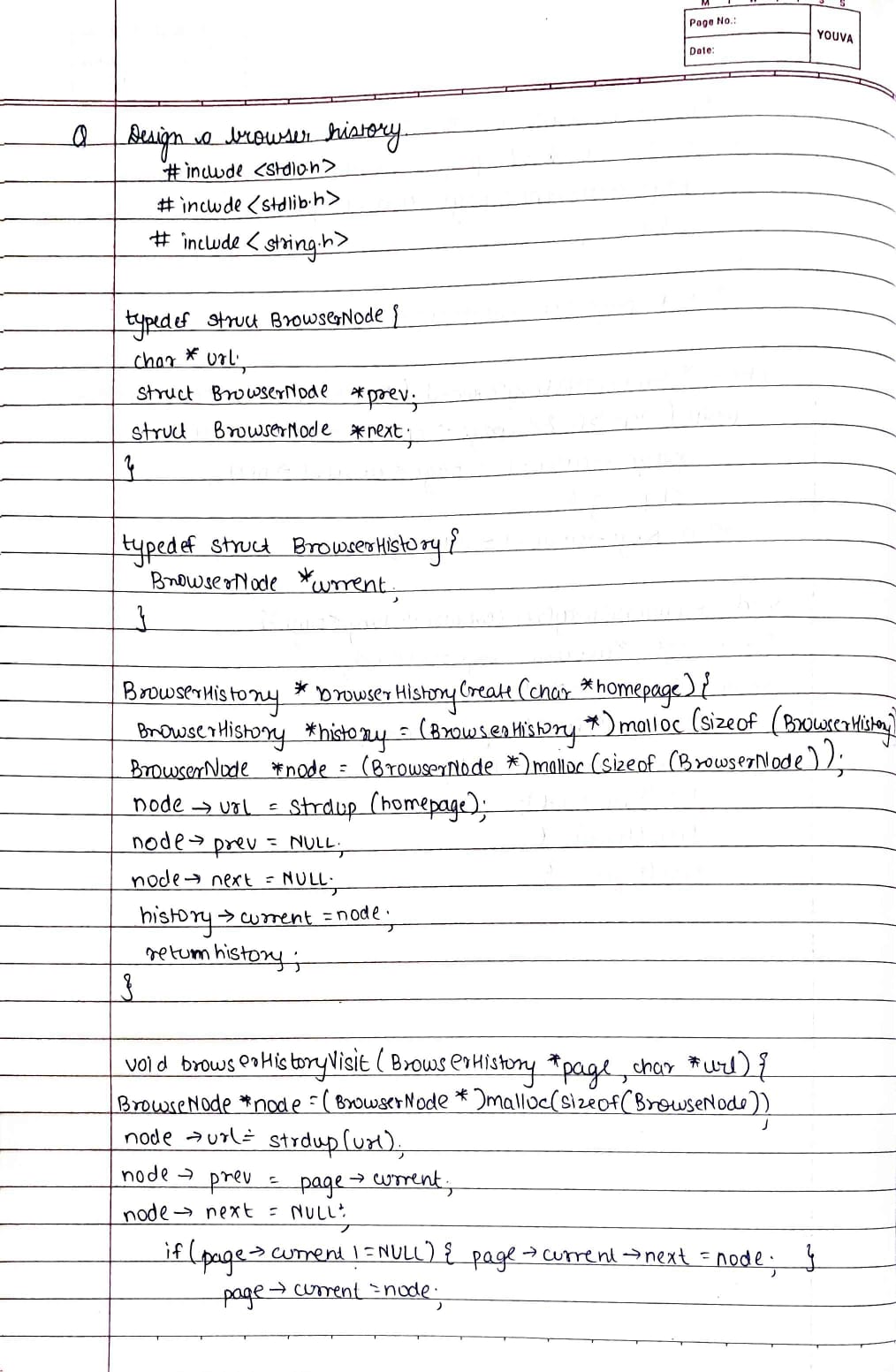
- Free the URL associated with the current node.

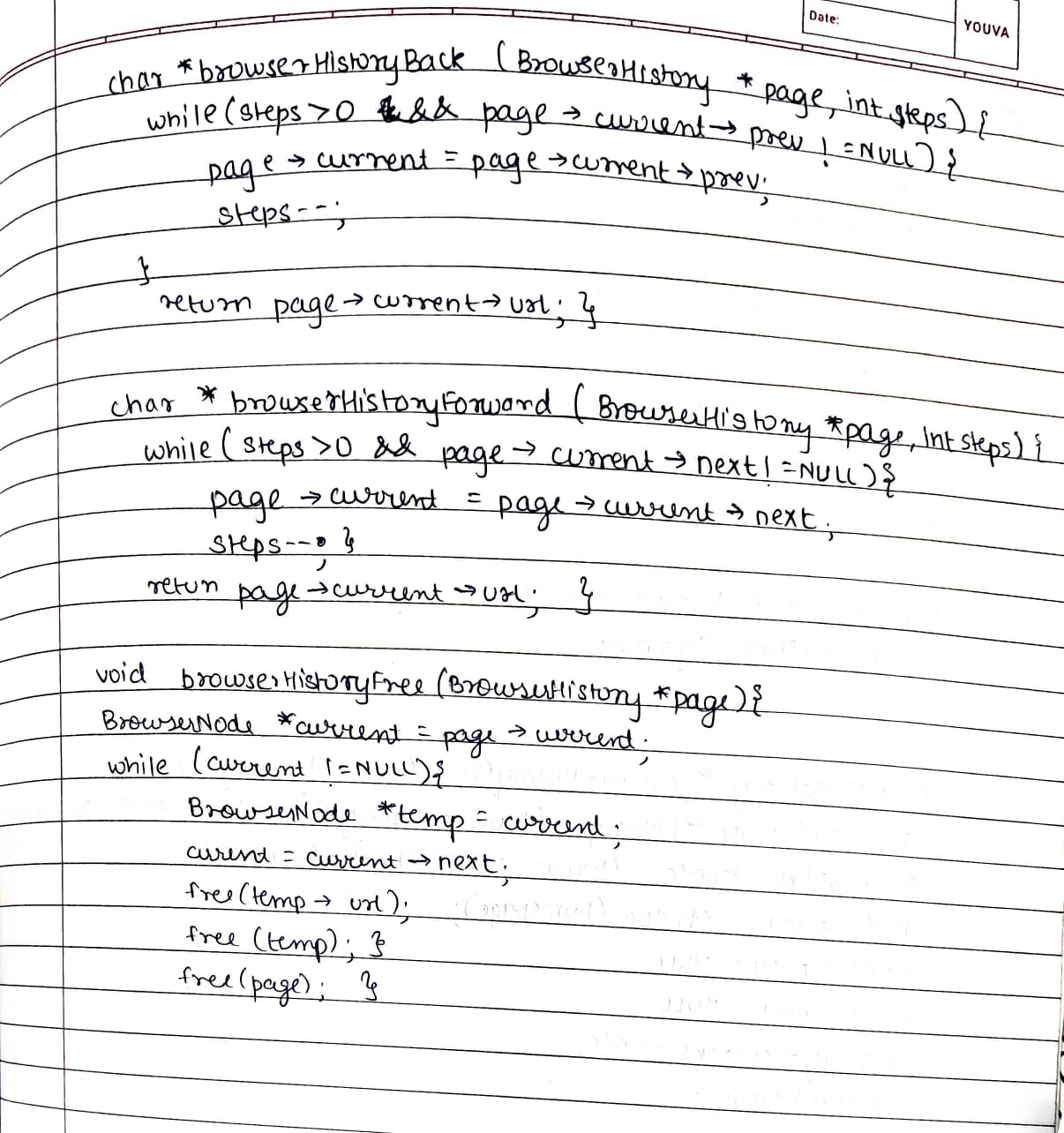
- Free the current node itself.

- Move to the next node.

3. Free the memory allocated for the `BrowserHistory` variable.

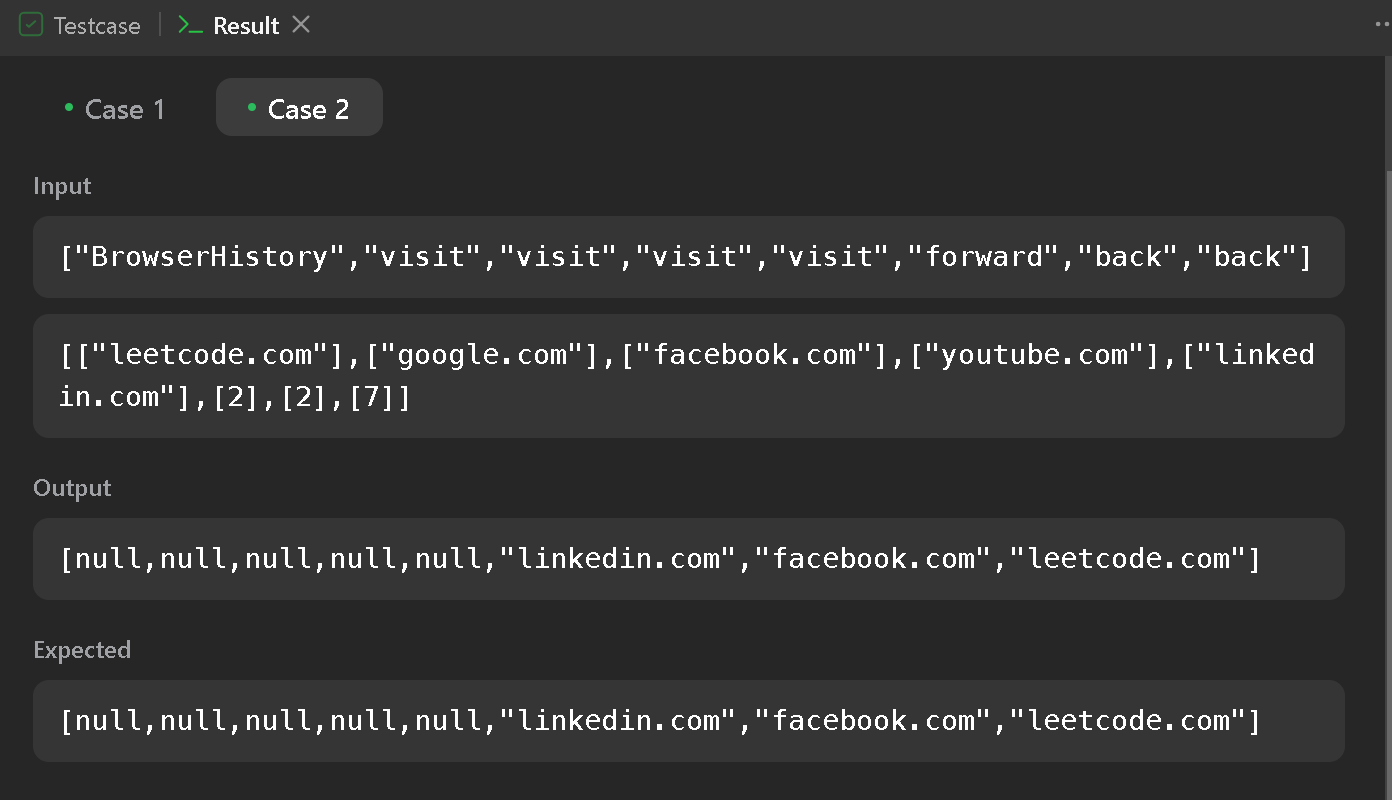
|  |
| --- |
| * **Solution** |





|  |
| --- |
| * **Test Cases** |





|  |
| --- |
| * **Code /Output** |

***/\****

***\* File: browser\_history.c***

***\* Author: Manjiri Chavande***

***\* Email:  manjiri.chavanded23@spit.ac.in***

***\* Created: 25th Sep 2023***

***\* Description: i.  Design Browser History.***

***\*/***

***#include* <stdio.h>**

***#include* <stdlib.h>**

***#include* <string.h>**

**typedef struct BrowserNode {**

**char \*url;**

**struct BrowserNode \*prev;**

**struct BrowserNode \*next;**

**} BrowserNode;**

**typedef struct BrowserHistory {**

**BrowserNode \*current;**

**} BrowserHistory;**

**BrowserHistory \*browserHistoryCreate(char \*homepage) {**

**BrowserHistory \*history = (BrowserHistory \*)malloc(sizeof(BrowserHistory));**

**BrowserNode \*node = (BrowserNode \*)malloc(sizeof(BrowserNode));**

**node->url = strdup(homepage);**

**node->prev = NULL;**

**node->next = NULL;**

**history->current = node;**

***return* history;**

**}**

***// Visit a URL and add it to the history***

**void browserHistoryVisit(BrowserHistory \*page, char \*url) {**

**BrowserNode \*node = (BrowserNode \*)malloc(sizeof(BrowserNode));**

**node->url = strdup(url); *// Copy the URL***

**node->prev = page->current;**

**node->next = NULL;**

***if* (page->current != NULL) {**

**page->current->next = node;**

**}**

**page->current = node;**

**}**

**char \*browserHistoryBack(BrowserHistory \*page, int steps) {**

***while* (steps > 0 && page->current->prev != NULL) {**

**page->current = page->current->prev;**

**steps--;**

**}**

***return* page->current->url;**

**}**

**char \*browserHistoryForward(BrowserHistory \*page, int steps) {**

***while* (steps > 0 && page->current->next != NULL) {**

**page->current = page->current->next;**

**steps--;**

**}**

***return* page->current->url;**

**}**

**void browserHistoryFree(BrowserHistory \*page) {**

**BrowserNode \*current = page->current;**

***while* (current != NULL) {**

**BrowserNode \*temp = current;**

**current = current->next;**

**free(temp->url);**

**free(temp);**

**}**

**free(page);**

**}**

**int main() {**

**BrowserHistory \*history = browserHistoryCreate("Homepage");**

**browserHistoryVisit(history, "github.com");**

**browserHistoryVisit(history, "gmail.com");**

**browserHistoryVisit(history, "youtube.com");**

**browserHistoryVisit(history, "javascript.com");**

**browserHistoryVisit(history, "java.com");**

**browserHistoryVisit(history, "google.com");**

**printf("Current URL: %s\n", browserHistoryBack(history, 2));**

**printf("Current URL: %s\n", browserHistoryForward(history, 1));**

**printf("Current URL: %s\n", browserHistoryBack(history, 0));**

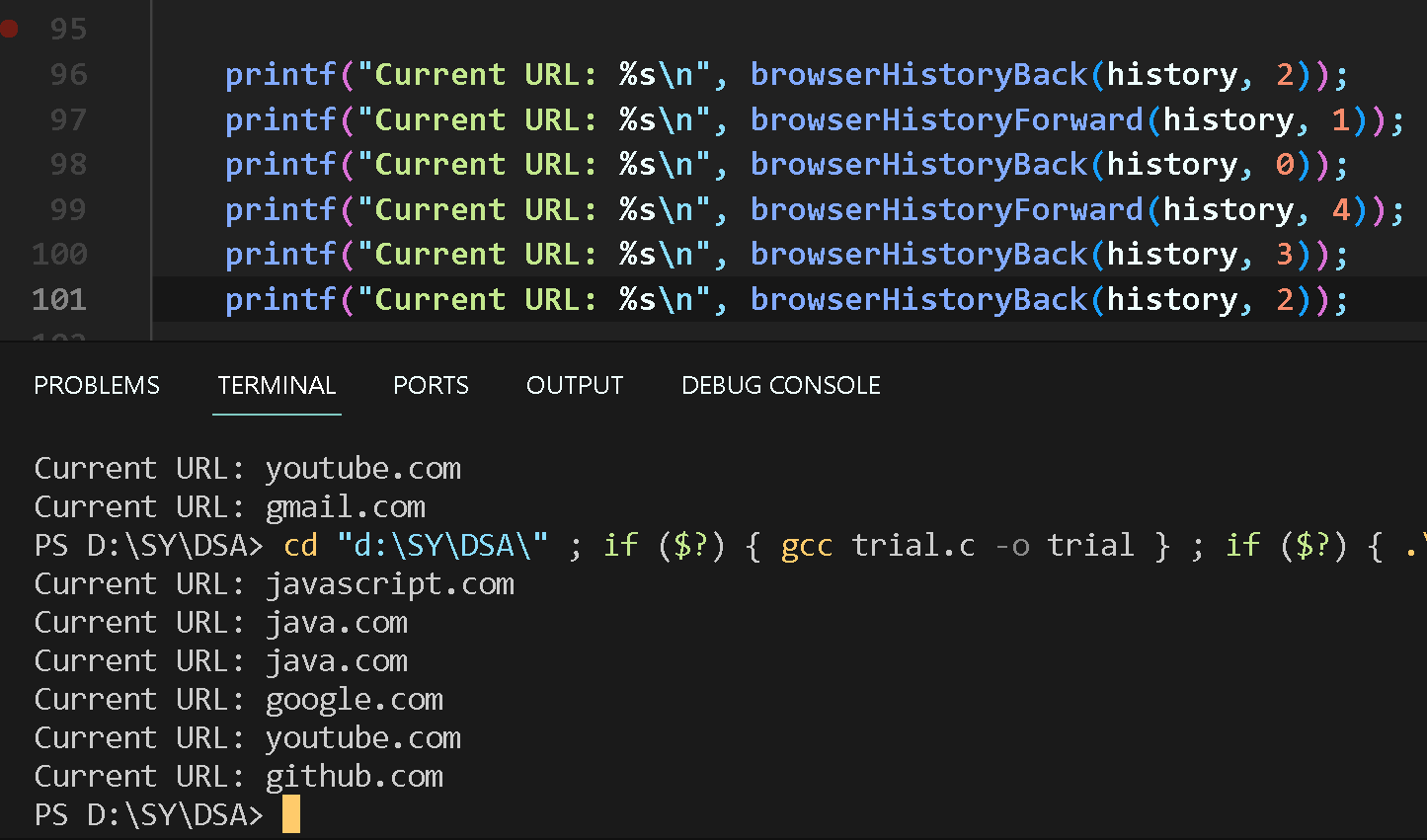
**printf("Current URL: %s\n", browserHistoryForward(history, 4));**

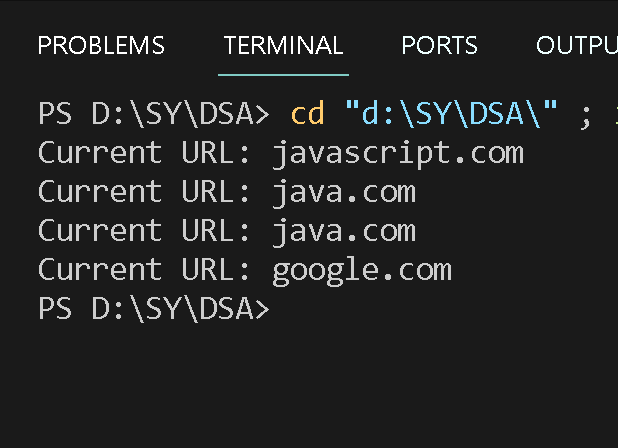
***// Free the memory when done***

**browserHistoryFree(history);**

***return* 0;**

**}**





|  |
| --- |
| * **Conclusion** |

Thus we have learned how the browser history is managed using a doubly linked list. The DLL structure allows for efficient navigation and management of visited URLs, both backward and forward in time.