Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester:(Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 05

Course Title: Data Structure Lab

Course Code: CSE 106 Section: 231-D1

Lab Experiment Name: Linked List Implementation of Stack.

Student Details

	Name	ID
1.	Promod Chandra Das	231002005

Lab Date :

Submission Date :

Course Teacher's Name : Md. Shihab Hossain

Lab Report Status

Marks: ………………………………… **Signature:.....................**

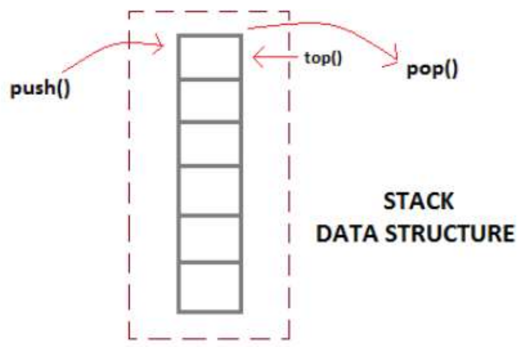
Comments:... **Date:..............................**

➢ Objective(s):

- To attain knowledge on the Stack data structure and Linked List.
- To implement Stack using Doubly Linked List.

➢ Problem Analysis

Stack is a linear data structure in which the insertion and deletion operations are performed at only one end. In a stack, adding and removing of elements are performed at a single position which is known as "top". Thatmeans, a new element is added at top of the stack and an element is removed from the top of the stack. In stack, the insertion and deletion operations are performed based on LIFO (Last In First Out) principle. In a stack, the insertion operation is performed using a function called "push" and deletion operation is performedusing a function called "pop".
A linked list is a linear collection of data elements whose order is not given by their physical placement in
memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodeswhich together represent a sequence.
This structure allows for efficient insertion or removal of elements from any position in the sequence duringiteration.
A stack can be easily implemented through the linked list.



a) Stack Data Structure

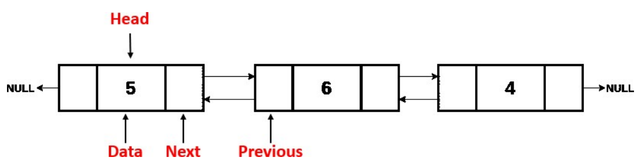


Figure 1: Illustration of Stack and Doubly Linked List

➢ Stack Operations using Linked List

- Inserting an element into the Stack
- Deleting an Element from a Stack
- Displaying stack of elements

➢ Flowchart

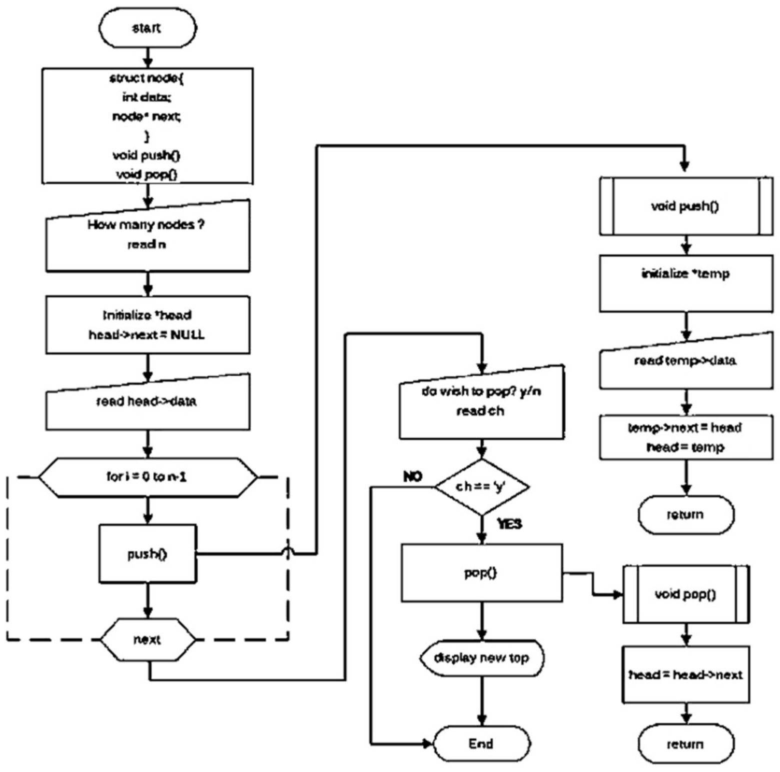


Figure 2: PUSH and POP operation with Doubly Linked List

- Lab Exercise (Submit as a report)
    - Find the specific node of an element that is present or not in the linked list.
    - Call a function that will generate the size of the linked list.

✓ **Answer :**

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

// Define the structure for a node in the linked list
struct Node {
   int data;
   struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

// Function to add a node at the end of the linked list
void appendNode(struct Node** headRef, int data) {
   struct Node* newNode = createNode(data);
   if (*headRef == NULL) {
      *headRef = newNode;
      return;
   }
   struct Node* temp = *headRef;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
}

// Function to check if a specific element is present in the linked list
int isElementPresent(struct Node* head, int element) {
   struct Node* temp = head;
   while (temp != NULL) {
      if (temp->data == element) {
            return 1; // Element found
        }
        temp = temp->next;
    }
    return 0; // Element not found
}

// Function to generate the size of the linked list
int getSize(struct Node* head) {
    int size = 0;
    struct Node* temp = head;
    while (temp != NULL) {
        size++;
        temp = temp->next;
    }
    return size;
}

// Function to print the linked list
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;

    // Add some nodes to the linked list
    appendNode(&head, 10);
    appendNode(&head, 20);
    appendNode(&head, 30);
    appendNode(&head, 40);

    // Print the linked list
    printf("Linked List: ");
    printList(head);

    // Check if an element is present in the linked list
    int element = 20;
    if (isElementPresent(head, element)) {
        printf("Element %d is present in the linked list.\n", element);
    } else {
        printf("Element %d is not present in the linked list.\n", element);
    }
```

```
 // Get the size of the linked list
   int size = getSize(head);
   printf("Size of the linked list: %d\n", size);

   return 0;
}
```

➢ Discussion & Conclusion

A stack is a data structure that follows the Last In, First Out (LIFO) principle, where the most recently added element is the first to be removed. Implementing a stack using a linked list involves leveraging the dynamic nature of linked lists, allowing for efficient insertion and deletion of elements.

❖ **Advantages**:

- **Dynamic Size**: The stack can grow and shrink dynamically with no need for resizing, unlike array-based implementations.
- **Memory Efficiency**: Memory is allocated only when necessary.

❖ **Disadvantages**:

- **Memory Overhead**: Each node requires additional memory for storing the reference to the next node.
- **Pointer Management**: Care must be taken to correctly manage the pointers to avoid memory leaks or corrupted data structures.

❖ **Complexity**:

- All basic operations (push, pop, peek, is_empty) have a time complexity of O(1)O(1)O(1).