Rajalakshmi Engineering College

Name: Kamaleshwaran K
Email: 241501079@rajalakshmi.edu.in
Roll no:
Phone: 9943398659
Branch: REC
Department: I AIML AD
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Ashwin is tasked with developing a simple application to manage a list of items in a shop inventory using a doubly linked list. Each item in the inventory has a unique identification number. The application should allow users to perform the following operations:

Create a List of Items: Initialize the inventory with a given number of items. Each item will be assigned a unique number provided by the user and insert the elements at end of the list.

Delete an Item: Remove an item from the inventory at a specific position.

Display the Inventory: Show the list of items before and after deletion.

If the position provided for deletion is invalid (e.g., out of range), it should

display an error message.

Input Format

The first line contains an integer n, representing the number of items to be initially entered into the inventory.

The second line contains n integers, each representing the unique identification number of an item separated by spaces.

The third line contains an integer p, representing the position of the item to be deleted from the inventory.

Output Format

The first line of output prints "Data entered in the list:" followed by the data values of each node in the doubly linked list before deletion.

If p is an invalid position, the output prints "Invalid position. Try again."

If p is a valid position, the output prints "After deletion the new list:" followed by the data values of each node in the doubly linked list after deletion.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 4
1 2 3 4
5
Output: Data entered in the list:
 node 1 : 1
 node 2 : 2
 node 3 : 3
 node 4 : 4
Invalid position. Try again.

Answer

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

```
typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;

typedef struct DoublyLinkedList {
    Node* head;
} DoublyLinkedList;

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

// Function to insert a node at the end of the doubly linked list
void insertAtEnd(DoublyLinkedList* list, int data) {
    Node* newNode = createNode(data);
    if (list->head == NULL) {
        list->head = newNode;
    } else {
        Node* temp = list->head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

// Function to delete a node at a given position
void deleteAtPosition(DoublyLinkedList* list, int position) {
    if (list->head == NULL) {
        printf("The list is empty.\n");
        return;
    }

    Node* temp = list->head;
```

```
    int count = 1;

    // If the position is 1, delete the head node
    if (position == 1) {
        list->head = temp->next;
        if (list->head != NULL) {
            list->head->prev = NULL;
        }
        free(temp);
        return;
    }

    // Traverse the list to find the node at the specified position
    while (temp != NULL && count < position) {
        temp = temp->next;
        count++;
    }

    // If position is invalid
    if (temp == NULL) {
        printf("Invalid position. Try again.\n");
        return;
    }

    // Update the previous and next pointers of the surrounding nodes
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    }

    // Free the memory of the node to be deleted
    free(temp);
}

// Function to display the list
void displayList(DoublyLinkedList* list) {
    Node* temp = list->head;
    int count = 1;

    if (temp == NULL) {
```

```
        printf("The list is empty.\n");
        return;
    }

    while (temp != NULL) {
        printf("node %d : %d\n", count, temp->data);
        temp = temp->next;
        count++;
    }
}

int main() {
    DoublyLinkedList list;
    list.head = NULL;

    int n, p;

    // Reading the number of items
    scanf("%d", &n);

    // Reading the items and inserting them at the end of the list
    for (int i = 0; i < n; i++) {
        int item;
        scanf("%d", &item);
        insertAtEnd(&list, item);
    }

    // Display the list before deletion
    printf("Data entered in the list:\n");
    displayList(&list);

    // Reading the position to delete
    scanf("%d", &p);

    // If position is valid, delete the item at that position
    if (p < 1 || p > n) {
        printf("Invalid position. Try again.\n");
    } else {
        // Perform the deletion
        deleteAtPosition(&list, p);

        // Display the list after deletion
```

```
        printf("\nAfter deletion the new list:\n");
        displayList(&list);
    }

    return 0;
}
```

Status : Correct **Marks : 10/10**