Rajalakshmi Engineering College

Name: vinnarasan j
Email: 241501245@rajalakshmi.edu.in
Roll no: 241501245
Phone: 7305999373
Branch: REC
Department: I AI & ML FC
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 0

Section 1 : Coding

1. Problem Statement

Your task is to create a program to manage a playlist of items. Each item is represented as a character, and you need to implement the following operations on the playlist.

Here are the main functionalities of the program:

Insert Item: The program should allow users to add items to the front and end of the playlist. Items are represented as characters.Display Playlist: The program should display the playlist containing the items that were added.

To implement this program, a doubly linked list data structure should be used, where each node contains an item character.

Input Format

The input consists of a sequence of space-separated characters, representing the items to be inserted into the doubly linked list.

The input is terminated by entering - (hyphen).

Output Format

The first line of output prints "Forward Playlist: " followed by the linked list after inserting the items at the end.

The second line prints "Backward Playlist: " followed by the linked list after inserting the items at the front.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: a b c -

Output: Forward Playlist: a b c
Backward Playlist: c b a

Answer

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

struct Node {
   char item;
   struct Node* next;
   struct Node* prev;
};

// You are using GCC
void insertAtEnd(struct Node** head, char item) {
  //type your code here
}
void displayForwa#include <stdio.h>
#include <stdlib.h>

// Structure for a node in the doubly linked list
struct Node {
   char data;
```

```
    struct Node* prev;
    struct Node* next;
};

// Function to insert a new node at the end of the list
struct Node* insertEnd(struct Node* head, char data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    if (new_node == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    new_node->data = data;
    new_node->next = NULL;

    if (head == NULL) {
        new_node->prev = NULL;
        return new_node;
    }

    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = new_node;
    new_node->prev = current;
    return head;
}

// Function to insert a new node at the front of the list
struct Node* insertFront(struct Node* head, char data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    if (new_node == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    new_node->data = data;
    new_node->prev = NULL;
    new_node->next = head;

    if (head != NULL) {
        head->prev = new_node;
    }
```

```
    return new_node;
}

// Function to display the linked list in forward direction
void displayForward(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%c ", current->data);
        current = current->next;
    }
    printf("\n");
}

// Function to display the linked list in backward direction
void displayBackward(struct Node* head) {
    struct Node* current = head;
    if (current == NULL) return;
    while (current->next != NULL) {
        current = current->next;
    }
    while (current != NULL) {
        printf("%c ", current->data);
        current = current->prev;
    }
    printf("\n");
}

// Function to free the memory allocated for the linked list
void freeList(struct Node* head) {
    struct Node* current = head;
    struct Node* next;
    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }
}

int main() {
    char data;
    struct Node* forward_head = NULL;
    struct Node* backward_head = NULL;
```

```
    while (scanf(" %c", &data) == 1 && data != '-') {
        forward_head = insertEnd(forward_head, data);
        backward_head = insertFront(backward_head, data);
    }

    printf("Forward Playlist: ");
    displayForward(forward_head);
    printf("Backward Playlist: ");
    displayBackward(backward_head);

    freeList(forward_head); // Freeing only one head is sufficient, as both point to
the same list.
    return 0;
}
int main() {
    struct Node* playlist = NULL;
    char item;

    while (1) {
        scanf(" %c", &item);
        if (item == '-') {
            break;
        }
        insertAtEnd(&playlist, item);
    }

    struct Node* tail = playlist;
    while (tail->next != NULL) {
        tail = tail->next;
    }

    printf("Forward Playlist: ");
    displayForward(playlist);

    printf("Backward Playlist: ");
    displayBackward(tail);

    freePlaylist(playlist);

    return 0;
}
```

Status : Wrong ***Marks : 0/10***