Rajalakshmi Engineering College

Name: Shremathi K
Email: 240701504@rajalakshmi.edu.in
Roll no:
Phone: 8870649491
Branch: REC
Department: I CSE FE
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Moniksha, a chess coach organizing a tournament, needs a program to manage participant IDs efficiently. The program maintains a doubly linked list of IDs and offers two functions: Append to add IDs as students register, and Print Maximum ID to identify the highest ID for administrative tasks.

This tool streamlines tournament organization, allowing Moniksha to focus on coaching her students effectively.

Input Format

The first line consists of an integer n, representing the number of participant IDs to be added.

The second line consists of n space-separated integers representing the participant IDs.

Output Format

The output displays a single integer, representing the maximum participant ID.

If the list is empty, the output prints "Empty list!".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3
163 137 155

Output: 163

Answer

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

struct Node {
    int id;          // Participant ID
    struct Node* next; // Pointer to next node
    struct Node* prev; // Pointer to previous node
};

// Function to create a new node with a given participant ID
struct Node* createNode(int id) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->id = id;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

// Function to append a participant ID to the doubly linked list
void append(struct Node** head, int id) {
    struct Node* newNode = createNode(id);
    if (*head == NULL) {
        // If the list is empty, the new node is the head of the list
        *head = newNode;
    } else {
```

```
        struct Node* temp = *head;
        // Traverse to the last node
        while (temp->next != NULL) {
            temp = temp->next;
        }
        // Add the new node to the end of the list
        temp->next = newNode;
        newNode->prev = temp;
    }
}

// Function to find the maximum participant ID in the list
int findMax(struct Node* head) {
    if (head == NULL) {
        return -1; // Return -1 if the list is empty
    }

    struct Node* temp = head;
    int max = temp->id;

    // Traverse the list and find the maximum ID
    while (temp != NULL) {
        if (temp->id > max) {
            max = temp->id;
        }
        temp = temp->next;
    }

    return max;
}

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

int main() {
```

```
    int n;
    scanf("%d", &n);

    if (n == 0) {
        // If no participant IDs are to be added, print "Empty list!"
        printf("Empty list!\n");
        return 0;
    }

    struct Node* head = NULL;

    // Read the participant IDs and append them to the list
    for (int i = 0; i < n; i++) {
        int id;
        scanf("%d", &id);
        append(&head, id);
    }

    // Find and print the maximum participant ID
    int max = findMax(head);
    if (max != -1) {
        printf("%d\n", max);
    } else {
        printf("Empty list!\n");
    }

    // Free the memory used by the list
    freeList(head);

    return 0;
}
```

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