Rajalakshmi Engineering College

Name: naveen prasath
Email: 240701352@rajalakshmi.edu.in
Roll no: 240701352
Phone: 9585322006
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 30

Section 1 : Coding

1. Problem Statement

Aarav is working on a program to analyze his test scores, which are stored in a doubly linked list. He needs a solution to input scores into the list and determine the highest score.

Help him by providing code that lets users enter test scores into the doubly linked list and find the maximum score efficiently.

Input Format

The first line consists of an integer N, representing the number of elements to be initially inserted into the doubly linked list.

The second line consists of N space-separated integers, denoting the score to be inserted.

Output Format

The output prints an integer, representing the highest score present in the list.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
89 71 2 70

Output: 89

Answer

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

typedef struct Node {
    int score;
    struct Node* next;
    struct Node* prev;
} Node;

Node* createNode(int score) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->score = score;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

void append(Node** head, int score) {
    Node* newNode = createNode(score);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
```

```
    temp->next = newNode;
    newNode->prev = temp;
}

int findMaxScore(Node* head) {
    if (head == NULL) {
        return -1; // Indicate that the list is empty
    }
    int maxScore = head->score;
    Node* temp = head;
    while (temp != NULL) {
        if (temp->score > maxScore) {
            maxScore = temp->score;
        }
        temp = temp->next;
    }
    return maxScore;
}

void freeList(Node* head) {
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    int N;
    scanf("%d", &N);

    Node* scoreList = NULL;

    for (int i = 0; i < N; i++) {
        int score;
        scanf("%d", &score);
        append(&scoreList, score);
    }

    int maxScore = findMaxScore(scoreList);
    printf("%d\n", maxScore);
```

```
    freeList(scoreList);
    return 0;
}
```

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

2. Problem Statement

You are required to implement a program that deals with a doubly linked list.

The program should allow users to perform the following operations:

Insertion at the End: Insert a node with a given integer data at the end of the doubly linked list.Insertion at a given Position: Insert a node with a given integer data at a specified position within the doubly linked list.Display the List: Display the elements of the doubly linked list.

Input Format

The first line of input consists of an integer n, representing the number of elements to be initially inserted into the doubly linked list.

The second line consists of n space-separated integers, denoting the elements to be inserted at the end.

The third line consists of integer m, representing the new element to be inserted.

The fourth line consists of an integer p, representing the position at which the new element should be inserted (1-based indexing).

Output Format

If p is valid, display the elements of the doubly linked list after performing the insertion at the specified position.

If p is invalid, display "Invalid position" in the first line and the second line prints the original list.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 25 34 48 57
35
4

Output: 10 25 34 35 48 57

Answer

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

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

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

void insertAtEnd(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

void insertAtPosition(Node** head, int data, int position) {
```

```
    Node* newNode = createNode(data);
    if (position < 1) {
        return; // Invalid position
    }
    if (position == 1) {
        newNode->next = *head;
        if (*head != NULL) {
            (*head)->prev = newNode;
        }
        *head = newNode;
        return;
    }
    Node* temp = *head;
    for (int i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        return; // Invalid position
    }
    newNode->next = temp->next;
    newNode->prev = temp;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
}

void displayList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void freeList(Node* head) {
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
```

```
    }
}

int main() {
    int n;
    scanf("%d", &n);

    Node* head = NULL;

    for (int i = 0; i < n; i++) {
        int element;
        scanf("%d", &element);
        insertAtEnd(&head, element);
    }

    int m, p;
    scanf("%d", &m);
    scanf("%d", &p);

    // Check if the position is valid
    if (p < 1 || p > n + 1) {
        printf("Invalid position\n");
        displayList(head);
    } else {
        insertAtPosition(&head, m, p);
        displayList(head);
    }

    freeList(head);
    return 0;
}
```

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

3. Problem Statement

Imagine you're managing a store's inventory list, and some products were accidentally entered multiple times. You need to remove the duplicate products from the list to ensure each product appears only once.

You have an unsorted doubly linked list of product IDs. Some of these product IDs may appear more than once, and your goal is to remove any duplicates.

Input Format

The first line of input consists of an integer n, representing the number of elements in the list.

The second line of input consists of n space-separated integers representing the list elements.

Output Format

The output prints the final after removing duplicate nodes, separated by a space.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 10
12 12 10 4 8 4 6 4 4 8
Output: 8 4 6 10 12

Answer

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

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

Node* createNode(int data) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    newNode->data = data;
    newNode->prev = newNode->next = NULL;
    return newNode;
}
```

```
void insertAtTail(Node** head, int data) {
    Node* newNode = createNode(data);
    if (!*head) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next) temp = temp->next;
    temp->next = newNode;
    newNode->prev = temp;
}

void removeDuplicatesKeepLast(Node** head) {
    int seen[101] = {0};
    Node* curr = *head;
    while (curr && curr->next) curr = curr->next;
    while (curr) {
        if (seen[curr->data]) {
            Node* toDelete = curr;
            if (curr->prev) curr->prev->next = curr->next;
            if (curr->next) curr->next->prev = curr->prev;
            if (curr == *head) *head = curr->next;
            curr = curr->prev;
            free(toDelete);
        } else {
            seen[curr->data] = 1;
            curr = curr->prev;
        }
    }
}

void printList(Node* head) {
    Node* temp=head;
    while(temp->next!=NULL){
        temp=temp->next;
    }
    while(temp){
        printf("%d ",temp->data);
        temp=temp->prev;
    }
    printf("\n");
}
```

```
int main() {
    int n, val;
    scanf("%d", &n);
    Node* head = NULL;
    for (int i = 0; i < n; i++) {
        scanf("%d", &val);
        insertAtTail(&head, val);
    }

    removeDuplicatesKeepLast(&head);
    printList(head);
    return 0;
}
```

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