Rajalakshmi Engineering College

Name: Phaveen S
Email: 240701383@rajalakshmi.edu.in
Roll no: 240701383
Phone: null
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

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

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.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;
}

Node* append(Node* head, int data) {
    Node* newNode = createNode(data);
    if (!head) return newNode;

    Node* curr = head;
    while (curr->next)
        curr = curr->next;

    curr->next = newNode;
```

```
    newNode->prev = curr;
    return head;
}

void removeNode(Node** head_ref, Node* del) {
    if (*head_ref == NULL || del == NULL) return;

    if (*head_ref == del)
        *head_ref = del->next;

    if (del->next)
        del->next->prev = del->prev;

    if (del->prev)
        del->prev->next = del->next;

    free(del);
}

void removeDuplicates(Node** head_ref) {
    bool seen[101] = { false };
    Node* tail = *head_ref;

    // Move to tail
    while (tail->next)
        tail = tail->next;

    // Traverse from tail to head
    while (tail) {
        Node* prev = tail->prev;
        if (seen[tail->data]) {
            removeNode(head_ref, tail);
        } else {
            seen[tail->data] = true;
        }
        tail = prev;
    }
}

void printFromTail(Node* head) {
    if (!head) return;
```

```
    Node* tail = head;
    while (tail->next)
        tail = tail->next;

    while (tail) {
        printf("%d ", tail->data);
        tail = tail->prev;
    }
}

int main() {
    int n, val;
    scanf("%d", &n);
    Node* head = NULL;

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

    removeDuplicates(&head);
    printFromTail(head);
    printf("\n");

    return 0;
}
```

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

2. Problem Statement

Imagine Anu is tasked with finding the middle element of a doubly linked list. Given a doubly linked list where each node contains an integer value and is inserted at the end, implement a program to find the middle element of the list. If the number of nodes is even, return the middle element pair.

Input Format

The first line of input consists of an integer N, representing the number of nodes in the doubly linked list.

The second line consists of N space-separated integers, representing the values of the nodes in the doubly linked list.

Output Format

The first line of output prints the space-separated elements of the doubly linked list.

The second line prints the middle element(s) of the doubly linked list, depending on whether the number of nodes is odd or even.

Refer to the sample outputs for the formatting specifications.

Sample Test Case

Input: 5
10 20 30 40 50

Output: 10 20 30 40 50
30

Answer

```
// You are using GCC
#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;
}

Node* append(Node* head, int data) {
    Node* newNode = createNode(data);
    if (!head) return newNode;
```

```
    Node* temp = head;
    while (temp->next)
        temp = temp->next;

    temp->next = newNode;
    newNode->prev = temp;
    return head;
}

void printList(Node* head) {
    Node* temp = head;
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

void printMiddle(Node* head, int n) {
    Node* temp = head;
    int mid = n / 2;

    for (int i = 0; i < mid; i++)
        temp = temp->next;

    if (n % 2 == 0)
        printf("%d %d", temp->prev->data, temp->data);
    else
        printf("%d", temp->data);
}

int main() {
    int n, value;
    scanf("%d", &n);
    Node* head = NULL;

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

    printList(head);
```

```
    printf(" ");
    printMiddle(head, n);
    printf("\n");

    return 0;
}
```

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

3. 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

```c
// You are using GCC
#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;
}

Node* append(Node* head, int data) {
    Node* newNode = createNode(data);
    if (!head) return newNode;

    Node* temp = head;
    while (temp->next)
        temp = temp->next;

    temp->next = newNode;
    newNode->prev = temp;
    return head;
}

int findMax(Node* head) {
    int max = head->data;
    Node* temp = head->next;

    while (temp) {
        if (temp->data > max)
            max = temp->data;
        temp = temp->next;
    }

    return max;
}
```

```
int main() {
    int n, value;
    scanf("%d", &n);
    Node* head = NULL;

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

    int maxScore = findMax(head);
    printf("%d\n", maxScore);

    return 0;
}
```

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