Rajalakshmi Engineering College

Name: NEIL DANIEL A
Email: 240701356@rajalakshmi.edu.in
Roll no: 240701356
Phone: 8925059757
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 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;
}

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

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

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

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

    return 0;
}
```

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

2. Problem Statement

Ashiq is developing a ticketing system for a small amusement park. The park issues tickets to visitors in the order they arrive. However, due to a system change, the oldest ticket (first inserted) must be revoked instead of the last one.

To manage this, Ashiq decided to use a doubly linked list-based stack, where:

Pushing adds a new ticket to the top of the stack.Removing the first inserted ticket (removing from the bottom of the stack).Printing the remaining tickets from bottom to top.

Input Format

The first line consists of an integer n, representing the number of tickets issued.

The second line consists of n space-separated integers, each representing a ticket number in the order they were issued.

Output Format

The output prints space-separated integers, representing the remaining ticket numbers in the order from bottom to top.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 7
24 96 41 85 97 91 13

Output: 96 41 85 97 91 13

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;
```

```
}

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

Node* removeBottom(Node* head) {
    if (head == NULL) return NULL;

    Node* temp = head;
    head = head->next;
    if (head != NULL) {
        head->prev = NULL;
    }
    free(temp);
    return head;
}

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

int main() {
    int n, value;
    Node* head = NULL;

    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
```

```
        scanf("%d", &value);
        head = push(head, value);
    }
    head = removeBottom(head);
    printTickets(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;

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

    Node* head = NULL;
    Node* tail = NULL;

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

        Node* newNode = (Node*)malloc(sizeof(Node));
        newNode->data = val;
        newNode->prev = NULL;
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    int seen[101] = {0};

    Node* current = tail;
    while (current != NULL) {
```

```
        Node* prev = current->prev;

        if (current->data >= 1 && current->data <= 100) {
            if (seen[current->data]) {
                Node* p = current->prev;
                Node* n = current->next;

                if (p) p->next = n;
                else head = n;

                if (n) n->prev = p;
                else tail = p;

                free(current);
            } else {
                seen[current->data] = 1;
            }
        }

        current = prev;
    }

    Node* ptr = tail;
    while (ptr != NULL) {
        printf("%d", ptr->data);
        if (ptr->prev != NULL) printf(" ");
        ptr = ptr->prev;
    }
    printf("\n");

    return 0;
}
```

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