Rajalakshmi Engineering College

Name: Pavithra J
Email: 240701381@rajalakshmi.edu.in
Roll no: 240701381
Phone: 9363364978
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

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

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

void insert(Node** head,int data){
   Node* newnode=(Node*)malloc(sizeof(Node));
   newnode->data=data;
   newnode->next=NULL;

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

   Node* temp=*head;
   while(temp->next){
      temp=temp->next;
   }
   temp->next=newnode;
   newnode->prev=temp;
}
```

```
void removeduplicatesandprint(Node* head){
    int freq[101]={0};
    Node* temp=head;

    while(temp){
        freq[temp->data]++;
        temp=temp->next;
    }

    temp=head;
    while(temp){
        if(freq[temp->data]>1){
            freq[temp->data]--;
        }
        temp=temp->next;
    }

    temp=head;
    while(temp->next){
        temp=temp->next;
    }

    while(temp){
        if(freq[temp->data]==1){
            printf("%d ",temp->data);
            freq[temp->data]=0;
        }
        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);
        insert(&head,val);
    }
```

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

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

2. Problem Statement

Krishna needs to create a doubly linked list to store and display a sequence of integers. Your task is to help write a program to read a list of integers from input, store them in a doubly linked list, and then display the list.

Input Format

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

The second line of input consists of n space-separated integers.

Output Format

The output prints a single line displaying the integers in the order they were added to the doubly linked list, separated by spaces.

If nothing is added (i.e., the list is empty), it will display "List is empty".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
1 2 3 4 5
Output: 1 2 3 4 5

Answer

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

// Define the structure for a doubly linked list node
```

```
typedef struct Node {
    int data;              // Data field to store the integer value
    struct Node* prev;     // Pointer to the previous node
    struct Node* next;     // Pointer to the next node
} Node;

// Function to create a new node with the given data
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node)); // Allocate memory for a new
node
    newNode->data = data; // Set the data for the new node
    newNode->prev = NULL;  // Initialize the previous pointer to NULL
    newNode->next = NULL;  // Initialize the next pointer to NULL
    return newNode;
}

// Function to append a new node to the end of the doubly linked list
void append(Node** head_ref, int data) {
    Node* newNode = createNode(data); // Create a new node with the given data

    if (*head_ref == NULL) {
        // If the list is empty, set the new node as the head
        *head_ref = newNode;
    } else {
        Node* temp = *head_ref;
        while (temp->next != NULL) {
            temp = temp->next;  // Traverse to the last node
        }
        temp->next = newNode;  // Set the new node as the next of the last node
        newNode->prev = temp;  // Set the previous pointer of the new node to the
last node
    }
}

// Function to print the doubly linked list
void printList(Node* head) {
    if (head == NULL) {
        printf("List is empty\n");  // If the list is empty, print "List is empty"
    } else {
        Node* temp = head;
        while (temp != NULL) {
            printf("%d", temp->data);  // Print the data of the current node
```

```
            if (temp->next != NULL) {
                printf(" ");  // Print a space if there is a next node
            }
            temp = temp->next;  // Move to the next node
        }
        printf("\n");  // Print a newline at the end
    }
}

// Main function to read input and perform the operations
int main() {
    int n;
    scanf("%d", &n);  // Read the number of elements in the list

    if (n == 0) {
        printf("List is empty\n");  // If no elements, print "List is empty"
        return 0;
    }

    Node* head = NULL;  // Initialize the head of the doubly linked list to NULL

    // Read the elements and append them to the doubly linked list
    for (int i = 0; i < n; i++) {
        int data;
        scanf("%d", &data);
        append(&head, data);  // Append each data to the doubly linked list
    }

    // Print the doubly linked list
    printList(head);

    return 0;
}
```

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

3. 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

```
// You are using GCC
#include <stdio.h>
```

```
#include <stdlib.h>

// Define the structure for a node in the doubly linked list
typedef struct Node {
    int data;              // Data field to store the integer value
    struct Node* prev;     // Pointer to the previous node
    struct Node* next;     // Pointer to the next node
} Node;

// Function to create a new node with the given data
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node)); // Allocate memory for a new
node
    newNode->data = data; // Set the data for the new node
    newNode->prev = NULL;  // Initialize the previous pointer to NULL
    newNode->next = NULL;  // Initialize the next pointer to NULL
    return newNode;
}

// Function to append a node to the end of the doubly linked list
void append(Node** head_ref, int data) {
    Node* newNode = createNode(data);  // Create a new node with the given data
    if (*head_ref == NULL) {
        *head_ref = newNode;  // If the list is empty, set the new node as the head
    } else {
        Node* temp = *head_ref;
        while (temp->next != NULL) {
            temp = temp->next;  // Traverse to the last node
        }
        temp->next = newNode;  // Set the new node as the next of the last node
        newNode->prev = temp;  // Set the previous pointer of the new node to the
last node
    }
}

// Function to insert a node at a given position
void insertAtPosition(Node** head_ref, int data, int position) {
    Node* newNode = createNode(data);

    if (position <= 0) {
        printf("Invalid position\n");
        return;
```

```
    }

    Node* temp = *head_ref;

    // Insertion at the beginning (position 1)
    if (position == 1) {
        newNode->next = *head_ref;
        if (*head_ref != NULL) {
            (*head_ref)->prev = newNode;
        }
        *head_ref = newNode;
        return;
    }

    // Traverse to the (position-1)th node
    for (int i = 1; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }

    // If position is greater than the number of nodes
    if (temp == NULL || temp->next == NULL) {
        printf("Invalid position\n");
        return;
    }

    // Insert the new node at the given position
    newNode->next = temp->next;
    newNode->prev = temp;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
}

// Function to print the elements of the doubly linked list
void printList(Node* head) {
    if (head == NULL) {
        return; // Do nothing if the list is empty
    }
    Node* temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);  // Print the data of the current node
```

```
        if (temp->next != NULL) {
            printf(" ");  // Print a space if there is a next node
        }
        temp = temp->next;  // Move to the next node
    }
    printf("\n");
}

// Main function to read input and perform the operations
int main() {
    int n, m, p;

    // Read the number of elements in the doubly linked list
    scanf("%d", &n);

    Node* head = NULL;  // Initialize the head of the doubly linked list to NULL

    // Read and append the elements to the doubly linked list
    for (int i = 0; i < n; i++) {
        int data;
        scanf("%d", &data);
        append(&head, data);
    }

    // Read the new element to insert and the position
    scanf("%d", &m);
    scanf("%d", &p);

    // Insert the new element at the specified position
    if (p > 0 && p <= n + 1) {
        insertAtPosition(&head, m, p);
        printList(head);  // Print the list after insertion
    } else {
        printf("Invalid position\n");
        printList(head);  // Print the original list
    }

    return 0;
}
```

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