Rajalakshmi Engineering College

Name: shobbika T
Email: 240701502@rajalakshmi.edu.in
Roll no: 240701502
Phone: 7305423247
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_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>
#include <set>

// Define the structure of a node in the doubly linked list
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

// Function to remove duplicates from the doubly linked list
void removeDuplicates(struct Node** head_ref) {
    // Create an empty set to track visited nodes
    std::set<int> visited;

    struct Node* current = *head_ref;

    while (current != NULL) {
        // If the current node's data is already in the set, remove it
        if (visited.find(current->data) != visited.end()) {
            struct Node* temp = current;
            current = current->next;
            // Update the previous pointer of the next node (if exists)
            if (current != NULL) {
                current->prev = temp->prev;
            }
```

```
            // Update the next pointer of the previous node (if exists)
            if (temp->prev != NULL) {
                temp->prev->next = current;
            }
            free(temp);
        } else {
            // If the current node's data is not in the set, add it and keep the node
            visited.insert(current->data);
            current = current->next;
        }
    }
}

// Function to insert a new node at the beginning of the list
void insertNodeAtBegin(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = *head_ref;
    new_node->prev = NULL;

    if (*head_ref != NULL) {
        (*head_ref)->prev = new_node;
    }
    *head_ref = new_node;
}

// Function to print the linked list
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

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

    struct Node* head = NULL;

    // Reading the product IDs and inserting them into the doubly linked list
```

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

    // Remove duplicates
    removeDuplicates(&head);

    // Print the final list after removing duplicates
    printList(head);

    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>

// Define the structure of a node in the doubly linked list
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

// Function to insert a node at the end of the doubly linked list
void insertAtEnd(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head_ref;
    new_node->data = new_data;
    new_node->next = NULL;
    new_node->prev = NULL;

    // If the list is empty, make the new node the head
    if (*head_ref == NULL) {
        *head_ref = new_node;
```

```
        return;
    }

    // Traverse to the last node
    while (last->next != NULL) {
        last = last->next;
    }

    // Update the last node's next pointer to the new node
    last->next = new_node;
    new_node->prev = last;
}

// Function to insert a node at a given position in the doubly linked list
void insertAtPosition(struct Node** head_ref, int new_data, int position) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* current = *head_ref;
    new_node->data = new_data;

    // If the position is 1 (insert at the beginning)
    if (position == 1) {
        new_node->next = *head_ref;
        new_node->prev = NULL;
        if (*head_ref != NULL) {
            (*head_ref)->prev = new_node;
        }
        *head_ref = new_node;
        return;
    }

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

    // If the position is invalid (greater than the length of the list)
    if (current == NULL) {
        printf("Invalid position\n");
        return;
    }

    // Insert the new node after the current node
```

```
    new_node->next = current->next;
    if (current->next != NULL) {
        current->next->prev = new_node;
    }
    current->next = new_node;
    new_node->prev = current;
}

// Function to print the doubly linked list
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

int main() {
    int n, m, p;

    // Read input values
    scanf("%d", &n);

    struct Node* head = NULL;

    // Read the initial n elements and insert them at the end of the list
    for (int i = 0; i < n; i++) {
        int value;
        scanf("%d", &value);
        insertAtEnd(&head, value);
    }

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

    // Try inserting the new element at the given position
    insertAtPosition(&head, m, p);

    // Print the final list (or print error message if position is invalid)
    if (p <= n && p > 0) {
        printList(head);
```

```
    } else {
        // Print the original list if the position was invalid
        printList(head);
    }

    return 0;
}
```

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

3. 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

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

// Define the structure for a node in the doubly linked list
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

// Function to insert a node at the end of the doubly linked list
void insertAtEnd(struct Node** head_ref, int new_data) {
    // Create a new node
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head_ref;

    new_node->data = new_data;
    new_node->next = NULL;
    new_node->prev = NULL;

    // If the list is empty, make the new node the head
    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }

    // Traverse to the last node
    while (last->next != NULL) {
        last = last->next;
    }

    // Link the new node after the last node
    last->next = new_node;
    new_node->prev = last;
}

// Function to display the doubly linked list
void printList(struct Node* node) {
    // If the list is empty, print "List is empty"
    if (node == NULL) {
        printf("List is empty\n");
        return;
```

```
    }

    // Traverse the list and print the elements
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

int main() {
    int n;

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

    // If the list is empty, output "List is empty"
    if (n == 0) {
        printf("List is empty\n");
        return 0;
    }

    // Create a pointer for the head of the doubly linked list
    struct Node* head = NULL;

    // Read the list elements and insert them at the end of the doubly linked list
    for (int i = 0; i < n; i++) {
        int value;
        scanf("%d", &value);
        insertAtEnd(&head, value);
    }

    // Print the elements of the list
    printList(head);

    return 0;
}
```

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