Rajalakshmi Engineering College

Name: Prithika S
Email: 240701400@rajalakshmi.edu.in
Roll no: 240701400
Phone: 9790212894
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 : 3
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>

// Node structure for doubly linked list
typedef struct Node {
   int data;
   struct Node* prev;
   struct Node* next;
} Node;

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

// Append node to the end
void append(Node** head_ref, Node** tail_ref, int data) {
   Node* newNode = createNode(data);
   if (*head_ref == NULL) {
      *head_ref = *tail_ref = newNode;
   } else {
      (*tail_ref)->next = newNode;
      newNode->prev = *tail_ref;
```

```
        *tail_ref = newNode;
    }
}

// Check if value already seen
int isSeen(int seen[], int count, int val) {
    for (int i = 0; i < count; i++) {
        if (seen[i] == val) return 1;
    }
    return 0;
}

// Print unique elements in reverse, keeping last occurrence
void printReverseUnique(Node* tail) {
    int seen[101] = {0};  // elements are between 1 and 100
    int result[30];
    int count = 0;

    Node* curr = tail;
    while (curr != NULL) {
        if (!seen[curr->data]) {
            seen[curr->data] = 1;
            result[count++] = curr->data;
        }
        curr = curr->prev;
    }

    // Print in reverse (so last occurrence appears first)
    for (int i = 0; i < count; i++) {
        printf("%d ", result[i]);
    }
    printf("\n");
}

// Main function
int main() {
    int n, val;
    scanf("%d", &n);

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

    printReverseUnique(tail);

    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

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

// Define node structure
typedef struct Node {
   int data;
   struct Node* prev;
   struct Node* next;
} Node;

// Create a new node
Node* createNode(int value) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = value;
   newNode->prev = newNode->next = NULL;
   return newNode;
}

// Append node to the end of the list
void append(Node** head_ref, Node** tail_ref, int value) {
   Node* newNode = createNode(value);
   if (*head_ref == NULL) {
      *head_ref = *tail_ref = newNode;
   } else {
      (*tail_ref)->next = newNode;
      newNode->prev = *tail_ref;
      *tail_ref = newNode;
   }
}

// Print the list in original order
void printList(Node* head) {
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }

   Node* current = head;
   while (current != NULL) {
      printf("%d ", current->data);
```

```
        current = current->next;
    }
    printf("\n");
}

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

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

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

    printList(head);

    return 0;
}
```

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

3. Problem Statement

Sam is learning about two-way linked lists. He came across a problem where he had to populate a two-way linked list and print the original as well as the reverse order of the list. Assist him with a suitable program.

Input Format

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

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

Output Format

The first line displays the message: "List in original order:"

The second line displays the elements of the doubly linked list in the original order.

The third line displays the message: "List in reverse order:"

The fourth line displays the elements of the doubly linked list in reverse order.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
1 2 3 4 5
Output: List in original order:
1 2 3 4 5
List in reverse order:
5 4 3 2 1

Answer

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

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

// Function to create a new node
Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = data;
   newNode->prev = newNode->next = NULL;
   return newNode;
}

// Append a node to the end of the list
void append(Node** head_ref, Node** tail_ref, int data) {
   Node* newNode = createNode(data);
   if (*head_ref == NULL) {
       *head_ref = *tail_ref = newNode;
   } else {
       (*tail_ref)->next = newNode;
       newNode->prev = *tail_ref;
       *tail_ref = newNode;
   }
}

// Print list in original order
void printOriginal(Node* head) {
   Node* current = head;
   while (current != NULL) {
       printf("%d ", current->data);
       current = current->next;
   }
}

// Print list in reverse order
void printReverse(Node* tail) {
   Node* current = tail;
   while (current != NULL) {
       printf("%d ", current->data);
       current = current->prev;
   }
}

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

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

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

   printf("List in original order: ");
   printOriginal(head);
   printf("\nList in reverse order: ");
   printReverse(tail);

   printf("\n");
   return 0;
}
```

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