Rajalakshmi Engineering College

Name: Pranav Narayanan
Email: 240701393@rajalakshmi.edu.in
Roll no:
Phone: 9500579427
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>

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

    int elements[n];
    int last_occurrence[101];  // Elements are <= 100
    for (int i = 0; i <= 100; i++) last_occurrence[i] = -1;

    for (int i = 0; i < n; i++) {
        scanf("%d", &elements[i]);
        last_occurrence[elements[i]] = i;  // Update last occurrence
    }

    // Count unique elements
    int unique_count = 0;
    int unique[101];

    for (int i = 0; i <= 100; i++) {
        if (last_occurrence[i] != -1) {
            unique[unique_count++] = i;
        }
    }

    // Sort unique elements by last_occurrence descending (simple bubble sort)
    for (int i = 0; i < unique_count - 1; i++) {
        for (int j = 0; j < unique_count - i - 1; j++) {
            if (last_occurrence[unique[j]] < last_occurrence[unique[j+1]]) {
                int temp = unique[j];
                unique[j] = unique[j+1];
                unique[j+1] = temp;
            }
        }
    }

    // Print the result
    for (int i = 0; i < unique_count; i++) {
        printf("%d ", unique[i]);
    }

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

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

2. 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

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

// Define doubly linked list node
typedef struct Node {
   int score;
   struct Node* prev;
   struct Node* next;
} Node;

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

// Function to insert node at the end of the list
void insertEnd(Node** head, int score) {
   Node* newNode = createNode(score);
   if (*head == NULL) {
      *head = newNode;
      return;
   }
   Node* temp = *head;
   while (temp->next != NULL)
      temp = temp->next;
   temp->next = newNode;
   newNode->prev = temp;
}

// Function to find max score in the doubly linked list
int findMax(Node* head) {
   if (head == NULL) return -1; // or some error value
   int maxScore = head->score;
   Node* temp = head->next;
   while (temp != NULL) {
      if (temp->score > maxScore)
         maxScore = temp->score;
      temp = temp->next;
   }
   return maxScore;
}

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

   Node* head = NULL;

   for (int i = 0; i < N; i++) {
      int score;
      scanf("%d", &score);
      insertEnd(&head, score);
   }

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

   // Free the list (optional but good practice)
   Node* temp;
   while (head != NULL) {
      temp = head;
      head = head->next;
      free(temp);
   }

   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>
```

```
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 = NULL;
   newNode->next = NULL;
   return newNode;
}

// Function to append a node at the end of the doubly linked list
void append(Node** head_ref, int data) {
   Node* newNode = createNode(data);
   if (*head_ref == NULL) {
      *head_ref = newNode;
      return;
   }
   Node* temp = *head_ref;
   while (temp->next != NULL)
      temp = temp->next;
   temp->next = newNode;
   newNode->prev = temp;
}

// Function to print list in original order
void printOriginal(Node* head) {
   printf("List in original order:");
   Node* temp = head;
   while (temp != NULL) {
      printf(" %d", temp->data);
      temp = temp->next;
   }
   printf("\n");
}

// Function to print list in reverse order
void printReverse(Node* head) {
   printf("List in reverse order:");
   if (head == NULL)
      return;
   Node* temp = head;
   // go to the last node
   while (temp->next != NULL)
      temp = temp->next;
   // print in reverse
   while (temp != NULL) {
      printf(" %d", temp->data);
      temp = temp->prev;
   }
   printf("\n");
}

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

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

   printOriginal(head);
   printReverse(head);

   // Free memory (optional for small programs but good practice)
   Node* current = head;
   while (current != NULL) {
      Node* next = current->next;
      free(current);
      current = next;
   }

   return 0;
}
```

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