Rajalakshmi Engineering College

Name: LIRESH NV
Email: 241501100@rajalakshmi.edu.in
Roll no: 241501100
Phone: 9840466142
Branch: REC
Department: I AI & ML FA
Batch: 2028
Degree: B.E - AI & ML

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

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

// Function to create a new node
Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (newNode == NULL) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->data = data;
   newNode->next = NULL;
   newNode->prev = NULL;
   return newNode;
}

// Function to display the doubly linked list
void displayList(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;
    scanf("%d", &n); // Read the number of integers

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

    // Read integers and create the doubly linked list
    if (n > 0) {
        for (int i = 0; i < n; i++) {
            int value;
            scanf("%d", &value);
            Node* newNode = createNode(value);
            if (head == NULL) {
                head = newNode; // First node
                tail = newNode;
            } else {
                tail->next = newNode; // Link the new node
                newNode->prev = tail;  // Set the previous pointer
                tail = newNode;        // Move the tail pointer
            }
        }
    }

    // Display the list
    displayList(head);

    // Free the allocated memory
    Node* current = head;
    while (current != NULL) {
        Node* nextNode = current->next;
        free(current);
        current = nextNode;
    }

    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

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

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

// Function to create a new node
Node* createNode(int ticketNumber) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (newNode == NULL) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->ticketNumber = ticketNumber;
   newNode->next = NULL;
   newNode->prev = NULL;
   return newNode;
}

// Function to push a ticket onto the stack
void push(Node** head, int ticketNumber) {
   Node* newNode = createNode(ticketNumber);
   if (*head == NULL) {
      *head = newNode; // First node
   } else {
      newNode->next = *head; // Link new node to the current head
      (*head)->prev = newNode; // Link current head back to new node
      *head = newNode; // Move head to the new node
   }
}

// Function to remove the oldest ticket (from the bottom of the stack)
void removeOldest(Node** head) {
   if (*head == NULL) {
      return; // Stack is empty
   }

   Node* current = *head;
   // Traverse to the last node (bottom of the stack)
   while (current->next != NULL) {
      current = current->next;
   }

   // Remove the last node
   if (current->prev != NULL) {
      current->prev->next = NULL; // Update the second last node's next pointer
   } else {
      *head = NULL; // If there was only one node
   }
   free(current); // Free the memory of the removed node
}

// Function to display the tickets from bottom to top
void displayTickets(Node* head) {
   if (head == NULL) {
      return; // No tickets to display
   }

   Node* current = head;
   // Traverse to the last node (bottom of the stack)
   while (current->next != NULL) {
      current = current->next;
   }

   // Print tickets from bottom to top
   while (current != NULL) {
      printf("%d ", current->ticketNumber);
      current = current->prev;
   }
   printf("\n");
}

int main() {
   int n;
   scanf("%d", &n); // Read the number of tickets
   Node* head = NULL;

   // Read ticket numbers and push them onto the stack
   for (int i = 0; i < n; i++) {
      int ticketNumber;
      scanf("%d", &ticketNumber);
      push(&head, ticketNumber);
   }

   // Remove the oldest ticket
   removeOldest(&head);

   // Display the remaining tickets from bottom to top
   displayTickets(head);

   // Free the remaining nodes
   Node* current = head;
   while (current != NULL) {
      Node* nextNode = current->next;
      free(current);
      current = nextNode;
   }

   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>

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

// Function to create a new node
Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (newNode == NULL) {
      printf("Memory allocation failed\n");
      exit(1);
```

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

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

// Function to display the list in original order
void displayOriginal(Node* head) {
    Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

// Function to display the list in reverse order
void displayReverse(Node* head) {
    if (head == NULL) return; // If the list is empty, do nothing

    Node* current = head;
    // Traverse to the end of the list
    while (current->next != NULL) {
        current = current->next;
    }
    // Print the list in reverse order
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->prev;
    }
    printf("\n");
}

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

    Node* head = NULL;

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

    // Display the list in original order
    printf("List in original order:\n");
    displayOriginal(head);

    // Display the list in reverse order
    printf("List in reverse order:\n");
    displayReverse(head);

    // Free the allocated memory
    Node* current = head;
    while (current != NULL) {
        Node* nextNode = current->next;
        free(current);
        current = nextNode;
    }

    return 0;
}
```

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