Rajalakshmi Engineering College

Name: Lakshmi Narayanan S
Email: 241801133@rajalakshmi.edu.in
Roll no: 241801133
Phone: 9345832054
Branch: REC
Department: I AI & DS - AE
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 21

Section 1 : Coding

1. 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

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

// Define the Node structure for the doubly linked list
struct Node {
   int ticket;
   struct Node* next;
   struct Node* prev;
};

// Define the TicketStack structure
struct TicketStack {
   struct Node* head;
   struct Node* tail;
};

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

```
}

// Function to initialize the ticket stack
void initializeStack(struct TicketStack* stack) {
    stack->head = NULL;
    stack->tail = NULL;
}

// Function to push a ticket to the stack (add to the top)
void push(struct TicketStack* stack, int ticket) {
    struct Node* newNode = createNode(ticket);

    if (stack->head == NULL) {  // The stack is empty
        stack->head = newNode;
        stack->tail = newNode;
    } else {  // Add to the top (tail of the list)
        stack->tail->next = newNode;
        newNode->prev = stack->tail;
        stack->tail = newNode;
    }
}

// Function to pop the oldest ticket (remove from the bottom of the stack)
void popOldest(struct TicketStack* stack) {
    if (stack->head == NULL) {
        return;  // Stack is empty, no ticket to remove
    }

    struct Node* temp = stack->head;
    if (stack->head == stack->tail) {  // Only one element in the stack
        stack->head = NULL;
        stack->tail = NULL;
    } else {
        stack->head = stack->head->next;
        stack->head->prev = NULL;
    }

    free(temp);  // Free the memory of the removed node
}

// Function to print the tickets from bottom to top (head to tail of the list)
void printTickets(struct TicketStack* stack) {
    struct Node* current = stack->head;

    while (current != NULL) {
        printf("%d ", current->ticket);
        current = current->next;
    }
    printf("\n");
}

int main() {
    int n;

    // Input the number of tickets
    scanf("%d", &n);

    // Initialize the ticket stack
    struct TicketStack stack;
    initializeStack(&stack);

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

    // Remove the oldest ticket
    popOldest(&stack);

    // Print the remaining tickets from bottom to top
    printTickets(&stack);

    return 0;
}
```

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

2. Problem Statement

Vanessa is learning about the doubly linked list data structure and is eager to play around with it. She decides to find out how the elements are inserted at the beginning and end of the list.

Help her implement a program for the same.

Input Format

The first line of input contains an integer N, representing the size of the doubly linked list.

The next line contains N space-separated integers, each representing the values to be inserted into the doubly linked list.

Output Format

The first line of output prints the integers, after inserting them at the beginning, separated by space.

The second line prints the integers, after inserting at the end, separated by space.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
1 2 3 4 5

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

Answer

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

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

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

// Function to insert a node at the beginning of the list
void insertAtBeginning(struct Node** head, int data) {
   struct Node* newNode = createNode(data);
   newNode->next = *head;
   if (*head != NULL) {
      (*head)->prev = newNode;
   }
   *head = newNode;
}

// Function to display the list from head to tail
void displayFromHead(struct Node* head) {
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }
   struct Node* temp = head;
   while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
   }
   printf("\n");
}

// Function to display the list from tail to head
void displayFromTail(struct Node* head) {
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }
   struct Node* temp = head;
   // Traverse to the last node
   while (temp->next != NULL) {
      temp = temp->next;
   }
   // Traverse back from the last node to the head
   while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->prev;
   }
   printf("\n");
}

int main() {
   int n;

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

   if (n == 0) {
      printf("List is empty\n");
      return 0;
   }

   struct Node* head = NULL;

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

   // Display the list from tail to head (reverse order

   // Display the list from head to tail (insertion order)
   displayFromHead(head);
   displayFromTail(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

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

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

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

// Function to insert at the beginning
void insertAtBeginning(struct Node** head, int data) {
   struct Node* newNode = createNode(data);
   newNode->next = *head;  // Point new node's next to current head
   if (*head != NULL) {    // If list is not empty, update the previous head's prev to
new node
      (*head)->prev = newNode;
   }
   *head = newNode;  // Move the head pointer to the new node
}

// Function to insert at the end
void insertAtEnd(struct Node** tail, int data) {
   struct Node* newNode = createNode(data);
   if (*tail == NULL) {  // If list is empty, new node will be both head and tail
      *tail = newNode;
   } else {
      (*tail)->next = newNode;  // Point current tail's next to the new node
      newNode->prev = *tail;    // Set new node's prev to current tail
      *tail = newNode;          // Move the tail pointer to the new node
   }
}

// Function to display the list from head to tail
void displayList(struct Node* head) {
   if(head == NULL){
      printf("List is empty");
   }
   struct Node* temp = head;
   while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
   }
```

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

int main() {
    int N;

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

    struct Node* head = NULL;  // Initialize the head pointer
    struct Node* tail = NULL;  // Initialize the tail pointer

    // Read the elements and insert them at the beginning first
    int arr[N];
    for (int i = 0; i < N; i++) {
        scanf("%d", &arr[i]);
        insertAtBeginning(&head, arr[i]);
    }

    // Display the list after inserting at the beginning


    // Now, reinitialize the list to NULL and insert the elements at the end
    head = NULL;
    tail = NULL;

    for (int i = 0; i < N; i++) {
        insertAtEnd(&tail, arr[i]);
    }

    // Display the list after inserting at the end
    displayList(head);

    return 0;
}
```

Status : Partially correct ***Marks : 1/10***