# Rajalakshmi Engineering College

Name: Piraisoodan R
Email: 240701384@rajalakshmi.edu.in
Roll no: 240701384
Phone: 8056892546
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 : 20

## Section 1 : Coding

1. 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>

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

Node* createNode(int data) {
   Node* newNode = (Node*) malloc(sizeof(Node));
   newNode->data = data;
   newNode->prev = newNode->next = NULL;
   return newNode;
```

```
}

void insertEnd(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }

    Node* temp = *head;
    while (temp->next)
        temp = temp->next;

    temp->next = newNode;
    newNode->prev = temp;
}

int getLength(Node* head) {
    int count = 0;
    while (head) {
        count++;
        head = head->next;
    }
    return count;
}


int insertAtPosition(Node** head, int data, int pos) {
    int len = getLength(*head);
    if (pos < 1 || pos > len + 1)
        return 0;

    Node* newNode = createNode(data);

    if (pos == 1) {
        newNode->next = *head;
        if (*head)
            (*head)->prev = newNode;
        *head = newNode;
        return 1;
    }
    Node* temp = *head;
    for (int i = 1; i < pos - 1 && temp; i++)
        temp = temp->next;

    newNode->next = temp->next;
    newNode->prev = temp;

    if (temp->next)
        temp->next->prev = newNode;

    temp->next = newNode;

    return 1;
}


void displayList(Node* head) {
    while (head) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}


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

    scanf("%d", &n);

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

    scanf("%d", &newElement);
    scanf("%d", &position);

    if (!insertAtPosition(&head, newElement, position)) {
        printf("Invalid position\n");
        displayList(head);
    } else {
        displayList(head);
    }

    return 0;
}
```

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

2. 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*

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

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

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

void insertEnd(Node** head, int data) {
   Node* newNode = createNode(data);

   if (*head == NULL) {
      *head = newNode;
      return;
   }

   Node* temp = *head;
   while (temp->next != NULL)
      temp = temp->next;

   temp->next = newNode;
   newNode->prev = temp;
}

void printOriginal(Node* head) {
   Node* temp = head;
```

```
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void printReverse(Node* head) {

    Node* temp = head;
    if (!temp) return;

    while (temp->next)
        temp = temp->next;


    while (temp) {
        printf("%d ", temp->data);
        temp = temp->prev;
    }
    printf("\n");
}

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

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

    printf("List in original order:\n");
    printOriginal(head);

    printf("List in reverse order:\n");
    printReverse(head);

    return 0;
}
```

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

3. 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>

// Doubly Linked List Node
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;
}

// Insert at end
void insertEnd(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;

    temp->next = newNode;
    newNode->prev = temp;
}

// Remove a node from list
void removeNode(Node** head, Node* del) {
    if (*head == NULL || del == NULL)
        return;

    if (*head == del)
        *head = del->next;

    if (del->prev != NULL)
        del->prev->next = del->next;

    if (del->next != NULL)
        del->next->prev = del->prev;

    free(del);
```

```
}

// Remove duplicates, keeping only last occurrence
void removeDuplicates(Node** head) {
   int seen[101] = {0}; // element constraints: 1 <= val <= 100

   // Go to tail
   Node* curr = *head;
   while (curr && curr->next)
      curr = curr->next;

   // Traverse from tail to head
   while (curr) {
      Node* prev = curr->prev;
      if (seen[curr->data]) {
         removeNode(head, curr);
      } else {
         seen[curr->data] = 1;
      }
      curr = prev;
   }
}

// Print list forward
void printList(Node* head) {
   Node* temp = head;
   while (temp) {
      printf("%d ", temp->data);
      temp = temp->next;
   }
   printf("\n");
}

// Main
int main() {
   int n, val;
   Node* head = NULL;

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

```
    }

    removeDuplicates(&head);
    printList(head);

    return 0;
}
```

***Status :*** Wrong ***Marks : 0/10***