Rajalakshmi Engineering College

Name: MOHANAPRIYA N
Email: 240801203@rajalakshmi.edu.in
Roll no: 240801203
Phone: 8667043590
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Your task is to create a program to manage a playlist of items. Each item is represented as a character, and you need to implement the following operations on the playlist.

Here are the main functionalities of the program:

Insert Item: The program should allow users to add items to the front and end of the playlist. Items are represented as characters.Display Playlist: The program should display the playlist containing the items that were added.

To implement this program, a doubly linked list data structure should be used, where each node contains an item character.

Input Format

The input consists of a sequence of space-separated characters, representing the items to be inserted into the doubly linked list.

The input is terminated by entering - (hyphen).

Output Format

The first line of output prints "Forward Playlist: " followed by the linked list after inserting the items at the end.

The second line prints "Backward Playlist: " followed by the linked list after inserting the items at the front.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: a b c -

Output: Forward Playlist: a b c
Backward Playlist: c b a

Answer

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

struct Node {
    char item;
    struct Node* next;
    struct Node* prev;
};

/*// You are using GCC
void insertAtEnd(struct Node** head, char item) {
   //type your code here
}
void displayForward(struct Node* head) {
    //type your code here
}

void displayBackward(struct Node* tail) {
    //type your code here
```

```
}

void freePlaylist(struct Node* head) {
    //type your code here
}*/

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




void insertAtEnd(struct Node** head, char item) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->item = item;
    newNode->next = NULL;
    newNode->prev = NULL;

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

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

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

void displayForward(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%c ", temp->item);
        temp = temp->next;
    }
    printf("\n");
```

```
}

void displayBackward(struct Node* tail) {
    struct Node* temp = tail;
    while (temp != NULL) {
        printf("%c ", temp->item);
        temp = temp->prev;
    }
    printf("\n");
}

void freePlaylist(struct Node* head) {
    struct Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

// Sample main function to help test the logic (can be omitted if not required)
/*
int main() {
    struct Node* head = NULL;
    char ch;
    while (1) {
        scanf(" %c", &ch);
        if (ch == '-') break;
        insertAtEnd(&head, ch);
    }

    printf("Forward: ");
    displayForward(head);

    struct Node* tail = head;
    while (tail && tail->next != NULL) {
        tail = tail->next;
    }

    printf("Backward: ");
    displayBackward(tail);
```

```
    freePlaylist(head);
    return 0;
}
*/

int main() {
    struct Node* playlist = NULL;
    char item;

    while (1) {
        scanf(" %c", &item);
        if (item == '-') {
            break;
        }
        insertAtEnd(&playlist, item);
    }

    struct Node* tail = playlist;
    while (tail->next != NULL) {
        tail = tail->next;
    }

    printf("Forward Playlist: ");
    displayForward(playlist);

    printf("Backward Playlist: ");
    displayBackward(tail);

    freePlaylist(playlist);

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: MOHANAPRIYA N
Email: 240801203@rajalakshmi.edu.in
Roll no: 240801203
Phone: 8667043590
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Moniksha, a chess coach organizing a tournament, needs a program to manage participant IDs efficiently. The program maintains a doubly linked list of IDs and offers two functions: Append to add IDs as students register, and Print Maximum ID to identify the highest ID for administrative tasks.

This tool streamlines tournament organization, allowing Moniksha to focus on coaching her students effectively.

Input Format

The first line consists of an integer n, representing the number of participant IDs to be added.

The second line consists of n space-separated integers representing the participant IDs.

Output Format

The output displays a single integer, representing the maximum participant ID.

If the list is empty, the output prints "Empty list!".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3
163 137 155
Output: 163

Answer

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

typedef struct Node{
   int id;
   struct Node*prev;
   struct Node*next;

}Node;

Node*append(Node*head,int id)
{
   Node*newNode=(Node*)malloc(sizeof(Node));
   if(newNode==NULL)
   {
      printf("Memory allocation failed");
      exit(EXIT_FAILURE);

   }
   newNode->id=id;
   newNode->next=NULL;
   if(head==NULL)
   {
      newNode->prev=NULL;
```

```
        return newNode;
    }
    Node*current=head;
    while(current->next!=NULL)
    {
        current=current->next;
    }
    current->next=newNode;
    newNode->prev=current;
    return head;
}

void printMaxId(Node*head)
{
    if(head==NULL)
    {
        printf("Empty list!");
        return;
    }
    int maxid=head->id;
    Node*current=head->next;
    while(current!=NULL)
    {
        if(current->id>maxid)
        {
            maxid=current->id;
        }
        current=current->next;
    }
    printf("%d",maxid);
}
void freeList(Node*head)
{
    Node*current=head;
    while(current!=NULL)
    {
        Node*temp=current;
        current=current->next;
        free(temp);
    }
}
int main()
```

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

    }
    printMaxId(head);
    freeList(head);
    return 0;
}
```

Status : Correct

Marks : 10/10

Rajalakshmi Engineering College

Name: MOHANAPRIYA N
Email: 240801203@rajalakshmi.edu.in
Roll no: 240801203
Phone: 8667043590
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Bob is tasked with developing a company's employee record management system. The system needs to maintain a list of employee records using a doubly linked list. Each employee is represented by a unique integer ID.

Help Bob to complete a program that adds employee records at the front, traverses the list, and prints the same for each addition of employees to the list.

Input Format

The first line of input consists of an integer N, representing the number of employees.

The second line consists of N space-separated integers, representing the employee IDs.

Output Format

For each employee ID, the program prints "Node Inserted" followed by the current state of the doubly linked list in the next line, with the data values of each node separated by spaces.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
101 102 103 104

Output: Node Inserted
101
Node Inserted
102 101
Node Inserted
103 102 101
Node Inserted
104 103 102 101

Answer

```
#include <iostream>
using namespace std;

struct node {
    int info;
    struct node* prev, * next;
};

struct node* start = NULL;

// You are using GCC
void traverse() {
   struct node* temp= start;
   while(temp!= NULL)
{
    printf("%d ",temp->info);
    temp = temp->next;
}
printf("\n");
```

```
}
void insertAtFront(int data) {
  struct node* newNode = (struct node*)malloc(sizeof(struct node));
  newNode->info = data;
  newNode->prev = NULL;
  newNode->next = start;

  if(start != NULL){
     start->prev = newNode;
  }
  start = newNode;
  printf("Node Inserted\n");
}
int main() {
   int n, data;
   cin >> n;
   for (int i = 0; i < n; ++i) {
      cin >> data;
      insertAtFront(data);
      traverse();
   }
   return 0;
}
```

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

Rajalakshmi Engineering College

Name: MOHANAPRIYA N
Email: 240801203@rajalakshmi.edu.in
Roll no: 240801203
Phone: 8667043590
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Ravi is developing a student registration system for a college. To efficiently store and manage the student IDs, he decides to implement a doubly linked list where each node represents a student's ID.

In this system, each student's ID is stored sequentially, and the system needs to display all registered student IDs in the order they were entered.

Implement a program that creates a doubly linked list, inserts student IDs, and displays them in the same order.

Input Format

The first line contains an integer N the number of student IDs.

The second line contains N space-separated integers representing the student IDs.

Output Format

The output should display the single line containing N space-separated integers representing the student IDs stored in the doubly linked list.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 20 30 40 50
Output: 10 20 30 40 50

Answer

```
// You are using GCC
#include<stdio.h>
#include<stdlib.h>
struct Node{
   int id;
   struct Node*prev;
   struct Node*next;

};

struct Node*insertAtEnd(struct Node*head,int id)
{
   struct Node*newNode=(struct Node*)malloc(sizeof(struct Node));
   if(newNode)
   {
      newNode->id=id;
      newNode->prev=NULL;
      newNode->next=NULL;
   }
   else{
      return head;
   }
   if(!head)
   {
```

```
        return newNode;

    }
    struct Node*current=head;
    while(current->next)
    {
        current=current->next;
    }
    current->next=newNode;
    current->prev=current;
    return head;
}
void displayList(struct Node*head)
{
    struct Node*current=head;
    while(current){
        printf("%d ",current->id);
        current=current->next;
    }
    printf("\n");
}

void freeList(struct Node*head)
{
    struct Node*current=head;
    while(current)
    {
        struct Node*temp=current;
        current=current->next;
        free(temp);
    }
}
int main()
{
    int n,id;
    struct Node*head=NULL;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        scanf("%d",&id);
        head=insertAtEnd(head,id);
```

```
    }
    displayList(head);
    freeList(head);
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: MOHANAPRIYA N
Email: 240801203@rajalakshmi.edu.in
Roll no: 240801203
Phone: 8667043590
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Ashwin is tasked with developing a simple application to manage a list of items in a shop inventory using a doubly linked list. Each item in the inventory has a unique identification number. The application should allow users to perform the following operations:

Create a List of Items: Initialize the inventory with a given number of items. Each item will be assigned a unique number provided by the user and insert the elements at end of the list.

Delete an Item: Remove an item from the inventory at a specific position.

Display the Inventory: Show the list of items before and after deletion.

If the position provided for deletion is invalid (e.g., out of range), it should

display an error message.

Input Format

The first line contains an integer n, representing the number of items to be initially entered into the inventory.

The second line contains n integers, each representing the unique identification number of an item separated by spaces.

The third line contains an integer p, representing the position of the item to be deleted from the inventory.

Output Format

The first line of output prints "Data entered in the list:" followed by the data values of each node in the doubly linked list before deletion.

If p is an invalid position, the output prints "Invalid position. Try again."

If p is a valid position, the output prints "After deletion the new list:" followed by the data values of each node in the doubly linked list after deletion.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 4
1 2 3 4
5
Output: Data entered in the list:
node 1 : 1
node 2 : 2
node 3 : 3
node 4 : 4
Invalid position. Try again.

Answer

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

```
// Node structure for doubly linked list
struct Node {
    int data;          // item identification number
    struct Node* next;  // pointer to next node
    struct Node* prev;  // pointer to previous node
};

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

// Function to insert a node at the end of the list
void insertAtEnd(struct Node** head, struct Node** tail, int data) {
    struct Node* newNode = createNode(data);

    if (*head == NULL) {
        // If the list is empty
        *head = newNode;
        *tail = newNode;
    } else {
        // Add the new node at the end
        (*tail)->next = newNode;
        newNode->prev = *tail;
        *tail = newNode;
    }
}

// Function to delete a node at a specific position
int deleteAtPosition(struct Node** head, struct Node** tail, int position) {
    // If list is empty
    if (*head == NULL) {
        return 0;
```

```
}

// Count the number of nodes
int count = 0;
struct Node* temp = *head;
while (temp != NULL) {
    count++;
    temp = temp->next;
}

// Check if position is valid
if (position < 1 || position > count) {
    return 0;
}

// If deleting the first node
if (position == 1) {
    struct Node* temp = *head;
    *head = (*head)->next;

    if (*head != NULL) {
        (*head)->prev = NULL;
    } else {
        // If the list becomes empty
        *tail = NULL;
    }

    free(temp);
    return 1;
}

// If deleting the last node
if (position == count) {
    struct Node* temp = *tail;
    *tail = (*tail)->prev;
    (*tail)->next = NULL;
    free(temp);
    return 1;
}

// If deleting a node in the middle
struct Node* current = *head;
```

```
    for (int i = 1; i < position; i++) {
        current = current->next;
    }

    current->prev->next = current->next;
    current->next->prev = current->prev;
    free(current);

    return 1;
}

// Function to display the list
void displayList(struct Node* head) {
    struct Node* current = head;
    int nodeCount = 1;

    while (current != NULL) {
        printf(" node %d : %d\n", nodeCount++, current->data);
        current = current->next;
    }
}

// Function to free the memory allocated for the list
void freeList(struct Node* head) {
    struct Node* current = head;
    struct Node* next;

    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }
}

int main() {
    struct Node* head = NULL;
    struct Node* tail = NULL;
    int n, item, position;

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

```
    // Read and insert each item
    for (int i = 0; i < n; i++) {
        scanf("%d", &item);
        insertAtEnd(&head, &tail, item);
    }

    // Read the position to delete
    scanf("%d", &position);

    // Display the initial list
    printf("Data entered in the list:\n");
    displayList(head);

    // Delete the item at the specified position
    int deleteResult = deleteAtPosition(&head, &tail, position);

    // Check if deletion was successful
    if (deleteResult == 0) {
        printf("Invalid position. Try again.\n");
    } else {
        printf("\n After deletion the new list:\n");
        displayList(head);
    }

    // Free allocated memory
    freeList(head);

    return 0;
}
```

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