Rajalakshmi Engineering College

Name: Harish Vidyarth
Email: 241901035@rajalakshmi.edu.in
Roll no: 241901035
Phone: 6382551983
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_MCQ_Updated

Attempt : 1
Total Mark : 20
Marks Obtained : 17

Section 1 : MCQ

1. What is the main advantage of a two-way linked list over a one-way linked list?

Answer

Two-way linked lists allow for traversal in both directions.

Status : Correct ***Marks : 1/1***

2. Consider the provided pseudo code. How can you initialize an empty two-way linked list?

Define Structure Node
  data: Integer
  prev: Pointer to Node
  next: Pointer to Node

End Define

Define Structure TwoWayLinkedList
  head: Pointer to Node
  tail: Pointer to Node
End Define

Answer

struct TwoWayLinkedList* list = malloc(sizeof(struct TwoWayLinkedList)); list->head = NULL; list->tail = NULL;

Status : Correct ***Marks : 1/1***

3. What is the correct way to add a node at the beginning of a doubly linked list?

Answer

void addFirst(int data){ Node* newNode = new Node(data); newNode->next = head; if (head != NULL) { head->prev = newNode; } head = newNode; }

Status : Correct ***Marks : 1/1***

4. Where Fwd and Bwd represent forward and backward links to the adjacent elements of the list. Which of the following segments of code deletes the node pointed to by X from the doubly linked list, if it is assumed that X points to neither the first nor the last node of the list?

A doubly linked list is declared as

```
struct Node {
    int Value;
    struct Node *Fwd;
    struct Node *Bwd;
);
```

Answer

X->Bwd.Fwd = X->Fwd ; X.Fwd->Bwd = X->Bwd;

Status : Wrong ***Marks : 0/1***

5. What will be the output of the following program?

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

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

int main() {
    struct Node* head = NULL;
    struct Node* tail = NULL;
    for (int i = 0; i < 5; i++) {
        struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = i + 1;
        temp->prev = tail;
        temp->next = NULL;
        if (tail != NULL) {
            tail->next = temp;
        } else {
            head = temp;
        }
        tail = temp;
    }
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    return 0;
}
```

Answer

1 2 3 4 5

Status : Correct ***Marks : 1/1***

6. Which pointer helps in traversing a doubly linked list in reverse order?

Answer

prev

Status : Correct ***Marks : 1/1***

7. What will be the output of the following code?

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

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

int main() {
    struct Node* head = NULL;
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = 2;
    temp->next = NULL;
    temp->prev = NULL;
    head = temp;
    printf("%d\n", head->data);
    free(temp);
    return 0;
}
```

Answer

2

Status : Correct ***Marks : 1/1***

8. Which of the following is true about the last node in a doubly linked list?

Answer

Its next pointer is NULL

Status : Correct ***Marks : 1/1***

9. Which of the following statements correctly creates a new node for a doubly linked list?

Answer

struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));

Status : Correct ***Marks : 1/1***

10. Which of the following information is stored in a doubly-linked list's nodes?

Answer

All of the mentioned options

Status : Correct ***Marks : 1/1***

11. How many pointers does a node in a doubly linked list have?

Answer

2

Status : Correct ***Marks : 1/1***

12. How do you reverse a doubly linked list?

Answer

By swapping the next and previous pointers of each node

Status : Correct ***Marks : 1/1***

13. What does the following code snippet do?

struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

```
newNode->data = value;
newNode->next = NULL;
newNode->prev = NULL;
```

Answer

Creates a new node and initializes its data to 'value'

Status : Correct ***Marks : 1/1***

14. What happens if we insert a node at the beginning of a doubly linked list?

Answer

The previous pointer of the new node is NULL

Status : Correct ***Marks : 1/1***

15. What is a memory-efficient double-linked list?

Answer

The list has breakpoints for faster traversal

Status : Wrong ***Marks : 0/1***

16. Consider the following function that refers to the head of a Doubly Linked List as the parameter. Assume that a node of a doubly linked list has the previous pointer as prev and the next pointer as next.

Assume that the reference of the head of the following doubly linked list is passed to the below function 1 <--> 2 <--> 3 <--> 4 <--> 5 <-->6. What should be the modified linked list after the function call?

```
Procedure fun(head_ref: Pointer to Pointer of node)
   temp = NULL
   current = *head_ref

   While current is not NULL
      temp = current->prev
```

```
    current->prev = current->next
    current->next = temp
    current = current->prev
  End While

  If temp is not NULL
    *head_ref = temp->prev
  End If
End Procedure
```

Answer

6 <--> 5 <--> 4 <--> 3 <--> 2 <--> 1.

Status : Correct ***Marks : 1/1***

17. Which code snippet correctly deletes a node with a given value from a doubly linked list?

```
void deleteNode(Node** head_ref, Node* del_node) {
  if (*head_ref == NULL || del_node == NULL) {
    return;
  }
  if (*head_ref == del_node) {
    *head_ref = del_node->next;
  }
  if (del_node->next != NULL) {
    del_node->next->prev = del_node->prev;
  }
  if (del_node->prev != NULL) {
    del_node->prev->next = del_node->next;
  }
  free(del_node);
}
```

Answer

Deletes the node at a given position in a doubly linked list.

Status : Wrong ***Marks : 0/1***

18. How do you delete a node from the middle of a doubly linked list?

Answer

All of the mentioned options

Status : Correct ***Marks : 1/1***

19. Which of the following is false about a doubly linked list?

Answer

Implementing a doubly linked list is easier than singly linked list

Status : Correct ***Marks : 1/1***

20. What will be the effect of setting the prev pointer of a node to NULL in a doubly linked list?

Answer

The node will become the new head

Status : Correct ***Marks : 1/1***

Rajalakshmi Engineering College

Name: Harish Vidyarth
Email: 241901035@rajalakshmi.edu.in
Roll no: 241901035
Phone: 6382551983
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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;
};

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

    if (*head == NULL) {
        newNode->prev = NULL;
        *head = newNode;
        return;
    }
```

```
    struct Node* last = *head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
    newNode->prev = last;
}
void displayForward(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%c ", current->item);
        current = current->next;
    }
}

void displayBackward(struct Node* tail) {
    struct Node* current = tail;
    if (current == NULL) return;
    while (current->next != NULL) {
        current = current->next;
    }

    while (current != NULL) {
        printf("%c ", current->item);
        current = current->prev;
    }
}

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

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: Harish Vidyarth
Email: 241901035@rajalakshmi.edu.in
Roll no: 241901035
Phone: 6382551983
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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

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

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

struct DoublyLinkedList {
    struct Node* head;
    struct Node* tail;
};

struct DoublyLinkedList* createList() {
    struct DoublyLinkedList* list = (struct DoublyLinkedList*)malloc(sizeof(struct
DoublyLinkedList));
    list->head = NULL;
    list->tail = NULL;
    return list;
}

void append(struct DoublyLinkedList* list, int id) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->id = id;
```

```
    newNode->next = NULL;
    if (list->head == NULL) {
        newNode->prev = NULL;
        list->head = newNode;
        list->tail = newNode;
        return;
    }
    newNode->prev = list->tail;
    list->tail->next = newNode;
    list->tail = newNode;
}

int printMaxID(struct DoublyLinkedList* list) {
    if (list->head == NULL) {
        return -1;
    }
    int maxID = list->head->id;
    struct Node* current = list->head;
    while (current != NULL) {
        if (current->id > maxID) {
            maxID = current->id;
        }
        current = current->next;
    }
    return maxID;
}

int main() {
    int n;
    scanf("%d", &n);
    struct DoublyLinkedList* list = createList();
    for (int i = 0; i < n; i++) {
        int id;
        scanf("%d", &id);
        append(list, id);
    }
    int maxID = printMaxID(list);
    if (maxID == -1) {
        printf("Empty list!\n");
    } else {
        printf("%d\n", maxID);
    }
```

```
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Harish Vidyarth
Email: 241901035@rajalakshmi.edu.in
Roll no: 241901035
Phone: 6382551983
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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;

void traverse() {
   struct node* current = start;
   while(current!=NULL) {
      printf("%d \n",current->info);
      current=current->next;
   }
}

void insertAtFront(int data) {
```

```
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->info = data;
    newNode->next = start;
    newNode->prev = NULL;
    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: Harish Vidyarth
Email: 241901035@rajalakshmi.edu.in
Roll no: 241901035
Phone: 6382551983
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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

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

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

struct DoublyLinkedList {
    struct Node* head;
    struct Node* tail;
};

struct DoublyLinkedList* createList() {
    struct DoublyLinkedList* list = (struct DoublyLinkedList*)malloc(sizeof(struct
DoublyLinkedList));
    list->head = NULL;
    list->tail = NULL;
    return list;
}

void insert(struct DoublyLinkedList* list, int id) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

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

void display(struct DoublyLinkedList* list) {
    struct Node* current = list->head;
    while (current != NULL) {
        printf("%d ", current->id);
        current = current->next;
    }
}

int main() {
    int N;
    scanf("%d", &N);
    struct DoublyLinkedList* list = createList();
    for (int i = 0; i < N; i++) {
        int id;
        scanf("%d", &id);
        insert(list, id);
    }
    display(list);
    printf("\n");
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Harish Vidyarth
Email: 241901035@rajalakshmi.edu.in
Roll no: 241901035
Phone: 6382551983
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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

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

```
struct Node {
    int id;
    struct Node* next;
    struct Node* prev;
};

struct DoublyLinkedList {
    struct Node* head;
    struct Node* tail;
};

struct DoublyLinkedList* createList() {
    struct DoublyLinkedList* list = (struct DoublyLinkedList*)malloc(sizeof(struct
DoublyLinkedList));
    list->head = NULL;
    list->tail = NULL;
    return list;
}

void insert(struct DoublyLinkedList* list, int id) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->id = id;
    newNode->next = NULL;
    if (list->head == NULL) {
        newNode->prev = NULL;
        list->head = newNode;
        list->tail = newNode;
        return;
    }
    newNode->prev = list->tail;
    list->tail->next = newNode;
    list->tail = newNode;
}

void display(struct DoublyLinkedList* list) {
    struct Node* current = list->head;
    int index = 1;
    while (current != NULL) {
        printf("node %d : %d\n", index, current->id);
        current = current->next;
        index++;
    }
```

```
}

void deleteNode(struct DoublyLinkedList* list, int position) {
    if (position < 1) {
        printf("Invalid position. Try again.\n");
        return;
    }
    struct Node* current = list->head;
    for (int i = 1; i < position && current != NULL; i++) {
        current = current->next;
    }
    if (current == NULL) {
        printf("Invalid position. Try again.\n");
        return;
    }
    if (current->prev != NULL) {
        current->prev->next = current->next;
    } else {
        list->head = current->next;
    }
    if (current->next != NULL) {
        current->next->prev = current->prev;
    } else {
        list->tail = current->prev;
    }
    free(current);
}

int main() {
    int n, p;
    scanf("%d", &n);
    struct DoublyLinkedList* list = createList();
    for (int i = 0; i < n; i++) {
        int id;
        scanf("%d", &id);
        insert(list, id);
    }
    printf("Data entered in the list:\n");
    display(list);
    scanf("%d", &p);
    deleteNode(list, p);
    printf("\n");
```

```
    if (p >= 1 && p <= n) {
        printf("After deletion the new list:\n");
        display(list);
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Harish Vidyarth
Email: 241901035@rajalakshmi.edu.in
Roll no: 241901035
Phone: 6382551983
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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

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>

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

struct DoublyLinkedList {
    struct Node* head;
};

struct DoublyLinkedList* createList() {
```

```
    struct DoublyLinkedList* list = (struct DoublyLinkedList*)malloc(sizeof(struct
DoublyLinkedList));
    list->head = NULL;
    return list;
}

void insertEnd(struct DoublyLinkedList* list, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (list->head == NULL) {
        newNode->prev = NULL;
        list->head = newNode;
        return;
    }
    struct Node* last = list->head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
    newNode->prev = last;
}

void insertAtPosition(struct DoublyLinkedList* list, int data, int position) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    if (position == 1) {
        newNode->next = list->head;
        newNode->prev = NULL;
        if (list->head != NULL) {
            list->head->prev = newNode;
        }
        list->head = newNode;
        return;
    }
    struct Node* temp = list->head;
    for (int i = 1; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        free(newNode);
        return;
```

```
    }
    newNode->next = temp->next;
    temp->next = newNode;
    newNode->prev = temp;
    if (newNode->next != NULL) {
        newNode->next->prev = newNode;
    }
}

void displayList(struct DoublyLinkedList* list) {
    struct Node* temp = list->head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int n;
    scanf("%d", &n);
    struct DoublyLinkedList* list = createList();
    for (int i = 0; i < n; i++) {
        int value;
        scanf("%d", &value);
        insertEnd(list, value);
    }
    int m, p;
    scanf("%d", &m);
    scanf("%d", &p);
    if (p < 1 || p > n + 1) {
        printf("Invalid position\n");
        displayList(list);
    } else {
        insertAtPosition(list, m, p);
        displayList(list);
    }
    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>

struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
```

```
};

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;
}

struct Node* insertAtBeginning(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head != NULL) {
        newNode->next = head;
        head->prev = newNode;
    }
    return newNode;
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

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

```
int main() {
    int N;
    scanf("%d", &N);
    int values[N];
    for (int i = 0; i < N; i++) {
        scanf("%d", &values[i]);
    }

    struct Node* head = NULL;
    for (int i = 0; i < N; i++) {
        head = insertAtBeginning(head, values[i]);
    }
    printList(head);

    struct Node* endHead = NULL;
    for (int i = 0; i < N; i++) {
        insertAtEnd(&endHead, values[i]);
    }
    printList(endHead);

    return 0;
}
```

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

3. 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

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

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

struct DoublyLinkedList {
    struct Node* head;
};

struct DoublyLinkedList* createList() {
    struct DoublyLinkedList* list = (struct DoublyLinkedList*)malloc(sizeof(struct
DoublyLinkedList));
    list->head = NULL;
    return list;
}

void insertEnd(struct DoublyLinkedList* list, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (list->head == NULL) {
        newNode->prev = NULL;
        list->head = newNode;
```

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

int findMaxScore(struct DoublyLinkedList* list) {
    struct Node* temp = list->head;
    int maxScore = temp->data;
    while (temp != NULL) {
        if (temp->data > maxScore) {
            maxScore = temp->data;
        }
        temp = temp->next;
    }
    return maxScore;
}

int main() {
    int N;
    scanf("%d", &N);
    struct DoublyLinkedList* list = createList();
    for (int i = 0; i < N; i++) {
        int score;
        scanf("%d", &score);
        insertEnd(list, score);
    }
    int maxScore = findMaxScore(list);
    printf("%d\n", maxScore);
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Harish Vidyarth
Email: 241901035@rajalakshmi.edu.in
Roll no: 241901035
Phone: 6382551983
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. Problem Statement

Rohan is a software developer who is working on an application that processes data stored in a Doubly Linked List. He needs to implement a feature that finds and prints the middle element(s) of the list. If the list contains an odd number of elements, the middle element should be printed. If the list contains an even number of elements, the two middle elements should be printed.

Help Rohan by writing a program that reads a list of numbers, prints the list, and then prints the middle element(s) based on the number of elements in the list.

Input Format

The first line of the input consists of an integer n the number of elements in the

doubly linked list.

The second line consists of n space-separated integers representing the elements of the list.

Output Format

The first line prints the elements of the list separated by space. (There is an extra space at the end of this line.)

The second line prints the middle element(s) based on the number of elements.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
20 52 40 16 18
Output: 20 52 40 16 18
40

Answer

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

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

struct Node* head = NULL;

// Function to insert a new node at the end of the list
void insert(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

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

```
        head = newNode;
        return;
    }

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

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

// Function to print the list
void printList() {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n"); // Print a newline at the end of the list
}

// Function to find and print the middle element(s)
void printMiddle() {
    struct Node* slow = head;
    struct Node* fast = head;

    // Move fast pointer two nodes and slow pointer one node
    while (fast != NULL && fast->next != NULL) {
        fast = fast->next->next;
        slow = slow->next;
    }

    // If fast is NULL, the number of nodes is even
    if (fast == NULL) {
        // Print the two middle nodes
        printf("%d %d\n", slow->prev->data, slow->data);
    } else { // If fast is not NULL, the number of nodes is odd
        // Print the middle node
        printf("%d\n", slow->data);
    }
```

```
}

int main() {
    int n;

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

    // Read elements and insert them into the doubly linked list
    for (int i = 0; i < n; i++) {
        scanf("%d", &element);
        insert(element);
    }

    // Print the list
    printList();

    // Print the middle element(s)
    printMiddle();

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

    return 0;
}
```

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

2. Problem Statement

Tom is a software developer working on a project where he has to check if a doubly linked list is a palindrome. He needs to write a program to solve this problem. Write a program to help Tom check if a given doubly linked

list is a palindrome or not.

Input Format

The first line consists of an integer N, representing the number of elements in the linked list.

The second line consists of N space-separated integers representing the linked list elements.

Output Format

The first line displays the space-separated integers, representing the doubly linked list.

The second line displays one of the following:

1. If the doubly linked list is a palindrome, print "The doubly linked list is a palindrome".
2. If the doubly linked list is not a palindrome, print "The doubly linked list is not a palindrome".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
1 2 3 2 1
Output: 1 2 3 2 1
The doubly linked list is a palindrome

Answer

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

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

```
struct Node* head = NULL;

void insert(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if (head == NULL) {
        newNode->prev = NULL;
        head = newNode;
        return;
    }

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

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

void printList() {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int isPalindrome() {
    struct Node* left = head;
    struct Node* right = head;

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

    while (left != NULL && right != NULL && left != right && left->prev != right) {
        if (left->data != right->data) {
            return 0;
```

```
        }
        left = left->next;
        right = right->prev;
    }
    return 1;
}

int main() {
    int n;


    scanf("%d", &n);
    int element;


    for (int i = 0; i < n; i++) {
        scanf("%d", &element);
        insert(element);
    }

    printList();


    if (isPalindrome()) {
        printf("The doubly linked list is a palindrome\n");
    } else {
        printf("The doubly linked list is not a palindrome\n");
    }


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

    return 0;
}
```

Status : Correct *Marks : 10/10*

3. Problem Statement

Pranav wants to clockwise rotate a doubly linked list by a specified number of positions. He needs your help to implement a program to achieve this. Given a doubly linked list and an integer representing the number of positions to rotate, write a program to rotate the list clockwise.

Input Format

The first line of input consists of an integer n, representing the number of elements in the linked list.

The second line consists of n space-separated linked list elements.

The third line consists of an integer k, representing the number of places to rotate the list.

Output Format

The output displays the elements of the doubly linked list after rotating it by k positions.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
1 2 3 4 5
1

Output: 5 1 2 3 4

Answer

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

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

```
struct Node* head = NULL;

void insert(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if (head == NULL) {
        newNode->prev = NULL;
        head = newNode;
        return;
    }

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

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


void printList() {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n"); // Print a newline at the end of the list
}

// Function to rotate the doubly linked list clockwise by k positions
void rotateClockwise(int k) {
    if (head == NULL || head->next == NULL || k == 0) {
        return; // No rotation needed
    }

    struct Node* current = head;
    struct Node* last = head;
    int count = 1;
```

```
    // Traverse to the end of the list and count the nodes
    while (last->next != NULL) {
        last = last->next;
        count++;
    }

    // If k is greater than count, reduce k
    k = k % count;

    // If k is 0 after modulo, no rotation is needed
    if (k == 0) {
        return;
    }

    // Traverse to the (count - k)th node
    current = head;
    for (int i = 1; i < count - k; i++) {
        current = current->next;
    }

    // Update the head and pointers
    struct Node* newHead = current->next;
    current->next = NULL;
    newHead->prev = NULL;
    last->next = head;
    head->prev = last;
    head = newHead;
}

int main() {
    int n, k;

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

    // Read elements and insert them into the doubly linked list
    for (int i = 0; i < n; i++) {
        scanf("%d", &element);
        insert(element);
    }
```

```
    // Read the number of positions to rotate
    scanf("%d", &k);

    // Rotate the list
    rotateClockwise(k);

    // Print the rotated list
    printList();

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

    return 0;
}
```

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

4. Problem Statement

Riya is developing a contact management system where recently added contacts should appear first. She decides to use a doubly linked list to store contact IDs in the order they are added. Initially, new contacts are inserted at the front of the list. However, sometimes she needs to insert a new contact at a specific position in the list based on priority.

Help Riya implement this system by performing the following operations:

Insert contact IDs at the front of the list as they are added.Insert a new contact at a given position in the list.

Input Format

The first line of input consists of an integer N, representing the initial size of the linked list.

The second line consists of N space-separated integers, representing the values

of the linked list to be inserted at the front.

The third line consists of an integer position, representing the position at which the new value should be inserted (position starts from 1).

The fourth line consists of integer data, representing the new value to be inserted.

Output Format

The first line of output prints the original list after inserting initial elements to the front.

The second line prints the updated linked list after inserting the element at the specified position.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
10 20 30 40
3
25

Output: 40 30 20 10
40 30 25 20 10

Answer

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

struct Node {
    int data;             // Data stored in the node
    struct Node* next;    // Pointer to the next node
    struct Node* prev;    // Pointer to the previous node
};

// Global head pointer for the doubly linked list
struct Node* head = NULL;

// Function to insert a new node at the front of the list
```

```
void insertAtFront(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = head;
    newNode->prev = NULL;

    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;
}

// Function to insert a new node at a specific position
void insertAtPosition(int data, int position) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;

    if (position == 1) {
        insertAtFront(data);
        return;
    }

    struct Node* current = head;
    for (int i = 1; i < position - 1 && current != NULL; i++) {
        current = current->next;
    }

    if (current == NULL) {
        printf("Position out of bounds\n");
        free(newNode);
        return;
    }

    newNode->next = current->next;
    newNode->prev = current;

    if (current->next != NULL) {
        current->next->prev = newNode;
    }
    current->next = newNode;
}
```

```
// Function to print the list
void printList() {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n"); // Print a newline at the end of the list
}

int main() {
    int N, position, data;

    // Read the initial size of the linked list
    scanf("%d", &N);
    int element;

    // Read elements and insert them at the front of the doubly linked list
    for (int i = 0; i < N; i++) {
        scanf("%d", &element);
        insertAtFront(element);
    }

    // Print the original list
    printList();

    // Read the position to insert the new value
    scanf("%d", &position);
    // Read the new value to be inserted
    scanf("%d", &data);

    // Insert the new value at the specified position
    insertAtPosition(data, position);

    // Print the updated list
    printList();

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

```
        free(current);
        current = nextNode;
    }

    return 0;
}
```

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

5. Problem Statement

Bala is a student learning about the doubly linked list and its functionalities. He came across a problem where he wanted to create a doubly linked list by appending elements to the front of the list.

After populating the list, he wanted to delete the node at the given position from the beginning. Write a suitable code to help Bala.

Input Format

The first line contains an integer N, the number of elements in the doubly linked list.

The second line contains N integers separated by a space, the data values of the nodes in the doubly linked list.

The third line contains an integer X, the position of the node to be deleted from the doubly linked list.

Output Format

The first line of output displays the original elements of the doubly linked list, separated by a space.

The second line prints the updated list after deleting the node at the given position X from the beginning.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 20 30 40 50
2
Output: 50 40 30 20 10
50 30 20 10

Answer

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

struct Node {
    int data;             // Data stored in the node
    struct Node* next;    // Pointer to the next node
    struct Node* prev;    // Pointer to the previous node
};

struct Node* head = NULL;

void insertAtFront(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = head;
    newNode->prev = NULL;

    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;
}

void deleteAtPosition(int position) {
    if (head == NULL || position <= 0) {
        return; // No deletion needed
    }

    struct Node* current = head;
    if (position == 1) {
        head = current->next;
        if (head != NULL) {
            head->prev = NULL;
```

```
        }
        free(current); // Free the old head
        return;
    }

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

    // If the position is out of bounds
    if (current == NULL) {
        return;
    }

    // Update pointers to remove the current node
    if (current->next != NULL) {
        current->next->prev = current->prev; // Update previous pointer of next node
    }
    if (current->prev != NULL) {
        current->prev->next = current->next; // Update next pointer of previous node
    }

    free(current); // Free the node
}

// Function to print the list
void printList() {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n"); // Print a newline at the end of the list
}

int main() {
    int N, X;

    // Read the number of elements in the doubly linked list
    scanf("%d", &N);
    int element;
```

```
    // Read elements and insert them at the front of the doubly linked list
    for (int i = 0; i < N; i++) {
        scanf("%d", &element);
        insertAtFront(element);
    }

    // Print the original list
    printList();

    // Read the position of the node to be deleted
    scanf("%d", &X);

    // Delete the node at the specified position
    deleteAtPosition(X);

    // Print the updated list
    printList();

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

    return 0;
}
```

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