# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

## REC_DS using C_Week 1_MCQ

Attempt : 1
Total Mark : 10
Marks Obtained : 6

### Section 1 : MCQ

1. Consider the singly linked list: 15 -> 16 -> 6 -> 7 -> 17. You need to delete all nodes from the list which are prime.

What will be the final linked list after the deletion?

***Answer***

16 -> 6

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

2. The following function reverse() is supposed to reverse a singly linked list. There is one line missing at the end of the function.

What should be added in place of "/*ADD A STATEMENT HERE*/", so that the function correctly reverses a linked list?

```
struct node {
   int data;
   struct node* next;
};
static void reverse(struct node** head_ref) {
   struct node* prev   = NULL;
   struct node* current = *head_ref;
   struct node* next;
   while (current != NULL) {
      next  = current->next;
      current->next = prev;
      prev = current;
      current = next;
   }
   /*ADD A STATEMENT HERE*/
}
```

***Answer***

***Status :*** Skipped ***Marks : 0/1***

3. Linked lists are not suitable for the implementation of?

***Answer***

Binary search

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

4. Consider an implementation of an unsorted singly linked list. Suppose it has its representation with a head pointer only. Given the representation, which of the following operations can be implemented in O(1) time?

i) Insertion at the front of the linked list

ii) Insertion at the end of the linked list

iii) Deletion of the front node of the linked list

iv) Deletion of the last node of the linked list

***Answer***

I and III

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

5. Which of the following statements is used to create a new node in a singly linked list?

```
struct node {
   int data;
   struct node * next;
}
typedef struct node NODE;
NODE *ptr;
```

***Answer***

ptr = (NODE*)malloc(sizeof(NODE));

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

6. Consider the singly linked list: 13 -> 4 -> 16 -> 9 -> 22 -> 45 -> 5 -> 16 -> 6, and an integer K = 10, you need to delete all nodes from the list that are less than the given integer K.

What will be the final linked list after the deletion?

***Answer***

13 -> 4 -> 16 -> 9 -> 22 -> 45 -> 5 -> 16 -> 6

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

7. Given the linked list: 5 -> 10 -> 15 -> 20 -> 25 -> NULL. What will be the output of traversing the list and printing each node's data?

***Answer***

5 10 15 20 25

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

8. In a singly linked list, what is the role of the "tail" node?

***Answer***

It stores the last element of the list

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

9. Given a pointer to a node X in a singly linked list. If only one point is given and a pointer to the head node is not given, can we delete node X from the given linked list?

***Answer***

Possible if X is not last node.

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

10. The following function takes a singly linked list of integers as a parameter and rearranges the elements of the lists.

The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the list after the function completes execution?

```
struct node {
   int value;
   struct node* next;
};

void rearrange (struct node* list) {
   struct node *p,q;
   int temp;
   if (! List || ! list->next) return;
   p=list; q=list->next;
   while(q) {
      temp=p->value; p->value=q->value;
      q->value=temp;p=q->next;
      q=p?p->next:0;
   }
```

}

***Answer***

***Status :*** Skipped ***Marks : 0/1***

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 1_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

Janani is a tech enthusiast who loves working with polynomials. She wants to create a program that can add polynomial coefficients and provide the sum of their coefficients.

The polynomials will be represented as a linked list, where each node of the linked list contains a coefficient and an exponent. The polynomial is represented in the standard form with descending order of exponents.

***Input Format***

The first line of input consists of an integer n, representing the number of terms in the first polynomial.

The following n lines of input consist of two integers each: the coefficient and the exponent of the term in the first polynomial.

The next line of input consists of an integer m, representing the number of terms in the second polynomial.

The following m lines of input consist of two integers each: the coefficient and the exponent of the term in the second polynomial.

***Output Format***

The output prints the sum of the coefficients of the polynomials.

***Sample Test Case***

Input: 3
2 2
3 1
4 0
3
2 2
3 1
4 0

Output: 18

***Answer***

```
// You are using GCC
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
   int data1,data2;
   int exp;
   struct node*link;
}node;
int main(){
   int n,m;
   int sum=0;
   struct node no;
   scanf("%d",&n);
   for(int i=0;i<n;i++){
      scanf("%d %d",&no.data1,&no.data2);
      sum+=no.data1;
   }
   scanf("%d",&m);
   for(int i=0;i<m;i++){
```

```
        scanf("%d %d",&no.data1,&no.data2);
        sum+=no.data1;
    }
    printf("%d",sum);
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 1_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

Arun is learning about data structures and algorithms. He needs your help in solving a specific problem related to a singly linked list.

Your task is to implement a program to delete a node at a given position. If the position is valid, the program should perform the deletion; otherwise, it should display an appropriate message.

***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 elements of the linked list.

The third line consists of an integer x, representing the position to delete.

Position starts from 1.

***Output Format***

The output prints space-separated integers, representing the updated linked list after deleting the element at the given position.

If the position is not valid, print "Invalid position. Deletion not possible."

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
8 2 3 1 7
2

Output: 8 3 1 7

***Answer***

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

void insert(int);
void display_List();
void deleteNode(int);

struct node {
   int data;
   struct node* next;
} *head = NULL, *tail = NULL;
```

```
// You are using GCC
#include<stdio.h>
#include<stdlib.h>
struct Node{
   int data;
   struct Node* next;
};
void deleteNode(struct Node** head_ref, int position){
   if(*head_ref == NULL){
      printf("Invalid position. Deletion not possible.\n");
```

```
        return;
    }
    struct Node* temp = *head_ref;
    if(position == 1){
        *head_ref =temp->next;
        free(temp);
        return;
    }
    for(int i=1; temp!= NULL && i<position-1;i++){
        temp=temp->next;
    }
    if(temp == NULL || temp->next == NULL){
        printf("Invalid position. Deletion not possible.\n");
        return;
    }
    struct Node* next = temp->next->next;
    free(temp->next);
    temp->next = next;
}
printf("\n");
}
void push(struct Node** head_ref, int new_data){
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = *head_ref;
    *head_ref = new_node;
}
int main() {
    int n, x;
    struct Node* head = NULL;
    scanf("%d", &n);
    int elements[n];
    for(int i=0;i<n;i++) {
        scanf("%d", &elements[i]);
    }
    scanf("%d", &x);
    if(x < 1 || x > n) {
        printf("Invalid position. Deletion not possible.\n");
    } else {
        deletion(&head, x);
        print_List(head);
    }
```

```
    return 0;
}

int main() {
    int num_elements, element, pos_to_delete;

    scanf("%d", &num_elements);

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

    scanf("%d", &pos_to_delete);

    deleteNode(pos_to_delete);

    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

## REC_DS using C_Week 1_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 0

## Section 1 : Coding

1. Problem Statement

Imagine you are working on a text processing tool and need to implement a feature that allows users to insert characters at a specific position.

Implement a program that takes user inputs to create a singly linked list of characters and inserts a new character after a given index in the list.

***Input Format***

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

The second line consists of a sequence of N characters, representing the linked list.

The third line consists of an integer index, representing the index(0-based) after

which the new character node needs to be inserted.

The fourth line consists of a character value representing the character to be inserted after the given index.

***Output Format***

If the provided index is out of bounds (larger than the list size):

1. The first line of output prints "Invalid index".
2. The second line prints "Updated list: " followed by the unchanged linked list values.

Otherwise, the output prints "Updated list: " followed by the updated linked list after inserting the new character after the given index.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
a b c d e
2
X
Output: Updated list: a b c X d e

***Answer***

-

***Status :*** Skipped ***Marks : 0/10***

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 1_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

As part of a programming assignment in a data structures course, students are required to create a program to construct a singly linked list by inserting elements at the beginning.

You are an evaluator of the course and guide the students to complete the task.

***Input Format***

The first line of input consists of an integer N, which is the number of elements.

The second line consists of N space-separated integers.

***Output Format***

The output prints the singly linked list elements, after inserting them at the beginning.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
78 89 34 51 67

Output: 67 51 34 89 78

***Answer***

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

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

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

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

int main(){
```

```
    struct Node* head = NULL;

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

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

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

    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

## REC_DS using C_Week 1_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 0

## Section 1 : Coding

1. Problem Statement

Imagine you are tasked with developing a simple GPA management system using a singly linked list. The system allows users to input student GPA values, insertion should happen at the front of the linked list, delete record by position, and display the updated list of student GPAs.

***Input Format***

The first line of input contains an integer n, representing the number of students.

The next n lines contain a single floating-point value representing the GPA of each student.

The last line contains an integer position, indicating the position at which a student record should be deleted. Position starts from 1.

***Output Format***

After deleting the data in the given position, display the output in the format "GPA: " followed by the GPA value, rounded off to one decimal place.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 4
3.8
3.2
3.5
4.1
2
Output: GPA: 4.1
GPA: 3.2
GPA: 3.8

***Answer***

-

***Status :*** Skipped ***Marks : 0/10***

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 1_COD_Question 6

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

John is tasked with creating a program to manage student roll numbers using a singly linked list.

Write a program for John that accepts students' roll numbers, inserts them at the end of the linked list, and displays the numbers.

***Input Format***

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

The second line consists of N space-separated integers, representing the roll numbers of students.

***Output Format***

The output prints the space-separated integers singly linked list, after inserting the roll numbers of students at the end.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
23 85 47 62 31

Output: 23 85 47 62 31

***Answer***

*-*

***Status :*** Skipped ***Marks : 0/10***

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 1_COD_Question 7

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

Dev is tasked with creating a program that efficiently finds the middle element of a linked list. The program should take user input to populate the linked list by inserting each element into the front of the list and then determining the middle element.

Assist Dev, as he needs to ensure that the middle element is accurately identified from the constructed singly linked list:

If it's an odd-length linked list, return the middle element.If it's an even-length linked list, return the second middle element of the two elements.

***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 integers, representing the elements of the list.

***Output Format***

The first line of output displays the linked list after inserting elements at the front.

The second line displays "Middle Element: " followed by the middle element of the linked list.

Refer to the sample output for formatting specifications.

***Sample Test Case***

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

***Answer***

-

***Status :*** Skipped ***Marks : 0/10***

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

## REC_DS using C_Week 1_PAH_modified

Attempt : 1
Total Mark : 5
Marks Obtained : 3.4

## Section 1 : Coding

1. Problem Statement

Write a program to manage a singly linked list. The program should allow users to perform various operations on the linked list, such as inserting elements at the beginning or end, deleting elements from the beginning or end, inserting before or after a specific value, and deleting elements before or after a specific value. After each operation, the updated linked list should be displayed.

***Input Format***

The first line contains an integer choice, representing the operation to perform:

- For choice 1 to create the linked list. The next lines contain space-separated integers, with -1 indicating the end of input.
- For choice 2 to display the linked list.
- For choice 3 to insert a node at the beginning. The next line contains an integer

data representing the value to insert.
- For choice 4 to insert a node at the end. The next line contains an integer data representing the value to insert.
- For choice 5 to insert a node before a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 6 to insert a node after a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 7 to delete a node from the beginning.
- For choice 8 to delete a node from the end.
- For choice 9 to delete a node before a specific value. The next line contains an integer value representing the node before which deletion occurs.
- For choice 10 to delete a node after a specific value. The next line contains an integer value representing the node after which deletion occurs.
- For choice 11 to exit the program.

***Output Format***

For choice 1, print "LINKED LIST CREATED".

For choice 2, print the linked list as space-separated integers on a single line. If the list is empty, print "The list is empty".

For choice 3, 4, 5, and 6, print the updated linked list with a message indicating the insertion operation.

For choice 7, 8, 9, and 10, print the updated linked list with a message indicating the deletion operation.

For any operation that is not possible print an appropriate error message such as "Value not found in the list".

For choice 11 terminate the program.

For any invalid option, print "Invalid option! Please try again".

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 1
5

3
7
-1
2
11
Output: LINKED LIST CREATED
5 3 7

***Answer***

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

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

// Function prototypes
void createLinkedList(struct Node** head);
void displayLinkedList(struct Node* head);
void insertAtBeginning(struct Node** head, int new_data);
void insertAtEnd(struct Node** head, int new_data);
void insertBeforeValue(struct Node** head, int target, int new_data);
void insertAfterValue(struct Node** head, int target, int new_data);
void deleteFromBeginning(struct Node** head);
void deleteFromEnd(struct Node** head);
void deleteBeforeValue(struct Node** head, int target);
void deleteAfterValue(struct Node** head, int target);
void freeList(struct Node** head);

int main() {
   struct Node* head = NULL;
   int choice;

   while (1) {
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            createLinkedList(&head);
            printf("LINKED LIST CREATED\n");
            break;
         case 2:
```

```
        displayLinkedList(head);
        break;
    case 3: {
        int data;
        scanf("%d", &data);
        insertAtBeginning(&head, data);
        printf("The linked list after insertion at the beginning is:\n");
        displayLinkedList(head);
        break;
    }
    case 4: {
        int data;
        scanf("%d", &data);
        insertAtEnd(&head, data);
        printf("The linked list after insertion at the end is:\n");
        displayLinkedList(head);
        break;
    }
    case 5: {
        int target, new_data;
        scanf("%d %d", &target, &new_data);
        insertBeforeValue(&head, target, new_data);
        printf("The linked list after insertion before a value is:\n");
        displayLinkedList(head);
        break;
    }
    case 6: {
        int target, new_data;
        scanf("%d %d", &target, &new_data);
        insertAfterValue(&head, target, new_data);
        printf("The linked list after insertion after a value is:\n");
        displayLinkedList(head);
        break;
    }
    case 7:
        deleteFromBeginning(&head);
        printf("The linked list after deletion from the beginning is:\n");
        displayLinkedList(head);
        break;
    case 8:
        deleteFromEnd(&head);
        printf("The linked list after deletion from the end is:\n");
```

```
            displayLinkedList(head);
            break;
        case 9: {
            int target;
            scanf("%d", &target);
            deleteBeforeValue(&head, target);
            printf("The linked list after deletion before a value is:\n");
            displayLinkedList(head);
            break;
        }
        case 10: {
            int target;
            scanf("%d", &target);
            deleteAfterValue(&head, target);
            printf("The linked list after deletion after a value is:\n");
            displayLinkedList(head);
            break;
        }
        case 11:
            freeList(&head);
            return 0;
        default:
            printf("Invalid option! Please try again\n");
      }
   }
   return 0;
}

void createLinkedList(struct Node** head) {
   int data;
   while (1) {
      scanf("%d", &data);
      if (data == -1) break;
      insertAtEnd(head, data);
   }
}

void displayLinkedList(struct Node* head) {
   if (head == NULL) {
      printf("The list is empty\n");
      return;
   }
```

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

void insertAtBeginning(struct Node** head, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = *head;
    *head = new_node;
}

void insertAtEnd(struct Node** head, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = NULL;

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

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

void insertBeforeValue(struct Node** head, int target, int new_data) {
    if (*head == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    if ((*head)->data == target) {
        insertAtBeginning(head, new_data);
        return;
    }
```

```
    struct Node* prev = NULL;
    struct Node* curr = *head;

    while (curr != NULL && curr->data != target) {
        prev = curr;
        curr = curr->next;
    }

    if (curr == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = curr;
    prev->next = new_node;
}

void insertAfterValue(struct Node** head, int target, int new_data) {
    struct Node* curr = *head;

    while (curr != NULL) {
        if (curr->data == target) {
            struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
            new_node->data = new_data;
            new_node->next = curr->next;
            curr->next = new_node;
            return;
        }
        curr = curr->next;
    }
    printf("Value not found in the list\n");
}

void deleteFromBeginning(struct Node** head) {
    if (*head == NULL) {
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
```

```
    free(temp);
}

void deleteFromEnd(struct Node** head) {
    if (*head == NULL) {
        return;
    }
    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        return;
    }

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

void deleteBeforeValue(struct Node** head, int target) {
    if (*head == NULL || (*head)->data == target) {
        printf("Value not found in the list\n");
        return;
    }

    struct Node* prev = NULL;
    struct Node* curr = *head;

    while (curr != NULL && curr->data != target) {
        prev = curr;
        curr = curr->next;
    }

    if (curr == NULL || prev == *head) {
        printf("Value not found in the list\n");
        return;
    }

    struct Node* to_delete = *head;
    while (to_delete->next != prev) {
```

```
        to_delete = to_delete->next;
    }
    to_delete->next = curr;
    free(prev);
}

void deleteAfterValue(struct Node** head, int target) {
    struct Node* curr = *head;

    while (curr != NULL) {
        if (curr->data == target) {
            if (curr->next == NULL) {
                printf("No node exists after the target value\n");
                return;
            }
            struct Node* to_delete = curr->next;
            curr->next = to_delete->next;
            free(to_delete);
            return;
        }
        curr = curr->next;
    }
    printf("Value not found in the list\n");
}

void freeList(struct Node** head) {
    struct Node* current = *head;
    struct Node* next;

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

***Status :*** Partially correct **Marks : 0.5/1**

2. Problem Statement

John is working on evaluating polynomials for his math project. He needs to compute the value of a polynomial at a specific point using a singly linked list representation.

Help John by writing a program that takes a polynomial and a value of x as input, and then outputs the computed value of the polynomial.

Example

Input:

2

13

12

11

1

Output:

36

Explanation:

The degree of the polynomial is 2.

Calculate the value of x2: 13 * 12 = 13.

Calculate the value of x1: 12 * 11 = 12.

Calculate the value of x0: 11 * 10 = 11.

Add the values of x2, x1 and x0 together: 13 + 12 + 11 = 36.

***Input Format***

The first line of input consists of the degree of the polynomial.

The second line consists of the coefficient x2.

The third line consists of the coefficient of x1.

The fourth line consists of the coefficient x0.

The fifth line consists of the value of x, at which the polynomial should be evaluated.

***Output Format***

The output is the integer value obtained by evaluating the polynomial at the given value of x.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 2
13
12
11
1

Output: 36

***Answer***

```
// You are using GCC
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
struct Node {
    int coefficient;
    struct Node* next;
};
struct Node* createNode(int coefficient) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->coefficient = coefficient;
    newNode->next = NULL;
    return newNode;
}
void appendNode(struct Node** head, int coefficient) {
    struct Node* newNode = createNode(coefficient);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
```

```
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
int evaluatePolynomial(struct Node* head, int x) {
    int result = 0;
    int power = 0;
    struct Node* current = head;
    while (current != NULL) {
        result += current->coefficient * pow(x, power);
        power++;
        current = current->next;
    }
    return result;
}
int main() {
    int degree, x, coefficient;
    printf("");
    scanf("%d", &degree);
    struct Node* polynomial = NULL;
    printf("");
    for (int i = 0; i <= degree; i++) {
        scanf("%d", &coefficient);
        appendNode(&polynomial, coefficient);
    }
    printf("");
    scanf("%d", &x);
    int result = evaluatePolynomial(polynomial, x);
    printf(" %d\n", result);
    struct Node* temp;
    while (polynomial != NULL) {
        temp = polynomial;
        polynomial = polynomial->next;
        free(temp);
    }

    return 0;
}
```

***Status :*** Partially correct ***Marks : 0.9/1***

3. Problem Statement

Imagine you are managing the backend of an e-commerce platform. Customers place orders at different times, and the orders are stored in two separate linked lists. The first list holds the orders from morning, and the second list holds the orders from the evening.

Your task is to merge the two lists so that the final list holds all orders in sequence from the morning list followed by the evening orders, in the same order

***Input Format***

The first line contains an integer n , representing the number of orders in the morning list.

The second line contains n space-separated integers representing the morning orders.

The third line contains an integer m , representing the number of orders in the evening list.

The fourth line contains m space-separated integers representing the evening orders.

***Output Format***

The output should be a single line containing space-separated integers representing the merged order list, with morning orders followed by evening orders.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 3
101 102 103
2
104 105
Output: 101 102 103 104 105

*Answer*

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int order_id;
    struct Node* next;
};
struct Node* createNode(int order_id) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->order_id = order_id;
    newNode->next = NULL;
    return newNode;
}
void appendNode(struct Node** head, int order_id) {
    struct Node* newNode = createNode(order_id);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->order_id);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    int n, m, order_id;
    printf("");
    scanf("%d", &n);
    struct Node* morningList = NULL;
    printf("");
    for (int i = 0; i < n; i++) {
        scanf("%d", &order_id);
        appendNode(&morningList, order_id);
```

```
    }
    printf("");
    scanf("%d", &m);
    struct Node* eveningList = NULL;
    printf("");
    for (int i = 0; i < m; i++) {
        scanf("%d", &order_id);
        appendNode(&eveningList, order_id);
    }
    struct Node* mergedList = morningList;
    struct Node* temp = morningList;
    if (temp != NULL) {
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = eveningList;
    } else {
        mergedList = eveningList;
    }
    printf("");
    printList(mergedList);
    struct Node* current;
    while (mergedList != NULL) {
        current = mergedList;
        mergedList = mergedList->next;
        free(current);
    }

    return 0;
}
```

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

4. Problem Statement

Bharath is very good at numbers. As he is piled up with many works, he decides to develop programs for a few concepts to simplify his work. As a first step, he tries to arrange even and odd numbers using a linked list. He stores his values in a singly-linked list.

Now he has to write a program such that all the even numbers appear before the odd numbers. Finally, the list is printed in such a way that all even numbers come before odd numbers. Additionally, the even numbers should be in reverse order, while the odd numbers should maintain their original order.

Example

Input:

6

3 1 0 4 30 12

Output:

12 30 4 0 3 1

Explanation:

Even elements: 0 4 30 12

Reversed Even elements: 12 30 4 0

Odd elements: 3 1

So the final list becomes: 12 30 4 0 3 1

***Input Format***

The first line consists of an integer n representing the size of the linked list.

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

***Output Format***

The output prints the rearranged list separated by a space.

The list is printed in such a way that all even numbers come before odd numbers and the even numbers should be in reverse order, while the odd numbers should maintain their original order.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 6
3 1 0 4 30 12

Output: 12 30 4 0 3 1

***Answer***

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void appendNode(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
struct Node* reverseList(struct Node* head) {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
```

```
    }
    return prev;
}
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
void rearrangeEvenOdd(struct Node* head) {
    struct Node* evenList = NULL;
    struct Node* oddList = NULL;
    while (head != NULL) {
        if (head->data % 2 == 0) {
            appendNode(&evenList, head->data);
        } else {
            appendNode(&oddList, head->data);
        }
        head = head->next;
    }
    evenList = reverseList(evenList);
    struct Node* mergedList = evenList;
    struct Node* temp = mergedList;
    if (temp == NULL) {
        mergedList = oddList;
    } else {
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = oddList;
    }
    printList(mergedList);
}
int main() {
    int n, data;
    printf("");
    scanf("%d", &n);

    struct Node* list = NULL;
    printf("");
```

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

    printf("");
    rearrangeEvenOdd(list);

    return 0;
}
```

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

5. Problem Statement

Emily is developing a program to manage a singly linked list. The program should allow users to perform various operations on the linked list, such as inserting elements at the beginning or end, deleting elements from the beginning or end, inserting before or after a specific value, and deleting elements before or after a specific value. After each operation, the updated linked list should be displayed.

Your task is to help Emily in implementing the same.

***Input Format***

The first line contains an integer choice, representing the operation to perform:

- For choice 1 to create the linked list. The next lines contain space-separated integers, with -1 indicating the end of input.
- For choice 2 to display the linked list.
- For choice 3 to insert a node at the beginning. The next line contains an integer data representing the value to insert.
- For choice 4 to insert a node at the end. The next line contains an integer data representing the value to insert.
- For choice 5 to insert a node before a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 6 to insert a node after a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 7 to delete a node from the beginning.
- For choice 8 to delete a node from the end.

- For choice 9 to delete a node before a specific value. The next line contains an integer value representing the node before which deletion occurs.
- For choice 10 to delete a node after a specific value. The next line contains an integer value representing the node after which deletion occurs.
- For choice 11 to exit the program.

***Output Format***

For choice 1, print "LINKED LIST CREATED".

For choice 2, print the linked list as space-separated integers on a single line. If the list is empty, print "The list is empty".

For choice 3, 4, 5, and 6, print the updated linked list with a message indicating the insertion operation.

For choice 7, 8, 9, and 10, print the updated linked list with a message indicating the deletion operation.

For any operation that is not possible print an appropriate error message such as "Value not found in the list".

For choice 11 terminate the program.

For any invalid option, print "Invalid option! Please try again".

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 1
5
3
7
-1
2
11
Output: LINKED LIST CREATED
5 3 7

***Answer***

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
    printf("");
    displayList(*head);
}
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
```

```
    printf("");
    displayList(*head);
}
void insertBeforeValue(struct Node** head, int value, int data) {
    if (*head == NULL) {
        printf("");
        return;
    }
    if ((*head)->data == value) {
        insertAtBeginning(head, data);
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL && temp->next->data != value) {
        temp = temp->next;
    }
    if (temp->next == NULL) {
        printf("");
    } else {
        struct Node* newNode = createNode(data);
        newNode->next = temp->next;
        temp->next = newNode;
        printf("");
        displayList(*head);
    }
}
void insertAfterValue(struct Node** head, int value, int data) {
    struct Node* temp = *head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("");
    } else {
        struct Node* newNode = createNode(data);
        newNode->next = temp->next;
        temp->next = newNode;
        printf("");
        displayList(*head);
    }
}
void deleteFromBeginning(struct Node** head) {
```

```
    if (*head == NULL) {
        printf("");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    printf("");
    displayList(*head);
}
void deleteFromEnd(struct Node** head) {
    if (*head == NULL) {
        printf("");
        return;
    }
    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
    } else {
        struct Node* temp = *head;
        while (temp->next->next != NULL) {
            temp = temp->next;
        }
        free(temp->next);
        temp->next = NULL;
    }
    printf("");
    displayList(*head);
}
void deleteBeforeValue(struct Node** head, int value) {
    if (*head == NULL || (*head)->data == value) {
        printf("");
        return;
    }
    struct Node* temp = *head;
    if ((*head)->next != NULL && (*head)->next->data == value) {
        *head = (*head)->next;
        free(temp);
        printf("");
        displayList(*head);
        return;
    }
```

```
    while (temp->next != NULL && temp->next->next != NULL && temp->next->next->data != value) {
        temp = temp->next;
    }
    if (temp->next == NULL || temp->next->next == NULL) {
        printf("");
    } else {
        struct Node* toDelete = temp->next;
        temp->next = temp->next->next;
        free(toDelete);
        printf("");
        displayList(*head);
    }
}
void deleteAfterValue(struct Node** head, int value) {
    struct Node* temp = *head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {
        printf("");
    } else {
        struct Node* toDelete = temp->next;
        temp->next = temp->next->next;
        free(toDelete);
        printf("");
        displayList(*head);
    }
}
int main() {
    struct Node* head = NULL;
    int choice, data, value;

    while (1) {
        printf("");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("");
                while (1) {
                    scanf("%d", &data);
```

```
            if (data == -1) break;
            insertAtEnd(&head, data);
        }
        printf("");
        break;

    case 2:
        displayList(head);
        break;

    case 3:
        printf("");
        scanf("%d", &data);
        insertAtBeginning(&head, data);
        break;

    case 4:
        printf("");
        scanf("%d", &data);
        insertAtEnd(&head, data);
        break;

    case 5:
        printf("");
        scanf("%d %d", &value, &data);
        insertBeforeValue(&head, value, data);
        break;

    case 6:
        printf("");
        scanf("%d %d", &value, &data);
        insertAfterValue(&head, value, data);
        break;

    case 7:
        deleteFromBeginning(&head);
        break;

    case 8:
        deleteFromEnd(&head);
        break;
```

```
            case 9:
                printf("");
                scanf("%d", &value);
                deleteBeforeValue(&head, value);
                break;

            case 10:
                printf("");
                scanf("%d", &value);
                deleteAfterValue(&head, value);
                break;

            case 11:
                printf("");
                return 0;

            default:
                printf("");
                break;
        }
    }
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

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
   struct Node* newn=(struct Node*)malloc(sizeof(struct Node));
   newn->item=item;
   newn->next=NULL;
   newn->prev=NULL;
   struct Node* temp=*head;
   if(*head==NULL){
      *head=newn;
```

```
  }
  else{
     while(temp->next!=NULL){
        temp=temp->next;
     }
     temp->next=newn;
     newn->prev=temp;
  }
}
void displayForward(struct Node* head) {
   //type your code here
   struct Node* temp=head;
   while(temp){
      printf("%c ",temp->item);
      temp=temp->next;
   }
   printf("\n");
}

void displayBackward(struct Node* tail) {
   //type your code here
   struct Node* temp=tail;
   while(temp->next){
      temp=temp->next;
   }
   while(temp){
      printf("%c ",temp->item);
      temp=temp->prev;
   }
}

void freePlaylist(struct Node* head) {
   //type your code here
   struct Node* temp=head;
   free(temp);
}

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: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

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>
struct node{
    int data;
    struct node* next;
    struct node* prev;
};
void insert(struct node** head,struct node** tail,int data){
    struct node* newn=(struct node*)malloc(sizeof(struct node));
    newn->data=data;
    newn->next=NULL;
    if(*tail==NULL){
        newn->prev=NULL;
        *tail=*head=newn;
    }else{
        newn->prev=*tail;
        (*tail)->next=newn;
        *tail=newn;
    }
}
int find_max(struct node* head){
    if(head==NULL){
        printf("Empty list!");
        exit(0);
```

```
    }
    int max=head->data;
    struct node*temp=head->next;
    while(temp!=NULL){
        if(temp->data>max){
            max=temp->data;
        }
        temp=temp->next;
    }
    return max;
}
int main(){
    struct node *head=NULL,*tail=NULL;
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        int jai_guru;
        scanf("%d",&jai_guru);
        insert(&head,&tail,jai_guru);
    }
    int max=find_max(head);
    printf("%d",max);
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

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
struct node* tail=NULL;
void traverse() {
  //type your code here
  struct node* temp=tail;
  printf("Node Inserted\n");
  while(temp){
     printf("%d ",temp->info);
     temp=temp->next;
```

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

void insertAtFront(int data) {
  //type your code here
  struct node* newn=(struct node*)malloc(sizeof(struct node));
  newn->info=data;
  newn->next=NULL;
  newn->prev=NULL;
  if(tail==NULL){
     newn->next=tail;
     tail=newn;
  }
  else{
     newn->prev=NULL;
     newn->next=tail;
     tail->prev=newn;
     tail=newn;
  }
}
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: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

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>
typedef struct node{
   int data;
   struct node *prev,*next;
}node;
node* cnode(int data){
   node* newn=(node*)malloc(sizeof(node));
   newn->data=data;
   newn->prev=NULL;
   newn->next=NULL;
   return newn;
}
void insert(node** head,int val){
   node* newn=cnode(val);
   if(*head==NULL){
      *head=newn;
   }
   else{
      node* temp=*head;
      while(temp->next!=NULL){
         temp=temp->next;
      }
      temp->next=newn;
```

```
    }
}
void dis(node* head){
    node* temp=head;
    while(temp!=NULL){
        printf("%d ",temp->data);
        temp=temp->next;
    }
}
int main()
{
    node* head=NULL;
    int n,val;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d",&val);
        insert(&head,val);
    }
    dis(head);
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

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>
```

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

typedef struct DoublyLinkedList {
    Node* head;
} DoublyLinkedList;

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

// Function to insert a node at the end of the list
void insertEnd(DoublyLinkedList* list, int data) {
    Node* newNode = createNode(data);
    if (list->head == NULL) {
        list->head = newNode;
        return;
    }
    Node* last = list->head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
    newNode->prev = last;
}

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

```
}

// Function to delete a node at a specific position
void deleteNode(DoublyLinkedList* list, int position) {
    if (position < 1) {
        printf("Invalid position. Try again.\n");
        return;
    }

    Node* current = list->head;
    for (int i = 1; current != NULL && i < position; 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; // Change head if needed
    }

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

    free(current);
}

int main() {
    DoublyLinkedList list;
    list.head = NULL;

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

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

```
        insertEnd(&list, id);
    }

    printf("Data entered in the list:\n");
    displayList(&list);

    int p;
    scanf("%d", &p);

    deleteNode(&list, p);

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

    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

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

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***

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = 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;
}
int isPalindrome(Node* head) {
    if (!head) return 1;
    Node* left = head;
    Node* right = head;
    while (right->next) {
        right = right->next;
    }
    while (left && right) {
        if (left->data != right->data) {
            return 0;
        }
        left = left->next;
        right = right->prev;
        if (left == right || left->prev == right) {
            break;
        }
    }
    return 1;
}
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    int N, value;
    Node* head = NULL;
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        scanf("%d", &value);
        insertEnd(&head, value);
    }
```

```
    printList(head);
    if (isPalindrome(head)) {
        printf("The doubly linked list is a palindrome\n");
    } else {
        printf("The doubly linked list is not a palindrome\n");
    }
    return 0;
}
```

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

2. 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***

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}
void insertFront(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head != NULL) {
        newNode->next = *head;
        (*head)->prev = newNode;
    }
    *head = newNode;
}
void insertAtPosition(Node** head, int position, int data) {
    Node* newNode = createNode(data);
```

```
    if (position == 1) {
        insertFront(head, data);
        return;
    }
    Node* temp = *head;
    for (int i = 1; i < position - 1 && temp->next != NULL; i++) {
        temp = temp->next;
    }
    newNode->next = temp->next;
    newNode->prev = temp;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
}
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    int N, value, position, data;
    Node* head = NULL;
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        scanf("%d", &value);
        insertFront(&head, value);
    }
    printList(head);
    scanf("%d", &position);
    scanf("%d", &data);
    insertAtPosition(&head, position, data);
    printList(head);
    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***

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;
Node* createNode(int data) {
```

```
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = 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 rotateClockwise(Node** head, int k) {
    if (*head == NULL || k == 0) return;

    Node* temp = *head;
    int count = 1;
    while (count < k && temp->next != NULL) {
        temp = temp->next;
        count++;
    }
    if (temp->next == NULL) return;
    Node* newHead = temp->next;
    newHead->prev = NULL;
    Node* last = newHead;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = *head;
    (*head)->prev = last;
    temp->next = NULL;
    *head = newHead;
}
void printList(Node* head) {
```

```
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    int n, k, value;
    Node* head = NULL;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        insertEnd(&head, value);
    }
    scanf("%d", &k);
    rotateClockwise(&head, n - k);
    printList(head);
    return 0;
}
```

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

4. 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***

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = 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 printMiddle(Node* head, int n) {
    Node* temp = head;
    int midIndex = n / 2;
    for (int i = 0; i < midIndex; i++) {
        temp = temp->next;
    }
    if (n % 2 != 0) {
        printf("%d\n", temp->data);
    }
    else {
        printf("%d %d\n", temp->prev->data, temp->data);
    }
}
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    int n, value;
    Node* head = NULL;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        insertEnd(&head, value);
    }
    printList(head);
    printMiddle(head, n);
    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*

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}
void insertFront(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head != NULL) {
        newNode->next = *head;
        (*head)->prev = newNode;
    }
    *head = newNode;
}
void deleteAtPosition(Node** head, int position) {
    if (*head == NULL || position < 1) return;
    Node* temp = *head;
    for (int i = 1; i < position && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) return;
    if (temp->prev == NULL) {
        *head = temp->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
    } else {
        temp->prev->next = temp->next;
        if (temp->next != NULL) {
            temp->next->prev = temp->prev;
        }
    }
```

```
    free(temp);
}
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    int N, value, X;
    Node* head = NULL;
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        scanf("%d", &value);
        insertFront(&head, value);
    }
    printList(head);
    scanf("%d", &X);
    deleteAtPosition(&head, X);
    printList(head);
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 3_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

In a coding competition, you are assigned a task to create a program that simulates a stack using a linked list.

The program should feature a menu-driven interface for pushing an integer to stack, popping, and displaying stack elements, with robust error handling for stack underflow situations. This challenge tests your data structure skills.

***Input Format***

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the integer value onto the stack. If the choice is 1, the following input is a space-separated integer, representing the element to be pushed onto

the stack.

Choice 2: Pop the integer from the stack.

Choice 3: Display the elements in the stack.

Choice 4: Exit the program.

***Output Format***

The output displays messages according to the choice and the status of the stack:

If the choice is 1, push the given integer to the stack and display the following: "Pushed element: " followed by the value pushed.

If the choice is 2, pop the integer from the stack and display the following: "Popped element: " followed by the value popped.

If the choice is 2, and if the stack is empty without any elements, print "Stack is empty. Cannot pop."

If the choice is 3, print the elements in the stack: "Stack elements (top to bottom): " followed by the space-separated values.

If the choice is 3, and there are no elements in the stack, print "Stack is empty".

If the choice is 4, exit the program and display the following: "Exiting program".

If any other choice is entered, print "Invalid choice".

Refer to the sample input and output for the exact format.

***Sample Test Case***

Input: 1 3
1 4
3
2
3
4

Output: Pushed element: 3
Pushed element: 4
Stack elements (top to bottom): 4 3
Popped element: 4
Stack elements (top to bottom): 3
Exiting program

***Answer***

-

***Status :*** Skipped ***Marks : 0/10***

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

## REC_DS using C_Week 4_MCQ_Updated

Attempt : 1
Total Mark : 20
Marks Obtained : 16

### Section 1 : MCQ

1. Which of the following properties is associated with a queue?

***Answer***

First In First Out

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

2. Front and rear pointers are tracked in the linked list implementation of a queue. Which of these pointers will change during an insertion into the EMPTY queue?

***Answer***

Only rear pointer

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

3. A normal queue, if implemented using an array of size MAX_SIZE, gets full when

***Answer***

Rear = MAX_SIZE – 1

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

4. In a linked list implementation of a queue, front and rear pointers are tracked. Which of these pointers will change during an insertion into a non-empty queue?

***Answer***

Only rear pointer

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

5. The process of accessing data stored in a serial access memory is similar to manipulating data on a

***Answer***

Queue

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

6. What will the output of the following code?

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
    int* arr;
    int front;
    int rear;
    int size;
} Queue;
Queue* createQueue() {
    Queue* queue = (Queue*)malloc(sizeof(Queue));
```

```
    queue->arr = (int*)malloc(5 * sizeof(int));
    queue->front = 0;
    queue->rear = -1;
    queue->size = 0;
    return queue;
}
int main() {
    Queue* queue = createQueue();
    printf("%d", queue->size);
    return 0;
}
```

***Answer***

0

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

7. What is the functionality of the following piece of code?

```
public void function(Object item)
{
    Node temp=new Node(item,trail);
    if(isEmpty())
    {
        head.setNext(temp);
        temp.setNext(trail);
    }
    else
    {
        Node cur=head.getNext();
        while(cur.getNext()!=trail)
        {
            cur=cur.getNext();
        }
        cur.setNext(temp);
    }
    size++;
}
```

***Answer***

Insert at the rear end of the dequeue

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

8. After performing this set of operations, what does the final list look to contain?

InsertFront(10);
InsertFront(20);
InsertRear(30);
DeleteFront();
InsertRear(40);
InsertRear(10);
DeleteRear();
InsertRear(15);
display();

***Answer***

10 30 40 15

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

9. When new data has to be inserted into a stack or queue, but there is no available space. This is known as

***Answer***

overflow

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

10. What does the front pointer in a linked list implementation of a queue contain?

***Answer***

The address of the first element

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

11. In linked list implementation of a queue, the important condition for a queue to be empty is?

***Answer***

FRONT is null

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

12. Which of the following can be used to delete an element from the front end of the queue?

***Answer***

public Object deleteFront() throws emptyDEQException{if(isEmpty())throw new emptyDEQException("Empty");else{Node temp = head.getNext();Node cur = temp.getNext();Object e = temp.getEle();head.setNext(temp);size--;return e;}}

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

13. What are the applications of dequeue?

***Answer***

Can be used as both stack and queue

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

14. The essential condition that is checked before insertion in a queue is?

***Answer***

Overflow

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

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

```
#include <stdio.h>
#define MAX_SIZE 5
typedef struct {
```

```
    int arr[MAX_SIZE];
    int front;
    int rear;
    int size;
} Queue;

void enqueue(Queue* queue, int data) {
    if (queue->size == MAX_SIZE) {
        return;
    }
    queue->rear = (queue->rear + 1) % MAX_SIZE;
    queue->arr[queue->rear] = data;
    queue->size++;
}
int dequeue(Queue* queue) {
    if (queue->size == 0) {
        return -1;
    }
    int data = queue->arr[queue->front];
    queue->front = (queue->front + 1) % MAX_SIZE;
    queue->size--;
    return data;
}
int main() {
    Queue queue;
    queue.front = 0;
    queue.rear = -1;
    queue.size = 0;
    enqueue(&queue, 1);
    enqueue(&queue, 2);
    enqueue(&queue, 3);
    printf("%d ", dequeue(&queue));
    printf("%d ", dequeue(&queue));
    enqueue(&queue, 4);
    enqueue(&queue, 5);
    printf("%d ", dequeue(&queue));
    printf("%d ", dequeue(&queue));
    return 0;
}
```

***Answer***

3 2 1 4

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

16. Insertion and deletion operation in the queue is known as

***Answer***

Enqueue and Dequeue

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

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

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 5
typedef struct {
    int* arr;
    int front;
    int rear;
    int size;
} Queue;
Queue* createQueue() {
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->arr = (int*)malloc(MAX_SIZE * sizeof(int));
    queue->front = -1;
    queue->rear = -1;
    queue->size = 0;
    return queue;
}
int isEmpty(Queue* queue) {
    return (queue->size == 0);
}
int main() {
    Queue* queue = createQueue();
    printf("Is the queue empty? %d", isEmpty(queue));
    return 0;
```

}

***Answer***

Is the queue empty? 1

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

18. Which one of the following is an application of Queue Data Structure?

***Answer***

All of the mentioned options

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

19. In what order will they be removed If the elements “A”, “B”, “C” and “D” are placed in a queue and are deleted one at a time

***Answer***

ABCD

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

20. Which operations are performed when deleting an element from an array-based queue?

***Answer***

Dequeue

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 4_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

Imagine a bustling coffee shop, where customers are placing their orders for their favorite coffee drinks. The cafe owner Sheeren wants to efficiently manage the queue of coffee orders using a digital system. She needs a program to handle this queue of orders.

You are tasked with creating a program that implements a queue for coffee orders. Each character in the queue represents a customer's coffee order, with 'L' indicating a latte, 'E' indicating an espresso, 'M' indicating a macchiato, 'O' indicating an iced coffee, and 'N' indicating a nabob.

Customers can place orders and enjoy their delicious coffee drinks.

***Input Format***

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Enqueue the coffee order into the queue. If the choice is 1, the following input is a space-separated character ('L', 'E', 'M', 'O', 'N').

Choice 2: Dequeue a coffee order from the queue.

Choice 3: Display the orders in the queue.

Choice 4: Exit the program.

***Output Format***

The output displays messages according to the choice and the status of the queue:

If the choice is 1:

1. Insert the given order into the queue and display "Order for [order] is enqueued." where [order] is the coffee order that is inserted.
2. If the queue is full, print "Queue is full. Cannot enqueue more orders."

If the choice is 2:

1. Dequeue a character from the queue and display "Dequeued Order: " followed by the corresponding order that is dequeued.
2. If the queue is empty without any orders, print "No orders in the queue."

If the choice is 3:

1. The output prints "Orders in the queue are: " followed by the space-separated orders present in the queue.
2. If there are no orders in the queue, print "Queue is empty. No orders available."

If the choice is 4:

1. Exit the program and print "Exiting program"

If any other choice is entered, the output prints "Invalid option."

Refer to the sample output for the exact text and format.

***Sample Test Case***

Input: 1 L
1 E
1 M
1 O
1 N
1 O
3
2
3
4

Output: Order for L is enqueued.
Order for E is enqueued.
Order for M is enqueued.
Order for O is enqueued.
Order for N is enqueued.
Queue is full. Cannot enqueue more orders.
Orders in the queue are: L E M O N
Dequeued Order: L
Orders in the queue are: E M O N
Exiting program

***Answer***

```
#include <stdio.h>
#define MAX_SIZE 5

char orders[MAX_SIZE];
int front = -1;
int rear = -1;

void initializeQueue() {
   front = -1;
   rear = -1;
}

#include <stdio.h>
```

```
#include <stdlib.h>

#define MAX_SIZE 5

struct Queue {
    char orders[MAX_SIZE];
    int front, rear;
};

void initQueue(struct Queue* q) {
    q->front = -1;
    q->rear = -1;
}

int isFull(struct Queue* q) {
    return q->rear == MAX_SIZE - 1;
}

int isEmpty(struct Queue* q) {
    return q->front == -1;
}

void enqueue(struct Queue* q, char order) {
    if (isFull(q)) {
        printf("Queue is full. Cannot enqueue more orders.\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear++;
    q->orders[q->rear] = order;
    printf("Order for %c is enqueued.\n", order);
}

void dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("No orders in the queue.\n");
        return;
    }
    printf("Dequeued Order: %c\n", q->orders[q->front]);
    for (int i = 0; i < q->rear; i++) {
```

```
        q->orders[i] = q->orders[i + 1];
    }
    q->rear--;
    if (q->rear == -1) {
        q->front = -1;
    }
}

void display(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty. No orders available.\n");
        return;
    }
    printf("Orders in the queue are: ");
    for (int i = q->front; i <= q->rear; i++) {
        printf("%c ", q->orders[i]);
    }
    printf("\n");
}

int main() {
    struct Queue q;
    initQueue(&q);

    int choice;
    char order;

    while (1) {
        printf("Enter your choice:\n");
        printf("1. Enqueue Order\n");
        printf("2. Dequeue Order\n");
        printf("3. Display Orders\n");
        printf("4. Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter coffee order (L/E/M/O/N): ");
                scanf(" %c", &order);
                enqueue(&q, order);
                break;
            case 2:
```

```
            dequeue(&q);
            break;
        case 3:
            display(&q);
            break;
        case 4:
            printf("Exiting program\n");
            exit(0);
        default:
            printf("Invalid option.\n");
      }
   }

   return 0;
}
int main() {
   char order;
   int option;
   initializeQueue();
   while (1) {
      if (scanf("%d", &option) != 1) {
         break;
      }
      switch (option) {
         case 1:
            if (scanf(" %c", &order) != 1) {
               break;
            }
            if (enqueue(order)) {
            }
            break;
         case 2:
            dequeue();
            break;
         case 3:
            display();
            break;
         case 4:
            printf("Exiting program");
            return 0;
         default:
            printf("Invalid option.\n");
```

```
            break;
        }
    }
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 4_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

In a bustling IT department, staff regularly submit helpdesk tickets to request technical assistance. Managing these tickets efficiently is vital for providing quality support.

Your task is to develop a program that uses an array-based queue to handle and prioritize helpdesk tickets based on their unique IDs.

Implement a program that provides the following functionalities:

Enqueue Helpdesk Ticket: Add a new helpdesk ticket to the end of the queue. Provide a positive integer representing the ticket ID for the new ticket.Dequeue Helpdesk Ticket: Remove and process the next helpdesk ticket from the front of the queue. The program will display the ticket ID of the processed ticket.Display Queue: Display the ticket IDs of all the

helpdesk tickets currently in the queue.

***Input Format***

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Enqueue the ticket ID into the queue. If the choice is 1, the following input is a space-separated integer, representing the ticket ID to be enqueued into the queue.

Choice 2: Dequeue a ticket from the queue.

Choice 3: Display the ticket IDs in the queue.

Choice 4: Exit the program.

***Output Format***

The output displays messages according to the choice and the status of the queue:

If the choice is 1:

1. Insert the given ticket ID into the queue and display "Helpdesk Ticket ID [id] is enqueued." where [id] is the ticket ID that is inserted.
2. If the queue is full, print "Queue is full. Cannot enqueue."

If the choice is 2:

1. Dequeue a ticket ID from the queue and display "Dequeued Helpdesk Ticket ID: " followed by the corresponding ID that is dequeued.
2. If the queue is empty without any elements, print "Queue is empty."

If the choice is 3:

1. The output prints "Helpdesk Ticket IDs in the queue are: " followed by the space-separated ticket IDs present in the queue.
2. If there are no elements in the queue, print "Queue is empty."

If the choice is 4:

1. Exit the program and print "Exiting the program"

If any other choice is entered, print "Invalid option."

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 1 101
1 202
1 203
1 204
1 205
1 206
3
2
3
4

Output: Helpdesk Ticket ID 101 is enqueued.
Helpdesk Ticket ID 202 is enqueued.
Helpdesk Ticket ID 203 is enqueued.
Helpdesk Ticket ID 204 is enqueued.
Helpdesk Ticket ID 205 is enqueued.
Queue is full. Cannot enqueue.
Helpdesk Ticket IDs in the queue are: 101 202 203 204 205
Dequeued Helpdesk Ticket ID: 101
Helpdesk Ticket IDs in the queue are: 202 203 204 205
Exiting the program

***Answer***

```
#include <stdio.h>
#define MAX_SIZE 5

int ticketIDs[MAX_SIZE];
int front = -1;
int rear = -1;
int lastDequeued;

void initializeQueue() {
   front = -1;
   rear = -1;
}
```

```
// Function to check if the queue is empty
int isEmpty() {
  return front == -1;
}

// Function to check if the queue is full
int isFull() {
  return (rear + 1) % MAX_SIZE == front;
}

// Function to enqueue a ticket ID into the queue
void enqueue(int ticketID) {
  if (isFull()) {
    printf("Queue is full. Cannot enqueue.\n");
    return;
  }
  if (isEmpty()) {
    front = 0;
  }
  rear = (rear + 1) % MAX_SIZE;
  ticketIDs[rear] = ticketID;
  printf("Helpdesk Ticket ID %d is enqueued.\n", ticketID);
}

// Function to dequeue a ticket ID from the queue
int dequeue() {
  if (isEmpty()) {
    return 0; // Queue is empty
  }
  lastDequeued = ticketIDs[front];
  if (front == rear) { // Single element in the queue
    front = -1;
    rear = -1;
  } else {
    front = (front + 1) % MAX_SIZE;
  }
  return 1; // Successfully dequeued
}

// Function to display the ticket IDs in the queue
void display() {
  if (isEmpty()) {
```

```
    printf("Queue is empty.\n");
    return;
  }
  printf("Helpdesk Ticket IDs in the queue are: ");
  int i = front;
  while (1) {
    printf("%d ", ticketIDs[i]);
    if (i == rear) {
      break;
    }
    i = (i + 1) % MAX_SIZE;
  }
  printf("\n");
}

int main() {
    int ticketID;
    int option;
    initializeQueue();
    while (1) {
        if (scanf("%d", &option) == EOF) {
            break;
        }
        switch (option) {
            case 1:
                if (scanf("%d", &ticketID) == EOF) {
                    break;
                }
                enqueue(ticketID);
                break;
            case 2:
                if (dequeue()) {
                    printf("Dequeued Helpdesk Ticket ID: %d\n", lastDequeued);
                } else {
                    printf("Queue is empty.\n");
                }
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting the program\n");
                return 0;
```

```
            default:
                printf("Invalid option.\n");
                break;
        }
    }
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 4_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

Write a program to implement a queue using an array and pointers. The program should provide the following functionalities:

Insert an element into the queue.Delete an element from the queue.Display the elements in the queue.

The queue has a maximum capacity of 5 elements. If the queue is full and an insertion is attempted, a "Queue is full" message should be displayed. If the queue is empty and a deletion is attempted, a "Queue is empty" message should be displayed.

***Input Format***

Each line contains an integer representing the chosen option from 1 to 3.

Option 1: Insert an element into the queue followed by an integer representing the element to be inserted, separated by a space.

Option 2: Delete an element from the queue.

Option 3: Display the elements in the queue.

***Output Format***

For option 1 (insertion):-

1. The program outputs: "<data> is inserted in the queue." if the data is successfully inserted.
2. "Queue is full." if the queue is already full and cannot accept more elements.

For option 2 (deletion):-

1. The program outputs: "Deleted number is: <data>" if an element is successfully deleted and returns the value of the deleted element.
2. "Queue is empty." if the queue is empty no elements can be deleted.

For option 3 (display):-

1. The program outputs: "Elements in the queue are: <element1> <element2> ... <elementN>" where <element1>, <element2>, ..., <elementN> represent the elements present in the queue.
2. "Queue is empty." if the queue is empty no elements can be displayed.

For invalid options, the program outputs: "Invalid option."

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 1 10

3
5
Output: 10 is inserted in the queue.
Elements in the queue are: 10
Invalid option.

***Answer***

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

#define max 5

int queue[max];
int front = -1, rear = -1;
```

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

#define MAX_SIZE 5

struct Queue {
    int data[MAX_SIZE];
    int front, rear;
};

void initQueue(struct Queue* q) {
    q->front = -1;
    q->rear = -1;
}

int isFull(struct Queue* q) {
    return q->rear == MAX_SIZE - 1;
}

int isEmpty(struct Queue* q) {
    return q->front == -1;
}

void enqueue(struct Queue* q, int value) {
    if (isFull(q)) {
        printf("Queue is full.\n");
        return;
```

```
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear++;
    q->data[q->rear] = value;
    printf("%d is inserted in the queue.\n", value);
}

void dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Deleted number is: %d\n", q->data[q->front]);
    for (int i = 0; i < q->rear; i++) {
        q->data[i] = q->data[i + 1];
    }
    q->rear--;
    if (q->rear == -1) {
        q->front = -1;
    }
}

void display(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Elements in the queue are: ");
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d ", q->data[i]);
    }
    printf("\n");
}

int main() {
    struct Queue q;
    initQueue(&q);

    int choice, value;
    while (1) {
```

```
        printf("\n1 Insert an element into the queue\n");
        printf("2 Delete an element from the queue\n");
        printf("3 Display the elements in the queue\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to insert: ");
                scanf("%d", &value);
                enqueue(&q, value);
                break;
            case 2:
                dequeue(&q);
                break;
            case 3:
                display(&q);
                break;
            default:
                printf("Invalid option.\n");
        }
    }

    return 0;
}

int main()
{
    int data, reply, option;
    while (1)
    {
        if (scanf("%d", &option) != 1)
            break;
        switch (option)
        {
            case 1:
                if (scanf("%d", &data) != 1)
                    break;
                reply = insertq(&data);
                if (reply == 0)
                    printf("Queue is full.\n");
                else
                    printf("%d is inserted in the queue.\n", data);
```

```
            break;
        case 2:
            delq();  //    Called without arguments
            break;
        case 3:
            display();
            break;
        default:
            printf("Invalid option.\n");
            break;
      }
   }
   return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 4_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

In an office setting, a print job management system is used to efficiently handle and process print jobs. The system is implemented using a queue data structure with an array.

The program provides the following operations:

Enqueue Print Job: Add a print job with a specified number of pages to the end of the queue.Dequeue Print Job: Remove and process the next print job in the queue.Display Queue: Display the print jobs in the queue

The program should ensure that print jobs are processed in the order they are received.

***Input Format***

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Enqueue the print job into the queue. If the choice is 1, the following input is a space-separated integer, representing the pages to be enqueued into the queue.

Choice 2: Dequeue a print job from the queue.

Choice 3: Display the print jobs in the queue.

Choice 4: Exit the program.

***Output Format***

The output displays messages according to the choice and the status of the queue:

If the choice is 1:

1. Insert the given page into the queue and display "Print job with [page] pages is enqueued." where [page] is the number of pages that are inserted.
2. If the queue is full, print "Queue is full. Cannot enqueue."

If the choice is 2:

1. Dequeue a page from the queue and display "Processing print job: [page] pages" where [page] is the corresponding page that is dequeued.
2. If the queue is empty without any elements, print "Queue is empty."

If the choice is 3:

1. The output prints "Print jobs in the queue: " followed by the space-separated pages present in the queue.
2. If there are no elements in the queue, print "Queue is empty."

If the choice is 4:

1. Exit the program and print "Exiting program"

If any other choice is entered, the output prints "Invalid option."

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 1
10
1
20
1
30
1
40
1
50
1
60
3
2
3
4
Output: Print job with 10 pages is enqueued.
Print job with 20 pages is enqueued.
Print job with 30 pages is enqueued.
Print job with 40 pages is enqueued.
Print job with 50 pages is enqueued.
Queue is full. Cannot enqueue.
Print jobs in the queue: 10 20 30 40 50
Processing print job: 10 pages
Print jobs in the queue: 20 30 40 50
Exiting program

***Answer***

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

#define MAX_SIZE 5
```

```
struct Queue {
    int jobs[MAX_SIZE];
    int front, rear;
};

void initializeQueue(struct Queue *q) {
    q->front = -1;
    q->rear = -1;
}

int isFull(struct Queue *q) {
    return q->rear == MAX_SIZE - 1;
}

int isEmpty(struct Queue *q) {
    return q->front == -1;
}

void enqueue(struct Queue *q, int pages) {
    if (isFull(q)) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear++;
    q->jobs[q->rear] = pages;
    printf("Print job with %d pages is enqueued.\n", pages);
}

void dequeue(struct Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Processing print job: %d pages\n", q->jobs[q->front]);
    for (int i = 0; i < q->rear; i++) {
        q->jobs[i] = q->jobs[i + 1];
    }
    q->rear--;
    if (q->rear == -1) {
```

```
        q->front = -1;
    }
}

void display(struct Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Print jobs in the queue: ");
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d ", q->jobs[i]);
    }
    printf("\n");
}

int main() {
    struct Queue q;
    initializeQueue(&q);

    int choice, pages;
    while (1) {
        printf("\n1. Enqueue Print Job\n");
        printf("2. Dequeue Print Job\n");
        printf("3. Display Queue\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter number of pages: ");
                scanf("%d", &pages);
                enqueue(&q, pages);
                break;
            case 2:
                dequeue(&q);
                break;
            case 3:
                display(&q);
                break;
            case 4:
```

```
            printf("Exiting program\n");
            exit(0);
        default:
            printf("Invalid option.\n");
    }
  }

  return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 4_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

You are tasked with implementing basic operations on a queue data structure using a linked list.

You need to write a program that performs the following operations on a queue:

Enqueue Operation: Implement a function that inserts an integer element at the rear end of the queue.Print Front and Rear: Implement a function that prints the front and rear elements of the queue. Dequeue Operation: Implement a function that removes the front element from the queue.

***Input Format***

The first line of input consists of an integer N, representing the number of elements to be inserted into the queue.

The second line consists of N space-separated integers, representing the queue elements.

***Output Format***

The first line prints "Front: X, Rear: Y" where X is the front and Y is the rear elements of the queue.

The second line prints the message indicating that the dequeue operation (front element removed) is performed: "Performing Dequeue Operation:".

The last line prints "Front: M, Rear: N" where M is the front and N is the rear elements after the dequeue operation.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 5
12 56 87 23 45

Output: Front: 12, Rear: 45
Performing Dequeue Operation:
Front: 56, Rear: 45

***Answer***

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

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

struct Node* front = NULL;
struct Node* rear = NULL;
```

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

// Node structure for the linked list
```

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

// Queue structure
struct Queue {
    struct Node* front;
    struct Node* rear;
};

// Function to create a new queue
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = queue->rear = NULL;
    return queue;
}

// Function to enqueue an element
void enqueue(struct Queue* queue, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    // If the queue is empty, then the new node is both front and rear
    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
        return;
    }

    // Add the new node at the end of the queue and update rear
    queue->rear->next = newNode;
    queue->rear = newNode;
}

// Function to print the front and rear elements of the queue
void printFrontAndRear(struct Queue* queue) {
    if (queue->front != NULL) {
        printf("Front: %d, Rear: %d\n", queue->front->data, queue->rear->data);
    } else {
        printf("Queue is empty.\n");
    }
```

```
}

// Function to dequeue an element from the queue
void dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("Queue is empty, nothing to dequeue.\n");
        return;
    }

    struct Node* temp = queue->front;
    queue->front = queue->front->next;

    // If the front becomes NULL, then also make rear NULL
    if (queue->front == NULL) {
        queue->rear = NULL;
    }

    free(temp);
}

// Main function
int main() {
    int N;
    scanf("%d", &N);

    struct Queue* queue = createQueue();
    int value;

    // Enqueue elements
    for (int i = 0; i < N; i++) {
        scanf("%d", &value);
        enqueue(queue, value);
    }

    // Print front and rear
    printFrontAndRear(queue);

    // Perform dequeue operation
    printf("Performing Dequeue Operation:\n");
    dequeue(queue);

    // Print front and rear after dequeue
```

```
    printFrontAndRear(queue);

    // Free the queue structure
    free(queue);
    return 0;
}
int main() {
    int n, data;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        enqueue(data);
    }
    printFrontRear();
    printf("Performing Dequeue Operation:\n");
    dequeue();
    printFrontRear();
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

## REC_DS using C_Week 4_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 50

## Section 1 : Coding

1. Problem Statement

Guide Harish in developing a simple queue system for a customer service center. The customer service center can handle up to 25 customers at a time. The queue needs to support basic operations such as adding a customer to the queue, serving a customer (removing them from the queue), and displaying the current queue of customers.

Use an array for implementation.

***Input Format***

The first line of the input consists of an integer N, the number of customers arriving at the service center.

The second line consists of N space-separated integers, representing the customer IDs in the order they arrive.

***Output Format***

After serving the first customer in the queue, display the remaining customers in the queue.

If a dequeue operation is attempted on an empty queue, display "Underflow".

If the queue is empty, display "Queue is empty".

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
101 102 103 104 105
Output: 102 103 104 105

***Answer***

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 25
int queue[MAX_SIZE];
int front = -1;
int rear =-1;
void enqueue(int customer_id) {
    if(rear == MAX_SIZE - 1) {
        return;
    }
    if(front == -1) {
        front =0;
    }
    rear++;
    queue[rear]= customer_id;
}
int dequeue() {
    if(front == -1 || front > rear) {
        return -1;
    }
    int served_customer = queue[front];
```

```
    front++;
    return served_customer;
}
void display_queue() {
    if(front == -1 || front > rear) {
        printf("Queue is empty\n");
        return;
    }
    for(int i=front;i<=rear;i++) {
        printf("%d" , queue[i]);
    }
    printf("\n");
}
int main() {
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++) {
        int customer_id;
        scanf("%d",&customer_id);
        enqueue(customer_id);
    }
    int served = dequeue();
    if(served == -1) {
        printf("Underflow\n");
    }
    display_queue();
    return 0;
}
```

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

2. Problem Statement

Sharon is developing a queue using an array. She wants to provide the functionality to find the Kth largest element. The queue should support the addition and retrieval of the Kth largest element effectively. The maximum capacity of the queue is 10.

Assist her in the program.

***Input Format***

The first line of input consists of an integer N, representing the number of elements in the queue.

The second line consists of N space-separated integers.

The third line consists of an integer K.

***Output Format***

For each enqueued element, print a message: "Enqueued: " followed by the element.

The last line prints "The [K]th largest element: " followed by the Kth largest element.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
23 45 93 87 25
4

Output: Enqueued: 23
Enqueued: 45
Enqueued: 93
Enqueued: 87
Enqueued: 25
The 4th largest element: 25

***Answer***

```
// You are using GCC
#include<stdio.h>
#include<stdio.h>
#define MAX_SIZE 10
int queue[MAX_SIZE];
int front = -1;
int rear = -1;
void enqueue(int element) {
   if(rear == MAX_SIZE - 1) {
```

```
        return;
    }
    if(front == -1) {
        front = 0;
    }
    rear++;
    queue[rear] = element;
    printf("Enqueued: %d\n", element);
}
int findKthLargest(int k) {
    if(front == -1 || front > rear) {
        return -1;
    }
    int current_size = rear-front + 1;
    if(k>current_size) {
        return -1;
    }
    int temp_arr[current_size];
    int index=0;
    for(int i=front; i<=rear;i++) {
        temp_arr[index++] = queue[i];
    }
    for(int i=0;i<current_size-1;i++) {
        for(int j=0;j<current_size - i - 1;j++) {
            if(temp_arr[j] < temp_arr[j + 1]) {
                int temp = temp_arr[j];
                temp_arr[j] = temp_arr[j + 1];
                temp_arr[j + 1] = temp;
            }
        }
    }
    return temp_arr[k - 1];
}
int main() {
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++) {
        int element;
        scanf("%d", &element);
        enqueue(element);
    }
    int k;
```

```
    scanf("%d", &k);
    int kth_largest = findKthLargest(k);
    if(kth_largest != -1) {
        printf("The %dth largest element: %d\n",k,kth_largest);
    } else {
    }
    return 0;
}
```

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

3. Problem Statement

You are tasked with developing a simple ticket management system for a customer support department. In this system, customers submit support tickets, which are processed in a First-In-First-Out (FIFO) order. The system needs to handle the following operations:

Ticket Submission (Enqueue Operation): New tickets are submitted by customers. Each ticket is assigned a unique identifier (represented by an integer). When a new ticket arrives, it should be added to the end of the queue.

Ticket Processing (Dequeue Operation): The support team processes tickets in the order they are received. The ticket at the front of the queue is processed first. After processing, the ticket is removed from the queue.

Display Ticket Queue: The system should be able to display the current state of the ticket queue, showing the sequence of ticket identifiers from front to rear.

***Input Format***

The first input line contains an integer n, the number of tickets submitted by customers.

The second line consists of a single integer, representing the unique identifier of each submitted ticket, separated by a space.

***Output Format***

The first line displays the "Queue: " followed by the ticket identifiers in the queue after all tickets have been submitted.

The second line displays the "Queue After Dequeue: " followed by the ticket identifiers in the queue after processing (removing) the ticket at the front.

Refer to the sample output for the exact text and format.

***Sample Test Case***

Input: 6
14 52 63 95 68 49

Output: Queue: 14 52 63 95 68 49
Queue After Dequeue: 52 63 95 68 49

***Answer***

```
// You are using GCC
#include<stdio.h>
#include<stdlib.h>
#define MAX_SIZE 20
int queue[MAX_SIZE];
int front = -1;
int rear = -1;
void enqueue(int element) {
   if(rear == MAX_SIZE -1) {
      return;
   }
   if (front == -1) {
      front =0;
   }
   rear++;
   queue[rear]=element;
}
int dequeue() {
   if(front == -1 || front > rear) {
      return -1;
   }
   int dequeued_element =queue[front];
   front++;
   if (front >rear) {
```

```
        front = rear=-1;
    }
    return dequeued_element;
}
void displayQueue() {
    if(front == -1 || front >rear) {
        return;
    }
    for(int i=front;i<=rear;i++) {
        printf("%d",queue[i]);
    }
}
int main() {
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++) {
        int ticket_id;
        scanf("%d",&ticket_id);
        enqueue(ticket_id);
    }
    printf("Queue: ");
    displayQueue();
    printf("\n");
    dequeue();
    printf("Queue After Dequeue: ");
    displayQueue();
    printf("\n");
    return 0;
}
```

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

4. Problem Statement

Amar is working on a project where he needs to implement a special type of queue that allows selective dequeuing based on a given multiple. He wants to efficiently manage a queue of integers such that only elements not divisible by a given multiple are retained in the queue after a selective dequeue operation.

Implement a program to assist Amar in managing his selective queue.

Example

Input:

5

10 2 30 4 50

5

Output:

Original Queue: 10 2 30 4 50

Queue after selective dequeue: 2 4

Explanation:

After selective dequeue with a multiple of 5, the elements that are multiples of 5 should be removed. Therefore, only 10, 30, and 50 should be removed from the queue. The updated Queue is 2 4.

***Input Format***

The first line contains an integer n, representing the number of elements initially present in the queue.

The second line contains n space-separated integers, representing the elements of the queue.

The third line contains an integer multiple, representing the divisor for selective dequeue operation.

***Output Format***

The first line of output prints "Original Queue: " followed by the space-separated elements in the queue before the dequeue operation.

The second line prints "Queue after selective dequeue: " followed by the remaining space-separated elements in the queue, after deleting elements that are the multiples of the specified number.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 5
10 2 30 4 50
5
Output: Original Queue: 10 2 30 4 50
Queue after selective dequeue: 2 4

***Answer***

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

int main() {
    int n, multiple;

    // Input handling
    scanf("%d", &n);
    int queue[n];

    for (int i = 0; i < n; i++) {
        scanf("%d", &queue[i]);
    }
    scanf("%d", &multiple);

    // Print original queue
    printf("Original Queue: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");

    // Selective dequeue operation
    printf("Queue after selective dequeue: ");
    for (int i = 0; i < n; i++) {
        if (queue[i] % multiple != 0) {
            printf("%d ", queue[i]);
        }
    }
    printf("\n");
```

```
    return 0;
}
```

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

5. Problem Statement

You've been assigned the challenge of developing a queue data structure using a linked list.

The program should allow users to interact with the queue by enqueuing positive integers and subsequently dequeuing and displaying elements.

***Input Format***

The input consists of a series of integers, one per line. Enter positive integers into the queue.

Enter -1 to terminate input.

***Output Format***

The output prints the space-separated dequeued elements.

Refer to the sample output for the exact text and format.

***Sample Test Case***

Input: 1
2
3
4
-1

Output: Dequeued elements: 1 2 3 4

***Answer***

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

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

// Queue structure
struct Queue {
    struct Node* front;
    struct Node* rear;
};

// Function to create a new queue
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = queue->rear = NULL;
    return queue;
}

// Function to enqueue an element
void enqueue(struct Queue* queue, int value) {
    if (value <= 0) return; // Only enqueue positive integers

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    // If the queue is empty, then the new node is both front and rear
    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
        return;
    }

    // Add the new node at the end of the queue and update rear
    queue->rear->next = newNode;
    queue->rear = newNode;
}

// Function to dequeue elements and return them as a space-separated string
void dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("Dequeued elements: \n");
```

```
        return; // Queue is empty
    }

    struct Node* temp = queue->front;
    printf("Dequeued elements: ");

    // Dequeue all elements
    while (temp != NULL) {
        printf("%d ", temp->data);
        queue->front = queue->front->next;
        free(temp); // Free the dequeued node
        temp = queue->front;
    }

    // Reset rear pointer
    queue->rear = NULL;
    printf("\n");
}

// Main function
int main() {
    struct Queue* queue = createQueue();
    int value;

    // Input handling
    while (1) {
        scanf("%d", &value);
        if (value == -1) break; // Terminate input on -1
        enqueue(queue, value);
    }

    // Dequeue and display elements
    dequeue(queue);

    // Free the queue structure
    free(queue);
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 5_MCQ

Attempt : 1
Total Mark : 15
Marks Obtained : 15

### Section 1 : MCQ

1. Find the postorder traversal of the given binary search tree.

***Answer***

1, 4, 2, 18, 14, 13

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

2. Which of the following is a valid preorder traversal of the binary search tree with nodes: 18, 28, 12, 11, 16, 14, 17?

***Answer***

18, 12, 11, 16, 14, 17, 28

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

3. Which of the following is the correct in-order traversal of a binary search tree with nodes: 9, 3, 5, 11, 8, 4, 2?

***Answer***

2, 3, 4, 5, 8, 9, 11

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

4. While inserting the elements 71, 65, 84, 69, 67, 83 in an empty binary search tree (BST) in the sequence shown, the element in the lowest level is _____.

***Answer***

67

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

5. How many distinct binary search trees can be created out of 4 distinct keys?

***Answer***

14

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

6. While inserting the elements 5, 4, 2, 8, 7, 10, 12 in a binary search tree, the element at the lowest level is _____.

***Answer***

12

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

7. Which of the following is the correct post-order traversal of a binary search tree with nodes: 50, 30, 20, 55, 32, 52, 57?

***Answer***

20, 32, 30, 52, 57, 55, 50

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

8. The preorder traversal of a binary search tree is 15, 10, 12, 11, 20, 18, 16, 19. Which one of the following is the postorder traversal of the tree?

***Answer***

11, 12, 10, 16, 19, 18, 20, 15

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

9. Which of the following is the correct pre-order traversal of a binary search tree with nodes: 50, 30, 20, 55, 32, 52, 57?

***Answer***

50, 30, 20, 32, 55, 52, 57

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

10. Find the preorder traversal of the given binary search tree.

***Answer***

9, 2, 1, 6, 4, 7, 10, 14

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

11. Find the in-order traversal of the given binary search tree.

***Answer***

1, 2, 4, 13, 14, 18

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

12. Which of the following operations can be used to traverse a Binary Search Tree (BST) in ascending order?

***Answer***

Inorder traversal

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

13. Find the post-order traversal of the given binary search tree.

***Answer***

10, 17, 20, 18, 15, 32, 21

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

14. Find the pre-order traversal of the given binary search tree.

***Answer***

13, 2, 1, 4, 14, 18

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

15. In a binary search tree with nodes 18, 28, 12, 11, 16, 14, 17, what is the value of the left child of the node 16?

***Answer***

14

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 5_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

John is learning about Binary Search Trees (BST) in his computer science class. He wants to create a program that allows users to delete a node with a given value from a BST and print the remaining nodes using an in-order traversal.

Implement a function to help him delete a node with a given value from a BST.

***Input Format***

The first line of input consists of an integer N, representing the number of nodes in the BST.

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

The third line consists of an integer V, which is the value to delete from the BST.

***Output Format***

The output prints the space-separated values in the BST in an in-order traversal, after the deletion of the specified value.

If the specified value is not available in the tree, print the given input values in-order traversal.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
10 5 15 2 7
15

Output: 2 5 7 10

***Answer***

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

struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};

struct TreeNode* createNode(int key) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
    newNode->data = key;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct TreeNode* insert(struct TreeNode* root, int key) {
    if (root == NULL) return createNode(key);
    if (key < root->data)
```

```
        root->left = insert(root->left, key);
    else if (key > root->data)
        root->right = insert(root->right, key);
    return root;
}

struct TreeNode* findMin(struct TreeNode* root) {
    while (root->left != NULL) {
        root = root->left;
    }
    return root;
}

struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL) return root;
    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    } else {
        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }
        struct TreeNode* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

void inorderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
```

```
}
int main()
{
    int N, rootValue, V;
    scanf("%d", &N);
    struct TreeNode* root = NULL;
    for (int i = 0; i < N; i++) {
        int key;
        scanf("%d", &key);
        if (i == 0) rootValue = key;
        root = insert(root, key);
    }
    scanf("%d", &V);
    root = deleteNode(root, V);
    inorderTraversal(root);
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 5_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

Mike is learning about Binary Search Trees (BSTs) and wants to implement various operations on them. He wants to write a basic program for creating a BST, inserting nodes, and printing the tree in the pre-order traversal.

Write a program to help him solve this program.

***Input Format***

The first line of input consists of an integer N, representing the number of values to insert into the BST.

The second line consists of N space-separated integers, representing the values to insert into the BST.

***Output Format***

The output prints the space-separated values of the BST in the pre-order traversal.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
3 1 5 2 4

Output: 3 1 2 5 4

***Answer***

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, int value) {
    if (root == NULL) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
```

```
void printPreorder(struct Node* node) {
    if (node == NULL)
        return;
    printf("%d ", node->data);
    printPreorder(node->left);
    printPreorder(node->right);
}

int main() {
    struct Node* root = NULL;

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

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

    printPreorder(root);
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 5_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

You are required to implement basic operations on a Binary Search Tree (BST), like insertion and searching.

Insertion: Given a list of integers, construct a Binary Search Tree by repeatedly inserting each integer into the tree according to the rules of a BST.

Searching: Given an integer, search for its presence in the constructed Binary Search Tree. Print whether the integer is found or not.

Write a program to calculate this efficiently.

***Input Format***

The first line of input consists of an integer n, representing the number of nodes

in the binary search tree.

The second line consists of the values of the nodes, separated by space as integers.

The third line consists of an integer representing, the value that is to be searched.

***Output Format***

The output prints, "Value <value> is found in the tree." if the given value is present, otherwise it prints: "Value <value> is not found in the tree."

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 7
8 3 10 1 6 14 23
6
Output: Value 6 is found in the tree.

***Answer***

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

struct Node {
   int data;
   Node* left;
   Node* right;
};

Node* createNode(int value) {
   Node* newNode = new Node();
   if (!newNode) {
      cout << "Memory allocation error!" << endl;
      return NULL;
   }
   newNode->data = value;
   newNode->left = newNode->right = NULL;
   return newNode;
}
```

```
Node* insertNode(Node* root, int value) {
    if (root == NULL) {
        return createNode(value);
    }

    if (value < root->data) {
        root->left = insertNode(root->left, value);
    } else if (value > root->data) {
        root->right = insertNode(root->right, value);
    }

    return root;
}

Node* searchNode(Node* root, int value) {
    if (root == NULL || root->data == value) {
        return root;
    }

    if (value < root->data) {
        return searchNode(root->left, value);
    }

    return searchNode(root->right, value);
}

int main() {
    Node* root = NULL;
    int numNodes, value, searchValue;

    cin >> numNodes;

    for (int i = 0; i < numNodes; i++) {
        cin >> value;
        root = insertNode(root, value);
    }

    cin >> searchValue;

    Node* searchResult = searchNode(root, searchValue);
    if (searchResult != NULL) {
```

```
        cout << "Value " << searchValue << " is found in the tree."<<endl;
    } else {
        cout << "Value " << searchValue << " is not found in the tree."<<endl;
    }

    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 5_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

John, a computer science student, is learning about binary search trees (BST) and their properties. He decides to write a program to create a BST, display it in post-order traversal, and find the minimum value present in the tree.

Help him by implementing the program.

***Input Format***

The first line of input consists of an integer N, representing the number of elements to insert into the BST.

The second line consists of N space-separated integers data, which is the data to be inserted into the BST.

***Output Format***

The first line of output prints the space-separated elements of the BST in post-order traversal.

The second line prints the minimum value found in the BST.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 3
5 10 15
Output: 15 10 5
The minimum value in the BST is: 5

***Answer***

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}


struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }

    if (data < root->data) {
        root->left = insert(root->left, data);
```

```
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }

    return root;
}

void displayTreePostOrder(struct Node* root) {
    if (root == NULL) {
        return;
    }
    displayTreePostOrder(root->left);
    displayTreePostOrder(root->right);
    printf("%d ", root->data);
}

int findMinValue(struct Node* root) {
    if (root == NULL) {
        return 1000000;
    }

    int leftMin = findMinValue(root->left);
    int rightMin = findMinValue(root->right);

    int min = root->data;
    if (leftMin < min) {
        min = leftMin;
    }
    if (rightMin < min) {
        min = rightMin;
    }

    return min;
}

int main() {
    struct Node* root = NULL;
    int n, data;
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        root = insert(root, data);
```

```
    }

    displayTreePostOrder(root);
    printf("\n");

    int minValue = findMinValue(root);
    printf("The minimum value in the BST is: %d", minValue);

    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 5_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

In his computer science class, John is learning about Binary Search Trees (BST). He wants to build a BST and find the maximum value in the tree.

Help him by writing a program to insert nodes into a BST and find the maximum value in the tree.

***Input Format***

The first line of input consists of an integer N, representing the number of nodes in the BST.

The second line consists of N space-separated integers, representing the values of the nodes to insert into the BST.

***Output Format***

The output prints the maximum value in the BST.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
10 5 15 2 7

Output: 15

***Answer***

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

struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* createNode(int key) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
    newNode->data = key;
    newNode->left = newNode->right = NULL;
    return newNode;
}


struct TreeNode* insert(struct TreeNode* root, int key) {
    if (root == NULL) return createNode(key);
    if (key < root->data)
        root->left = insert(root->left, key);
    else if (key > root->data)
        root->right = insert(root->right, key);
    return root;
}

int findMax(struct TreeNode* root) {
    if (root == NULL) return -1;
    while (root->right != NULL) {
```

```
        root = root->right;
    }
    return root->data;
}

int main() {
    int N, rootValue;
    scanf("%d", &N);

    struct TreeNode* root = NULL;

    for (int i = 0; i < N; i++) {
        int key;
        scanf("%d", &key);
        if (i == 0) rootValue = key;
        root = insert(root, key);
    }

    int maxVal = findMax(root);
    if (maxVal != -1) {
        printf("%d", maxVal);
    }

    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

## REC_DS using C_Week 5_PAH_Updated

Attempt : 1
Total Mark : 50
Marks Obtained : 40

## Section 1 : Coding

1. Problem Statement

Aishu is participating in a coding challenge where she needs to reconstruct a Binary Search Tree (BST) from given preorder traversal data and then print the in-order traversal of the reconstructed BST.

Since Aishu is just learning about tree data structures, she needs your help to write a program that does this efficiently.

***Input Format***

The first line consists of an integer n, representing the number of nodes in the BST.

The second line of input contains n integers separated by spaces, which represent the preorder traversal of the BST.

***Output Format***

The output displays n space-separated integers, representing the in-order traversal of the reconstructed BST.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 6
10 5 1 7 40 50

Output: 1 5 7 10 40 50

***Answer***

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

struct Node* buildBST(int preorder[], int* index, int min, int max, int n) {
    if (*index >= n) return NULL;
    int val = preorder[*index];
    if (val < min || val > max) return NULL;
    struct Node* root = newNode(val);
    (*index)++;
    root->left = buildBST(preorder, index, min, val - 1, n);
    root->right = buildBST(preorder, index, val + 1, max, n);
    return root;
```

```
}

void inorder(struct Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

int main() {
    int n;
    scanf("%d", &n);
    int preorder[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &preorder[i]);
    }
    int index = 0;
    struct Node* root = buildBST(preorder, &index, 0, 100, n);
    inorder(root);
    return 0;
}
```

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

2. Problem Statement

Viha, a software developer, is working on a project to automate searching for a target value in a Binary Search Tree (BST). She needs to create a program that takes an integer target value as input and determines if that value is present in the BST or not.

Write a program to assist Viha.

***Input Format***

The first line of input consists of integers separated by spaces, which represent the elements to be inserted into the BST. The input is terminated by entering -1.

The second line consists of an integer target, which represents the target value to be searched in the BST.

***Output Format***

If the target value is found in the BST, print "[target] is found in the BST".

Else, print "[target] is not found in the BST"

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5 3 7 1 4 6 8 -1
4
Output: 4 is found in the BST

***Answer***

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) return newNode(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data);
    return root;
}
```

```
int search(struct Node* root, int target) {
    if (root == NULL) return 0;
    if (root->data == target) return 1;
    if (target < root->data) return search(root->left, target);
    return search(root->right, target);
}

int main() {
    struct Node* root = NULL;
    int data;
    while (scanf("%d", &data) && data != -1) {
        root = insert(root, data);
    }
    int target;
    scanf("%d", &target);
    if (search(root, target))
        printf("%d is found in the BST\n", target);
    else
        printf("%d is not found in the BST\n", target);
    return 0;
}
```

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

3. Problem Statement

Yogi is working on a program to manage a binary search tree (BST) containing integer values. He wants to implement a function that removes nodes from the tree that fall outside a specified range defined by a minimum and maximum value.

Help Yogi by writing a function that achieves this.

***Input Format***

The first line of input consists of an integer N, representing the number of elements to be inserted into the BST.

The second line consists of N space-separated integers, representing the elements to be inserted into the BST.

The third line consists of two space-separated integers min and max, representing the minimum value and the maximum value of the range.

***Output Format***

The output prints the remaining elements of the BST in an in-order traversal, after removing nodes that fall outside the specified range.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
10 5 15 20 12
5 15

Output: 5 10 12 15

***Answer***

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) return newNode(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else
```

```
        root->right = insert(root->right, data);
    return root;
}

struct Node* removeOutOfRange(struct Node* root, int min, int max) {
    if (root == NULL) return NULL;
    root->left = removeOutOfRange(root->left, min, max);
    root->right = removeOutOfRange(root->right, min, max);
    if (root->data < min) {
        struct Node* temp = root->right;
        free(root);
        return temp;
    }
    if (root->data > max) {
        struct Node* temp = root->left;
        free(root);
        return temp;
    }
    return root;
}

void inorder(struct Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

int main() {
    int N;
    scanf("%d", &N);
    int elements[N];
    for (int i = 0; i < N; i++) {
        scanf("%d", &elements[i]);
    }
    int min, max;
    scanf("%d %d", &min, &max);
    struct Node* root = NULL;
    for (int i = 0; i < N; i++) {
        root = insert(root, elements[i]);
    }
    root = removeOutOfRange(root, min, max);
```

```
    inorder(root);
    return 0;
}
```

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

4. Problem Statement

Arun is exploring operations on binary search trees (BST). He wants to write a program with an unsorted distinct integer array that represents the BST keys and construct a height-balanced BST from it.

After constructing, he wants to perform the following operations that can alter the structure of the tree and traverse them using a level-order traversal:

InsertionDeletion

Your task is to assist Arun in completing the program without any errors.

***Input Format***

The first line of input consists of an integer N, representing the number of initial keys in the BST.

The second line consists of N space-separated integers, representing the initial keys.

The third line consists of an integer X, representing the new key to be inserted into the BST.

The fourth line consists of an integer Y, representing the key to be deleted from the BST.

***Output Format***

The first line of output prints "Initial BST: " followed by a space-separated list of keys in the initial BST after constructing it in level order traversal.

The second line prints "BST after inserting a new node X: " followed by a space-separated list of keys in the BST after inserting X n level order traversal.

The third line prints "BST after deleting node Y: " followed by a space-separated list of keys in the BST after deleting Y n level order traversal.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
25 14 56 28 12
34
12
Output: Initial BST: 25 14 56 12 28
BST after inserting a new node 34: 25 14 56 12 28 34
BST after deleting node 12: 25 14 56 28 34

***Answer***

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

void inorder(int arr[], int start, int end) {
    if (start > end) return;
    int mid = (start + end) / 2;
    inorder(arr, start, mid - 1);
    printf("%d ", arr[mid]);
    inorder(arr, mid + 1, end);
}
```

```
struct Node* sortedArrayToBST(int arr[], int start, int end) {
    if (start > end) return NULL;
    int mid = (start + end) / 2;
    struct Node* root = newNode(arr[mid]);
    root->left = sortedArrayToBST(arr, start, mid - 1);
    root->right = sortedArrayToBST(arr, mid + 1, end);
    return root;
}

void levelOrder(struct Node* root) {
    if (root == NULL) return;
    struct Node* queue[100];
    int front = 0, rear = 0;
    queue[rear++] = root;
    while (front < rear) {
        struct Node* node = queue[front++];
        printf("%d ", node->data);
        if (node->left) queue[rear++] = node->left;
        if (node->right) queue[rear++] = node->right;
    }
}

struct Node* insert(struct Node* root, int key) {
    if (root == NULL) return newNode(key);
    if (key < root->data)
        root->left = insert(root->left, key);
    else
        root->right = insert(root->right, key);
    return root;
}

struct Node* minValueNode(struct Node* node) {
    struct Node* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

struct Node* deleteNode(struct Node* root, int key) {
    if (root == NULL) return root;
    if (key < root->data)
```

```
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }
        struct Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

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

    int sortedKeys[N];
    for (int i = 0; i < N; i++) {
        sortedKeys[i] = keys[i];
    }

    qsort(sortedKeys, N, sizeof(int), (int (*)(const void*, const void*))strcmp);

    struct Node* root = sortedArrayToBST(sortedKeys, 0, N - 1);

    printf("Initial BST: ");
    levelOrder(root);
```

```
    printf("\n");

    root = insert(root, X);
    printf("BST after inserting a new node %d: ", X);
    levelOrder(root);
    printf("\n");

    root = deleteNode(root, Y);
    printf("BST after deleting node %d: ", Y);
    levelOrder(root);
    printf("\n");

    return 0;
}
```

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

5. Problem Statement

Joseph, a computer science student, is interested in understanding binary search trees (BST) and their node arrangements. He wants to create a program to explore BSTs by inserting elements into a tree and displaying the nodes using post-order traversal of the tree.

Write a program to help Joseph implement the program.

***Input Format***

The first line of input consists of an integer N, representing the number of elements to insert into the BST.

The second line consists of N space-separated integers data, which is the data to be inserted into the BST.

***Output Format***

The output prints N space-separated integer values after the post-order traversal.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 4
10 15 5 3

Output: 3 5 15 10

***Answer***

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) return newNode(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data);
    return root;
}

void postOrder(struct Node* root) {
    if (root == NULL) return;
    postOrder(root->left);
    postOrder(root->right);
    printf("%d ", root->data);
}

int main() {
```

```
    int N;
    scanf("%d", &N);
    int data;
    struct Node* root = NULL;
    for (int i = 0; i < N; i++) {
        scanf("%d", &data);
        root = insert(root, data);
    }
    postOrder(root);
    printf("\n");
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 6_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

John and Mary are collaborating on a project that involves data analysis. They each have a set of age data, one sorted in ascending order and the other in descending order. However, their analysis requires the data to be in ascending order.

Write a program to help them merge the two sets of age data into a single sorted array in ascending order using merge sort.

***Input Format***

The first line of input consists of an integer N, representing the number of age values in each dataset.

The second line consists of N space-separated integers, representing the ages of participants in John's dataset (in ascending order).

The third line consists of N space-separated integers, representing the ages of participants in Mary's dataset (in descending order).

***Output Format***

The output prints a single line containing space-separated integers, which represents the merged dataset of ages sorted in ascending order.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
1 3 5 7 9
10 8 6 4 2

Output: 1 2 3 4 5 6 7 8 9 10

***Answer***

```
#include <stdio.h>

void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[20], R[20];
    for (int i = 0; i < n1; i++) L[i] = arr[l + i];
    for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];
    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
```

```
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

int main() {
    int n, m;
    scanf("%d", &n);
    int arr1[n], arr2[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr1[i]);
    }
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr2[i]);
    }
    int merged[n + n];
    mergeSort(arr1, n);
    mergeSort(arr2, n);
    merge(merged, arr1, arr2, n, n);
    for (int i = 0; i < n + n; i++) {
        printf("%d ", merged[i]);
    }
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 6_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

Nandhini asked her students to arrange a set of numbers in ascending order. She asked the students to arrange the elements using insertion sort, which involves taking each element and placing it in its appropriate position within the sorted portion of the array.

Assist them in the task.

***Input Format***

The first line of input consists of the value of n, representing the number of array elements.

The second line consists of n elements, separated by a space.

***Output Format***

The output prints the sorted array, separated by a space.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
67 28 92 37 59

Output: 28 37 59 67 92

***Answer***

```
#include <stdio.h>

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

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int n;
    scanf("%d", &n);

    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    insertionSort(arr, n);

    for (int i = 0; i < n; i++) {
```

```
        printf("%d ", arr[i]);
    }

    printf("\n");
    return 0;
}
int main() {
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    insertionSort(arr, n);
    printArray(arr, n);
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 6_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

You are the lead developer of a text-processing application that assists writers in organizing their thoughts. One crucial feature is a character-sorting service that helps users highlight the most critical elements of their text.

To achieve this, you decide to enhance the service to sort characters in descending order using the Quick-Sort algorithm. Implement the algorithm to efficiently rearrange the characters, ensuring that it is sorted in descending order.

***Input Format***

The first line of the input consists of a positive integer value N, representing the number of characters to be sorted.

The second line of input consists of N space-separated lowercase alphabetical characters.

***Output Format***

The output displays the set of alphabetical characters, sorted in descending order.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 5
a d g j k
Output: k j g d a

***Answer***

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

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

void swap(char *a, char *b) {
   char temp = *a;
   *a = *b;
   *b = temp;
}

int partition(char arr[], int low, int high) {
   char pivot = arr[high];
   int i = low - 1;
   for (int j = low; j < high; j++) {
      if (arr[j] > pivot) {
         i++;
         swap(&arr[i], &arr[j]);
      }
   }
   swap(&arr[i + 1], &arr[high]);
   return i + 1;
```

```
}

void quickSort(char arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    int n;
    scanf("%d", &n);

    char arr[n];
    for (int i = 0; i < n; i++) {
        scanf(" %c", &arr[i]);
    }

    quickSort(arr, 0, n - 1);

    for (int i = 0; i < n; i++) {
        printf("%c ", arr[i]);
    }

    printf("\n");
    return 0;
}
int main() {
    int n;
    scanf("%d", &n);

    char characters[n];

    for (int i = 0; i < n; i++) {
        char input;
        scanf(" %c", &input);
        characters[i] = input;
    }

    quicksort(characters, 0, n - 1);
```

```
    for (int i = 0; i < n; i++) {
        printf("%c ", characters[i]);
    }

    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 6_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

Kavya, a software developer, is analyzing data trends. She has a list of integers and wants to identify the nth largest number in the list after sorting the array using QuickSort.

To optimize performance, Kavya is required to use QuickSort to sort the list before finding the nth largest number.

***Input Format***

The first line of input consists of an integer n, representing the size of the array.

The second line consists of n space-separated integers, representing the elements of the array nums.

The third line consists of an integer k, representing the position of the largest

number you need to print after sorting the array.

***Output Format***

The output prints the k-th largest number in the sorted array (sorted in ascending order).

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 6
-1 0 1 2 -1 -4
3

Output: 0

***Answer***

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


void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}
```

```
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void findNthLargest(int* nums, int n, int k) {
    quickSort(nums, 0, n - 1);
    printf("%d\n", nums[n - k]);
}


int main() {
    int n, k;
    scanf("%d", &n);
    int* nums = (int*)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        scanf("%d", &nums[i]);
    }
    scanf("%d", &k);
    findNthLargest(nums, n, k);
    free(nums);
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 6_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

Jose has an array of N fractional values, represented as double-point numbers. He needs to sort these fractions in increasing order and seeks your help.

Write a program to help Jose sort the array using the merge sort algorithm.

***Input Format***

The first line of input consists of an integer N, representing the number of fractions to be sorted.

The second line consists of N double-point numbers, separated by spaces, representing the fractions array.

***Output Format***

The output prints N double-point numbers, sorted in increasing order, and rounded to three decimal places.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 4
0.123 0.543 0.321 0.789

Output: 0.123 0.321 0.543 0.789

***Answer***

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

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

void merge(double arr[], int left, int mid, int right) {
    int n1 = mid - left + 1, n2 = right - mid;
    double *L = (double*)malloc(n1 * sizeof(double));
    double *R = (double*)malloc(n2 * sizeof(double));

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];

    free(L);
    free(R);
}

void mergeSort(double arr[], int left, int right) {
```

```
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    int n;
    scanf("%d", &n);

    double *arr = (double*)malloc(n * sizeof(double));
    for (int i = 0; i < n; i++) {
        scanf("%lf", &arr[i]);
    }

    mergeSort(arr, 0, n - 1);

    for (int i = 0; i < n; i++) {
        printf("%.3f ", arr[i]);
    }

    printf("\n");
    free(arr);
    return 0;
}
```

```
int main() {
    int n;
    scanf("%d", &n);
    double fractions[n];
    for (int i = 0; i < n; i++) {
        scanf("%lf", &fractions[i]);
    }
    mergeSort(fractions, 0, n - 1);
    for (int i = 0; i < n; i++) {
        printf("%.3f ", fractions[i]);
    }
    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

## REC_DS using C_Week 6_PAH_Updated

Attempt : 1
Total Mark : 50
Marks Obtained : 10

## Section 1 : Coding

1. Problem Statement

Alex is working on a project that involves merging and sorting two arrays. He wants to write a program that merges two arrays, sorts the merged array in ascending order, removes duplicates, and prints the sorted array without duplicates.

Help Alex to implement the program using the merge sort algorithm.

***Input Format***

The first line of input consists of an integer N, representing the number of elements in the first array.

The second line consists of N integers, separated by spaces, representing the elements of the first array.

The third line consists of an integer M, representing the number of elements in the second array.

The fourth line consists of M integers, separated by spaces, representing the elements of the second array.

***Output Format***

The output prints space-separated integers, representing the merged and sorted array in ascending order, with duplicate elements removed.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 4
1 2 3 4
3
3 4 5

Output: 1 2 3 4 5

***Answer***

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

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1, n2 = right - mid;
    int *L = (int*)malloc(n1 * sizeof(int));
    int *R = (int*)malloc(n2 * sizeof(int));

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
```

```
    free(L);
    free(R);
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

void removeDuplicates(int arr[], int *n) {
    int temp[*n], index = 0;
    temp[index++] = arr[0];

    for (int i = 1; i < *n; i++) {
        if (arr[i] != arr[i - 1]) {
            temp[index++] = arr[i];
        }
    }

    for (int i = 0; i < index; i++) {
        arr[i] = temp[i];
    }
    *n = index;
}

int main() {
    int n, m;
    scanf("%d", &n);
    int arr1[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr1[i]);
    }

    scanf("%d", &m);
    int arr2[m];
    for (int i = 0; i < m; i++) {
        scanf("%d", &arr2[i]);
```

```
    }

    int mergedSize = n + m;
    int mergedArr[mergedSize];

    for (int i = 0; i < n; i++) mergedArr[i] = arr1[i];
    for (int i = 0; i < m; i++) mergedArr[n + i] = arr2[i];

    mergeSort(mergedArr, 0, mergedSize - 1);
    removeDuplicates(mergedArr, &mergedSize);

    for (int i = 0; i < mergedSize; i++) {
        printf("%d ", mergedArr[i]);
    }

    printf("\n");
    return 0;
}
```

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

2. Problem Statement

You're a coach managing a list of finishing times for athletes in a race. The times are stored in an array, and you need to sort this array in ascending order to determine the rankings.

You'll use the insertion sort algorithm to accomplish this.

***Input Format***

The first line of input contains an integer n, representing the number of athletes.

The second line contains n space-separated integers, each representing the finishing time of an athlete in seconds.

***Output Format***

The output prints the sorted finishing times of the athletes in ascending order.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
75 89 65 90 70

Output: 65 70 75 89 90

***Answer***

-

***Status :*** -  ***Marks : 0/10***

3. Problem Statement

You are working on an optimization task for a sorting algorithm that uses insertion sort. Your goal is to determine the efficiency of the algorithm by counting the number of swaps needed to sort an array of integers.

Write a program that takes an array as input and calculates the number of swaps performed during the insertion sort process.

Example 1:

Input:

5

2 1 3 1 2

Output:

4

Explanation:

Step 1: [2, 1, 3, 1, 2] (No swaps)

Step 2: [1, 2, 3, 1, 2] (1 swap, element 1 shifts 1 place to the left)

Step 3: [1, 2, 3, 1, 2] (No swaps)

Step 4: [1, 1, 2, 3, 2] (2 swaps; element 1 shifts 2 places to the left)

Step 5: [1, 1, 2, 2, 3] (1 swap, element 2 shifts 1 place to the left)

Total number of swaps: 1 + 2 + 1 = 4

Example 2:

Input:

7

12 15 1 5 6 14 11

Output:

10

Explanation:

Step 1: [12, 15, 1, 5, 6, 14, 11] (No swaps)

Step 2: [12, 15, 1, 5, 6, 14, 11] (1 swap, element 15 shifts 1 place to the left)

Step 3: [12, 15, 1, 5, 6, 14, 11] (No swaps)

Step 4: [1, 12, 15, 5, 6, 14, 11] (2 swaps, element 1 shifts 2 places to the left)

Step 5: [1, 5, 12, 15, 6, 14, 11] (1 swap, element 5 shifts 1 place to the left)

Step 6: [1, 5, 6, 12, 15, 14, 11] (2 swaps, element 6 shifts 2 places to the left)

Step 7: [1, 5, 6, 12, 14, 15, 11] (1 swap, element 14 shifts 1 place to the left)

Step 8: [1, 5, 6, 11, 12, 14, 15] (3 swaps, element 11 shifts 3 places to the left)

Total number of swaps: 1 + 2 + 1 + 2 + 1 + 3 = 10

***Input Format***

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

The second line of input consists of n space-separated integers, representing the elements of the array.

***Output Format***

The output prints the number of swaps performed during the insertion sort process.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 5
2 1 3 1 2
Output: 4

***Answer***

-

***Status :*** -  ***Marks : 0/10***

4. Problem Statement

Vishnu, a math enthusiast, is given a task to explore the magic of numbers. He has an array of positive integers, and his goal is to find the integer with the highest digit sum in the sorted array using the merge sort algorithm.

You have to assist Vishnu in implementing the merge sort algorithm.

***Input Format***

The first line of input consists of an integer N, representing the number of elements in the array.

The second line consists of N space-separated integers, representing the array elements.

***Output Format***

The first line of output prints "The sorted array is: " followed by the sorted array, separated by a space.

The second line prints "The integer with the highest digit sum is: " followed by an integer representing the highest-digit sum.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
123 456 789 321 654
Output: The sorted array is: 123 321 456 654 789
The integer with the highest digit sum is: 789

***Answer***

-

***Status :*** -  ***Marks : 0/10***

5. Problem Statement

You are working as a programmer at a sports academy, and the academy holds various sports competitions regularly.

As part of the academy's system, you need to sort the scores of the participants in descending order using the Quick Sort algorithm.

Write a program that takes the scores of n participants as input and uses the Quick Sort algorithm to sort the scores in descending order. Your program should display the sorted scores after the sorting process.

***Input Format***

The first line of input consists of an integer n, which represents the number of scores.

The second line of input consists of n integers, which represent scores separated by spaces.

***Output Format***

Each line of output represents an iteration of the Quick Sort algorithm, displaying the elements of the array at that iteration.

After the iterations are complete, the last line of output prints the sorted scores in descending order separated by space.

Refer to the sample outputs for the formatting specifications.

***Sample Test Case***

Input: 5
78 54 96 32 53
Output: Iteration 1: 78 54 96 53 32
Iteration 2: 96 54 78
Iteration 3: 78 54
Sorted Order: 96 78 54 53 32

***Answer***

*-*

***Status :*** *-* ***Marks : 0/10***

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 7_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

Ravi is building a basic hash table to manage student roll numbers for quick lookup. He decides to use Linear Probing to handle collisions.

Implement a hash table using linear probing where:

The hash function is: index = roll_number % table_sizeOn collision, check subsequent indexes (i+1, i+2, ...) until an empty slot is found.

You need to:

Insert a list of n student roll numbers into the hash table.Print the final state of the hash table. If a slot is empty, print -1.

***Input Format***

The first line of the input contains two integers n and table_size, where n is the

number of roll numbers to be inserted, and table_size is the size of the hash table.

The second line contains n space-separated integers — the roll numbers to insert into the hash table.

***Output Format***

The output should print a single line with table_size space-separated integers representing the final state of the hash table after all insertions.

If any slot remains unoccupied, it should be represented as -1.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 4 7
50 700 76 85

Output: 700 50 85 -1 -1 -1 76

***Answer***

-

***Status :*** Skipped ***Marks : 0/10***

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 7_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

Priya is developing a simple student management system. She wants to store roll numbers in a hash table using Linear Probing, and later search for specific roll numbers to check if they exist.

Implement a hash table using linear probing with the following operations:

Insert all roll numbers into the hash table.For a list of query roll numbers, print "Value x: Found" or "Value x: Not Found" depending on whether it exists in the table.

***Input Format***

The first line contains two integers, n and table_size — the number of roll numbers to insert and the size of the hash table.

The second line contains n space-separated integers — the roll numbers to insert.

The third line contains an integer q — the number of queries.

The fourth line contains q space-separated integers — the roll numbers to search for.

***Output Format***

The output print q lines — for each query value x, print: "Value x: Found" or "Value x: Not Found"

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5 10
21 31 41 51 61
3
31 60 51
Output: Value 31: Found
Value 60: Not Found
Value 51: Found

***Answer***

```
#include <stdio.h>

#define MAX 100


void initializeTable(int table[], int size)

{


   for (int i = 0; i < size; i++)

{
```

```
        table[i] = -1;

}

}

int linearProbe(int table[], int size, int num)

{

   int index = num % size;
   while (table[index] != -1)

{

        index = (index + 1) % size;

}
   return index;

}

void insertIntoHashTable(int table[], int size, int arr[], int n)

{

   for (int i = 0; i < n; i++)

{

        int index = linearProbe(table, size, arr[i]);
        table[index] = arr[i];

}
```

```
}

void searchInHashTable(int table[], int size, int queries[], int q)

{


   for (int i = 0; i < q; i++)

{


      int num = queries[i];
      int index = num % size;
      int found = 0;

      while (table[index] != -1)

{


         if (table[index] == num)

{


            found = 1;
            break;


}
         index = (index + 1) % size;


}

      if (found)

{
```

```
            printf("Value %d: Found\n", num);


} else

{


            printf("Value %d: Not Found\n", num);


}


}

}
int main() {
    int n, table_size;
    scanf("%d %d", &n, &table_size);

    int arr[MAX], table[MAX];
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    initializeTable(table, table_size);
    insertIntoHashTable(table, table_size, arr, n);

    int q, x;
    scanf("%d", &q);
    for (int i = 0; i < q; i++) {
        scanf("%d", &x);
        if (searchInHashTable(table, table_size, x))
            printf("Value %d: Found\n", x);
        else
            printf("Value %d: Not Found\n", x);
    }

    return 0;
}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 7_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

In a messaging application, users maintain a contact list with names and corresponding phone numbers. Develop a program to manage this contact list using a dictionary implemented with hashing.

The program allows users to add contacts, delete contacts, and check if a specific contact exists. Additionally, it provides an option to print the contact list in the order of insertion.

***Input Format***

The first line consists of an integer n, representing the number of contact pairs to be inserted.

Each of the next n lines consists of two strings separated by a space: the name of the contact (key) and the corresponding phone number (value).

The last line contains a string k, representing the contact to be checked or removed.

***Output Format***

If the given contact exists in the dictionary:

1. The first line prints "The given key is removed!" after removing it.
2. The next n - 1 lines print the updated contact list in the format: "Key: X; Value: Y" where X represents the contact's name and Y represents the phone number.

If the given contact does not exist in the dictionary:

1. The first line prints "The given key is not found!".
2. The next n lines print the original contact list in the format: "Key: X; Value: Y" where X represents the contact's name and Y represents the phone number.

Refer to the sample outputs for the formatting specifications.

***Sample Test Case***

Input: 3
Alice 1234567890
Bob 9876543210
Charlie 4567890123
Bob

Output: The given key is removed!
Key: Alice; Value: 1234567890
Key: Charlie; Value: 4567890123

***Answer***

```
// You are using GCC
n = int(input().strip())

contacts =

{
```

```
}
order = []

for _ in range(n):
    name, number = input().strip().split()
    contacts[name] = number
    order.append(name)

k = input().strip()

if k in contacts:
    print("The given key is removed!")
    contacts.pop(k)
    order.remove(k)
else:
    print("The given key is not found!")

for name in order:
    print(f"Key")



name

}; Value:

{

contacts[name]

}")



{

name

}; Value:

{
```

contacts[name]

name

}; Value:

{

contacts[name]

}"

{

name

}; Value:

{

contacts[name]

}

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 7_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

### Section 1 : Coding

1. Problem Statement

Develop a program using hashing to manage a fruit contest where each fruit is assigned a unique name and a corresponding score. The program should allow the organizer to input the number of fruits and their names with scores.

Then, it should enable them to check if a specific fruit, identified by its name, is part of the contest. If the fruit is registered, the program should display its score; otherwise, it should indicate that it is not included in the contest.

***Input Format***

The first line consists of an integer N, representing the number of fruits in the contest.

The following N lines contain a string K and an integer V, separated by a space, representing the name and score of each fruit in the contest.

The last line consists of a string T, representing the name of the fruit to search for.

***Output Format***

If T exists in the dictionary, print "Key "T" exists in the dictionary.".

If T does not exist in the dictionary, print "Key "T" does not exist in the dictionary.".

Refer to the sample outputs for the formatting specifications.

***Sample Test Case***

Input: 2
banana 2
apple 1
Banana
Output: Key "Banana" does not exist in the dictionary.

***Answer***

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

typedef struct

{


    char key[20]; // Fruit name
    int value;    // Score

} KeyValuePair;

typedef struct
```

```
{


    KeyValuePair *pairs;
    int size;
    int capacity;

} Dictionary;

void initDictionary(Dictionary *dict, int capacity)

{


    dict->size = 0;
    dict->capacity = capacity;
    dict->pairs = (KeyValuePair *)malloc(capacity * sizeof(KeyValuePair));

}

void insertKeyValuePair(Dictionary *dict, const char *key, int value)

{


    if (dict->size < dict->capacity)

{


        strcpy(dict->pairs[dict->size].key, key);
        dict->pairs[dict->size].value = value;
        dict->size++;


}

}

int searchKey(Dictionary *dict, const char *key)
```

```
{

    for (int i = 0; i < dict->size; i++)

{

        if (strcmp(dict->pairs[i].key, key) == 0)

{

            printf("Key \"%s\" exists in the dictionary.\n", key);
            return 1;


}


}
    printf("Key \"%s\" does not exist in the dictionary.\n", key);
    return 0;

}

void freeDictionary(Dictionary *dict)

{


    free(dict->pairs);

}

int main()

{


    int N;
    scanf("%d", &N);
```

```
    Dictionary dict;
    initDictionary(&dict, N);

    char key[20];
    int value;
    for (int i = 0; i < N; i++)

{


        scanf("%s %d", key, &value);
        insertKeyValuePair(&dict, key, value);


}

    scanf("%s", key);
    searchKey(&dict, key);

    freeDictionary(&dict);
    return 0;

}
```

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

# Rajalakshmi Engineering College

Name: GIFFIN M  STEVE
Email: 240701145@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE AG
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

### REC_DS using C_Week 7_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 0

### Section 1 : Coding

1. Problem Statement

You are provided with a collection of numbers, each represented by an array of integers. However, there's a unique scenario: within this array, one element occurs an odd number of times, while all other elements occur an even number of times. Your objective is to identify and return the element that occurs an odd number of times in this arrangement.

Utilize mid-square hashing by squaring elements and extracting middle digits for hash codes. Implement a hash table for efficient integer occurrence tracking.

Note: Hash function: squared = key * key.

Example

Input:

7

2 2 3 3 4 4 5

Output:

5

Explanation

The hash function and the calculated hash indices for each element are as follows:

2 -> hash(2*2) % 100 = 4

3 -> hash(3*3) % 100 = 9

4 -> hash(4*4) % 100 = 16

5 -> hash(5*5) % 100 = 25

The hash table records the occurrence of each element's hash index:

Index 4: 2 occurrences

Index 9: 2 occurrences

Index 16: 2 occurrences

Index 25: 1 occurrence

Among the elements, the integer 5 occurs an odd number of times (1 occurrence) and satisfies the condition of the problem. Therefore, the program outputs 5.

***Input Format***

The first line of input consists of an integer N, representing the size of the array.

The second line consists of N space-separated integers, representing the elements of the array.

***Output Format***

The output prints a single integer representing the element that occurs an odd

number of times.

If no such element exists, print -1.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 7
2 2 3 3 4 4 5
Output: 5

***Answer***

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

#define MAX_SIZE 100

void initializeTable(int table[])

{

   for (int i = 0; i < TABLE_SIZE; i++)

{

      table[i] = 0;

}

}

int midSquareHash(int key)
```

```
{

    return (key * key) % TABLE_SIZE; // Extract middle digits using modulo

}

int findOddOccurrence(int arr[], int n)

{


    int hashTable[TABLE_SIZE];
    initializeTable(hashTable);

    // Populate the hash table with occurrences
    for (int i = 0; i < n; i++)

{


        int hashIndex = midSquareHash(arr[i]);
        hashTable[hashIndex]++; // Count occurrences


}

    // Find element with an odd occurrence
    for (int i = 0; i < n; i++)

{


        int hashIndex = midSquareHash(arr[i]);
        if (hashTable[hashIndex] % 2 == 1)

{


            return arr[i];
```

```
}


}

    return -1; // Return -1 if no odd occurrences exist

}
int main() {
    int n;
    scanf("%d", &n);

    int arr[MAX_SIZE];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    printf("%d\n", getOddOccurrence(arr, n));

    return 0;
}
```

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