Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_PAH_modified

Attempt : 2
Total Mark : 5
Marks Obtained : 3

Section 1 : Coding

1. 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

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

```
#include<stdlib.h>
struct node{
   int data;
   struct node *next;
};struct node*ptr,*newnode,*head=NULL,*last=NULL;
int main()
{int a;

    scanf("%d",&a);
    int even[a],odd[a],n=0,m=0;
    for(int i=0;i<a;i++)
    {newnode=(struct node*)malloc(sizeof(struct node));
       scanf("%d",&newnode->data);
       newnode->next=NULL;
       if(head==NULL)
       {head=newnode;}
       else
       {last->next=newnode;}
       last=newnode;
    }ptr=head;
    while(ptr!=NULL)
    {if((ptr->data) %2==0)
    {even[n++]=ptr->data;}
    else
    {odd[m++]=ptr->data;}
    ptr=ptr->next;
    }

    for(int i=n-1;i>=0;i--)
    {printf("%d ",even[i]);}
    for(int i=0;i<m;i++)
    {printf("%d ",odd[i]);}
    return 0;
}
```

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

2. 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

```
// You are using GCC
#include<stdio.h>
#include<stdlib.h>
 struct node{
     int data;
     struct node*next;};
     struct node *head=NULL,*last=NULL,*newnode,*ptr;
void create(int a)
{
 for(int i=0;i<a;i++)
{
newnode=(struct node*)malloc(sizeof(struct node));
scanf("%d",&newnode->data);
newnode->next=NULL;
if(head==NULL)
{head=newnode;}
else
{last->next=newnode;}
last=newnode;
}}
void display()
{ptr=head;
    while(ptr!=NULL)
    {printf("%d ",ptr->data);
        ptr=ptr->next;
        }

 }
   int main()
   {int a,b;
   scanf("%d",&a);
   create(a);
   scanf("%d",&b);
   create(b);
   display();
   return 0;
}
```

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

3. 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<stdlib.h>
#include<math.h>
struct node{
    int data;
    struct node *next;
};struct node *head=NULL,*last=NULL,*ptr,*newnode;
int main()
{int a,b,s=0;
scanf("%d",&a);
for(int i=0;i<a+1;i++)
{newnode=(struct node*)malloc(sizeof(struct node));
   scanf("%d",&newnode->data);
   newnode->next=NULL;
   if(head==NULL)
   {head=newnode;}
   else
```

```
  {last->next=newnode;}
  last=newnode;
}
scanf("%d",&b);
int n=a;
ptr=head;
while(ptr!=NULL)
{s+=(pow(b,n--))*(ptr->data);
ptr=ptr->next;}
printf("%d",s);
}
```

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

4. 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

// You are using GCC

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

// Define a structure for a singly linked list node
struct Node {
    int data;
    struct Node* next;
};

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

// Function to insert a node at the beginning
void insertBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
    printf("The linked list after insertion at the beginning is:\n");
}

// Function to insert a node at the end
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    printf("The linked list after insertion at the end is:\n");
}

// Function to insert before a specific value
void insertBefore(struct Node** head, int value, int data) {
```

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

    if ((*head)->data == value) {
        insertBeginning(head, data);
        return;
    }

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

    if (temp->next == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    struct Node* newNode = createNode(data);
    newNode->next = temp->next;
    temp->next = newNode;
    printf("The linked list after insertion before a value is:\n");
}

// Function to insert after a specific value
void insertAfter(struct Node** head, int value, int data) {
    struct Node* temp = *head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }

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

    struct Node* newNode = createNode(data);
    newNode->next = temp->next;
    temp->next = newNode;
    printf("The linked list after insertion after a value is:\n");
```

```
}

// Function to delete the first node
void deleteBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty\n");
        return;
    }

    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    printf("The linked list after deletion from the beginning is:\n");
}

// Function to delete the last node
void deleteEnd(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty\n");
        return;
    }

    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        printf("The linked list after deletion from the end is:\n");
        return;
    }

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

    free(temp->next);
    temp->next = NULL;
    printf("The linked list after deletion from the end is:\n");
}

// Function to delete a node before a specific value
void deleteBefore(struct Node** head, int value) {
    if (*head == NULL || (*head)->data == value) {
```

```
        printf("Value not found in the list\n");
        return;
    }

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

    while (temp->next != NULL && temp->next->data != value) {
        prev = temp;
        temp = temp->next;
    }

    if (temp->next == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    if (prev == NULL) {
        *head = temp->next;
    } else {
        prev->next = temp->next;
    }

    free(temp);
    printf("The linked list after deletion before a value is:\n");
}

// Function to delete a node after a specific value
void deleteAfter(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("Value not found in the list\n");
        return;
    }

    struct Node* nodeToDelete = temp->next;
    temp->next = nodeToDelete->next;
```

```
    free(nodeToDelete);
    printf("The linked list after deletion after a value is:\n");
}

// Function to display the linked list
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty\n");
        return;
    }

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

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

    while (1) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("LINKED LIST CREATED\n");
                while (1) {
                    scanf("%d", &data);
                    if (data == -1) break;
                    insertEnd(&head, data);
                }
                break;
            case 2:
                displayList(head);
                break;
            case 3:
                scanf("%d", &data);
                insertBeginning(&head, data);
                displayList(head);
                break;
```

```
            case 4:
                scanf("%d", &data);
                insertEnd(&head, data);
                displayList(head);
                break;
            case 5:
                scanf("%d %d", &value, &data);
                insertBefore(&head, value, data);
                displayList(head);
                break;
            case 6:
                scanf("%d %d", &value, &data);
                insertAfter(&head, value, data);
                displayList(head);
                break;
            case 7:
                deleteBeginning(&head);
                displayList(head);
                break;
            case 8:
                deleteEnd(&head);
                displayList(head);
                break;
            case 9:
                scanf("%d", &value);
                deleteBefore(&head, value);
                displayList(head);
                break;
            case 10:
                scanf("%d", &value);
                deleteAfter(&head, value);
                displayList(head);
                break;
            case 11:
                return 0;
            default:
                printf("Invalid option! Please try again\n");
                break;
        }
    }
}
```

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

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

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

// Define the structure for a singly linked list node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
```

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

// Function to insert a node at the beginning
void insertBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
    printf("The linked list after insertion at the beginning is:\n");
}

// Function to insert a node at the end
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    printf("The linked list after insertion at the end is:\n");
}

// Function to insert before a specific value
void insertBefore(struct Node** head, int value, int data) {
    if (*head == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    if ((*head)->data == value) {
        insertBeginning(head, data);
        return;
    }
```

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

    if (temp->next == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    struct Node* newNode = createNode(data);
    newNode->next = temp->next;
    temp->next = newNode;
    printf("The linked list after insertion before a value is:\n");
}

// Function to insert after a specific value
void insertAfter(struct Node** head, int value, int data) {
    struct Node* temp = *head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }

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

    struct Node* newNode = createNode(data);
    newNode->next = temp->next;
    temp->next = newNode;
    printf("The linked list after insertion after a value is:\n");
}

// Function to delete the first node
void deleteBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty\n");
        return;
    }

    struct Node* temp = *head;
```

```
    *head = (*head)->next;
    free(temp);
    printf("The linked list after deletion from the beginning is:\n");
}

// Function to delete the last node
void deleteEnd(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty\n");
        return;
    }

    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        printf("The linked list after deletion from the end is:\n");
        return;
    }

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

    free(temp->next);
    temp->next = NULL;
    printf("The linked list after deletion from the end is:\n");
}

// Function to delete a node before a specific value
void deleteBefore(struct Node** head, int value) {
    if (*head == NULL || (*head)->data == value) {
        printf("Value not found in the list\n");
        return;
    }

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

    while (temp->next != NULL && temp->next->data != value) {
        prev = temp;
        temp = temp->next;
```

```
    }

    if (temp->next == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    if (prev == NULL) {
        *head = temp->next;
    } else {
        prev->next = temp->next;
    }

    free(temp);
    printf("The linked list after deletion before a value is:\n");
}

// Function to delete a node after a specific value
void deleteAfter(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("Value not found in the list\n");
        return;
    }

    struct Node* nodeToDelete = temp->next;
    temp->next = nodeToDelete->next;
    free(nodeToDelete);
    printf("The linked list after deletion after a value is:\n");
}

// Function to display the linked list
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty\n");
        return;
    }
```

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

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

    while (1) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("LINKED LIST CREATED\n");
                while (1) {
                    scanf("%d", &data);
                    if (data == -1) break;
                    insertEnd(&head, data);
                }
                break;
            case 2:
                displayList(head);
                break;
            case 3:
                scanf("%d", &data);
                insertBeginning(&head, data);
                displayList(head);
                break;
            case 4:
                scanf("%d", &data);
                insertEnd(&head, data);
                displayList(head);
                break;
            case 5:
                scanf("%d %d", &value, &data);
                insertBefore(&head, value, data);
                displayList(head);
                break;
```

```
            case 6:
                scanf("%d %d", &value, &data);
                insertAfter(&head, value, data);
                displayList(head);
                break;
            case 7:
                deleteBeginning(&head);
                displayList(head);
                break;
            case 8:
                deleteEnd(&head);
                displayList(head);
                break;
            case 9:
                scanf("%d", &value);
                deleteBefore(&head, value);
                displayList(head);
                break;
            case 10:
                scanf("%d", &value);
                deleteAfter(&head, value);
                displayList(head);
                break;
            case 11:
                return 0;
            default:
                printf("Invalid option! Please try again\n");
                break;
        }
    }
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. Problem Statement

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

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

Input Format

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

doubly linked list.

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

Output Format

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

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

Refer to the sample output for formatting specifications.

Sample Test Case

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

Answer

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

// Structure for a node in the doubly linked list
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to insert a node at the end of the list
void insert_at_end(struct Node** head, int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;
    new_node->data = data;
    new_node->next = NULL;
    new_node->prev = NULL;
```

```
    // If the list is empty, make the new node the head
    if (*head == NULL) {
        *head = new_node;
        return;
    }

    // Traverse to the last node and insert the new node
    while (temp->next != NULL) {
        temp = temp->next;
    }

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

// Function to print the list
void print_list(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" "); // Print space only between numbers
        }
        temp = temp->next;
    }
    printf(" \n"); // Extra space at the end
}

// Function to find and print the middle element(s)
void print_middle(struct Node* head, int n) {
    struct Node* temp = head;
    int mid1, mid2;

    if (n % 2 == 0) {
        // Even number of elements
        mid1 = n / 2;
        mid2 = mid1 + 1;
        for (int i = 1; i < mid1; i++) {
            temp = temp->next;
        }
        printf("%d ", temp->data); // First middle element
        temp = temp->next; // Move to the second middle element
```

```
        printf("%d\n", temp->data); // Second middle element
    } else {
        // Odd number of elements
        mid1 = (n / 2) + 1;
        for (int i = 1; i < mid1; i++) {
            temp = temp->next;
        }
        printf("%d\n", temp->data); // Middle element
    }
}

// Main function
int main() {
    int n;
    struct Node* head = NULL;

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

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

    // Print the data entered in the list
    print_list(head);

    // Print the middle element(s)
    print_middle(head, n);

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

    return 0;
```

}

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

2. Problem Statement

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>
```

```
// Structure for a node in the doubly linked list
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to insert a node at the end of the list
void insert_at_end(struct Node** head, int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;
    new_node->data = data;
    new_node->next = NULL;
    new_node->prev = NULL;

    // If the list is empty, make the new node the head
    if (*head == NULL) {
        *head = new_node;
        return;
    }

    // Traverse to the last node and insert the new node
    while (temp->next != NULL) {
        temp = temp->next;
    }

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

// Function to rotate the doubly linked list clockwise by k positions
void rotate_clockwise(struct Node** head, int k) {
    if (*head == NULL || k == 0) {
        return;
    }

    struct Node* current = *head;
    struct Node* tail = NULL;
    int count = 1;

    // Traverse to the end of the list to find the tail
    while (current->next != NULL) {
```

```
        current = current->next;
        count++;
    }
    tail = current;

    // If k is greater than the number of nodes, adjust k
    k = k % count;

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

    // Find the new head after rotation
    current = *head;
    for (int i = 1; i < count - k; i++) {
        current = current->next;
    }

    // Update the head and tail pointers
    struct Node* new_head = current->next;
    current->next = NULL;
    new_head->prev = NULL;
    tail->next = *head;
    (*head)->prev = tail;
    *head = new_head;
}

// Function to print the list
void print_list(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" "); // Print space only between numbers
        }
        temp = temp->next;
    }
    printf("\n"); // New line at the end
}

// Main function
```

```
int main() {
    int n, k;
    struct Node* head = NULL;

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

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

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

    // Rotate the list clockwise by k positions
    rotate_clockwise(&head, k);

    // Print the rotated list
    print_list(head);

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

    return 0;
}
```

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

3. 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>

// Structure for a node in the doubly linked list
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to insert a node at the front of the list
void insert_at_front(struct Node** head, int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = data;
    new_node->next = *head;
    new_node->prev = NULL;

    // If the list is not empty, update the previous head's prev pointer
    if (*head != NULL) {
        (*head)->prev = new_node;
    }

    *head = new_node; // Update head to the new node
}

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

    // If inserting at the front
    if (position == 1) {
        insert_at_front(head, data);
        return;
    }

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

```
    // If the position is valid
    if (temp != NULL) {
        new_node->next = temp->next;
        new_node->prev = temp;
        temp->next = new_node;

        // If the new node is not inserted at the end
        if (new_node->next != NULL) {
            new_node->next->prev = new_node;
        }
    }
}

// Function to print the list
void print_list(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" "); // Print space only between numbers
        }
        temp = temp->next;
    }
    printf("\n"); // New line at the end
}

// Main function
int main() {
    int N, position, data;
    struct Node* head = NULL;

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

    // Read and insert the initial elements at the front
    for (int i = 0; i < N; i++) {
        int item;
        scanf("%d", &item);
        insert_at_front(&head, item);
    }
```

```
    // Print the original list
    print_list(head);

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

    // Insert the new value at the specified position
    insert_at_position(&head, position, data);

    // Print the updated list
    print_list(head);

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

    return 0;
}
```

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

4. Problem Statement

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>

// Structure for a node in the doubly linked list
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to insert a node at the end of the list
void insert_at_end(struct Node** head, int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = data;
    new_node->next = NULL;
    new_node->prev = NULL;
```

```
    // If the list is empty, make the new node the head
    if (*head == NULL) {
        *head = new_node;
        return;
    }

    // Traverse to the last node and insert the new node
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }

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

// Function to print the list
void print_list(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" "); // Print space only between numbers
        }
        temp = temp->next;
    }
    printf("\n"); // New line at the end
}

// Function to check if the doubly linked list is a palindrome
int is_palindrome(struct Node* head) {
    struct Node* left = head;
    struct Node* right = head;

    // Move right to the end of the list
    while (right->next != NULL) {
        right = right->next;
    }

    // Compare elements from the start and end
    while (left != NULL && right != NULL && left != right && left->prev != right) {
        if (left->data != right->data) {
```

```
            return 0; // Not a palindrome
        }
        left = left->next;
        right = right->prev;
    }

    return 1; // Is a palindrome
}

// Main function
int main() {
    int N;
    struct Node* head = NULL;

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

    // Read and insert the items into the doubly linked list
    for (int i = 0; i < N; i++) {
        int item;
        scanf("%d", &item);
        insert_at_end(&head, item);
    }

    // Print the list
    print_list(head);

    // Check if the list is a palindrome
    if (is_palindrome(head)) {
        printf("The doubly linked list is a palindrome\n");
    } else {
        printf("The doubly linked list is not a palindrome\n");
    }

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

```
    return 0;
}
```

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

5. Problem Statement

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

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

Input Format

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

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

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

Output Format

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

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

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 20 30 40 50

2

Output: 50 40 30 20 10
50 30 20 10

Answer

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

// Structure for a node in the doubly linked list
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to insert a node at the front of the list
void insert_at_front(struct Node** head, int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = data;
    new_node->next = *head;
    new_node->prev = NULL;

    // If the list is not empty, update the previous head's prev pointer
    if (*head != NULL) {
        (*head)->prev = new_node;
    }

    *head = new_node; // Update head to the new node
}

// Function to delete a node at a specific position
void delete_at_position(struct Node** head, int position) {
    if (*head == NULL || position <= 0) {
        return; // Invalid position
    }

    struct Node* temp = *head;

    // If the head needs to be removed
    if (position == 1) {
        *head = temp->next; // Change head
```

```
        if (*head != NULL) {
            (*head)->prev = NULL; // Update the new head's prev pointer
        }
        free(temp); // Free the old head
        return;
    }

    // Traverse to the node at the given position
    for (int i = 1; temp != NULL && i < position; i++) {
        temp = temp->next;
    }

    // If the position is more than the number of nodes
    if (temp == NULL) {
        return;
    }

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

    free(temp); // Free the node
}

// Function to print the list
void print_list(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" "); // Print space only between numbers
        }
        temp = temp->next;
    }
    printf("\n"); // New line at the end
}

// Main function
```

```
int main() {
    int N, X;
    struct Node* head = NULL;

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

    // Read and insert the items into the doubly linked list
    for (int i = 0; i < N; i++) {
        int item;
        scanf("%d", &item);
        insert_at_front(&head, item);
    }

    // Print the original list
    print_list(head);

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

    // Delete the node at the specified position
    delete_at_position(&head, X);

    // Print the updated list
    print_list(head);

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

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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;

void insert(int value){
   if(head==NULL){
      head=(struct node*)malloc(sizeof(struct node));
      head->data=value;
      head->next=NULL;
   }
   else{
      struct node* temp=head;
      while(temp->next != NULL){
         temp=temp->next;
```

```
        }
        temp->next=(struct node*)malloc(sizeof(struct node));
        temp->next->data=value;
        temp->next->next=NULL;
    }
}

void display_List(){
    struct node* list=head;
    while(list!=NULL){
        printf("%d",list->data);
        list=list->next;
    }
}

void deleteNode(int pos){
    int size=0;
    struct node* temp=head;

    while(temp!=NULL){
        size++;
        temp=temp->next;
    }

    if(size<pos){
        printf("Invalid position.Deletion not possible.",size);
    }
    else{
        pos-=1;
        if(pos==0){
            temp=head->next;
            free(head);
            head=temp;
        }
        else{
            temp=head;
            while(--pos){
                temp=temp->next;
            }
            struct node* temp1 =temp->next;
            temp->next=temp->next->next;
            free(temp1);
```

```
        }
        display_List();
    }
  }
```

```
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 : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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 Poly{
    int coeff;
    int expon;
    struct Poly* next;
}Node;

Node*newnode(int coeff,int expon){
    Node*new_node = (Node*)malloc (sizeof(Node));
    new_node->coeff = coeff;
    new_node->expon = expon;
    new_node->next = NULL;
    return new_node;
}

void insertNode(Node**head,int coeff,int expon){
    Node*temp = *head;
```

```
    if(temp==NULL) {
        *head = newnode(coeff,expon);
        return;
    }
    while(temp->next!=NULL){
        temp=temp->next;
    }
    temp ->next = newnode(coeff,expon);
}

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

    Node*poly1;
    Node*poly2;

    for(int i=0;i<n;i++){
        scanf("%d %d ", &coeff, &expon);
        insertNode(&poly1,coeff,expon);
    }

    scanf("%d", &n);

    for (int i=0;i<n;i++){
        scanf("%d %d", &coeff, &expon);
        insertNode(&poly2,coeff,expon);
    }

    int sum=0;
    while(poly1!=NULL){
        sum+=poly1->coeff;
        poly1=poly1->next;
    }
    while(poly2!=NULL){
        sum+=poly2->coeff;
        poly2 = poly2->next;
    }
    printf("%d",sum);
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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;
};

struct Node *head=NULL,*newnode,*ptr;
void insertAtFront(struct Node **head,int a)
{newnode=(struct Node*)malloc(sizeof(struct Node));
newnode->data=a;
newnode->next=NULL;
newnode->next=*head;
   *head=newnode;

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


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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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

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

typedef struct Char{
   char value;
   struct Char*next;
}Node;

Node*newnode(char value){
   Node*new_node = (Node*)malloc(sizeof(Node));
   new_node->value=value;
```

```
    new_node->next=NULL;
    return new_node;
}

void insertNode( Node**head,char value) {
    Node*temp=*head;
    if(temp == NULL){
        *head = newnode(value);
        return;
    }
    while(temp->next!=NULL){
        temp = temp->next;
    }
    temp->next = newnode(value);
}

int length(Node*head){
    int len =0;
    while(head !=NULL){
        head=head->next;
        len++;
    }
    return len;
}

void traverse(Node*head){
    while(head!=NULL){
        printf("%c",head->value);
        head=head->next;
    }
    printf("\n");
}

void insert(Node**head,int pos,char value){
    if(pos >= length(*head)){
        printf("Invalid index\n");
        return;
    }
    Node*temp=*head;

    for(int i=0;i<pos;i++){
        temp = temp->next;
```

```
    }
    Node*new_node = newnode(value);
    new_node->next = temp->next;
    temp->next = new_node;
}

int main(){
    int n;
    char value;
    Node*head = NULL;
    scanf("%d",& n);

    for(int i=0;i<=n;i++){
        scanf("%c ", &value);
        if(value == ' '|| value == '\n'){
            continue;
        }
        insertNode(&head, value);
    }
    scanf("%d %c", &n, &value);
    insert(&head, n, value);
    printf("Updated list:");
    traverse(head);
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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

```
#include<stdio.h>
#include<stdlib.h>
typedef struct gpa
{
   float value;
   struct gpa* next;
}Node;
Node* newnode(float value)
{
   Node* newgpa = (Node*) malloc(sizeof(Node));
   newgpa->value = value;
   newgpa->next = NULL;
   return newgpa;
}
Node* insertAtStart(Node* head, float value)
{
   Node* newgpa = newnode(value);
   newgpa->next = head;
   return newgpa;
}
```

```
void traverse(Node* head)
{
   while(head != NULL)
   {
      printf("GPA: %.1f\n",head->value);
      head = head->next;
   }
}
void deleteAtPosition(Node** head, int pos)
{
   pos -= 1;
   Node* temp = *head;
   if(pos == 0)
   {
      *head = temp->next;
      free(temp);
      return;
   }
   while(--pos)
   {
      temp = temp->next;
   }
   Node* temp1 = temp->next;
   temp->next = temp->next->next;
   free(temp1);
}
int main()
{
   int n,pos;
   float value;
   scanf("%d",&n);
   Node* head = NULL;
   for(int i=0; i<n; i++)
   {
      scanf("%f",&value);
      head = insertAtStart(head, value);
   }
   scanf("%d",&pos);
   deleteAtPosition(&head, pos);
   traverse(head);
}
```

Status : Correct

Marks : 10/10

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 7

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};

// You are using GCC
struct Node*push(Node*head,int value)
{
    Node*newnode=(struct Node*)malloc(sizeof(struct Node));
    newnode->next = head;
    newnode->data= value;
    return newnode;
}
int printMiddle(struct Node*head){
    int len =0;
    Node*temp=head;
```

```
    while (temp!=NULL){
        len++;
        temp=temp->next;

    }
    int pos = len/2;
    for(int i=0;i<pos;i++)
    {
        head = head->next ;
    }
    return head->data;
}

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

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

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

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


    int middle_element = printMiddle(head);
    printf("Middle Element: %d\n", middle_element);


    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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 6

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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

```
#include<stdio.h>
#include<stdlib.h>
typedef struct student
{
   int roll;
   struct student* next;
}Node;
Node* newnode(int rollno)
{
   Node* data = (Node*) malloc(sizeof(Node));
   data->roll = rollno;
   data->next = NULL;
   return data;
}
void traverse(Node* head)
{
   while(head != NULL)
   {
      printf("%d ",head->roll);
      head = head->next;
   }
}
int main()
{
   int n,rollno;
   scanf("%d",&n);
   scanf("%d",&rollno);
   Node* head = newnode(rollno);
```

```
    Node* temp = head;
    while(--n)
    {
        scanf("%d",&rollno);
        temp->next = newnode(rollno);
        temp = temp->next;
    }
    traverse(head);
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

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

Here are the main functionalities of the program:

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

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

Input Format

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

The input is terminated by entering - (hyphen).

Output Format

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

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

Refer to the sample output for formatting specifications.

Sample Test Case

Input: a b c -

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

Answer

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

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

// You are using GCC
struct Node* tail;
void insertAtEnd(struct Node** head, char item) {
   //type your code here
   struct Node* newnode=(struct Node*)malloc(sizeof(struct Node));
   newnode->item=item;
   newnode->next=NULL;
   newnode->prev=NULL;
   if(*head==NULL){
      *head=tail=newnode;
```

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

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

void freePlaylist(struct Node* head) {
   //type your code here
   struct Node* temp=head;
   while(temp!=NULL){
      Node* nextnode=temp->next;
      free(temp);
      temp=nextnode;
   }
   head=NULL;
   tail=NULL;
   }

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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

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

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

Input Format

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

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

Output Format

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

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

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3
163 137 155
Output: 163

Answer

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

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

}node;
node* tail=NULL;
void insert(node** head,int value){
    node* newnode=(node*)malloc(sizeof(node));
    newnode->data=value;
    newnode->prev=NULL;
    newnode->next=NULL;
    if(* head==NULL){
        *head=tail=newnode;
        return;
    }
    else{
        tail->next=newnode;
        newnode->prev=tail;
        tail=newnode;
     }
```

```
}
void print(node* head){

  if(head==NULL){
     printf("Empty list!\n");
     return;
  }
  node* temp=head;
  int max=temp->data;
  while(temp!=NULL){
     if(temp->data>max){
        max=temp->data;
     }
       temp=temp->next;
     }

   printf("%d\n",max);

}
int main(){
    node* head=NULL;
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
       int val;
       scanf("%d",&val);
       insert(&head,val);
  }
  print(head);
  return 0;
}
```

Status : Correct

Marks : 10/10

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

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

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

Input Format

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

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

Output Format

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

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
101 102 103 104
Output: Node Inserted
101
Node Inserted
102 101
Node Inserted
103 102 101
Node Inserted
104 103 102 101

Answer

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

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

struct node* start = NULL;

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

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

void insertAtFront(int data) {
    //type your code here
    struct node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->info=data;
    newnode->prev=NULL;
    newnode->next=start;
    start=newnode;

}


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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

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

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

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

Input Format

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

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

Output Format

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

Refer to the sample output for formatting specifications.

Sample Test Case

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

Answer

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

typedef struct node{
   int data;
   struct node* prev,*next;
}node;
node* tail=NULL;
void insert(node** head,int value){
   node* newnode=(node*)malloc(sizeof(node));
   newnode->data=value;
   newnode->prev=NULL;
   newnode->next=NULL;

   if(*head==NULL){
      *head=tail=newnode;
      return;
   }
   else{
      tail->next=newnode;
      newnode->prev=tail;
      tail=newnode;
   }
```

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

int main(){
    node* head=NULL;
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        int val;
        scanf("%d",&val);
        insert(&head,val);
    }
    display(head);
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 5

Attempt : 2
Total Mark : 10
Marks Obtained : 0

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;

Node* head = NULL;

void append(int data) {
    Node* new_node = (Node*)malloc(sizeof(Node));
    new_node->data = data;
    new_node->next = NULL;

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

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

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

void display() {
    Node* temp = head;
    int pos = 1;
    while (temp) {
        printf(" node %d : %d\n", pos++, temp->data);
        temp = temp->next;
    }
}

void deleteNode(int position, int n) {
    if (position < 1 || position > n) {
        printf("Invalid position. Try again.\n");
        return;
```

```
    }

    Node* temp = head;

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

    if (temp->prev) {
        temp->prev->next = temp->next;
    } else {
        head = temp->next;
    }

    if (temp->next) {
        temp->next->prev = temp->prev;
    }

    free(temp);

    printf("\nAfter deletion the new list:\n");
    display();
}

int main() {
    int n, p;

    scanf("%d", &n);
    int elements[n];

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

    scanf("%d", &p);

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

    deleteNode(p, n);
```

```
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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;
}
```

```
void enqueue(int ticketID) {
    if ((rear + 1) % MAX_SIZE == front) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }
    if (front == -1) { // first element
        front = 0;
        rear = 0;
    } else {
        rear = (rear + 1) % MAX_SIZE;
    }
    ticketIDs[rear] = ticketID;
    printf("Helpdesk Ticket ID %d is enqueued.\n", ticketID);
}

int dequeue() {
    if (front == -1) {
        return 0; // queue empty
    }
    lastDequeued = ticketIDs[front];
    if (front == rear) { // single element
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % MAX_SIZE;
    }
    return 1;
}

void display() {
    if (front == -1) {
        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;
        printf(" ");
        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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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;
}
```

```
int isFull() {
    return rear == MAX_SIZE - 1;
}

int isEmpty() {
    return front == -1 || front > rear;
}

int enqueue(char order) {
    if (isFull()) {
        printf("Queue is full. Cannot enqueue more orders.\n");
        return 0;
    }
    if (front == -1) {
        front = 0;
    }
    rear++;
    orders[rear] = order;
    printf("Order for %c is enqueued.\n", order);
    return 1;
}

void dequeue() {
    if (isEmpty()) {
        printf("No orders in the queue.\n");
        return;
    }
    printf("Dequeued Order: %c\n", orders[front]);
    front++;
    if (front > rear) {
        front = -1;
        rear = -1;
    }
}

void display() {
    if (isEmpty()) {
        printf("Queue is empty. No orders available.\n");
        return;
    }
    printf("Orders in the queue are: ");
```

```
    for (int i = front; i <= rear; i++) {
        printf("%c ", orders[i]);
    }
    printf("\n");
}

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 : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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;

int insertq(int *data) {
    if (rear == max - 1) {
        return 0; // Queue is full
    }
    if (front == -1) {
        front = 0;
    }
    rear++;
    queue[rear] = *data;
    return 1; // Successful insert
}

void delq() {
    if (front == -1 || front > rear) {
        printf("Queue is empty.\n");
    } else {
        printf("Deleted number is: %d\n", queue[front]);
        front++;
        if (front > rear) {
            front = rear = -1; // Reset queue when empty
        }
    }
}

void display() {
    if (front == -1 || front > rear) {
        printf("Queue is empty.\n");
```

```
    } else {
        printf("Elements in the queue are: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

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 : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 4

Attempt : 2
Total Mark : 10
Marks Obtained : 9

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>
#define MAX_SIZE 5

int queue[MAX_SIZE];
int front = 0, rear = 0;
```

```
void enqueue(int page) {
    if (rear == MAX_SIZE) {
        printf("Queue is full. Cannot enqueue. ");
    } else {
        queue[rear] = page;
        rear++;
        printf("Print job with %d pages is enqueued. ", page);
    }
    printf("\n");
}

void dequeue() {
    if (front == rear) {
        printf("Queue is empty. ");
    } else {
        printf("Processing print job: %d pages ", queue[front]);
        front++;
        if (front == rear) {
            // Reset the queue if empty
            front = 0;
            rear = 0;
        }
    }
}

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

int main() {
    int input[100], n = 0, value, choice;
    // Read all input as integers
    while (scanf("%d", &input[n]) != EOF) {
```

```
        n++;
        if (input[n - 1] == 4) break;  // stop reading after exit choice
    }

    for (int i = 0; i < n; i++) {
        choice = input[i];
        if (choice == 1) {
            i++;
            if (i < n) {
                value = input[i];
                enqueue(value);
            }
        } else if (choice == 2) {
            dequeue();
        } else if (choice == 3) {
            display();
        } else if (choice == 4) {
            printf("Exiting program");
            break;
        } else {
            printf("Invalid option. ");
        }
    }

    return 0;
}
```

Status : Partially correct ***Marks : 9/10***

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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;

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

```
    if (rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
}

void printFrontRear() {
    if (front == NULL) {
        printf("Front: -1, Rear: -1\n");
    } else {
        printf("Front: %d, Rear: %d\n", front->data, rear->data);
    }
}

void dequeue() {
    if (front == NULL) {
        return;
    }
    struct Node* temp = front;
    front = front->next;
    if (front == NULL) {
        rear = NULL;
    }
    free(temp);
}
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 : Correct **Marks : 10/10**

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 1

Attempt : 2
Total Mark : 10
Marks Obtained : 10

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

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

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

struct Node* top = NULL;


void push(int value) {

    struct Node* newnode=(struct Node*)malloc(sizeof(struct Node));
    printf("Pushed element: %d\n",value);
    newnode->data=value;
    newnode->next=top;
    top=newnode;
}
```

```
void pop(){

   if(top==NULL){
      printf("Stack is empty.Cannot pop.\n");
      return;
   }
   struct Node* temp=top;
   top=top->next;
   printf("Popped element: %d\n",temp->data);
   free(temp);
}

void displayStack(){

   if(top==NULL){
      printf("Stack is empty");
      return;
   }else{
   printf("Stack elements (top to bottom): \n");
   struct Node* temp=top;
   while(temp!=NULL){
      printf("%d ",temp->data);
      temp=temp->next;
   }
   }
}
int main() {
   int choice, value;
   do {
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            scanf("%d", &value);
            push(value);
            break;
         case 2:
            pop();
            break;
         case 3:
            displayStack();
            break;
```

```
            case 4:
                printf("Exiting program\n");
                return 0;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 4);

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 2

Attempt : 2
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Sanjeev is in charge of managing a library's book storage, and he wants to create a program that simplifies this task. His goal is to implement a program that simulates a stack using an array.

Help him in writing a program that provides the following functionality:

Add Book ID to the Stack (Push): You can add a book ID to the top of the book stack. Remove Book ID from the Stack (Pop): You can remove the top book ID from the stack and display its details. If the stack is empty, you cannot remove any more book IDs.Display Books ID in the Stack (Display): You can view the books ID currently on the stack.Exit the Library: You can choose to exit the program.

Input Format

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

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

Choice 2: Pop the book ID from the stack.

Choice 3: Display the book ID in the stack.

Choice 4: Exit the program.

Output Format

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

1. If the choice is 1, push the given book ID to the stack and display the corresponding message.
2. If the choice is 2, pop the book ID from the stack and display the corresponding message.
3. If the choice is 2, and if the stack is empty without any book ID, print "Stack Underflow"
4. If the choice is 3, print the book IDs in the stack.
5. If the choice is 3, and there are book IDs in the stack, print "Stack is empty"
6. If the choice is 4, exit the program and display the corresponding message.
7. If any other choice is entered, print "Invalid choice"

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

Sample Test Case

Input: 1 19
1 28
2
3
2
4
Output: Book ID 19 is pushed onto the stack
Book ID 28 is pushed onto the stack

Book ID 28 is popped from the stack
Book ID in the stack: 19
Book ID 19 is popped from the stack
Exiting the program

Answer

```
// You are using GCC

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

typedef struct node{
   int data;
   struct node*next;
}node;

node* top=NULL;

void push(int value){
   printf("Book ID %d is pushed onto the stack\n",value);
   node* newnode=(node*)malloc(sizeof(node));
   newnode->data=value;
   newnode->next=top;
   top=newnode;
}

void pop(){
   if(top==NULL){
      printf("Stack Underflow\n");
      return;
   }else{
   node* temp=top;
   top=top->next;
   printf("Book ID %d is popped from the stack\n",temp->data);
   free(temp);
   }
}

void display(){
   if(top==NULL){
      printf("Stack is empty");
      return;
```

```
    } else {
    printf("Book ID in the stack: ");
    node* temp=top;
    while(temp!=NULL){
        printf("%d ",temp->data);
        temp=temp->next;
    }
    }
    printf("\n");

}

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

    switch(choice){
        case 1:
        int val;
        scanf("%d",&val);
        push(val);
        break;
        case 2:
        pop();
        break;
        case 3:
        display();
        break;
        case 4:
        printf("Exiting the program");
        break;
        default:
        printf("Invalid choice");
    }
    }while(choice!=4);
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 3

Attempt : 2
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Sharon is developing a programming challenge for a coding competition. The challenge revolves around implementing a character-based stack data structure using an array.

Sharon's project involves a stack that can perform the following operations:

Push a Character: Users can push a character onto the stack.Pop a Character: Users can pop a character from the stack, removing and displaying the top character.Display Stack: Users can view the current elements in the stack.Exit: Users can exit the stack operations application.

Write a program to help Sharon to implement a program that performs the given operations.

Input Format

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

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

Choice 2: Pop the character from the stack.

Choice 3: Display the characters in the stack.

Choice 4: Exit the program.

Output Format

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

1. If the choice is 1, push the given character to the stack and display the pushed character having the prefix "Pushed: ".
2. If the choice is 2, undo the character from the stack and display the character that is popped having the prefix "Popped: ".
3. If the choice is 2, and if the stack is empty without any characters, print "Stack is empty. Nothing to pop."
4. If the choice is 3, print the elements in the stack having the prefix "Stack elements: ".
5. If the choice is 3, and there are no characters in the stack, print "Stack is empty."
6. If the choice is 4, exit the program.
7. If any other choice is entered, print "Invalid choice"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 2
4

Output: Stack is empty. Nothing to pop.

Answer

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

```
#include <stdbool.h>

#define MAX_SIZE 100

char items[MAX_SIZE];
int top = -1;

void initialize() {
    top = -1;
}
bool isFull() {
    return top == MAX_SIZE - 1;
}

bool isEmpty() {
    return top == -1;
}

// You are using GCC
void push(char value) {
    if(isFull()){
        printf("Stack is full.Cannot push.\n");
    }else{
        items[++top]=value;
        printf("Pushed: %c\n",value);
    }

}

char pop() {
    if(isEmpty()){
        printf("Stack is empty.Nothing to pop.\n");
        return '\0';
    }else{
        char poppeditem=items[top--];
        printf("Popped: %c\n",poppeditem);
        return poppeditem;
    }

}
void display() {
    if(isEmpty()){
        printf("Stack is empty.\n");
```

```
    }else{
        printf("Stack elements: ");
        for(int i=top;i>=0;i--){
            printf("%c ",items[i]);
        }
        printf("\n");
    }

}
int main() {
    initialize();
    int choice;
    char value;

    while (true) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                scanf(" %c", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 4

Attempt : 2
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

You are a software developer tasked with building a module for a scientific calculator application. The primary function of this module is to convert infix mathematical expressions, which are easier for users to read and write, into postfix notation (also known as Reverse Polish Notation). Postfix notation is more straightforward for the application to evaluate because it removes the need for parentheses and operator precedence rules.

The scientific calculator needs to handle various mathematical expressions with different operators and ensure the conversion is correct. Your task is to implement this infix-to-postfix conversion algorithm using a stack-based approach.

Example

Input:

a+b

Output:

ab+

Explanation:

The postfix representation of (a+b) is ab+.

Input Format

The input is a string, representing the infix expression.

Output Format

The output displays the postfix representation of the given infix expression.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: a+(b*e)
Output: abe*+

Answer

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

struct Stack {
    int top;
    unsigned capacity;
    char* array;
};

struct Stack* createStack(unsigned capacity) {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));

    if (!stack)
```

```
    return NULL;

  stack->top = -1;
  stack->capacity = capacity;
  stack->array = (char*)malloc(stack->capacity * sizeof(char));

  return stack;
}

int isEmpty(struct Stack* stack) {
  return stack->top == -1;
}

char peek(struct Stack* stack) {
  return stack->array[stack->top];
}

char pop(struct Stack* stack) {
  if (!isEmpty(stack))
    return stack->array[stack->top--];
  return '$';
}

void push(struct Stack* stack, char op) {
  stack->array[++stack->top] = op;
}

// You are using GCC
int isOperand(char ch) {
  return(ch>='a'&& ch<='z')||(ch>='A'&& ch<='Z')||(ch>='0' && ch<='9');

}

int Prec(char ch) {
  switch(ch){
    case '+':
    case '-': return 1;
    case '*':
    case '/': return 2;
    case '^':return 3;
    default: return -1;
  }
```

```
}
int isRight(char ch){
    return ch=='^';
}
void infixToPostfix(char*exp){
    int i,k;
    int len=strlen(exp);
    char*result=(char*)malloc((len+1)*sizeof(char));
    struct Stack*stack=createStack(len);
    if(!stack) return;

    for(i=0,k=0;exp[i];i++){
        if(isOperand(exp[i])){
        result[k++]=exp[i];
        }
        else if(exp[i]=='('){
        push(stack,exp[i]);
        }
        else if(exp[i]==')'){
            while(!isEmpty(stack)&&peek(stack)!='('
                result[k++]=pop(stack);
            if(!isEmpty(stack)&&peek(stack)!='('){
                free(stack->array);
                free(stack);
                free(result);
                return;
            }
            else{
            pop(stack);
            }
        }else{
            while(!isEmpty(stack)&&((Prec(exp[i])<Prec(peek(stack)))||
(Prec(exp[i])==Prec(peek(stack))&& !isRight(exp[i])))){
                result[k++]=pop(stack);

            }
            push(stack,exp[i]);
        }
    }
    while(!isEmpty(stack))
    result[k++]=pop(stack);
    result[k]='\0';
```

```
    printf("%s\n",result);
}

int main() {
    char exp[100];
    scanf("%s", exp);

    infixToPostfix(exp);
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 5

Attempt : 2
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Milton is a diligent clerk at a school who has been assigned the task of managing class schedules. The school has various sections, and Milton needs to keep track of the class schedules for each section using a stack-based system.

He uses a program that allows him to push, pop, and display class schedules for each section. Milton's program uses a stack data structure, and each class schedule is represented as a character. Help him write a program using a linked list.

Input Format

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

Choice 1: Push the character onto the stack. If the choice is 1, the following input is a space-separated character, representing the class schedule to be pushed onto the stack.

Choice 2: Pop class schedule from the stack

Choice 3: Display the class schedules 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 class schedule to the stack and display the following: "Adding Section: [class schedule]"
- If the choice is 2, pop the class schedule from the stack and display the following: "Removing Section: [class schedule]"
- If the choice is 2, and if the stack is empty without any class schedules, print "Stack is empty. Cannot pop."
- If the choice is 3, print the class schedules in the stack in the following: "Enrolled Sections: " followed by the class schedules separated by space.
- If the choice is 3, and there are no class schedules in the stack, print "Stack is empty"
- If the choice is 4, exit the program and display the following: "Exiting the program"
- If any other choice is entered, print "Invalid choice"

Refer to the sample output for the exact format.

Sample Test Case

Input: 1 d
1 h
3
2

3
4
Output: Adding Section: d
Adding Section: h
Enrolled Sections: h d
Removing Section: h
Enrolled Sections: d
Exiting program

Answer

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

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

struct Node* top = NULL;


void push(char value) {

    printf("Adding section: %c\n",value);
    struct Node* newnode = (struct Node*)malloc(sizeof(struct Node));
    newnode->data=value;
    newnode->next = top;
    top = newnode;
}

void pop() {

    if (top == NULL) {
        printf("Stack is empty. Cannot pop.\n");
        return;
    }else{
    struct Node* temp = top;
    top = top->next;
    printf("Removing section: %c\n", temp->data);
    free(temp);
    }
```

```
}
void displayStack() {

    if (top == NULL) {
        printf("Stack is empty\n");
        return;
    }
    else {
        printf("Enrolled sections: ");
        struct Node* temp = top;
        while (temp != NULL) {
            printf("%c ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}



int main() {
    int choice;
    char value;
    do {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                scanf(" %c", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                printf("Exiting program\n");
                break;
            default:
                printf("Invalid choice\n");
```

```
    }
  } while (choice != 4);

  return 0;
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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;

int insertq(int *data) {
    if (rear == max - 1) {
        return 0; // Queue is full
    }
    if (front == -1) {
        front = 0;
    }
    rear++;
    queue[rear] = *data;
    return 1; // Successful insert
}

void delq() {
    if (front == -1 || front > rear) {
        printf("Queue is empty.\n");
    } else {
        printf("Deleted number is: %d\n", queue[front]);
        front++;
        if (front > rear) {
            front = rear = -1; // Reset queue when empty
        }
    }
}

void display() {
    if (front == -1 || front > rear) {
        printf("Queue is empty.\n");
```

```
    } else {
        printf("Elements in the queue are: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

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 : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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;
}
```

```
void enqueue(int ticketID) {
    if ((rear + 1) % MAX_SIZE == front) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }
    if (front == -1) { // first element
        front = 0;
        rear = 0;
    } else {
        rear = (rear + 1) % MAX_SIZE;
    }
    ticketIDs[rear] = ticketID;
    printf("Helpdesk Ticket ID %d is enqueued.\n", ticketID);
}

int dequeue() {
    if (front == -1) {
        return 0; // queue empty
    }
    lastDequeued = ticketIDs[front];
    if (front == rear) { // single element
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % MAX_SIZE;
    }
    return 1;
}

void display() {
    if (front == -1) {
        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;
        printf(" ");
        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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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;
}
```

```
int isFull() {
    return rear == MAX_SIZE - 1;
}

int isEmpty() {
    return front == -1 || front > rear;
}

int enqueue(char order) {
    if (isFull()) {
        printf("Queue is full. Cannot enqueue more orders.\n");
        return 0;
    }
    if (front == -1) {
        front = 0;
    }
    rear++;
    orders[rear] = order;
    printf("Order for %c is enqueued.\n", order);
    return 1;
}

void dequeue() {
    if (isEmpty()) {
        printf("No orders in the queue.\n");
        return;
    }
    printf("Dequeued Order: %c\n", orders[front]);
    front++;
    if (front > rear) {
        front = -1;
        rear = -1;
    }
}

void display() {
    if (isEmpty()) {
        printf("Queue is empty. No orders available.\n");
        return;
    }
    printf("Orders in the queue are: ");
```

```
    for (int i = front; i <= rear; i++) {
        printf("%c ", orders[i]);
    }
    printf("\n");
}

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 : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 4

Attempt : 2
Total Mark : 10
Marks Obtained : 9

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>
#define MAX_SIZE 5

int queue[MAX_SIZE];
int front = 0, rear = 0;
```

```
void enqueue(int page) {
    if (rear == MAX_SIZE) {
        printf("Queue is full. Cannot enqueue. ");
    } else {
        queue[rear] = page;
        rear++;
        printf("Print job with %d pages is enqueued. ", page);
    }
    printf("\n");
}

void dequeue() {
    if (front == rear) {
        printf("Queue is empty. ");
    } else {
        printf("Processing print job: %d pages ", queue[front]);
        front++;
        if (front == rear) {
            // Reset the queue if empty
            front = 0;
            rear = 0;
        }
    }
}

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

int main() {
    int input[100], n = 0, value, choice;
    // Read all input as integers
    while (scanf("%d", &input[n]) != EOF) {
```

```
        n++;
        if (input[n - 1] == 4) break;  // stop reading after exit choice
    }

    for (int i = 0; i < n; i++) {
        choice = input[i];
        if (choice == 1) {
            i++;
            if (i < n) {
                value = input[i];
                enqueue(value);
            }
        } else if (choice == 2) {
            dequeue();
        } else if (choice == 3) {
            display();
        } else if (choice == 4) {
            printf("Exiting program");
            break;
        } else {
            printf("Invalid option. ");
        }
    }

    return 0;
}
```

Status : Partially correct ***Marks : 9/10***

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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;

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

```
    if (rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
}

void printFrontRear() {
    if (front == NULL) {
        printf("Front: -1, Rear: -1\n");
    } else {
        printf("Front: %d, Rear: %d\n", front->data, rear->data);
    }
}

void dequeue() {
    if (front == NULL) {
        return;
    }
    struct Node* temp = front;
    front = front->next;
    if (front == NULL) {
        rear = NULL;
    }
    free(temp);
}
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 : Correct **Marks : 10/10**

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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) {
        struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
```

```
        newNode->data = key;
        newNode->left = newNode->right = NULL;
        return newNode;
    }
    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 && 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)
        return;
    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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = value;
        newNode->left = NULL;
        newNode->right = NULL;
        return newNode;
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else {
        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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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

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

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

    return root;
}

// Function to search for a value in the BST
struct Node* searchNode(struct Node* root, int key) {
    if (root == NULL || root->data == key) {
        return root;
    }
```

```
    if (key < root->data) {
        return searchNode(root->left, key);
    } else {
        return searchNode(root->right, key);
    }
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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 {
        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 -1;
    }
    struct Node* current = root;
    while (current->left != NULL) {
        current = current->left;
    }
    return current->data;
}

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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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 {
        root->right = insert(root->right, key);
    }
    return root;
}

// Function to find the maximum value in the BST
```

```
int findMax(struct TreeNode* root) {
    if (root == NULL) {
        return -1; // Return -1 if the tree is empty
    }
    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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 6_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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 left[], int right[], int left_size, int right_size) {
    int i = 0, j = 0, k = 0;
    while (i < left_size && j < right_size) {
        if (left[i] < right[j])
            arr[k++] = left[i++];
        else
            arr[k++] = right[j++];
    }
    while (i < left_size)
        arr[k++] = left[i++];
    while (j < right_size)
        arr[k++] = right[j++];
}
void mergeSort(int arr[], int size) {
    if (size < 2)
        return;
    int mid = size / 2;
    int left[mid], right[size - mid];
    for (int i = 0; i < mid; i++)
```

```
        left[i] = arr[i];
    for (int i = mid; i < size; i++)
        right[i - mid] = arr[i];

    mergeSort(left, mid);
    mergeSort(right, size - mid);
    merge(arr, left, right, mid, size - mid);
}



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 : Correct **Marks : 10/10**

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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>

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

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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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>


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 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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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>


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) { // Descending order
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    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[k - 1]);
}

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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 6_COD_Question 5

Attempt : 2
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>


// Compare two double values
int compare(double a, double b) {
    return (a < b);
}

// Merge two subarrays
void merge(double arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    double L[n1], R[n2];

    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;

    // Merge the temp arrays back into arr[]
    while (i < n1 && j < n2) {
        if (compare(L[i], R[j])) {
            arr[k++] = L[i++];
```

```
        } else {
            arr[k++] = R[j++];
        }
    }

    // Copy remaining elements
    while (i < n1) {
        arr[k++] = L[i++];
    }
    while (j < n2) {
        arr[k++] = R[j++];
    }
}

// MergeSort algorithm
void mergeSort(double 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;
    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: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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

```
#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;
    int original_index = index;
    while (table[index] != -1) {
        index = (index + 1) % size;
        if (index == original_index) {
            return -1;
```

```
        }
    }
    return index;
}

void insertIntoHashTable(int table[], int size, int arr[], int n) {
    initializeTable(table, size);
    for (int i = 0; i < n; i++) {
        int index = linearProbe(table, size, arr[i]);
        if (index != -1) {
            table[index] = arr[i];
        }
    }
}

void printTable(int table[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d", table[i]);
        if (i != size - 1) {
            printf(" ");
        }
    }
    printf("\n");
}

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

    int arr[MAX];
    int table[MAX];

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

    initializeTable(table, table_size);
    insertIntoHashTable(table, table_size, arr, n);
    printTable(table, table_size);

    return 0;
}
```

Status : Correct

Marks : 10/10

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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;
   int original_index = index;
```

```
    while (table[index] != -1) {
        index = (index + 1) % size;
        if (index == original_index) {
            return -1;
        }
    }
    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]);
        if (index != -1) {
            table[index] = arr[i];
        }
    }
}

int searchInHashTable(int table[], int size, int num) {
    int index = num % size;
    int original_index = index;
    while (table[index] != -1) {
        if (table[index] == num) {
            return 1;
        }
        index = (index + 1) % size;
        if (index == original_index) {
            break;
        }
    }
    return 0;
}


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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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

```
void insertKeyValuePair(Dictionary *dict, const char *key, const char *value) {
   if (dict->size == dict->capacity) {
      dict->capacity *= 2;
      dict->pairs = (KeyValuePair *)realloc(dict->pairs, dict->capacity *
sizeof(KeyValuePair));
   }
```

```
    strcpy(dict->pairs[dict->size].key, key);
    strcpy(dict->pairs[dict->size].value, value);
    dict->size++;
}

void removeKeyValuePair(Dictionary *dict, const char *key) {
    for (int i = 0; i < dict->size; i++) {
        if (strcmp(dict->pairs[i].key, key) == 0) {
            for (int j = i; j < dict->size - 1; j++) {
                dict->pairs[j] = dict->pairs[j + 1];
            }
            dict->size--;
            return;
        }
    }
}

int doesKeyExist(Dictionary *dict, const char *key) {
    for (int i = 0; i < dict->size; i++) {
        if (strcmp(dict->pairs[i].key, key) == 0) {
            return 1;
        }
    }
    return 0;
}

void printDictionary(Dictionary *dict) {
    for (int i = 0; i < dict->size; i++) {
        printf("Key: %s; Value: %s\n", dict->pairs[i].key, dict->pairs[i].value);
    }
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 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

```
int keyExists(KeyValuePair* dictionary, int size, const char* key) {
    for (int i = 0; i < size; i++) {
        if (strcmp(dictionary[i].key, key) == 0) {
            return 1;
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: vasugi e.v.n
Email: 240801370@rajalakshmi.edu.in
Roll no: 240801370
Phone: 7708989508
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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

typedef struct {
    int key;
    int count;
} HashEntry;

// Function to find the element with an odd occurrence
int getOddOccurrence(int arr[], int n) {
    HashEntry hashTable[MAX_SIZE];

    // Initialize hash table
    for (int i = 0; i < MAX_SIZE; i++) {
        hashTable[i].key = -1;
        hashTable[i].count = 0;
    }

    // Insert each number using mid-square hash: (key * key) % 100
    for (int i = 0; i < n; i++) {
        int key = arr[i];
        int hashIndex = (key * key) % MAX_SIZE;
```

```
        // Handle collisions with linear probing
        while (hashTable[hashIndex].key != -1 && hashTable[hashIndex].key != key) {
            hashIndex = (hashIndex + 1) % MAX_SIZE;
        }

        if (hashTable[hashIndex].key == -1) {
            hashTable[hashIndex].key = key;
            hashTable[hashIndex].count = 1;
        } else {
            hashTable[hashIndex].count++;
        }
    }

    // Find and return the key that appears an odd number of times
    for (int i = 0; i < MAX_SIZE; i++) {
        if (hashTable[i].key != -1 && hashTable[i].count % 2 != 0) {
            return hashTable[i].key;
        }
    }

    return -1; // If no element occurs an odd number of times
}

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 : Correct **Marks : 10/10**