	Exp. Name: ***Operations on Array***	**Date:**

Aim:

Write a menu driven program to insert at specified position, delete at specified position and display element(s) in one dimensional array.

Input and Output Format

```
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice : 1
Enter the size of the array elements : 3
Enter the elements for the array : 1 2 3
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice : 2
The array elements are : 1 2 3
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice : 3
Enter the position(index) for the new element : 5
Enter the element to be inserted : 4
Invalid position.
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice : 3
Enter the position(index) for the new element : 2
Enter the element to be inserted : 5
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice : 2
The array elements are : 1 2 5 3
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice : 4
Enter the position(index) of the element to be deleted : 5
Invalid position.
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice : 4
```

```
Enter the position(index) of the element to be deleted : 2
The deleted element is : 5
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice : 2
The array elements are : 1 2 3
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice : 5
```

Source Code:

arrayoperations.c

```
#include<stdio.h>
#include<stdlib.h>
#define max 100
int s[max];
int size = 0,i;
void create();
void display();
void insert();
void del();
// write the required code..

int main() {
        int choice;
        do{
                printf("1.Create\n");
                printf("2.Display\n");
                printf("3.Insert\n");
                printf("4.Delete\n");
                printf("5.Exit\n");
                printf("Enter your choice : ");
                scanf("%d",&choice);
                switch(choice) {
                        case 1:
                                create();
                                break;
                        case 2:
                                display();
                                break;
                        case 3:
                                insert();
                                break;
                        case 4:
                                del();
                                break;
                        case 5:
                                exit(0);
                                break;
                        default:
                                printf("Invalid choice. Enter valid
input.\n");
                                break;
                }
        }while(choice!=5);
        return 0;
}
```

```
void create() {
        printf("Enter the size of the array elements : ");
scanf("%d",&size);
        printf("Enter the elements for the array : ");
        for (i =0;i < size;i++){
                scanf("%d",&s[i]);
        }
}

void display() {

        printf("The array elements are : ");
for (i =0;i < size;i++){
        printf("%d ",s[i]);

}printf("\n");
}

void insert() {
        int pos, value;
        printf("Enter the position(index) for the new element : ");
        scanf("%d",&pos);
        pos = pos +1;
        printf("Enter the element to be inserted : ");
        scanf("%d",&value);
        if (pos < 1 || pos > size +1){
                printf("Invalid position.\n");
                return;
        }
        for(i = size; i >= pos; i--){
                s[i] = s[i - 1];
        }
        s[pos - 1] = value;
        size++;
}
void del() {
        int pos;
        printf("Enter the position(index) of the element to be deleted
: ");
        scanf("%d",&pos);
        pos = pos +1;
        if (pos < 1 || pos > size){
                printf("Invalid position.\n");
                return;
        }
        printf("The deleted element is : %d\n",s[pos-1]);
```

```
        for(i =pos -1;i<size -1; i++)
                {
                        s[i]=s[i+1];
                }
              size--;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
1
Enter the size of the array elements :
5
Enter the elements for the array :
1 2 3 4 5
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
2
The array elements are : 1 2 3 4 5
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
3
Enter the position(index) for the new element :

Enter the element to be inserted :
8
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
2
The array elements are : 1 2 3 8 4 5
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
4
Enter the position(index) of the element to be deleted :
3
The deleted element is : 8
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
5

Test Case - 2
User Output
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
1
Enter the size of the array elements :
5

```
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
2
The array elements are : 2 3 4 1 5
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
3
Enter the position(index) for the new element :
4
Enter the element to be inserted :
23
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
2
The array elements are : 2 3 4 1 23 5
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
4
Enter the position(index) of the element to be deleted :
1
The deleted element is : 3
1.Create
2.Display
3.Insert
4.Delete
```

Enter your choice :
2
The array elements are : 2 4 1 23 5
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
2
The array elements are : 2 4 1 23 5
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
3
Enter the position(index) for the new element :
2
Enter the element to be inserted :
56
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
2
The array elements are : 2 4 56 1 23 5
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
4
Enter the position(index) of the element to be deleted :
3
The deleted element is : 1
1.Create

3.Insert
4.Delete
5.Exit
Enter your choice :
5

Test Case - 3
User Output
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
1
Enter the size of the array elements :
3
Enter the elements for the array :
1 2 3
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
2
The array elements are : 1 2 3
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
3
Enter the position(index) for the new element :
5
Enter the element to be inserted :
4
Invalid position.

3.Insert
4.Delete
5.Exit
Enter your choice :
3
Enter the position(index) for the new element :
2
Enter the element to be inserted :
5
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
2
The array elements are : 1 2 5 3
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
4
Enter the position(index) of the element to be deleted :
5
Invalid position.
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
4
Enter the position(index) of the element to be deleted :
2
The deleted element is : 5
1.Create
2.Display
3.Insert
4.Delete

Enter your choice :
2
The array elements are : 1 2 3
1.Create
2.Display
3.Insert
4.Delete
5.Exit
Enter your choice :
5

	Exp. Name: ***Maximum and Minimum Elements***	**Date:**

Aim:

Write a C program to read n integers into an array, and find and display the maximum and minimum values among them.

Input Format:

- The first line contains a positive integer n representing the number of elements in the array.
- The second line contains n space-separated integers representing the elements of the array.

Output Format:

- On the first line, print an integer representing the maximum element in the format: "Maximum element is <value>"
- On the second line, print an integer representing the minimum element in the format: "Minimum element is <value>"

Source Code:

MaximumandMinimumElements.c

```
#include <stdio.h>
int main () {
        int n, i;
        scanf("%d", &n);
        int arr[n];
        for (i = 0; i < n; i++){
                scanf("%d", &arr[i]);
        }
        int max = arr[0];
        int min = arr[0];
        for (i = 1; i < n; i++) {
                if (arr[i] > max) {
                        max = arr[i];
                }
                if (arr[i] < min) {
                        min = arr[i];
                }
        }
        printf("Maximum element is %d\n",max);
        printf("Minimum element is %d\n", min);
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
5
52 63 45 12 90
Maximum element is 90
Minimum element is 12

Test Case - 2
User Output
4
1 1 1 1
Maximum element is 1
Minimum element is 1

	Exp. Name: ***Array Reverse***	**Date:**

Aim:

Write a C program to reverse all the elements in the array.

Input Format:

- The first line of input contains an integer N representing the size of the array.
- The second line of input contains N no.of space-separated integers representing the array elements.

Output Format:

- Print the elements of the array in reverse order.

Constraints:

- $1 \leq N \leq 1000$
- $0 \leq arr[i] \leq 1000$

Source Code:

ArrayReverse.c

```c
#include <stdio.h>
void reverseArray(int arr[], int size) {
        int start = 0;
        int end = size - 1;
        while (start < end) {
                // Swap elements at start and end indices
                int temp = arr[start];
                arr[start] = arr[end];
                arr[end] = temp;
                // Move indices towards the center
                start++;
                end--;
        }
}
int main() {
        int N;
        scanf("%d", &N);
        int arr[N];
        for (int i = 0; i < N; i++) {
                scanf("%d", &arr[i]);
        }
        // Reverse the array
        reverseArray(arr, N);
        // print the reversed array
        for (int i = 0; i < N; i++) {
                printf("%d ", arr[i]);
        }
        printf("\n");
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
3
15 24 62
62 24 15

Test Case - 2

User Output
4
-54 63 -21 51
51 -21 63 -54

Test Case - 3
User Output
5
-50 -60 -70 100 80
80 100 -70 -60 -50

Test Case - 4
User Output
6
12 15 19 8 63 -78
-78 63 8 19 15 12

Test Case - 5
User Output
5
-5 -10 -15 -20 -25
-25 -20 -15 -10 -5

	Exp. Name: ***Write a C program to Insert an element at Begin and Delete at End in Singly Linked List.***	**Date:**

Aim:

Fill in the missing code in the below functions `insertAtBegin(NODE first, int x)` and `deleteAtEnd(NODE first)` in the file `InsAtBeginAndDelEnd.c`.

Source Code:

`SingleLL2.c`

ID: 24F11A05K1

2024-2028-CSE-D

Narayana Engineering College - Gudur

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

#include "InsAtBeginAndDelEnd.c"

void main() {
        NODE first = NULL;
        int x, op;
        while(1) {
                printf("1.Insert At Begin 2.Delete at End 3.Traverse
the List 4.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1: printf("Enter an element : ");
                                        scanf("%d", &x);
                                        first = insertAtBegin(first,
x);
                                        break;
                        case 2:if (first == NULL) {
                                                printf("Single Linked
List is empty so deletion is not possible\n");
                                        } else {
                                                first =
deleteAtEnd(first);
                                        }
                                        break;
                        case 3: if (first == NULL) {
                                                printf("Single Linked
List is empty\n");
                                        } else {
                                                printf("The elements in
SLL are : ");
                                                traverseList(first);
                                        }
                                        break;
                        case 4: exit(0);
                }
        }
}
```

InsAtBeginAndDelEnd.c

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

NODE createNode() {
        NODE temp;
        temp = (NODE)malloc(sizeof(struct node));
        temp -> next = NULL;
        return temp;
}

NODE insertAtBegin(NODE first, int x) {
        NODE temp;
        temp = createNode();
        temp -> data = x;
        temp -> next = first;
        first = temp;
        return first;
}
NODE deleteAtEnd(NODE first){
        NODE last = first;
        NODE prev;
        if (last -> next == NULL) {
                first = first -> next;
        }else{
                while (last -> next != NULL) {
                        prev = last;
                        last =last -> next;
                }
                prev -> next = NULL;
        }
        printf("The deleted item from SLL : %d\n", last -> data);
        free (last);
        return first;
}

void traverseList(NODE first) {
        NODE temp = first;
        while (temp != NULL) {
                printf("%d --> ",temp -> data);
                temp = temp -> next;
        }
        printf("NULL\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
15
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
49
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
26
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
3
The elements in SLL are : 26 --> 49 --> 15 --> NULL
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
2
The deleted item from SLL : 15
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
3
The elements in SLL are : 26 --> 49 --> NULL
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
4

Test Case - 2
User Output
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit

2
Single Linked List is empty so deletion is not possible
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
3
Single Linked List is empty
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
15
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
2
The deleted item from SLL : 15
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
3
Single Linked List is empty
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
4

	Exp. Name: ***Write a C program to Insert an element at End in Singly Linked List***	**Date:**

Aim:

Write a C program to implement a single linked list that allows inserting elements at the end and displaying the elements of the list.

Requirements:

- Implement the following functions:
- createNode(): This function creates a new node with its next pointer set to NULL.
- insertAtEnd(NODE first, int x): This function inserts a new node with the value x at the end of the linked list.
- traverseList(NODE first): This function displays all elements in the linked list in the format: **data1 --> data2 --> ... --> NULL**.

Input Format:

The program presents a menu with three options:

- Insert an element at the end of the linked list.
- Traverse and display the elements of the linked list.
- Exit the program.

When the user selects option 1, they will be prompted to enter an integer value as: "**Enter an element :**"

Output Format:

- When traversing the list, the output should be formatted as follows: **The elements in SLL are : data1 --> data2 --> ... --> NULL**
- If the list is empty, the output should be: **Single Linked List is empty**

Note: The driver code is already provided in SingleLL3.c. You need to implement the functions in the appropriate sections of InsAtEnding.c to fulfill the program's requirements.

Refer to visible test cases for better understanding.

Source Code:

```
SingleLL3.c
```

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

#include "InsAtEnding.c"

void main() {
        NODE first = NULL;
        int x, op;
        while(1) {
                printf("1.Insert At End 2.Traverse the List 3.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1: printf("Enter an element : ");
                                        scanf("%d", &x);
                                        first = insertAtEnd(first, x);
                                        break;
                        case 2: if (first == NULL) {
                                                printf("Single Linked
List is empty\n");
                                        } else {
                                                printf("The elements in
SLL are : ");
                                                traverseList(first);
                                        }
                                        break;
                        case 3: exit(0);
                }
        }
}
```

InsAtEnding.c

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

NODE createNode() {
        NODE temp;
        temp = (NODE)malloc(sizeof(struct node));
        temp -> next = NULL;
        return temp;
}

NODE insertAtEnd(NODE first, int x) {
        NODE temp;
        temp = createNode();
        temp -> data = x;
        if (first == NULL) {
                first = temp;
        }else{
                NODE lastNode = first;
                while (lastNode -> next != NULL) {
                        lastNode = lastNode->next;
                }
                lastNode -> next = temp;
        }
        return first;
}
NODE deleteAtEnd(NODE first){
        NODE last=first;
        NODE prev;
        if (last->next== NULL) {
                first= first -> next;
        }else{
                while(last -> next != NULL) {
                        prev=last;
                        last=last->next;
                }
                prev->next=NULL;
        }
        printf("The deleted item from SLL: %d\n", last->data);
        free(last);
        return first;
}
void traverseList(NODE first) {
        NODE temp=first;
```

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

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
10
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
20
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
30
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
2
The elements in SLL are : 10 --> 20 --> 30 --> NULL
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
3

Test Case - 2
User Output
1.Insert At End 2.Traverse the List 3.Exit

2
Single Linked List is empty
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
99
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
29
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
59
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
2
The elements in SLL are : 99 --> 29 --> 59 --> NULL
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
3

	Exp. Name: ***Write a C program to reverse the Singly Linked List.***	**Date:**

Aim:

Write a C program to reverse the elements of a single linked list.

Requirements:

The program should allow the user to:

- Create a linked list of a specified number of nodes.
- Reverse the linked list.
- Display the reversed linked list.

Input Format:

- The program prompts the user with the following statement: "**Enter no. of nodes:**"**.** Then user inputs an integer n, which indicates the number of nodes in the linked list.
- For each node, the program prompts the user with: "**Enter data:**". Then user inputs an integer value to populate the data field of each node.

Output Format:

- After the linked list is reversed, the program displays the reversed list with the following statement: "**Reversed the list:**". The output will list the elements in the reversed order, separated by spaces.
- If the list is empty or if the user tries to create a list with zero or negative nodes, the program will print: "**List size must be greater than zero:**"

Source Code:

ReverseList.c

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
int data;
struct Node* next;
};
void reverselist(struct Node** head) {
        struct Node* prevnode = NULL;
        struct Node* curnode = *head;
        struct Node* nextnode = NULL;
        while(curnode != NULL) {
                nextnode = curnode->next;
                curnode->next=prevnode;
                prevnode = curnode;
                curnode = nextnode;
        }
        *head = prevnode;
}
void printlist(struct Node* head) {
        struct Node* temp = head;
        while(temp != NULL) {
                printf("%d ",temp->data);
                temp=temp->next;
        }
        printf("\n");
}
void main(){
        int n;
        do {
                printf("Enter no.of nodes: ");
                scanf("%d",&n);
                if(n<=0) {
                        printf("List size must be greater than
zero:\n");
                }
        }while(n<=0);
        struct Node* head = NULL;
        struct Node* temp = NULL;
        printf("Enter data: ");
        for(int i=0;i<n;i++){
                int data;
                scanf("%d",&data);
                struct Node* newnode = (struct
Node*)malloc(sizeof(struct Node));
                newnode->data=data;
                newnode->next=NULL;
```

```
                if(head==NULL) {
                        head=newnode;
                }else{
                        temp->next=newnode;
                }
                temp=newnode;
        }
                reverselist(&head);
                printf("Reversed the list: ");
                printlist(head);

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter no.of nodes:
4
Enter data:
1 2 3 4
Reversed the list: 4 3 2 1

Test Case - 2
User Output
Enter no.of nodes:
0
List size must be greater than zero:
Enter no.of nodes:
10
Enter data:
15 12 31 14 158 140 465 235 48 49
Reversed the list: 49 48 235 465 140 158 14 31 12 15

	Exp. Name: ***Reverse of a Single Linked List Recursively***	**Date:**

Aim:

Write a C program to create a linked list, reverse it, and display both the original and reversed linked lists.

Requirements:

The program should allow the user to:

- Specify the number of nodes in the linked list.
- Input data for each node.
- Display the original linked list.
- Reverse the linked list and display the reversed list.

Input Format:

- The program prompts the user to input an integer n, indicating the number of nodes in the linked list.
- For each node, the program prompts the user to enter an integer value, indicating the data for the node. The prompt will specify the node number, ranging from 1 to n.

Output Format:

- After the linked list is created, the program displays the original list. The output will list the elements in the order they were added, followed by Null.
- After reversing the linked list, the program displays the reversed linked list. The output will show the elements in reversed order, followed by Null.

Note:

- Add new line char (\n) at the end of each output.
- The partial code has been provided to you in the editor, you are required to fill in the missing code.

Source Code:

```
RecursiveReverse.c
```

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

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

// Write a function to create a new node with the given data
struct Node* createNode(int data) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
        newNode->data = data;
        newNode->next = NULL;
        return newNode;
}
// Write a function to print the linked list
void printList(struct Node* head) {
        struct Node* temp = head;
        while (temp != NULL) {
                printf("%d -> ", temp->data);
                temp = temp->next;
        }
        printf("Null\n");
}
// Write a function to reverse the linked list recursively
struct Node* reverseList(struct Node* current, struct Node* prev) {
        if (current == NULL) return prev; // Base case
        struct Node* nextNode = current->next;
        current->next = prev;

        return reverseList(nextNode, current);
}
// Write your main function here
int main() {
    int n, data;

    // Getting the number of nodes from the user
    printf("No of nodes: ");
    scanf("%d", &n);

    // Creating the linked list based on user input
    struct Node* head = NULL;
    struct Node* tail = NULL;

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

```
        printf("Data for node %d: ", i + 1);
        scanf("%d", &data);

        struct Node* newNode = createNode(data);

        if (head == NULL) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }

    printf("Original linked list: ");
    printList(head);

    // Reversing the linked list
    head = reverseList(head, NULL);

    printf("Reversed linked list: ");
    printList(head);

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

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
No of nodes:
5
Data for node 1:
5
Data for node 2:

Data for node 3:
3
Data for node 4:
2
Data for node 5:
1
Original linked list: 5 -> 4 -> 3 -> 2 -> 1 -> Null
Reversed linked list: 1 -> 2 -> 3 -> 4 -> 5 -> Null

Test Case - 2
User Output
No of nodes:
7
Data for node 1:
1
Data for node 2:
2
Data for node 3:
3
Data for node 4:
4
Data for node 5:
3
Data for node 6:
2
Data for node 7:
1
Original linked list: 1 -> 2 -> 3 -> 4 -> 3 -> 2 -> 1 -> Null
Reversed linked list: 1 -> 2 -> 3 -> 4 -> 3 -> 2 -> 1 -> Null

	Exp. Name: ***Single Linked List operations***	**Date:**

Aim:

Write a C program to implement a menu driven Program for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Branch, Sem, PhNo

1. Create a SLL of N Students Data by using front insertion.
2. Display the status of SLL and count the number of nodes in it
3. Insertion at End of SLL
4. Deletion at End of SLL
5. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)
6. Exit.

Source Code:

sll.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node {
char USN[25], Name[25],Branch[25];
int Sem;
long long int PhNo;
struct node* next;
};
struct node* insertAtBegin(struct node* head) {
struct node* newNode = (struct node*)malloc(sizeof(struct node));
scanf("%s %s %s %d %lld", newNode->USN, newNode->Name, newNode->Branch,
&newNode->Sem, &newNode->PhNo);
newNode->next = head;
return newNode; }
struct node* insertAtEnd(struct node* head) {
struct node* cur = head;
struct node* newNode = (struct node*)malloc(sizeof(struct
node));
scanf("%s %s %s %d %lld", newNode->USN, newNode->Name, newNode->Branch,
&newNode->Sem, &newNode->PhNo);
newNode->next = NULL;
if (cur == NULL)
return newNode;
while (cur->next != NULL)
cur = cur->next;
cur->next = newNode;
return head; }
struct node*
deleteAtEnd(struct node* head) {
struct node* cur = head;
struct node* prev = NULL;
if (cur == NULL) {
printf("Linked List is empty\n");
return NULL; }
if (cur->next == NULL) {
printf("The student node with usn: %s is deleted\n", cur->USN);
free(cur);
return NULL;
}
while (cur->next != NULL) {
prev = cur;
cur = cur->next;
        }
printf("The student node with usn: %s is deleted\n", cur->USN);
free(cur);
```

```
prev->next = NULL;
return head; }
void display(struct node*
head) {
struct node* temp = head;
int i = 0;
printf("The details of the students: \n");
while (temp != NULL) {
printf("--%d-- USN:%s| Name:%s| Branch:%s| Sem:%d| Ph:%lld|\n", ++i,
temp->USN, temp->Name, temp->Branch, temp->Sem, temp->PhNo);
temp = temp->next;
}
printf("No of student nodes is %d\n",i);
}
struct node*
deleteFirst(struct node* head) {
if (head == NULL) {
printf("Linked list is empty\n");
return NULL;
}
struct node* cur = head;
head = head->next;
printf("The Student node with usn: %s is deleted\n", cur->USN);
free(cur);
return head;
}
int main() {
struct node* head = NULL;
int ch, ch1, n, i, Sem;
char USN[25], Name[25], Branch[25];
long long int PhNo;
do {
printf("1: Create SLL of Student Nodes\n2: DisplayStatus\n3:
InsertAtEnd\n");
printf("4: DeleteAtEnd\n5: Stack Demo using SLL(Insertion and Deletion
at Front)\n6: Exit\n");
printf("Enter your choice: ");
scanf("%d", &ch);
switch (ch) {
case 1:
printf("Enter the no of students: ");
scanf("%d", &n);
        for (i = 0; i < n; i++) {
printf("Enter usn, Name, Branch, sem, PhoneNo of student: ");
head = insertAtBegin(head);
}
```

```
break;
case 2:
if (head == NULL)
printf("---No elements to display---\n");
else
display(head);
break;
case 3:
printf("Enter usn, Name, Branch, sem, PhoneNo of student: ");
head = insertAtEnd(head);
break;
case 4:
head = deleteAtEnd(head);
break;
case 5:
do {
printf("1: Push operation\n2: Pop operation\n3: Display\n4: Exit\n");
printf("Enter your choice for stack demo: ");
scanf("%d", &ch1);
switch (ch1) {
case 1:
printf("Enter usn, Name, Branch, sem, PhoneNo of student: ");
head = insertAtEnd(head);
break;
case 2:
head = deleteFirst(head);
break;
case 3:
if (head == NULL)
printf("---No elements to display---\n");
else
display(head);
break; }
} while (ch1 != 4);
break;
case 6:
exit(0); }
} while (1);
return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
2
---No elements to display---
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
4
Linked List is empty
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
5
1: Push operation
2: Pop operation
3: Display
4: Exit
Enter your choice for stack demo:
2
Linked list is empty
1: Push operation
2: Pop operation
3: Display
4: Exit
Enter your choice for stack demo:

1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
3
Enter usn, Name, Branch, sem, PhoneNo of student:
0956 Tanjiro Anime 2 1234
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
2
The details of the students:
--1-- USN:0956| Name:Tanjiro| Branch:Anime| Sem:2| Ph:1234|
No of student nodes is 1
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
6

Test Case - 2
User Output
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
1

3
Enter usn, Name, Branch, sem, PhoneNo of student:
1328 Prasanth CSE 3 1234567890
Enter usn, Name, Branch, sem, PhoneNo of student:
1207 Sashank CSE 4 0987654321
Enter usn, Name, Branch, sem, PhoneNo of student:
1209 Akhil ECE 3 6574839201
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
2
The details of the students:
--1-- USN:1209| Name:Akhil| Branch:ECE| Sem:3| Ph:6574839201|
--2-- USN:1207| Name:Sashank| Branch:CSE| Sem:4| Ph:987654321|
--3-- USN:1328| Name:Prasanth| Branch:CSE| Sem:3| Ph:1234567890|
No of student nodes is 3
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
4
The student node with usn: 1328 is deleted
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
2
The details of the students:
--1-- USN:1209| Name:Akhil| Branch:ECE| Sem:3| Ph:6574839201|
--2-- USN:1207| Name:Sashank| Branch:CSE| Sem:4| Ph:987654321|
No of student nodes is 2

2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
3
Enter usn, Name, Branch, sem, PhoneNo of student:
1238 Prasanth CSE 4 1234567890
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
2
The details of the students:
--1-- USN:1209| Name:Akhil| Branch:ECE| Sem:3| Ph:6574839201|
--2-- USN:1207| Name:Sashank| Branch:CSE| Sem:4| Ph:987654321|
--3-- USN:1238| Name:Prasanth| Branch:CSE| Sem:4| Ph:1234567890|
No of student nodes is 3
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
5
1: Push operation
2: Pop operation
3: Display
4: Exit
Enter your choice for stack demo:
2
The Student node with usn: 1209 is deleted
1: Push operation
2: Pop operation
3: Display
4: Exit

The details of the students:
--1-- USN:1207| Name:Sashank| Branch:CSE| Sem:4| Ph:987654321|
--2-- USN:1238| Name:Prasanth| Branch:CSE| Sem:4| Ph:1234567890|
No of student nodes is 2
1: Push operation
2: Pop operation
3: Display
4: Exit
Enter your choice for stack demo:
2
The Student node with usn: 1207 is deleted
1: Push operation
2: Pop operation
3: Display
4: Exit
Enter your choice for stack demo:
3
The details of the students:
--1-- USN:1238| Name:Prasanth| Branch:CSE| Sem:4| Ph:1234567890|
No of student nodes is 1
1: Push operation
2: Pop operation
3: Display
4: Exit
Enter your choice for stack demo:
4
1: Create SLL of Student Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: Stack Demo using SLL(Insertion and Deletion at Front)
6: Exit
Enter your choice:
6

	Exp. Name: ***Write code for removing duplicate elements in Singly Linked List***	**Date:**

Aim:

Write a program to remove all the duplicate elements that are present in the given singly linked lists.

Sample Input and Output:

```
Enter list elements :
Enter element : 5
Enter element : 4
Enter element : 3
Enter element : 3
Enter element : 5
Enter element : 6
Enter element : -1
List before removing duplicates : 3 3 4 5 5 6
List after removing duplicates : 3 4 5 6
```

The algorithm is as follows:

```
Step-1: Take input elements of the linked list.
Step-2: Arrange the elements in sorted order.
Step-3: Traverse from the head of the sorted linked list
Step-4: While traversing, compare the current node with the next
node.
Step-5: If data of the next node is the same as the current node
then delete the next node.
Step-6: Print the resultant list elements
```

Fill the missing code in the `NODE removeDuplicates` function in the file RemoveLL.c

Source Code:

`SingleLL10.c`

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

int main() {
        NODE l1;
        l1 = NULL;
        printf("Enter list elements :\n");
        l1 = createAndAddNodes(l1);
        sort(l1);
        printf("List before removing duplicates : ");
        print(l1);
        printf("\n");
        printf("List after removing duplicates : ");
        removeDuplicates(l1);
        print(l1);
}
```

RemoveLL.c

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

NODE createAndAddNodes(NODE first) {
        NODE temp, q;
        int x;
        printf("Enter element : ");
        scanf("%d", &x);
        while(x != -1) {
                temp = (NODE)malloc(sizeof(struct node));
                temp->data = x;
                temp->next = NULL;
                if(first == NULL) {
                        first = temp;
                } else {
                        q->next = temp;
                }
                q = temp;
                printf("Enter element : ");
                scanf("%d", &x);
        }
        return first;
}
void print(NODE node) {
        while (node != NULL) {
                printf("%d ", node->data);
                node = node -> next;
        }
}
NODE sort(NODE first) {
        NODE t1, t2;
        int x;
        for(t1 = first; t1 -> next != NULL; t1 = t1 -> next) {
                for(t2 = t1 -> next; t2 != NULL; t2 = t2 -> next) {
                        if (t1 -> data > t2 -> data) {
                                x = t1 -> data;
                                t1 -> data = t2 -> data;
                                t2 -> data = x;
                        }
                }
        }
        return first;
}
```

```
NODE removeDuplicates(NODE head) {
     NODE current = head;
        NODE nextd;
        if(head==NULL)
                return NULL;
        while(current != NULL && current ->next !=NULL){
                if(current->data==current ->next->data){
                        nextd=current->next->next;
                        free(current ->next);
                        current ->next=nextd;
                }else{
                        current=current->next;
                }
        }
     return head;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter list elements :
Enter element :
5
Enter element :
4
Enter element :
3
Enter element :
3
Enter element :
5
Enter element :
6
Enter element :
-1
List before removing duplicates : 3 3 4 5 5 6
List after removing duplicates : 3 4 5 6

	Exp. Name: ***Polynomial Operations - Adding Polynomials using Linked List***	**Date:**

ID: 24F11A05K1

Aim:

Write a C program to add two polynomials using linked lists.

Input Format:

- The first line contains an integer $n1$, the number of terms in the first polynomial.
- The next $n1$ lines each contain two integers:
  - The coefficient and the exponent of each term.
- The following line contains an integer $n2$, the number of terms in the second polynomial.
- The next $n2$ lines each contain two integers:
  - The coefficient and the exponent of each term.

Output Format:

- The first line displays: "First polynomial: <poly1 terms>"
- The second line displays: "Second polynomial: <poly2 terms>"
- The third line displays: "Addition: <result terms>" where,
  - Each polynomial is printed as terms of the form: "<coefficient> X^<exponent>"
  - Terms are joined by " + ".
  - Terms with the same exponent are added.
  - If no terms are present, print: "0".

Note:

- The terms can be in any order, but the output will be printed in descending order of exponents.
- Driver code is provided in PolyLLMain1.c. You need to implement the functions in the appropriate sections of AddPolyLL.c to fulfill the program's requirements.

Source Code:

```
PolyLLMain1.c
```

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

poly create(poly head) {
        poly temp;
        char ch;
        int coeff, exp;
        do {
                temp = (poly)malloc(sizeof(struct polynomial));
                printf("Coeff and Power of the term: ");
                scanf("%d%d", &coeff, &exp);
                temp -> coeff = coeff;
                temp -> exp = exp;
                temp -> next = NULL;
                head = addTerm(head, temp);
                printf("Want to add more terms?(y/n): ");
                scanf(" %c", &ch);
        } while(ch != 'n');
        return head;
}

void main() {
        poly head1=NULL, head2= NULL, result = NULL;
        int ch;
        printf("First polynomial: \n");
        head1 = create(head1);
        printf("Second polynomial: \n");
        head2 = create(head2);
        result = add(head1, head2);
        printf("First polynomial: ");
        print(head1);
        printf("Second polynomial: ");
        print(head2);
        printf("Addition: ");
        print(result);
}
```

AddPolyLL.c

```
#include<stdio.h>
#include<stdlib.h>
typedef struct polynomial {
        int coeff;
        int exp;
        struct polynomial *next;
} *poly;

poly addTerm(poly head, poly term) {
        poly temp, prev;
temp = head;
if (head == NULL) {
head = term;
return head;
}
if (term->exp > head->exp) {
term->next = head;
head = term;
return head;
}
if (term->exp == head->exp) {
head->coeff += term->coeff;
free(term);
return head;
}
while (temp->next != NULL && temp->next->exp > term->exp) {
temp = temp->next;
}
if (temp->next != NULL && temp->next->exp == term->exp) {
temp->next->coeff += term->coeff;
free(term);
} else {
term->next = temp->next;
temp->next = term;
}
return head;
}


void print(poly head) {
        if (head == NULL) {
printf("0");
return;
}
while (head != NULL) {
printf("%d X^%d", head->coeff, head->exp);
```

```
head = head->next;
if (head != NULL)
printf(" + ");
}
printf("\n");
}

poly add(poly poly1, poly poly2) {
        poly result = NULL;
while (poly1 != NULL && poly2 != NULL) {
poly term = (poly)malloc(sizeof(struct polynomial));
        if (poly1->exp > poly2->exp) {
term->coeff = poly1->coeff;
term->exp = poly1->exp;
poly1 = poly1->next;
} else if (poly1->exp < poly2->exp) {
term->coeff = poly2->coeff;
term->exp = poly2->exp;
poly2 = poly2->next;
} else {
term->coeff = poly1->coeff + poly2->coeff;
term->exp = poly1->exp;
poly1 = poly1->next;
poly2 = poly2->next;
}
term->next = NULL;
result = addTerm(result, term);
}
while (poly1 != NULL) {
result = addTerm(result, poly1);
poly1 = poly1->next;
}
while (poly2 != NULL) {
result = addTerm(result, poly2);
poly2 = poly2->next;
}
while (poly2 != NULL) {
result = addTerm(result, poly2);
poly2 = poly2->next;
}
return result;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
First polynomial:
Coeff and Power of the term:
2 3
Want to add more terms?(y/n):
y
Coeff and Power of the term:
4 2
Want to add more terms?(y/n):
y
Coeff and Power of the term:
6 1
Want to add more terms?(y/n):
y
Coeff and Power of the term:
8 0
Want to add more terms?(y/n):
n
Second polynomial:
Coeff and Power of the term:
1 3
Want to add more terms?(y/n):
y
Coeff and Power of the term:
3 2
Want to add more terms?(y/n):
y
Coeff and Power of the term:
5 1
Want to add more terms?(y/n):
y
Coeff and Power of the term:
7 0
Want to add more terms?(y/n):
n
First polynomial: 2 X^3 + 4 X^2 + 6 X^1 + 8 X^0
Second polynomial: 1 X^3 + 3 X^2 + 5 X^1 + 7 X^0

Test Case - 2
User Output
First polynomial:
Coeff and Power of the term:
1 3
Want to add more terms?(y/n):
y
Coeff and Power of the term:
2 3
Want to add more terms?(y/n):
n
Second polynomial:
Coeff and Power of the term:
3 4
Want to add more terms?(y/n):
y
Coeff and Power of the term:
4 4
Want to add more terms?(y/n):
n
First polynomial: 3 X^3
Second polynomial: 7 X^4
Addition: 7 X^4 + 3 X^3

	Exp. Name: ***Implementation of double ended queue using linked list - inject, eject and display operations***	**Date:**

Aim:

Implementation of double ended queue using linked list

Fill the missing code int functions `inject (int ele)`, `eject`, and `display()` in the below code.

The function `inject(int ele)` inserts an element `ele` into the `rear` of double ended queue and it gives **"Dequeue is overflow."** error if the maximum capacity of the dequeue is reached.

The function `eject()` removes an element from the `rear` of the double ended queue and it gives **"Dequeue is underflow."** error if there are no more elements in the dequeue.

The function `display()` prints all the elements of the double ended queue.

Source Code:

`DequeueListMain1.c`

```
#include <stdio.h>
#include <stdlib.h>
#include "DequeueListInjectEject.c"
int main() {
  int op, x;
  while (1) {
    printf("1.Inject 2.Eject 3.Display 4.Exit\n");
    printf("Enter your option : ");
    scanf("%d", & op);
    switch (op) {
    case 1:
      printf("Enter element : ");
      scanf("%d", & x);
      inject(x);
      break;
    case 2:
      eject();
      break;
    case 3:
      display();
      break;
    case 4:
      exit(0);
    }
  }
}
```

DequeueListInjectEject.c

```
struct queue {
int data;
struct queue *next;
};
typedef struct queue *DeQueue;
DeQueue front = NULL, rear = NULL;
void inject(int ele) {
DeQueue temp = (DeQueue)malloc(sizeof(struct queue));
if (temp==NULL){
printf("Dequeue is overflow.\n");
return;
}
temp->data=ele;
temp->next=NULL;
if(front==NULL)
front=temp;
else
rear->next=temp;
rear=temp;
printf("Successfully inserted at rear side.\n");
}
void eject() {
if (rear==NULL){
printf("Dequeue is underflow.\n");
return; }
DeQueue temp = front;
if(front==rear)
front=rear=NULL;
else{
while(temp->next!=rear)
temp=temp->next;
rear=temp;
temp=rear->next;
rear->next=NULL;
}
printf("Deleted element %d from the rear side.\n",temp->data);
free(temp);
}
void display() {
if (front==NULL){
printf("Double ended queue is empty.\n");
return;
}
printf("Elements in the double ended queue : \n");
DeQueue temp=front;
while(temp!=NULL){
```

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

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Inject 2.Eject 3.Display 4.Exit
Enter your option :
1
Enter element :
12
Successfully inserted at rear side.
1.Inject 2.Eject 3.Display 4.Exit
Enter your option :
1
Enter element :
13
Successfully inserted at rear side.
1.Inject 2.Eject 3.Display 4.Exit
Enter your option :
1
Enter element :
14
Successfully inserted at rear side.
1.Inject 2.Eject 3.Display 4.Exit
Enter your option :
3
Elements in the double ended queue :
12 13 14
1.Inject 2.Eject 3.Display 4.Exit
Enter your option :
2
Deleted element 14 from the rear side.
1.Inject 2.Eject 3.Display 4.Exit
Enter your option :

1.Inject 2.Eject 3.Display 4.Exit
Enter your option :
3
Elements in the double ended queue :
12 13
1.Inject 2.Eject 3.Display 4.Exit
Enter your option :
2
Deleted element 13 from the rear side.
1.Inject 2.Eject 3.Display 4.Exit
Enter your option :
3
Elements in the double ended queue :
12
1.Inject 2.Eject 3.Display 4.Exit
Enter your option :
4

	Exp. Name: ***Double Linked List Operations***	**Date:**

Aim:

Write a C program to implement a menu-driven program for the following operations on Doubly Linked List (DLL) of Employee Data with the fields:

SSN, Name, Dept, Designation, Salary, PhNo

1. Create a DLL of N Employees Data by using end insertion.
2. Display the status of DLL and count the number of nodes in it
3. Perform Insertion and Deletion at End of DLL
4. Perform Insertion and Deletion at Front of DLL
5. Exit

Source Code:

`dllOps.c`

```
/*#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

struct node
{   char ssn[25], name[25], dept[50], designation[25];
    int sal;
    long long int phone;
    struct node *llink;
    struct node *rlink;
};

typedef struct node* NODE;
NODE first = NULL;
int count = 0;

NODE create() {
        printf("Enter ssn, Name, Department, Designation, Salary,
PhoneNo of employee: ");
}

NODE insertfront() {
}

void display()
{   NODE cur;
    int nodeno=1;
    cur = first;
    if(cur == NULL)
        printf("DLL is Empty\n");
    else{
            while(cur != NULL)
            {   printf("SSN:%s| Name:%s| Department:%s| Designation:%s|
Salary:%d| Phone no:%lld", cur->ssn, cur->name,cur->dept, cur-
>designation, cur->sal, cur->phone);
                cur = cur->rlink;
                nodeno++;
                printf("\n");
            }
            printf("No of employees: %d\n",count);
    }
}

NODE deletefront() {
                printf("DLL is empty\n");
```

```
                printf("employee with ssn: %s is deleted\n", first-
>ssn);
        printf("employee with ssn: %s is deleted\n",temp->ssn);
    free(temp);
    count--;
    return first;
}

NODE insertend() {
}

NODE deleteend() {
                printf("DLL is empty\n");
                printf("employee with ssn: %s is deleted\n",first-
>ssn);
        printf("employee with ssn: %s is deleted\n",cur->ssn);
        free(cur);
    prev->rlink = NULL;
    count--;
    return first;

}

void main() {
    int ch,i,n;
    while(1){
        printf("1: Create DLL of Employee Nodes");
        printf("\n2: DisplayStatus");
        printf("\n3: InsertAtEnd");
        printf("\n4: DeleteAtEnd");
        printf("\n5: InsertAtFront");
        printf("\n6: DeleteAtFront");
        printf("\n7: Exit");
        printf("\nPlease enter your choice: ");
        scanf("%d",&ch);
        switch(ch)
        {   case 1 : printf("Enter no of Employees: ");
                     scanf("%d",&n);
                     for(i=1;i<=n;i++)
                        first = insertend();
                     break;
            case 2 : display();
                     break;
            case 3 : first = insertend();
                     break;
```

```
            case 5 : first = insertfront();
                     break;
            case 6 : first = deletefront();
                     break;
            case 7 : exit(0);
            default: printf("Please Enter valid choice\n");
        }
    }
} */
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node {
char ssn[25], name[25], dept[25],designation[25];
int salary;
long long int phno;
struct node *prev,*next;
};
struct node* createDLL(int);
void display(struct node*);
struct node* insertEnd(struct node*);
struct node* deleteEnd(struct node*);
struct node* insertFront(struct node*);
struct node* deleteFront(struct node*);
int main() {
        struct node *head = NULL;
        int choice, n;
        do {
                printf("1: Create DLL of Employee Nodes\n2:
DisplayStatus\n3: InsertAtEnd\n");
                printf("4: DeleteAtEnd\n5: InsertAtFront\n6:
DeleteAtFront\n7: Exit\n");
                printf("Please enter your choice: ");
                scanf("%d", &choice);
                switch (choice) {
                        case 1:
                        printf("Enter no of Employees: ");
                        scanf("%d", &n);
                        head = createDLL(n);
                        break;
                        case 2:
                        if (head == NULL)
                                printf("DLL is Empty\n");
                        else
                                display(head);
                        break;
```

```
                        case 3:
                        head = insertEnd(head);
                        break;
                        case 4:
                        head = deleteEnd(head);
                        break;
                        case 5:
                        head = insertFront(head);
                        break;
                        case 6:
                        head = deleteFront(head);
                        break;
                        case 7:
                        exit(0);
                        default:
                        printf("Please Enter valid choice\n");
                }
        } while (1);
        return 0;
}
struct node* createDLL(int n) {
        struct node * head = NULL, *temp;
        int i;
        for (i = 0; i < n; i++) {
                struct node *newNode = (struct
node*)malloc(sizeof(struct node));
                if (newNode == NULL)
                {
                        printf("Memory allocation failed\n");
                        exit(1);
                }
                printf("Enter ssn, Name, Department, Designation,
Salary, PhoneNo of employee: ");
                scanf("%s %s %s %s %d %lld", newNode->ssn, newNode-
>name, newNode->dept, newNode->designation, &newNode->salary, &newNode-
>phno);
                newNode->prev = NULL;
                newNode->next = NULL;
                if (head == NULL) {
                        head = newNode;
                        temp = newNode;
                } else {
                        temp->next = newNode;
                        newNode->prev = temp;
                        temp = newNode;
                }
```

```
        }
        return head;
}
void display(struct node* head){
        struct node* temp = head;
        int count = 0;
        while (temp != NULL) {
                printf("SSN:%s| Name:%s| Department:%s| Designation:%s|
Salary:%d| Phone no:%lld\n", temp->ssn, temp->name, temp->dept, temp-
>designation, temp->salary, temp->phno);
                temp = temp->next;
                count++;
        }
        printf("No of employees: %d\n", count);
}
struct node* insertEnd(struct node* head) {
        struct node *temp = head, *newNode;
        newNode = (struct node*)malloc(sizeof(struct node));
        if (newNode == NULL) {
                printf("Memory allocation failed\n");
        exit(1);
        }
        printf("Enter ssn, Name, Department, Designation, Salary,
PhoneNo of employee: ");
        scanf("%s %s %s %s %d %lld", newNode->ssn, newNode->name,
newNode->dept, newNode->designation, &newNode->salary, &newNode->phno);
        newNode->next = NULL;
        if (head == NULL) {
                head = newNode;
                newNode->prev = NULL;
        } else {
                while (temp->next != NULL)
                        temp = temp->next;
                temp->next = newNode;
                newNode->prev = temp;
        }
        return head;
}
struct node* deleteEnd(struct node* head) {
        if (head == NULL) {
                printf("DLL is empty\n");
                return NULL;
        } else if (head->next == NULL) {
                printf("employee with ssn: %s is deleted\n", head-
>ssn);
```

```
        } else {
                struct node *temp = head;
                while (temp->next != NULL)
                        temp = temp->next;
                printf("employee with ssn: %s is deleted\n", temp-
>ssn);
                temp->prev->next = NULL;
                free(temp);
                return head;
        }
}
struct node* insertFront(struct node* head) {
        struct node *newNode;
        newNode = (struct node*)malloc(sizeof(struct node));
        if (newNode == NULL) {
                printf("Memory al;location failed\n");
                exit(1);
        }
        printf("Enter ssn, Name, Department, Designation, Salary,
PhoneNo of employee: ");
        scanf("%s %s %s %s %d %lld", newNode->ssn, newNode->name,
newNode->dept, newNode->designation, &newNode->salary, &newNode->phno);
        newNode->prev = NULL;
        newNode->next = head;
        if (head != NULL)
                head->prev = newNode;
        return newNode;
}
struct node* deleteFront(struct node* head) {
        if (head == NULL) {
                printf("DLL is empty\n");
                return NULL;
        } else {
                struct node* temp = head;
                head = head->next;
                printf("employee with ssn: %s is deleted\n", temp-
>ssn);
                free(temp);
                if (head != NULL)
                        head->prev = NULL;
                return head;
        }
}
```

Test Case - 1
User Output
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
DLL is Empty
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
1
Enter no of Employees:
2
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
CT156 Hema Support PSE 30000 1234567890
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
CT188 Prasanth Support PSE 30000 1234567890
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2

SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
SSN:CT188| Name:Prasanth| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
No of employees: 2
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
3
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
CT226 Swathi Support PSE 30000 1234567890
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
SSN:CT188| Name:Prasanth| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
SSN:CT226| Name:Swathi| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
No of employees: 3
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:

employee with ssn: CT226 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
SSN:CT188| Name:Prasanth| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
No of employees: 2
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
5
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
CT156 Bhanu Support PSE 34000 1234567890
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT156| Name:Bhanu| Department:Support| Designation:PSE| Salary:34000| Phone no:1234567890
SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890

No of employees: 3
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT156| Name:Bhanu| Department:Support| Designation:PSE| Salary:34000| Phone no:1234567890
SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
SSN:CT188| Name:Prasanth| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
No of employees: 3
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
6
employee with ssn: CT156 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
4
employee with ssn: CT188 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront

7: Exit
Please enter your choice:
2
SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
No of employees: 1
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
6
employee with ssn: CT156 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
DLL is Empty
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
4
DLL is empty
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront

Please enter your choice:
6
DLL is empty
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
3
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
198 Tanjiro Anime Hero 49000 1029384756
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:198| Name:Tanjiro| Department:Anime| Designation:Hero| Salary:49000| Phone no:1029384756
No of employees: 1
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
4
employee with ssn: 198 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd

6: DeleteAtFront
7: Exit
Please enter your choice:
7

Test Case - 2
User Output
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
3
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
CT590 Japan Korea Indonesia 39000 0987654321
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT590| Name:Japan| Department:Korea| Designation:Indonesia| Salary:39000| Phone no:987654321
No of employees: 1
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:

employee with ssn: CT590 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
DLL is Empty
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
5
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
AP996 Sampath Support PSE 15000 9086745231
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:AP996| Name:Sampath| Department:Support| Designation:PSE| Salary:15000| Phone no:9086745231
No of employees: 1
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront

Please enter your choice:
4
employee with ssn: AP996 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
DLL is Empty
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
1
Enter no of Employees:
1
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
TS289 Kalkanrat PublicSector Government 58000 1029238845
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:TS289| Name:Kalkanrat| Department:PublicSector| Designation:Government| Salary:58000| Phone no:1029238845
No of employees: 1
1: Create DLL of Employee Nodes

```
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
8
Please Enter valid choice
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
7
```

	Exp. Name: ***C program which performs all operations in Circular linked list.***	**Date:**

Aim:

Write a program that uses functions to perform the following operations on circularlinked list.

i) Creation
ii) Insertion
iii) Deletion
iv) Traversal

Source Code:

```
AlloperationsinCLL.c
```

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

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

NODE createNodeInCLL() {
}

NODE insertAtPositionInCLL(NODE first, int pos, int x) {
        printf("No such position in CLL so insertion is not
possible\n");
}

NODE deleteAtPositionInCLL(NODE first, int pos) {
void traverseListInCLL(NODE first) {

}

void main() {
        NODE first = NULL;
        int x, pos, op;
        while(1) {
                printf("1.Insert 2.Delete 3.Print 4.Exit\n");
                printf("Enter your option: ");
                scanf("%d", &op);
                switch(op) {
                        case 1: printf("Enter a position: ");
                                        scanf("%d", &pos);
                                        if (pos <= 0) {
                                                printf("No such
position in CLL so insertion is not possible\n");
                                        } else {
                                                printf("Enter an
element: ");
                                                scanf("%d", &x);
                                                first =
insertAtPositionInCLL(first, pos, x);
                                        }
                                        break;
                case 2: if (first == NULL) {
                                                printf("Circular Linked
List is empty so deletion is not possible\n");
```

```
                                        } else {
                                                printf("Enter position
: ");
                                                scanf("%d", &pos);
                                                first =
deleteAtPositionInCLL(first, pos);
                                        }
                                        break;
                        case 3: if (first == NULL) {
                                                printf("Circular Linked
List is empty\n");
                                        } else {
                                                printf("The elements in
CLL are: ");

traverseListInCLL(first);
                                        }
                                        break;
                        case 4: exit(0);
                }
        }
}
*/
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node* next;
};
struct Node* head =NULL;
void insert(int position, int data)
{
        struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
        newNode->data=data;
        if(position==1||head==NULL){
                if(head==NULL){
                        newNode->next = newNode;
                        head = newNode;
                } else {
                        struct Node* temp = head;
                        while(temp->next!=head)
temp=temp->next;
temp->next=newNode;
newNode->next=head;
head=newNode; }
```

```
} else {
                struct Node* temp=head;
int count=1;
while(count < position-1&&temp->next!=head){
temp = temp->next;
count++;
}
if (count!=position-1) {
printf("No such position in CLL so insertion is not possible\n");
return;
}
newNode->next = temp->next;
temp->next = newNode;
        }
}void delete(int position) {
if (head == NULL) {
printf("Circular Linked List is empty so deletion is not possible\n");
return;
}
struct Node* temp = head;
struct Node* prev = NULL;
int deletedData;
if (position == 1) {
while (temp->next != head)
temp = temp->next;
if(temp==head){
deletedData = head->data;
free(head);
head = NULL;
printf("The deleted element from CLL : %d\n",deletedData);
return;
}
struct Node* delNode=head;
deletedData=delNode->data;
temp->next=head->next;
head=head->next;
free(delNode);
} else {
int count=1;
while(count<position && temp->next != head){
prev=temp;
temp=temp->next;
count++;
}
if(count != position || temp == head){
printf("No such position in CLL so deletion is not possible\n");
```

```
        return;
}
struct Node* delNode=temp;
deletedData=delNode->data;
prev->next=temp->next;
free(delNode);
}
printf("The deleted element from CLL : %d\n",deletedData);
}
void display ( ) {
if (head == NULL){
printf("Circular Linked List is empty\n");
return;
}
struct Node* temp = head;
printf("The elements in CLL are: ");
do {
printf("%d --> ", temp->data);
temp = temp->next;
} while (temp != head);
printf("\n");
}
void main(){
int choice, position, data;
while (1) {
printf("1.Insert 2.Delete 3.Print 4.Exit\n");
printf("Enter your option: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter a position: ");
scanf("%d", & position);
printf("Enter an element: ");
scanf("%d", &data);
insert(position, data);
break;
case 2:
if (head == NULL) {
printf("Circular Linked List is empty so deletion is not possible\n");
} else {
printf("Enter position : ");
scanf("%d", &position);
delete (position);
}
break;
case 3:
```

```
display();
break;
case 4:
exit(0);
default:
printf("Invalid option! Please choose again.\n");
}
}
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
1
Enter an element:
1
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
2
Enter an element:
2
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
3
Enter an element:
3
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1

4
Enter an element:
4
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
5
Enter an element:
5
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
6
Enter an element:
6
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
3
The elements in CLL are: 1 --> 2 --> 3 --> 4 --> 5 --> 6 -->
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
2
Enter position :
3
The deleted element from CLL : 3
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
2
Enter position :
3
The deleted element from CLL : 4
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
2
Enter position :
3
The deleted element from CLL : 5
1.Insert 2.Delete 3.Print 4.Exit

3
The elements in CLL are: 1 --> 2 --> 6 -->
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
3
Enter an element:
3
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
4
Enter an element:
4
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
5
Enter an element:
5
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
3
The elements in CLL are: 1 --> 2 --> 3 --> 4 --> 5 --> 6 -->
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
4

Test Case - 2
User Output
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
2
Circular Linked List is empty so deletion is not possible
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:

Circular Linked List is empty
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
1
Enter an element:
15
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
3
Enter an element:
17
No such position in CLL so insertion is not possible
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
2
Enter position :
3
No such position in CLL so deletion is not possible
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
4

	Exp. Name: ***Stack using Arrays***	**Date:**

Aim:

Write a C program to implement stack operations using arrays.

Input Format

The program presents a menu with six options. The user inputs a choice corresponding to one of these options:

1. **Push Operation:** Input is an integer value to push onto the stack.
2. **Pop Operation:** No additional input is required.
3. **Display Operation:** No additional input is required.
4. **Is Empty Operation:** No additional input is required.
5. **Peek Operation:** No additional input is required.
6. **Exit Operation:** No additional input is required.

Output Format

The output will vary based on the selected option:

1. **Push Operation:**
2. If the stack is not full, the output will be: **Successfully pushed**
3. If the stack is full, the output will be: **Stack is overflow**
4. **Pop Operation:**
5. If the stack is not empty, it will print: **Popped value: X** where X is the element removed from the stack.
6. If the stack is empty, it will print: **Stack is underflow**
7. **Display Operation:**
8. If the stack is not empty, it will print: **Elements: X Y Z ...** where X, Y, Z, etc., are the elements of the stack from top to bottom.
9. If the stack is empty, it will print: **Stack is empty**
10. **Is Empty Operation:**
11. If the stack is empty, it will print: **Stack is empty**
12. If the stack is not empty, it will print: **Stack is not empty**
13. **Peek Operation:**
14. If the stack is not empty, it will print: **Peek value: X** where X is the top element of the stack.
15. If the stack is empty, it will print: **Stack is underflow**
16. **Exit Operation:**
17. The program will terminate with no additional output beyond the program's exit.

Source Code:

StackUsingArray.c

```
#include <stdio.h>
#include <stdlib.h>
#define STACK_MAX_SIZE 10
#include "StackOperations.c"

int main() {
        int op, x;
        while(1) {
                printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek
6.Exit\n");
                printf("Option: ");
                scanf("%d", &op);
                switch(op) {
                        case 1:
                                printf("element: ");
                                scanf("%d", &x);
                                push(x);
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                peek();
                                break;
                        case 6:
                                exit(0);
                }
        }
}
```

StackOperations.c

```
// declare the size of the array
int stack[STACK_MAX_SIZE];
// define the top to -1
int top = -1;
void push(int element) {
        // write your code here to push an element
        if (top == STACK_MAX_SIZE - 1) {
                printf("Stack is overflow\n");
        } else {
                top++;
                stack[top] = element;
                printf("Successfully pushed\n");
        }
}
void pop() {
        if (top == -1) {
                printf("Stack is underflow\n");
        }else {
                printf("Popped value: %d\n", stack[top]);
                top--;
        }
}
void display() {
        // write your code here to display the stack
        if (top == -1) {
                printf("Stack is empty\n");
        }else {
                printf("Elements: ");
                for (int i = top; i >= 0; i--) {
                        printf("%d ", stack[i]);
                }
                printf("\n");
        }
}
void isEmpty() {
        if (top == -1) {
                printf("Stack is empty\n");
        }else {
                printf("Stack is not empty\n");
        }
}
void peek() {
        if (top == -1) {
                printf("Stack is underflow\n");
        }else{
                printf("Peek value: %d\n",stack[top]);
```

```
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
4
Stack is empty
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
2
Stack is underflow
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
3
Stack is empty
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
5
Stack is underflow
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
25
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
26
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
3

1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
2
Popped value: 26
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
4
Stack is not empty
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
5
Peek value: 25
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
6

Test Case - 2
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
1
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
2
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
3
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1

4
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
5
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
6
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
7
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
8
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
9
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
10
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:

element:
11
Stack is overflow
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
6

	Exp. Name: ***Stack Implementation Using Linked Lists***	**Date:**

Aim:

Write a program to implement a stack using linked lists.

Input Format:

The user is presented with a menu of options and provides input according to the desired operation:

1. **Push Operation:**
2. Input: Integer value to be pushed onto the stack.
3. **Pop Operation:**
4. No additional input is required.
5. **Display Operation:**
6. No additional input is required.
7. **Is Empty Operation:**
8. No additional input is required.
9. **Peek Operation:**
10. No additional input is required.
11. **Exit Operation:**
12. No additional input is required.

Output Format:

The output will vary depending on the selected option:

1. **Push Operation:**
2. If the stack is not full (no overflow), the output will be: "Successfully pushed."
3. **Pop Operation:**
4. If the stack is not empty, it will print: "Popped value = X" where X is the value removed from the stack. If the stack is empty, it will print: "Stack is underflow."
5. **Display Operation:**
6. If the stack is not empty, it will print: "Elements of the stack are : X Y Z ..." where X, Y, Z, etc., are the elements from top to bottom. If the stack is empty, it will print: "Stack is empty."
7. **Is Empty Operation:**
8. If the stack is empty, it will print: "Stack is empty." If the stack is not empty, it will print: "Stack is not empty."
9. **Peek Operation:**
10. If the stack is not empty, it will print: "Peek value = X" where X is the top element of the stack.If the stack is empty, it will print: "Stack is underflow."
11. **Exit Operation:**
12. The program terminates with no additional output.

Note:

- The partial code has been provided to you in the editor, you are required to fill in the missing code.

Source Code:

StackUsingLL.c

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

int main() {
        int op, x;
        while(1) {
                printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek
6.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d", &x);
                                push(x);
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                peek();
                                break;
                        case 6:
                                exit(0);
                }
        }
}
```

StackOperationsLL.c

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

typedef struct stack *stk;
stk top = NULL;

stk push(int x) {

        stk temp;
        temp=(stk)malloc(sizeof(struct stack));
        if(temp==NULL){
                printf("Stack is overflow.\n");
        }else{
                temp -> data = x;
                temp -> next = top;
                top=temp;
                printf("Successfully pushed.\n");
        }
        return temp;
}
stk pop() {
        stk temp;
        if(top == NULL){
                printf("Stack is underflow.\n");
        } else {
                temp=top;
                top= top -> next;
                printf("Popped value = %d\n", temp -> data);
                free(temp);
        }
        return temp;
}

void peek() {
        stk temp;
        if(top==NULL){
                printf("Stack is underflow.\n");
        }else{
                temp=top;
                printf("Peek value = %d\n",temp -> data);
        }
}
void isEmpty() {
        if(top==NULL){
```

```
                printf("Stack is empty.\n");
        }else{
                printf("Stack is not empty.\n");
        }
}

void display() {
        stk temp = top;
        if(temp == NULL) {
                printf("Stack is empty.\n");
        } else {
                printf("Elements of the stack are : ");
                while(temp != NULL) {
                        printf("%d ", temp -> data);
                        temp = temp -> next;
                }
                printf("\n");
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
33
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
22
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :

55
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
66
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 66 55 22 33
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 66
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 55
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 22 33
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 22
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is not empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

2
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
23
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
24
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 24 23
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 24
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 24
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :

1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

	Exp. Name: ***Evaluation of a Postfix expression***	**Date:**

Aim:

Write a C program to evaluate a postfix expression.

Write the code in the functions **isEmpty()**, **push(int x)**, **pop(),** and **evaluatePostfix(char *e)** in the program according to the hints given as comment lines.

Input Format:

- The user will provide a postfix expression as a single string of characters. The expression can contain digits (0-9) and operators (+, -, *, /, %).

Output Format:

- If the postfix expression is valid, the program prints the result of the evaluation in the format: "Result : <result>".
- If the postfix expression is invalid (e.g., insufficient operands for the operators or extra operands remaining), the program prints: "Invalid postfix expression."

Note: Refer to the visible test cases and strictly match with the input and outputs.

Source Code:

```
PostfixEvaluation.c
```


ID: 24F11A05K1

Narayana Engineering College - Gudur

```
#include <ctype.h>
#include <stdio.h>
#define STACK_MAX_SIZE 20
//Declare the required stack variables.
int stack[STACK_MAX_SIZE];
int top=-1;
//Return 1 if stack is empty else return 0.
int isEmpty() {
        if(top<0)
                return 1;
        else
                return 0;
}

//Push the character into stack
void push(int x) {
        if(top==STACK_MAX_SIZE-1){
                printf("stack is overflow.\n");
        }else{
                top=top+1;
                stack[top]=x;
        }
}

//pop a character from stack
int pop() {
        if(top<0){
                return -1;
        }
        else
                return stack[top--];
}

//Output Format - Result : <result> if the input postfix expression is
vaild.
//Output Format - Invalid postfix expression,. - if the input
expression is invalid.
//postfix expression is given as the parameter.
void evaluatePostfix(char * e) {
        int a,b,result;
        while(*e != '\0'){
                if(isdigit(*e))
                        push(*e-'0');
                else if(*e == '+' || *e == '-' || *e =='*' || *e == '/'
|| *e == '%'){
                        if(isEmpty()){
```

```
                                printf("Invalid postfix
expression.\n");
                                return;
                        }
                        a=pop();
                        if(isEmpty()){
                                printf("Ivalid postfix expression.\n");
                                return;
                        }
                        b=pop();
                        switch(*e){
                                case '+':
                                result =b+a;
                                break;
                                case '-':
                                result = b-a;
                                break;
                                case '*':
                                result = b * a;
                                break;
                                case '/':
                                result = b/a;
                                break;
                                case '%':
                                result = b%a;
                                break;
                        }
                        push(result);
                }
                e++;
        }
        result=pop();
        if(!isEmpty()){
                printf("Invalid postfix expression.\n");
                return;
        }
        printf("Result : %d\n",result);
}
//Read a postfix expression and evaluate it.
int main() {
        char exp[20];
        char *e, x;
        printf("Enter the postfix expression : ");
        scanf("%s",exp);
        e = exp;
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the postfix expression :
234+-
Result : -5

Test Case - 2
User Output
Enter the postfix expression :
-456+5+
Invalid postfix expression.

	Exp. Name: ***Check for the balanced parenthesis using a stack***	**Date:**

Aim:

Write a C program to check if the parentheses in a given expression are balanced. The program will recognise the following types of brackets: (), {}, and [].

Input Format:

- The user is prompted to enter an expression containing various types of parentheses. The input is a single line string.

Output Format:

The program will display one of the following messages:

- "balanced" if all parentheses are properly matched and nested.
- "not balanced" if there are any mismatches or unclosed parentheses.

Source Code:

```
BalancedParenthesis.c
```

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define STACK_MAX_SIZE 30
char arr[STACK_MAX_SIZE];
int top = -1;
void push(char element) {
        if (top == STACK_MAX_SIZE - 1) {
                printf("stack Overflow\n");
                return;
        }
        arr[++top] = element;
}

char pop() {
        if (top == -1) {
                printf("Stack Underflow\n");
                return '\0';
        }
        return arr[top--];
}

int isempty() {
        return top == -1;
}

int isBalanced(char exp[]) {
        for (int i = 0; i < strlen(exp); i++) {
                char ch = exp[i];
                if (ch == '(' || ch == '{' || ch == '[') {
                        push(ch);
                } else if (ch == ')' || ch == '}' || ch == ']') {
                        if (isempty()) {
                                return 0; // Stack is empty,so not
balanced
                        }
                        char topElement = pop();
                        if ((ch == ')' && topElement != '(') ||
                                (ch == '}' && topElement != '{') ||
                                (ch == ']' && topElement != '[')) {
                                return 0; // Mismatch found
                                }
                }
        }
        return isempty(); // If stack is empty, it's balanced
}
```

```
void main() {
        char ch[80], temp;
        printf("Enter an expression: ");
        scanf("%s", ch);
        if(isBalanced(ch) == 1) {
                printf("balanced\n");
        } else {
                printf("not balanced\n");
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter an expression:
1+2*3+(3+4)
balanced

Test Case - 2
User Output
Enter an expression:
1+2*(3+([4+5])
not balanced

Test Case - 3
User Output
Enter an expression:
{[]}
balanced

Test Case - 4
User Output

{[()}
not balanced

	Exp. Name: ***C program to implement Operations on Queue using static Array***	**Date:**

Aim:
Write a program to implement queue operations using static arrays.

Note: Driver code is provided in QueueUsingArray.c. You need to implement the functions in the appropriate sections of QueueOperations.c to fulfill the program's requirements.

Source Code:

```
QueueUsingArray.c
```

```
#include <stdlib.h>
#include <stdio.h>
#include "QueueOperations.c"
int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size
6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
        return 0;
}
```

QueueOperations.c

```
#define MAX 10
int queue[MAX];
int front = -1, rear = -1;

void enqueue(int x) {
        if(rear == MAX-1){
                printf("Queue is overflow.\n");
        }else{
                rear++;
                queue[rear]=x;
                printf("Successfully inserted.\n");
        }
        if(front == -1) {
                front++;
        }
}

void dequeue() {
        if(front ==-1) {
                printf("Queue is underflow.\n");
        }else{
                printf("Deleted element = %d\n",queue[front]);
                if(rear == front){
                        rear = front=-1;
                }else{
                        front++;
                }
        }
}

void display() {
        if(front == -1 && rear== -1){
                printf("Queue is empty.\n");
        }else{
                printf("Elements in the queue : ");
                for(int i = front; i <=rear; i++){
                        printf("%d ",queue[i]);
                }
                printf("\n");
        }
}

void size() {
        if(front == -1 && rear == -1)
                printf("Queue size : 0\n");
                else
```

```
                printf("Queue size : %d\n",rear-front+1);
}

void isEmpty() {
        if(front == -1 && rear == -1)

                printf("Queue is empty.\n");
                else
                printf("Queue is not empty.\n");


}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
14
Successfully inserted.

Enter your option :
1
Enter element :
78
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
53
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 14 78 53
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
25
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 25
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.

Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
65
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 65
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 65
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
63
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit

	Exp. Name: ***Queue using Linked Lists***	**Date:**

Aim:

Write a program that allows users to perform the following operations on a queue:

1. Enqueue an element (add to the rear).
2. Dequeue an element (remove from the front).
3. Display all elements in the queue.
4. Check if the queue is empty.
5. Get the size of the queue.
6. Exit the program.

Input Format:

The program displays a menu with the following options:

1. Enqueue
2. Dequeue
3. Display
4. Is Empty
5. Size
6. Exit

The user selects an option by entering a number corresponding to the desired operation.

For the Enqueue operation, the user is prompted to enter the integer element to be added to the queue.

Output Format:

For each operation, the program outputs the result:

1. For Enqueue, print: "Successfully inserted"
2. For Dequeue, print: "Deleted value: <X>" where <X> is the dequeued element, or "Queue is underflow" if the queue is empty.
3. For Display, print: "Elements: <element1><element2>....." showing all elements in the queue or "Queue is empty" if there are no elements.
4. For Is Empty, print: "Queue is empty" or "Queue is not empty".
5. For Size, print: "Queue size: <N>" where <N> is the number of elements in the queue. If the queue has no elements, print: "Queue size: 0"
6. For Exit, terminate the program without additional output.

Source Code:

QueueUsingLL.c

```
#include <stdlib.h>
#include <stdio.h>
#include "QueueOperationsLL.c"
int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size
6.Exit\n");
                printf("Option: ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("element: ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
}
```

QueueOperationsLL.c

```
#include <stdio.h>
#include <stdio.h>
struct Node{
int data;
struct Node* next;
};
struct Node *front = NULL;
struct Node *rear = NULL;
struct Node*createNode(int data) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
        if (newNode == NULL) {
                printf("Memory allocation failed.\n");
                exit(1);
        }
        newNode->data = data;
        newNode->next = NULL;
        return newNode;
}
void isEmpty() {
        if (front == NULL)
                printf("Queue is empty\n");
        else
                printf("Queue is not empty\n");
}
void enqueue(int data){
        struct Node* newNode = createNode(data);
        if (rear == NULL) {
                front = rear = newNode;
        }else{
                rear->next = newNode;
                rear = newNode;
        }
        printf("Successfully inserted\n");
}
void dequeue() {
        if (front == NULL) {
                printf("Queue is underflow\n");
                return;
        }
        struct Node* temp = front;
        front = front->next;
        if (front == NULL)
                rear = NULL;
        printf("Deleted value: %d\n", temp->data);
        free(temp);
```

```
}
void display() {
        if (front == NULL) {
                printf("Queue is empty\n");
                return;
        }
        struct Node* current = front;
        printf("Elements: ");
        while (current != NULL) {
                printf("%d ", current->data);
                current = current->next;
        }
        printf("\n");
}
void size() {
        int count = 0;
        struct Node* current = front;
        while (current != NULL) {
                count++;
                current = current->next;
        }
        printf("Queue size: %d\n", count);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Queue is underflow
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
3
Queue is empty
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
4
Queue is empty

```
Option:
5
Queue size: 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
44
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
55
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
66
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
67
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
3
Elements: 44 55 66 67
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Deleted value: 44
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Deleted value: 55
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
```

Queue size: 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
4
Queue is not empty
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
23
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
234
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
45
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
456
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Deleted value: 23

Option:
3
Elements: 234 45 456
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Deleted value: 234
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
3
Elements: 45 456
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
4
Queue is not empty
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
5
Queue size: 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
6

	Exp. Name: ***Simulation of a simple printer queue system.***	**Date:**

Aim:

Write a C program to implement a Printer Queue System using a linked list. The program should allow the user to add print jobs to the queue and process (remove) jobs from the queue. The program must display a menu with the following options:

```
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice: <Choice>
```

Input Format:

The input begins with a menu choice entered by the user.

- For choice 1, the program must prompt with "Job ID: " followed by an integer that is added to the queue.
- For choice 2, the program must remove the front job from the queue and process it.
- For choice 3, the program must terminate and display "Exiting".
- If the user enters any number other than 1, 2, or 3, the program should display: "Invalid choice".

Output Format:

The output should display the result of each command as follows:

- For choice 1, the program must print "Job <job_id> added", indicating that the job has been successfully added to the queue.
- For choice 2, the program must print "Job <job_id> removed from queue and sent to the printer", showing that the front job has been processed.
- If the queue is empty when this option is selected, the program must instead print "No job to dequeue".
- For choice 3, the program must print "Exiting" and then end the program.
- If the queue is empty, only the word "Exiting" is printed.If the queue still contains jobs, the program must remove them automatically and print "Job <job_id> removed from queue and sent to the printer" for each remaining job before terminating.

Note:

- Refer to the visible test cases for better understanding of the input/output formats.

Source Code:

PrinterQueue.c

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

// Define a structure for a print job
struct PrintJob {
    int jobId;
    struct PrintJob* next;
};

// Define a structure for the printer queue
struct PrinterQueue {
    struct PrintJob* front;
    struct PrintJob* rear;
};

// Function to initialize an empty printer queue
void initializeQueue(struct PrinterQueue* queue) {

        // Code here..


}

// Function to check if the queue is empty
int isQueueEmpty(struct PrinterQueue* queue) {

        // Code here..

}

// Function to enqueue a print job
void enqueue(struct PrinterQueue* queue, int jobId) {

        // Code here..







        printf("Job %d added\n", jobId);
}

// Function to dequeue a print job
void dequeue(struct PrinterQueue* queue) {
```

```
        // Code here..

                printf("No job to dequeue\n");



        printf("Job %d removed from queue and sent to the printer\n",
temp->jobId);




}

int main() {
    struct PrinterQueue printerQueue;
    initializeQueue(&printerQueue);

    int choice, jobId;

    do {
        printf("Printer Queue System\n");
        printf("1. Add a job to queue\n");
        printf("2. Process the next job\n");
        printf("3. Exit\n");
        printf("Choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Job ID: ");
                scanf("%d", &jobId);
                enqueue(&printerQueue, jobId);
                break;
            case 2:
                dequeue(&printerQueue);
                break;
            case 3:
                printf("Exiting\n");
                break;
            default:
                printf("Invalid choice\n");
        }

    } while (choice != 3);
```

```
    // Cleanup: free remaining print jobs in the queue
    while (!isQueueEmpty(&printerQueue)) {
        dequeue(&printerQueue);
    }

    return 0;
}*/
#include <stdio.h>
#include <stdlib.h>

#define MAX_QUEUE_SIZE 100

typedef struct {
    int id;
} Job;

typedef struct {
    Job jobs[MAX_QUEUE_SIZE];
    int front, rear;
} Queue;

// Initialize the queue
void initQueue(Queue *q) {
    q->front = -1;
    q->rear = -1;
}

// Check if the queue is empty
int isQueueEmpty(Queue *q) {
    return (q->front == -1);
}

// Check if the queue is full
int isQueueFull(Queue *q) {
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}

// Enqueue a job (Add to Queue)
void enqueue(Queue *q, int jobID) {
    if (isQueueFull(q)) {
        printf("Queue is full. Cannot add more jobs.\n");
        return;
    }

    if (isQueueEmpty(q)) {
        q->front = q->rear = 0;
```

```
    } else {
        q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    }

    q->jobs[q->rear].id = jobID;
    printf("Job %d added\n", jobID);
}

// Dequeue a job (Process next job)
void dequeue(Queue *q) {
    if (isQueueEmpty(q)) {
        printf("No job to dequeue\n");
        return;
    }

    int jobID = q->jobs[q->front].id;
    printf("Job %d removed from queue and sent to the printer\n",
jobID);

    if (q->front == q->rear) {
        q->front = q->rear = -1;  // Reset queue
    } else {
        q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    }
}

// Process remaining jobs after exiting
void processAllJobs(Queue *q) {
    while (!isQueueEmpty(q)) {
        dequeue(q);
    }
}

// Main function
int main() {
    Queue queue;
    initQueue(&queue);
    int choice, jobID;

    do {
        printf("Printer Queue System\n");
        printf("1. Add a job to queue\n");
        printf("2. Process the next job\n");
        printf("3. Exit\n");
        printf("Choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Job ID: ");
                scanf("%d", &jobID);
                enqueue(&queue, jobID);
                break;
            case 2:
                dequeue(&queue);
                break;
            case 3:
                printf("Exiting\n");  // First print exiting
                processAllJobs(&queue);  // Then process remaining jobs
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 3);

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
2
No job to dequeue
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
1
Job ID:

Job 1245 added
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
1
Job ID:
2345
Job 2345 added
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
1
Job ID:
2145
Job 2145 added
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
5
Invalid choice
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
2
Job 1245 removed from queue and sent to the printer
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
2
Job 2345 removed from queue and sent to the printer
Printer Queue System

2. Process the next job
3. Exit
Choice:
2
Job 2145 removed from queue and sent to the printer
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
2
No job to dequeue
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
3
Exiting

Test Case - 2
User Output
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
1
Job ID:
1245
Job 1245 added
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
1
Job ID:
3654

1. Add a job to queue
2. Process the next job
3. Exit
Choice:
1
Job ID:
8569
Job 8569 added
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
3
Exiting
Job 1245 removed from queue and sent to the printer
Job 3654 removed from queue and sent to the printer
Job 8569 removed from queue and sent to the printer

	Exp. Name: ***Circular Queues using arrays***	**Date:**

Aim:

Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX = 6)

a) Insert an Element on to Circular QUEUE
b) Delete an Element from Circular QUEUE
c) Demonstrate Overflow and Underflow situations on Circular QUEUE
d) Display the status of Circular QUEUE
e) Exit

Support the program with appropriate functions for each of the above operations.

Source Code:

cQue.c

```
#include <stdio.h>
#include<stdlib.h>
#include<stdio_ext.h>
#define MAX 6

int cq[MAX];
int front = -1, rear = -1;

void insert(int);
void delete();
void display();

void insert(int item)
{   if(front == (rear+1)%MAX) {
        printf("~~~Circular Queue Overflow~~~\n");    }
    else {
        if(rear == MAX - 1) {
                        rear = -1;
                        }
        if(front == -1)
                        front = 0;
        rear++;
                cq[rear] = item;
    }
}

void delete()
{   char item;
    if(front == -1)
    {   printf("~~~Circular Queue Underflow~~~\n");   }
    else
    {   item = cq[front];
        printf("Deleted element from the queue is: %d\n",item );
        front++;

    }
}

void display ()
{   int i ;
    if(front == -1)
    {   printf("~~~Circular Queue Empty~~~\n");   }
    else
    {   printf("Circular Queue contents are:\n");
    if(front <= rear) {
                for (i= front; i <= rear; i++)
```

```
             {
            if (i == rear)
         printf("%d", cq[i]);
                                  else
                                 printf("%d ", cq[i]);
                                 }
                }else {
                for (i= front; i < MAX; i++)

                        printf("%d", cq[i]);
                for(i= 0; i< rear; i++) {
                        if (i == rear)
                                printf("%d ", cq[i]);
                        else
                                printf("%d ", cq[i]);}
                }
                printf("\n");
    }
}

void main()
{   int ch, item;
    while(1)
    {   printf("~~Main Menu~~");
        printf("\n=> 1. Insertion and Overflow Demo");
        printf("\n=> 2. Deletion and Underflow Demo");
        printf("\n=> 3. Display");
        printf("\n=> 4. Exit");
        printf("\nEnter Your Choice: ");
        scanf("%d", &ch);
        __fpurge(stdin);
        switch(ch)
        {   case 1: printf("Enter the element to be inserted: ");
                    scanf("%d", &item);
                    insert(item);
                    break;
            case 2: delete();
                    break;
            case 3: display();
                    break;
            case 4: exit(0);
            default: printf("Please enter a valid choice\n");
        }
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
1
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
2
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
3
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
4

```
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
5
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
3
Circular Queue contents are:
1 2 3 4 5
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
6
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
3
Circular Queue contents are:
1 2 3 4 5 6
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
```

Enter the element to be inserted:
7
~~Circular Queue Overflow~~~
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
3
Circular Queue contents are:
1 2 3 4 5 6
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
2
Deleted element from the queue is: 1
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
2
Deleted element from the queue is: 2
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
2
Deleted element from the queue is: 3
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit

Circular Queue contents are:
4 5 6
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
4

Test Case - 2
User Output
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
3
~~Circular Queue Empty~~~
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
3
~~Circular Queue Empty~~~
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
2
~~Circular Queue Underflow~~~
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display

Enter Your Choice:
1
Enter the element to be inserted:
4
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
4

Test Case - 3
User Output
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
5
Please enter a valid choice
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
4

	Exp. Name: ***Circular Queue using Linked List***	**Date:**

Aim:

Write a C program to implement a circular queue using linked lists.

The program should include the following functions:

- **enqueue(int element):** Insert an element into the circular queue.
- This function should update the rear pointer and maintain circular linking.
- **dequeue():** Delete an element from the circular queue.
- This function should remove the front node and adjust links, printing underflow if empty.
- **display():** Show all elements in the circular queue.
- This function should print elements by traversing nodes until it reaches the front again.
- **size():** Print the current number of elements in the queue.
- This function should count nodes in a circular manner starting from the front.
- **isEmpty():** Check whether the circular queue is empty.
- This function should print a message indicating whether the list has any nodes.

Your program should correctly perform all operations while maintaining circular linked list logic, ensuring that the last node always points back to the front.

Note:

- The driver code is provided; implement the logic in the file **CQueueOperationsLL.c**.
- Refer to the visible test cases to strictly match the input/output layout.

Source Code:

```
CQueueLL.c
```


ID: 24F11A05K1

Narayana Engineering College - Gudur

```
#include <stdlib.h>
#include <stdio.h>
#include "CQueueOperationsLL.c"
int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size
6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
}
```

CQueueOperationsLL.c

```
struct queue {
        int data;
        struct queue *next;
};
typedef struct queue *CircularQueue;
CircularQueue front = NULL, rear = NULL;
//complete the below dequeue() and enqueue() functions
void dequeue() {
        CircularQueue temp = NULL;
        if(front == NULL) {
                printf("Circular queue is underflow.\n");
        } else {
                temp = front;
                if(front == rear) {
                        front = rear = NULL;
                } else {
                        front = front -> next;
                        rear-> next = front;
        }
                printf("Deleted value = %d\n", temp -> data);
        }
}

void size() {
        int count =0;
        if(front == NULL) {
                printf("Circular queue size : 0\n");
                return;
        }
        CircularQueue temp = front;
        do {
                temp = temp -> next;
                count = count + 1;
        } while(temp != front);
        printf("Circular queue size : %d\n",count);
}

void isEmpty() {
        if(front == NULL ) {
                printf("Circular queue is empty.\n");
        } else {
                printf("Circular queue is not empty.\n");
        }
}

void enqueue(int element) {
```

```
        CircularQueue temp = NULL;
        temp = (CircularQueue)malloc(sizeof(struct queue));
        if (temp == NULL){
                printf("Circular queue is overflow.\n");
        } else{
                temp -> data = element;
                temp->next = NULL;
                if(front == NULL) {
                        front = temp;
                } else{
                        rear -> next = temp;
                }
                rear = temp;
                rear -> next = front;
                printf("Successfully inserted.\n");
        }




}

void display() {
        if(front == NULL) {
                printf("Circular queue is empty.\n");
        } else {
                CircularQueue temp = front;
                printf("Elements in the circular queue : ");
                do {
                        printf("%d ", temp -> data);
                        temp = temp -> next;
                } while(temp != front);
                printf("\n");
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :

1
Enter element :
15
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
16
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
17
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Elements in the circular queue : 15 16 17
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 15
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 16
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 17
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Circular queue is empty.

4
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Circular queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
4
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
143
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
153

1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
163
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
173
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Elements in the circular queue : 143 153 163 173
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 143
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 153
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
4
Circular queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
6

	Exp. Name: ***Infix Expression to Postfix and Evaluation***	**Date:**

Aim:

Write a C program that uses a stack to evaluate an infix expression and convert it to a postfix expression.

Input Format:

- The input is a string representing the infix mathematical expression.

Output Format:

- The first line of output represents the converted postfix notation of the infix expression.
- The second line of output represents the result of evaluating the postfix expression.

Note:

- Only single-digit positive integers and **+, -, *, /, %** operators are allowed.
- Refer to the visible test cases to strictly match the input/output layout.

Source Code:

```
EvaluateConverted.c
```


ID: 24F11A05K1
2024-2028-CSE-D
Narayana Engineering College - Gudur

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

#define MAX_SIZE 100
typedef struct {
char item[MAX_SIZE];
int top;
} Stack;
void push(Stack *s, char c) {
if (s->top == MAX_SIZE - 1) {
printf("Stack Overflow\n");
exit(EXIT_FAILURE); }
s->item[++(s->top)] = c; }
char pop(Stack *s) {
if (s->top == -1) {
printf("Stack Underflow\n");
exit(EXIT_FAILURE); }
return s->item[(s->top)--]; }
int is_operator(char c){
return (c == '+' || c == '-' || c == '*' || c == '/' || c == '%'); }
int precedence(char c) {
switch (c) {
case '+':
case '-':
return 1;
case '*':
case '/':
case '%':
return 2;
default:
return 0; }}
//Function to convert infix expression to postfix notation
void infixToPostfix(char *infix, char *postfix) {
Stack stack;
stack.top = -1;
int i = 0, j = 0;
while (infix[i] != '\0') {
        if (isdigit(infix[i])) {
postfix[j++] = infix[i++];
} else if (is_operator(infix[i])) {
while (stack.top != -1 && precedence(stack.item[stack.top]) >=
precedence(infix[i])) {
postfix[j++] = pop(&stack); }
push(&stack, infix[i++]);
} else if (infix[i] == '(') {
```

```
push(&stack, infix[i++]);
} else if (infix[i] ==')') {
while (stack.item[stack.top] != '(') {
postfix[j++] = pop(&stack); }
pop(&stack);
i++;
} else {
printf("Invalid character in infix expression\n");
exit(EXIT_FAILURE); }}
while (stack.top != -1) {
postfix[j++] = pop(&stack); }
postfix[j] = '\0'; }
// Function to evaluate a postfix expression
int evaluatePostfixExpression(char *postfix) {
Stack stack;
stack.top = -1;
int i = 0;
while (postfix[i] != '\0') {
if (isdigit(postfix[i])) {
push(&stack, postfix[i] - '0');
} else if (is_operator(postfix[i])) {
int operand2 = pop(&stack);
int operand1 = pop(&stack);
switch (postfix[i]) {
case '+':
push(&stack, operand1 + operand2);
break;
case '-':
push(&stack, operand1 - operand2);
break;
case '*':
push(&stack, operand1 * operand2);
break;
case '/':
push(&stack, operand1 / operand2);
break;
case '%':
push(&stack, operand1 / operand2);
break;
}
}
i++;
}
return pop(&stack);
}
```

```
int main() {
    char infixExpression[MAX_SIZE];
    char postfixExpression[MAX_SIZE];

    printf("Infix expression: ");
    scanf("%s", infixExpression);

    infixToPostfix(infixExpression, postfixExpression);

    printf("Postfix expression: %s\n", postfixExpression);

    int result = evaluatePostfixExpression(postfixExpression);

    printf("Result: %d\n", result);

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Infix expression:
2+3*4
Postfix expression: 234*+
Result: 14

Test Case - 2
User Output
Infix expression:
8%3+6*(2-1)
Postfix expression: 83%621-*+
Result: 8

	Exp. Name: ***Check whether the given string is palindrome or not using stack.***	**Date:**

Aim:

Write a C program to determine whether a given string is a palindrome. A palindrome is a string that reads the same forward and backward, ignoring case sensitivity.

Input Format:

The user is prompted to enter a string. The input string is limited to a maximum length of 99 characters (excluding the null terminator).

Output Format:

The program will display one of the following messages:

- "<input_string> is a palindrome" if the input string is a palindrome.
- "<input_string> is not a palindrome" if the input string is not a palindrome.

Source Code:

```
StringPalinUsingStack.c
```

```
/*
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

struct Stack {
    int top;
    char items[100];
};

// Write a function to initialize the stack


// Write a function to check if the stack is empty


// Write a function to push a character onto the stack


// Write a function to pop a character from the stack


// Write a function to check if a given string is a palindrome


int main() {
    char inputString[100];

    printf("String: ");
    scanf("%s", inputString);

    if (isPalindrome(inputString)) {
        printf("%s is a palindrome\n", inputString);
    } else {
        printf("%s is not a palindrome\n", inputString);
    }

    return 0;
}
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
struct Stack {
```

```
        int top;
        char items[100];
};
void initialize(struct Stack *stack) {
        stack->top = -1;
}
// Write a function to check if the stack is empty
int isEmpty(struct Stack *stack) {
        return stack->top == -1;
}
// Write a function to push a character onto the stack
void push(struct Stack *stack, char c) {
        if (stack->top == 100 - 1) {
                printf("Stack Overflow\n");
                exit(EXIT_FAILURE);
        }
        stack->items[++stack->top] = c;
}
// Write a function to pop a character from the stack
char pop(struct Stack *stack) {
        if (isEmpty(stack)) {
                printf("Stack Underflow\n");
                exit(EXIT_FAILURE);
        }
        return stack->items[stack->top--];
}// write a function to check if a given string is a palindrome
int isPalindrome(char *str) {
        struct Stack stack;
        initialize(&stack);
        int len = strlen(str);
        int i;
        for (i = 0; i < len / 2; i++) {
                push(&stack,tolower(str[i]));
        }
        if (len % 2 !=0) {
                len /= 2;
                i++;
        }
        while (str[i] != '\0') {
                char c = pop(&stack);
                if (c != str[i]) {
                        return 0;
                }
                i++;
        }
        return 1;
```

```
}
int main() {
        char inputString[100];
        printf("String: ");
        scanf("%s", inputString);
        if (isPalindrome(inputString)) {
                printf("%s is a palindrome\n", inputString);
        } else {
                printf("%s is not a palindrome\n", inputString);
        }
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
String:
Madam
Madam is a palindrome

Test Case - 2
User Output
String:
Aplha
Aplha is not a palindrome

	Exp. Name: ***Symmetry of a String using Stack***	**Date:**

Aim:

Implement a C program that uses a stack to determine whether a given string is symmetric. A symmetric string reads the same forward and backward, ignoring case sensitivity.

Input Format:

- The input is a single line of string prompts to enter: "String: " without spaces.

Output Format:

After processing, the program should print one of the following messages:

- "<input_string> is symmetric"
- "<input_string> is not symmetric"

The printed <input_string> must remain unchanged (case preserved).

Note:

- Perform case-insensitive comparison internally, but don't modify the original input string when printing.
- Maximum length of string: 99 characters.
- Use stack operations to check whether the string is symmetric.

Source Code:

```
StringSymmetry.c
```

```
/*#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

// Define a stack structure
struct Stack {
    int top;
    char items[100];
};

// Function to initialize the stack
void initializeStack(struct Stack* stack) {
    stack->top = -1;
}
int main() {
    char inputString[100];

    printf("String: ");
    scanf("%s", inputString);

    if (isSymmetricUsingStack(inputString)) {
        printf("%s is symmetric\n", inputString);
    } else {
        printf("%s is not symmetric\n", inputString);
    }

    return 0;
}*/
#include<stdio.h>
#include <stdlib.h>
#include<string.h>
#include<ctype.h>
#define MAX_SIZE 100
typedef struct {
int top;
char items[MAX_SIZE];
} Stack;
void push(Stack *s, char c) {
if (s->top == MAX_SIZE - 1) {
printf("Stack Overflow\n");
exit(EXIT_FAILURE);
} else {
s->items[++(s->top)] = c;
}
}
```

```
char pop(Stack *s) {
if (s->top == -1) {
printf("Stack Underflow\n");
exit(EXIT_FAILURE);
} else {
return s->items[(s->top)--];
}
}
int isSymmetric(char *str) {
Stack s;
s.top = -1;
int len = strlen(str);
for (int i = 0; i < len; i++) {
if (isalpha(str[i])) {
push(&s, tolower(str[i])); }
} for (int i = 0; i < len; i++){
if (isalpha(str[i])) {
char c = pop(&s);
if (tolower(str[i]) != c) {
return 0; // Not symmetric
}
}
}
return 1; // symmetric
}
int main() {
char str[MAX_SIZE];
// Test case 1
printf("String: ");
scanf("%s",str);
if (isSymmetric(str)) {
printf("%s is symmetric\n", str);
} else {
printf("%s is not symmetric\n", str);
}
return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output

String:
Emualator
Emualator is not symmetric

Test Case - 2
User Output
String:
Madam
Madam is symmetric

	Exp. Name: ***Graph traversals implementation - Depth First Search***	**Date:**

Aim:

Write a C Program to implement the DFS algorithm for a given graph.

Source Code:

GraphsDFS.c

```
#include<stdio.h>
#include<stdlib.h>
struct node{
struct node *next;
int vertex;
};
typedef struct node * GNODE;
GNODE graph[20];
int visited[20];
int n;
void DFS(int i) {
        GNODE p;
        printf("\n%d", i);
        p=graph[i];
        visited[i]=1;
        while(p!=NULL) {
                i=p->vertex;
                        if(!visited[i])
                                DFS(i);
                p=p->next;
        }
}
void main() {
        int N, E, i,s,d,v;
        GNODE q,p;
        printf("Enter the number of vertices : ");
        scanf("%d", &N);
        printf("Enter the number of edges : ");
        scanf("%d",&E);
        for(i = 1;i<=E; i++){
                printf("Enter source : ");
                scanf("%d", &s);
                printf("Enter destination : ");
                scanf("%d", &d);
                q=(GNODE)malloc(sizeof(struct node));
                q->vertex=d;
                q->next=NULL;
                if(graph[s]==NULL)
                        graph[s]=q;
                else{
                        p=graph [s];
                        while(p->next!=NULL)
                                p = p ->next;
                                p ->next=q;
                }
        }
```

```
        for(i=0;i<n;i++)
                visited[i]=0;
        printf("Enter Start Vertex for DFS : ");
        scanf("%d", &v);
        printf("DFS of graph : ");
        DFS(v);
        printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices :
6
Enter the number of edges :
7
Enter source :
1
Enter destination :
2
Enter source :
1
Enter destination :
4
Enter source :
4
Enter destination :
2
Enter source :
2
Enter destination :
3
Enter source :
4
Enter destination :
5
Enter source :

Enter destination :
3
Enter source :
3
Enter destination :
6
Enter Start Vertex for DFS :
1
DFS of graph :
1
2
3
6
4
5

	Exp. Name: ***Graph traversals implementation - Breadth First Search***	**Date:**

Aim:

Write a C program to implement BFS algorithm for a given graph

Source Code:

GraphsBFS.c

ID: 24F11A05K1

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 99
struct node{
struct node *next;
int vertex;
};
typedef struct node * GNODE;
GNODE graph[20];
int visited[20];
int queue[MAX],front=-1,rear = -1;
int n;
void insertQueue(int vertex){
if(rear == MAX-1)
printf("Queue Overflow.\n");
else{
if(front == -1)
front = 0;
rear = rear+1;
queue[rear]=vertex;
}
}
int isEmptyQueue(){
if(front == -1 || front > rear)
return 1;
else
return 0;
}
int deleteQueue(){
int deleteItem;
if(front == -1 || front > rear){
printf("Queue Underflow\n");
exit(1);
}
deleteItem = queue[front];
front=front+1;
return deleteItem;
}
void BFS(int v){
int w;
insertQueue(v);
while(!isEmptyQueue()){
v=deleteQueue( );
printf("\n%d",v);
visited[v]=1;
GNODE g=graph[v];
```

```
        for(;g!=NULL;g=g->next){
w=g->vertex;
if(visited[w]==0){
insertQueue(w);
visited[w]=1;
}
}
}
}
void main(){
int N,E,s,d,i,j,v;
GNODE p,q;
printf("Enter the number of vertices : ");
scanf("%d",&N);
printf("Enter the number of edges : ");
scanf("%d",&E);
for(i=1;i<=E;i++){
printf("Enter source : ");
scanf("%d",&s);
printf("Enter destination : ");
scanf("%d",&d);
q=(GNODE)malloc(sizeof(struct node));
q->vertex=d;
q->next=NULL;
if(graph[s]==NULL){
graph[s]=q;
}else{
p=graph[s];
while(p->next!=NULL)
p=p->next;
p->next=q;
}
}
for(i=1;i<=n;i++)
visited[i]=0;
printf("Enter Start Vertex for BFS : ");
scanf("%d",&v);
printf("BFS of graph : ");
BFS(v);
printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices :
5
Enter the number of edges :
5
Enter source :
1
Enter destination :
2
Enter source :
1
Enter destination :
4
Enter source :
4
Enter destination :
2
Enter source :
2
Enter destination :
3
Enter source :
4
Enter destination :
5
Enter Start Vertex for BFS :
1
BFS of graph :
1
2
4
3
5

	Exp. Name: ***Collision Resolution Techniques - Linear Probing***	**Date:**

ID: 24F11A05K1

2024-2028-CSE-D

Narayana Engineering College - Gudur

Aim:

Write a C program to implement a hash table using linear probing.

Your program should use a fixed hash table size and support the following operations:

- Insert an element
- Delete an element
- Search for an element
- Print the current contents of the hash table

The hash table should use linear probing to resolve collisions.

Hash Table Requirements:

1. Use the constant for table size as 10
2. Use an integer array **HashTable[SIZE]** where, empty slots must be initialized to -1.
3. **int hash(int x)** - Computes the hash index of x.
4. **insert(int x)** - Insert the element x into the hash table using linear probing.
   - If insertion is successful, print: "Successfully inserted.".
   - If the table is full after probing one full cycle, print: "Hash table is full."
5. **void deleteElement(int x) -** Deletes the element x from the hash table using linear probing.
   - If the element is found and deleted, print: "Successfully deleted."
   - If the element is not found after probing one full cycle, print: "Element not found. So cannot delete the element."
6. **void search(int x) -** Searches for the element x using linear probing.
   - If the element found, print: "Element found."
   - If the element not found, print: "Element not found."
7. **void print() -** Prints all occupied indices and corresponding values

Note:

- Driver code is provided in **HashingMain2.c**. You need to implement the functions in the appropriate sections of **HashingLinearProbing.c** to fulfill the program's requirements.
- Refer to visible test cases to strictly match with input/output layout.

Source Code:

```
HashingMain2.c
```

```
#include <stdio.h>
#include <stdlib.h>
#include "HashingLinearProbing.c"
int main() {
        int x, op, i = 0;
        for (i = 0; i < SIZE; i++)
                HashTable[i] = -1;
        while (1) {
                printf("1.Insert 2.Delete 3.Search 4.Print 5.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch (op) {
                        case 1: printf("Enter an element to be inserted
: ");
                                        scanf("%d", &x);
                                        insert(x);
                                        break;
                        case 2:
                                        printf("Enter an element to be
deleted : ");
                                        scanf("%d", &x);
                                        deleteElement(x);
                                        break;
                        case 3:
                                        printf("Enter an element to be
searched : ");
                                        scanf("%d", &x);
                                        search(x);
                                        break;
                        case 4:
                                        print();
                                        break;
                        case 5: exit(0);
                }
        }
}
```

HashingLinearProbing.c

```
#define SIZE 10
int HashTable[SIZE];
int hash(int x) {
        return x % SIZE;
}
void insert(int x) {
        int index, start;
        index=start=x%SIZE;
        while(1){
                if(HashTable[index]==-1){
                        HashTable[index]=x;
                        printf("Successfully inserted.\n");
                        return;
                }
                index=(index+1)%SIZE;
                if(index==start){
                        printf("Hash tablet is full cannot insert the
element.\n");
                        return;
                }
        }
}
void deleteElement(int x) {
        int index,start;
        index=start=x%SIZE;
        while(HashTable[index] != x){
                if(HashTable[index] == -1){
                        printf("Element not found cannot delete the
element.\n");
                        return;
                }
                index=(index + 1)%SIZE;
                if(index==start){
                        printf("Element not found cannot delete the
element.\n");
                        return;
                }
        }
        HashTable[index]=-1;
        printf("Successfully deleted.\n");
        }
void search(int x) {
        int index,start;
        index=start=x%SIZE;
        while(HashTable[index]!=x){
                if(HashTable[index]==-1){
```

```
                        printf("Element not found.\n");
                        return;
                }
                index=(index+1)%SIZE;
                if(index == start){
                        printf("Element not found.\n");
                        return;
                }
        }
        printf("Element found.\n");
}
void print() {
        for(int i=0;i<SIZE;i++){
                if(HashTable[i]!=-1){
                        printf("[%d]=>%d ",i,HashTable[i]);
                }
        }
        printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
11
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
22
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :

```
33
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
43
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
53
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[1]=>11 [2]=>22 [3]=>33 [4]=>43 [5]=>53
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
44
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[1]=>11 [2]=>22 [3]=>33 [4]=>43 [5]=>53 [6]=>44
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
3
Enter an element to be searched :
34
Element not found.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
2
Enter an element to be deleted :
33
Successfully deleted.
```

4
[1]=>11 [2]=>22 [4]=>43 [5]=>53 [6]=>44
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
5

Test Case - 2
User Output
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
10
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
20
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
30
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
40
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
50
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit

2
Enter an element to be deleted :
30
Successfully deleted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[0]=>10 [1]=>20 [3]=>40 [4]=>50
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
2
Enter an element to be deleted :
60
Element not found. So cannot delete the element.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
3
Enter an element to be searched :
20
Element found.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
2
Enter an element to be deleted :
70
Element not found. So cannot delete the element.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[0]=>10 [1]=>20 [3]=>40 [4]=>50
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
5

Test Case - 3
User Output
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1

10
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
20
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
30
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
40
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
50
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
60
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
70
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :

Enter an element to be inserted :
80
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
90
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
100
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
110
Hash table is full.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
5

	Exp. Name: ***Collision Resolution Techniques - Separate Chaining***	**Date:**

ID: 24F11A05K1

2024-2028-CSE-D

Narayana Engineering College - Gudur

Aim:

Write a C program to implement Hashing using Separate Chaining.

Your program should use a fixed hash table size and support the following operations:

- Insert an element
- Delete an element
- Search for an element
- Print the current contents of the hash table

The hash table should use separate chaining.

Hash Table Requirements:

1. Use the constant for table size as 10
2. **int hash(int x)** - Computes the hash index of$x$.
3. **struct node newNode(int x)** - Inserts the value x into the hash table using separate chaining. Insert at the beginning of the linked list in that bucket.
4. **void deleteElement(int key)** - Deletes key from the hash table.

Prints:

- "Successfully deleted." → when deleted
- "Element not found. So cannot delete." → if key does not exist

5. **void search(int x) -** Searches for x in the hash table.

Prints:

- "Element found." → if found
- "Element not found. So cannot delete." → if not found

6. **void print()** - Prints the entire hash table.

Note:

- Driver code is provided in **HashingMain4.c**. You need to implement the functions in the appropriate sections of **HashingSeperateChaining.c** to fulfill the program's requirements.
- Refer to visible test cases to strictly match with input/output layout.

Source Code:

HashingMain4.c

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include "HashingSeperateChaining.c"
int main() {
        int x, op, i=0;
        for(i=0;i<SIZE;i++)
                HashTable[i]=NULL;
        while(1) {
                printf("1.Insert 2.Delete 3.Search 4.Print 5.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1: printf("Enter an element to be inserted
: ");
                                        scanf("%d", &x);
                                        insert(x);
                                        break;
                        case 2:
                                        printf("Enter an element to be
deleted : ");
                                        scanf("%d", &x);
                                        deleteElement(x);
                                        break;
                        case 3:
                                        printf("Enter an element to be
searched : ");
                                        scanf("%d", &x);
                                        search(x);
                                        break;
                        case 4:
                                        print();
                                        break;
                        case 5: exit(0);
                }
        }
}
```

HashingSeperateChaining.c

```
#define SIZE 10

struct node {
        int data;
        struct node * next;
};
struct node * HashTable[SIZE];

int hash(int x) {
        return x % SIZE;
}

struct node * newNode(int x) {
        struct node * temp = (struct node *) malloc(sizeof(struct node
*));
        temp->data = x;
        temp->next = NULL;
        return temp;
}
void insert(int x) {
        int index = hash(x);
        struct node *new_node= newNode(x);
        if(HashTable [index] ==NULL) {
                HashTable[index] = new_node;
        }else{
                new_node->next = HashTable[index];
                HashTable[index] = new_node;
        }
}
void deleteElement(int x) {
        int index = hash(x);
        struct node * temp = HashTable[index];
        struct node * prev = NULL;
        while (temp != NULL && temp-> data != x) {
                prev = temp;
                temp = temp->next;
        }
        if (temp == NULL) {
                printf("Element not found. So cannot delete.\n");
                return;
        }
        if (prev == NULL) {
                HashTable[index] = temp->next;
        } else {
                prev->next = temp->next;
        }
```

```
        free(temp);
        printf("Successfully deleted.\n");
}
void search(int x) {
        int index = hash(x);
        struct node * temp = HashTable[index];
        while (temp != NULL) {
                if (temp->data == x) {
                        printf("Element found.\n");
                        return;
                }
                temp = temp->next;
        }
        printf("Element not found.\n");
}
void print() {
        for(int i = 0; i < SIZE; i++) {
                struct node * temp = HashTable[i];
                if(temp != NULL) {
                        printf("[%d]=> ", i);
                        while (temp != NULL){
                                printf("%d ", temp->data);
                                temp = temp->next;
                        }
                }
        }
        printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
113
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1

Enter an element to be inserted :
134
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
56
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
76
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[3]=> 113 [4]=> 134 [6]=> 76 56
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
2
Enter an element to be deleted :
76
Successfully deleted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
3
Enter an element to be searched :
44
Element not found.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[3]=> 113 [4]=> 134 [6]=> 56
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
5

Test Case - 2
User Output
1.Insert 2.Delete 3.Search 4.Print 5.Exit

```
1
Enter an element to be inserted :
3
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
4
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
5
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
3
Enter an element to be searched :
5
Element found.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
3
Enter an element to be searched :
8
Element not found.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[3]=> 3 [4]=> 4 [5]=> 5
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
2
Enter an element to be deleted :
3
Successfully deleted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
2
Enter an element to be deleted :
7
```

1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[4]=> 4 [5]=> 5
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
5

	Exp. Name: ***Simple Cache using Hashing***	**Date:**

Aim:

Write a C program to implement a simple cache using hashing with linear probing.

You are required to implement a cache of fixed size (10 entries) that stores integer keys and their associated string values. Collisions must be handled using linear probing.

The following functions must be implemented exactly as described:

1. **void initializeCache(struct Cache* cache) -** This function should prepare the cache before any operations are performed.
   - Set every key in the cache to -1 (meaning the slot is empty)
   - Set every value to an empty string ("")
   - Set **usedSlots** to 0 (no entries inserted yet)
2. **int hashFunction(int key) -** This function computes the index for a given key.
   - Return the value **key % CACHE_SIZE**
   - This ensures the key maps to a valid index between 0 and **CACHE_SIZE** - **1**
3. **void insert(struct Cache* cache, int key, const char* value)**
   - If the cache is full, print: "Cache is full".
   - Otherwise, insert the key-value pair using linear probing, and store the value.
4. **void retrieve(struct Cache* cache, int key) -** Retrieve the value for a given key.
   - If the key is found, print: "Value for key <key>: <value>"
   - If the key is not found, print: "Key <key> not found in the cache".

Menu - Driven Interface:

Your program should repeatedly display the following menu:

```
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
```

- If the user enter 1 for insert operation, prompt:

```
Key:
Value:
```

- If the user enter 2 for Key for retrieval operation, prompt:

```
Key for retrieval:
```

- If the user enter 3 for exit, it prints:

```
Exiting
```

- If the user enter invalid option, rather then 1-3, then it prints:

```
Invalid choice
```

Note:

- Your output format must match these statements exactly, including spacing and punctuation.
- Refer to the visible test cases for better understanding.

Cache.c

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

#define CACHE_SIZE 10

// Define a key-value pair structure
struct KeyValuePair {
    int key;
    char value[50];
};

// Define a cache structure
struct Cache {
    struct KeyValuePair table[CACHE_SIZE];
    int usedSlots;
};

int main() {
    struct Cache cache;
    initializeCache(&cache);

    int choice;
    int key;
    char value[50];

    do {
        printf("Cache Menu:\n");
        printf("1. Insert a key-value pair\n");
        printf("2. Retrieve value for a key\n");
        printf("3. Exit\n");
        printf("Choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Key: ");
                scanf("%d", &key);
                printf("Value: ");
                scanf("%s", value);
                insert(&cache, key, value);
                break;
            case 2:
                printf("Key for retrieval: ");
                scanf("%d", &key);
                retrieve(&cache, key);
```

```
                break;
            case 3:
                printf("Exiting\n");
                break;
            default:
                printf("Invalid choice\n");
        }

    } while (choice != 3);

    return 0;
}
*/
#include <stdio.h>
#include <stdbool.h>
#define MAX_CACHE_SIZE 100
// Structure to represent key-value pairs
typedef struct {
int key;
char value[100];
} KeyValuePair;
//Function prototype
void insertKeyValuePair(KeyValuePair cache[], int *cacheSize);
char*
retrieveValueForKey(KeyValuePair cache[], int cacheSize, int key);
int main() {
KeyValuePair cache[MAX_CACHE_SIZE];
int cacheSize = 0;
int choice;
bool running = true;
while (running) {
printf("Cache Menu:\n");
printf("1. Insert a key-value pair\n");
printf("2. Retrieve value for a key\n");
printf("3. Exit\n");
printf("Choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
insertKeyValuePair(cache, &cacheSize);
break;
case 2: {
int key;
printf("Key for retrieval: ");
scanf("%d", &key);
char* value = retrieveValueForKey(cache, cacheSize, key);
```

```
if (value != NULL)
printf("Value for key %d: %s\n", key, value);
else
printf("Key %d not found in the cache\n", key);
break;
}
case 3:
printf("Exiting\n");
running = false;
break;
default:
printf("Invalid choice\n");
break;
}
}
return 0;
}
void insertKeyValuePair(KeyValuePair cache[], int *cacheSize) {
if(*cacheSize >= MAX_CACHE_SIZE) {
printf("Cache is full\n");
return;
}
printf("Key: ");
scanf("%d", &cache[*cacheSize].key);
printf("Value: ");
scanf("%s", cache[*cacheSize].value);
(*cacheSize)++;
}
char* retrieveValueForKey(KeyValuePair cache[], int cacheSize, int key)
{
 for (int i = 0; i < cacheSize; ++i) {
 if (cache[i].key == key) {
 return cache[i].value;
 }
 }
 return NULL;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output

Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
2
Key for retrieval:
10
Key 10 not found in the cache
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
1
Key:
10
Value:
010
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
1
Key:
20
Value:
020
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
2
Key for retrieval:
30
Key 30 not found in the cache
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key

Choice:
303
Invalid choice
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
2
Key for retrieval:
30
Key 30 not found in the cache
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
10
Invalid choice
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
10
Invalid choice
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
2
Key for retrieval:
10
Value for key 10: 010
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
2

```
20
Value for key 20: 020
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
2
Key for retrieval:
10
Value for key 10: 010
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
3
Exiting
```

	Exp. Name: ***Binary Tree Traversals using Linked List - Recursive***	**Date:**

Aim:

Write a C program to implement Binary tree traversals using Linked list.
You have to complete the function inorder, preorder and postorder in Traversarls.c where parameters passed are the root reference's of the binary tree **T1.**

Input Format:

- The program prompts the user to enter the values prefixed by "Enter value :" until -1 is entered.

Output Format:

- The first line of the output displays inorder traversal.
- The second line of the output displays preorder traversal.
- The third line of the output displays postorder traversal.

Note:

- Refer to the sample test cases for better understanding regarding input and output formats.

Source Code:

```
TreeMain.c
```


ID: 24F11A05K1
2024-2028-CSE-D
Narayana Engineering College - Gudur

```
// Program for linked implementation of complete binary tree
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
#include<stdbool.h>
#include"TreeStructure.c"
#include"Traversals.c"


// For Queue Size
#define SIZE 100000



// A utility function to create a new Queue
struct Queue* createQueue(int size)
{
    struct Queue* queue = (struct Queue*) malloc(sizeof( struct Queue
));

    queue->front = queue->rear = -1;
    queue->size = size;

    queue->array = (struct node**) malloc
                   (queue->size * sizeof( struct node* ));

    int i;
    for (i = 0; i < size; ++i)
        queue->array[i] = NULL;

    return queue;
}

// Standard Queue Functions
int isEmpty(struct Queue* queue)
{
    return queue->front == -1;
}

int isFull(struct Queue* queue)
{ return queue->rear == queue->size - 1;

}

int hasOnlyOneItem(struct Queue* queue)
{  return queue->front == queue->rear;
```

```
}

void Enqueue(struct node *root, struct Queue* queue)
{
    if (isFull(queue))
        return;
    queue->array[++queue->rear] = root;
    if (isEmpty(queue))
        ++queue->front;
}
struct node* Dequeue(struct Queue* queue)
{
    if (isEmpty(queue))
        return NULL;
    struct node* temp = queue->array[queue->front];

    if (hasOnlyOneItem(queue))
        queue->front = queue->rear = -1;
    else
        ++queue->front;
    return temp;
}

struct node* getFront(struct Queue* queue)
{  return queue->array[queue->front]; }

// A utility function to check if a tree node
// has both left and right children
int hasBothChild(struct node* temp)
{
    return temp && temp->left && temp->right;
}

// Function to insert a new node in complete binary tree
void insert(struct node **root, int data, struct Queue* queue)
{
    // Create a new node for given data
    struct node *temp = newNode(data);
    // If the tree is empty, initialize the root with new node.
    if (!*root)
        *root = temp;
    else
    {
        // get the front node of the queue.
        struct node* front = getFront(queue);
```

```
        // If the left child of this front node doesn’t exist, set the
        // left child as the new node
        if (!front->left)
        {
                front->left = NULL;
            front->left = temp;
        }
        // If the right child of this front node doesn’t exist, set the
        // right child as the new node
        else if (!front->right)
        {
                front->right = NULL;
            front->right = temp;
        }
        // If the front node has both the left child and right child,
        // Dequeue() it.
        if (hasBothChild(front))
        {
                Dequeue(queue);
        }
    }
    // Enqueue() the new node for later insertions
            Enqueue(temp, queue);
}

// Standard level order traversal to test above function
void levelOrder(struct node* root)
{
    struct Queue* queue = createQueue(SIZE);
    Enqueue(root, queue);
    while (!isEmpty(queue))
    {
        struct node* temp = Dequeue(queue);
        printf("%d ", temp->data);
        if (temp->left)
            Enqueue(temp->left, queue);
        if (temp->right)
            Enqueue(temp->right, queue);
    }
}

// Driver program to test above functions
int main()
{
    struct node* T1 = NULL;
    struct Queue* queue1 = createQueue(SIZE);
```

```
    int ele;

    while(1){
       printf("Enter value : ");
       scanf("%d",&ele);
    if(ele==-1){
        break;
    }
    else{
           insert(&T1, ele, queue1);
       }
    }
    inorder(T1);
    printf("\n");
    preorder(T1);
    printf("\n");
    postorder(T1);
}
```

TreeStructure.c

```
// A tree node
struct node
{
    int data;
    struct node *right,*left;
    struct node *root;
};

// A queue node
struct Queue
{
    int front, rear;
    int size;
    struct node* *array;
};

// A utility function to create a new tree node
struct node* newNode(int data)
{
    struct node* temp = (struct node*) malloc(sizeof( struct node ));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}
```

Traversals.c

```
void inorder( struct node *ptr)
{
        if(ptr!=NULL) {
                inorder(ptr->left);
                printf("%d ",ptr->data);
                inorder(ptr->right);
        }
    // write your code here to compute
    // and print inorder traversal
}

void preorder( struct node *ptr){
        if(ptr != NULL)
        {
                printf("%d ", ptr->data);
                preorder (ptr->left);
                preorder (ptr->right);
        }
        // write your code here to compute
    // and print preorder traversal
}

void postorder(struct node *ptr)
{
        if(ptr != NULL) {
                postorder (ptr->left);
                postorder (ptr->right);
                printf("%d ", ptr->data);
        }
    // write your code here to compute
    // and print postorder traversal
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter value :
45
Enter value :

Enter value :
56
Enter value :
12
Enter value :
45
Enter value :
26
Enter value :
78
Enter value :
45
Enter value :
-1
45 12 89 45 45 26 56 78
45 89 12 45 45 56 26 78
45 12 45 89 26 78 56 45

Test Case - 2
User Output
Enter value :
15
Enter value :
12
Enter value :
14
Enter value :
18
Enter value :
-1
18 12 15 14
15 12 18 14
18 12 14 15

	Exp. Name: ***Write a C program to implement topological sort on a graph***	**Date:**

Aim:

Write a program to implement topological sort on a given graph.

You should write the program to match the sample input and outputs shown below:

Sample input and output:

```
Enter the number of vertices : 4
Enter the number of edges : 4
Enter source : 1
Enter destination : 2
Enter source : 2
Enter destination : 3
Enter source : 3
Enter destination : 4
Enter source : 1
Enter destination : 4
The topological order is:1 2 3 4
```

Source Code:

TopologicalSort2.c

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
int adj[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES];
int stack[MAX_VERTICES];
int top = -1;
void DFS(int v, int vertices) {
visited[v] = 1;
for (int i = 0; i < vertices; i++) {
if (adj[v][i] && !visited[i]) {
DFS(i, vertices);
}
}
stack[++top] = v;
}void topologicalSort(int vertices) {
for (int i = 0; i < vertices; i++) {
if (!visited[i]) {
DFS(i, vertices);
}
}
printf("The topological order is:");
while (top >= 0) {
printf("%d ", stack[top--] + 1);
}
printf("\n");
}
int main() {
int vertices, edges;
printf("Enter the number of vertices : ");
scanf("%d", &vertices);
printf("Enter the number of edges : ");
scanf("%d", &edges);
for (int i = 0; i < edges; i++) {
int source, destination;
printf("Enter source : ");
scanf("%d", &source);
printf("Enter destination : ");
scanf("%d", &destination);
adj[source - 1][destination - 1] = 1;
}
topologicalSort(vertices);
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices :
4
Enter the number of edges :
4
Enter source :
1
Enter destination :
4
Enter source :
2
Enter destination :
1
Enter source :
2
Enter destination :
3
Enter source :
4
Enter destination :
3
The topological order is:2 1 4 3