1)Single Linked List

CODE:-

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

/* For defining of the structure of a node */
struct node
{
    int info;
    struct node *link;
};

/* To create a linked list */
struct node *
create_linked_list(struct node *start)
{
    struct node *temp, *p;
    int n;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    start = NULL;
    if (n == 0)
        return start;
    for (int i = 0; i < n; i++)
    {
        temp = (struct node *)malloc(sizeof(struct node));
        printf("\nEnter the data for node %d: ", i + 1);
        scanf("%d", &temp->info);
        temp->link = NULL;
        if (start == NULL)
            start = temp;
        else
        {
            p = start;
            while (p->link != NULL)
                p = p->link;
            p->link = temp;
        }
    }
    return start;
};

/* To display the linked list */
void display_linked_list(struct node *start)
{
    struct node *p;
    if (start == NULL)
    {
        printf("Linked list is empty\n");
        return;
    }
    p = start;
    printf("Linked list is: \n");
    while (p->link != NULL)
    {
        printf("%d->", p->info);
        p = p->link;
    }
    printf("%d\n", p->info);
};

/* To count the number of nodes in the linked list */
int count_nodes(struct node *start)
{
    struct node *p;
    int count = 0;
    p = start;
    while (p != NULL)
    {
        count++;
        p = p->link;
    }
    return count;
};

/*To search for an element in the linked list */
void list_search(struct node *start)
{
    struct node *p = start;
    int pos = 1, item;
    printf("Enter the element to be searched.\n");
    scanf("%d", &item);
    while (p != NULL)
    {
        if (p->info == item)
        {
            printf("Item %d found at position %d \n", item, pos);
            return;
        }
        p = p->link;
        pos++;
    }
    printf("Item %d not found in list \n", item);
}

/* To insert a node at the beginning of the linked list */
struct node *
insert_at_beginning(struct node *start)
{
    struct node *temp;
    temp = (struct node *)malloc(sizeof(struct node));
    printf("Enter the data for the node: \n");
    scanf("%d", &temp->info);
    temp->link = start;
    start = temp;
    return start;
};

/* To insert a node at the end of the linked list */
struct node *
insert_at_end(struct node *start)
{
    struct node *temp, *p;
    temp = (struct node *)malloc(sizeof(struct node));
    printf("Enter the data for the node:\n ");
    scanf("%d", &temp->info);
    temp->link = NULL;
    p = start;
    while (p->link != NULL)
        p = p->link;
    p->link = temp;
    return start;
};

/* To insert a node at a given position in the linked list */
struct node *
insert_at_position(struct node *start)
{
    struct node *temp, *p;
    int pos, i;
    printf("Enter the position:\n ");
    scanf("%d", &pos);
    int count = count_nodes(start);
    if (pos > count + 1 || pos < 1)
    {
        printf("Invalid position\n");
        return start;
    }
    if (pos == 1)
        start = insert_at_beginning(start);
    else if (pos == count + 1)
        start = insert_at_end(start);
    else
    {
        temp = (struct node *)malloc(sizeof(struct node));
        printf("Enter the data for the node: \n");
        scanf("%d", &temp->info);
        p = start;
        for (i = 1; i < pos - 1; i++)
            p = p->link;
        temp->link = p->link;
        p->link = temp;
    }
    return start;
};
```

```
/*To add before in the linked list*/
struct node *addbefore(struct node *start)
{
    struct node *p, *tmp;
    if (start == NULL)
    {
        printf("List is empty.\n");
        return start;
    }
    int data, item;
    printf("Enter the element in LL to be inserted before and the data to be inserted.\n");
    scanf("%d%d", &item, &data);
    if (start->info == item)
    {
        tmp = (struct node *)malloc(sizeof(struct node));
        tmp->info = data;
        tmp->link = start;
        start = tmp;
        return start;
    }
    p = start;
    while (p->link != NULL)
    {
        if (p->link->info == item)
        {
            tmp = (struct node *)malloc(sizeof(struct node));
            tmp->info = data;
            tmp->link = p->link;
            p->link = tmp;
            return start;
        }
        p = p->link;
    }
    printf("Item %d not found in LL.\n", item);
    return start;
}

/*To add after in the linked list*/
struct node *addafter(struct node *start)
{
    struct node *p, *tmp;
    p = start;
    int data, item;
    printf("Enter the element in LL to be inserted after and the data to be inserted.\n");
    scanf("%d%d", &item, &data);
    while (p != NULL)
    {
        if (p->info == item)
        {
            tmp = (struct node *)malloc(sizeof(struct node));
            tmp->info = data;
            tmp->link = p->link;
            p->link = tmp;
            return start;
        }
        p = p->link;
    }
    printf("Item %d not found in LL.\n", item);
    return start;
}

/* To delete a node from the linked list */
struct node *del(struct node *start)
{
    struct node *tmp, *p;
    if (start == NULL)
    {
        printf("List is empty.\n");
        return start;
    }
    int data;
    printf("Enter the data for the node:\n");
    scanf("%d", &data);
    if (start->info == data)
    {
        tmp = start;
        start = start->link;
        free(tmp);
        return start;
    }
    p = start;
    while (p->link != NULL)
    {
        if (p->link->info == data)
        {
            tmp = p->link;
            p->link = tmp->link;
            free(tmp);
            return start;
        }
        p = p->link;
    }
    printf("Element %d not found in LL.\n", data);
    return start;
}

/* To reverse the linked list */
struct node *
reverse_linked_list(struct node *start)
{
    struct node *prev, *next, *p;
    p = start;
    prev = NULL;
    while (p != NULL)
    {
        next = p->link;
        p->link = prev;
        prev = p;
        p = next;
    }
    start = prev;
    return start;
};

int main()
{
    struct node *start = NULL;
    int choice;
    while (1)
    {
        printf("Enter 1 to create linked list.\n");
        printf("Enter 2 to display linked list.\n");
        printf("Enter 3 to count the number of nodes.\n");
        printf("Enter 4 to search for an element.\n");
        printf("Enter 5 to insert a node at the beginning.\n");
        printf("Enter 6 to insert a node at the end.\n");
        printf("Enter 7 to insert a node at a given position.\n");
        printf("Enter 8 to insert node before another node.\n");
        printf("Enter 9 to insert node after specified node.\n");
        printf("Enter 10 to delete a node.\n");
        printf("Enter 11 to reverse the linked list.\n");
        printf("Enter 12 to exit.\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            start = create_linked_list(start);
            break;
        case 2:
            display_linked_list(start);
            break;
        case 3:
            printf("Number of nodes in the linked list is: %d\n",
                   count_nodes(start));
            break;
        case 4:
            list_search(start);
            break;
        case 5:
            start = insert_at_beginning(start);
            break;
        case 6:
            start = insert_at_end(start);
            break;
        case 7:
            start = insert_at_position(start);
            break;
        case 8:
            start = addbefore(start);
            break;
        case 9:
            start = addafter(start);
            break;
        case 10:
            start = del(start);
            break;
        case 11:
            start = reverse_linked_list(start);
            break;
        case 12:
            exit(1);
        default:
            printf("Erroneous input.\n");
        }
    }
    return 0;
}
```

OUTPUT:-

```
[sorciermahep@fedora DS_Labs] $ cd "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/" && gcc --std=c17 2.c -o 2 && "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/"2
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 1
Enter the number of nodes: 5

Enter the data for node 1: 1

Enter the data for node 2: 3

Enter the data for node 3: 5

Enter the data for node 4: 7

Enter the data for node 5: 9
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 2
Linked list is:
1->3->5->7->9
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 3
Number of nodes in the linked list is: 5
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 4
Enter the element to be searched.
2
Item 2 not found in list
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 5
Enter the data for the node:
7
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 6
Enter the data for the node:
 11
```

```
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 7
Enter the position:
 2
Enter the data for the node:
9
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 2
Linked list is:
7->9->1->3->5->7->9->11
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 8
Enter the element in LL to be inserted before and the data to be inserted.
1
13
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 9
Enter the element in LL to be inserted after and the data to be inserted.
5
8
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 2
Linked list is:
7->9->13->1->3->5->8->7->9->11
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 10
Enter the data for the node:
3
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 2
Linked list is:
7->9->13->1->5->8->7->9->11
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 11
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 2
Linked list is:
11->9->7->8->5->1->13->9->7
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to count the number of nodes.
Enter 4 to search for an element.
Enter 5 to insert a node at the beginning.
Enter 6 to insert a node at the end.
Enter 7 to insert a node at a given position.
Enter 8 to insert node before another node.
Enter 9 to insert node after specified node.
Enter 10 to delete a node.
Enter 11 to reverse the linked list.
Enter 12 to exit.
Enter your choice: 12
```

2)Stacks

CODE:-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define MAX 10000
char s1[MAX], s2[MAX];
int top = -1, nums = 0;
void push(char[], char);
char pop(char[]);
int isFull();
void parentheses(char[]);
void dec_to_base(int, int);
void palicheck(char[]);
void rev_string(char[]);
int isEmpty();
int main()
{

    int choice, num, base;
    char c;
    while (1)
    {
        memset(s1, '\0', MAX);
        memset(s2, '\0', MAX);
        top = -1;
        printf("Enter 1 for parentheses checking.\n");
        printf("Enter 2 for reversal of string.\n");
        printf("Enter 3 for palindrome checking.\n");
        printf("Enter 4 for decimal to base conversion.\n");
        printf("Enter 5 to exit.\n");
        scanf("%d", &choice);
        getchar();
        if (choice >= 1 && choice < 6)
        {
            if (choice == 1 || choice == 2 || choice == 3)
            {
                printf("Enter the string.\n");
                while ((c = getchar()) != 10)
                    push(s1, c);
            }
            else if (choice == 4)
            {
                printf("Enter the decimal number and the base to be converted.\n");
                scanf("%d%d", &num, &base);
            }
        }
        switch (choice)
        {
        case 1:
            parentheses(s1);
            break;
        case 2:
            rev_string(s1);
            break;
        case 3:
            palicheck(s1);
            break;
        case 4:
            dec_to_base(num, base);
            printf("The converted equivalent is : ");
            for (int i = 0; i < nums; i++)
                printf("%c", pop(s1));
            printf("\n");
            break;
        case 5:
            exit(1);
        default:
            printf("Invalid input.\n");
        }
    }
    return 0;
}
int isFull()
{
    if (top == MAX - 1)
        return 1;
    else
        return 0;
}
int isEmpty()
{
    if (top == -1)
        return 1;
    else
        return 0;
}
void push(char a[], char item)
{
    if (isFull())
    {
        printf("Stack Overflow\n");
        exit(1);
    }
    a[++top] = item;
}
char pop(char a[])
{
    char item;
    if (isEmpty())
    {
        printf("Stack Underflow.\n");
        exit(1);
    }
    else if (top > -1)
    {
        return a[top--];
    }
}
void parentheses(char a[])
{
    char ch;
    int flag = 0, brack_count = 0;
    for (int i = 0; a[i] != '\0'; i++)
    {
        ch = a[i];
        switch (ch)
        {
        case '(':
        case '[':
        case '{':
            push(s1, ch);
            break;
        case ')':
            if ('(' == pop(s1))
                brack_count++;
            else
                flag = 1;
            break;
        case ']':
            if ('[' == pop(s1))
                brack_count++;
            else
                flag = 1;
            break;
        case '}':
            if ('{' == pop(s1))
                brack_count++;
            else
                flag = 1;
            break;
        }
    }
    if (flag == 0)
        printf("%d pairs of parentheses matched.\n", brack_count);
    else if (flag == 1)
        printf("There was parentheses mismatch.\n");
}
void dec_to_base(int n, int base)
{
    if (n > 0)
    {
        int m = n % base;
        if (m <= 9)
```

```
            push(s1, m + 48);
        else
            push(s1, m - 10 + 'A');
        nums++;
        dec_to_base(n / base, base);
    }
}
void rev_string(char a[])
{
    int k;
    s1[top + 1] = '\0';
    for (k = 0; k < strlen(a); k++)
        s2[k] = pop(a);
    s2[k] = '\0';
    printf("The reversed string is : ");
    puts(s2);
}
void palicheck(char a[])
{
    int flag = 0, k;
    s1[top + 1] = '\0';
    for (k = 0; k < strlen(a); k++)
        s2[k] = pop(a);
    s2[k] = '\0';
    for (int i = 0; s1[i] != '\0'; i++)
    {
        if (a[i] != s2[i])
        {
            flag = 1;
            break;
        }
    }
    if (flag == 0)
        printf("The string is a palindrome.\n");
    else
        printf("The string is not a palindrome.\n");
}
```

OUTPUT:-

```
[sorciermahep@fedora DS_Labs] $ cd "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/" && gcc --std=c17 4.c -o 4 && "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/"4
Enter 1 for parentheses checking.
Enter 2 for reversal of string.
Enter 3 for palindrome checking.
Enter 4 for decimal to base conversion.
Enter 5 to exit.
1
Enter the string.
{[(])}
There was parentheses mismatch.
Enter 1 for parentheses checking.
Enter 2 for reversal of string.
Enter 3 for palindrome checking.
Enter 4 for decimal to base conversion.
Enter 5 to exit.
1
Enter the string.
{[()]}
3 pairs of parentheses matched.
Enter 1 for parentheses checking.
Enter 2 for reversal of string.
Enter 3 for palindrome checking.
Enter 4 for decimal to base conversion.
Enter 5 to exit.
2
Enter the string.
Hello There
The reversed string is : erehT olleH
Enter 1 for parentheses checking.
Enter 2 for reversal of string.
Enter 3 for palindrome checking.
Enter 4 for decimal to base conversion.
Enter 5 to exit.
3
Enter the string.
reviver
The string is a palindrome.
Enter 1 for parentheses checking.
Enter 2 for reversal of string.
Enter 3 for palindrome checking.
Enter 4 for decimal to base conversion.
Enter 5 to exit.
3
Enter the string.
renew
The string is not a palindrome.
Enter 1 for parentheses checking.
Enter 2 for reversal of string.
Enter 3 for palindrome checking.
Enter 4 for decimal to base conversion.
Enter 5 to exit.
4
Enter the decimal number and the base to be converted.
16
16
The converted equivalent is : 10
Enter 1 for parentheses checking.
Enter 2 for reversal of string.
Enter 3 for palindrome checking.
Enter 4 for decimal to base conversion.
Enter 5 to exit.
5
```

3)Queue

CODE:-

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *front = NULL;
struct node *rear = NULL;
void insert(int);
void del();
void display();
void peek();
void insert(int data)
{
    struct node *temp;
    temp = (struct node *)malloc(sizeof(struct node));
    temp->data = data;
    temp->next = NULL;
    if (front == NULL)
        front = rear = temp;
    else
    {
        rear->next = temp;
        rear = temp;
    }
}
void del()
{
    if (front == NULL)
    {
        printf("Queue is empty.\n\n");
    }
    else if (front == rear)
    {
        printf("Deleted %d.\n", front->data);
        free(front);
        front = rear = NULL;
    }
    else
    {
        struct node *temp = front;
        printf("Deleted %d.\n", front->data);
        front = front->next;
        free(temp);
    }
}
void display()
{
    if (front == NULL)
    {
        printf("Empty Queue.\n");
        return;
    }
    struct node *temp = front;
    printf("Queue contents are : \n");
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
void peek()
{
    if (front == NULL)
    {
        printf("Empty Queue.\n");
        return;
    }
    else
        printf("The front element is : %d.\n", front->data);
    printf("\n");
}
int main()
{
    int choice, data;
    while (1)
    {
        printf("\n1:Insert \n2:Delete \n3:Display\n4:Peek\n5:Exit\n");
        printf("Enter your choice.\n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter the data :\n");
            scanf("%d", &data);
            insert(data);
            break;
        case 2:
            del();
            break;
        case 3:
            display();
            break;
        case 4:
            peek();
            break;
        case 5:
            exit(1);
        default:
            printf("Erroneous input.\n");
            break;
        }
    }
    return 0;
}
```

OUTPUT:-

```
[sorciermahep@fedora DS_Labs] $ cd "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/" && gcc --std=c17 6.c -o 6 && "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/"6

1:Insert
2:Delete
3:Display
4:Peek
5:Exit
Enter your choice.
1
Enter the data :
3

1:Insert
2:Delete
3:Display
4:Peek
5:Exit
Enter your choice.
1
Enter the data :
5

1:Insert
2:Delete
3:Display
4:Peek
5:Exit
Enter your choice.
1
Enter the data :
7

1:Insert
2:Delete
3:Display
4:Peek
5:Exit
Enter your choice.
1
Enter the data :
9

1:Insert
2:Delete
3:Display
4:Peek
5:Exit
Enter your choice.
```

```
3
Queue contents are :
3 5 7 9

1:Insert
2:Delete
3:Display
4:Peek
5:Exit
Enter your choice.
4
The front element is : 3.


1:Insert
2:Delete
3:Display
4:Peek
5:Exit
Enter your choice.
2
Deleted 3.

1:Insert
2:Delete
3:Display
4:Peek
5:Exit
Enter your choice.
3
Queue contents are :
5 7 9

1:Insert
2:Delete
3:Display
4:Peek
5:Exit
Enter your choice.
4
The front element is : 5.


1:Insert
2:Delete
3:Display
4:Peek
5:Exit
Enter your choice.
5
```

4)Circular Queue

CODE:-

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
int cqueue_arr[MAX];
int rear = -1;
int front = -1;
void insert(int item);
int del();
int peek();
int isFull();
int isEmpty();
void display();
int main()
{

    int choice, item;
    while (1)
    {

        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Display the element at the front.\n");
        printf("4.Display all the elements of the queue.\n");
        printf("5.Quit.\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:

            printf("Enter the element.\n");
            scanf("%d", &item);
            insert(item);
            break;
        case 2:

            item = del();
            printf("Deleted item is : %d\n ", item);
            break;
        case 3:

            printf("Item at the front is : %d\n ", peek());
            break;
        case 4:

            display();
            break;
        case 5:
            exit(1);
        default:

            printf("Wrong choice\n");
        }
    }
    return 0;
}

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

int isFull()
{
    if ((front == 0 && rear == MAX - 1) || (front == rear + 1))
        return 1;
    else
        return 0;
}

int del()
{
    int item;
    if (isEmpty())
    {
        printf("Circular Queue Underflow\n");
        exit(1);
    }
    item = cqueue_arr[front];
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else if (front == MAX - 1)
        front = 0;
    else
        front = front + 1;
    return item;
}
```

```
int isEmpty()
{
   if (front == -1)
      return 1;
   else
      return 0;
}

int peek()
{
   if (isEmpty())
   {
      printf("Circular Queue Underflow\n");
      exit(1);
   }
   return cqueue_arr[front];
}

void display()
{
   int i;
   if (isEmpty())
   {
      printf("Circular Queue is empty\n");
      return;
   }
   printf("Queue is :\n");
   i = front;
   if (front <= rear)
   {
      while (i <= rear)
         printf("%d ", cqueue_arr[i++]);
   }
   else
   {
      while (i <= MAX - 1)
         printf("%d ", cqueue_arr[i++]);
      i = 0;
      while (i <= rear)
         printf("%d ", cqueue_arr[i++]);
   }
   printf("\n");
}
```

OUTPUT:-

```
[sorciermahep@fedora DS_Labs] $ cd "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/" && gcc --std=c17 7.c -o 7 && "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/"7
1.Insert
2.Delete
3.Display the element at the front.
4.Display all the elements of the queue.
5.Quit.
Enter your choice : 1
Enter the element.
3
1.Insert
2.Delete
3.Display the element at the front.
4.Display all the elements of the queue.
5.Quit.
Enter your choice : 1
Enter the element.
5
1.Insert
2.Delete
3.Display the element at the front.
4.Display all the elements of the queue.
5.Quit.
Enter your choice : 1
Enter the element.
7
1.Insert
2.Delete
3.Display the element at the front.
4.Display all the elements of the queue.
5.Quit.
Enter your choice : 1
Enter the element.
9
1.Insert
2.Delete
3.Display the element at the front.
4.Display all the elements of the queue.
5.Quit.
Enter your choice : 4
Queue is :
3 5 7 9
1.Insert
2.Delete
3.Display the element at the front.
4.Display all the elements of the queue.
5.Quit.
Enter your choice : 3
Item at the front is : 3
```

```
 1.Insert
2.Delete
3.Display the element at the front.
4.Display all the elements of the queue.
5.Quit.
Enter your choice : 2
Deleted item is : 3
 1.Insert
2.Delete
3.Display the element at the front.
4.Display all the elements of the queue.
5.Quit.
Enter your choice : 4
Queue is :
5 7 9
1.Insert
2.Delete
3.Display the element at the front.
4.Display all the elements of the queue.
5.Quit.
Enter your choice : 3
Item at the front is : 5
 1.Insert
2.Delete
3.Display the element at the front.
4.Display all the elements of the queue.
5.Quit.
Enter your choice : 5
```

5)Doubly Linked List

CODE:-

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

node *create_linked_list(node *start)
{
    node *temp, *p;
    int n;
    printf("Enter the number of nodes.\n");
    scanf("%d", &n);
    start = NULL;
    if (n == 0)
        return start;
    for (int i = 0; i < n; i++)
    {
        temp = (node *)malloc(sizeof(node));
        printf("\nEnter the data for node %d: ", i + 1);
        scanf("%d", &temp->data);
        temp->next = NULL;
        if (start == NULL)
        {
            temp->prev = NULL;
            start = temp;
        }
        else
        {
            p = start;
            while (p->next != NULL)
                p = p->next;
            temp->prev = p;
            p->next = temp;
        }
    }
    return start;
}

void display_linked_list(node *start)
{
    node *p;
    if (start == NULL)
    {
        printf("Linked list is empty\n");
        return;
    }
    p = start;
    printf("Linked list is: \n");
    while (p->next != NULL)
    {
        printf("%d->", p->data);
        p = p->next;
    }
    printf("%d\n", p->data);
}

void list_search(node *start)
{
    node *p = start;
    int pos = 1, item;
    printf("Enter the element to be searched.\n");
    scanf("%d", &item);
    while (p != NULL)
    {
        if (p->data == item)
        {
            printf("Item %d found at position %d \n", item, pos);
            return;
        }
        p = p->next;
        pos++;
    }
    printf("Item %d not found in list \n", item);
}

int count_nodes(struct node *start)
{
    struct node *p;
    int count = 0;
    p = start;
    while (p != NULL)
    {
        count++;
        p = p->next;
    }
    return count;
}

node *add_to_empty(node *start)
{
    node *temp;
    int item;
    if (start != NULL)
    {
        printf("List is not empty.\n");
        return start;
    }
    printf("Enter the data to be inserted.\n");
    scanf("%d", &item);
    temp = (node *)malloc(sizeof(node));
    temp->data = item;
    temp->prev = temp->next = NULL;
    start = temp;
    return start;
}

node *add_at_beginning(node *start)
{
    node *temp;
    int item;
    if (start == NULL)
    {
        printf("List is empty.\n");
        return start;
    }
    printf("Enter the data to be inserted.\n");
    scanf("%d", &item);
    temp = (node *)malloc(sizeof(node));
    temp->data = item;
    temp->prev = NULL;
    temp->next = start;
    start = temp;
    return start;
}

node *add_at_end(node *start)
{
    node *temp, *p;
    int item;
    if (start == NULL)
    {
        printf("List is empty.\n");
        return start;
    }
    printf("Enter the data to be inserted.\n");
    scanf("%d", &item);
    temp = (node *)malloc(sizeof(node));
    temp->data = item;
    temp->next = NULL;
    p = start;
    while (p->next != NULL)
        p = p->next;
    p->next = temp;
    temp->prev = p;
    return start;
}

node *add_before(node *start)
{
    node *temp, *p;
    int elem, item;
    if (start == NULL)
    {
        printf("List is empty.\n");
        return start;
    }
    printf("Enter the node value and the data to be inserted.\n");
    scanf("%d%d", &elem, &item);
    if (start->data == elem)
    {
        temp = (node *)malloc(sizeof(node));
        temp->data = item;
        temp->prev = NULL;
        temp->next = start;
        start->prev = temp;
        start = temp;
        return start;
    }
    p = start;
    while (p != NULL)
    {
        if (p->data == elem)
        {
            temp = (node *)malloc(sizeof(node));
            temp->data = item;
            temp->prev = p->prev;
            temp->next = p;
            p->prev->next = temp;
            p->prev = temp;
            return start;
        }
        p = p->next;
    }
    printf("Entered item not found in list.\n");
    return start;
}

node *add_after(node *start)
{
    node *temp, *p;
    int elem, item;
    if (start == NULL)
    {
        printf("List is empty.\n");
        return start;
    }
    printf("Enter the node value and the data to be inserted.\n");
    scanf("%d%d", &elem, &item);
    p = start;
    while (p != NULL)
    {
        if (p->data == elem)
        {
            temp = (node *)malloc(sizeof(node));
            temp->data = item;
            temp->prev = p;
            temp->next = p->next;
            p->next = temp;
            p->next->prev = temp;
            return start;
        }
        p = p->next;
    }
    printf("Entered item not found in list.\n");
    return start;
}

node *add_at_position(node *start)
{
    node *temp, *p;
    int posn, item;
    printf("Enter the position.\n");
    scanf("%d", &posn);
    if (start == NULL)
    {
        if (posn == 1)
        {
            start = add_to_empty(start);
            return start;
        }
        else
        {
            printf("Empty list.\n");
            return start;
        }
    }
    if (posn == 1)
    {
        start = add_at_beginning(start);
        return start;
    }
    else if (posn == count_nodes(start) + 1)
    {
        start = add_at_end(start);
        return start;
    }
    else
    {
        printf("Enter the data to be added.\n");
        scanf("%d", &item);
        temp = (node *)malloc(sizeof(node));
        temp->data = item;
        p = start;
        while (posn >= 1)
        {
            if (posn == 1)
            {
                temp->next = p;
                temp->prev = p->prev;
                p->prev->next = temp;
                p->next->prev = temp;
            }
            posn--;
            p = p->next;
        }
        return start;
    }
}

struct node *del(struct node *start)
{

    struct node *tmp;
    if (start == NULL)
    {
        printf("List is empty.\n");
        return start;
    }
    int data;
    printf("Enter the data to be deleted.\n");
    scanf("%d", &data);
    if (start->next == NULL)
    {
        if (start->data == data)
        {
            tmp = start;
            start = NULL;
            free(tmp);
            return start;
        }
        else
        {
            printf("Element %d not found in LL.\n", data);
            return start;
        }
    }
    if (start->data == data)
    {
        tmp = start;
        start = start->next;
        start->prev = NULL;
        free(tmp);
        return start;
    }
    tmp = start->next;
    while (tmp->next != NULL)
    {
        if (tmp->data == data)
        {
            tmp->prev->next = tmp->next;
            tmp->next->prev = tmp->prev;
            free(tmp);
            return start;
        }
        tmp = tmp->next;
    }
    if (tmp->data == data)
    {
        tmp->prev->next = NULL;
        free(tmp);
        return start;
    }
    printf("Element %d not found in LL.\n", data);
    return start;
}

node *reverse_linked_list(node *start)
{
    node *p1, *p2;
    p1 = start;
    p2 = p1->next;
    p1->prev = p2;
    p1->next = NULL;
    while (p2 != NULL)
    {
        p2->prev = p2->next;
        p2->next = p1;
        p1 = p2;
        p2 = p2->prev;
    }
    start = p1;
    return start;
}

int main()
{
    node *start = NULL;
    int choice;
    while (1)
    {
        printf("Enter 1 to create linked list.\n");
        printf("Enter 2 to display linked list.\n");
        printf("Enter 3 to search for an element.\n");
        printf("Enter 4 to count the number of nodes.\n");
        printf("Enter 5 to add to empty.\n");
        printf("Enter 6 to add at beginning.\n");
        printf("Enter 7 to add at end.\n");
        printf("Enter 8 to add before a node.\n");
        printf("Enter 9 to add after a node.\n");
        printf("Enter 10 to add at a position.\n");
        printf("Enter 11 to delete a node.\n");
        printf("Enter 12 to reverse the linked list.\n");
        printf("Enter 13 to exit.\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            start = create_linked_list(start);
            break;
        case 2:
            display_linked_list(start);
            break;
        case 3:
            list_search(start);
            break;
        case 4:
            printf("Number of nodes in the linked list is: %d\n",
                   count_nodes(start));
            break;
        case 5:
            start = add_to_empty(start);
            break;
        case 6:
            start = add_at_beginning(start);
            break;
        case 7:
            start = add_at_end(start);
            break;
        case 8:
            start = add_before(start);
            break;
        case 9:
            start = add_after(start);
            break;
        case 10:
            start = add_at_position(start);
            break;
        case 11:
            start = del(start);
            break;
        case 12:
            start = reverse_linked_list(start);
            break;
        case 13:
            exit(1);
        default:
            printf("Erroneous input.\n");
        }
```

```
    }
    return 0;
}
```

OUTPUT:-

```
[sorciermahep@fedora DS_Labs] $ cd "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/" && gcc --std=c17 9.c -o 9 --no-warnings &&
"/home/sorciermahep/Desktop/Mahendra Priolkar/C/DS_Labs/"9
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 1
Enter the number of nodes.
5

Enter the data for node 1: 1

Enter the data for node 2: 3

Enter the data for node 3: 5

Enter the data for node 4: 7

Enter the data for node 5: 9
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 2
Linked list is:
1->3->5->7->9
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 3
Enter the element to be searched.
2
Item 2 not found in list
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 4
Number of nodes in the linked list is: 5
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 5
List is not empty.
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 6
Enter the data to be inserted.
2
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 7
Enter the data to be inserted.
13
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 2
Linked list is:
2->1->3->5->7->9->13
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 8
Enter the node value and the data to be inserted.
3
15
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 9
Enter the node value and the data to be inserted.
7
17
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 10
Enter the position.
4
Enter the data to be added.
20
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 2
```

```
Linked list is:
2->1->15->20->3->5->7->17->9->13
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 11
Enter the data to be deleted.
3
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 2
Linked list is:
2->1->15->20->5->7->17->9->13
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 12
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 2
Linked list is:
13->9->17->7->5->20->15->1->2
Enter 1 to create linked list.
Enter 2 to display linked list.
Enter 3 to search for an element.
Enter 4 to count the number of nodes.
Enter 5 to add to empty.
Enter 6 to add at beginning.
Enter 7 to add at end.
Enter 8 to add before a node.
Enter 9 to add after a node.
Enter 10 to add at a position.
Enter 11 to delete a node.
Enter 12 to reverse the linked list.
Enter 13 to exit.
Enter your choice: 13
```

6)Circular Linked List

CODE:-

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node node;
struct node
{
    int info;
    node *link;
};

node *create_list(node *last)
{
    node *temp, *p;
    int n;
    printf("Enter the number of nodes.\n");
    scanf("%d", &n);
    if (n == 0)
        return last;
    printf("Enter the data.\n");
    for (int i = 0; i < n; i++)
    {
        temp = (node *)malloc(sizeof(node));
        scanf("%d", &temp->info);
        temp->link = NULL;
        if (last == NULL)
        {
            last = temp;
            last->link = last;
        }
        else
        {
            temp->link = last->link;
            last->link = temp;
            last = temp;
        }
    }
    return last;
}

node *del(node *last, int data)
{
    struct node *tmp, *p;
    if (last->link == last && last->info == data)
    {
        tmp = last;
        last = NULL;
        free(tmp);
        return last;
    }
    if (last->link->info == data)
    {
        tmp = last->link;
        last->link = tmp->link;
        free(tmp);
        return last;
    }
    p = last->link;
    while (p->link != last)
    {
        if (p->link->info == data)
        {
            tmp = p->link;
            p->link = tmp->link;
            free(tmp);
            return last;
        }
        p = p->link;
    }
    if (last->info == data)
    {
        tmp = last;
        p->link = last->link;
        last = p;
        free(tmp);
        return last;
    }
    return last;
}

void display_list(node *last)
{
    node *p;
    if (last == NULL)
    {
        printf("The circular linked list is empty.\n");
        return;
    }
    p = last->link;
    printf("The elements of the circular linked list are : ");
    do
    {
        printf("%d ", p->info);
        p = p->link;
    } while (p != last->link);
    printf("\n");
}

node *addtoempty(node *last, int data)
{
    struct node *tmp;
    tmp = (struct node *)malloc(sizeof(struct node));
    tmp->info = data;
    last = tmp;
    last->link = last;
    return last;
}

node *addatbeg(node *last, int data)
{
    node *tmp;
    tmp = (node *)malloc(sizeof(node));
    tmp->info = data;
    tmp->link = last->link;
    last->link = tmp;
    return last;
}
node *addatend(node *last, int data)
{
    node *tmp;
    tmp = (node *)malloc(sizeof(node));
    tmp->info = data;
    tmp->link = last->link;
    last->link = tmp;
    last = tmp;
    return last;
}

void listsplit(node *last, node **last1, node **last2)
{
    node *temp = last->link;
    int count = 0;
    do
    {
        if (count % 2 == 0 && count == 0)
            *last1 = addtoempty(*last1, temp->info);
        else if (count % 2 == 0)
            *last1 = addatend(*last1, temp->info);
        else if (count % 2 == 1 && count == 1)
            *last2 = addtoempty(*last2, temp->info);
        else if (count % 2 == 1)
            *last2 = addatend(*last2, temp->info);
        temp = temp->link;
        count++;
    } while (temp != last->link);
}

int main()
{
    int ch, elem;
    struct node *last = NULL, *last1 = NULL, *last2 = NULL;
    while (1)
    {
        printf("\n1.Create list.\n");
        printf("2.Add at beginning.\n");
        printf("3.Add at end.\n");
        printf("4.Display.\n");
        printf("5.Delete.\n");
        printf("6.Split.\n");
        printf("7.Exit.\n");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            last = create_list(last);
            break;
        case 2:
            printf("Enter the element to be added.\n");
            scanf("%d", &elem);
            last = addatbeg(last, elem);
            break;
        case 3:
            printf("Enter the element to be added.\n");
            scanf("%d", &elem);
            last = addatend(last, elem);
            break;
        case 4:
            display_list(last);
            break;
        case 5:
            printf("Enter the element to be added.\n");
            scanf("%d", &elem);
            last = del(last, elem);
            break;
        case 6:
            listsplit(last, &last1, &last2);
            printf("\nEven list is:\n");
            display_list(last1);
            printf("\nOdd list is:\n");
            display_list(last2);
            break;
        case 7:
            exit(1);
        default:
            printf("Erroneous input.\n");
        }
    }
    return 0;
}
```

OUTPUT:-

```
[sorciermahep@fedora DS_Labs] $ cd "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/" && gcc --std=c17 21.c -o 21 && "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/"21

1.Create list.
2.Add at beginning.
3.Add at end.
4.Display.
5.Delete.
6.Split.
7.Exit.
1
Enter the number of nodes.
5
Enter the data.
1 2 5 7 9

1.Create list.
2.Add at beginning.
3.Add at end.
4.Display.
5.Delete.
6.Split.
7.Exit.
4
The elements of the circular linked list are : 1 2 5 7 9

1.Create list.
2.Add at beginning.
3.Add at end.
4.Display.
5.Delete.
6.Split.
7.Exit.
2
Enter the element to be added.
11

1.Create list.
2.Add at beginning.
3.Add at end.
4.Display.
5.Delete.
6.Split.
7.Exit.
3
Enter the element to be added.
13
```

```
1.Create list.
2.Add at beginning.
3.Add at end.
4.Display.
5.Delete.
6.Split.
7.Exit.
4
The elements of the circular linked list are : 11 1 2 5 7 9 13

1.Create list.
2.Add at beginning.
3.Add at end.
4.Display.
5.Delete.
6.Split.
7.Exit.
5
Enter the element to be added.
2

1.Create list.
2.Add at beginning.
3.Add at end.
4.Display.
5.Delete.
6.Split.
7.Exit.
4
The elements of the circular linked list are : 11 1 5 7 9 13

1.Create list.
2.Add at beginning.
3.Add at end.
4.Display.
5.Delete.
6.Split.
7.Exit.
6

Even list is:
The elements of the circular linked list are : 11 5 9

Odd list is:
The elements of the circular linked list are : 1 7 13

1.Create list.
2.Add at beginning.
3.Add at end.
4.Display.
5.Delete.
6.Split.
```

```
7.Exit.
7
```

7)Binary Search Tree

CODE:-

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   struct node *lchild;
   int info;
   struct node *rchild;
};

struct node *insert(struct node *ptr, int ikey)
{
   if (ptr == NULL)
   {
      ptr = (struct node *)malloc(sizeof(struct node));
      ptr->info = ikey;
      ptr->lchild = NULL;
      ptr->rchild = NULL;
   }
   else if (ikey < ptr->info)
      ptr->lchild = insert(ptr->lchild, ikey);
   else if (ikey > ptr->info)
      ptr->rchild = insert(ptr->rchild, ikey);
   else
      printf("Duplicate key.\n");
   return ptr;
}

struct node *search(struct node *ptr, int skey)
{
   if (ptr == NULL)
   {
      printf("Key not found in tree.\n");
      return NULL;
   }
   else if (skey < ptr->info)
      return search(ptr->lchild, skey);
   else if (skey > ptr->info)
      return search(ptr->rchild, skey);
   else
      return ptr;
}

struct node *del(struct node *ptr, int dkey)
{
   struct node *tmp, *succ;
   if (ptr == NULL)
   {
      printf("Element %d not present in the tree.\n", dkey);
      return ptr;
```

```
    }
    if (dkey < ptr->info)
        ptr->lchild = del(ptr->lchild, dkey);
    else if (dkey > ptr->info)
        ptr->rchild = del(ptr->rchild, dkey);
    else
    {
        if (ptr->lchild != NULL && ptr->rchild != NULL)
        {
            succ = ptr->rchild;
            while (succ->lchild != NULL)
                succ = succ->lchild;
            ptr->info = succ->info;
            ptr->rchild = del(ptr->rchild, succ->info);
        }
        else
        {
            tmp = ptr;
            if (ptr->lchild != NULL)
                ptr = ptr->lchild;
            else if (ptr->rchild != NULL)
                ptr = ptr->rchild;
            else
                ptr = NULL;
            free(tmp);
        }
    }
    return ptr;
}

int height(struct node *ptr)
{
    int h_left, h_right;
    if (ptr == NULL)
        return 0;
    h_left = height(ptr->lchild);
    h_right = height(ptr->rchild);
    if (h_left > h_right)
        return 1 + h_left;
    else
        return 1 + h_right;
}

void displaygivenlevel(struct node *ptr, int level)
{
    if (ptr == NULL)
        return;
    if (level == 1)
        printf("%d ", ptr->info);
    else if (level > 1)
    {
        displaygivenlevel(ptr->lchild, level - 1);
        displaygivenlevel(ptr->rchild, level - 1);
    }
}

void levelorder(struct node *ptr)
{
    int h = height(ptr);
    int i;
    for (i = 1; i <= h; i++)
        displaygivenlevel(ptr, i);
}

void inorder(struct node *ptr)
{
    if (ptr == NULL)
        return;
    inorder(ptr->lchild);
    printf("%d ", ptr->info);
    inorder(ptr->rchild);
}

void preorder(struct node *ptr)
{
    if (ptr == NULL)
        return;
    printf("%d ", ptr->info);
    preorder(ptr->lchild);
    preorder(ptr->rchild);
}

void postorder(struct node *ptr)
{
    if (ptr == NULL)
        return;
    postorder(ptr->lchild);
    postorder(ptr->rchild);
    printf("%d ", ptr->info);
}

int main()
{
    int ch, elem;
    struct node *root = NULL, *tmp = NULL;
    while (1)
    {
        printf("\n1.Insertion.\n2.Deletion.\n3.Searching.\n4.Levelorder.\n5.Preorder.\n6.Postorder.\
n7.Inorder.\n8.Exit.\n");
        printf("Enter your choice.\n");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Enter element to be inserted.\n");
            scanf("%d", &elem);
            root = insert(root, elem);
            break;
        case 2:
            printf("Enter element to be deleted.\n");
            scanf("%d", &elem);
            root = del(root, elem);
            break;
        case 3:
            printf("Enter element to be searched.\n");
            scanf("%d", &elem);
            tmp = search(root, elem);
            if (tmp != NULL)
                printf("Key found in tree.\n");
            break;
        case 4:
            printf("Levelorder traversal is:\n");
            levelorder(root);
            break;
        case 5:
            printf("Preorder traversal is:\n");
            preorder(root);
            break;
        case 6:
            printf("Postorder traversal is:\n");
            postorder(root);
            break;
        case 7:
            printf("Inorder traversal is:\n");
            inorder(root);
            break;
        case 8:
            exit(1);
        default:
            printf("Erroneous input.\n");
        }
    }
    return 0;
}
```

OUTPUT:-

```
[sorciermahep@fedora DS_Labs] $ cd "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/" && gcc --std=c17 20.c -o 20 && "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/"20

1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
1
Enter element to be inserted.
6

1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
1
Enter element to be inserted.
3

1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
1
Enter element to be inserted.
1

1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
```

```
8.Exit.
Enter your choice.
1
Enter element to be inserted.
8

1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
1
Enter element to be inserted.
7

1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
1
Enter element to be inserted.
5

1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
4
Levelorder traversal is:
6 3 8 1 5 7
1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
```

```
Enter your choice.
5
Preorder traversal is:
6 3 1 5 8 7
1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
6
Postorder traversal is:
1 5 3 7 8 6
1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
7
Inorder traversal is:
1 3 5 6 7 8
1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
3
Enter element to be searched.
3
Key found in tree.

1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
3
```

```
Enter element to be searched.
2
Key not found in tree.

1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
2
Enter element to be deleted.
3

1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
4
Levelorder traversal is:
6 5 8 1 7
1.Insertion.
2.Deletion.
3.Searching.
4.Levelorder.
5.Preorder.
6.Postorder.
7.Inorder.
8.Exit.
Enter your choice.
8
```

8.1)Undirected Graph Adjacency Matrix

CODE:-

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int adj[MAX][MAX];
int n;
void create_graph();
void display();
void insert_edge(int origin, int destin);
void del_edge(int origin, int destin);
int main()
{
    int choice, origin, destin;
    create_graph();
    while (1)
    {
        printf("1.Insert an edge.\n");
        printf("2.Delete an edge.\n");
        printf("3.Display.\n");
        printf("4.Exit.\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter an edge to be inserted : ");
            scanf("%d %d", &origin, &destin);
            insert_edge(origin, destin);
            break;
        case 2:
            printf("Enter an edge to be deleted : ");
            scanf("%d %d", &origin, &destin);
            del_edge(origin, destin);
            break;
        case 3:
            display();
            break;
        case 4:
            exit(1);
        default:
            printf("Erroneous input.\n");
            break;
        }
    }
}
void create_graph()
{
    int max_edges, i, origin, destin;
    printf("Enter number of vertices : ");
    scanf("%d", &n);
    max_edges = n * (n - 1) / 2;
    for (i = 1; i <= max_edges; i++)
    {

        printf("Enter edge %d( Enter -1 -1 to quit ) : ", i);
        scanf("%d %d", &origin, &destin);
        if ((origin == -1) && (destin == -1))
            break;
        if (origin >= n || destin >= n || origin < 0 || destin < 0)
        {
            printf("Invalid vertex entered.\n");
            i--;
        }
        else
        {
            adj[origin][destin] = 1;
            adj[destin][origin] = 1;
        }
    }
}
void del_edge(int origin, int destin)
{
    if (origin < 0 || origin >= n | destin < 0 || destin >= n || adj[origin][destin] == 0)
    {
        printf("This edge does not exist.\n");
        return;
    }
    adj[origin][destin] = 0;
    adj[destin][origin] = 0;
}
void insert_edge(int origin, int destin)
{
    if (origin < 0 || origin >= n)
    {
        printf("Origin vertex does not exist.\n");
        return;
    }
    if (destin < 0 || destin >= n)
    {
        printf("Destination vertex does not exist.\n");
        return;
    }
    adj[origin][destin] = 1;
    adj[destin][origin] = 1;
}

void display()
{
    int i, j;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
```

```
            printf("%d", adj[i][j]);
        printf("\n");
    }
}
```

OUTPUT:-

```
[sorciermahep@fedora DS_Labs] $ cd "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/" && gcc --std=c17 16a.c -o 16a && "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/"16a
Enter number of vertices : 4
Enter edge 1( Enter -1 -1 to quit ) : 1
2
Enter edge 2( Enter -1 -1 to quit ) : 0
3
Enter edge 3( Enter -1 -1 to quit ) : 1
3
Enter edge 4( Enter -1 -1 to quit ) : 2
1
Enter edge 5( Enter -1 -1 to quit ) : 1
0
Enter edge 6( Enter -1 -1 to quit ) : -1
-1
1.Insert an edge.
2.Delete an edge.
3.Display.
4.Exit.
Enter your choice : 3
0101
1011
0100
1100
1.Insert an edge.
2.Delete an edge.
3.Display.
4.Exit.
Enter your choice : 2
Enter an edge to be deleted : 1
0
1.Insert an edge.
2.Delete an edge.
3.Display.
4.Exit.
Enter your choice : 3
0001
0011
0100
1100
1.Insert an edge.
2.Delete an edge.
3.Display.
4.Exit.
Enter your choice : 4
```

8.2)Undirected Graph Adjacency List

CODE:-

```
#include <stdio.h>
#include <stdlib.h>
struct Edge;
struct Vertex
{
    int info;
    struct Vertex *nextVertex;
    struct Edge *firstEdge;
} *start = NULL;
struct Edge
{
    struct Vertex *destVertex;
    struct Edge *nextEdge;
};
void insertVertex(int u);
void insertEdge(int u, int v);
struct Vertex *findVertex(int u);
void deleteIncomingEdges(int u);
void deleteVertex(int u);
void deleteEdge(int u, int v);
void display();
int main()
{
    int ch, u, origin, destin;
    struct Vertex *tmp = NULL;
    while (1)
    {
        printf("1.Insert a vertex.\n");
        printf("2.Insert an edge.\n");
        printf("3.Delete a vertex.\n");
        printf("4.Delete an edge.\n");
        printf("5.Search vertex.\n");
        printf("6.Display.\n");
        printf("7.Exit.\n");
        printf("Enter the choice: \n");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Enter a vertex to be inserted : ");
            scanf("%d", &u);
            insertVertex(u);
            break;
        case 2:
            printf("Enter an Edge to be inserted :\n");
            printf("origin:");
            scanf("%d", &origin);
            printf("destination: ");
            scanf("%d", &destin);
            insertEdge(origin, destin);
            insertEdge(destin, origin);
            break;
        case 3:
            printf("Enter a vertex to be deleted : ");
            scanf("%d", &u);
            deleteIncomingEdges(u);
            deleteVertex(u);
            break;
        case 4:
            printf("Enter an edge to be deleted : ");
            printf("origin:");
            scanf("%d", &origin);
            printf("destination: ");
            scanf("%d", &destin);
            deleteEdge(origin, destin);
            deleteEdge(destin, origin);
            break;
        case 5:
            printf("Enter the element to be searched.\n");
            scanf("%d", &u);
            tmp = findVertex(u);
            if (tmp == NULL)
                printf("Vertex not found.\n");
            else
                printf("Vertex found.\n");
            break;
        case 6:
            display();
            break;
        case 7:
            exit(1);
        default:
            printf("Erroneous input.\n");
            break;
        }
    }
}
void insertVertex(int u)
{
    struct Vertex *tmp, *ptr;
    tmp = malloc(sizeof(struct Vertex));
    tmp->info = u;
    tmp->nextVertex = NULL;
    tmp->firstEdge = NULL;
    if (start == NULL)
    {
        start = tmp;
        return;
    }
    ptr = start;
    while (ptr->nextVertex != NULL)
```

```
        ptr = ptr->nextVertex;

    ptr->nextVertex = tmp;
}
struct Vertex *findVertex(int u)
{
    struct Vertex *ptr, *loc;
    ptr = start;
    while (ptr != NULL)
    {

        if (ptr->info == u)
        {
            loc = ptr;
            return loc;
        }
        else

            ptr = ptr->nextVertex;
    }
    loc = NULL;
    return loc;
}
void insertEdge(int u, int v)
{

    struct Vertex *locu, *locv;
    struct Edge *ptr, *tmp;
    locu = findVertex(u);
    locv = findVertex(v);
    if (locu == NULL)
    {

        printf("Start vertex not present, first insert vertex %d.\n", u);
        return;
    }
    if (locv == NULL)
    {

        printf("End vertex not present, first insert vertex %d.\n", v);
        return;
    }
    tmp = malloc(sizeof(struct Edge));
    tmp->destVertex = locv;
    tmp->nextEdge = NULL;
    if (locu->firstEdge == NULL)
    {
        locu->firstEdge = tmp;
        return;
    }
    ptr = locu->firstEdge;
    while (ptr->nextEdge != NULL)
```

```
        ptr = ptr->nextEdge;

    ptr->nextEdge = tmp;
}
void deleteIncomingEdges(int u)
{
    struct Vertex *ptr;
    struct Edge *q, *tmp;
    ptr = start;
    while (ptr != NULL)
    {

        if (ptr->firstEdge == NULL)
        {
            ptr = ptr->nextVertex;
            continue;
        }
        if (ptr->firstEdge->destVertex->info == u)
        {
            tmp = ptr->firstEdge;
            ptr->firstEdge = ptr->firstEdge->nextEdge;
            free(tmp);
            continue;
        }
        q = ptr->firstEdge;
        while (q->nextEdge != NULL)
        {

            if (q->nextEdge->destVertex->info == u)
            {
                tmp = q->nextEdge;
                q->nextEdge = tmp->nextEdge;
                free(tmp);
                continue;
            }
            q = q->nextEdge;
        }
        ptr = ptr->nextVertex;
    }
}
void deleteVertex(int u)
{

    struct Vertex *tmp, *q;
    struct Edge *p, *temporary;
    if (start == NULL)
    {

        printf("No vertices present.\n");
        return;
    }
    if (start->info == u)
    {
        tmp = start;
        start = start->nextVertex;
    }
    else
    {

        q = start;
        while (q->nextVertex != NULL)
        {

            if (q->nextVertex->info == u)
                break;
            q = q->nextVertex;
        }
        if (q->nextVertex == NULL)
        {

            printf("Vertex not found.\n");
            return;
        }
        else
        {

            tmp = q->nextVertex;
            q->nextVertex = tmp->nextVertex;
        }
    }
    p = tmp->firstEdge;
    while (p != NULL)
    {
        temporary = p;
        p = p->nextEdge;
        free(temporary);
    }
    free(tmp);
}
void deleteEdge(int u, int v)
{
    struct Vertex *locu;
    struct Edge *tmp, *q;
    locu = findVertex(u);
    if (locu == NULL)
    {

        printf("Start vertex not present.\n");
        return;
    }
    if (locu->firstEdge == NULL)
    {

        printf("Edge not present.\n");
        return;
    }
    if (locu->firstEdge->destVertex->info == v)
    {
        tmp = locu->firstEdge;
        locu->firstEdge = locu->firstEdge->nextEdge;
        free(tmp);
        return;
    }
    q = locu->firstEdge;
    while (q->nextEdge != NULL)
    {

        if (q->nextEdge->destVertex->info == v)
        {
            tmp = q->nextEdge;
            q->nextEdge = tmp->nextEdge;
            free(tmp);
            return;
        }
        q = q->nextEdge;
    }
    printf("This Edge not present in the graph.\n");
}
void display()
{

    struct Vertex *ptr;
    struct Edge *q;
    ptr = start;
    while (ptr != NULL)
    {

        printf("%d ->", ptr->info);
        q = ptr->firstEdge;
        while (q != NULL)
        {

            printf(" %d", q->destVertex->info);
            q = q->nextEdge;
        }
        printf("\n");
        ptr = ptr->nextVertex;
    }
}
```

OUTPUT:-

```
[sorciermahep@fedora DS_Labs] $ cd "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/" && gcc --std=c17 16b.c -o 16b &&
"/home/sorciermahep/Desktop/Mahendra Priolkar/C/DS_Labs/"16b
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
1
Enter a vertex to be inserted : 3
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
1
Enter a vertex to be inserted : 4
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
1
Enter a vertex to be inserted : 5
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
5
Enter the element to be searched.
5
Vertex found.
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
```

```
6.Display.
7.Exit.
Enter the choice:
2
Enter an Edge to be inserted :
origin:3
destination: 4
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
6
3 -> 4
4 -> 3
5 ->
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
1
Enter a vertex to be inserted : 6
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
2
Enter an Edge to be inserted :
origin:5
destination: 6
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
3
Enter a vertex to be deleted : 6
1.Insert a vertex.
```

```
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
6
3 -> 4
4 -> 3
5 ->
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
4
Enter an edge to be deleted : origin:4
destination: 3
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
6
3 ->
4 ->
5 ->
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
7
```

8.3)Directed Graph Adjacency Matrix

CODE:-

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int adj[MAX][MAX];
int n;
void create_graph();
void display();
void insert_edge(int origin, int destin);
void del_edge(int origin, int destin);
int main()
{
    int choice, origin, destin;
    create_graph();
    while (1)
    {
        printf("1.Insert an edge.\n");
        printf("2.Delete an edge.\n");
        printf("3.Display.\n");
        printf("4.Exit.\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter an edge to be inserted : ");
            scanf("%d %d", &origin, &destin);
            insert_edge(origin, destin);
            break;
        case 2:
            printf("Enter an edge to be deleted : ");
            scanf("%d %d", &origin, &destin);
            del_edge(origin, destin);
            break;
        case 3:
            display();
            break;
        case 4:
            exit(1);
        default:
            printf("Erroneous input.\n");
            break;
        }
    }
}
void create_graph()
{
    int max_edges, i, origin, destin;
    printf("Enter number of vertices : ");
    scanf("%d", &n);
    max_edges = n * (n - 1);
    for (i = 1; i <= max_edges; i++)
    {

        printf("Enter edge %d( Enter -1 -1 to quit ) : ", i);
        scanf("%d %d", &origin, &destin);
        if ((origin == -1) && (destin == -1))
            break;
        if (origin >= n || destin >= n || origin < 0 || destin < 0)
        {
            printf("Invalid vertex entered.\n");
            i--;
        }
        else
        {
            adj[origin][destin] = 1;
        }
    }
}
void del_edge(int origin, int destin)
{
    if (origin < 0 || origin >= n | destin < 0 || destin >= n || adj[origin][destin] == 0)
    {
        printf("This edge does not exist.\n");
        return;
    }
    adj[origin][destin] = 0;
}
void insert_edge(int origin, int destin)
{
    if (origin < 0 || origin >= n)
    {
        printf("Origin vertex does not exist.\n");
        return;
    }
    if (destin < 0 || destin >= n)
    {
        printf("Destination vertex does not exist.\n");
        return;
    }
    adj[origin][destin] = 1;
}

void display()
{
    int i, j;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
            printf("%d", adj[i][j]);
        printf("\n");
    }
```

```
}
```

OUTPUT:-

```
[sorciermahep@fedora DS_Labs] $ cd "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/" && gcc --std=c17 16c.c -o 16c && "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/"16c
Enter number of vertices : 4
Enter edge 1( Enter -1 -1 to quit ) : 1
0
Enter edge 2( Enter -1 -1 to quit ) : 2
3
Enter edge 3( Enter -1 -1 to quit ) : 3
1
Enter edge 4( Enter -1 -1 to quit ) : 1
2
Enter edge 5( Enter -1 -1 to quit ) : 2
1
Enter edge 6( Enter -1 -1 to quit ) : -1
-1
1.Insert an edge.
2.Delete an edge.
3.Display.
4.Exit.
Enter your choice : 3
0000
1010
0101
0100
1.Insert an edge.
2.Delete an edge.
3.Display.
4.Exit.
Enter your choice : 2
Enter an edge to be deleted : 1
0
1.Insert an edge.
2.Delete an edge.
3.Display.
4.Exit.
Enter your choice : 3
0000
0010
0101
0100
1.Insert an edge.
2.Delete an edge.
3.Display.
4.Exit.
Enter your choice : 4
```

8.4)Directed Graph Adjacency List

CODE:-

```
#include <stdio.h>
#include <stdlib.h>
struct Edge;
struct Vertex
{
    int info;
    struct Vertex *nextVertex;
    struct Edge *firstEdge;
} *start = NULL;
struct Edge
{
    struct Vertex *destVertex;
    struct Edge *nextEdge;
};
void insertVertex(int u);
void insertEdge(int u, int v);
struct Vertex *findVertex(int u);
void deleteIncomingEdges(int u);
void deleteVertex(int u);
void deleteEdge(int u, int v);
void display();
int main()
{
    int ch, u, origin, destin;
    struct Vertex *tmp = NULL;
    while (1)
    {
        printf("1.Insert a vertex.\n");
        printf("2.Insert an edge.\n");
        printf("3.Delete a vertex.\n");
        printf("4.Delete an edge.\n");
        printf("5.Search vertex.\n");
        printf("6.Display.\n");
        printf("7.Exit.\n");
        printf("Enter the choice: \n");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Enter a vertex to be inserted : ");
            scanf("%d", &u);
            insertVertex(u);
            break;
        case 2:
            printf("Enter an Edge to be inserted :\n");
            printf("origin:");
            scanf("%d", &origin);
            printf("destination: ");
            scanf("%d", &destin);
```

```
                        insertEdge(origin, destin);
                        break;
                case 3:
                        printf("Enter a vertex to be deleted : ");
                        scanf("%d", &u);
                        deleteIncomingEdges(u);
                        deleteVertex(u);
                        break;
                case 4:
                        printf("Enter an edge to be deleted : ");
                        printf("origin:");
                        scanf("%d", &origin);
                        printf("destination: ");
                        scanf("%d", &destin);
                        deleteEdge(origin, destin);
                        break;
                case 5:
                        printf("Enter the element to be searched.\n");
                        scanf("%d", &u);
                        tmp = findVertex(u);
                        if (tmp == NULL)
                                printf("Vertex not found.\n");
                        else
                                printf("Vertex found.\n");
                        break;
                case 6:
                        display();
                        break;
                case 7:
                        exit(1);
                default:
                        printf("Erroneous input.\n");
                        break;
                }
        }
}
void insertVertex(int u)
{
        struct Vertex *tmp, *ptr;
        tmp = malloc(sizeof(struct Vertex));
        tmp->info = u;
        tmp->nextVertex = NULL;
        tmp->firstEdge = NULL;
        if (start == NULL)
        {
                start = tmp;
                return;
        }
        ptr = start;
        while (ptr->nextVertex != NULL)
                ptr = ptr->nextVertex;
```

```
    ptr->nextVertex = tmp;
}
struct Vertex *findVertex(int u)
{
    struct Vertex *ptr, *loc;
    ptr = start;
    while (ptr != NULL)
    {

        if (ptr->info == u)
        {
            loc = ptr;
            return loc;
        }
        else

            ptr = ptr->nextVertex;
    }
    loc = NULL;
    return loc;
}
void insertEdge(int u, int v)
{

    struct Vertex *locu, *locv;
    struct Edge *ptr, *tmp;
    locu = findVertex(u);
    locv = findVertex(v);
    if (locu == NULL)
    {

        printf("Start vertex not present, first insert vertex %d.\n", u);
        return;
    }
    if (locv == NULL)
    {

        printf("End vertex not present, first insert vertex %d.\n", v);
        return;
    }
    tmp = malloc(sizeof(struct Edge));
    tmp->destVertex = locv;
    tmp->nextEdge = NULL;
    if (locu->firstEdge == NULL)
    {
        locu->firstEdge = tmp;
        return;
    }
    ptr = locu->firstEdge;
    while (ptr->nextEdge != NULL)
        ptr = ptr->nextEdge;
    ptr->nextEdge = tmp;
}
void deleteIncomingEdges(int u)
{
    struct Vertex *ptr;
    struct Edge *q, *tmp;
    ptr = start;
    while (ptr != NULL)
    {

        if (ptr->firstEdge == NULL)
        {
            ptr = ptr->nextVertex;
            continue;
        }
        if (ptr->firstEdge->destVertex->info == u)
        {
            tmp = ptr->firstEdge;
            ptr->firstEdge = ptr->firstEdge->nextEdge;
            free(tmp);
            continue;
        }
        q = ptr->firstEdge;
        while (q->nextEdge != NULL)
        {

            if (q->nextEdge->destVertex->info == u)
            {
                tmp = q->nextEdge;
                q->nextEdge = tmp->nextEdge;
                free(tmp);
                continue;
            }
            q = q->nextEdge;
        }
        ptr = ptr->nextVertex;
    }
}
void deleteVertex(int u)
{

    struct Vertex *tmp, *q;
    struct Edge *p, *temporary;
    if (start == NULL)
    {

        printf("No vertices present.\n");
        return;
    }
    if (start->info == u)
    {
        tmp = start;
        start = start->nextVertex;
    }
    else
    {

        q = start;
        while (q->nextVertex != NULL)
        {

            if (q->nextVertex->info == u)
                break;
            q = q->nextVertex;
        }
        if (q->nextVertex == NULL)
        {

            printf("Vertex not found.\n");
            return;
        }
        else
        {

            tmp = q->nextVertex;
            q->nextVertex = tmp->nextVertex;
        }
    }
    p = tmp->firstEdge;
    while (p != NULL)
    {
        temporary = p;
        p = p->nextEdge;
        free(temporary);
    }
    free(tmp);
}
void deleteEdge(int u, int v)
{
    struct Vertex *locu;
    struct Edge *tmp, *q;
    locu = findVertex(u);
    if (locu == NULL)
    {

        printf("Start vertex not present.\n");
        return;
    }
    if (locu->firstEdge == NULL)
    {

        printf("Edge not present.\n");
        return;
    }
```

```
        if (locu->firstEdge->destVertex->info == v)
        {
                tmp = locu->firstEdge;
                locu->firstEdge = locu->firstEdge->nextEdge;
                free(tmp);
                return;
        }
        q = locu->firstEdge;
        while (q->nextEdge != NULL)
        {

                if (q->nextEdge->destVertex->info == v)
                {
                        tmp = q->nextEdge;
                        q->nextEdge = tmp->nextEdge;
                        free(tmp);
                        return;
                }
                q = q->nextEdge;
        }
        printf("This Edge not present in the graph.\n");
}
void display()
{

        struct Vertex *ptr;
        struct Edge *q;
        ptr = start;
        while (ptr != NULL)
        {

                printf("%d ->", ptr->info);
                q = ptr->firstEdge;
                while (q != NULL)
                {

                        printf(" %d", q->destVertex->info);
                        q = q->nextEdge;
                }
                printf("\n");
                ptr = ptr->nextVertex;
        }
}
```

OUTPUT:-

```
[sorciermahep@fedora DS_Labs] $ cd "/home/sorciermahep/Desktop/Mahendra
Priolkar/C/DS_Labs/" && gcc --std=c17 16d.c -o 16d &&
"/home/sorciermahep/Desktop/Mahendra Priolkar/C/DS_Labs/"16d
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
1
Enter a vertex to be inserted : 3
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
1
Enter a vertex to be inserted : 4
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
1
Enter a vertex to be inserted : 5
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
2
Enter an Edge to be inserted :
origin:3
destination: 4
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
```

```
6.Display.
7.Exit.
Enter the choice:
2
Enter an Edge to be inserted :
origin:4
destination: 5
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
6
3 -> 4
4 -> 5
5 ->
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
5
Enter the element to be searched.
5
Vertex found.
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
3
Enter a vertex to be deleted : 4
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
6
3 ->
5 ->
```

```
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
2
Enter an Edge to be inserted :
origin:3
destination: 5
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
1
Enter a vertex to be inserted : 7
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
4
Enter an edge to be deleted : origin:3
destination: 5
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
Enter the choice:
6
3 ->
5 ->
7 ->
1.Insert a vertex.
2.Insert an edge.
3.Delete a vertex.
4.Delete an edge.
5.Search vertex.
6.Display.
7.Exit.
```

```
Enter the choice:
7
```