

Kathmandu Bernhardt College



Laboratory Manual

Data Structure and Algorithms
(CSC-211)

By: Abhimanu Yadav

Department of Computer Science and Information Technology

Kathmandu Bernhardt College

Tribhuvan University

Bafal, Kathmandu, Nepal

Tel: 5237330, 5237361

Table of Contents

S. No.	Name of Experiments	Page No.
1.	Write a program to read n numbers and display it.	
2.	Write a program to demonstrate the concept of one dimensional array finding the sum of array elements.	
3.	Write a program to display Fibonacci series up to a range.	
4.	Write a program to insert an element in an array.	
5.	Write a program to delete an element from an array.	
6.	Write a program to add two matrix A and B.	
7.	Write a program to multiply two matrix A and B.	
8.	Write a menu driven program to implement the stack operations using array.	
9.	Write a program to convert the given infix expression to postfix expression.	
10.	Write a program to evaluate the postfix expression.	
11.	Write a menu driven program to implement the linear queue (its operations) using array.	
12.	Write a program to implement the circular queue.	
13.	Write a program to implement the priority queue.	
14.	Write a program to find the factorial using recursion.	
15.	Write a program to find the Fibonacci series using recursion.	
16.	Write a program to find the GCD using recursion.	
17.	Write a program to find the LCM using recursion.	
18.	Write a program to simulate the TOH.	
19.	Write a menu driven program to implement the singly linked list (its operations).	
20.	Write a program to implement the doubly linked list.	
21.	Write a program to implement the circular linked list.	
22.	Write a program to implement the following sorting: i. Bubble Sort ii. Selection Sort iii. Insertion Sort iv. Shell Sort	
23.	Write a program to implement divide and conquer algorithms in following sorting: i. Merge Sort ii. Quick Sort iii. Heap Sort	
24.	Write a program to implement the sequential and binary search.	
25.	Write a program to implement the tree traversal method.	

26.	Write a program to perform the following operations: i. Insert an element into a BST. ii. Delete an element from a BST. iii. Search for a key element in a BST.	
27.	Write a program to perform the following operations: i. Insert an element into an AVL tree. ii. Delete an element from an AVL tree. iii. Search for a key element in an AVL tree.	
28.	Write a program to implement the BFS and DFS traversal.	
29.	Write a program to find MST using prim's and kruskal's algorithm.	
30.	Write a program to find shortest path using Dijkstra's algorithm.	
31.	Write a program to implement the Stack and Queue using linked list.	
32.	Write a program to Subtract two matrix A and B.	

Experiment No. : 1

Write a program to read n numbers and display it.

Source Code:

```
#include<stdio.h>

int main()
{
    int i,n, a[10];
    printf("\nEnter the number of element : \n");
    scanf("%d",&n);
    printf("Enter element: \n");
        for(i=0;i<n;i++)
        {
            printf("a[%d]=",i);
            scanf("%d",&a[i]);
        }
    printf("\n Display array element: \n");
    for(i=0;i<n;i++)
    {
        printf("a[%d]=%d\n",i,a[i]);
    }
    return 0;
}
```

Output:

```
Enter the number of element :
5
Enter element:
a[0]=5
a[1]=4
a[2]=8
a[3]=7
a[4]=9

Display array element:
a[0]=5
a[1]=4
a[2]=8
a[3]=7
a[4]=9

-----
Process exited after 20.99 seconds with return value 0
Press any key to continue . . .
```

Experiment No. : 2

Write a program to demonstrate the concept of one dimensional array finding the sum of array elements.

Source Code:

```
#include<stdio.h>

int main()
{
    int i,n, a[10],s;
    printf("enter the number of elements:\t");
    scanf("%d",&n);
    s=0;
    printf("Enter element:\n");
    for(i=0;i<n;i++)
    {
        printf("a[%d]=",i);
        scanf("%d",&a[i]);
```

```

        s=s+a[i];

    }

    printf("Sum of array element:%d",s);

    return 0;

}

```

Output:

```

enter the number of elements: 5
Enter element:
a[0]=4
a[1]=2
a[2]=5
a[3]=6
a[4]=2
Sum of array element:19
-----
Process exited after 11.3 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 3

Write a program to display Fibonacci series up to a range.

Source Code:

```

#include<stdio.h>

int main()

{

    int a,b,c,n;

    printf("\nEnter range:");

    scanf("%d",&n);

    a=0,b=1,c=0;

    printf("%d \t %d",a,b);

```

```

c=a+b;
while(c<=n)
{
printf("\t%d",c);
a=b;
b=c;
c=a+b;
}
return 0;
}

```

Output:

```

Enter range:7
0      1      1      2      3      5
-----
Process exited after 26.39 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 4

Write a program to insert an element in an array.

Source Code:

```

#include<stdio.h>

int main()
{
int i,n,pos,num, a[10];
printf("Enter the number of element :\n");
scanf("%d",&n);
printf("Enter element:\n");

```

```

for(i=0;i<n;i++)
{
    printf("a[%d]=",i);
    scanf("%d",&a[i]);
}
printf("\nEnter the pos where the no. is to be inserted :");
scanf("%d",&pos);
printf("\nEnter the the no. is to be inserted :");
scanf("%d",&num);
for(i=n-1;i>=pos;i--)
    a[i+1]=a[i];
    n=n+1;
    a[pos]=num;
printf("\n Insert an element into an array:");
for(i=0;i<n;i++)
{
    printf("a[%d]=%d\n",i,a[i]);
}
return 0;
}

```

Output:


```

Enter the number of element :
4
Enter element:
a[0]=5
a[1]=2
a[2]=6
a[3]=9

Enter the pos where the no. is to be inserted :2

Enter the the no. is to be inserted :1

Insert an element into an array:a[0]=5
a[1]=2
a[2]=1
a[3]=6
a[4]=9

-----
Process exited after 26.68 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 5

Write a program to delete an element from an array.

Source Code:

```

#include<stdio.h>

int main()
{
    int i,n,pos, a[10];
    printf("Enter the number of elements :\n");
    scanf("%d",&n);
    printf("Enter element: \n ");
    for(i=0;i<n;i++)
    {
        printf("a[%d]=",i);
        scanf("%d",&a[i]);
    }
}

```

```

printf("\nEnter the pos from which the no. has to be deleted :");
scanf("%d",&pos);
for(i=pos;i<n;i++)
a[i]=a[i+1];
n=n-1;
printf("\n Displar array after deletion: \n ");
for(i=0;i<n;i++)
{
    printf("\n a[%d]=%d",i,a[i]);
}
return 0;
}

```

Output:

```

Select C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\Delete_From_Array.exe
Enter the number of elements :
5
Enter element:
a[0]=2
a[1]=4
a[2]=8
a[3]=32
a[4]=6

Enter the pos from which the no. has to be deleted :0

Displar array after deletion:

a[0]=4
a[1]=8
a[2]=32
a[3]=6
-----
Process exited after 20.99 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 6

Write a program to add two matrix A and B.

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,j,m,n,p,q;
    int a[10][10], b[10][10], c[10][10];
    printf("\nEnter no of rows and column of matrix A:");
    scanf("%d\t%d",&m,&n);
    printf("\nEnter no of rows and column of matrix B:");
    scanf("%d\t%d",&p,&q);
    if(m!=p && n!=q)
    {
        printf("\n Matrix cannot be added.");
        exit(0);
    }
    printf("\n Matrix can be added");
    printf("\n Enter elements of matrix A:");
    for(i=0;i<m;i++)
    for(j=0;j<n;j++)
    scanf("%d",&a[i][j]);
    printf("\n Enter elements of matrix B:");
    for(i=0;i<p;i++)
```

```

for(j=0;j<q;j++)
scanf("%d",&b[i][j]);
for(i=0;i<m;i++)
for(j=0;j<n;j++)
c[i][j]=a[i][j]+b[i][j];
printf("\n Display matrix A:\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
        printf("%d\t",a[i][j]);
    printf("\n");
}
printf("\n Display matrix B:\n");
for(i=0;i<p;i++)
{
    for(j=0;j<q;j++)
        printf("%d\t",b[i][j]);
    printf("\n");
}
printf("\n Display matrix C:\n");
for(i=0;i<p;i++)
{
    for(j=0;j<q;j++)
        printf("%d\t",c[i][j]);
    printf("\n");
}

```

```

}
return 0;
}

```

Output:

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\Add_two_matrix.exe

```

Enter no of rows and column of matrix B:3 3

Matrix can be added
Enter elements of matrix A:2 3 4
2 4 3
1 7 6

Enter elements of matrix B:1 3 4
2 5 6
7 8 4

Display matrix A:
2      3      4
2      4      3
1      7      6

Display matrix B:
1      3      4
2      5      6
7      8      4

Display matrix C:
3      6      8
4      9      9
8      15     10

-----
Process exited after 148 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 7

Write a program to multiply two matrix A and B.

Source Code:

```

#include<stdio.h>

#include<stdlib.h>

int main()

{

int i,j,m,n,p,q,k;

```

```

int a[10][10], b[10][10], c[10][10];
printf("\nEnter no of rows and column of matrix A:");
scanf("%d%d",&m,&n);
printf("\nEnter no of rows and column of matrix B:");
scanf("%d%d",&p,&q);
printf("\n Enter elements of matrix A:\n");
for(i=0;i<m;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
printf("\n Enter elements of matrix B:\n");
for(i=0;i<p;i++)
for(j=0;j<q;j++)
scanf("%d",&b[i][j]);
if(n==p)
{
for(i=0;i<m;i++)
for(j=0;j<q;j++)
{
c[i][j]=0;
for(k=0;k<n;k++)
c[i][j]=c[i][j]+(a[i][k]*b[k][j]);
}
}
else
{

```

```
printf("\n Matrix cannot be multiplied");
exit(1);
}
printf("\n Display matrix A:\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
printf("%d\t",a[i][j]);
printf("\n");
}
printf("\n Display matrix B:\n");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
printf("%d\t",b[i][j]);
printf("\n");
}
printf("\n Display Product:\n");
for(i=0;i<m;i++)
{
for(j=0;j<q;j++)
printf("%d\t",c[i][j]);
printf("\n");
}
return 0;;
```

}

Output:

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\Matrix_multiplication.exe

```
Enter no of rows and column of matrix A:3 3
Enter no of rows and column of matrix B:3 3

Enter elements of matrix A:
3 4 5
2 5 3
4 6 7

Enter elements of matrix B:
2 4 7
5 3 6
4 8 9

Display matrix A:
3      4      5
2      5      3
4      6      7

Display matrix B:
2      4      7
5      3      6
4      8      9

Display Product:
46      64      90
41      47      71
66      90      127

-----
Process exited after 44.08 seconds with return value 0
Press any key to continue . . .
```

Experiment No. : 8

Write a menu driven program to implement the stack operations using array.

Source Code:

```
#include<stdio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()
```



```

{
    top=-1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d",&n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t-----");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {

```

```

        display();
        break;
    }
    case 4:
    {
        printf("\n\t EXIT POINT ");
        break;
    }
    default:
    {
        printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
    }

}

}

while(choice!=4);
return 0;
}

void push()
{
    if(top>=n-1)
    {
        printf("\n\tSTACK is over flow");

    }
}

```

```

else
{
    printf(" Enter a value to be pushed:");
    scanf("%d",&x);
    top++;
    stack[top]=x;
}
}

void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}

void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK \n");


```

```

        for(i=top; i>=0; i--)
            printf("\n%d",stack[i]);
        printf("\n Press Next Choice");
    }
else
{
    printf("\n The STACK is empty");
}
}

```

Output:

 C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\stack.exe

```

Enter the size of STACK[MAX=100]:5

    STACK OPERATIONS USING ARRAY
    -----
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT
Enter the Choice:1
Enter a value to be pushed:5

Enter the Choice:1
Enter a value to be pushed:5

Enter the Choice:3

The elements in STACK
5
5
Press Next Choice
Enter the Choice:

```

Experiment No. : 9

Write a program to convert the given infix expression to postfix expression.

Source Code:

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

char stack[100];
int top = -1;

void push(char x)
{
    stack[++top] = x;
}

char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}

int priority(char x)
{
    if(x == '(')
```

```

        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/')
        return 2;
    return 0;
}

int main()
{
    char exp[100];
    char *e, x;
    printf("Enter the expression : ");
    scanf("%s",exp);
    printf("\n");
    e = exp;

    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c ",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {

```

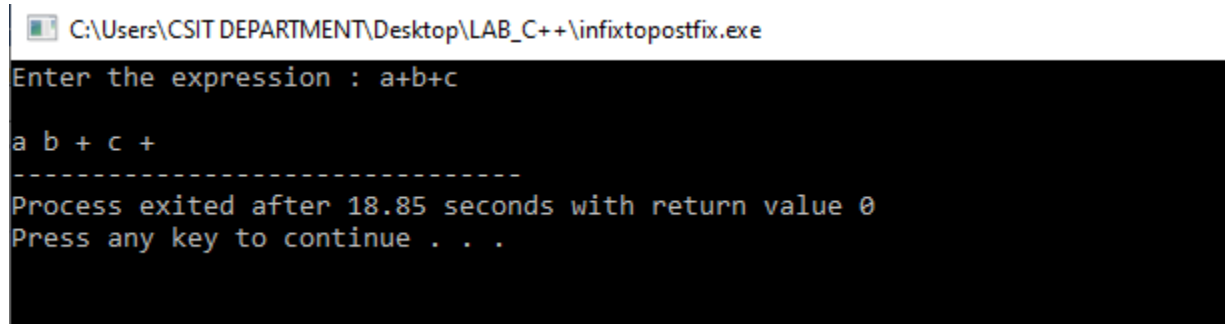
```

        while((x = pop()) != '(')
            printf("%c ", x);
    }
    else
    {
        while(priority(stack[top]) >= priority(*e))
            printf("%c ",pop());
        push(*e);
    }
    e++;
}

while(top != -1)
{
    printf("%c ",pop());
}return 0;
}

```

Output:



```

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\infixtopostfix.exe
Enter the expression : a+b+c
a b + c +
-----
Process exited after 18.85 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 10

Write a program to evaluate the postfix expression.

Source Code:

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

// Stack type
struct Stack {
    int top;
    unsigned capacity;
    int* array;
};

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

    if (!stack)
        return NULL;

    stack->top = -1;
```



```

    stack->capacity = capacity;
    stack->array
        = (int*)malloc(stack->capacity * sizeof(int));

    if (!stack->array)
        return NULL;

    return stack;
}

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

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

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

```

```
}
```

```
void push(struct Stack* stack, char op)
```

```
{
```

```
    stack->array[++stack->top] = op;
```

```
}
```

```
// The main function that returns value
```

```
// of a given postfix expression
```

```
int evaluatePostfix(char* exp)
```

```
{
```

```
    // Create a stack of capacity equal to expression size
```

```
    struct Stack* stack = createStack(strlen(exp));
```

```
    int i;
```

```
    // See if stack was created successfully
```

```
    if (!stack)
```

```
        return -1;
```

```
    // Scan all characters one by one
```

```
    for (i = 0; exp[i]; ++i) {
```

```
        // If the scanned character is an operand
```

```
        // (number here), push it to the stack.
```

```
        if (isdigit(exp[i]))
```

```

    push(stack, exp[i] - '0');

// If the scanned character is an operator,
// pop two elements from stack apply the operator
else {
    int val1 = pop(stack);
    int val2 = pop(stack);
    switch (exp[i]) {
    case '+':
        push(stack, val2 + val1);
        break;
    case '-':
        push(stack, val2 - val1);
        break;
    case '*':
        push(stack, val2 * val1);
        break;
    case '/':
        push(stack, val2 / val1);
        break;
    }
}
return pop(stack);
}

```

```
// Driver code

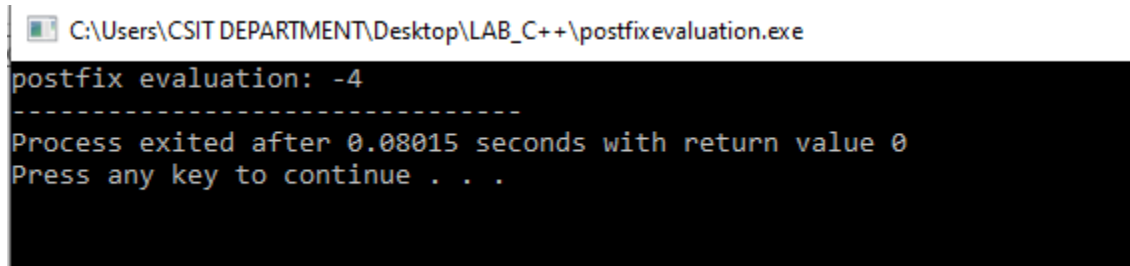
int main()
{
    char exp[] = "231*+9-";

    // Function call

    printf("postfix evaluation: %d", evaluatePostfix(exp));

    return 0;
}
```

Output:



```
C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\postfixevaluation.exe
postfix evaluation: -4
-----
Process exited after 0.08015 seconds with return value 0
Press any key to continue . . .
```

Experiment No. : 11

Write a menu driven program to implement the linear queue (its operations) using array.

Source Code:

```
#include<stdio.h>

#define n 5

int main()
{
    int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;

    printf("Queue using Array");
```

```

printf("\n1.Insertion \n2.Deletion \n3.Display ");
while(ch)
{
    printf("\nEnter the Choice:");
    scanf("%d",&ch);
    switch(ch)
    {
    case 1:
        if(rear==x)
            printf("\n Queue is Full");
        else
        {
            printf("\n Enter no %d:",j++);
            scanf("%d",&queue[rear++]);
        }
        break;
    case 2:
        if(front==rear)
        {
            printf("\n Queue is empty");
        }
        else
        {
            printf("\n Deleted Element is %d",queue[front++]);
            x++;
        }
    }
}

```

```

    }
    break;
case 3:
    printf("\nQueue Elements are:\n ");
    if(front==rear)
        printf("\n Queue is Empty");
    else
    {
        for(i=front; i<rear; i++)
        {
            printf("%d",queue[i]);
            printf("\n");
        }
        break;

    default:
        printf("Wrong Choice: please see the options");
    }
}
}
return 0;
}

```

Output:

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\linear queue.exe

```
Queue using Array
1.Insertion
2.Deletion
3.Display
Enter the Choice:1

Enter no 1:8

Enter the Choice:1

Enter no 2:8

Enter the Choice:3

Queue Elements are:
8
8
```

Experiment No. : 12

Write a program to implement the circular queue.

Source Code:

```
#include<stdio.h>

# define MAX 5

int cqueue_arr[MAX];

int front = -1;

int rear = -1;

void insert(int item)

{

if((front == 0 && rear == MAX-1) || (front == rear+1))

{

printf("Queue Overflow \n");

return;

}
```

```
if(front == -1)
{
front = 0;
rear = 0;
}
else
{
if(rear == MAX-1)
rear = 0;
else
rear = rear+1;
}
cqueue_arr[rear] = item ;
}
void deletion()
{
if(front == -1)
{
printf("Queue Underflow \n");
return ;
}
printf("Element deleted from queue is : %d \n",cqueue_arr[front]);
if(front == rear)
{
front = -1;
```



```

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

```

```

else
{
while(front_pos <= MAX-1)
{
printf("%d ",cqueue_arr[front_pos]);
front_pos++;
}
front_pos = 0;
while(front_pos <= rear_pos)
{
printf("%d ",cqueue_arr[front_pos]);
front_pos++;
}
}
printf("\n");
}

int main()
{
int choice,item;
do
{
printf("1.Insert\n");
printf("2.Delete\n");
printf("3.Display\n");
printf("4.Quit\n");

```

```
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1 :
printf("Input the element for insertion in queue : ");
scanf("%d", &item);
insert(item);
break;
case 2 :
deletion();
break;
case 3:
display();
break;
case 4:
break;
default:
printf("Wrong choice \n");
}
}while(choice!=4);
return 0;
}
```

Output:

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\Circular_queue.exe

```
1.Insertn
2.Deleten
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue : 8
1.Insertn
2.Deleten
3.Display
4.Quit
Enter your choice :
```

Experiment No. : 13

Write a program to implement the priority queue.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10

void create_queue();
void insert_element(int);
void delete_element(int);
void check_priority(int);
void display_priorityqueue();

int pqueue[MAX];
int front, rear;

int main()
```

```

{
    int n, choice;
    printf("\nEnter 1 to insert element by priority ");
    printf("\nEnter 2 to delete element by priority ");
    printf("\nEnter 3 to display priority queue ");
    printf("\nEnter 4 to exit");
    create_queue();
    while (1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                printf("\nEnter element to insert : ");
                scanf("%d",&n);
                insert_element(n);
                break;
            case 2:
                printf("\nEnter element to delete : ");
                scanf("%d",&n);
                delete_element(n);
                break;
            case 3:
                display_priorityqueue();

```

```

        break;
    case 4:
        exit(0);
    default:
        printf("\n Please enter valid choice");
    }
}
return 0;
}

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

void insert_element(int data)
{
    if (rear >= MAX - 1)
    {
        printf("\nQUEUE OVERFLOW");
        return;
    }
    if ((front == -1) && (rear == -1))
    {
        front++;
        rear++;
        pqueue[rear] = data;
    }
}

```

```

        return;
    }
    else
        check_priority(data);
    rear++;
}

void check_priority(int data)
{
    int i,j;
    for (i = 0; i <= rear; i++)
    {
        if (data >= pqueue[i])
        {
            for (j = rear + 1; j > i; j--)
            {
                pqueue[j] = pqueue[j - 1];
            }
            pqueue[i] = data;
            return;
        }
    }
    pqueue[i] = data;
}

void delete_element(int data)
{

```

```

int i;
if ((front==-1) && (rear==-1))
{
    printf("\nEmpty Queue");
    return;
}
for (i = 0; i <= rear; i++)
{
    if (data == pqueue[i])
    {
        for (; i < rear; i++)
        {
            pqueue[i] = pqueue[i + 1];
        }
        pqueue[i] = -99;
        rear--;
        if (rear == -1)
            front = -1;
        return;
    }
}
printf("\n%d element not found in queue", data);
}

void display_priorityqueue()
{

```




```

if ((front == -1) && (rear == -1))
{
    printf("\nEmpty Queue ");
    return;
}
for (; front <= rear; front++)
{
    printf(" %d ", pqueue[front]);
}
front = 0;
}

```

Output:

 C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\priority_queue.exe

```

Enter 1 to insert element by priority
Enter 2 to delete element by priority
Enter 3 to display priority queue
Enter 4 to exit
Enter your choice : 1

Enter element to insert : 8

Enter your choice : 1

Enter element to insert : 25

Enter your choice : 3
25 8
Enter your choice :

```

Experiment No. : 14

Write a program to find the factorial using recursion.

Source Code:

```
#include<stdio.h>

int fact(int);

int main()
{
    int x,n;

    printf(" Enter the Number to Find Factorial :");

    scanf("%d",&n);

    x=fact(n);

    printf(" Factorial of %d is %d",n,x);

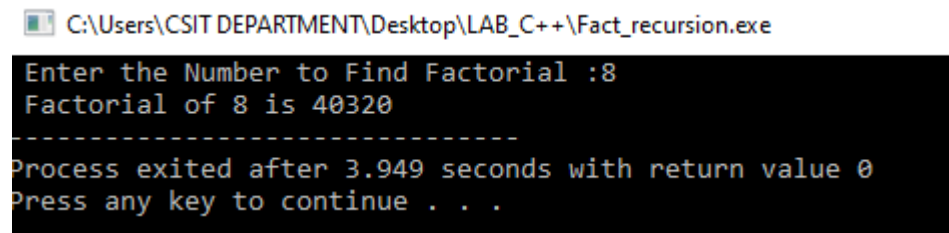
    return 0;
}

int fact(int n)
{
    if(n==0)

        return(1);

    return(n*fact(n-1));
}
```

Output:



```
C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\Fact_recursion.exe
Enter the Number to Find Factorial :8
Factorial of 8 is 40320
-----
Process exited after 3.949 seconds with return value 0
Press any key to continue . . .
```

Experiment No. : 15

Write a program to find the Fibonacci series using recursion.

Source Code:

```
#include<stdio.h>

int Fibonacci(int);

int main()
{
    int n, i = 0, c;
    printf("Enter the number : ");
    scanf("%d",&n);

    printf("Fibonacci series\n");

    for ( c = 1 ; c <= n ; c++ )
    {
        printf("%d\n", Fibonacci(i));
        i++;
    }

    return 0;
}

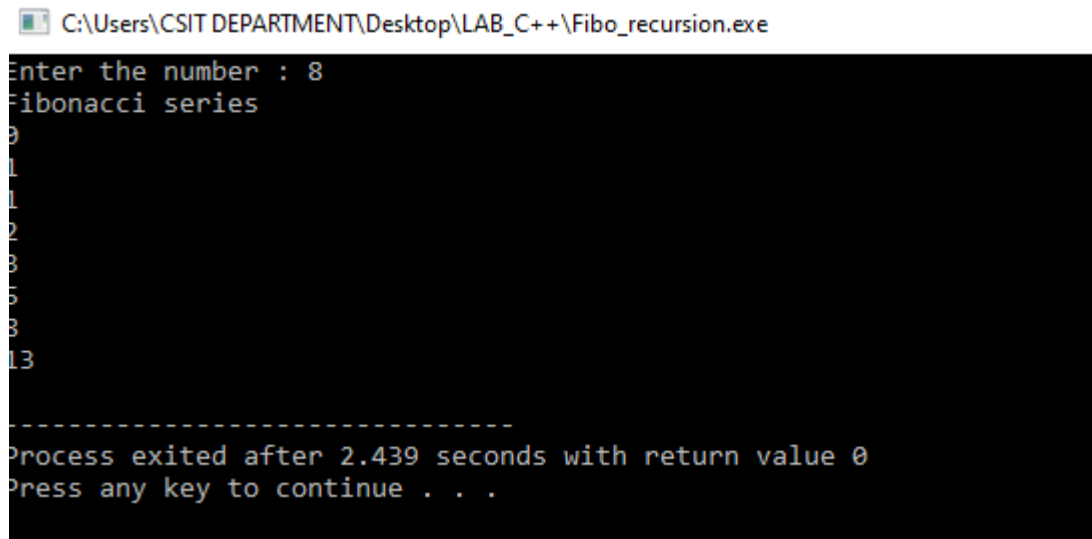
int Fibonacci(int n)
```

```

{
    if ( n == 0 )
        return 0;
    else if ( n == 1 )
        return 1;
    else
        return ( Fibonacci(n-1) + Fibonacci(n-2) );
}

```

Output:



```

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\Fibo_recursion.exe
Enter the number : 8
Fibonacci series
0
1
1
2
3
5
8
13
-----
Process exited after 2.439 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 16

Write a program to find the GCD using recursion.

Source Code:

```

#include <stdio.h>

int hcf(int n1, int n2);

int main() {
    int n1, n2;

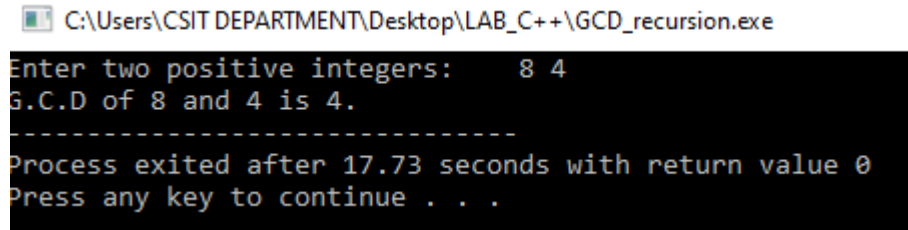
```

```

printf("Enter two positive integers: \t");
scanf("%d %d", &n1, &n2);
printf("G.C.D of %d and %d is %d.", n1, n2, hcf(n1, n2));
return 0;
}
int hcf(int n1, int n2) {
    if (n2 != 0)
        return hcf(n2, n1 % n2);
    else
        return n1;
}

```

Output:



```

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\GCD_recursion.exe
Enter two positive integers:    8 4
G.C.D of 8 and 4 is 4.
-----
Process exited after 17.73 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 17

Write a program to find the LCM using recursion.

Source Code:

```

#include <stdio.h>

int calculateLCM(int num1, int num2)
{
    static int comn = 1;

```

```

    if (comn % num1 == 0 && comn % num2 == 0)
        return comn;
    comn++;
    calculateLCM(num1, num2);
    return comn;
}

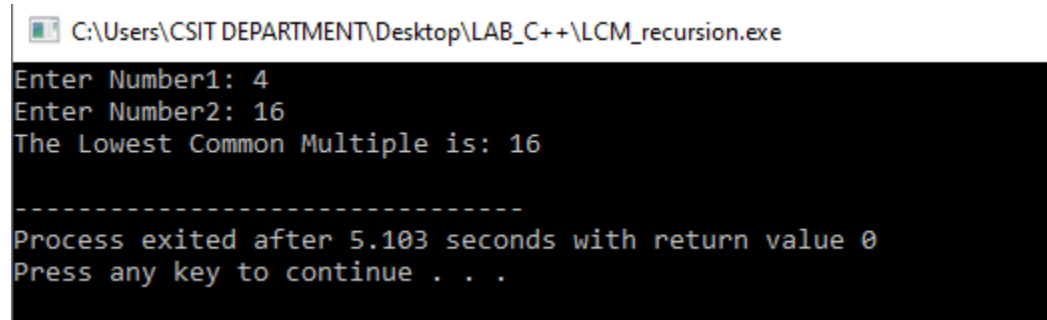
int main()
{
    int num1 = 0;
    int num2 = 0;

    printf("Enter Number1: ");
    scanf("%d", &num1);
    printf("Enter Number2: ");
    scanf("%d", &num2);

    printf("The Lowest Common Multiple is: %d\n", calculateLCM(num1, num2));
    return 0;
}

```

Output:



```

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\LCM_recursion.exe
Enter Number1: 4
Enter Number2: 16
The Lowest Common Multiple is: 16
-----
Process exited after 5.103 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 18

Write a program to simulate the TOH.

Source Code:

```
#include <stdio.h>

void toH(int n, char rodA, char rodC, char rodB)
{
    if (n == 1)
    {
        printf("\n Move disk 1 from rod %c to rod %c",rodA ,rodC );
        return;
    }
    toH(n-1, rodA, rodB, rodC);
    printf("\n Move disk %d from rod %c to rod %c", n, rodA, rodC);
    toH(n-1, rodB, rodC,rodA);
}

int main()
{
    int no_of_disks ;
    printf("Enter number of disks: ");
    scanf("%d", &no_of_disks);
    toH(no_of_disks, 'A','C','B');
    return 0;
}
```

Output:

```
C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\TOH_recursion.exe
Enter number of disks: 3
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
-----
Process exited after 2.888 seconds with return value 0
Press any key to continue . . .
```

Experiment No. : 19

Write a menu driven program to implement the singly linked list.

Source Code:

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

void begininsert ();
void lastinsert ();
void randominsert();
void begin_delete();
```



```

void last_delete();
void random_delete();
void display();
void search();
int main ()
{
    int choice =0;
    while(choice != 9)
    {
        printf("\n\n*****Main Menu*****\n");
        printf("\nChoose one option from the following list ...\n");

        printf("\n=====");

        printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random
location\n4.Delete from Beginning\n5.Delete from last\n6.Delete node after
specified location\n7.Search for an element\n8.Show\n9.Exit\n");

        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
                begininsert();
                break;
            case 2:
                lastinsert();
                break;

```

```
case 3:
    randominsert();
    break;
case 4:
    begin_delete();
    break;
case 5:
    last_delete();
    break;
case 6:
    random_delete();
    break;
case 7:
    search();
    break;
case 8:
    display();
    break;
case 9:
    exit(0);
    break;
default:
    printf("Please enter valid choice..");
}
}
```

```

}

void begininsert()
{
    struct node *ptr;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("\nNode inserted");
    }

}

void lastinsert()
{
    struct node *ptr,*temp;
    int item;

```

```
ptr = (struct node*)malloc(sizeof(struct node));
if(ptr == NULL)
{
    printf("\nOVERFLOW");
}
else
{
    printf("\nEnter value?\n");
    scanf("%d",&item);
    ptr->data = item;
    if(head == NULL)
    {
        ptr -> next = NULL;
        head = ptr;
        printf("\nNode inserted");
    }
    else
    {
        temp = head;
        while (temp -> next != NULL)
        {
            temp = temp -> next;
        }
        temp->next = ptr;
        ptr->next = NULL;
    }
}
```

```

        printf("\nNode inserted");

    }

}

void randominsert()
{
    int i,loc,item;
    struct node *ptr, *temp;
    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter element value");
        scanf("%d",&item);
        ptr->data = item;
        printf("\nEnter the location after which you want to insert ");
        scanf("\n%d",&loc);
        temp=head;
        for(i=0;i<loc;i++)
        {
            temp = temp->next;

```

```

        if(temp == NULL)
        {
            printf("\ncan't insert\n");
            return;
        }

    }

    ptr ->next = temp ->next;
    temp ->next = ptr;
    printf("\nNode inserted");
}
}

void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nList is empty\n");
    }
    else
    {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("\nNode deleted from the begining ...\n");
    }
}

```

```

    }
}
void last_delete()
{
    struct node *ptr,*ptr1;
    if(head == NULL)
    {
        printf("\nlist is empty");
    }
    else if(head -> next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nOnly node of the list deleted ...\n");
    }

    else
    {
        ptr = head;
        while(ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr ->next;
        }
        ptr1->next = NULL;
    }
}

```

```

        free(ptr);
        printf("\nDeleted Node from the last ...\n");
    }
}

void random_delete()
{
    struct node *ptr,*ptr1;
    int loc,i;

    printf("\n Enter the location of the node after which you want to perform
deletion \n");
    scanf("%d",&loc);
    ptr=head;
    for(i=0;i<loc;i++)
    {
        ptr1 = ptr;
        ptr = ptr->next;

        if(ptr == NULL)
        {
            printf("\nCan't delete");
            return;
        }
    }
    ptr1 ->next = ptr ->next;
    free(ptr);

```



```

    printf("\nDeleted node %d ",loc+1);
}
void search()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
            }
            else
            {
                flag=1;
            }
        }
    }
}

```

```

    }
    i++;
    ptr = ptr -> next;
}
if(flag==1)
{
    printf("Item not found\n");
}
}

void display()
{
    struct node *ptr;
    ptr = head;
    if(ptr == NULL)
    {
        printf("Nothing to print");
    }
    else
    {
        printf("\nprinting values . . . . \n");
        while (ptr!=NULL)
        {

```

```

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

```

Output:

```

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\Simple_Linked_list.exe

*****Main Menu*****
Choose one option from the following list ...
=====
1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete node after specified location
7.Search for an element
8.Show
9.Exit
Enter your choice?
1
Enter value
3

```

Experiment No. : 20

Write a program to implement the doubly linked list.

Source Code:

```

#include<stdio.h>
#include<stdlib.h>
struct node
{

```

```

    struct node *prev;
    struct node *next;
    int data;
};

struct node *head;
void insertion_beginning();
void insertion_last();
void insertion_specified();
void deletion_beginning();
void deletion_last();
void deletion_specified();
void display();
void search();
int main ()
{
    int choice =0;
    while(choice != 9)
    {
        printf("\n*****Main Menu*****\n");
        printf("\nChoose one option from the following list ...\n");

        printf("\n=====");

        printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random
location\n4.Delete from Beginning\n 5.Delete from last\n6.Delete the node after
the given data\n7.Search\n8.Show\n9.Exit\n");

        printf("\nEnter your choice?\n");

```

```
scanf("\n%d",&choice);
switch(choice)
{
    case 1:
        insertion_beginning();
        break;
    case 2:
        insertion_last();
        break;
    case 3:
        insertion_specified();
        break;
    case 4:
        deletion_beginning();
        break;
    case 5:
        deletion_last();
        break;
    case 6:
        deletion_specified();
        break;
    case 7:
        search();
        break;
    case 8:
```

```

        display();
        break;
    case 9:
        exit(0);
        break;
    default:
        printf("Please enter valid choice..");
    }
}
}

void insertion_beginning()
{
    struct node *ptr;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter Item value");
        scanf("%d",&item);

        if(head==NULL)

```

```

{
    ptr->next = NULL;
    ptr->prev=NULL;
    ptr->data=item;
    head=ptr;
}
else
{
    ptr->data=item;
    ptr->prev=NULL;
    ptr->next = head;
    head->prev=ptr;
    head=ptr;
}
printf("\nNode inserted\n");
}

}

void insertion_last()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node));
    if(ptr == NULL)
    {

```

```
    printf("\nOVERFLOW");
}
else
{
    printf("\nEnter value");
    scanf("%d",&item);
    ptr->data=item;
    if(head == NULL)
    {
        ptr->next = NULL;
        ptr->prev = NULL;
        head = ptr;
    }
    else
    {
        temp = head;
        while(temp->next!=NULL)
        {
            temp = temp->next;
        }
        temp->next = ptr;
        ptr ->prev=temp;
        ptr->next = NULL;
    }
}
```



```

    }
    printf("\nnode inserted\n");
}

void insertion_specified()
{
    struct node *ptr,*temp;
    int item,loc,i;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\n OVERFLOW");
    }
    else
    {
        temp=head;
        printf("Enter the location");
        scanf("%d",&loc);
        for(i=0;i<loc;i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                printf("\n There are less than %d elements", loc);
                return;
            }
        }
    }
}

```

```

    }
    printf("Enter value");
    scanf("%d",&item);
    ptr->data = item;
    ptr->next = temp->next;
    ptr -> prev = temp;
    temp->next = ptr;
    temp->next->prev=ptr;
    printf("\nnode inserted\n");
}
}
void deletion_beginning()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else

```

```

{
    ptr = head;
    head = head -> next;
    head -> prev = NULL;
    free(ptr);
    printf("\nnode deleted\n");
}

}

void deletion_last()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;

```

```

        if(ptr->next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }
}

void deletion_specified()
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the data after which the node is to be deleted : ");
    scanf("%d", &val);
    ptr = head;
    while(ptr -> data != val)
        ptr = ptr -> next;
    if(ptr -> next == NULL)
    {
        printf("\nCan't delete\n");
    }
    else if(ptr -> next -> next == NULL)
    {
        ptr ->next = NULL;
    }
}

```

```

    }
else
{
    temp = ptr -> next;
    ptr -> next = temp -> next;
    temp -> next -> prev = ptr;
    free(temp);
    printf("\nnode deleted\n");
}
}

void display()
{
    struct node *ptr;
    printf("\n printing values...\n");
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d\n",ptr->data);
        ptr=ptr->next;
    }
}

void search()
{
    struct node *ptr;
    int item,i=0,flag;

```

```

ptr = head;
if(ptr == NULL)
{
    printf("\nEmpty List\n");
}
else
{
    printf("\nEnter item which you want to search?\n");
    scanf("%d",&item);
    while (ptr!=NULL)
    {
        if(ptr->data == item)
        {
            printf("\nitem found at location %d ",i+1);
            flag=0;
            break;
        }
        else
        {
            flag=1;
        }
        i++;
        ptr = ptr -> next;
    }
    if(flag==1)

```

```

    {
        printf("\nItem not found\n");
    }
}
}

```

Output:

```

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\Double_Linked.exe
1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit
Enter your choice?
1
Enter Item value8
Node inserted

```

Experiment No. : 21

Write a program to implement the circular linked list.

Source Code:

```

#include<stdio.h>
#include<stdlib.h>
struct Node;
typedef struct Node * PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;
struct Node

```

```

{
    int e;
    Position next;
};

void Insert(int x, List l, Position p)
{
    Position TmpCell;
    TmpCell = (struct Node*) malloc(sizeof(struct Node));
    if(TmpCell == NULL)
        printf("Memory out of space\n");
    else
    {
        TmpCell->e = x;
        TmpCell->next = p->next;
        p->next = TmpCell;
    }
}

int isLast(Position p, List l)
{
    return (p->next == l);
}

Position FindPrevious(int x, List l)
{
    Position p = l;
    while(p->next != l && p->next->e != x)

```



```

        p = p->next;
    return p;
}
Position Find(int x, List l)
{
    Position p = l->next;
    while(p != l && p->e != x)
        p = p->next;
    return p;
}
void Delete(int x, List l)
{
    Position p, TmpCell;
    p = FindPrevious(x, l);
    if(!isLast(p, l))
    {
        TmpCell = p->next;
        p->next = TmpCell->next;
        free(TmpCell);
    }
    else
        printf("Element does not exist!!!\n");
}
void Display(List l)
{

```

```

printf("The list element are :: ");
Position p = l->next;
while(p != l)
{
    printf("%d -> ", p->e);
    p = p->next;
}
}

int main()
{
    int x, pos, ch, i;
    List l, ll;
    l = (struct Node *) malloc(sizeof(struct Node));
    l->next = l;
    List p = l;
    printf("CIRCULAR LINKED LIST IMPLEMENTATION OF LIST \n\n");
    do
    {
        printf("\n\n1. INSERT\t 2. DELETE\t 3. FIND\t 4. PRINT\t 5.
QUIT\n\nEnter the choice :: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:
                p = l;

```

```
printf("Enter the element to be inserted :: ");
scanf("%d",&x);
printf("Enter the position of the element :: ");
scanf("%d",&pos);
for(i = 1; i < pos; i++)
{
    p = p->next;
}
Insert(x,l,p);
break;
```

case 2:

```
p = l;
printf("Enter the element to be deleted :: ");
scanf("%d",&x);
Delete(x,p);
break;
```

case 3:

```
p = l;
printf("Enter the element to be searched :: ");
scanf("%d",&x);
p = Find(x,p);
if(p == l)
    printf("Element does not exist!!!\n");
```

```

        else
            printf("Element exist!!!\n");
            break;

    case 4:
        Display(l);
        break;

    }
}while(ch<5);
return 0;
}

```

Output:

```

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\Circular_linked_list.exe
CIRCULAR LINKED LIST IMPLEMENTATION OF LIST ADT

1. INSERT      2. DELETE      3. FIND      4. PRINT      5. QUIT

Enter the choice :: 1
Enter the element to be inserted :: 8
Enter the position of the element :: 1

1. INSERT      2. DELETE      3. FIND      4. PRINT      5. QUIT

Enter the choice :: 2
Enter the element to be deleted :: 8

```

Experiment No. : 22

Write a program to implement the following sorting:

- i. Bubble Sort ii. Selection Sort iii. Insertion Sort*
- iv. Shell Sort*

Source Code:

----- (Do Yourself) -----

Experiment No. : 23

Write a program to implement divide and conquer algorithms in following sorting: i. Merge Sort ii. Quick Sort iii. Heap Sort

Source Code:

----- (Do Yourself) -----

Experiment No. : 24

Write a program to implement the sequential and binary search.

Source Code:

```
#include <stdio.h>

/* Function for sequential search */
void sequential_search(int array[], int size, int n)
{
    int i;
    for (i = 0; i < size; i++)
    {
        if (array[i] == n)
        {
            printf("%d found at location %d.\n", n, i+1);
            break;
        }
    }
    if (i == size)
        printf("Not found! %d is not present in the list.\n", n);
}
```

```

}

/* End of sequential_search() */

/* Function for binary search */
void binary_search(int array[], int size, int n)
{
    int i, first, last, middle;
    first = 0;
    last = size - 1;
    middle = (first+last) / 2;

    while (first <= last) {
        if (array[middle] < n)
            first = middle + 1;
        else if (array[middle] == n) {
            printf("%d found at location %d.\n", n, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last) / 2;
    }
    if ( first > last )
        printf("Not found! %d is not present in the list.\n", n);
}

```

```

}

/* End of binary_search() */

/* The main() begins */
int main()
{
    int a[200], i, j, n, size;
    printf("Enter the size of the list:");
    scanf("%d", &size);
    printf("Enter %d Integers in ascending order\n", size);
    for (i = 0; i < size; i++)
        scanf("%d", &a[i]);
    printf("Enter value to find\n");
    scanf("%d", &n);
    printf("Sequential search\n");
    sequential_search(a, size, n);
    printf("Binary search\n");
    binary_search(a, size, n);
    return 0;
}

```

Output:

```
C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\sequential_binarySearch.exe
Enter the size of the list:5
Enter 5 Integers in ascending order
12
14
16
17
18
Enter value to find
14
Sequential search
14 found at location 2.
Binary search
14 found at location 2.

-----
Process exited after 36.24 seconds with return value 0
Press any key to continue . . .
```

Experiment No. : 25

Write a program to implement the tree traversal method.

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

// Structure of a node of tree
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

// Create a New Node
struct node* createNode(int data)
{
```



```

// Allocate memory equivalent to node structure and hold address in node pointer
struct node* newNode = (struct node*)malloc(sizeof(struct node));
newNode -> data = data;
newNode -> left = NULL;
newNode -> right = NULL;
return newNode;
}

// Insert a new node to left of the given node
struct node* insertLeft(struct node* root, int data)
{
    root -> left = createNode(data);
    return root;
}

// Insert a new node to right of the given node
struct node* insertRight(struct node* root, int data)
{
    root -> right = createNode(data);
    return root;
}

// Inorder traversal
void inorder(struct node* root)
{
    if (root == NULL) return;
    inorder(root -> left);
    printf("%d \t", root -> data);
}

```

```

        inorder(root -> right);
    }
// Preorder traversal
void preorder(struct node* root)
{
    if (root == NULL) return;
    printf("%d \t", root -> data);
    preorder(root -> left);
    preorder(root -> right);
}
// Postorder traversal
void postorder(struct node* root)
{
    if (root == NULL) return;
    postorder(root -> left);
    postorder(root -> right);
    printf("%d \t", root -> data);
}
// Main Code
int main()
{
    struct node* root = createNode(4);
    insertLeft(root, 21);
    insertRight(root, 13);
    insertLeft(root -> left, 34);

```

```

insertRight(root -> left, 0);
insertLeft(root -> right, 18);
insertRight(root -> right, 19);
printf("Inorder traversal of the Tree is : ");
inorder(root);
printf("\n");
    printf("Preorder traversal of the Tree is : ");
preorder(root);
printf("\n");
printf("Postorder traversal of the Tree is : ");
postorder(root);
return 0;
}

```

Output:

```

Inorder traversal of the Tree is : 34  21  0  4  18  13  19
Preorder traversal of the Tree is : 4  21  34  0  13  18  19
Postorder traversal of the Tree is : 34  0  21  18  19  13  4
-----
Process exited after 0.05816 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 26

Write a program to perform the following operations:

- I. Insert an element into a BST.
- II. Delete an element from a BST.
- III. Search for a key element in a BST.

Source Code:

```

#include <stdio.h>

#include <stdlib.h>

```

```

struct node {
    int data; //node will store some data
    struct node *right_child; // right child
    struct node *left_child; // left child
};

//function to create a node
struct node* new_node(int x) {
    struct node *temp = (struct node*)malloc(sizeof(struct node));
    temp -> data = x;
    temp -> left_child = NULL;
    temp -> right_child = NULL;
    return temp;
}

// searching operation
struct node* search(struct node * root, int x) {
    if (root == NULL || root -> data == x) //if root->data is x then the element is found
        return root;
    else if (x > root -> data) // x is greater, so we will search the right subtree
        return search(root -> right_child, x);
    else //x is smaller than the data, so we will search the left subtree
        return search(root -> left_child, x);
}

// insertion
struct node* insert(struct node * root, int x) {
    //searching for the place to insert

```

```

if (root == NULL)
    return new_node(x);
else if (x > root->data) // x is greater. Should be inserted to the right
    root->right_child = insert(root->right_child, x);
else // x is smaller and should be inserted to left
    root->left_child = insert(root->left_child, x);
return root;
}

//function to find the minimum value in a node
struct node* find_minimum(struct node * root) {
    if (root == NULL)
        return NULL;
    else if (root->left_child != NULL) // node with minimum value will have no left
child
        return find_minimum(root->left_child); // left most element will be minimum
    return root;
}

// deletion
struct node* delet(struct node * root, int x) {
    //searching for the item to be deleted
    if (root == NULL)
        return NULL;
    if (x > root->data)
        root->right_child = delet(root->right_child, x);
    else if (x < root->data)

```

```

    root -> left_child = delet(root -> left_child, x);
else {
    //No Child node
    if (root -> left_child == NULL && root -> right_child == NULL) {
        free(root);
        return NULL;
    }
    //One Child node
    else if (root -> left_child == NULL || root -> right_child == NULL) {
        struct node *temp;
        if (root -> left_child == NULL)
            temp = root -> right_child;
        else
            temp = root -> left_child;
        free(root);
        return temp;
    }
    //Two Children
    else {
        struct node *temp = find_minimum(root -> right_child);
        root -> data = temp -> data;
        root -> right_child = delet(root -> right_child, temp -> data);
    }
}

return root;

```

```

}

// Inorder Traversal

void inorder(struct node *root) {
    if (root != NULL) // checking if the root is not null
    {
        inorder(root -> left_child); // traversing left child
        printf(" %d ", root -> data); // printing data at root
        inorder(root -> right_child); // traversing right child
    }
}

int main() {
    struct node *root;
    root = new_node(20);
    insert(root, 5);
    insert(root, 1);
    insert(root, 15);
    insert(root, 9);
    insert(root, 7);
    insert(root, 12);
    insert(root, 30);
    insert(root, 25);
    insert(root, 40);
    insert(root, 45);
    insert(root, 42);
    inorder(root);
}

```

```

printf("\n");
printf("%d",delet(root, 1));
printf("\n");
root = delet(root, 40);
root = delet(root, 45);
root = delet(root, 9);
inorder(root);
printf("\n");
return 0;
}

```

Output:

```

1  5  7  9 12 15 20 25 30 40 42 45
11932672
5  7 12 15 20 25 30 42
-----
Process exited after 0.04987 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 27

Menu-driven Program for the AVL Tree.

OR

Write a program to perform the following operations:

- i. Insert an element into an AVL tree.
- ii. Delete an element from an AVL tree.
- iii. Search for a key element in an AVL tree.

Source Code:

```
#include<conio.h>
```



```

#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
    int data;
    struct node *left,*right;
    int ht;
}node;
node *insert(node *,int);
node *Delete(node *,int);
void preorder(node *);
void inorder(node *);
int height( node *);
node *rotateright(node *);
node *rotateleft(node *);
node *RR(node *);
node *LL(node *);
node *LR(node *);
node *RL(node *);
int BF(node *);
int main()
{
    node *root=NULL;
    int x,n,i,op;
    do

```

```

{
    printf("\n");
    printf("\n1) Create the AVL Tree");
    printf("\n2) Insert Element into the AVL Tree");
    printf("\n3) Delete Element from the AVL Tree ");
    printf("\n4) Print the AVL Tree");
    printf("\n5) Quit");
    printf("\nEnter Your Choice: ");
    scanf("%d",&op);
    switch(op)
    {
        case 1:printf("\nEnter Total Number of Elements in the AVL Tree: ");
            scanf("%d",&n);
            printf("\n Enter AVL Tree Elements: ");
            root=NULL;
            for(i=0;i<n;i++)
            {
                scanf("%d",&x);
                root=insert(root,x);
            }
            break;
        case 2:printf("\nEnter a Element to Insert in the AVL Tree: ");
            scanf("%d",&x);
            root=insert(root,x);
            break;
    }
}

```

```

        case 3:printf("\nEnter a Element to Delete from the AVL Tree: ");
                scanf("%d",&x);
                root=Delete(root,x);
                break;
        case 4:  printf("\nPreorder Sequence of the AVL Tree:\n");
                preorder(root);
                printf("\nInorder sequence of the AVL Tree:\n");
                inorder(root);
                break;
    }
}while(op!=5);
return 0;
}

```

```

node * insert(node *T,int x)
{
    if(T==NULL)
    {
        T=(node*)malloc(sizeof(node));
        T->data=x;
        T->left=NULL;
        T->right=NULL;
    }
    else
        if(x > T->data)           // Insert in Right Subtree

```

```

{
    T->right=insert(T->right,x);
    if(BF(T)==-2)
        if(x>T->right->data)
            T=RR(T);
        else
            T=RL(T);
    }
else
    if(x<T->data)
    {
        T->left=insert(T->left,x);
        if(BF(T)==2)
            if(x < T->left->data)
                T=LL(T);
            else
                T=LR(T);
        }
    T->ht=height(T);
    return(T);
}

```

node * Delete(node *T,int x)

```

{
    node *p;

```

```

if(T==NULL)
{
    return NULL;
}
else

    if(x > T->data)           // Insert in Right Subtree
    {
        T->right=Delete(T->right,x);
        if(BF(T)==2)
            if(BF(T->left)>=0)
                T=LL(T);
            else
                T=LR(T);
    }
    else
        if(x<T->data)
        {
            T->left=Delete(T->left,x);
            if(BF(T)==-2)           // Rebalance during windup
                if(BF(T->right)<=0)
                    T=RR(T);
                else
                    T=RL(T);
        }
}

```

```

else
{
    // Element to be deleted is found
    if(T->right !=NULL)
    { // Delete its inorder succesor
        p=T->right;
        while(p->left != NULL)
            p=p->left;

        T->data=p->data;
        T->right=Delete(T->right,p->data);
        if(BF(T)==2)           // Rebalance during windup
            if(BF(T->left)>=0)
                T=LL(T);
            else
                T=LR(T);
    }
    else
        return(T->left);

}

T->ht=height(T);
return(T);
}

```

```

int height(node *T)
{
    int lh,rh;
    if(T==NULL)
        return(0);
    if(T->left==NULL)
        lh=0;
    else
        lh=1+T->left->ht;
    if(T->right==NULL)
        rh=0;
    else
        rh=1+T->right->ht;
    if(lh>rh)
        return(lh);
    return(rh);
}

```

```

node * rotateright(node *x)
{
    node *y;
    y=x->left;
    x->left=y->right;
    y->right=x;
    x->ht=height(x);
}

```

```
    y->ht=height(y);  
    return(y);  
}
```

```
node * rotateleft(node *x)  
{  
    node *y;  
    y=x->right;  
    x->right=y->left;  
    y->left=x;  
    x->ht=height(x);  
    y->ht=height(y);  
    return(y);  
}
```

```
node * RR(node *T)  
{  
    T=rotateleft(T);  
    return(T);  
}
```

```
node * LL(node *T)  
{  
    T=rotateright(T);  
    return(T);  
}
```



```

node * LR(node *T)
{
    T->left=rotateleft(T->left);
    T=rotateright(T);
    return(T);
}

node * RL(node *T)
{
    T->right=rotateright(T->right);
    T=rotateleft(T);
    return(T);
}

```

```

int BF(node *T)
{
    int lh,rh;
    if(T==NULL)
        return(0);
    if(T->left==NULL)
        lh=0;
    else
        lh=1+T->left->ht;
    if(T->right==NULL)
        rh=0;
    else

```

```

        rh=1+T->right->ht;
    return(lh-rh);
}

void preorder(node *T)
{
    if(T!=NULL)
    {
        printf(" %d(Bf=%d)",T->data,BF(T));
        preorder(T->left);
        preorder(T->right);
    }
}

void inorder(node *T)
{
    if(T!=NULL)
    {
        inorder(T->left);
        printf(" %d(Bf=%d)",T->data,BF(T));
        inorder(T->right);
    }
}

```

Source Code:

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\AVL_tree.exe

```
1) Create the AVL Tree
2) Insert Element into the AVL Tree
3) Delete Element from the AVL Tree
4) Print the AVL Tree
5) Quit
Enter Your Choice: 1
```

Enter Total Number of Elements in the AVL Tree: 5

Enter AVL Tree Elements: 6

```
12
14
5
7
```

```
1) Create the AVL Tree
2) Insert Element into the AVL Tree
3) Delete Element from the AVL Tree
4) Print the AVL Tree
5) Quit
Enter Your Choice: 4
```

Preorder Sequence of the AVL Tree:

12(Bf=1) 6(Bf=0) 5(Bf=0) 7(Bf=0) 14(Bf=0)

Inorder sequence of the AVL Tree:

5(Bf=0) 6(Bf=0) 7(Bf=0) 12(Bf=1) 14(Bf=0)

Experiment No. : 28

Write a program to implement the BFS and DFS traversal.

Source Code_ (BFS):

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

```
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
```

```
void bfs(int v) {
```

```
    for (i=1;i<=n;i++)
```

```
        if(a[v][i] && !visited[i])
```

```
            q[++r]=i;
```

```
    if(f<=r) {
```

```
        visited[q[f]]=1;
```

```
        bfs(q[f++]);
```

```

    }
}

int main() {
    int v;
    printf("\n Enter the number of vertices:");
    scanf("%d",&n);
    for (i=1;i<=n;i++) {
        q[i]=0;
        visited[i]=0;
    }
    printf("\n Enter graph data in matrix form:\n");
    for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
            scanf("%d",&a[i][j]);
    printf("\n Enter the starting vertex:");
    scanf("%d",&v);
    bfs(v);
    printf("\n The node which are reachable are:\n");
    for (i=1;i<=n;i++)
        if(visited[i])
            printf("%d\t",i); else
            printf("\n Bfs is not possible");
return 0;
}

```

Output:

C:\Users\CSIT DEPARTMENT\Desktop\LAB_C++\BFS.exe

```
Enter the number of vertices:3
Enter graph data in matrix form:
4      2      3
2      4      1
1      0
0

Enter the starting vertex:2

The node which are reachable are:
1      2      3
-----
Process exited after 43.31 seconds with return value 0
Press any key to continue . . .
```

Source Code_ (DFS): Using [Adjacency Matrix]

```
#include<stdio.h>

void DFS(int);

int G[10][10],visited[10],n; //n is no of vertices and graph is sorted in array
G[10][10]

int main()
{
    int i,j;

    printf("Enter number of vertices:");

    scanf("%d",&n);

    //read the adjecency matrix

    printf("\nEnter adjecency matrix of the graph:");
```

```

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

//visited is initialized to zero
for(i=0;i<n;i++)
    visited[i]=0;

DFS(0);

return 0;
}

void DFS(int i)
{
    int j;
printf("\n%d",i);
    visited[i]=1;
for(j=0;j<n;j++)
    if(!visited[j]&&G[i][j]==1)
        DFS(j);
}

```

Output:

```

Enter number of vertices:8
Enter adjacency matrix of the graph:0 1 1 1 1 0 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 0 0 0 0 1
0 0 0 1 1 0 0 1
0 0 0 0 0 1 1 0

0
1
5
2
7
6
3
4
Process returned 8 (0x8)   execution time : 64.785 s
Press any key to continue.

```

Source Code_ (DFS): Using [Adjacency List]

```

#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
    struct node *next;
    int vertex;
}node;
node *G[20];
//heads of linked list
int visited[20];
int n;
void read_graph();
//create adjacency list

```

```

void insert(int,int);
//insert an edge (vi,vj) in te adjacency list
void DFS(int);
int main()
{
    int i;
    read_graph();
    //initialised visited to 0

for(i=0;i<n;i++)
    visited[i]=0;
    DFS(0);
    return 0;
}
void DFS(int i)
{
    node *p;
printf("\n%d",i);
    p=G[i];
    visited[i]=1;
    while(p!=NULL)
    {
        i=p->vertex;
        if(!visited[i])
            DFS(i);
    }
}

```



```

        p=p->next;
    }
}

void read_graph()
{
    int i,vi,vj,no_of_edges;
    printf("Enter number of vertices:");

scanf("%d",&n);
    //initialise G[] with a null
    for(i=0;i<n;i++)
    {
        G[i]=NULL;
        //read edges and insert them in G[]

printf("Enter number of edges:");
        scanf("%d",&no_of_edges);
        for(i=0;i<no_of_edges;i++)
        {
            printf("Enter an edge(u,v):");
scanf("%d%d",&vi,&vj);
            insert(vi,vj);
        }
    }
}

```

```

void insert(int vi,int vj)
{
    node *p,*q;

    //acquire memory for the new node
    q=(node*)malloc(sizeof(node));
    q->vertex=vj;
    q->next=NULL;
    //insert the node in the linked list number vi
    if(G[vi]==NULL)
        G[vi]=q;
    else
    {
        //go to end of the linked list
        p=G[vi];

        while(p->next!=NULL)
            p=p->next;
        p->next=q;
    }
}

```

Output:

```

Enter number of vertices:8
Enter number of edges:10
Enter an edge(u,v):0 1
Enter an edge(u,v):0 2
Enter an edge(u,v):0 3
Enter an edge(u,v):0 4
Enter an edge(u,v):1 5
Enter an edge(u,v):2 5
Enter an edge(u,v):3 6
Enter an edge(u,v):4 6
Enter an edge(u,v):5 7
Enter an edge(u,v):6 7

0
1
5
7
2
3
6
4
Process returned 0 (0x0)   execution time : 28.955 s
Press any key to continue.

```

Experiment No. : 29

Write a program to find MST using prim's and kruskal's algorithm.

Source Code: [using Prim's Algorithm]

```

#include<stdio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

int main()
{
    printf("\nEnter the number of nodes:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");

    for(i=1;i<=n;i++)

```

```

for(j=1;j<=n;j++)
{
    scanf("%d",&cost[i][j]);
    if(cost[i][j]==0)
        cost[i][j]=999;
}
visited[1]=1;
printf("\n");
while(ne < n)
{
    for(i=1,min=999;i<=n;i++)
        for(j=1;j<=n;j++)
            if(cost[i][j]< min)
                if(visited[i]!=0)
                {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
                }
    if(visited[u]==0 || visited[v]==0)
    {
        printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
        mincost+=min;
        visited[b]=1;
    }
}

```

```

    }
    cost[a][b]=cost[b][a]=999;
}

printf("\n Minimun cost=%d",mincost);

return 0;
}

```

Output:

```

Enter the number of nodes:3
Enter the adjacency matrix:
3 2 4
1 2 4
1 4 6

Edge 1:(1 2) cost:2
Edge 2:(1 3) cost:4
Minimun cost=6
-----
Process exited after 35.14 seconds with return value 0
Press any key to continue . . .

```

Source Code: [using Kruskal's Algorithm]

```

#include<stdio.h>

#include<stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

int main()

{

    printf("\n\tImplementation of Kruskal's algorithm\n");

```

```

printf("\nEnter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the cost adjacency matrix:\n");
for(i=1;i<=n;i++)
{
    for(j=1;j<=n;j++)
    {
        scanf("%d",&cost[i][j]);
        if(cost[i][j]==0)
            cost[i][j]=999;
    }
}
printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
{
    for(i=1,min=999;i<=n;i++)
    {
        for(j=1;j <= n;j++)
        {
            if(cost[i][j] < min)
            {
                min=cost[i][j];
                a=u=i;
                b=v=j;
            }
        }
    }
}

```

```

        }
    }
    u=find(u);
    v=find(v);
    if(uni(u,v))
    {
        printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
        mincost +=min;
    }
    cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
return 0;
}
int find(int i)
{
    while(parent[i])
        i=parent[i];
    return i;
}
int uni(int i,int j)
{
    if(i!=j)
    {
        parent[j]=i;
    }
}

```

```

        return 1;
    }
    return 0;
}

```

Output:

```

    Implementation of Kruskal's algorithm
Enter the no. of vertices:3
Enter the cost adjacency matrix:
1 2 4
2 4 6
3 2 1
The edges of Minimum Cost Spanning Tree are
1 edge (1,2) =2
2 edge (3,2) =2

    Minimum cost = 4

-----
Process exited after 24.84 seconds with return value 0
Press any key to continue . . .

```

Experiment No. : 30

Write a program to find shortest path using Dijkstra's algorithm.

Source Code:

```

#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
    int G[MAX][MAX],i,j,n,u;

```



```

printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{
int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
//pred[] stores the predecessor of each node
//count gives the number of nodes seen so far
//create the cost matrix
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
//initialize pred[],distance[] and visited[]

```

```

for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
mindistance=INFINITY;
//nextnode gives the node at minimum distance
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
//check if a better path exists through nextnode
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{

```

```

distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}
//print the path and distance of each node
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
}while(j!=startnode);
}
}

```

Output:

```
Enter no. of vertices:3
Enter the adjacency matrix:
12 15 17
14 20 32
10 15 37

Enter the starting node:0

Distance of node1=15
Path=1<-0
Distance of node2=17
Path=2<-0
-----
Process exited after 62.95 seconds with return value 0
Press any key to continue . . .
```

Experiment No. : 31

Write a program to implement the Stack and Queue using linked list.

(Do Yourself)[Assignment]

Experiment No. : 32

Write a program to Subtract two matrix A and B.

(Do Yourself)[Assignment]

********THE - END********