NITRA TECHNICAL CAMPUS, GHAZIABAD

College Code-802

Department of Computer Science and Engineering

SESSION 2023-24

Year: II **Semester: III**

Data Structure Lab (BCS-351)

LAB FILE

Submitted To:

DIVYA PACHAURI
Asst. Prof.
CSE(AIML) Department

Submitted By:

Student name: Ankit kushwaha
Roll No: 2208020100024
Branch-Year: CSE-2nd

NDEX

14.	Write a program for implementation of addition and deletion in Btree operation.	67-81	20/12/23	
15.	Write a program for graph implementation in C.	82-83	03/01/24	
16.	Write a program for BFS algorithm implementation in C.	84-90	03/01/24	
17.	Write a program for prim's algorithm implementation in C.	91-93	10/01/24	
18.	Write a program for kruskal's algorithm implementation in C.	94-99	10/01/24	
19.	Write a program for Dijkstra algorithm implementation in C.	100-104	17/01/24	
20.	Write a program for Floyd Warshall algorithm implementation in C.	105-107	17/01/24	

1.) Write a Program to add two matrices and perform the multiplication operation on the same matrix.

Source Code-

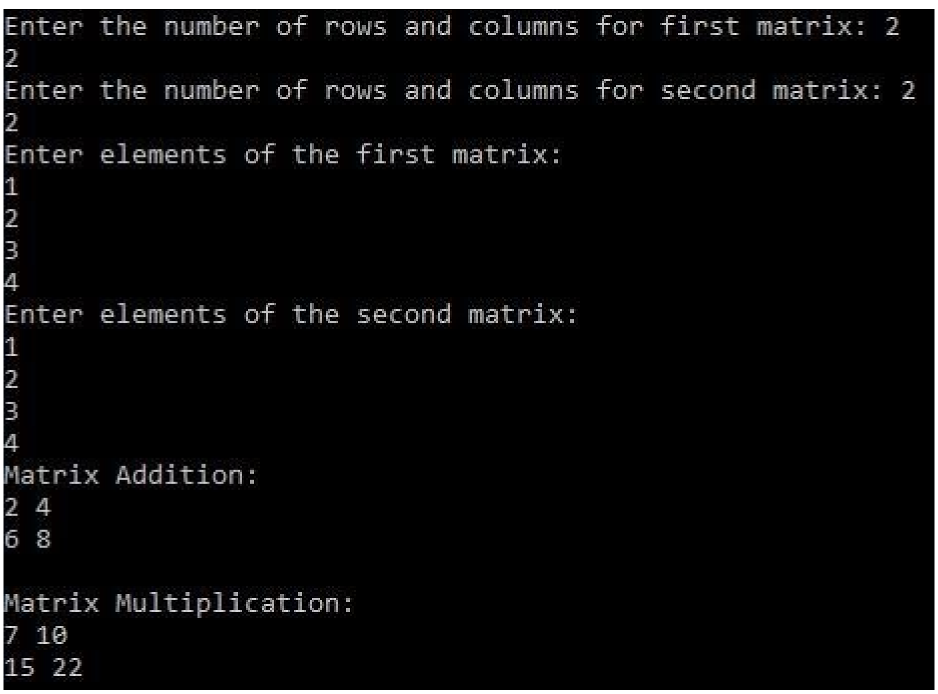
```
#include <stdio.h>

#define MAX_SIZE 10

void addMatrices(int rows, int cols, int matrix1[MAX_SIZE][MAX_SIZE], int
matrix2[MAX_SIZE][MAX_SIZE], int result[MAX_SIZE][MAX_SIZE])

{

    for (int i = 0; i < rows; i++)

    {

        for (int j = 0; j < cols; j++)

        {

            result[i][j] = matrix1[i][j] + matrix2[i][j];

        }

    }

}

void multiplyMatrices(int rows1, int cols1, int cols2, int matrix1[MAX_SIZE][MAX_SIZE], int
matrix2[MAX_SIZE][MAX_SIZE], int result[MAX_SIZE][MAX_SIZE])

{

    for (int i = 0; i < rows1; i++)

    {

        for (int j = 0; j < cols2; j++)

        {

            result[i][j] = 0;

            for (int k = 0; k < cols1; k++)
```

```
            {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }
}
void displayMatrix(int rows, int cols, int matrix[MAX_SIZE][MAX_SIZE])
{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
int main()
{
int rows1, cols1, rows2, cols2;
    printf("Enter the number of rows and columns for first matrix: ");
    scanf("%d %d", &rows1, &cols1);
    printf("Enter the number of rows and columns for second matrix: ");
    scanf("%d %d", &rows2, &cols2);
```

```
    if (cols1 != rows2)
    {
        printf("Matrices cannot be multiplied due to incompatible dimensions.\n");
        return 1;
    }
    if (rows1 > MAX_SIZE || cols1 > MAX_SIZE || rows2 > MAX_SIZE || cols2 > MAX_SIZE)
    {
        printf("Matrix size exceeds maximum limit.\n");
        return 1;
    }
    int matrix1[MAX_SIZE][MAX_SIZE], matrix2[MAX_SIZE][MAX_SIZE],
resultAddition[MAX_SIZE][MAX_SIZE], resultMultiplication[MAX_SIZE][MAX_SIZE];
    printf("Enter elements of the first matrix:\n");
    for (int i = 0; i < rows1; i++)
    {
        for (int j = 0; j < cols1; j++)
        {
            scanf("%d", &matrix1[i][j]);
        }
    }
printf("Enter elements of the second matrix:\n");
for (int i = 0; i < rows2; i++)
    {
        for (int j = 0; j < cols2; j++)
        {
```

```
            scanf("%d", &matrix2[i][j]);
        }
    }

    addMatrices(rows1, cols1, matrix1, matrix2, resultAddition);
    printf("Matrix Addition:\n");
    displayMatrix(rows1, cols1, resultAddition);
    multiplyMatrices(rows1, cols1, cols2, matrix1, matrix2, resultMultiplication);
    printf("Matrix Multiplication:\n");
    displayMatrix(rows1, cols2, resultMultiplication);
    return 0;
}
```

Output-



```
Enter the number of rows and columns for first matrix: 2
2
Enter the number of rows and columns for second matrix: 2
2
Enter elements of the first matrix:
1
2
3
4
Enter elements of the second matrix:
1
2
3
4
Matrix Addition:
2 4
6 8

Matrix Multiplication:
7 10
15 22
```

2.) Write a Program for representation of linked list in C.

Source Code-

```
#include<stdlib.h>

#include <stdio.h>

void create();

void display();

void insert_begin();

void insert_end();

void insert_pos();

void delete_begin();

void delete_end();

void delete_pos();

struct node

{

 int info;

 struct node *next;

};

struct node *start=NULL;

int main()

{

 int choice;

 while(1)

{

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

```
printf("\n 1.Create \n");

printf("\n 2.Display \n");

printf("\n 3.Insert at the beginning \n");

printf("\n 4.Insert at the end \n ");

printf("\n 5.Insert at specified position \n ");

printf("\n 6.Delete from beginning \n ");

printf("\n 7.Delete from the end \n ");

printf("\n 8.Delete from specified position \n ");

printf("\n 9.Exit \n");

printf("\nEnter your choice:\t");

scanf("%d",&choice);

switch(choice)

{

 case 1:

 create();

 break;

 case 2:

 display();

 break;

 case 3:

 insert_begin();

 break;

 case 4:

 insert_end();

 break;
```

```
 case 5:
 insert_pos();
 break;
 case 6:
 delete_begin();
 break;
 case 7:
 delete_end();
 break;
 case 8:
 delete_pos();
 break;
 case 9:
 exit(0);
 break;
 default:
 printf("\n Wrong Choice:\n");
 break;
 }
}
return 0;
}
void create()
{
 struct node *temp,*ptr;
```

```
 temp=(struct node *)malloc(sizeof(struct node));
 if(temp==NULL)
 {
  printf("\nOut of Memory Space:\n");
  exit(0);
 }
 printf("\nEnter the data value for the node:\t");
 scanf("%d",&temp->info);
 temp->next=NULL;
 if(start==NULL)
 {
  start=temp;
 }
 else
 {
  ptr=start;
  while(ptr->next!=NULL)
 {
 ptr=ptr->next;
 }
 ptr->next=temp;
 }
}
void display()
{
```

```
 struct node *ptr;
 if(start==NULL)
{
 printf("\nList is empty:\n");
 return;
 }
 else
 {
ptr=start;
printf("\nThe List elements are:\n");
while(ptr!=NULL)
{
printf("%dt",ptr->info );
ptr=ptr->next ;
}
}
}
void insert_begin()
{
struct node *temp;
temp=(struct node *)malloc(sizeof(struct node));
if(temp==NULL)
{
printf("\nOut of Memory Space:\n");
return;
```

```
}
printf("\nEnter the data value for the node:\t" );
scanf("%d",&temp->info);
temp->next =NULL;
if(start==NULL)
{
start=temp;
}
else
{
temp->next=start;
start=temp;
}
}
void insert_end()
{
struct node *temp,*ptr;
temp=(struct node *)malloc(sizeof(struct node));
if(temp==NULL)
{
printf("\nOut of Memory Space:\n");
return;
}
printf("\nEnter the data value for the node:\t" );
scanf("%d",&temp->info );
```

```
temp->next =NULL;
if(start==NULL)
{
start=temp;
}
else
{
ptr=start;
while(ptr->next !=NULL)
{
ptr=ptr->next ;
}
ptr->next =temp;
}
}
void insert_pos()
{
struct node *ptr,*temp;
int i,pos;
temp=(struct node *)malloc(sizeof(struct node));
if(temp==NULL)
{
printf("\nOut of Memory Space:\n");
return;
}
```

```
printf("\nEnter the position for the new node to be inserted:\t");

scanf("%d",&pos);

printf("\nEnter the data value of the node:\t");

scanf("%d",&temp->info) ;

temp->next=NULL;

if(pos==0)

{

temp->next=start;

start=temp;

}

else

{

for(i=0,ptr=start;i<pos-1;i++) { ptr=ptr->next;

if(ptr==NULL)

{

printf("\nPosition not found:\n");

return;

}

}

temp->next =ptr->next ;

ptr->next=temp;

}

}

void delete_begin()

{
```

```
struct node *ptr;
if(ptr==NULL)
{
printf("\nList is Empty:\n");
return;
}
else
{
ptr=start;
start=start->next ;
printf("\nThe deleted element is :%d\t",ptr->info);
free(ptr);
}
}
void delete_end()
{
struct node *temp,*ptr;
if(start==NULL)
{
printf("\nList is Empty:");
exit(0);
}
else if(start->next ==NULL)
{
ptr=start;
```

```
start=NULL;
printf("\nThe deleted element is:%d\t",ptr->info);
free(ptr);
}
else
{
ptr=start;
while(ptr->next!=NULL)
{
temp=ptr;
ptr=ptr->next;
}
temp->next=NULL;
printf("\nThe deleted element is:%d\t",ptr->info);
free(ptr);
}
}
void delete_pos()
{
int i,pos;
struct node *temp,*ptr;
if(start==NULL)
{
printf("\nThe List is Empty:\n");
exit(0);
```

```
}
else
{
printf("\nEnter the position of the node to be deleted:\t");
scanf("%d",&pos);
if(pos==0)
{
ptr=start;
start=start->next ;
printf("\nThe deleted element is:%d\t",ptr->info );
free(ptr);
}
else
{
ptr=start;
for(i=0;i<pos;i++) { temp=ptr; ptr=ptr->next ;
if(ptr==NULL)
{
printf("\nPosition not Found:\n");
return;
}
}
temp->next =ptr->next ;
printf("\nThe deleted element is:%d\t",ptr->info );
free(ptr);
```

}

}

}

Output-

```
MENU
1.Create
2.Display
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Delete from specified position
9.Exit
Enter your choice:      1
Enter the data value for the node:      100
MENU
1.Create
2.Display
```

```
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Delete from specified position
9.Exit
Enter your choice:      1
Enter the data value for the node:      200
MENU
1.Create
2.Display
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
```

```
6.Delete from beginning
7.Delete from the end
8.Delete from specified position
9.Exit
Enter your choice:      1
Enter the data value for the node:      300
MENU
1.Create
2.Display
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Delete from specified position
```

```
6.Delete from beginning
7.Delete from the end
8.Delete from specified position
9.Exit
Enter your choice:      1
Enter the data value for the node:      300
MENU
1.Create
2.Display
3.Insert at the beginning
4.Insert at the end
5.Insert at specified position
6.Delete from beginning
7.Delete from the end
8.Delete from specified position
```

```
MENU

1.Create

2.Display

3.Insert at the beginning

4.Insert at the end

5.Insert at specified position

6.Delete from beginning

7.Delete from the end

8.Delete from specified position

9.Exit

Enter your choice:      2

The List elements are:
100t200t300t400t500t
MENU

1.Create

2.Display
```

```
3.Insert at the beginning

4.Insert at the end

5.Insert at specified position

6.Delete from beginning

7.Delete from the end

8.Delete from specified position

9.Exit

Enter your choice:      3

Enter the data value for the node:      80

MENU

1.Create

2.Display

3.Insert at the beginning

4.Insert at the end

5.Insert at specified position
```

```
6.Delete from beginning

7.Delete from the end

8.Delete from specified position

9.Exit

Enter your choice:      4

Enter the data value for the node:      600

MENU

1.Create

2.Display

3.Insert at the beginning

4.Insert at the end

5.Insert at specified position

6.Delete from beginning

7.Delete from the end

8.Delete from specified position
```

```
9.Exit

Enter your choice:      5

Enter the position for the new node to be inserted:     43

Enter the data value of the node:       250

Position not found:

MENU

1.Create

2.Display

3.Insert at the beginning

4.Insert at the end

5.Insert at specified position

6.Delete from beginning

7.Delete from the end

8.Delete from specified position

9.Exit
```

```
Enter your choice:      6

The deleted element is :80
 MENU

 1.Create

 2.Display

 3.Insert at the beginning

 4.Insert at the end

 5.Insert at specified position

 6.Delete from beginning

 7.Delete from the end

 8.Delete from specified position

 9.Exit

Enter your choice:      7

The deleted element is:600
 MENU

 1.Create
```

```
 2.Display

 3.Insert at the beginning

 4.Insert at the end

 5.Insert at specified position

 6.Delete from beginning

 7.Delete from the end

 8.Delete from specified position

 9.Exit

Enter your choice:      8

Enter the position of the node to be deleted:   4

The deleted element is:500
 MENU

 1.Create

 2.Display

 3.Insert at the beginning
```

```
 4.Insert at the end

 5.Insert at specified position

 6.Delete from beginning

 7.Delete from the end

 8.Delete from specified position

 9.Exit

Enter your choice:      9
Press any key to continue . . .
```

3.) Write a Program for polynomial representation in C.

Source Code-

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

struct Node

{

  int coeff;

  int exp;

  struct Node * next;

}* poly = NULL;

void create()

{

 struct Node * t, * last = NULL;

 int num, i;

 printf("Enter number of terms: ");

 scanf("%d", & num);

 printf("Enter each term with coeff and exp:\n");

 for (i = 0; i < num; i++)

 {

  t = (struct Node * ) malloc(sizeof(struct Node));

  scanf("%d%d", & t -> coeff, & t -> exp);

  t-> next = NULL;

  if (poly == NULL)
```

```
  {
   poly = last = t;
  }
  else
  {
   last -> next = t;
   last = t;
  }
 }
}
void Display(struct Node * p)
{
 printf("%dx%d ", p -> coeff, p -> exp);
 p = p -> next;
 while (p)
 {
  printf("+ %dx%d ", p -> coeff, p -> exp);
  p = p -> next;
 }
 printf("\n");
}
long Eval(struct Node * p, int x)
{
 long val = 0;
 while (p)
```

```
 {
  val += p -> coeff * pow(x, p -> exp);
  p = p -> next;
 }
return val;
}
int main()
{
 int x;
 create();
 Display(poly);
 printf("Enter value of x: ");
 scanf("%d", &x);
 printf("%ld\n", Eval(poly, x));
 return 0;
}
```

Output-

```
Enter number of terms: 2
Enter each term with coeff and exp:
3
1 3
2 4
3x1 + 3x2
Enter value of x: 60
```

4.) Write a program for implementation of stack using array.

Source Code-

```
#include<stdio.h>

#include<stdlib.h>

#define MAX 10

int stack_arr[MAX];

int top = -1;

void push(int item);

int pop();

int peek();

int isEmpty();

int isFull();

void display();

int main()

{

 int choice,item;

 while(1)

 {

  printf("\n1.Push\n");

  printf("2.Pop\n");

  printf("3.Display the top element\n");

  printf("4.Display all stack elements\n");

  printf("5.Quit\n");

  printf("\nEnter your choice : ");
```

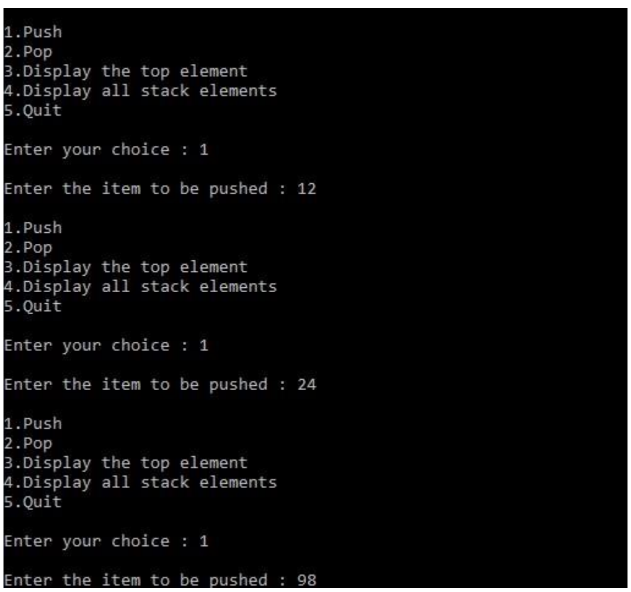
```
scanf("%d",&choice);
switch(choice)
{
 case 1 :
 printf("\nEnter the item to be pushed : ");
 scanf("%d",&item);
 push(item);
 break;
 case 2:
 item = pop();
 printf("\nPopped item is : %d\n",item );
 break;
 case 3:
 printf("\nItem at the top is : %d\n", peek() );
 break;
 case 4:
 display();
 break;
 case 5:
 exit(1);
 default:
 printf("\nWrong choice\n");
}
}
return 0;
```

```
}
void push(int item)
{
 if( isFull() )
 {
  printf("\nStack Overflow\n");
  return;
 }
 top = top+1;
 stack_arr[top] = item;
}
int pop()
{
 int item;
 if( isEmpty() )
 {
  printf("\nStack Underflow\n");
  exit(1);
 }
 item = stack_arr[top];
 top = top-1;
 return item;
}
int peek()
{
```

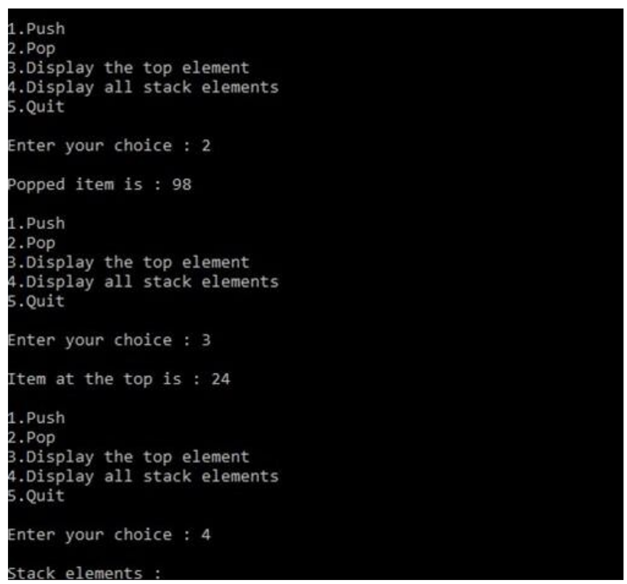
```
 if( isEmpty() )
 {
  printf("\nStack Underflow\n");
  exit(1);
 }
 return stack_arr[top];
}
int isEmpty()
{
 if( top == -1 )
 return 1;
 else
 return 0;
}
int isFull()
{
 if( top == MAX-1 )
 return 1;
 else
 return 0;
}
void display()
{
 int i;
 if( isEmpty() )
```

```
{
 printf("\nStack is empty\n");
 return;
}
printf("\nStack elements :\n\n");
for(i=top;i>=0;i--)
printf(" %d\n", stack_arr[i] );
printf("\n");
}
```

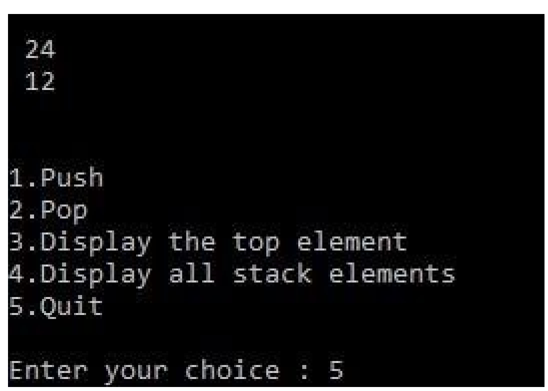
Output-



```
1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 1

Enter the item to be pushed : 12

1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 1

Enter the item to be pushed : 24

1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 1

Enter the item to be pushed : 98
```



```
1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 2

Popped item is : 98

1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 3

Item at the top is : 24

1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 4

Stack elements :
```



```
 24
 12


1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 5
```

5.) Write a program for implementation of stack using linked list.

Source Code-

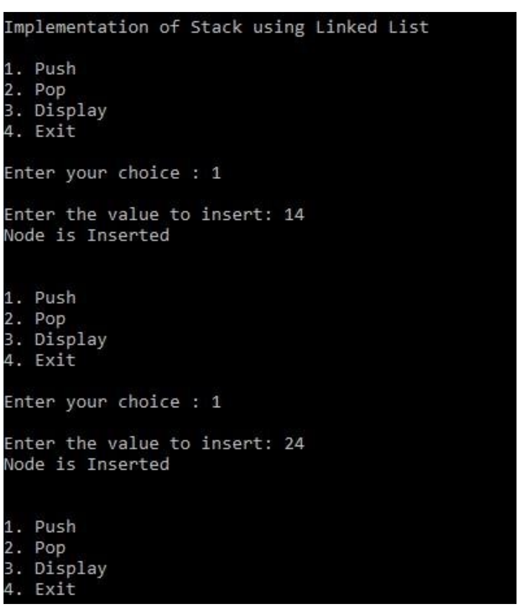
```
#include <stdio.h>
#include <stdlib.h>
struct node {
int info;
struct node *ptr;
}*top,*top1,*temp;
int count = 0;
void push(int data)
{
 if (top == NULL)
 {
  top =(struct node *)malloc(1*sizeof(struct node));
  top->ptr = NULL;
  top->info = data;
 }
 else
 {
  temp =(struct node *)malloc(1*sizeof(struct node));
  temp->ptr = top;
  temp->info = data;
  top = temp;
 }
```

```
 count++;
 printf("Node is Inserted\n\n");
}
int pop()
{
 top1 = top;
 if (top1 == NULL)
 {
  printf("\nStack Underflow\n");
  return -1;
 }
 else
  top1 = top1->ptr;
 int popped = top->info;
 free(top);
 top = top1;
 count--;
 return popped;
}
void display()
{
 top1 = top;
 if (top1 == NULL)
 {
  printf("\nStack Underflow\n");
```

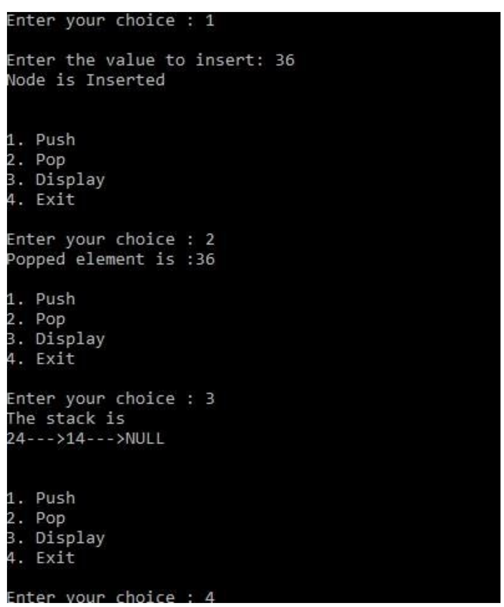
```
 return;
 }
 printf("The stack is \n");
 while (top1 != NULL)
 {
  printf("%d--->", top1->info);
  top1 = top1->ptr;
 }
 printf("NULL\n\n");
}
int main()
{
 int choice, value;
 printf("\nImplementation of Stack using Linked List\n");
 while (1)
 {
  printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
  printf("\nEnter your choice : ");
  scanf("%d", &choice);
  switch (choice)
  {
   case 1:
   printf("\nEnter the value to insert: ");
   scanf("%d", &value);
   push(value);
```

```
  break;

  case 2:

  printf("Popped element is :%d\n", pop());

  break;

  case 3:

  display();

  break;

  case 4:

  exit(0);

  break;

  default:

  printf("\nWrong Choice\n");

 }

}

}
```

Output-



```
Implementation of Stack using Linked List

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 14
Node is Inserted


1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 24
Node is Inserted


1. Push
2. Pop
3. Display
4. Exit
```



```
Enter your choice : 1

Enter the value to insert: 36
Node is Inserted


1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 2
Popped element is :36

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 3
The stack is
24--->14--->NULL


1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 4
```

6.) Write a program for implementation of queue using array.

Source Code-

```
#include <stdio.h>

#define SIZE 5

void enQueue(int);

void deQueue();

void display();

int items[SIZE], front = -1, rear = -1;

int main()

{

 deQueue();

 enQueue(1);

 enQueue(2);

 enQueue(3);

 enQueue(4);

 enQueue(5);

 enQueue(6);

 display();

 deQueue();

 display();

 return 0;

}

void enQueue(int value)

{
```

```
 if (rear == SIZE - 1)
 printf("\nQueue is Full!!");
 else
 {
  if (front == -1)
  front = 0;
  rear++;
  items[rear] = value;
  printf("\nInserted -> %d", value);
 }
}
void deQueue()
{
 if (front == -1)
 printf("\nQueue is Empty!!");
 else
 {
  printf("\nDeleted : %d", items[front]);
  front++;
  if (front > rear)
  front = rear = -1;
 }
}
void display()
{
```

```
if (rear == -1)
printf("\nQueue is Empty!!!");
else
{
 int i;
 printf("\nQueue elements are:\n");
 for (i = front; i <= rear; i++)
 printf("%d ", items[i]);
}
printf("\n");
}
```

Output-

```
Queue is Empty!!
Inserted -> 1
Inserted -> 2
Inserted -> 3
Inserted -> 4
Inserted -> 5
Queue is Full!!
Queue elements are:
1 2 3 4 5

Deleted : 1
Queue elements are:
2 3 4 5
```

7.) Write a program for implementation of queue using linked list.

Source Code-

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *next;
};
struct node *front;
struct node *rear;
void insert();
void delete();
void display();
void main ()
{
 int choice;
 while(choice != 4)
 {
  printf("\nMain Menu\n");
  printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");
  printf("\nEnter your choice...");
  scanf("%d",& choice);
  switch(choice)
```

```
  {
   case 1:
   insert();
   break;
   case 2:
   delete();
   break;
   case 3:
   display();
   break;
   case 4:
   exit(0);
   break;
   default:
   printf("\nEnter valid choice??\n");
  }
 }
}
void insert()
{
 struct node *ptr;
 int item;
 ptr = (struct node *) malloc (sizeof(struct node));
 if(ptr == NULL)
```

```
{
 printf("\nOVERFLOW\n");
 return;
}
else
{
 printf("\nEnter the value....\n");
 scanf("%d",&item);
 ptr -> data = item;
 if(front == NULL)
 {
  front = ptr;
  rear = ptr;
  front -> next = NULL;
  rear -> next = NULL;
 }
 else
 {
  rear -> next = ptr;
  rear = ptr;
  rear->next = NULL;
 }
}
}
```

```
void delete ()
{
 struct node *ptr;
 if(front == NULL)
 {
  printf("\nUNDERFLOW\n");
  return;
 }
 else
 {
  ptr = front;
  front = front -> next;
  free(ptr);
 }
}
void display()
{
 struct node *ptr;
 ptr = front;
 if(front == NULL)
 {
  printf("\nEmpty queue\n");
 }
 else
```

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

Output-

```
Main Menu

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice...1

Enter the value....
12

Main Menu

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice...1

Enter the value....
25

Main Menu
```

```
1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice...1

Enter the value....
38

Main Menu

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice...1

Enter the value....
45

Main Menu
```

```
1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice...2

Main Menu

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice...3

printing values .....

25

38

45
```

```
Main Menu

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice...4
```

8.) Write a program for implementation of circular queue using array.

Source Code-

```
#include<stdio.h>
#define capacity 6
int queue[capacity];
int front = -1, rear = -1;
int checkFull ()
{
 if ((front == rear + 1) || (front == 0 && rear == capacity - 1))
 {
  return 1;
 }
 return 0;
}
int checkEmpty ()
{
 if (front == -1)
 {
  return 1;
 }
 return 0;
}
void enqueue (int value)
```

```
{
 if (checkFull ())
 printf ("Overflow condition\n");
 else
 {
  if (front == -1)
  front = 0;
  rear = (rear + 1) % capacity;
  queue[rear] = value;
  printf ("%d was enqueued to circular queue\n", value);
 }
}
int dequeue ()
{
 int variable;
 if (checkEmpty ())
 {
  printf ("Underflow condition\n");
  return -1;
 }
 else
 {
  variable = queue[front];
  if (front == rear)
  {
```

```
  front = rear = -1;
 }
 else
 {
  front = (front + 1) % capacity;
 }
 printf ("%d was dequeued from circular queue\n", variable);
 return 1;
}
}
void print ()
{
int i;
if (checkEmpty ())
 printf ("Nothing to dequeue\n");
else
{
 printf ("\nThe queue looks like: \n");
 for (i = front; i != rear; i = (i + 1) % capacity)
 {
  printf ("%d ", queue[i]);
 }
 printf ("%d \n\n", queue[i]);
}
}
```

```
int main ()

{

  dequeue ();

  enqueue (15);

  enqueue (20);

  enqueue (25);

  enqueue (30);

  enqueue (35);

  print ();

  dequeue ();

  dequeue ();

  print ();

  enqueue (40);

  enqueue (45);

  enqueue (50);

  enqueue (55);

  print ();

  return 0;

}
```

Output-

```
Underflow condition
15 was enqueued to circular queue
20 was enqueued to circular queue
25 was enqueued to circular queue
30 was enqueued to circular queue
35 was enqueued to circular queue

The queue looks like:
15 20 25 30 35

15 was dequeued from circular queue
20 was dequeued from circular queue
```

```
The queue looks like:
25 30 35

40 was enqueued to circular queue
45 was enqueued to circular queue
50 was enqueued to circular queue
Overflow condition

The queue looks like:
25 30 35 40 45 50
```

9.) Write a program for implementation of circular queue using linked list.

Source Code-

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

 int data;

 struct node *next;

};

struct node *f = NULL;

struct node *r = NULL;

void enqueue (int d)

{

 struct node *n;

 n = (struct node *) malloc (sizeof (struct node));

 n->data = d;

 n->next = NULL;

 if ((r == NULL) && (f == NULL))

 {

  f = r = n;

  r->next = f;

 }

 else
```

```
 {
  r->next = n;
  r = n;
  n->next = f;
 }
}
void dequeue ()
{
 struct node *t;
 t = f;
 if ((f == NULL) && (r == NULL))
 printf ("\nQueue is Empty");
 else if (f == r)
 {
  f = r = NULL;
  free (t);
 }
 else
 {
  f = f->next;
  r->next = f;
  free (t);
 }
}
```

```
void display ()
{
 struct node *t;
 t = f;
 if ((f == NULL) && (r == NULL))
  printf ("\nQueue is Empty");
 else
 {
  do
  {
   printf (" %d", t->data);
   t = t->next;
  }
  while (t != f);
 }
}
int main ()
{
 enqueue (34);
 enqueue (22);
 enqueue (75);
 enqueue (99);
 enqueue (27);
 printf ("Circular Queue: ");
```

```
display ();
printf ("\n");
dequeue ();
printf ("Circular Queue After dequeue: ");
display ();
return 0;
}
```

Output-

```
Circular Queue:  34 22 75 99 27
Circular Queue After dequeue:  22 75 99 27
```

10.) Write a program for implementation of creation of binary tree.

Source Code-

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

    int data;

    struct node *leftChild, *rightChild;

};

struct node *root=NULL;

struct node *newNode(int item)

{

    struct node *temp=(struct node*)malloc(sizeof(struct node));

    temp->data=item;

    temp->leftChild=temp->rightChild=NULL;

    return temp;

}

void insert(int data)

{

    struct node *tempNode=(struct node*)malloc(sizeof(struct node));

    struct node *current;

    struct node *parent;

    tempNode->data=data;

    tempNode->leftChild=NULL;
```

```
tempNode->rightChild=NULL;
if(root==NULL)
{
    root=tempNode;
}
else
{
    current=root;
    parent=NULL;
    while(1)
    {
        parent=current;
        if(data<parent->data)
        {
            current=current->leftChild;
            if(current==NULL)
            {
                parent->leftChild=tempNode;
                return;
            }
        }
        else
        {
            current=current->rightChild;
```

```
            if(current==NULL)
            {
                parent->rightChild=tempNode;
                return;
            }
        }
    }
  }
}
struct node* search(int data)
{
    struct node *current=root;
    printf("\nVisiting elements:");
    while(current->data !=data)
    {
        if(current !=NULL)
        {
            printf("%d",current->data);
            if(current->data>data)
            {
                current=current->leftChild;
            }
            else
            {
```

```
                current=current->rightChild;
            }
            if(current==NULL)
            {
                return NULL;
            }
        }
    }
    return current;
}
void printTree(struct node*Node)
{
    if(Node==NULL)
     return;
    printTree(Node->leftChild);
    printf("--%d",Node->data);
    printTree(Node->rightChild);
}
int main()
{
    insert(55);
    insert(20);
    insert(90);
    insert(50);
```

```
    insert(35);

    insert(15);

    insert(65);

    printf("Insertion done\n");

    printTree(root);

    struct node* k;

    k=search(35);

    if(k !=NULL)

     printf("\nElement %d found",k->data);

    else

     printf("\nElement not found");

    return 0;

}
```

Output-

```
Insertion done
--15--20--35--50--55--65--90
Visiting elements:552050
Element 35 foundPress any key to continue . . .
```

11.) Write a program for implementation of tree traversal in binary tree.

Source Code-

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

   int data;

   struct node *leftChild;

   struct node *rightChild;

};

struct node *root=NULL;

void insert(int data)

{

   struct node *tempNode=(struct node*)malloc(sizeof(struct node));

   struct node *current;

   struct node *parent;

   tempNode->data=data;

   tempNode->leftChild=NULL;

   tempNode->rightChild=NULL;

   if(root==NULL)

   {

      root=tempNode;

   }
```

```
else
{
    current=root;
    parent=NULL;
    while(1)
    {
        parent=current;
        if(data<parent->data)
        {
            current=current->leftChild;
            if(current==NULL)
            {
                parent->leftChild=tempNode;
                return;
            }
        }
        else
        {
            current=current->rightChild;
            if(current==NULL)
            {
                parent->rightChild=tempNode;
                return;
            }
        }
```

```
        }
    }
}
struct node* search(int data)
{
    struct node *current=root;
    printf("Visiting elements:");
    while(current->data !=data)
    {
        if(current !=NULL)
         printf("%d",current->data);
        if(current->data>data)
        {
            current=current->leftChild;
        }
        else
        {
            current=current->rightChild;
        }
        if(current==NULL)
        {
            return NULL;
        }
    }
    return current;
```

```
}
void pre_order_traversal(struct node* root)
{
   if(root !=NULL)
   {
      printf("%d",root->data);
      pre_order_traversal(root->leftChild);
      pre_order_traversal(root->rightChild);
   }
}
void inorder_traversal(struct node* root)
{
   if(root !=NULL)
   {
      inorder_traversal(root->leftChild);
      printf("%d",root->data);
      inorder_traversal(root->rightChild);
   }
}
void post_order_traversal(struct node* root)
{
   if(root !=NULL)
   {
      post_order_traversal(root->leftChild);
      post_order_traversal(root->rightChild);
```

```
        printf("%d",root->data);
    }
}
int main()
{
    int i;
    int array[7]={27,14,35,10,19,31,42};
    for(i=0;i<7;i++)
     insert(array[i]);
    i=31;
    struct node *temp=search(i);
    if(temp !=NULL)
    {
        printf("\n[%d]Element found",temp->data);
        printf("\n");
    }
    else
    {
        printf("\n[%d]Element not found\n",i);
    }
    i=11;
    temp=search(i);
    if(temp !=NULL)
    {
        printf("\n[%d]Element found",temp->data);
```

```
        printf("\n");
    }
    else
    {
        printf("\n[%d]Element not found \n",i);
    }
    printf("\nPreorder traversal:");
    pre_order_traversal(root);
    printf("\nInorder traversal:");
    inorder_traversal(root);
    printf("\nPost order traversal:");
    post_order_traversal(root);
    return 0;
}
```

Output-

```
Visiting elements:2735
[31]Element found
Visiting elements:271410
[11]Element not found

Preorder traversal:27141019353142
Inorder traversal:10141927313542
Post order traversal:10191431423527
```

12.) Write a program for implementation of insertion and deletion operation in binary search tree.

Source Code-

```
#include <stdio.h>

#include <stdlib.h>

struct node

{

 int key;

 struct node *left, *right;

};

struct node *newNode(int item)

{

 struct node *temp = (struct node *)malloc(sizeof(struct node));

 temp->key = item;

 temp->left = temp->right = NULL;

 return temp;

}

void inorder(struct node *root)

{

 if (root != NULL)

 {

  inorder(root->left);

  printf("%d -> ", root->key);

  inorder(root->right);
```

```
 }
}
struct node *insert(struct node *node, int key)
{
 if (node == NULL) return newNode(key);
 if (key < node->key)
  node->left = insert(node->left, key);
 else
  node->right = insert(node->right, key);
 return node;
}
struct node *minValueNode(struct node *node)
{
 struct node *current = node;
 while (current && current->left != NULL)
 current = current->left;
 return current;
}
struct node *deleteNode(struct node *root, int key)
{
 if (root == NULL) return root;
 if (key < root->key)
  root->left = deleteNode(root->left, key);
 else if (key > root->key)
```

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

```
root = insert(root, 8);

root = insert(root, 3);

root = insert(root, 1);

root = insert(root, 6);

root = insert(root, 7);

root = insert(root, 10);

root = insert(root, 14);

root = insert(root, 4);

printf("Inorder traversal: ");

inorder(root);

printf("\nAfter deleting 10\n");

root = deleteNode(root, 10);

printf("Inorder traversal: ");

inorder(root);

}
```

Output-

```
Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 10 -> 14 ->
After deleting 10
Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 14 -> Press any key to continue . . .
```

13.) Write a Program for implementation of binary heap in C.

Source Code-

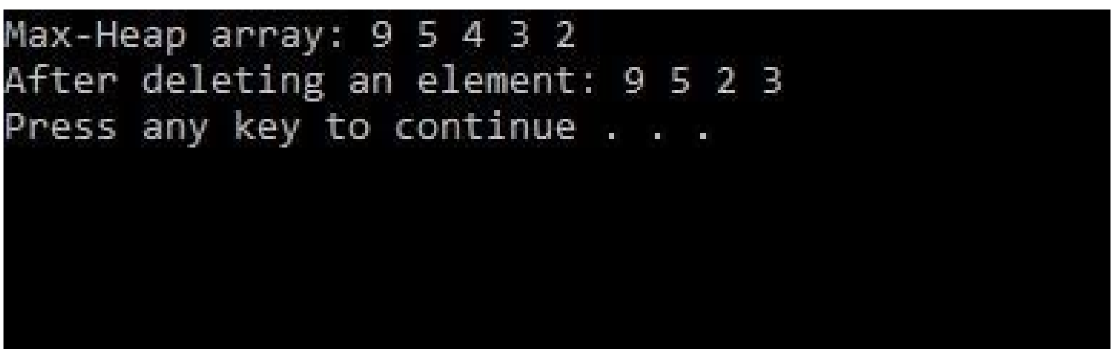
```
#include <stdio.h>
int size = 0;
void swap(int *a, int *b)
{
 int temp = *b;
 *b = *a;
 *a = temp;
}
void heapify(int array[], int size, int i)
{
 if (size == 1)
 {
  printf("Single element in the heap");
 }
 else
 {
  int largest = i;
  int l = 2 * i + 1;
  int r = 2 * i + 2;
  if (l < size && array[l] > array[largest])
  largest = l;
  if (r < size && array[r] > array[largest])
```

```
  largest = r;
  if (largest != i)
  {
   swap(&array[i], &array[largest]);
   heapify(array, size, largest);
  }
 }
}
void insert(int array[], int newNum)
{
 if (size == 0)
 {
  array[0] = newNum;
  size += 1;
 }
 else
 {
  array[size] = newNum;
  size += 1;
  for (int i = size / 2 - 1; i >= 0; i--)
  {
   heapify(array, size, i);
  }
 }
```

```
}
void deleteRoot(int array[], int num)
{
 int i;
 for (i = 0; i < size; i++)
 {
  if (num == array[i])
  break;
 }
 swap(&array[i], &array[size - 1]);
 size -= 1;
 for (int i = size / 2 - 1; i >= 0; i--)
 {
  heapify(array, size, i);
 }
}
void printArray(int array[], int size)
{
 for (int i = 0; i < size; ++i)
 printf("%d ", array[i]);
 printf("\n");
}
int main()
{
```

```
int array[10];

insert(array, 3);

insert(array, 4);

insert(array, 9);

insert(array, 5);

insert(array, 2);

printf("Max-Heap array: ");

printArray(array, size);

deleteRoot(array, 4);

printf("After deleting an element: ");

printArray(array, size);

}
```

Output-



14.) Write a Program for implementation of addition and deletion in Btree operation.

Source Code-

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 3

#define MIN 2

struct BTreeNode

{

 int item[MAX + 1], count;

 struct BTreeNode *linker[MAX + 1];

};

struct BTreeNode *root;

struct BTreeNode *createNode(int item, struct BTreeNode *child)

{

 struct BTreeNode *newNode;

 newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));

 newNode->item[1] = item;

 newNode->count = 1;

 newNode->linker[0] = root;

 newNode->linker[1] = child;

 return newNode;

}

void addValToNode(int item, int pos, struct BTreeNode *node,
```

```
struct BTreeNode *child)
{
 int j = node->count;
 while (j > pos)
 {
  node->item[j + 1] = node->item[j];
  node->linker[j + 1] = node->linker[j];
  j--;
 }
 node->item[j + 1] = item;
 node->linker[j + 1] = child;
 node->count++;
}
void splitNode(int item, int *pval, int pos, struct BTreeNode *node,
struct BTreeNode *child, struct BTreeNode **newNode)
{
 int median, j;
 if (pos > MIN)
  median = MIN + 1;
 else
  median = MIN;
 *newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
 j = median + 1;
 while (j <= MAX)
```

```
{
 (*newNode)->item[j - median] = node->item[j];
 (*newNode)->linker[j - median] = node->linker[j];
 j++;
}
node->count = median;
(*newNode)->count = MAX - median;
if (pos <= MIN)
{
 addValToNode(item, pos, node, child);
}
else
{
 addValToNode(item, pos - median, *newNode, child);
}
*pval = node->item[node->count];
(*newNode)->linker[0] = node->linker[node->count];
node->count--;
}
int setValueInNode(int item, int *pval,
struct BTreeNode *node, struct BTreeNode **child)
{
 int pos;
 if (!node)
```

```
{
 *pval = item;
 *child = NULL;
 return 1;
}
if (item < node->item[1])
{
 pos = 0;
}
else
{
 for (pos = node->count;
  (item < node->item[pos] && pos > 1); pos--);
  if (item == node->item[pos])
  {
   printf("Duplicates not allowed\n");
   return 0;
  }
}
if (setValueInNode(item, pval, node->linker[pos], child))
{
 if (node->count < MAX)
 {
  addValToNode(*pval, pos, node, *child);
```

```
  }
  else
  {
   splitNode(*pval, pval, pos, node, *child, child);
   return 1;
  }
 }
 return 0;
}
void insert(int item)
{
 int flag, i;
 struct BTreeNode *child;
 flag = setValueInNode(item, &i, root, &child);
 if (flag)
  root = createNode(i, child);
}
void copySuccessor(struct BTreeNode *myNode, int pos)
{
 struct BTreeNode *dummy;
 dummy = myNode->linker[pos];
 for (; dummy->linker[0] != NULL;)
  dummy = dummy->linker[0];
  myNode->item[pos] = dummy->item[1];
```

```
}
void removeVal(struct BTreeNode *myNode, int pos)
{
 int i = pos + 1;
 while (i <= myNode->count)
 {
  myNode->item[i - 1] = myNode->item[i];
  myNode->linker[i - 1] = myNode->linker[i];
  i++;
 }
 myNode->count--;
}
void rightShift(struct BTreeNode *myNode, int pos)
{
 struct BTreeNode *x = myNode->linker[pos];
 int j = x->count;
 while (j > 0)
 {
  x->item[j + 1] = x->item[j];
  x->linker[j + 1] = x->linker[j];
 }
 x->item[1] = myNode->item[pos];
 x->linker[1] = x->linker[0];
 x->count++;
```

```
 x = myNode->linker[pos - 1];
 myNode->item[pos] = x->item[x->count];
 myNode->linker[pos] = x->linker[x->count];
 x->count--;
 return;
}
void leftShift(struct BTreeNode *myNode, int pos)
{
 int j = 1;
 struct BTreeNode *x = myNode->linker[pos - 1];
 x->count++;
 x->item[x->count] = myNode->item[pos];
 x->linker[x->count] = myNode->linker[pos]->linker[0];
 x = myNode->linker[pos];
 myNode->item[pos] = x->item[1];
 x->linker[0] = x->linker[1];
 x->count--;
 while (j <= x->count)
 {
  x->item[j] = x->item[j + 1];
  x->linker[j] = x->linker[j + 1];
  j++;
 }
 return;
```

```
}
void mergeNodes(struct BTreeNode *myNode, int pos)
{
 int j = 1;
 struct BTreeNode *x1 = myNode->linker[pos], *x2 = myNode->linker[pos - 1];
 x2->count++;
 x2->item[x2->count] = myNode->item[pos];
 x2->linker[x2->count] = myNode->linker[0];
 while (j <= x1->count)
 {
  x2->count++;
  x2->item[x2->count] = x1->item[j];
  x2->linker[x2->count] = x1->linker[j];
  j++;
  j = pos;
  while (j < myNode->count)
  {
   myNode->item[j] = myNode->item[j + 1];
   myNode->linker[j] = myNode->linker[j + 1];
   j++;
  }
  myNode->count--;
  free(x1);
 }
```

```
}
void adjustNode(struct BTreeNode *myNode, int pos)
{
 if (!pos)
 {
  if (myNode->linker[1]->count > MIN)
  {
   leftShift(myNode, 1);
  }
  else
  {
   mergeNodes(myNode, 1);
  }
 }
 else
 {
  if (myNode->count != pos)
  {
   if (myNode->linker[pos - 1]->count > MIN)
   {
    rightShift(myNode, pos);
   }
   else
   {
```

```
    if (myNode->linker[pos + 1]->count > MIN)
    {
     leftShift(myNode, pos + 1);
    }
    else
    {
     mergeNodes(myNode, pos);
    }
   }
  }
  else
  {
   if (myNode->linker[pos - 1]->count > MIN)
    rightShift(myNode, pos);
   else
    mergeNodes(myNode, pos);
  }
 }
}
int delValFromNode(int item, struct BTreeNode *myNode)
{
 int pos, flag = 0;
 if (myNode)
 {
```

```
if (item < myNode->item[1])
{
 pos = 0;
 flag = 0;
}
else
{
 for (pos = myNode->count; (item < myNode->item[pos] && pos > 1); pos--);
  if (item == myNode->item[pos])
  {
   flag = 1;
  }
  else
  {
   flag = 0;
  }
 }
 if (flag)
 {
  if (myNode->linker[pos - 1])
  {
   copySuccessor(myNode, pos);
   flag = delValFromNode(myNode->item[pos], myNode->linker[pos]);
   if (flag == 0)
```

```
     {
      printf("Given data is not present in B-Tree\n");
     }
    }
    else
    {
     removeVal(myNode, pos);
    }
   }
   else
   {
    flag = delValFromNode(item, myNode->linker[pos]);
   }
   if (myNode->linker[pos])
   {
    if (myNode->linker[pos]->count < MIN)
     adjustNode(myNode, pos);
   }
  }
 return flag;
}
void delete (int item, struct BTreeNode *myNode)
{
 struct BTreeNode *tmp;
```

```
 if (!delValFromNode(item, myNode))
 {
  printf("Not present\n");
  return;
 }
 else
 {
  if (myNode->count == 0)
  {
   tmp = myNode;
   myNode = myNode->linker[0];
   free(tmp);
  }
 }
 root = myNode;
 return;
}
void display(struct BTreeNode *myNode)
{
 int i;
 if (myNode)
 {
  for (i = 0; i < myNode->count; i++)
  {
```

```
   display(myNode->linker[i]);
   printf("%d ", myNode->item[i + 1]);
  }
  display(myNode->linker[i]);
 }
}
int main()
{
 int item, ch;
 insert(8);
 insert(9);
 insert(10);
 insert(11);
 insert(15);
 insert(16);
 insert(17);
 insert(18);
 insert(20);
 insert(23);
 printf("Insertion Done");
 printf("\nBTree elements before deletion: \n");
 display(root);
 int ele = 20;
 printf("\nThe element to be deleted: %d", ele);
```

```
 delete (ele, root);

 printf("\nBTree elements after deletion: \n");

 display(root);

}
```

Output-

```
Insertion Done
BTree elements before deletion:
8 9 10 11 15 16 17 18 20 23
The element to be deleted: 20
BTree elements after deletion:
8 9 10 11 15 16 17 18 23 8 9 23
```

15.) Write a Program for graph implementation in C.

Source Code-

```
#include <stdio.h>

#define V 4

void init(int arr[][V])

{

 int i, j;

 for (i = 0; i < V; i++)

 for (j = 0; j < V; j++)

 arr[i][j] = 0;

}

void insertEdge(int arr[][V], int i, int j)

{

 arr[i][j] = 1;

 arr[j][i] = 1;

}

void printAdjMatrix(int arr[][V])

{

 int i, j;

 for (i = 0; i < V; i++)

 {

  printf("%d: ", i);

  for (j = 0; j < V; j++)

  {
```

```
   printf("%d ", arr[i][j]);
  }
  printf("\n");
 }
}
int main()
{
 int adjMatrix[V][V];
 init(adjMatrix);
 insertEdge(adjMatrix, 0, 1);
 insertEdge(adjMatrix, 0, 2);
 insertEdge(adjMatrix, 1, 2);
 insertEdge(adjMatrix, 2, 0);
 insertEdge(adjMatrix, 2, 3);
 printAdjMatrix(adjMatrix);
 return 0;
}
```

Output-

```
0: 0 1 1 0
1: 1 0 1 0
2: 1 1 0 1
3: 0 0 1 0
Press any key to continue . . .
```

16.) Write a Program for BFS algorithm implementation in C.

Source Code-

```
#include <stdio.h>

#include <stdlib.h>

#define SIZE 40

struct queue

{

 int items[SIZE];

 int front;

 int rear;

};

struct queue* createQueue();

void enqueue(struct queue* q, int);

int dequeue(struct queue* q);

void display(struct queue* q);

int isEmpty(struct queue* q);

void printQueue(struct queue* q);

struct node

{

 int vertex;

 struct node* next;

};

struct node* createNode(int);

struct Graph
```

```
{
 int numVertices;
 struct node** adjLists;
 int* visited;
};
void bfs(struct Graph* graph, int startVertex)
{
 struct queue* q = createQueue();
 graph->visited[startVertex] = 1;
 enqueue(q, startVertex);
 while (!isEmpty(q))
 {
  printQueue(q);
  int currentVertex = dequeue(q);
  printf("\nVisited %d\n", currentVertex);
  struct node* temp = graph->adjLists[currentVertex];
  while (temp)
  {
   int adjVertex = temp->vertex;
   if (graph->visited[adjVertex] == 0)
   {
    graph->visited[adjVertex] = 1;
    enqueue(q, adjVertex);
   }
```

```
   temp = temp->next;
  }
 }
}
struct node* createNode(int v)
{
 struct node* newNode = malloc(sizeof(struct node));
 newNode->vertex = v;
 newNode->next = NULL;
 return newNode;
}
struct Graph* createGraph(int vertices)
{
 struct Graph* graph = malloc(sizeof(struct Graph));
 graph->numVertices = vertices;
 graph->adjLists = malloc(vertices * sizeof(struct node*));
 graph->visited = malloc(vertices * sizeof(int));
 int i;
 for (i = 0; i < vertices; i++)
 {
  graph->adjLists[i] = NULL;
  graph->visited[i] = 0;
 }
 return graph;
```

```
}
void addEdge(struct Graph* graph, int src, int dest)
{
 struct node* newNode = createNode(dest);
 newNode->next = graph->adjLists[src];
 graph->adjLists[src] = newNode;
 newNode = createNode(src);
 newNode->next = graph->adjLists[dest];
 graph->adjLists[dest] = newNode;
}
struct queue* createQueue()
{
 struct queue* q = malloc(sizeof(struct queue));
 q->front = -1;
 q->rear = -1;
 return q;
}
int isEmpty(struct queue* q)
{
 if (q->rear == -1)
 return 1;
 else
 return 0;
}
```

```
void enqueue(struct queue* q, int value)
{
 if (q->rear == SIZE - 1)
 printf("\nQueue is Full!!");
 else
 {
  if (q->front == -1)
  q->front = 0;
  q->rear++;
  q->items[q->rear] = value;
 }
}
int dequeue(struct queue* q)
{
 int item;
 if (isEmpty(q))
 {
  printf("Queue is empty");
  item = -1;
 }
 else
 {
  item = q->items[q->front];
  q->front++;
```

```
 if (q->front > q->rear)
 {
  printf("\nResetting queue ");
  q->front = q->rear = -1;
 }
}
return item;
}
void printQueue(struct queue* q)
{
 int i = q->front;
 if (isEmpty(q))
 {
  printf("Queue is empty");
 }
 else
 {
  printf("\nQueue contains \n");
  for (i = q->front; i < q->rear + 1; i++)
  {
   printf("%d ", q->items[i]);
  }
 }
}
```

```
int main()

{

 struct Graph* graph = createGraph(6);

 addEdge(graph, 0, 1);

 addEdge(graph, 0, 2);

 addEdge(graph, 1, 2);

 addEdge(graph, 1, 4);

 addEdge(graph, 1, 3);

 addEdge(graph, 2, 4);

 addEdge(graph, 3, 4);

 bfs(graph, 0);

 return 0;

}
```

Output-

```
Queue contains
0
Resetting queue
Visited 0

Queue contains
2 1
Visited 2

Queue contains
1 4
Visited 1

Queue contains
4 3
Visited 4

Queue contains
3
Resetting queue
Visited 3
```

17.) Write a Program for prim's algorithm implementation in C.

Source Code-

```
#include <stdio.h>

#include <limits.h>

#define vertices 5

int minimum_key(int k[], int mst[])

{

 int minimum = INT_MAX, min,i;

 for (i = 0; i < vertices; i++)

 if (mst[i] == 0 && k[i] < minimum )

  minimum = k[i], min = i;

 return min;

}

void prim(int g[vertices][vertices])

{

 int parent[vertices];

 int min_cost=0;

 int k[vertices];

 int mst[vertices];

 int i, count,edge,v;

 for (i = 0; i < vertices; i++)

 {

  k[i] = INT_MAX;

  mst[i] = 0;
```

```
 }
 k[0] = 0;
 parent[0] = -1;
 for (count = 0; count < vertices-1; count++)
 {
  edge = minimum_key(k, mst);
  mst[edge] = 1;
  for (v = 0; v < vertices; v++)
  {
   if (g[edge][v] && mst[v] == 0 && g[edge][v] < k[v])
   {
    parent[v] = edge, k[v] = g[edge][v];
   }
  }
 }
 printf("\n Edge \t Weight\n");
 for (i = 1; i < vertices; i++)
 {
  printf(" %d <-> %d %d \n", parent[i], i, g[i][parent[i]]);
  min_cost=min_cost+i;
  }
  printf("The cost of spanning tree:%d",min_cost);
}
int main()
```

```
{
 int g[vertices][vertices] = {{0, 0, 3, 0, 0},
                              {0, 0, 10, 4, 0},
                              {3, 10, 0, 2, 6},
                              {0, 4, 2, 0, 1},
                              {0, 0, 6, 1, 0},
                              };
 prim(g);
 return 0;
}
```

Output-

```
Edge    Weight
3 <-> 1 4
0 <-> 2 3
2 <-> 3 2
3 <-> 4 1
The cost of spanning tree:10Press any key to continue . . .
```

18.) Write a Program for kruskal's algorithm implementation in C.

Source Code-

```
#include <stdio.h>

#define MAX 30

typedef struct edge

{

 int u, v, w;

}

edge;

typedef struct edge_list

{

 edge data[MAX];

 int n;

}

edge_list;

edge_list elist;

int Graph[MAX][MAX], n;

edge_list spanlist;

void kruskalAlgo();

int find(int belongs[], int vertexno);

void applyUnion(int belongs[], int c1, int c2);

void sort();

void print();

void kruskalAlgo()
```

```
{
 int belongs[MAX], i, j, cno1, cno2;
 elist.n = 0;
 for (i = 1; i < n; i++)
 for (j = 0; j < i; j++)
 {
  if (Graph[i][j] != 0)
  {
   elist.data[elist.n].u = i;
   elist.data[elist.n].v = j;
   elist.data[elist.n].w = Graph[i][j];
   elist.n++;
  }
 }
 sort();
 for (i = 0; i < n; i++)
 belongs[i] = i;
 spanlist.n = 0;
 for (i = 0; i < elist.n; i++)
 {
  cno1 = find(belongs, elist.data[i].u);
  cno2 = find(belongs, elist.data[i].v);
  if (cno1 != cno2)
  {
```

```
   spanlist.data[spanlist.n] = elist.data[i];
   spanlist.n = spanlist.n + 1;
   applyUnion(belongs, cno1, cno2);
  }
 }
}
int find(int belongs[], int vertexno)
{
 return (belongs[vertexno]);
}
void applyUnion(int belongs[], int c1, int c2)
{
 int i;
 for (i = 0; i < n; i++)
 if (belongs[i] == c2)
 belongs[i] = c1;
}
void sort()
{
 int i, j;
 edge temp;
 for (i = 1; i < elist.n; i++)
 for (j = 0; j < elist.n - 1; j++)
 if (elist.data[j].w > elist.data[j + 1].w)
```

```
 {
  temp = elist.data[j];
  elist.data[j] = elist.data[j + 1];
  elist.data[j + 1] = temp;
 }
}
void print()
{
 int i, cost = 0;
 for (i = 0; i < spanlist.n; i++)
 {
  printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
  cost = cost + spanlist.data[i].w;
 }
 printf("\nSpanning tree cost: %d", cost);
}
int main()
{
 int i, j, total_cost;
 n = 6;
 Graph[0][0] = 0;
 Graph[0][1] = 4;
 Graph[0][2] = 4;
 Graph[0][3] = 0;
```

```
Graph[0][4] = 0;
Graph[0][5] = 0;
Graph[0][6] = 0;
Graph[1][0] = 4;
Graph[1][1] = 0;
Graph[1][2] = 2;
Graph[1][3] = 0;
Graph[1][4] = 0;
Graph[1][5] = 0;
Graph[1][6] = 0;
Graph[2][0] = 4;
Graph[2][1] = 2;
Graph[2][2] = 0;
Graph[2][3] = 3;
Graph[2][4] = 4;
Graph[2][5] = 0;
Graph[2][6] = 0;
Graph[3][0] = 0;
Graph[3][1] = 0;
Graph[3][2] = 3;
Graph[3][3] = 0;
Graph[3][4] = 3;
Graph[3][5] = 0;
Graph[3][6] = 0;
```

```
Graph[4][0] = 0;

Graph[4][1] = 0;

Graph[4][2] = 4;

Graph[4][3] = 3;

Graph[4][4] = 0;

Graph[4][5] = 0;

Graph[4][6] = 0;

Graph[5][0] = 0;

Graph[5][1] = 0;

Graph[5][2] = 2;

Graph[5][3] = 0;

Graph[5][4] = 3;

Graph[5][5] = 0;

Graph[5][6] = 0;

kruskalAlgo();

print();

}
```

Output-

```
2 - 1 : 2
5 - 2 : 2
3 - 2 : 3
4 - 3 : 3
1 - 0 : 4
Spanning tree cost: 14Press any key to continue . . .
```

19.) Write a Program for Dijkstra algorithm implementation in C.

Source Code-

```
#include <stdio.h>

#define INF 9999

#define MAX 10

void DijkstraAlgorithm(int Graph[MAX][MAX], int size, int start);

void DijkstraAlgorithm(int Graph[MAX][MAX], int size, int start)

{

 int cost[MAX][MAX], distance[MAX], previous[MAX];

 int visited_nodes[MAX], counter, minimum_distance, next_node, i, j;

 for (i = 0; i < size; i++)

  for (j = 0; j < size; j++)

   if (Graph[i][j] == 0)

    cost[i][j] = INF;

   else

    cost[i][j] = Graph[i][j];

  for (i = 0; i < size; i++)

  {

   distance[i] = cost[start][i];

   previous[i] = start;

   visited_nodes[i] = 0;

  }

  distance[start] = 0;

  visited_nodes[start] = 1;
```

```
counter = 1;
while (counter < size - 1)
{
 minimum_distance = INF;
 for (i = 0; i < size; i++)
  if (distance[i] < minimum_distance && !visited_nodes[i])
          {
   minimum_distance = distance[i];
   next_node = i;
  }
  visited_nodes[next_node] = 1;
for (i = 0; i < size; i++)
 if (!visited_nodes[i])
  if (minimum_distance + cost[next_node][i] < distance[i])
          {
   distance[i] = minimum_distance + cost[next_node][i];
   previous[i] = next_node;
  }
 counter++;
}
for (i = 0; i < size; i++)
if (i != start)
{
 printf("\nDistance from the Source Node to %d: %d", i, distance[i]);
```

```
  }
}
int main()
{
 int Graph[MAX][MAX], i, j, size, source;
 size = 7;
 Graph[0][0] = 0;
 Graph[0][1] = 4;
 Graph[0][2] = 0;
 Graph[0][3] = 0;
 Graph[0][4] = 0;
 Graph[0][5] = 8;
 Graph[0][6] = 0;

 Graph[1][0] = 4;
 Graph[1][1] = 0;
 Graph[1][2] = 8;
 Graph[1][3] = 0;
 Graph[1][4] = 0;
 Graph[1][5] = 11;
 Graph[1][6] = 0;

 Graph[2][0] = 0;
 Graph[2][1] = 8;
```

```
Graph[2][2] = 0;
Graph[2][3] = 7;
Graph[2][4] = 0;
Graph[2][5] = 4;
Graph[2][6] = 0;

Graph[3][0] = 0;
Graph[3][1] = 0;
Graph[3][2] = 7;
Graph[3][3] = 0;
Graph[3][4] = 9;
Graph[3][5] = 14;
Graph[3][6] = 0;

Graph[4][0] = 0;
Graph[4][1] = 0;
Graph[4][2] = 0;
Graph[4][3] = 9;
Graph[4][4] = 0;
Graph[4][5] = 10;
Graph[4][6] = 2;

Graph[5][0] = 0;
Graph[5][1] = 0;
```

```
Graph[5][2] = 4;
Graph[5][3] = 14;
Graph[5][4] = 10;
Graph[5][5] = 0;
Graph[5][6] = 2;

Graph[6][0] = 0;
Graph[6][1] = 0;
Graph[6][2] = 0;
Graph[6][3] = 0;
Graph[6][4] = 2;
Graph[6][5] = 0;
Graph[6][6] = 1;
source = 0;
DijkstraAlgorithm(Graph, size, source);
return 0;
}
```

Output-

```
Distance from the Source Node to 1: 4
Distance from the Source Node to 2: 12
Distance from the Source Node to 3: 19
Distance from the Source Node to 4: 12
Distance from the Source Node to 5: 8
Distance from the Source Node to 6: 10Press any key to continue . . .
```

20.) Write a Program for Floyd Warshall algorithm implementation in C.

Source Code-

```
#include <stdio.h>

#define V 4

#define INF 99999

void printSolution(int dist[][V]);

void floydWarshall(int dist[][V])

{

 int i, j, k;

 for (k = 0; k < V; k++)

 {

  for (i = 0; i < V; i++)

  {

   for (j = 0; j < V; j++)

   {

   if (dist[i][k] + dist[k][j] < dist[i][j])

    dist[i][j] = dist[i][k] + dist[k][j];

   }

  }

 }

 printSolution(dist);

}

void printSolution(int dist[][V])

{
```

```
printf(
"The following matrix shows the shortest distances"
" between every pair of vertices \n");
for (int i = 0; i < V; i++)
{
 for (int j = 0; j < V; j++)
 {
  if (dist[i][j] == INF)
   printf("%7s", "INF");
  else
   printf("%7d", dist[i][j]);
 }
 printf("\n\n");
}
}
int main()
{
int graph[V][V] = { { 0, 5, INF, 10 },
                    { INF, 0, 3, INF },
                    { INF, INF, 0, 1 },
                    { INF, INF, INF, 0 }};
floydWarshall(graph);
return 0;
}
```

Output-

```
The following matrix shows the shortest distances between every pair of vertices
      0      5      8      9

    INF      0      3      4

    INF    INF      0      1

    INF    INF    INF      0
```