SVR ENGINEERING COLLEGE

AYYALURUMETTA (V), NANDYAL, KURNOOL
DT.ANDHRA PRADESH – 518502

2018-19

LABORATORY MANUAL

OF

DATA STRUCTURES LABORATORY

(15A05202)

(R-15 REGULATION)

Prepared by

Mr. V.RAJA SEKHAR
Asst. Professor

For

B.Tech I YEAR – II SEM. (CSE)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SVR ENGINEERING COLLEGE

(AFFILIATED TO JNTUA ANANTHAPURAM- AICITE-INDIA)
AYYALURUMETTA (V), NANDYAL, KURNOOL
DT.ANDHRA PRADESH – 518502

LAB MANUAL CONTENT

DATA STRUCTURES LABORATORY

(15A05202)

Institute Vision & Mission, Department Vision & Mission

1. PO, PEO& PSO Statements.
2. List of Experiments
3. CO-PO Attainment
4. Experiment Code and Outputs

1. Institute Vision & Mission, Department Vision & MissionInstitute Vision:

To produce Competent Engineering Graduates & Managers with a strong base of Technical & Managerial Knowledge and the Complementary Skills needed to be Successful Professional Engineers & Managers.

Institute Mission:

To fulfill the vision by imparting Quality Technical & Management Education to the Aspiring Students, by creating Effective Teaching/Learning Environment and providing State – of the – Art Infrastructure and Resources.

Department Vision:

To produce Industry ready Software Engineers to meet the challenges of 21st Century.

Department Mission:

- Impart core knowledge and necessary skills in Computer Science and Engineering through innovative teaching and learning methodology.
- Inculcate critical thinking, ethics, lifelong learning and creativity needed for industry and society.
- Cultivate the students with all-round competencies, for career, higher education and self-employability.

2. PO, PEO& PSO

StatementsPROGRAMME OUTCOMES (POs)

PO-1: Engineering knowledge - Apply the knowledge of mathematics, science, engineering fundamentals of Computer Science& Engineering to solve complex real-life engineering problems related to CSE.

PO-2: Problem analysis - Identify, formulate, review research literature, and analyze complex engineering problems related to CSE and reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO-3: Design/development of solutions - Design solutions for complex engineering problems related to CSE and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, cultural, societal and environmental considerations.

PO-4: Conduct investigations of complex problems - Use research-based knowledge and research methods, including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.

PO-5: Modern tool usage - Select/Create and apply appropriate techniques, resources and modern engineering and IT tools and technologies for rapidly changing computing needs, including prediction and modeling to complex engineering activities, with an understanding of the limitations.

PO-6: The engineer and society - Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the CSE professional engineering practice.

PO-7: Environment and Sustainability - Understand the impact of the CSE professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development.

PO-8: Ethics - Apply ethical principles and commit to professional ethics and responsibilities and norms of the relevant engineering practices.

PO-9: Individual and team work - Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO-10: Communication - Communicate effectively on complex engineering activities with the engineering community and with the society-at-large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, give and receive clear instructions.

PO-11: Project management and finance - Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO-12: Life-long learning - Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadcast context of technological changes.

Program Educational Objectives (PEOs):

PEO 1:Graduates will be prepared for analyzing, designing, developing and testing the software solutions and products with creativity and sustainability.

PEO 2: Graduates will be skilled in the use of modern tools for critical problem solvingand analyzing industrial and societal requirements.

PEO 3:Graduates will be prepared with managerial and leadership skills for career and starting up own firms.

Program Specific Outcomes (PSOs):

PSO 1: Develop creative solutions by adapting emerging technologies / tools for real time applications.

PSO 2: Apply the acquired knowledge to develop software solutions and innovative mobile apps for various automation applications

2.1 Subject Time Table

SVR ENGINEERING COLLEGE::NANDYAL DEPARTMENT OF CSE								
Mr. V.RAJA SEKHAR						I-II		
Day/ Time	9:30 AM	10:20 AM	11:30 AM	12:20 PM-	LUNCH BREAK	02:00 PM	02:50 PM	03:40 PM
	10:20 AM	11:10AM	12:20 PM	01:10 PM		02:50 PM	03:40 PM	04:30 PM
MON								
TUE								
WED						DSLAB		
THU								
FRI								
SAT								

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR
ANANTHAPURAMU

B.Tech. I - II Sem. (CSE) **P C**
4 2

(15A05202) DATA STRUCTURES LAB
(Common to CSE & IT Branches of Engineering)

Course Objectives:

- To strengthen the ability to identify and apply the suitable data structure for the given real world problem

Course Outcomes:

- Apply problem solving techniques to find solutions to problems
- Able to identify the appropriate data structure for a given problem or application.
- Improve logical skills

List of Experiments/Tasks

1. Write a program to sort the elements of an array using sorting by exchange.
2. Write a program to sort the elements of an array using Selection Sort.
3. Write a program to implement heap sort.
4. Write a program to perform Linear Search on the elements of a given array.
5. Write a program to perform Binary Search on the elements of a given array.
6. Write a program to convert infix expression to postfix expression and evaluate postfix expression.
7. Write a program to implement stack, queue, circular queue using arrays and linked lists.
8. Write a program to perform the operations creation, insertion, deletion, and traversing a singly linked list.
9. Write a program to perform the operations creation, insertion, deletion, and traversing a Doubly linked list.
10. Write a program to remove duplicates from ordered and unordered

arrays.

11. Write a program to sort numbers using insertion sort.
12. Write a program to implement quick sort using non-recursive and recursive approaches. Use randomized element as partitioning element.
13. Write a program to search a word in a given file and display all its positions.
14. Write a program for tic-tac-toe game.
15. Write a program to perform operations creation, insertion, deletion and traversing on a binary search tree.
16. Write a program to implement depth first search and breadth first search on graphs.
17. Write a program to perform different operations on Red Black trees.
18. Write a program to implement external sorting.
19. Write a program to perform different operations of B Tree.

Note:

1. Instructors are advised to conduct the lab in LINUX/UNIX environment
2. The above list consists of only sample programs. Instructors may choose other programs to illustrate certain concepts, wherever is necessary. Programs should be there on all the concepts studied in Theory. Instructors are advised to change atleast 25% of the programs every year until the next syllabus revision.

References:

1. Fundamentals of Data Structures in C", Horowitz, Sahni, Anderson-freed, Second Edition, Universities Press.
2. Data structures and Algorithms using C++, Ananda Rao Akepogu and Radhika Raju Palagiri, Pearson Education.

SVR ENGINEERING COLLEGE			
Department:	**COMPUTER SCIENCE & ENGINEERING**		
Course Outcome Attainment - Internal Assessments			
Name of the faculty :	V.RAJA SEKHAR	Academic Year:	2018-19
Branch & Section:	COMPUTER SCIENCE & ENGINEERING	Exam:	EXTERNAL LAB
Course:	DATA STRUCTURES LAB	Semester:	I-II SEM

Course Outcomes	**Internal Lab**			**University Exam**
15A05202.1	3			3
15A05202.2	3			3
15A05202.3	3			3
15A05202.4	3			3
15A05202.5	3			3

Course Outcomes		Attainment Level
15A05202.1	Apply problem solving techniques to find solutions to problems	**3.00**
15A05202.2	Able to identify the appropriate data structure for a given problem or application.	**3.00**
15A05202.3	Improve logical skills	**3.00**
15A05202.4	It strengthen the ability to the students to identify and apply the suitable data structure for the given real world problem	**3.00**
15A05202.5	It enables them to gain knowledge in practical applications of data structures .	**3.00**
	Average Attainment	**3.00**

Overall Course Attainment	**3**

SVR ENGINEERING COLLEGE

DEPARTMENT	**COMPUTER SCIENCE & ENGINEERING**		
PROGRAM OUTCOME ATTAINMENT			
Name of Faculty:	V.RAJA SEKHAR	Academic Year	2018-19
Branch & Section:	COMPUTER SCIENCE & ENGINEERING	SUB CODE:	15A05202
Course:	DATA STRUCTURES LAB	Semester:	I-II

COURSE OUTCOME ATTAINMENT

Course outcome attainment			**Internal lab**	**External lab**
15A05202.1			3	3
15A05202.2			3	3
15A05202.3			3	3
15A05202.4			3	3
15A05202.5			3	3
Average of CO Attainment			3	3
Attainment of the Course			3	

COURSE OUTCOMES AND PROGRAM OUTCOMES MAPPING

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
15A05202.1	3	3	2	2	1	2			1		1	2	3	1
15A05202.2	3	2	1	1		1		1				1	2	2
15A05202.3	3	3	2	2	2	1	1			1			2	1
15A05202.4	3	3	1	2		2			2		2	2	2	2
15A05202.5	3	2	2	2		2		2		1			3	1
Average	3.0	2.6	1.6	1.8	1.5	1.6	1.0	1.5	1.5	1.0	1.5	1.7	2.4	1.4
Overall Attainment of the Course	**3**	**2.6**	**1.6**	**1.8**	**1.5**	**1.6**	**1**	**1.5**	**1.5**	**1**	**1.5**	**1.67**	**2.4**	**1.4**

Faculty V.RAJA SEKHAR Head of the Department

1. **Write a program to sort the elements of an array using sorting by exchange.**

Program:

```
#include<stdio.h>
#define MAX 50
void main()

{
   int arr[MAX],i,j,n,small,temp;
  clrscr();
   printf("\n Enter No.of elements in the list\n");
   scanf("%d", &n);
   printf("\n Enter the elements in the list\n");
   for(i=0;i<n;i++)
         scanf("%d",&arr[i]);
   printf("\n Array before sorting\n");
   for(i=0;i<n;i++)
         printf("%d \t",arr[i]);
   for(i=0;i<n-1;i++)
   {
         small=i;
         for(j=i+1;j<n;j++)
         {
               if(arr[j]<arr[small])
                     small=j;
         }
         temp=arr[i];
         arr[i]=arr[small];
         arr[small]=temp;
   }
   printf("\n Array after sorting\n");
   for(i=0;i<n;i++)
         printf("\t%d",arr[i]);
  getch();
}
```

Sample Output:

```
Enter No. of elements in the list
5
Enter the elements in the list
23
12
7
56
43
Array before sorting
23      12     7       56     43
Array after sorting
 7       12     23     43     56
```

2. Write a program to sort the elements of an array using Selection Sort.

```
/*Write a program to sort the elements of an array using
Selection Sort.*/
#include<stdio.h>
#define MAX 50
void main()
{
        int arr[MAX],i,j,n,small,temp;
       clrscr();
        printf("\n Enter No.of elements in the list\n");
        scanf("%d", &n);
        printf("\n Enter the elements in the list\n");
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        printf("\n Array before sorting\n");
        for(i=0;i<n;i++)
                printf("%d \t",arr[i]);
        for(i=0;i<n-1;i++)
        {
                small=i;
                for(j=i+1;j<n;j++)
                {
                        if(arr[j]<arr[small])
                                small=j;
                }
                temp=arr[i];
                arr[i]=arr[small];
                arr[small]=temp;
        }
        printf("\n Array after sorting\n");
        for(i=0;i<n;i++)
```

```
                printf("\t%d",arr[i]);
        getch();
    }
```

Sample Output:

```
Enter No. of elements in the list
5
Enter the elements in the list
15
4
32
7
89
Array before sorting
15      4       32      7       89
Array after sorting
```

3 .Write a program to implement heap sort.

```
#include<stdio.h>
int i,j,k,flag;
clrscr();
void swap(int *x, int *y)
{
        int temp;
       temp=*x;
       *x=*y;
       *y=temp;
}
void adjust(int a[],int i,int n)
{
       k=a[i];
       flag=1;
       j=2*i;
       while(j<=n && flag)
       {
              if(j<n && a[j]<a[j+1])
              j++;
              if(k>=a[j])
              flag=0;
              else
              {
                     a[j/2]=a[j];
                     j=j*2;
              }
       }
        a[j/2]=k;
}

void buildheap(int a[],int n)
       {
              for(i=(n/2);i>=0;i--)
```

```
                adjust(a,i,n-1);
        }
void heapsort(int a[],int n)
{
    buildheap(a,n);
   for(i=(n-2);i>=0;i--)
{
    swap(&a[0],&a[i+1]);
    adjust(a,0,i);
}
}
void main()
{
   int a[50],n;

printf("Enter Size of the array\n");
scanf("%d",&n);
printf("Enter elements\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("List before sorting\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
heapsort(a,n);
printf("\n The list after sorting\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
getch();
}
```

Sample output:

```
Enter Size of the array
5
Enter elements
12
6
22
44
10
List before sorting
12      6       22      44      10
The list after sorting
6       10      12      22      44
```

4.Write a program to perform Linear Search on the elements of a given array.

```
#include<stdio.h>
#define MAX 50
int linearsearch(int [],int,int);
void main()
{
        int arr[MAX],i,n,key,pos;
       clrscr();
        printf("\n Enter No.of elements in the list\n");
        scanf("%d",&n);
        printf("\nEnter the elements\n");
        for(i=0;i<n;i++)
        scanf("%d",&arr[i]);
        printf("\n Enter the element to be searched for\n");
        scanf("%d",&key);
        pos=linearsearch(arr,n,key);
        if(pos!=-1)
        printf("\n The element %d is found at position:%d",key,pos+1);
        else
       printf("\n Element not found\n");

}
int linearsearch(int arr[],int n,int key)
{
        int i;
        for(i=0;i<n-1;i++)
        {
                if(key==arr[i])
               return i;
        }
        return -1;
getch();
```

```
}
```

Sample Output:

```
Enter No. of elements in the list
5
Enter the elements
45
34
23
12
67
Enter the element to be searched for
23
The element 23 is found at position:3
```

5.Write a program to perform Binary Search on the elements of a given array.

```
#include<stdio.h>
#define MAX 50
int binarysearch(int [],int,int);
void bubblesort(int [],int);
void main()
{
        int arr[MAX],i,n,pos,key;
       clrscr();
        printf("\n Enter No.of elements in the list:\n");
        scanf("%d",&n);
        printf("\n Enter the elements\n");
        for(i=0;i<n;i++)
       scanf("%d",&arr[i]);
        printf("Enter element to be searched for\n");
        scanf("%d",&key);
        bubblesort(arr,n);
        printf("\n Array after sorting\n");
        for(i=0;i<n;i++)
       printf("\t %d",arr[i]);
        pos=binarysearch(arr,n,key);
        if(pos!=-1)
        printf("\n The element %d is found at position %d",key,pos+1);
        else
       printf("\n Element is not found\n");
}
void bubblesort(int arr[],int n)
{
       int i,j,temp;
       for(i=0;i<n-1;i++)
       {
              for(j=i+1;j<n;j++)
              {
                     if(arr[i]>arr[j])
                     {
                            temp=arr[i];
```

```
                        arr[i]=arr[j];
                        arr[j]=temp;
                    }
                }
            }
}
int binarysearch(int arr[],int n,int key)
{
        int i,first,middle,last;
        first=0;
        last=n-1;
        while(last>=first)
        {
                middle=(first+last)/2;
                if(key>arr[middle])
                        first=middle+1;
                else if(key<arr[middle])
                        last=middle-1;
                else
                        return middle;
        }
        return -1;

}
```

Sample Output:

```
Enter No. of elements in the list:
5
Enter the elements
32
45
21
78
44
Enter the element to be searched for
78
Array after sorting
21    32   44    45    78
The element 78 is found at position 5
```

6.Write a program to convert infix expression to postfix expression and evaluate postfix expression.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
char stack[20];
int top=-1;
void topost(char []);
void push(char);
char pop();
int priority(char);
void main()
{
  char infix[25];
printf("\n Enter an infix expression:\n");
gets(infix);
printf("\n Equivalent postfix expression is:\n");
topost(infix);
}
int priority(char ch)
{
  if(ch=='*') return(5);
else if(ch=='/') return(4);
else if(ch=='+') return(3);
else
  return(2);
}

void topost(char infix[])
{
  int len;
static int index=0,pos=0;
char s,t,t1;
char postfix[40];
len=strlen(infix);
push('#');
while(index<len)
```

```
{
s=infix[index];
switch(s)
{
   case '(':push(s);
        break;
   case ')':t=pop();
            while(t1='(')
        {
           postfix[pos]=t;
            pos++;
           t=pop();
        }
        break;
    case '+':
    case '-':
     case '*':
     case '/':
     case '^': while(priority(stack[top])>priority(s))
               {
                 t=pop();
                postfix[pos]=t;
                 pos++;
               }
               push(s);break;
        default:postfix[pos++]=s;
               break;
}
index++;
}
while(top>0)
{
  t=pop();
   postfix[pos++]=t;
}
postfix[pos++]='\0';
puts(postfix);
}
void push(char s)
```

```
{
  if(top>=19)
   {
    puts("\n Stack overflow\n");
    exit(1);
    }
   top=top+1;
stack[top]=s;
}
char pop()
{
 char num;
if(top==-1)
{
   puts("\n Stack underflow\n");
exit(1);
}
num=stack[top];
top=top-1;
return num;
}
```

Sample Output:

```
Enter an infix expression
a+b*c
Equivalent postfix expression is:
abc*+
```

7.Write a program to implement stack, queue, circular queue using arrays and linked lists.

PROGRAM TO CREATE STACK BY USING ARRAYS

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
# define MAX 6
int stack[MAX];
int top = 0;
int menu()
{
        int ch;
        clrscr();
        printf("\n … Stack operations using ARRAY… ");
        printf("\n -----------**********-------------\n");
        printf("\n 1. Push ");
        printf("\n 2. Pop ");
        printf("\n 3. Display");
        printf("\n 4. Quit ");
        printf("\n Enter your choice: ");
        scanf("%d", &ch);
        return ch;
}
```

```
void display()
{
	int i;
	if(top == 0)
	{
		printf("\n\nStack empty..");
		return;
	}
	else
	{
		printf("\n\nElements in stack:");
		for(i = 0; i < top; i++)
		printf("\t%d", stack[i]);
	}
}
void pop()
{
	if(top == 0)
	{
		printf("\n\nStack Underflow..");
		return;
	}
	else
	printf("\n\npopped element is: %d ", stack[--top]);
}
```

```
void push()
{
	int data;
	if(top == MAX)
	{
		printf("\n\nStack Overflow..");
		return;
	}
	else
	{
		printf("\n\nEnter data: ");
		scanf("%d", &data);
		stack[top] = data;
		top = top + 1;
		printf("\n\nData Pushed into the stack");
	}
}
void main()
{
	int ch;
	do
	{
		ch = menu();
		switch(ch)
		{
```

```
                case 1:
                        push();
                        break;
                case 2:
                        pop();
                        break;
                case 3:
                        display();
                        break;
                case 4:
                        exit(0);
                }
            getch();
        } while(1);
}
```

output:

```
Stack operations using ARRAY…
 ----------**********-------------

 1. Push
 2. Pop
 3. Display
 4. Quit
Enter your choice: 1

Enter data: 12

Data Pushed into the stack
```

Enter your choice: 1

Enter data: 56

Data Pushed into the stack

Enter your choice: 1

Enter data: 48

Data Pushed into the stack

Enter your choice: 1

Enter data: 32

Data Pushed into the stack

Enter your choice: 3

Elements in stack: 12 12 56 48 32

Enter your choice: 2

popped element is: 32

Enter your choice: 4

Exit the program

/* cprogram to implement the circular queue using linked lists*/

```
#include<stdio.h>
#include<stdlib.h>
#define que struct queue
#define pf printf
#define sf scanf
Clrscr();
struct queue{
int info;
struct queue *link;
};
 que *front=NULL,*rear=NULL;
int count=0;
void push(int n)
{
que *newnode;
newnode=(struct queue*)malloc(sizeof(struct queue));
newnode->info=n;
newnode->link=NULL;
if(count==0)
front=newnode;
else
        rear->link=newnode;
   rear=newnode;
   rear->link=front;
count++;
}
int pop(void)
{
int n;
que *temp;
if(count==0)
```

```
return (-1);
count--;
   if(front==rear)
   {
      n=front->info;
      free(front);
      front=NULL;
      rear=NULL;
   }else
   {
         temp= front ;
         n = temp-> info ;
         front = front -> link ;
         rear -> link = front ;
         free ( temp ) ;
   }
return n;
}
void display(void)
{
que *temp;
int i;
if(count==0)
pf("Empty");
else
{
temp=front;
for(i=0;i<count;i++)
{
pf("%d ",temp->info);
temp=temp->link;
}
}
pf("\n");
```

```
}
int size(void)
{
return count;
}
int main()
{
int n,ch=10;
while(ch!=0)
{
pf("\n What do you want to do??\n");
pf("1.Push\n");
pf("2.Pop\n");
pf("3.SizeOfQueue\n");
pf("4.Display\n");
pf("0.EXIT\n");
sf("%d",&ch);
switch(ch)
{
case 1:
{
pf("What no. do you want to push in queue\n");
sf("%d",&n);
push(n);
break;
}
case 2:
{
n=pop();
if(n==-1)
pf("Queue is empty\n");
else
pf("Number poped from queue is %d\n",n);
break;
```

```
}
case 3:
{
n=size();
pf("Size of queue is %d\n",n);
break;
}
case 4:
{
pf("Queue is -->> ");
display();
}
case 0:
break;
default:
pf("Wrong Choice\n");
break;
}
}
}
```

output:

```
what do you want to do??
1.push
2.pop
3.size of queue
4.display
0.exit
1
what no.do you want to push in queue
25what do you want to do??
1.push
2.pop
3.size of queue
```

```
4.display
0.exit
1
what no.do you want to push in queue
45
what do you want to do??
1.push
2.pop
3.size of queue
4.display
0.exit
1
what no.do you want to push in queue
86
what do you want to do??
1.push
2.pop
3.size of queue
4.display
0.exit
1
what no.do you want to push in queue
12
what do you want to do??
1.push
2.pop
3.size of queue
4.display
0.exit
4
queue is -->25 45 86 12
2
Number poped from queue is 25
```

```
 What do you want to do??
1.Push
2.Pop
3.SizeOfQueue
4.Display
0.EXIT
4
Queue is -->> 45 86 12

 What do you want to do??
1.Push
2.Pop
3.SizeOfQueue
4.Display
0.EXIT
3
Size of queue is 3

 What do you want to do
1.Push
2.Pop
3.SizeOfQueue
4.Display
0.EXIT
0
exit the program
```

/* c program to implement the stack using linked list*/

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

struct node
{
   int info;
   struct node *ptr;
}*top,*top1,*temp;

int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();

int count = 0;

void main()
{
   int no, ch, e;

   printf("\n 1 - Push");
   printf("\n 2 - Pop");
   printf("\n 3 - Top");
   printf("\n 4 - Empty");
   printf("\n 5 - Exit");
   printf("\n 6 - Dipslay");
   printf("\n 7 - Stack Count");
   printf("\n 8 - Destroy stack");

   create();

   while (1)
   {
```

```
printf("\n Enter choice : ");
scanf("%d", &ch);

switch (ch)
{
case 1:
   printf("Enter data : ");
   scanf("%d", &no);
   push(no);
   break;
case 2:
   pop();
   break;
case 3:
   if (top == NULL)
      printf("No elements in stack");
   else
   {
      e = topelement();
      printf("\n Top element : %d", e);
   }
   break;
case 4:
   empty();
   break;
case 5:
   exit(0);
case 6:
   display();
   break;
case 7:
   stack_count();
   break;
case 8:
   destroy();
   break;
default :
   printf(" Wrong choice, Please enter correct choice  ");
   break;
}
```

```
    }
}

/* Create empty stack */
void create()
{
    top = NULL;
}

/* Count stack elements */
void stack_count()
{
    printf("\n No. of elements in stack : %d", count);
}

/* Push data into stack */
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++;
}

/* Display stack elements */
void display()
{
    top1 = top;

    if (top1 == NULL)
```

```
    {
        printf("Stack is empty");
        return;
    }

    while (top1 != NULL)
    {
        printf("%d ", top1->info);
        top1 = top1->ptr;
    }
 }

/* Pop Operation on stack */
void pop()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("\n Error : Trying to pop from empty stack");
        return;
    }
    else
        top1 = top1->ptr;
    printf("\n Popped value : %d", top->info);
    free(top);
    top = top1;
    count--;
}

/* Return top element */
int topelement()
{
    return(top->info);
}

/* Check if stack is empty or not */
void empty()
{
    if (top == NULL)
```

```
        printf("\n Stack is empty");
    else
        printf("\n Stack is not empty with %d elements", count);
}

/* Destroy entire stack */
void destroy()
{
    top1 = top;

    while (top1 != NULL)
    {
        top1 = top->ptr;
        free(top);
        top = top1;
        top1 = top1->ptr;
    }
    free(top1);
    top = NULL;

    printf("\n All stack elements destroyed");
    count = 0;
}
```

OUTPUT:

```
1 - Push
2 - Pop
3 - Top
4 - Empty
5 - Exit
6 - Dipslay
7 - Stack Count
8 - Destroy stack
Enter choice : 1
Enter data : 56

Enter choice : 1
Enter data : 80

Enter choice : 2
```

Popped value : 80
Enter choice : 3

Top element : 56
Enter choice : 1
Enter data : 78

Enter choice : 1
Enter data : 90

Enter choice : 6
90 78 56
Enter choice : 7

No. of elements in stack : 3
Enter choice : 8

All stack elements destroyed
Enter choice : 4

Stack is empty
Enter choice : 5

/*PROGRAM FOR QUEUE USING ARRAYS*/

```
# include <conio.h>
# define MAX 6
int Q[MAX];
int front, rear;
void insertQ()
{
        int data;
        if(rear == MAX)
        {
                printf("\n Linear Queue is full");
                return;
        }
        else
        {
                printf("\n Enter data: ");
                scanf("%d", &data);
                Q[rear] = data;
                rear++;
                printf("\n Data Inserted in the Queue ");
        }
}
```

```
void deleteQ()
{
	if(rear == front)
	{
		printf("\n\n Queue is Empty..");
		return;
	}
	else
	{
		printf("\n Deleted element from Queue is %d", Q[front]);
		front++;
	}
}
void displayQ()
{
	int i;
	if(front == rear)
	{
		printf("\n\n\t Queue is Empty");
		return;
	}
	else
	{
		printf("\n Elements in Queue are: ");
		for(i = front; i < rear; i++)
		{
			printf("%d\t", Q[i]);
```

```
                }
        }
}
int menu()
{
        int ch;
        clrscr();
        printf("\n \tQueue operations using ARRAY..");
        printf("\n -----------**********-------------\n");
        printf("\n 1. Insert ");
        printf("\n 2. Delete ");
        printf("\n 3. Display");
        printf("\n 4. Quit ");
        printf("\n Enter your choice: ");
        scanf("%d", &ch);
        return ch;
}
void main()
{
        int ch;
        do
        {
                ch = menu();
                switch(ch)
                {
                        case 1:
                                insertQ();
```

```
                    break;
                case 2:
                    deleteQ();
                    break;
                case 3:
                    displayQ();
                    break;
                    case 4:
                    return;
                }
            getch();
        } while(1);
}
```

OUTPUT:

```
    Queue operations using ARRAY..
 ----------**********----------

 1. Insert
 2. Delete
 3. Display
 4. Quit
 Enter your choice: 1

 Enter data: 26
```

Data Inserted in the Queue

Enter your choice: 1

Enter data: 58

Data Inserted in the Queue

Enter your choice: 1

Enter data: 37

Data Inserted in the Queue

Enter your choice: 1

Enter data: 24

Data Inserted in the Queue

Enter your choice: 3

Elements in Queue are: 26 58 37 24

Enter your choice: 2

Deleted element from Queue is 26

Enter your choice: 4

Exit the program

PROGRAM FOR QUEUE USING Linked list

```
# include <stdlib.h>
# include <conio.h>
struct queue
{
	int data;
	struct queue *next;
};
typedef struct queue node;
node *front = NULL;
node *rear = NULL;
node* getnode()
{
	node *temp;
	temp = (node *) malloc(sizeof(node)) ;
	printf("\n Enter data ");
	scanf("%d", &temp -> data);
	temp -> next = NULL;
	return temp;
}
void insertQ()
{
	node *newnode;
	newnode= getnode();
	if(newnode== NULL)
	{
		printf("\n Queue Full");
		return;
}
	if(front == NULL)
	{
		front = newnode;
		rear=newnode;
	}
	else
	{
		rear -> next = newnode;
		rear=newnode;
```

```
        }
        printf("\n\n\t Data Inserted into the Queue..");
}
void deleteQ()
{
        node *temp;
        if(front == NULL)
        {
                printf("\n\n\t Empty Queue..");
                return;
        }
                temp = front;
                front = front -> next;
                printf("\n\n\t Deleted element from queue is %d ", temp
        -> data);
                free(temp);
        }

void displayQ()
{
        node *temp;
        if(front == NULL)
        {
        printf("\n\n\t\t Empty Queue ");
        }
        else
        {
                temp = front;
                printf("\n\n\n\t\t Elements in the Queue are: ");
                while(temp != NULL)
                {
                        printf("%5d ", temp -> data);
                        temp = temp -> next;
                }
        }
}


char menu()
```

```
{
        char ch;
        clrscr();
        printf("\n \t..Queue operations using linked list.. ");
        printf("\n\t ----------**********------------\n");
        printf("\n 1. Insert ");
        printf("\n 2. Delete ");
        printf("\n 3. Display");
        printf("\n 4. Quit ");
        printf("\n Enter your choice: ");
        ch = getche();
        return ch;
}
void main()
{
        char ch;
        do
        {
                ch = menu();
                switch(ch)
                {
                        case '1' :
                                        insertQ();
                                        break;
                        case '2' :
                                        deleteQ();
                                        break;
                        case '3' :
                                        displayQ();
                                        break;
                        case '4':
                                        return;
                }
                getch();
        }
        while(ch != '4');
}
```

Output:

```
    ..Queue operations using linked list..
       ----------**********-------------

 1. Insert
 2. Delete
 3. Display
 4. Quit
 Enter your choice: 1
 Enter data
10
Data Inserted into the Queue..
Enter your choice: 1
 Enter data
20
Data Inserted into the Queue..
Enter your choice: 1
 Enter data
30
Data Inserted into the Queue..

Enter your choice: 1
 Enter data
40
Data Inserted into the Queue..


Enter your choice: 1
 Enter data
50
Data Inserted into the Queue..

Enter your choice: 3
 Elements in the Queue are:    10    20    30    40  50


Enter your choice: 2

       Deleted element from queue is  10
```

Enter your choice: 3

Elements in the Queue are: 20 30 40 50

Enter your choice: 4
Exit the program

/*write a program to create a circular list*/

```
# include <stdio.h>
# include <conio.h>
# define MAX 6
int CQ[MAX];
int front = 0;
int rear = 0;
int count = 0;
void insertCQ()
{
	int data;
	if(count ==MAX)
	{
		printf("\n Circular Queue is Full");
	}
	else
	{
		printf("\n Enter data: ");
		scanf("%d", &data);
		CQ[rear] = data;
		rear = (rear + 1) % MAX;
		count ++;
		printf("\n Data Inserted in the Circular Queue ");
	}
}
void deleteCQ()
```

```
{
        if(count ==0)
        {
                printf("\n\nCircular Queue is Empty..");
        }
        else
        {
                printf("\n Deleted element from Circular Queue is %d ",
        CQ[front]);
                front = (front + 1) % MAX;
                count --;
        }
}
void displayCQ()
{
        int i, j;
        if(count ==0)
        {
                printf("\n\n\t Circular Queue is Empty ");
        }
        else
        {
                printf("\n Elements in Circular Queue are: ");
                j = count;
                for(i = front; j != 0; j--)
                {
```

```
                printf("%d\t", CQ[i]);
                i = (i + 1) % MAX;
            }
        }
}
int menu()
{
        int ch;
        clrscr();
        printf("\n \t Circular Queue Operations using ARRAY..");
        printf("\n -----------**********-------------\n");
        printf("\n 1. Insert ");
        printf("\n 2. Delete ");
        printf("\n 3. Display");
        printf("\n 4. Quit ");
        printf("\n Enter Your Choice: ");
        scanf("%d", &ch);
        return ch;
}
void main()
{
        int ch;
        do
        {
            ch = menu();
            switch(ch)
```

```
            {
                case 1:
                    insertCQ();
                    break;
                case 2:
                    deleteCQ();
                    break;
                case 3:
                    displayCQ();
                    break;
                case 4:
                    return;
                default:
                printf("\n Invalid Choice ");
            }
            getch();
        } while(1);
}
```

Output:

Circular Queue Operations using ARRAY..

----------**********----------

1. Insert
2. Delete
3. Display
4. Quit

Enter Your Choice: 1

Enter data: 10

Data Inserted in the Circular Queue

Enter Your Choice: 1

Enter data: 20

Data Inserted in the Circular Queue

Enter Your Choice: 1

Enter data: 30

Data Inserted in the Circular Queue

Enter Your Choice: 1

Enter data: 40

Data Inserted in the Circular Queue

Enter Your Choice: 1

Enter data: 50

Data Inserted in the Circular Queue
Enter Your Choice: 3
Elements in Circular Queue are: 10 20 30 40 50

Enter Your Choice: 2
Deleted element from Circular Queue is 10
Enter Your Choice: 4
Exit the program

8.Write a program to perform the operations creation, insertion, deletion, and traversing a singly linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct list
{
        int num;
        struct list *next;
}       *header,*first,*rear;
void create();
void atbeg();
void atend();
void atpos(int);
void show();
void main()
{
        int pos,choice;
        create();
        show();
        while(1)
        {
                printf("\n MENU\n");
                printf("\n 1. At Beginning 2. At End  3. At position  4. Show  5.Exit\n");
                printf("\n Enter Ur choice\n");
                scanf("%d", &choice);
                switch(choice)
                {
                        case 1:atbeg();break;
                        case 2: atend(); break;
                        case 3:printf("\n Enter position where to be inserted\n");
                                scanf("%d", &pos);
                                atpos(pos);
                                break;
                        case 4: show();break;
                        case 5:exit(0);
                        break;
```

```
                default:printf("\n Invalid choice\n");
            }
        }
}

void create()
{
        struct list *node;
        printf("\n Enter the Elements , press zero to stop\n");
        if(header==NULL)
        {
                first=(struct list *) malloc(sizeof(struct list));
                scanf("%d", &first->num);
                first->next=header;
                header=first;
                rear=first;
        }
        while(1)
        {
                node=(struct list *) malloc(sizeof(struct list));
                scanf("%d", &node->num);
                if(node->num==0)
                break;
                node->next=NULL;
                rear->next=node;
                rear=node;
        }
}
void atbeg()
{
        struct list *node,*t;
        node =(struct list *)malloc(sizeof(struct list));
        printf("\n Enter an element to insert at beginning:\n");
        scanf("%d",&node->num);
        t=first;
        header=node;
        header->next=t;
        first=header;
}
```

```
void atend()
{
        struct list *t;
       t=rear;
       rear=(struct list *)malloc(sizeof(struct list));
       printf("\n Enter an alement to insert at the end:\n");
       scanf("%d", &rear->num);
       t->next=rear;
       rear->next=NULL;
}

void atpos(int k)
{
        struct list *nw,*suc,*pr;
       int c=1;
       while(header!=NULL)
       {
              c++;
              header=header->next;
              if(k==1)
              pr=first;
              if(c==k)
              pr=header;
              if(c==(k+1))
              suc=header;
       }

              nw=(struct list *) malloc(sizeof(struct list));
              printf("\n Enter an element to insert\n");
              scanf("%d",&nw->num);
              pr->next=nw;
              nw->next=suc;
              header=first;
}

void show()
{
       printf("\n Linked List elements are\n");
```

```
        while(header!=NULL)
        {
                printf("\t %d", header->num);
               header=header->next;
        }
        header=first;
}
```

Sample Output:

```
Enter the elements, press zero to stop
45
23
0
Linked list elements are
45    23
Menu
1. At Beginning   2. At End   3. At position  4. Show  5. Exit
Enter Ur choice
2
Enter an element to insert at the end:
54
Menu
1. At Beginning   2. At End   3. At position  4. Show  5. Exit
Enter Ur choice
1
Enter an element to insert at beginning:
12
Menu
1. At Beginning   2. At End   3. At position  4. Show  5. Exit
Enter Ur choice
4
Linked List elements are
12    45    23    54
Menu
1. At Beginning   2. At End   3. At position  4. Show  5. Exit
```

9. /* c program for double linked list*/

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

struct node
{
   struct node *prev;
   int n;
   struct node *next;
}*h,*temp,*temp1,*temp2,*temp4;

void insert1();
void insert2();
void insert3();
void traversebeg();
void traverseend(int);
void sort();
void search();
void update();
void delete();

int count = 0;

void main()
{
   int ch;

   h = NULL;
   temp = temp1 = NULL;

   printf("\n 1 - Insert at beginning");
   printf("\n 2 - Insert at end");
   printf("\n 3 - Insert at position i");
   printf("\n 4 - Delete at i");
   printf("\n 5 - Display from beginning");
   printf("\n 6 - Display from end");
```

```
printf("\n 7 - Search for element");
printf("\n 8 - Sort the list");
printf("\n 9 - Update an element");
printf("\n 10 - Exit");

while (1)
{
   printf("\n Enter choice : ");
   scanf("%d", &ch);
   switch (ch)
   {
   case 1:
      insert1();
      break;
   case 2:
      insert2();
      break;
   case 3:
      insert3();
      break;
   case 4:
      delete();
      break;
   case 5:
      traversebeg();
      break;
   case 6:
      temp2 = h;
      if (temp2 == NULL)
         printf("\n Error : List empty to display ");
      else
      {
         printf("\n Reverse order of linked list is : ");
         traverseend(temp2->n);
      }
      break;
   case 7:
      search();
```

```
            break;
        case 8:
            sort();
            break;
        case 9:
            update();
            break;
        case 10:
            exit(0);
        default:
            printf("\n Wrong choice menu");
        }
    }
}

/* TO  create an empty node */
void create()
{
    int data;

    temp =(struct node *)malloc(1*sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    printf("\n Enter value to node : ");
    scanf("%d", &data);
    temp->n = data;
    count++;
}

/*  TO insert at beginning */
void insert1()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
```

```
    else
    {
        create();
        temp->next = h;
        h->prev = temp;
        h = temp;
    }
}

/* To insert at end */
void insert2()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {
        create();
        temp1->next = temp;
        temp->prev = temp1;
        temp1 = temp;
    }
}

/* To insert at any position */
void insert3()
{
    int pos, i = 2;

    printf("\n Enter position to be inserted : ");
    scanf("%d", &pos);
    temp2 = h;

    if ((pos < 1) || (pos >= count + 1))
    {
```

```
        printf("\n Position out of range to insert");
        return;
    }
    if ((h == NULL) && (pos != 1))
    {
        printf("\n Empty list cannot insert other than 1st position");
        return;
    }
    if ((h == NULL) && (pos == 1))
    {
        create();
        h = temp;
        temp1 = h;
        return;
    }
    else
    {
        while (i < pos)
        {
            temp2 = temp2->next;
            i++;
        }
        create();
        temp->prev = temp2;
        temp->next = temp2->next;
        temp2->next->prev = temp;
        temp2->next = temp;
    }
}

/* To delete an element */
void delete()
{
    int i = 1, pos;

    printf("\n Enter position to be deleted : ");
    scanf("%d", &pos);
    temp2 = h;
```

```
if ((pos < 1) || (pos >= count + 1))
{
   printf("\n Error : Position out of range to delete");
   return;
}
if (h == NULL)
{
   printf("\n Error : Empty list no elements to delete");
   return;
}
else
{
   while (i < pos)
   {
      temp2 = temp2->next;
      i++;
   }
   if (i == 1)
   {
      if (temp2->next == NULL)
      {
         printf("Node deleted from list");
         free(temp2);
         temp2 = h = NULL;
         return;
      }
   }
   if (temp2->next == NULL)
   {
      temp2->prev->next = NULL;
      free(temp2);
      printf("Node deleted from list");
      return;
   }
   temp2->next->prev = temp2->prev;
   if (i != 1)
      temp2->prev->next = temp2->next;
```

```
 /* Might not need this statement if i == 1 check */
       if (i == 1)
          h = temp2->next;
       printf("\n Node deleted");
       free(temp2);
   }
   count--;
}

/* Traverse from beginning */
void traversebeg()
{
   temp2 = h;

   if (temp2 == NULL)
   {
      printf("List empty to display \n");
      return;
   }
   printf("\n Linked list elements from begining : ");

   while (temp2->next != NULL)
   {
      printf(" %d ", temp2->n);
      temp2 = temp2->next;
   }
   printf(" %d ", temp2->n);
}

/* To traverse from end recursively */
void traverseend(int i)
{
   if (temp2 != NULL)
   {
      i = temp2->n;
      temp2 = temp2->next;
      traverseend(i);
      printf(" %d ", i);
```

```
    }
}

/* To search for an element in the list */
void search()
{
    int data, count = 0;
    temp2 = h;

    if (temp2 == NULL)
    {
        printf("\n Error : List empty to search for data");
        return;
    }
    printf("\n Enter value to search : ");
    scanf("%d", &data);
    while (temp2 != NULL)
    {
        if (temp2->n == data)
        {
            printf("\n Data found in %d position",count + 1);
            return;
        }
        else
             temp2 = temp2->next;
            count++;
    }
    printf("\n Error : %d not found in list", data);
}

/* To update a node value in the list */
void update()
{
    int data, data1;

    printf("\n Enter node data to be updated : ");
    scanf("%d", &data);
    printf("\n Enter new data : ");
```

```
   scanf("%d", &data1);
   temp2 = h;
   if (temp2 == NULL)
   {
      printf("\n Error : List empty no node to update");
      return;
   }
   while (temp2 != NULL)
   {
      if (temp2->n == data)
      {

         temp2->n = data1;
         traversebeg();
         return;
      }
      else
         temp2 = temp2->next;
   }

   printf("\n Error : %d not found in list to update", data);
}

/* To sort the linked list */
void sort()
{
   int i, j, x;

   temp2 = h;
   temp4 = h;

   if (temp2 == NULL)
   {
      printf("\n List empty to sort");
      return;
   }

   for (temp2 = h; temp2 != NULL; temp2 = temp2->next)
```

```
   {
      for (temp4 = temp2->next; temp4 != NULL; temp4 = temp4->next)
      {
         if (temp2->n > temp4->n)
         {
            x = temp2->n;
            temp2->n = temp4->n;
            temp4->n = x;
         }
      }
   }
   traversebeg();
}
```

OUTPUT:

```
1 - Insert at beginning
2 - Insert at end
3 - Insert at position i
4 - Delete at i
5 - Display from beginning
6 - Display from end
7 - Search for element
8 - Sort the list
9 - Update an element
10 - Exit
Enter choice : 1

Enter value to node : 10

Enter choice : 2

Enter value to node : 50

Enter choice : 4

Enter position to be deleted : 1
```

Node deleted
Enter choice : 1

Enter value to node : 34

Enter choice : 3

Enter position to be inserted : 2

Enter value to node : 13

Enter choice : 4

Enter position to be deleted : 4

Error : Position out of range to delete
Enter choice : 1

Enter value to node : 15

Enter choice : 1

Enter value to node : 67

Enter choice : 3

Enter position to be inserted : 2

Enter value to node : 34

Enter choice : 4

Enter position to be deleted : 3

Node deleted
Enter choice : 7

```
Enter value to search : 15

Error : 15 not found in list
Enter choice : 8

Linked list elements from begining :  13  34  34  50  67
Enter choice : 9

Enter node data to be updated : 45

Enter new data : 89

Error : 45 not found in list to update
Enter choice : 9

Enter node data to be updated : 50

Enter new data : 90
Enter choice : 5

Linked list elements from begining :  13  34  34  90  67
Enter choice : 6

Reverse order of linked list is :  67  90  34  34  13
Enter choice : 7

Enter value to search : 90

Data found in 4 position
Enter choice : 8

Linked list elements from begining :  13  34  34  67  90
Enter choice : 7

Enter value to search : 90

Data found in 5 position
Enter choice : 9
```

```
Enter node data to be updated : 34

Enter new data : 56

Linked list elements from begining :  13  56  34  67  90
Enter choice : 10
```

10.Write a program to remove duplicates from ordered and unordered arrays.

```
#include<stdio.h>
void main()
{
   int a[20], i, j, k, n;

   printf("\nEnter array size : ");
   scanf("%d",&n);

   printf("\nEnter %d array element : ", n);
   for(i = 0; i < n; i++)
   {
      scanf("%d",&a[i]);
    }

   printf("\nOriginal array is : ");
   for(i=0;i< n;i++)
   {
      printf(" %d",a[i]);
   }

   printf("\nNew array is  : ");
   for(i=0; i < n; i++)
   {
      for(j=i+1; j < n; )
      {
         if(a[j] == a[i])
         {
            for(k=j; k < n;k++)
            {
               a[k] = a[k+1];
            }
            n--;
         }
         else {
            j++;
         }
```

```
      }
   }

   for(i=0; i < n; i++)
   {
      printf("%d ", a[i]);
   }

}
```

Sample Output:

```
Enter array size:5
Enter 5 array element:
4
1
6
2
1
Original array is :4    1      6      2      1

New array is: 4         1      6      2
```

11.Write a program to sort numbers using insertion sort.

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 10
void insertionsort(int elements[], int n);
int elements[MAXSIZE],n;
int main()
{
        int i;
        printf("\n Enter No. of elements to sort\n");
        scanf("%d",&n);
          printf("\n Enter the elements\n");
          for(i=0;i<n;i++)
          {
                scanf("%d",&elements[i]);
           }
         printf("\n Array before sorting\n");
         for(i=0;i<n;i++)
         {
                printf("%d\t",elements[i]);
          }
        insertionsort(elements,n);
        printf("\n Array after sorting\n");
        for(i=0;i<n;i++)
         {
               printf("%d\t",elements[i]);
         }

}

void insertionsort(int elements[], int n)
{
         int i,j,index;
        for(i=1;i<n;i++)
        {
         index=elements[i];
        j=i;
```

```
        while((j>0) && (elements[j-1]>index))
        {
                elements[j]=elements[j-1];
                j=j-1;
        }
        elements[j]=index;
        }
}
```

Sample Output:

```
Enter No. of elements to sort
56
45
78
12
90

Array before sorting
56    45    78    12    90
Array after sorting
12    45    56    78    90
```

12.Write a program to implement quick sort using non-recursive and recursive approaches. Use randomized element as partitioning element.

```
/*
 * C Program to Implement Quick Sort Using Randomization
 */
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
void random_shuffle(int arr[])
{
    srand(time(NULL));
    int i, j, temp;
    for (i = MAX - 1; i > 0; i--)
    {
        j = rand()%(i + 1);
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
int partion(int arr[], int p, int r)
{
    int pivotIndex = p + rand()%(r - p + 1); //generates a random
number as a pivot
    int pivot;
    int i = p - 1;
    int j;
    pivot = arr[pivotIndex];
    swap(&arr[pivotIndex], &arr[r]);
    for (j = p; j < r; j++)
```

```
    {
        if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }

    }
    swap(&arr[i+1], &arr[r]);
    return i + 1;
}

void quick_sort(int arr[], int p, int q)
{
    int j;
    if (p < q)
    {
        j = partion(arr, p, q);
        quick_sort(arr, p, j-1);
        quick_sort(arr, j+1, q);
    }
}
int main()
{
    int i;
    int arr[MAX];
    for (i = 0; i < MAX; i++)
        arr[i] = i;
    random_shuffle(arr); //To randomize the array
    quick_sort(arr, 0, MAX-1); //function to sort the elements of
array
    for (i = 0; i < MAX; i++)
         printf("%d \n", arr[i]);
    return 0;
}
```

Output:

13.Write a program to search a word in a given file and display all its positions. C program for given a word and display all its positions

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

void main()
{
int a[5],i;
int ele,temp=0,pos=0;

clrscr();
printf("enter the array elements\n");
for (i=0; i<5; i++)
scanf("%d",&a[i]);
printf("Enter the element to be search\n");
scanf("%d",&ele);

// searching for the element

for (i=0; i<5; i++)
{
if (a[i]==ele)
{
temp=1;
pos=i;
}
}

if (temp==1)
printf("Element found %d , position==%d",ele,pos);
else
printf("Element not found\n");
} // end of main()
```

output:

enter the array elements
12
16
25
32
58
Enter the element to be search
25
Element found 25 , position==2

14.Write a program for tic-tac-toe game.

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

char matrix[3][3];  /* the tic tac toe matrix */

char check(void);
void init_matrix(void);
void get_player_move(void);
void get_computer_move(void);
void disp_matrix(void);

int main(void)
{
  char done;

  printf("This is the game of Tic Tac Toe.\n");
  printf("You will be playing against the computer.\n");

  done =  ' ';
  init_matrix();

  do {
    disp_matrix();
    get_player_move();
    done = check(); /* see if winner */
    if(done!= ' ') break; /* winner!*/
    get_computer_move();
    done = check(); /* see if winner */
  } while(done== ' ');

  if(done=='X') printf("You won!\n");
  else printf("I won!!!!\n");
```

```
  disp_matrix(); /* show final positions */

  return 0;
}

/* Initialize the matrix. */
void init_matrix(void)
{
  int i, j;

  for(i=0; i<3; i++)
    for(j=0; j<3; j++) matrix[i][j] =  ' ';
}

/* Get a player's move. */
void get_player_move(void)
{
  int x, y;

  printf("Enter X,Y coordinates for your move: ");
  scanf("%d%*c%d", &x, &y);

  x--; y--;

  if(matrix[x][y]!= ' '){
    printf("Invalid move, try again.\n");
    get_player_move();
  }
  else matrix[x][y] = 'X';
}

/* Get a move from the computer. */
void get_computer_move(void)
{
```

```
  int i, j;
  for(i=0; i<3; i++){
    for(j=0; j<3; j++)
      if(matrix[i][j]==' ') break;
    if(matrix[i][j]==' ') break;
  }

  if(i*j==9)  {
    printf("draw\n");
    exit(0);
  }
  else
    matrix[i][j] = 'O';
}

/* Display the matrix on the screen. */
void disp_matrix(void)
{
  int t;

  for(t=0; t<3; t++) {
    printf(" %c | %c | %c ",matrix[t][0],
           matrix[t][1], matrix [t][2]);
    if(t!=2) printf("\n---|---|---\n");
  }
  printf("\n");
}

/* See if there is a winner. */
char check(void)
{
  int i;

  for(i=0; i<3; i++)  /* check rows */
```

```
    if(matrix[i][0]==matrix[i][1] &&
       matrix[i][0]==matrix[i][2]) return matrix[i][0];

  for(i=0; i<3; i++)  /* check columns */
    if(matrix[0][i]==matrix[1][i] &&
       matrix[0][i]==matrix[2][i]) return matrix[0][i];

  /* test diagonals */
  if(matrix[0][0]==matrix[1][1] &&
     matrix[1][1]==matrix[2][2])
       return matrix[0][0];

  if(matrix[0][2]==matrix[1][1] &&
     matrix[1][1]==matrix[2][0])
       return matrix[0][2];

  return ' ';
}
```

Output:

```
This is the game of Tic Tac Toe.
You will be playing against the computer.
     |   |
---|---|---
 |   |
---|---|---
|   |
Enter X,Y coordinates for your move:1,1
 X | O |
---|---|---
   |   |
---|---|---
   |   |
```

```
Enter X,Y coordinates for your move: 2,2
 X | O | O
---|---|---
   | X |
---|---|---
   |   |
Enter X,Y coordinates for your move:3,3


You won!
 X | O | O
---|---|---
   | X |
---|---|---
|   | X
```

15.Write a program to perform operations creation, insertion, deletion and traversing on a binary search tree.

```
#include<stdio.h>
#include<stdlib.h>
struct tree
{
    int info;
    struct tree *left;
    struct tree *right;
};

struct tree *insert(struct tree *,int);
void inorder(struct tree *);
void findroot(struct tree *);
struct tree *del(struct tree *,int);
int main(void)
{
   struct tree *root;
   int ch,item,item_no;
    root=NULL;
   do
   {
      do
      {
         printf("\n\n\t\t\tMENU");

printf("\n\t#######################################");
         printf("\n\n\t1. Binary Search Tree Insertion  ");
         printf("\n\t2. Binary Search Tree Deletion ");
         printf("\n\t3. Inorder Traversal of Binary Search Tree");

         printf("\n\t4. Find Root ");
         printf("\n\t5. Exit\n\n");
         printf("\nEnter the Choice : ");
```

```
            scanf("%d",&ch);
            if(ch<1 || ch>5)
            {
              printf("\nInvalid choice: Try again!!!");
            }
          }while (ch<1 || ch>5);
          switch(ch)
          {
            case 1:
              printf("\nEnter the New Element: ");
              scanf("%d",&item);
              root=insert(root,item);
            break;
            case 2:
              if(root!=NULL)
              {
                printf("\nEnter the Element to be Deleted: ");
                scanf("%d",&item_no);
                root=del(root,item_no);
                printf("The Binary Search Tree is now:\n");
                inorder(root);
              }
              else
              {
                printf("\nBinary Search Tree is Empty!!!\n");
              }
              break;
            case 3:
              if(root!=NULL)
              {
                printf("\nInorder Traversal of Binary Search Tree is:
");
                inorder(root);
              }
```

```
            else
            {
              printf("\nBinary Search Tree is Empty!!!\n");
            }
            break;

          case 4:
            findroot(root);
            break;
          default:
            printf("\nProgram Terminating\n\n");
      }
    }while(ch !=7);
    return(0);
}


struct tree *insert(struct tree *root, int x)
{
    if(!root)
    {
      root=(struct tree*)malloc(sizeof(struct tree));
      root->info=x;
      root->left=NULL;
      root->right=NULL;
      return(root);
    }
    if(root->info>x)
    {
      root->left=insert(root->left,x);
    }
    else
    {
      if(root->info<x)
```

```
        {
          root->right=insert(root->right,x);
        }
      }
      return(root);
}

void inorder(struct tree *root)
{
   if(root!=NULL)
   {
      inorder(root->left);
      printf(" %d",root->info);
      inorder(root->right);
   }
   return;
}


void findroot(struct tree *root)
{
    if(root!=NULL)
    {
       printf("\nRoot is %d\n",root->info);
    }
    else
    {
       printf("\nBinary Search Tree is Empty!!!\n");
    }
}

struct tree *del(struct tree *ptr,int x)
{
  struct tree *p1,*p2;
```

```
if(!ptr)
{
   printf("\nElement not Found!!!\n");
   return(ptr);
}
else
{
   if(ptr->info<x)
   {
      ptr->right=del(ptr->right,x);
   }
   else if(ptr->info>x)
   {
      ptr->left=del(ptr->left,x);
      return ptr;
   }
   else
   {
      if(ptr->info==x)
      {
         if(ptr->left==ptr->right)
         {
            free(ptr);
            return(NULL);
         }
         else if(ptr->left==NULL)
         {
            p1=ptr->right;
            free(ptr);
            return p1;
         }
         else if(ptr->right==NULL)
         {
            p1=ptr->left;
```

```
            free(ptr);
            return p1;
          }
          else
          {
            p1=ptr->right;
            p2=ptr->right;
            while(p1->left != NULL)
            {
               p1=p1->left;
            }
            p1->left=ptr->left;
            free(ptr);
            return p2;
          }
        }
      }
    }
    return(ptr);
}
```

Sample Output:

```
        MENU
##################################################
 1. Binary Search Tree Insertion
 2. Binary Search Tree Deletion
 3. Inorder Traversal of Binary search Tree
 4. Find Root
 5. Exit

Enter the Choice:1
Enter the New Element:34

        MENU
```

```
##############################################
 1. Binary Search Tree Insertion
 2. Binary Search Tree Deletion
 3. Inorder Traversal of Binary search Tree
 4. Find Root
 5. Exit

Enter the Choice:1
Enter the New Element:23

        MENU
##############################################
 1. Binary Search Tree Insertion
 2. Binary Search Tree Deletion
 3. Inorder Traversal of Binary search Tree
 4. Find Root
 5. Exit

Enter the Choice:1
Enter the New Element:67

        MENU
##############################################
 1. Binary Search Tree Insertion
 2. Binary Search Tree Deletion
 3. Inorder Traversal of Binary search Tree
 4. Find Root
 5. Exit

Enter the Choice:3
Inorder Traversal of Binary Search Tree is:23     34     67

        MENU
##############################################
 1. Binary Search Tree Insertion
 2. Binary Search Tree Deletion
 3. Inorder Traversal of Binary search Tree
 4. Find Root
 5. Exit
```

```
Enter the Choice:4
Root is      34

        MENU
##################################################
 1. Binary Search Tree Insertion
 2. Binary Search Tree Deletion
 3. Inorder Traversal of Binary search Tree
 4. Find Root
 5. Exit

Enter the Choice:2
Enter the element to be deleted:23
The Binary Search Tree is now:
34     67

        MENU
##################################################
 1. Binary Search Tree Insertion
 2. Binary Search Tree Deletion
 3. Inorder Traversal of Binary search Tree
 4. Find Root
 5. Exit

Enter the Choice:5
Program Terminating

Enter the New Element:34
```

16.Write a program to implement depth first search and breadth first search on graphs.

/* Implementation of DFS*/

```
#include<stdio.h>
#include<stdlib.h>
void dfsearch();
int cost[10][10],visited[10],visit[10],stack[10],top=-1,i,j,k,n,v;
void main()
{
        int m;
        printf("\nEnter number of vertices in the graph\n");
        scanf("%d",&n);
        printf("Enter no of edges\n");
        scanf("%d",&m);
        for(k=1;k<=m;k++)
        {
                printf("Edge %d between\t",k);
                scanf("%d%d",&i,&j);
                cost[i][j]=1;
        }

        dfsearch();
}

void dfsearch()
{
        printf("Enter initial vertex\n");
        scanf("%d",&v);
        printf("Order of visited vertices\n");
        printf("%d\t",v);
        visited[v]=1;
        k=1;
        while(k<n)
        {
                for(j=n;j>=1;j--)
                {
                        if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
```

```
                {
                        visit[j]=1;
                         stack[top]=j;
                        top++;
                 }
             }
            v=stack[--top];
             printf("%d\t",v);;
            k++;
            visit[v]=0;
        visited[v]=1;
        }
    }
```

Sample Output:

```
Enter number of vertices in the graph
4
Enter no of edges
3
edge 1 between 1   2
edge 2 between 1   3
edge 3 between 2   4
Enter initial vertex
1
Order of visited vertices
1       2       4       3
```

b**) Program for implementation of BFS.**

```
#include<stdio.h>
 void bfsearch();
 int
cost[10][10],i,j,k,m,n,queue[10],front,rear,visit[10],visited[10],v;
 void main()
{
        printf("Enter no. of vertices\n");
        scanf("%d",&n);
        printf("Enter no. of edges\n");
        scanf("%d",&m);
        printf("\n Edges\n");
        for(k=1;k<=m;k++)
        {
                printf("Edge %d between\t",k);
                scanf("%d%6d",&i,&j);
                cost[i][j]=1;
        }
                bfsearch();
  }
  void bfsearch()
  {
        printf("\n Enter initial vertex\n");
         scanf("%d",&v);
        printf("\n Order of the visited vertices\n");
        printf("%d\t",v);
        visited[v]=1;
         k=1;
  while(k<n)
  {
         for(j=1;j<=n;j++)
          if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
         {
                visit[j]=1;
                queue[rear++]=j;
         }
       v=queue[front++];
       printf("%d\t",v);
```

```
        k++;
        visit[v]=0;
        visited[v]=1;
    }
    }
```

Sample Output:

```
Enter number of vertices in the graph
4
Enter no of edges
3
edge 1 between 1   2
edge 2 between 1   3
edge 3 between 2   4
Enter initial vertex
1
Order of visited vertices
1       2       3       4
```

17.Write a program to perform different operations on Red Black trees.

```
#include <stdio.h>
 #include <stdlib.h>
 enum nodeColor {
      RED,
      BLACK
 };

 struct rbNode {
      int data, color;
      struct rbNode *link[2];
 };

 struct rbNode *root = NULL;

 struct rbNode * createNode(int data) {
      struct rbNode *newnode;
      newnode = (struct rbNode *)malloc(sizeof(struct rbNode));
      newnode->data = data;
      newnode->color = RED;
      newnode->link[0] = newnode->link[1] = NULL;
      return newnode;
 }

 void insertion (int data) {
      struct rbNode *stack[98], *ptr, *newnode, *xPtr, *yPtr;
      int dir[98], ht = 0, index;
      ptr = root;
      if (!root) {
            root = createNode(data);
            return;
      }
      stack[ht] = root;
      dir[ht++] = 0;
      /* find the place to insert the new node */
      while (ptr != NULL) {
            if (ptr->data == data) {
                  printf("Duplicates Not Allowed!!\n");
```

```
            return;
        }
        index = (data - ptr->data) > 0 ? 1 : 0;
        stack[ht] = ptr;
        ptr = ptr->link[index];
        dir[ht++] = index;
    }
    /* insert the new node */
    stack[ht - 1]->link[index] = newnode = createNode(data);
    while ((ht >= 3) && (stack[ht - 1]->color == RED)) {
        if (dir[ht - 2] == 0) {
            yPtr = stack[ht - 2]->link[1];
            if (yPtr != NULL && yPtr->color == RED) {
                /*
                 * Red node having red child. B- black, R-red
                 *     B               R
                 *    / \            /   \
                 *   R   R  =>     B     B
                 *  /              /
                 * R              R
                 */
                stack[ht - 2]->color = RED;
                stack[ht - 1]->color = yPtr->color = BLACK;
                ht = ht -2;
            } else {
                if (dir[ht - 1] == 0) {
                    yPtr = stack[ht - 1];
                } else {
                    /*
                     * XR - node X with red color
                     * YR - node Y with red color
                     * Red node having red child
                     *(do single rotation left b/w X and Y)
                     *         B             B
                     *        /             /
                     *      XR     =>      YR
                     *        \             /
                     *         YR          XR
                     * one more additional processing will be
                     * performed after this else part.  Since
```

```
                    * we have red node (YR) with red child(XR)
                    */
                   xPtr = stack[ht - 1];
                   yPtr = xPtr->link[1];
                   xPtr->link[1] = yPtr->link[0];
                   yPtr->link[0] = xPtr;
                   stack[ht - 2]->link[0] = yPtr;
              }
              /*
               *  Red node(YR) with red child (XR) - single
               *  rotation b/w YR and XR for height balance. Still,
               *  red node (YR) is having red child.  So, change the
               *  color of Y to black and Black child B to Red R
               *         B          YR          YB
               *        /          / \        / \
               *       YR  =>    XR   B  =>  XR  R
               *      /
               *     XR
               */
              xPtr = stack[ht - 2];
              xPtr->color = RED;
              yPtr->color = BLACK;
              xPtr->link[0] = yPtr->link[1];
              yPtr->link[1] = xPtr;
              if (xPtr == root) {
                   root = yPtr;
              } else {
                   stack[ht - 3]->link[dir[ht - 3]] = yPtr;
              }
              break;
         }
    } else {
         yPtr = stack[ht - 2]->link[0];
         if ((yPtr != NULL) && (yPtr->color == RED)) {
              /*
               * Red node with red child
               *       B            R
               *     /   \        /   \
               *    R     R =>  B     B
               *          \            \
```

```
                        *            R            R
                        *
                        */
                       stack[ht - 2]->color = RED;
                       stack[ht - 1]->color = yPtr->color = BLACK;
                       ht = ht - 2;
                   } else {
                       if (dir[ht - 1] == 1) {
                           yPtr = stack[ht - 1];
                       } else {
                           /*
                            * Red node(XR) with red child(YR)
                            *   B          B
                            *    \          \
                            *     XR  => YR
                            *    /           \
                            *   YR            XR
                            * Single rotation b/w XR(node x with red color) &
YR
                            */
                           xPtr = stack[ht - 1];
                           yPtr = xPtr->link[0];
                           xPtr->link[0] = yPtr->link[1];
                           yPtr->link[1] = xPtr;
                           stack[ht - 2]->link[1] = yPtr;
                       }
                       /*
                        *   B             YR         YB
                        *    \           /  \       /  \
                        *     YR  =>    B   XR => R    XR
                        *      \
                        *       XR
                        * Single rotation b/w YR and XR and change the color
to
                        * satisfy rebalance property.
                        */
                       xPtr = stack[ht - 2];
                       yPtr->color = BLACK;
                       xPtr->color = RED;
                       xPtr->link[1] = yPtr->link[0];
```

```
                        yPtr->link[0] = xPtr;
                        if (xPtr == root) {
                                root = yPtr;
                        } else {
                                stack[ht - 3]->link[dir[ht - 3]] = yPtr;
                        }
                        break;
                }
        }
    }
    root->color = BLACK;
}

void deletion(int data) {
    struct rbNode *stack[98], *ptr, *xPtr, *yPtr;
    struct rbNode *pPtr, *qPtr, *rPtr;
    int dir[98], ht = 0, diff, i;
    enum nodeColor color;

    if (!root) {
        printf("Tree not available\n");
        return;
    }

    ptr = root;
    /* search the node to delete */
    while (ptr != NULL) {
        if ((data - ptr->data) == 0)
            break;
        diff = (data - ptr->data) > 0 ? 1 : 0;
        stack[ht] = ptr;
        dir[ht++] = diff;
        ptr = ptr->link[diff];
    }

    if (ptr->link[1] == NULL) {
        /* node with no children */
        if ((ptr == root) && (ptr->link[0] == NULL)) {
            free(ptr);
            root = NULL;
```

```
        } else if (ptr == root) {
                /* deleting root - root with one child */
                root = ptr->link[0];
                free(ptr);
        } else {
                /* node with one child */
                stack[ht - 1]->link[dir[ht - 1]] = ptr->link[0];
        }
} else {
        xPtr = ptr->link[1];
        if (xPtr->link[0] == NULL) {
                /*
                 * node with 2 children - deleting node
                 * whose right child has no left child
                 */
                xPtr->link[0] = ptr->link[0];
                color = xPtr->color;
                xPtr->color = ptr->color;
                ptr->color = color;

                if (ptr == root) {
                        root = xPtr;
                } else {
                        stack[ht - 1]->link[dir[ht - 1]] = xPtr;
                }

                dir[ht] = 1;
                stack[ht++] = xPtr;
        } else {
                /* deleting node with 2 children */
                i = ht++;
                while (1) {
                        dir[ht] = 0;
                        stack[ht++] = xPtr;
                        yPtr = xPtr->link[0];
                        if (!yPtr->link[0])
                                break;
                        xPtr = yPtr;
                }
```

```
                dir[i] = 1;
                stack[i] = yPtr;
                if (i > 0)
                        stack[i - 1]->link[dir[i - 1]] = yPtr;

                yPtr->link[0] = ptr->link[0];
                xPtr->link[0] = yPtr->link[1];
                yPtr->link[1] = ptr->link[1];

                if (ptr == root) {
                        root = yPtr;
                }

                color = yPtr->color;
                yPtr->color = ptr->color;
                ptr->color = color;
        }
}
if (ht < 1)
        return;
if (ptr->color == BLACK) {
        while (1) {
                pPtr = stack[ht - 1]->link[dir[ht - 1]];
                if (pPtr && pPtr->color == RED) {
                        pPtr->color = BLACK;
                        break;
                }

                if (ht < 2)
                        break;

                if (dir[ht - 2] == 0) {
                        rPtr = stack[ht - 1]->link[1];

                        if (!rPtr)
                                break;

                        if (rPtr->color == RED) {
                                /*
                                 * incase if rPtr is red, we need
```

```
                * change it to black..
                *    aB               rPtr (red)  rPtr(black)
                *   / \      =>      /    \ =>    /   \
                *  ST  rPtr(red)  aB    cB      aR   cB
                *      / \       / \          / \
                *     bB  cB   ST  bB       ST  bB
                *  ST - subtree
                *  xB - node x with Black color
                *  xR - node x with Red color
                * the above operation will simply rebalace
                * operation in RB tree
                */
                stack[ht - 1]->color = RED;
                rPtr->color = BLACK;
                stack[ht - 1]->link[1] = rPtr->link[0];
                rPtr->link[0] = stack[ht - 1];

                if (stack[ht - 1] == root) {
                        root = rPtr;
                } else {
                        stack[ht - 2]->link[dir[ht - 2]] = rPtr;
                }
                dir[ht] = 0;
                stack[ht] = stack[ht - 1];
                stack[ht - 1] = rPtr;
                ht++;

                rPtr = stack[ht - 1]->link[1];
        }


        if ( (!rPtr->link[0] || rPtr->link[0]->color == BLACK)
&&
                (!rPtr->link[1] || rPtr->link[1]->color == BLACK))
{
                /*
                *      rPtr(black)          rPtr(Red)
                *     /   \          =>  /    \
                *    B      B            R      R
                *
```

```
                        */
                        rPtr->color = RED;
                    } else {
                        if (!rPtr->link[1] || rPtr->link[1]->color == BLACK)
{
                            /*
                             * Below is a subtree. rPtr with red left child
                             * single rotation right b/w yR and rPtr  &
                             * change the color as needed
                             *       wR                    wR
                             *      / \                  / \
                             *     xB   rPtr(Black) =>   xB  yB
                             *    / \   / \            / \ / \
                             *   a   b yR   e          a   b c   rPtr(Red)
                             *        / \                  / \
                             *       c   d                 d   e
                             */
                            qPtr = rPtr->link[0];
                            rPtr->color = RED;
                            qPtr->color = BLACK;
                            rPtr->link[0] = qPtr->link[1];
                            qPtr->link[1] = rPtr;
                            rPtr = stack[ht - 1]->link[1] = qPtr;
                        }
                        /*
                         * Below is a subtree. rPtr with Right red child
                         * single rotation b/w rPtr & wR and change colors
                         *      wR (stack[ht-1])      rPtr(Red)
                         *    /   \                  /   \
                         *   xB    rPtr(black)     wB     yB
                         *  / \   / \       =>    /   \   / \
                         * a   b c   yR          xB    c d    e
                         *           / \        / \
                         *          d   e      a   b
                         */
                        rPtr->color = stack[ht - 1]->color;
                        stack[ht - 1]->color = BLACK;
                        rPtr->link[1]->color = BLACK;
                        stack[ht - 1]->link[1] = rPtr->link[0];
                        rPtr->link[0] = stack[ht - 1];
```

```
                        if (stack[ht - 1] == root) {
                            root = rPtr;
                        } else {
                            stack[ht - 2]->link[dir[ht - 2]] = rPtr;
                        }
                        break;
                    }
                } else {
                    rPtr = stack[ht - 1]->link[0];
                    if (!rPtr)
                        break;

                    if (rPtr->color == RED) {
                        stack[ht - 1]->color = RED;
                        rPtr->color = BLACK;
                        stack[ht - 1]->link[0] = rPtr->link[1];
                        rPtr->link[1] = stack[ht - 1];

                        if (stack[ht - 1] == root) {
                            root = rPtr;
                        } else {
                            stack[ht - 2]->link[dir[ht - 2]] = rPtr;
                        }
                        dir[ht] = 1;
                        stack[ht] = stack[ht - 1];
                        stack[ht - 1] = rPtr;
                        ht++;

                        rPtr = stack[ht - 1]->link[0];
                    }
                    if ( (!rPtr->link[0] || rPtr->link[0]->color == BLACK)
&&
                        (!rPtr->link[1] || rPtr->link[1]->color == BLACK))
{
                        rPtr->color = RED;
                    } else {
                        if (!rPtr->link[0] || rPtr->link[0]->color == BLACK)
{
                            qPtr = rPtr->link[1];
                            rPtr->color = RED;
```

```
                        qPtr->color = BLACK;
                        rPtr->link[1] = qPtr->link[0];
                        qPtr->link[0] = rPtr;
                        rPtr = stack[ht - 1]->link[0] = qPtr;
                    }
                    rPtr->color = stack[ht - 1]->color;
                    stack[ht - 1]->color = BLACK;
                    rPtr->link[0]->color = BLACK;
                    stack[ht - 1]->link[0] = rPtr->link[1];
                    rPtr->link[1] = stack[ht - 1];
                    if (stack[ht - 1] == root) {
                        root = rPtr;
                    } else {
                        stack[ht - 2]->link[dir[ht - 2]] = rPtr;
                    }
                    break;
                }
            }
            ht--;
        }
    }
}

void searchElement(int data) {
    struct rbNode *temp = root;
    int diff;

    while (temp != NULL) {
        diff = data - temp->data;
        if (diff > 0) {
            temp = temp->link[1];
        } else if (diff < 0) {
            temp = temp->link[0];
        } else {
            printf("Search Element Found!!\n");
            return;
        }
    }
    printf("Given Data Not Found in RB Tree!!\n");
    return;
```

```
}

void inorderTraversal(struct rbNode *node) {
    if (node) {
        inorderTraversal(node->link[0]);
        printf("%d  ", node->data);
        inorderTraversal(node->link[1]);
    }
    return;
}

int main() {
    int ch, data;
    while (1) {
        printf("1. Insertion\t2. Deletion\n");
        printf("3. Searching\t4. Traverse\n");
        printf("5. Exit\nEnter your choice:");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("Enter the data to insert:");
                scanf("%d", &data);
                insertion(data);
                break;
            case 2:
                printf("Enter the data to delete:");
                scanf("%d", &data);
                deletion(data);
                break;
            case 3:
                printf("Enter the search element:");
                scanf("%d", &data);
                searchElement(data);
                break;
            case 4:
                inorderTraversal(root);
                printf("\n");
                break;
            case 5:
                exit(0);
```

```
                default:
                        printf("You have entered wrong option!!\n");
                        break;
                }
                printf("\n");
        }
        return 0;
 }
```

Sample Output:

```
1. Insertion    2.deletion     3. Searching 4. Traverse   5.Exit
Enter your choice:1
Enter the data to insert:42

1. Insertion    2.deletion     3. Searching 4. Traverse   5.Exit
Enter your choice:1
Enter the data to insert:27

1. Insertion    2.deletion     3. Searching 4. Traverse   5.Exit
Enter your choice:3
Enter the search element:89
Given Data Not Found in RB Tree !!

1. Insertion    2.deletion     3. Searching 4. Traverse   5.Exit
Enter your choice:4
27       42

1. Insertion    2.deletion     3. Searching 4. Traverse   5.Exit
Enter your choice:2
Enter the data to delete:27

1. Insertion    2.deletion     3. Searching 4. Traverse   5.Exit
Enter your choice:5
```

18.Write a program to implement external sorting.

```
#include<stdio.h>
void mergesort(int *,int,int);
void merge(int *,int,int,int);
int a[20],i,n,b[20];
void main()
{
        printf("Enter No.of elements\n");
        scanf("%d",&n);
        printf("Enter the elements\n");
       for(i=0;i<n;i++)
        scanf("%d",&a[i]);
        printf("\n Array before sorting\n");
        for(i=0;i<n;i++)
        printf("%d \t",a[i]);
        mergesort(a,0,n-1);
       printf("\n Array after sorting\n");
       for(i=0;i<n;i++)
      printf("%d \t",a[i]);
}
void mergesort(int a[],int i,int j)
{
        int mid;
       if(i<j)
       {
              mid=(i+j)/2;
              mergesort(a,i,mid);
             mergesort(a,mid+1,j);
            merge(a,i,mid,j);
       }
}
void merge(int a[],int low,int mid,int high)
{
        int h,i,j,k;
       h=low;
        i=low;
        j=mid+1;
       while(h<=mid && j<=high)
```

```
        {
                if(a[h]<=a[j])
                b[i]=a[h++];
         else
          b[i]=a[j++];
          i++;
}
  if(h>mid)
for(k=j;k<=high;k++)
     b[i++]=a[k];
else
  for(k=h;k<=mid;k++)
      b[i++]=a[k];
printf("\n");
for(k=low;k<=high;k++)
{
   a[k]=b[k];
   printf("%d\t",a[k]);
}
}
```

Sample Output:

```
Enter No. of elements
5
Enter the elements
6
23
77
43
19
Array before sorting
6       23    77    43    19
6       23
6       23    77
19     43
6       19    23    43    77
Array after sorting
6  19 23    43    77
```

19 .Write a program to perform different operations of B Tree.

```
#include<stdio.h>
#include<stdlib.h>
#define M 5

struct node{
int n; /* n < M No. of keys in node will always less than order
of B
tree */
int keys[M-1]; /*array of keys*/
struct node *p[M]; /* (n+1 pointers will be in use) */
}*root=NULL;

enumKeyStatus{Duplicate,SearchFailure,Success,InsertIt,Less
Keys };

void insert(int key);
void display(struct node *root,int);
void DelNode(int x);
void search(int x);
enum KeyStatus ins(struct node *r, int x, int* y, struct node**
u);
int searchPos(int x,int *key_arr, int n);
enum KeyStatus del(struct node *r, int x);

int main()
{
int key;
int choice;
printf("Creation of B tree for node %d\n",M);
while(1)
{
printf("1.Insert\n");
```

```
printf("2.Delete\n");
printf("3.Search\n");
printf("4.Display\n");
printf("5.Quit\n");
printf("Enter your choice : ");
scanf("%d",&choice);

switch(choice)
{
case 1:
printf("Enter the key : ");
scanf("%d",&key);
insert(key);
break;
case 2:
printf("Enter the key : ");
scanf("%d",&key);
DelNode(key);
break;
case 3:
printf("Enter the key : ");
scanf("%d",&key);
search(key);
break;
case 4:
printf("Btree is :\n");
display(root,0);
break;
case 5:
exit(1);
default:
printf("Wrong choice\n");
break;
}/*End of switch*/
```

```
}/*End of while*/
return 0;
}/*End of main()*/

void insert(int key)
{
struct node *newnode;
int upKey;
enum KeyStatus value;
value = ins(root, key, &upKey, &newnode);
if (value == Duplicate)
printf("Key already available\n");
if (value == InsertIt)
{
struct node *uproot = root;
root=malloc(sizeof(struct node));
root->n = 1;
root->keys[0] = upKey;
root->p[0] = uproot;
root->p[1] = newnode;
}/*End of if */
}/*End of insert()*/

enum KeyStatus ins(struct node *ptr, int key, int
*upKey,struct node **newnode)
{
struct node *newPtr, *lastPtr;
int pos, i, n,splitPos;
int newKey, lastKey;
enum KeyStatus value;
if (ptr == NULL)
{
*newnode = NULL;
*upKey = key;
```

```
return InsertIt;
}
n = ptr->n;
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos])
return Duplicate;
value = ins(ptr->p[pos], key, &newKey, &newPtr);
if (value != InsertIt)
return value;
/*If keys in node is less than M-1 where M is order of B tree*/
if (n < M - 1)
{
pos = searchPos(newKey, ptr->keys, n);
/*Shifting the key and pointer right for inserting the new key*/
for (i=n; i>pos; i--)
{
ptr->keys[i] = ptr->keys[i-1];
ptr->p[i+1] = ptr->p[i];
}
/*Key is inserted at exact location*/
ptr->keys[pos] = newKey;
ptr->p[pos+1] = newPtr;
++ptr->n; /*incrementing the number of keys in node*/
return Success;
}/*End of if */
/*If keys in nodes are maximum and position of node to be
inserted is
last*/
if (pos == M - 1)
{
lastKey = newKey;
lastPtr = newPtr;
}
```

```
else /*If keys in node are maximum and position of node to be inserted
is not last*/
{
lastKey = ptr->keys[M-2];
lastPtr = ptr->p[M-1];
for (i=M-2; i>pos; i--)
{
ptr->keys[i] = ptr->keys[i-1];
ptr->p[i+1] = ptr->p[i];
}
ptr->keys[pos] = newKey;
ptr->p[pos+1] = newPtr;
}
splitPos = (M - 1)/2;
(*upKey) = ptr->keys[splitPos];

(*newnode)=malloc(sizeof(struct node));/*Right node after split*/
ptr->n = splitPos; /*No. of keys for left splitted node*/
(*newnode)->n = M-1-splitPos;/*No. of keys for right splitted node*/
for (i=0; i < (*newnode)->n; i++)
{
(*newnode)->p[i] = ptr->p[i + splitPos + 1];
if(i < (*newnode)->n - 1)
(*newnode)->keys[i] = ptr->keys[i + splitPos + 1];
else
(*newnode)->keys[i] = lastKey;
}
(*newnode)->p[(*newnode)->n] = lastPtr;
return InsertIt;
}/*End of ins()*/
```

```
void display(struct node *ptr, int blanks)
{
if (ptr)
{
int i;
for(i=1;i<=blanks;i++)
printf(" ");
for (i=0; i < ptr->n; i++)
printf("%d ",ptr->keys[i]);
printf("\n");
for (i=0; i <= ptr->n; i++)
display(ptr->p[i], blanks+10);
}/*End of if*/
}/*End of display()*/

void search(int key)
{
int pos, i, n;
struct node *ptr = root;
printf("Search path:\n");
while (ptr)
{
n = ptr->n;
for (i=0; i < ptr->n; i++)
printf(" %d",ptr->keys[i]);
printf("\n");
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos])
{
printf("Key %d found in position %d of last dispalyed
node\n",key,i);
return;
}
ptr = ptr->p[pos];
```

```
}
printf("Key %d is not available\n",key);
}/*End of search()*/

int searchPos(int key, int *key_arr, int n)
{
int pos=0;
while (pos < n && key > key_arr[pos])
pos++;
return pos;
}/*End of searchPos()*/

void DelNode(int key)
{
struct node *uproot;
enum KeyStatus value;
value = del(root,key);
switch (value)
{
case SearchFailure:
printf("Key %d is not available\n",key);
break;
case LessKeys:
uproot = root;
root = root->p[0];
free(uproot);
break;
}/*End of switch*/
}/*End of delnode()*/

enum KeyStatus del(struct node *ptr, int key)
{
int pos, i, pivot, n ,min;
int *key_arr;
```

```
enum KeyStatus value;
struct node **p,*lptr,*rptr;

if (ptr == NULL)
return SearchFailure;
/*Assigns values of node*/
n=ptr->n;
key_arr = ptr->keys;
p = ptr->p;
min = (M - 1)/2;/*Minimum number of keys*/

pos = searchPos(key, key_arr, n);
if (p[0] == NULL)
{
if (pos == n || key < key_arr[pos])
return SearchFailure;
/*Shift keys and pointers left*/
for (i=pos+1; i < n; i++)
{
key_arr[i-1] = key_arr[i];
p[i] = p[i+1];
}
return --ptr->n >= (ptr==root ? 1 : min) ? Success : LessKeys;
}/*End of if */

if (pos < n && key == key_arr[pos])
{
struct node *qp = p[pos], *qp1;
int nkey;
while(1)
{
nkey = qp->n;
qp1 = qp->p[nkey];
if (qp1 == NULL)
```

```
break;
qp = qp1;
}/*End of while*/
key_arr[pos] = qp->keys[nkey-1];
qp->keys[nkey - 1] = key;
}/*End of if */
value = del(p[pos], key);
if (value != LessKeys)
return value;

if (pos > 0 && p[pos-1]->n > min)
{
pivot = pos - 1; /*pivot for left and right node*/
lptr = p[pivot];
rptr = p[pos];
/*Assigns values for right node*/
rptr->p[rptr->n + 1] = rptr->p[rptr->n];
for (i=rptr->n; i>0; i--)
{
rptr->keys[i] = rptr->keys[i-1];
rptr->p[i] = rptr->p[i-1];
}
rptr->n++;
rptr->keys[0] = key_arr[pivot];
rptr->p[0] = lptr->p[lptr->n];
key_arr[pivot] = lptr->keys[--lptr->n];
return Success;
}/*End of if */
if (p[pos]->n > min)
{
pivot = pos; /*pivot for left and right node*/
lptr = p[pivot];
rptr = p[pivot+1];
/*Assigns values for left node*/
```

```
lptr->keys[lptr->n] = key_arr[pivot];
lptr->p[lptr->n + 1] = rptr->p[0];
key_arr[pivot] = rptr->keys[0];
lptr->n++;
rptr->n--;
for (i=0; i < rptr->n; i++)
{
rptr->keys[i] = rptr->keys[i+1];
rptr->p[i] = rptr->p[i+1];
}/*End of for*/
rptr->p[rptr->n] = rptr->p[rptr->n + 1];
return Success;
}/*End of if */

if(pos == n)
pivot = pos-1;
else
pivot = pos;

lptr = p[pivot];
rptr = p[pivot+1];
/*merge right node with left node*/
lptr->keys[lptr->n] = key_arr[pivot];
lptr->p[lptr->n + 1] = rptr->p[0];
for (i=0; i < rptr->n; i++)
{
lptr->keys[lptr->n + 1 + i] = rptr->keys[i];
lptr->p[lptr->n + 2 + i] = rptr->p[i+1];
}
lptr->n = lptr->n + rptr->n +1;
free(rptr); /*Remove right node*/
for (i=pos+1; i < n; i++)
{
key_arr[i-1] = key_arr[i];
```

```
p[i] = p[i+1];
}
return --ptr->n >= (ptr == root ? 1 : min) ? Success : LessKeys;
}/*End of del()*/
```

Sample Output:

```
Creation of B tree for node 5
1. Insert
2. Delete
3. Search
4. Display
5. Quit

Enter your choice:1
Enter the key:43
1. Insert
2. Delete
3. Search
4. Display
5. Quit

Enter your choice:1
Enter the key:23
1. Insert
2. Delete
3. Search
4. Display
5. Quit
Enter your choice:4
Btree is:
23    43
1. Insert
2. Delete
3. Search
4. Display
5. Quit
Enter your choice:5
Quit the program
```