Sri Venkateswara
College of Engineering & Technology

Thirupachur, Thiruvallur Taluk & District
Pin code – 631203. Ph.: 27664444, 27665566

REGISTER NO. :

CERTIFICATE

Certified that this is a Bonafide record of the practical work done by Mr. /Ms.....of B.E / B.Tech / M.C.A / M.B.A..... in the ..... laboratory during the academic year 201 - 201

Staff – incharge

Head of the Department

Submitted for the University Practical Examination held on.....

Internal Examiner

External Examiner

INDEX

Sl.No.	Date	Name of the Experiment	Page No.	TEACHERS INITIAL

INDEX

Sl.No.	Date	Name of the Experiment	Page No.	TEACHERS INITIAL

Ex. No: 1.a. **ARRAY IMPLEMENTATION OF LIST ADT**

Date:

AIM:

To implement the List Abstract Data Type (ADT) for performing creation, insertion, deletion, searching and displaying the list using arrays.

ALGORITHM:

Step 1: Start.
Step 2: Declare the necessary variables, conditions and functions for implementation.
Step 3: Get the input from the user and store it an array 'a'. Then obtain the choice of operation from the user using switch case.
Step 4a: In creation, obtain the number of nodes and elements from the user
Step 4b: In Insertion, half of the elements are shifted upwards and in deletion half of the elements are shifted downwards after the user specifies the position.
Step 4c: In searching, value of the element given by the user is compared with the list and prints it if the value is in list
Step 5: Display the output using an array.
Step 6: Stop.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define MAX 50
void creation();
void insertion();
void deletion();
void  search();
void display();
int ch,a[20],n,pos,i,insert,s;
char g;
// ch for choice,g for getting, a for array, n for node
// pos for position, i for temporary variable, insert for insertion, s for search
void main()
{
clrscr();
do
{
printf("\n main menu");
printf("\n 1.creation \n 2.insertion \n 3.deletion \n 4.search \n 5.display \n 6.exit");
printf("\n enter your choice");
scanf("%d",&ch);
switch(ch)
{
case 1:
creation();
break;
case 2:
insertion();
break;
```

```
case 3:
deletion();
break;
case 4:
search();
break;
case 5:
display();
break;
case 6:
exit();
break;
default:
printf("\n enter the correct choice");
}
printf("\n do you want to continue");
scanf("\n %c",&g);
}
while(g=='y' || g=='Y');
getch();
}
void creation()
{
printf("\n enter the number of node:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n enter the element:",i+1);
scanf("%d",&a[i]);
}
}
void insertion()
{
printf("\n enter the position to insert:");
scanf("%d",&pos);
if(pos>=n)
{
printf("\n invalid location");
}
else
{
for(i=MAX-1;i>=pos-1;i--)
{
a[i+1]=a[i];
}
printf("\n enter the element to insert:");
```

```
scanf("%d",&insert);
a[pos]=insert;
n++;
}
printf("\n the list after insertion");
display();
}
void deletion()
{
printf("\n enter the position you want to delete:");
scanf("%d",&pos);
if(pos>=n)
{
printf("\n invalid location");
}
else
{
for(i=pos+1;i<n;i++)
{
a[i-1]=a[i];
}
n--;
}
printf("\n the elements after deletion");
for(i=0;i<n;i++)
{
printf("\t %d",a[i]);
}
}
void search()
{
printf("\n enter the elements to be searched:");
scanf("%d",&s);
for(i=0;i<n;i++)
{
if(a[i]==s)
{
printf("\n value is in position: %d",i);
}
}
}
 void display()
 {
 printf("\n elements of the list ADT are:");
 for(i=0;i<n;i++)
 {
```

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

OUTPUT:

```
Main menu
1. creation
2.insertion
3.deletion
4.search
5.display
6.exit
Enter your choice: 1
Enter the number of nodes: 5
Enter the element: 11
Enter the element: 22
Enter the element: 33
Enter the element: 44
Enter the element: 55
Do you want to continue: y
Main menu
1. creation
2.insertion
3.deletion
4.search
5.display
6.exit
Enter your choice: 5
Elements of the list ADT are
11
22
33
44
55
Do you want to continue: y
Main menu
1. creation
2.insertion
3.deletion
4.search
5.display
6.exit
Enter your choice: 2
Enter the position to insert: 3
Enter the element to insert: 100
The list after insertion
Elements of the list ADT are:
```

```
11
22
33
100
44
55
Do you want to continue: y
Main menu
1. creation
2.insertion
3.deletion
4.search
5.display
6.exit
Enter your choice: 3
Enter the position you want to delete: 1
The elements after deletion: 1
The elements after deletion are 11 33 100 44 55
Do you want to continue: y
Main menu
1. creation
2.insertion
3.deletion
4.search
5.display
6.exit
Enter your choice:4
Enter the element to be searched: 100
Value is in position: 2
Do you want to continue: n
Exit
```

RESULT:

Thus, a C program for list ADT using arrays was implemented successfully.

Ex. No: 1.b. LINKED LIST IMPLEMENTATION OF LIST ADT (SINGLY)
Date:

AIM:

To write a C program to implement linked list (singly) using list ADT (Abstract Data Type)

ALGORITHM:

Step 1: Start

Step 2: Creation: Get the number of elements, and create the nodes having structures DATA, LINK and store the element in Data field, link them together to form a linked list.

Step 3: Insertion: Get the number to be inserted and create a new node store the value in DATA field. And insert the node in the required position.

Step 4: Deletion: Get the number to be deleted. Search the list from the beginning and locate the node then delete the node.

Step 5: Display: Display all the nodes in the list.

Step 6: Stop.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define NULL 0
typedef struct list
{
int no;
struct list *next;
}LIST;
LIST *p,*t,*h,*pt;
// h for head and t for tail
void create(void);
void insert(void);
void delet(void);
void display(void);
int i=1,j,k=1,n,opt,pos,count;
//  n for nodes, opt for option, pos for position,
// i,j,k for temporary variables
void main()
{
clrscr();
p=NULL;
```

```
printf("\n size of the list and its operations is:");
printf("%d",sizeof(LIST));
printf("\n enter the number of nodes:\n");
scanf("%d",&n);
count=n;
while(i<=n)
{
create();
i++;
}
printf("\n 1.insert \n 2.delete \n 3.display \n 4.exit");
do
{
printf("\n enter your option:");
scanf("%d",&opt);
switch(opt)
{
case 1:
insert();
count++;
break;
case 2:
delet();
count--;
if(count==0)
{
printf("\n list is empty ");
}
break;
case 3:
printf("\n list elements are:\n");
display();
break;
}
}while(opt!=4);
getch();
}
void create()
{
if(p==NULL)
{
p=(LIST*)malloc(sizeof(LIST));
printf("\n enter the element");
scanf("%d",&p->no);
p->next=NULL;
h=p;
```

```
}
else
{
t=(LIST*)malloc(sizeof(LIST));
printf("\n enter the element");
scanf("%d",&t->no);
t->next=NULL;
p->next=t;
p=t;
}
printf("\n new element created");
}
void insert()
{
t=h;
p= (LIST*)malloc(sizeof(LIST));
printf("\n enter the element to be inserted:");
scanf("%d",&p->no);
printf("\n enter the position to insert:");
scanf("%d",&pos);
if(pos==1)
{
h=p;
h->next=t;
}
else
{
for(j=1;j<(pos-1);j++)
t=t->next;
p->next=t->next;
t->next=p;
t=p;
}
printf("\n new elements inserted");
}
void delet()
{
printf("\n enter the position to delete:");
scanf("%d",&pos);
if(pos==1)
{
h=h->next;
}
else
{
t=h;
```

```
for(j=1;j<(pos-1);j++)
t=t->next;
pt=t->next->next;
free(t->next);
t->next=pt;
}
printf("\n element deleted");
}
void display()
{
t=h;
while(t->next!=NULL)
{
printf("\t %d",t->no);
t=t->next;
}
printf("\t %d \t",t->no);
printf("\n All the elements are displayed");
}
```

OUTPUT:

```
Size of the list and its operations: 4
Enter the number of nodes: 5
Enter the element: 11
New element created
Enter the element: 22
New element created
Enter the element: 33
New element created
Enter the element: 44
New element created
Enter the element: 55
New element created
1.insert
2.delete
3.display
4.exit
Enter your option: 3
List elements are:
11      22      33      44      55
All the elements are displayed
Enter your option: 1
Enter the element to be inserted: 100
Enter the position to insert: 3
New element inserted
Enter your option: 3
List elements are:
```

```
11      22      100     33      44      55
All the elements are displayed
Enter your option: 2
Enter the position to delete: 4
Element deleted
Enter your option: 3
List elements are:
11      22      100     44      55
All the elements are displayed
Enter your option: 4
(program exits)
```

RESULT:

Thus, a C program for (Singly) Linked List implementation of list ADT was implemented successfully.

EX.NO: 1.c. POLYNOMIAL ADDITION USING LINKED LIST ADT
Date:

Aim:

To write a C program to implement the application of list ADT for polynomial addition using linked list.

Algorithm:

Step 1: Build a polynomial by entering it term by term.

Step 2: The function must first check to see if there is a term with that exponent in the polynomial

Step 3: If there is not the function will insert the term. If there is, the function will first remove the existing term and then insert the new term.

Step 4: Add the two polynomials to produce a new polynomial and return their sum.

Step 5: Print out the three polynomials in the form.

Term1+term2+……+term N +term 1+term 2+……+term N Term 1+term 2+ .....+term N

Step 6: Evaluate the polynomial that was the result of 2 given a value form the user.

Step 7: Clear the polynomials and start again.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<alloc.h>
#include<stdlib.h>
#define TRUE 1
#define FALSE 0
typedef struct pnode
{
float coef;
int exp;
struct pnode *next;
}p;
void main()
{
p *p1,*p2,*p3;
p *getpoly(), *add(p*,p*);
void display(p*);
clrscr();
printf("\n polynomial addition \n");
printf("\n enter the first polynomial\n");
p1=getpoly();
printf("\n enter the second polynomial\n");
```

```
p2=getpoly();
printf("\n the first polynomial is:\t\n");
display(p1);
printf("\n the second polynomial is :\t\n");
display(p2);
p3=add(p1,p2);
printf("\n the addition of polynomial is:\n ---------- \n");
display(p3);
exit(0);
}
p *getpoly()
{
p *temp, *neww, *last;
int exp,flag;
float coef;
p *getnode();
char ans='y';
temp=NULL;
flag=TRUE;
printf("\n enter the polynomial in descending order.. ");
do
{
printf("\n enter coefficient and exponent of term:\n");
scanf("%f\t %d", &coef,&exp);
neww=getnode();
if(neww==NULL)
printf("\n memory cannot be allocated..");
neww->coef=coef;
neww->exp=exp;
if(flag==1)
{
temp=neww;
last=temp;
flag=FALSE;
}
else
{
last->next=neww;
last=neww;
}
printf("\n do you want to add more terms(y/n)? ....\n");
ans=getch();
}
while(ans=='y');
return(temp);
}
```

```
p *getnode()
{
p *temp;
temp=(p*)malloc(sizeof(p));
temp->next=NULL;
return(temp);
}
void display(p *head)
{
p *temp;
temp=head;
if(temp==NULL)
{
printf("\n polynomial is empty\n");
getch();
return;
}
while(temp->next!=NULL)
{
printf("%0.1fx^%d+",temp->coef,temp->exp);
temp=temp->next;
}
printf("%0.1fx^%d+",temp->coef,temp->exp);
getch();
}
p *add(p* first, p* second)
{
p *p1,*p2,*temp,*dummy;
char ch;
float coef;
p *append(int,float,p*);
p1=first;
p2=second;
temp=(p*)malloc(sizeof(p));
if(temp==NULL)
printf("\n memory cannot be allocated\n");
dummy=temp;
while(p1!=NULL && p2!=NULL)
{
if(p1->exp==p2->exp)
{
coef=p1->coef+p2->coef;
temp=append(p1->exp,coef,temp);
p1=p1->next;
p2=p2->next;
}
```

```
else if(p1->exp<p2->exp)
{
coef=p2->coef;
temp=append(p2->exp,coef,temp);
p2=p2->next;
}
else if(p1->exp>p2->exp)
{
coef=p1->coef;
temp=append(p1->exp,coef,temp);
p1=p1->next;
}
}
while(p1!=NULL)
{
temp=append(p1->exp,p1->coef,temp);
p1=p1->next;
}
while(p2!=NULL)
{
temp=append(p2->exp,p2->coef,temp);
p2=p2->next;
}
temp->next=NULL;
temp=dummy->next;
free(dummy);
return(temp);
}
p *append(int exp, float coef,p *temp)
{
p *neww,*dummy;
neww=(p*)malloc(sizeof(p));
if(neww==NULL)
printf("\n memory cannot be allocated");
neww->exp=exp;
neww->coef=coef;
neww->next=NULL;
dummy=temp;
dummy->next=neww;
dummy=neww;
return(dummy);
}
```

OUTPUT:

```
Polynomial addition
Enter the first polynomial
Enter the polynomial in descending order
```

```
Enter coefficient and exponent of term:
4
3
Do you want to add more terms(y/n)?….y
Enter coefficient and exponent of term:
3
2
Do you want to add more terms(y/n)?….y
Enter coefficient and exponent of term:
2
1
Do you want to add more terms(y/n)?….n
Enter the second polynomial
Enter the polynomial in descending order
Enter coefficient and exponent of term:
5
3
Do you want to add more terms(y/n)?….y
Enter coefficient and exponent of term:
4
2
Do you want to add more terms(y/n)?....
The first polynomial is…
4.0x^3+3.0x^2+2.0x^1+
The second polynomial is…
5.0x^3+4.0x^2+
The addition of polynomials are
-------------------------------------------------------------------------------
9.0x^3+7.0x^2+2.0x^1+
```

RESULT:

Thus, a C program to implement the application of list ADT for polynomial addition was implemented successfully.

Ex. No: 2.a. **ARRAY IMPLEMENTATION OF STACK ADT**
Date:

AIM:
To write a C program to implement Stack operations such as push, pop and display using array.
ALGORITHM:
Step 1: Start.
Step 2: Initially top = -1;
Step 3: push operation increases top by one and writes pushed element to storage[top];
Step 4: pop operation checks that top is not equal to -1 and decreases top variable by 1;
Step 5a: display operation checks that top is not equal to -1 and returns storage[top];
Step 5b: If stack is full or empty, it is displayed when we push or pop.
Step 6: Stop.
PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define size 5
int  item;
int s[10];
int top;

void push()
{
if(top==size-1)
{
printf("\n stack is full");
return;
}
printf("\n enter item:");
scanf("%d",&item);
s[++top]=item;
}
void pop()
{
if(top==-1)
{
printf("\n stack is empty");
return;
}
printf("\n deleted item is:%d",s[top]);
top--;
}
void display()
{
int i;
if(top==1)
```

```
{
printf("\n stack is empty");
return;
}
printf("\n content of the stack is:");
for(i=0;i<=top;i++)
printf("%d\t",s[i]);
}
void main()
{
int ch;
top=-1;
clrscr();
printf("\n MENU \n 1.push \t 2.pop \t 3.display \t 4.exit \n");
do
{
printf("\n enter your choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
push();
break;
case 2:
pop();
break;
case 3:
display();
break;
case 4:
exit(0);
default:
printf("\n wrong entry... try again");
}
}while(ch<=4);
getch();
}
```

OUTPUT:

```
MENU
1. push          2.pop          3.display       `4.exit
Enter your choice: 1
Enter item:10
Enter your choice: 1
Enter item:20
```

```
Enter your choice: 1
Enter item:30
Enter your choice: 1
Enter item:40
Enter your choice: 1
Enter item:50
Enter your choice: 1
Stack is full
Enter your choice: 3
Content of the stack is : 10     20      30      40      50
Enter your choice: 2
Deleted item is: 50
Enter your choice: 2
Deleted item is: 40
Enter your choice: 3
Content of the stack is : 10     20      30
Enter your choice: 2
Deleted item is: 30
Enter your choice: 2
Deleted item is: 20
Enter your choice: 2
Deleted item is: 10
Enter your choice: 2
Stack is empty
Enter your choice: 4
(program exits)
```

RESULT:

Thus a C program for Stack operations such as push, pop and display using array was implemented successfully.

Ex.No: 2.b. IMPLEMENTATION OF STACK USING LINKED LIST
Date:

AIM:
To write a C program to implement stack ADT for performing push, pop and display operations using linked list.

ALGORITHM:
Step 1: Get the elements.
Step 2: Push the elements in to the stack by adding new element in the stack.
Step 3: Check whether stack is full or not.
Step 4: Pop elements from the stack by deleting element in the top.
Step 5: Check whether stack is empty or not.
Step 6: Display the top of the element in the stack using Top.
Step 7: Display all the elements in the list.
Step 8: Stop

PROGRAM:

```
#include <stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<alloc.h>
void push(int value);
void pop();
void  display();
int choice,data;
struct node
{
int data;
struct node *link;
};
struct node *top=NULL, *temp;
void main()
{
clrscr();
printf("\n MENU \n 1.push \n 2.pop \n 3.display");
do
{
printf("\n enter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("\n enter new element:");
scanf("%d",&data);
push(data);
printf("\n new element pushed in stack \n");
break;
case 2:
```

```
pop();
printf("\n element popped from stack \n");
break;
case 3:
printf("\n elements of stack are...");
display();
break;
default:
printf("\n invalid choice!!!");
exit(0);
}
}
while(choice<4);
getch();
}
void display()
{
temp=top;
if(temp==NULL)
{
printf("\n stack is empty");
}
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->link;
}
}
void push(int data)
{
temp=(struct node*)malloc(sizeof(struct node));
temp->data=data;
temp->link=top;
top=temp;
}
void pop()
{
if(top!=NULL)
{
printf("\n the popped element is %d",top->data);
top=top->link;
}
else
{
printf("\n stack underflow");
}}
```

OUTPUT:

```
MENU
1. push
2. pop
3. display
Enter your choice: 1
Enter new element: 10
New element inserted in stack
Enter your choice: 1
Enter new element: 20
New element inserted in stack
Enter your choice: 1
Enter new element: 30
New element inserted in stack
Enter your choice: 3
Elements of the stack are….
30->20->10
Enter your choice: 2
The popped element is 30
Element popped from stack
Enter your choice: 3
Elements of the stack are….
20->10
Enter your choice: 2
The popped element is 20
Element popped from stack
Enter your choice: 2
The popped element is 10
Element popped from stack


Enter your choice: 3
Stack is empty
Enter your choice: 4
(program exits)
```

RESULT:

Thus a C program for Stack and its operations such as push, pop and display using linked list was implemented successfully.

Ex.No: 2.c. **CONVERSION OF INFIX TO POSTFIX EXPRESSION**
Date:

AIM:

To write a program for Conversion of infix expression to postfix expression using stack

ALGORITHM:

Step 1: Get the expression as input.
Step 2: Check whether the input is an alphabet means, get the value for the alphabet.
Step 3: if it is alphabet means get the value for the alphabet.
Step 4: Now push that value into the stack.
Step 5: if it is operator means get the value for the operator.
Step 6: Now pop two values from the stack.
Step 7: perform that respective operation.
Step 8: push the result in the stack again.
Step 9: Repeat the steps until all the characters in the string is read.
Step 10: pop the result from the stack and display it.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
char inf[40],post[40];
int top=0,st[20];
void postfix();
void push(int);
char pop();
void main()
{
clrscr();
printf("\n INFIX TO POSTFIX CONVERSION");
printf("\n enter the infix expression:");
scanf("%s",&inf);
postfix();
getch();
}
void postfix()
{
int i,j=0;
for(i=0;inf[i]!='\0';i++)
{
switch(inf[i])
{
case '+':
while(st[top]>=1)
post[j++]=pop();
push(1);
```

```
break;
case '-':
while(st[top]>=1)
post[j++]=pop();
push(2);
break;
case '*':
while(st[top]>=3)
post[j++]=pop();
push(3);
break;
case '/':
while(st[top]>=3)
post[j++]=pop();
push(4);
break;
case '^':
while(st[top]>=4)
post[j++]=pop();
push(5);
break;
case '(':
push(0);
break;
case ')':
while(st[top]!=0)
post[j++]=pop();
top--;
break;
default:
post[j++]=inf[i];
}
}
while(top>0)
post[j++]=pop();
printf("\n postfix expression is:%s",post);
}
void push( int ele)
{
top++;
st[top]=ele;
}
char pop()
{
int el;
char e;
```

```
el=st[top];
top--;
switch(el)
{
case 1:
e='+';
break;
case 2:
e='-';
break;
case 3:
e='*';
break;
case 4:
e='/';
break;
case 5:
e='^';
break;
}
return(e);
}
```

OUTPUT:

```
INFIX TO POSTFIX CONVERSION
Enter the infix expression: (a+b)*(b-c^e)/d
Postfix expression is: ab+bce^-*d/
```

RESULT:

Thus a C program for application of converting infix to postfix expression using stack is implemented successfully

Ex. No: 3.a. **ARRAY IMPLEMENTATION OF QUEUE ADT**
Date:

AIM:

To implement the Queue ADT for performing enqueue (insertion), dequeue (deletion) and displaying the queue using arrays.

ALGORITHM:

Step 1: Start.
Step 2: Initialize front as f=0; rear as r=0.
Step 3: Enqueue operation moves a rear by one position and inserts a element at the rear in array.
Step 4: Dequeue operation deletes the element at the front of the queue and moves the front by one position in array
Step 5: Display operation displays the entire element in the queue.
Step 6: Stop.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10
void main()
 {
 int que[SIZE],opt,f=0,r=0,i,j=1,x=SIZE;
 clrscr();
 printf("\n array implementation of queue");
 printf("\n MENU \n 1.Insert \n 2.delete\n 3.display\n 4.exit");
 while(opt) {
 printf("\n enter your option:");
 scanf("%d",&opt);
 switch(opt) {
 case 1:
 if(r==x)
 printf("\n queue is full");
 else{
 printf("\n enter the element to be inserted at %d position: ", j++);
 scanf("%d",&que[r++]);
 }
 break;
 case 2:
 if(f==r){
 printf("\n queue is empty");
 }
 else{
 printf("\n deleted element is :%d",que[f++]);
 x++;
 }
 break;
 case 3:
 printf("\n Displaying the elements of queue...");
if(f==r){
```

```
printf("\n queue is empty");
}
else  {
for(i=f;i<r;i++) {
printf("%d\t",que[i]);
}
break;
case 4:
printf("\n terminating the program");
exit(0);
default:
printf("\n invalid option..try again...");
}}}
getch();
}
```

OUTPUT:

Array implementation of queue
MENU
1.Insert
2. Delete
3.Display
4.Exit
Enter your option: 1
Enter the element to be inserted at 1 position: 100
Enter your option: 1
Enter the element to be inserted at 1 position: 200
Enter your option: 1
Enter the element to be inserted at 1 position: 300
Enter your option: 1
Enter the element to be inserted at 1 position: 400
Enter your option: 1
Enter the element to be inserted at 1 position: 500
Enter your option: 3
Displaying the elements of queue… 100 200 300 400 500
Enter your option: 2
Deleted element is: 100
Enter your option: 2
Deleted element is: 200
Enter your option: 3
Displaying the elements of queue… 300 400 500
Enter your option: 4
(program exits)

RESULT:

Thus, Queue ADT for performing enqueue (insertion), dequeue (deletion) and displaying the queue using arrays is implemented successfully

Ex. No: 3.b. **LINKED LIST IMPLEMENTATION OF QUEUE ADT**
Date:

AIM:

To implement the Queue ADT for performing enqueue (insertion), dequeue (deletion) and displaying the queue using Linked List.

ALGORITHM:

Step 1: Start.
Step 2: Initialize front as f=0; rear as r=0.
Step 3: Enqueue operation moves a rear by one position and inserts a element at the rear in list.
Step 4: Dequeue operation deletes the element at the front of the queue and moves the front by one position in list
Step 5: Display operation displays the entire element in the queue using list
Step 6: Stop.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct node {
int no;
struct node *link;
}*f=0,*r=0;
void insert();
void delet();
void display();
int item;
void main()
{
int opt;
clrscr();
printf("\n LINKED LIST IMPLEMENTATION OF QUEUE");
printf("\n MENU \n 1.Enqueue(insert) \n 2.Dequeue(delete) \n 3.Display \n 4.exit");
while(opt) {
printf("\n enter your choice:");
scanf("%d",&opt);
switch(opt) {
case 1: insert();
break;
case 2: delet();
break;
case 3: display();
break;
case 4: exit(0);
default:
```

```
printf("\n invalid choice:try again");
```

```
} }
getch();
}
void insert()
{
printf("\n enter item:");
scanf("%d",&item);
if(r==0)
{
r=(struct node*)malloc(sizeof(struct node));
r->no=item;
r->link=0;
f=r;
}
else
{
r->link=(struct node*)malloc(sizeof(struct node));
r=r->link;
r->no=item;
r->link=0;
} }
void delet()
{
struct node *ptr;
if(f==0)
printf("\n queue is empty");
else
{
ptr=f;
item=f->no;
f=f->link;
free(ptr);
printf("\n item deleted is:%d",item);
if(f==0)
r=0;
} }
void display(){
struct node *ptr=f;
if(r==0)
printf("\n queue empty");
```

```
else {
printf("\n the elements in the queue are:\n");
while(ptr!=0) {
printf("%d\t",ptr->no);
ptr=ptr->link;
} } }
```

OUTPUT:

```
LINKED LIST IMPLEMENTATION OF QUEUE
MENU
1.Enqueue(Insert)
2.Dequeue (Delete)
3.Display
4.Exit
enter your choice: 2
Queue is empty
enter your choice: 3
Queue is empty
enter your choice: 1
enter item : 100
enter your choice: 1
enter item : 200
enter your choice: 1
enter item : 300
enter your choice: 3
the elements in the queue are: 100 200 300
enter your choice: 2
item deleted is: 100
enter your choice: 2
item deleted is: 200
enter your choice: 3
the elements in the queue are: 300
enter your choice: 2
item deleted is: 300
enter your choice: 3
queue is empty
enter your choice: 4
(program exits)
```

RESULT:

Thus, Queue ADT for performing enqueue (insertion), dequeue (deletion) and displaying the queue using linked list is implemented successfully

Ex. No: 3.c. **IMPLEMENTATION OF PRIORITY QUEUE**

Date:

AIM:

To write a C program for implementing the priority queue using Queue ADT

ALGORITHM:

Step 1: Define the size of queue and declare the variables top, bottom and priority
Step 2.a: Obtain the choice from user to push, pop, display or exit
Step 2.b: Before inserting the elements, check whether the queue is full.
Step 2.c: If queue is full, display error message.
Step 3: If not, after insertion, the elements are ordered based on their priority
Step 4.a: Delete the elements with the lower priority
Step 4.b: Before deletion, check whether the queue is empty.
Step 4.c: If queue is empty, display the error message.
Step 5: After deletion, remaining part of the queue will be zero
Step 6: While displaying the elements in the queue, the elements are printed on the higher order of priority and with empty positions as zero.
Step 7: Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define size 5
int queue[5][2]={0};
int top=-1;
int bottom;
void push(int value,int pr)
{
int i,j,k;
if(top<size-1) {
if(queue[top][1]>pr) {
for(i=0;i<top;i++) {
if(queue[i][1]>pr){
break;
} }
for(j=top;j>=i;j--) {
queue[j+1][0]=queue[j][0];
queue[j+1][1]=queue[j][1];
}
top++;
queue[1][0]=value;
queue[i][1]=pr;
}
else{
top++;
queue[top][0]=value;
queue[top][1]=pr;
}
```

```
}
else{
printf("\n queue overflow");
}}
void pop()
{
int i;
if(queue[0][0]==0){
printf("\n queueis empty");
}
else{
printf("\n deleted value is:%d",queue[0][0]);
for(i=0;i<top;i++)
{
queue[i][0]=queue[i+1][0];
queue[i][1]=queue[i+1][1];
}
queue[top][0]=0;
queue[top][1]=0;
top--;
}}
void display()
{
int i,j;
printf("\n element \t priority \n");
for(i=size-1;i>=0;i--)
{
for(j=0;j<2;j++)
{
printf("%d\t",queue[i][j]);
}}}
int main()
{
int i,j,ch=0,value=0,pr=0;
clrscr();
printf("\n MENU \n 1.enqueue \n 2.dequeue \n 3.display \n 4.exit");
while(1)
{
printf("\n enter the choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\n enter the number to be inserted:");
scanf("%d",&value);
printf("\n enter the priority:");
```

```
scanf("%d",&pr);
push(value,pr);
break;
case 2: pop();
break;
case 3: display();
break;
case 4: exit(0);
default:
printf("\n enter the correct choice... you entered wrongly");
}}}
```

OUTPUT:

```
MENU
1.enqueue
2.dequeue
3.display
4.exit
Enter the choice: 1
Enter the number to be inserted: 100
Enter the priority: 1
Enter the choice: 1
Enter the number to be inserted: 500
Enter the priority: 3
Enter the choice: 1
Enter the number to be inserted: 300
Enter the priority: 4
Enter the choice: 1
Enter the number to be inserted: 200
Enter the priority: 2
Enter the choice: 1
Enter the number to be inserted: 400
Enter the priority: 5
Enter the choice: 3
Element priority 400  5        300    4        500    3        200    2        100    1
Enter the choice: 2
Deleted value is 100
Enter the choice: 2
Deleted value is 200
Enter the choice: 3
Element priority 0    0        0      0        400    5        300    4        500    3
Enter the choice: 4
(program exits)
```

RESULT:

Thus the C program for the implementation of priority queue based on Queue Application is completed successfully.

Ex. No: 3.d. **ARRAY IMPLEMENTATION OF CIRCULAR QUEUE**

Date:

AIM:

To write a C program for implementing the circular queue using array.

ALGORITHM:

Step 1: Start.
Step 2: Define the size of circular queue and Initialize front as -1; rear as -1.
Step 3: Obtain the choice from user to insert, delete, display or exit
Step 4.a: Before inserting the elements, check whether the queue is full.
Step 4.b: If queue is full, there will be a overflow error.
Step 4.c: If not, insert the element into the queue.
Step 5.a: Before deletion, check whether the queue is empty.
Step 5.b: If queue is empty, display the underflow error message.
Step 5.c: Deletion operation deletes the element at the front of the queue and moves the front by one position in array
Step 6: Display operation displays the entire element in the queue.
Step 7: Stop.

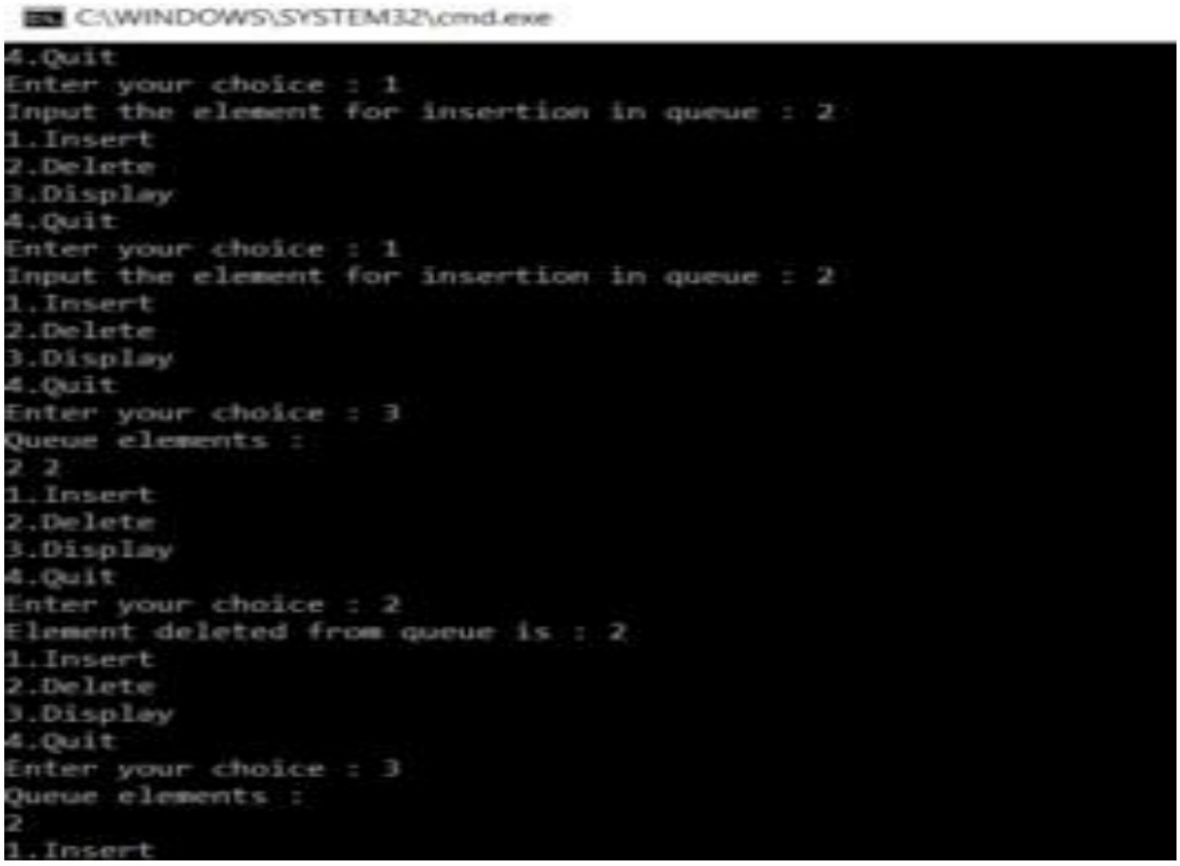
PROGRAM

```
#include<stdio.h>
# define MAX 5
int cqueue_arr[MAX];
int front = -1;
int rear = -1;
void insert(int item)
{
if((front == 0 && rear == MAX-1) || (front == rear+1))
{
printf("Queue Overflow \n");
return;
}
if(front == -1)
{
front = 0;
rear = 0;
}
else
{
if(rear == MAX-1)
rear = 0;
else
rear = rear+1;
}
cqueue_arr[rear] = item ;
}
```

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

```
printf("%d ",cqueue_arr[front_pos]);
front_pos++;
}
}
printf("\n");
}
int main()
{
int choice,item;
do
{
printf("1.Insert\n");
printf("2.Delete\n");
printf("3.Display\n");
printf("4.Quit\n");
printf("Enter your choice : \n");
scanf("%d",&choice);
switch(choice)
{
case 1 :
printf("Input the element for insertion in queue : ");
scanf("%d", &item);
insert(item);
break;
case 2 :
deletion();
break;
case 3:
display();
break;
case 4:
break;
default:
printf("Wrong choice\n");
}
}while(choice!=4);
return 0;
}
```

OUTPUT



RESULT:

Thus a C program to implement the circular queue and its operations were performed successfully.

Ex. No: 4.a. **IMPLEMENTATION OF BINARY TREE**
Date:

AIM:

To write a C program for implementing the Binary tree and its traversal operations

ALGORITHM:

Step 1: Start
Step 2: Declare the structure of binary tree with data (root), left,right.
Step 3: Declare a structure to create and allocate memory for binary tree
Step 4.a: After creating the binary tree, check if it is empty
Step 4.b: If not, create a binary tree that is a complete binary tree
Step 5: Then insert the data into the complete binary tree
Step 6:Tree tarversal operations can be done in three ways namely postorder, preorder and inorder traversals.
Step 7.a: Perform postorder,preorder and inorder traversal functions for the complete binary tree
Step 7.b.: This is done using switch case and while loop
Step 8: Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<stdlib.h>
struct btnode
{
int data;
struct btnode *left,*right;
};
struct btnode* create()
{
struct btnode* t;
t=(struct btnode*)malloc(sizeof(struct btnode));
t=NULL;
return t;
}
int complete(struct btnode *t,int i)
{
int x,y;
if(t==NULL)
return 0;
else
{
if(t->left==NULL && t->right==NULL)
return i+1;
else if(t->left&&t->right)
{
x=complete(t->left,i+1);
y=complete(t->right,i+1);
if(x==y)
```

```
return x;
return 0;
}
return 0;
}
}
struct btnode* insert(struct btnode *t,int x)
{
struct btnode *temp;
temp=(struct btnode*)malloc(sizeof(struct btnode));
temp->data=x;
temp->left=NULL;
temp->right=NULL;
if(t==NULL)
t=temp;
else if(t->left==NULL)
t->left=temp;
else if(t->right==NULL)
t->right=temp;
else
{
if(complete(t->left,1)==complete(t->right,1))
t->left=insert(t->left,x);
else if(complete(t->left,1))
t->right=insert(t->right,x);
else
t->left=insert(t->left,x);
}
return t;
}
void postorder(struct btnode *t)
{
if(t==NULL)
printf("\n empty tree");
else
{
if(t->left!=NULL)
postorder(t->left);
if(t->right!=NULL)
postorder(t->right);
printf("%d\t",t->data);
}
}
void preorder(struct btnode *t)
{
if(t==NULL)
```

```
printf("\n empty tree");
else
{
printf("%d\t",t->data);
if(t->left!=NULL)
preorder(t->left);
if(t->right!=NULL)
preorder(t->right);
}
}
void inorder(struct btnode *t)
{
if(t==NULL)
printf("\n empty tree");
else
{
if(t->left!=NULL)
inorder(t->left);
printf("%d\t",t->data);
if(t->right!=NULL)
inorder(t->right);
}
}
void main()
{
struct btnode *t;
int x,ch;
clrscr();
t=create();
printf("\n MENU \t 1.insert \t 2. postorder \t 3.preorder \t 4.inorder \t 5.exit");
do
{
printf("\n enter choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\n enter the data to be inserted:");
scanf("%d",&x);
t=insert(t,x);
break;
case 2:
postorder(t);
printf("\n the elements in postorder are printed");
break;
case 3:
```

```
preorder(t);
printf("\n the elements in preorder are printed");
break;
case 4:
inorder(t);
printf("\n the elements in inorder are printed");
break;
case 5:
exit(0);
}
}while(6);
}
```

OUTPUT:

```
MENU
1.insert        2.postorder     3.preorder      4.inorder       5.exit
Enter choice: 1
Enter the data to be inserted:10
Enter choice: 1
Enter the data to be inserted:20
Enter choice: 1
Enter the data to be inserted:30
Enter choice:2
20      30      10
The elements in postorder are printed
Enter choice:3
10      20      30
The elements in preorder are printed
Enter choice:4
20      10      30
The elements in inorder are printed
Enter choice: 1
Enter the data to be inserted: 40
Enter choice: 2
40      20      30      10
The elements in postorder are printed
Enter choice: 3
10      20      40      30
The elements in preorder are printed
Enter choice: 4
40      20      10      30
The elements in inorder are printed
Enter choice:5
(program exits)
```

RESULT:

Thus a C program to implement the creation and insertion in binary tree and its traversal operations were performed successfully

Ex. No: 4.b. **IMPLEMENTATION OF BINARY SEARCH TREE**
Date:

AIM:
To write a C program for implementing the Binary Search Tree (BST) and its traversal operations
ALGORITHM:
1. Start
2.a. Declare the structure variables for binary search tree
2.b. Define variables to find element type,position,minimum,maximum,insert and delete
3.a.Insert the elements into the binary search tree by checking left and right of the tree
3.b. Search the tree to find the position of the element and print it
3.c. If, while searching position it is not found, then item is not found,
3.d. Else , when found the item is deleted
4.Then perform inorder, preorder and postorder traversal operations for the elements
5. Using do while loop and switch case, elements are inserted, deleted, found and traversal operations were performed
6. Stop
PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#ifndef tree h
struct treenode;
typedef struct treenode *position;
typedef struct treenode *searchtree;
typedef struct treenode ptrtonode;
typedef int elementtype;
position find(elementtype x,searchtree t);
position findmin(searchtree t);
position findmax(searchtree t);
searchtree insert(elementtype x,searchtree t);
searchtree delet(elementtype x,searchtree t);
elementtype retrieve(position p);
#endif
struct treenode
{
elementtype element;
struct treenode *left;
struct treenode *right;
};
searchtree insert(elementtype x,searchtree t)
{
if(t==NULL)
{
t=(ptrtonode*)malloc(sizeof(struct treenode));
if(t==NULL)
printf("\n no space");
else
{
```

```
t->element=x;
t->left=NULL;
t->right=NULL;
} }
else if(x<t->element)
{
t->left=insert(x,t->left);
}
else if(x>t->element)
{
t->right=insert(x,t->right);
}
return t;
}
searchtree delet(elementtype x,searchtree t)
{
position tmp;
if(t==NULL)
{
}
else if(x<t->element)
{
t->left=delet(x,t->left);
}
else if(x>t->element)
{
t->right=delet(x,t->right);
}
else
{
if(t->right&&t->left)
{
tmp=findmin(t->right);
t->element=tmp->element;
t->right=delet(tmp->element,t->right);
}
else
{
tmp=t;
if(t->left==NULL)
t=t->right;
else if(t->right==NULL)
t=t->left;
free(tmp);
}
}
```

```
return t;
}
position find(elementtype x,searchtree t)
{
if(t==NULL)
return NULL;
else if(x<t->element)
{
return find(x,t->left);
}
else if(x>t->element)
{
return find(x,t->right);
}
else
return t;
}
position findmin(searchtree t)
{
if(t==NULL)
return NULL;
else if(t->left==NULL)
return t;
else
return findmin(t->left);
}
position findmax(searchtree t)
{
if(t==NULL)
return NULL;
else if(t->right==NULL)
return t;
else
return findmax(t->right);
}
void inorder(searchtree t)
{
if(t!=NULL)
{
inorder(t->left);
printf("\n %d",t->element);
inorder(t->right);
}
 }
void preorder(searchtree t)
{
```

```
if(t!=NULL)
{
printf("\n %d",t->element);
preorder(t->left);
preorder(t->right);
}
 }
void postorder(searchtree t)
{
if(t!=NULL)
{
postorder(t->left);
postorder(t->right);
printf("\n %d",t->element);
}
 }
int main()
{
int opt,item;
searchtree t;
position p;
t=NULL;
clrscr();
printf("\n MENU \n 1.insert \t 2.delete \t 3.find");
printf("\n 4.inorder traversal \t 5.preorder traversal \t 6.postorder traversal \t 7.exit");
do
{
printf("\n enter your choice:");
scanf("%d",&opt);
switch(opt)
{
case 1:
printf("\n enter the item to be inserted:");
scanf("%d",&item);
t=insert(item,t);
printf("\n item inserted");
break;
case 2:
printf("\n enter the item to be deleted:");
scanf("%d",&item);
if(find(item,t))
{
t=delet(item,t);
printf("\n item deleted");
}
else
```

```
{
printf("\n item not found");
}
break;
case 3:
printf("\n enter the item to find:");
scanf("%d",&item);
p=find(item,t);
if(p)
printf("\n item found");
else
printf("\n item not found");
break;
case 4:
if(t)
{
printf("\n inorder tree is:");
inorder(t);
}
else
printf("\n tree empty");
break;
case 5:
if(t)
{
printf("\n preorder tree is:");
preorder(t);
}
else
printf("\n tree empty");
break;
case 6:
if(t)
{
printf("\n postorder tree is:");
postorder(t);
}
else
printf("\n tree empty");
break;
case 7:
exit(0);
}
}while(1);
}
```

OUTPUT:

MENU

1.insert 2.delete 3.find

4.inorder traversal 5.preorder traversal 6.postorder traversal 7.exit

Enter your choice: 1

Enter the item to be inserted: 10

Item inserted Enter your choice: 1

Enter the item to be inserted: 20

Item inserted

Enter your choice: 1

Enter the item to be inserted: 40

Item inserted

Enter your choice: 1

Enter the item to be inserted: 50

Item inserted

Enter your choice: 1

Enter the item to be inserted: 60

Item inserted

Enter your choice: 3

Enter the item to find: 30

Item not found

Enter your choice: 3

Enter the item to find: 40

Item found

Enter your choice: 2

Enter the item to be deleted: 40

Item deleted

Enter your choice: 3

Enter the item to find: 40

Item not found

Enter your choice: 4

Inorder tree is:

10

20

50

60

Enter your choice: 5

Preorder tree is:

10

20

50

60

Enter your choice: 6

Postorder tree is:

60

50

20

10

Enter your choice: 7

(program exits)

RESULT:

Thus a C program for implementing the Binary Search Tree to insert, delete and find the element in tree and its inorder, preorder and postorder traversal operations were completed successfully

Ex. No: 5 **IMPLEMENTATION OF AVL TREES**
Date:

AIM:

To implement the AVL Trees for performing creation, insertion, deletion, searching and tree traversing operations in the AVL tree elements

ALGORITHM:

Step 1: Start.
Step 2: Declare the structure of AVL tree with data (root), left, right.
Step 3: Declare a structure to create and allocate memory for AVL tree
Step 4.a: After creating the AVL tree, check if it is empty
Step 4.b: If not, create AVL tree that is a complete binary tree
Step 5.a:Perform single left rotation and single right rotation in the AVL tree
Step 5.b: Perform double left rotation and double right rotation in the AVL tree;
Step 5.c: Then insert the data into the AVL tree
Step 6: Tree tarversal operations can be done in three ways namely postorder, preorder and inorder traversals.
Step 7.a: Perform postorder,preorder and inorder traversal functions for the AVL tree
Step 7.b.: This is done using switch case and while loop
Step 8: Stop

PROGRAM::

```
#include<stdio.h>
#include<conio.h>
struct treenode;
typedef struct treenode *position;
typedef struct treenode *avltree;
typedef struct treenode ptrtonode;
typedef int elementtype;
position find(elementtype x,avltree t);
position findmin(avltree t);
position findmax(avltree t);
avltree insert(elementtype x,avltree t);
avltree delet(elementtype x,avltree t);
elementtype retrieve(position p);
struct treenode
{
elementtype element;
avltree left;
avltree right;
int height;
};
int height(position p)
{
if(p!=NULL)
return p->height;
else
return 0;
```

```
}
int max(int h1,int h2)
{
if(h1>h2)
return h1;
else
return h2;
}
position singlerightrotation(position k2)
{
position k1;
k1=k2->left;
k2->left=k1->right;
k1->right=k2;
k2->height=max(height(k2->left),height(k2->right))+1;
k1->height=max(height(k1->left),height(k1->right))+1;
return k1;
}
position singleleftrotation(position k1)
{
position k2;
k2=k1->right;
k1->right=k2->left;
k2->left=k1;
k1->height=max(height(k1->left),height(k1->right))+1;
k2->height=max(height(k2->left),height(k2->right))+1;
return k2;
}
position doublerightrotation(position k3)
{
k3->left=singleleftrotation(k3->left);
return singlerightrotation(k3);
}
position doubleleftrotation(position k3)
{
k3->right=singleleftrotation(k3->right);
return singleleftrotation(k3);
}
avltree insert(elementtype x, avltree t)
{
if(t==NULL){
t=(ptrtonode*)malloc(sizeof(struct treenode));
if(t==NULL)
printf("\n no space");
else{
t->element=x;
```

```
t->left=NULL;
t->right=NULL;
t->height=0;
} }
else if(x<t->element){
t->left=insert(x,t->left);
if(height(t->left)-height(t->right)==2)
if(x<t->left->element)
t=singlerightrotation(t);
else
t=doublerightrotation(t);
}
else if(x>t->element){
t->right=insert(x,t->right);
if(height(t->right)-height(t->left)==2)
if(x<t->right->element)
t=singleleftrotation(t);
else
t=doubleleftrotation(t);
}
t->height=max(height(t->left),height(t->right))+1;
return t;
}
avltree delet(elementtype x,avltree t)
{
position tmp;
if(t==NULL)
{
}
else if(x<t->element){
t->left=delet(x,t->left);
}
else if(x>t->element){
t->right=delet(x,t->right);
}
else{
if(t->right&&t->left){
tmp=findmin(t->right);
t->element=tmp->element;
t->right=delet(tmp->element,t->right);
}
else{
tmp=t;
if(t->left==NULL)
t=t->right;
else if(t->right==NULL)
```

```
t=t->left;
free(tmp);
} }
return t;
}
position find(elementtype x,avltree t)
{
if(t==NULL)
return NULL;
else if(x<t->element){
return find(x,t->left);
}
else if(x>t->element){
return find(x,t->right);
}
else
return t;
}
position findmin(avltree t)
{
if(t==NULL)
return NULL;
else if(t->left==NULL)
return t;
else
return findmin(t->left);
}
position findmax(avltree t)
{
if(t==NULL)
return NULL;
else if(t->right==NULL)
return t;
else
return findmax(t->right);
}
void inorder(avltree t)
{
if(t!=NULL){
inorder(t->left);
printf("\n %d",t->element);
inorder(t->right);
}}
void preorder(avltree t)
{
if(t!=NULL){
```

```
printf("\n %d",t->element);
preorder(t->left);
preorder(t->right);
} }
void postorder(avltree t)
{
if(t!=NULL){
postorder(t->left);
postorder(t->right);
printf("\n %d",t->element);
} }
int main()
{
int opt,item;
avltree t;
position p;
t=NULL;
clrscr();
printf("\n MENU \n 1.insert \t 2.delete \t 3.find");
printf("\n 4.inorder traversal \t 5.preorder traversal \t 6.postorder traversal \t 7.exit");
do{
printf("\n enter your choice:");
scanf("%d",&opt);
switch(opt){
case 1:
printf("\n enter the item to be inserted:");
scanf("%d",&item);
t=insert(item,t);
printf("\n item inserted");
break;
case 2:
printf("\n enter the item to be deleted:");
scanf("%d",&item);
if(find(item,t)){
t=delet(item,t);
printf("\n item deleted");
}
else{
printf("\n item not found");
}
break;
case 3:
printf("\n enter the item to find:");
scanf("%d",&item);
p=find(item,t);
if(p)
```

```
printf("\n item found");
else
printf("\n item not found");
break;
case 4:
if(t){
printf("\n inorder tree is:");
inorder(t);
}
else
printf("\n tree empty");
break;
case 5:
if(t){
printf("\n preorder tree is:");
preorder(t);
}
else
printf("\n tree empty");
break;
case 6:
if(t){
printf("\n postorder tree is:");
postorder(t);
}
else
printf("\n tree empty");
break;
case 7:
exit(0);
}
}while(1);
}
```

OUTPUT:

```
1.insert          2.delete          3.find
4.inorder traversal      5.preorder traversal          6.postorder traversal          7.exit
Enter your choice: 1
Enter the item to be inserted: 10
Item inserted Enter your choice: 1
Enter the item to be inserted: 20
Item inserted
Enter your choice: 1
Enter the item to be inserted: 40
Item inserted
Enter your choice: 1
```

Enter the item to be inserted: 50
Item inserted
Enter your choice: 1
Enter the item to be inserted: 60
Item inserted
Enter your choice: 3
Enter the item to find: 30
Item not found
Enter your choice: 3
Enter the item to find: 40
Item found
Enter your choice: 2
Enter the item to be deleted: 40
Item deleted
Enter your choice: 3
Enter the item to find: 40
Item not found
Enter your choice: 4
Inorder tree is:
10
20
50
60
Enter your choice: 5
Preorder tree is:
50
10
20
60
Enter your choice: 6
Postorder tree is:
20
10
50
60
Enter your choice: 7
(program exits)

RESULT:
Thus a C program to perform creation, insertion, deletion, searching, inorder, preorder and postorder traversal operations in AVL trees was completed successfully

Ex. No: 6 **IMPLEMENTATION OF PRIORITY QUEUE USING HEAPS**

Date:

AIM:

To write a C program for implementing the priority queue using heap sort

ALGORITHM:

Step 1: Declare the structure of the priority queue
Step 2.a: Allocate memory for storing values and assigning priority
Step 2.b: Before inserting the elements, check whether the queue is full.
Step 3: After insertion, the elements are ordered based on their priority
Step 4.a: Delete the elements with the minimum or maximum priority
Step 4.b: Before deletion, check whether the queue is empty.
Step 4.c: If queue is empty, display the error message.
Step 5: After deletion, remaining part of the queue will be zero
Step 6: Display the elements in the queue, then elements are printed on order of priority
Step 7: Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<malloc.h>
void insert();
void delet();
void display();
struct hnode
{
int priority;
int data;
struct hnode *next;
}*start=NULL,*q,*temp,*neww;
typedef struct hnode *n;
void main()
{
int ch;
clrscr();
printf("\n MENU \n 1.insertion \t 2.deletion \t 3.display \t 4.exit");
do
{
printf("\n enter choice:");
scanf("%d",&ch);
switch(ch)
{
case 1: insert();
break;
case 2:delet();
break;
case 3: display();
```

```
break;
case 4: exit(0);
} }while(ch<4);
}
void insert()
{
int item,itprior;
neww=(n*)malloc(sizeof(n));
printf("\n enter the element to be inserted:");
scanf("%d",&item);
printf("\n enter its priority");
scanf("%d",&itprior);
neww->data=item;
neww->priority=itprior;
neww->next=NULL;
if(start==NULL)
start=neww;
else if(start!=NULL && itprior<=start->priority)
{
neww->next=start;
start=neww;
}
else
{
q=start;
while(q->next!=NULL && q->next->priority<=itprior)
{
q=q->next;
}
neww->next=q->next;
q->next=neww;
}}
void delet()
{
if(start==NULL)
{
printf("\n queue underflow");
}
else
{
neww=start;
printf("\n deleted item is %d \n:",neww->data);
start=start->next;
}}
void display()
{
```

```
temp=start;
if(start==NULL)
printf("\n queue is empty");
else
{
printf("\n queue is:");
if(temp!=NULL)
for(temp=start;temp!=NULL;temp=temp->next)
{
printf("\n %d priority =%d \n",temp->data, temp->priority);
}}}
```

OUTPUT:

```
MENU
1.insertion      2.deletion      3.display      4.exit
Enter choice: 1
Enter the element to be inserted: 10
Enter its priority: 1
Enter choice: 1
Enter the element to be inserted: 20
Enter its priority: 2
Enter choice: 1
Enter the element to be inserted: 30
Enter its priority: 3
Enter choice: 1
Enter the element to be inserted: 40
Enter its priority: 4
Enter choice: 3
Queue is
10      priority=1
20      priority=2
30      priority=3
40      priority=4
Enter choice: 2
Deleted item is: 10
Enter choice: 2
Deleted item is: 20
Enter choice: 3
Queue is
30      priority=3
40      priority=4
Enter choice: 4
(program exits)
```

RESULT:

Thus a C program for implementing the priority queues using heap sort is completed successfully

Ex. No: 7.a. **DIJKSTRA'S SHORTEST PATH**

Date:

AIM:

To find the shortest path for the given graph from a specified source to all other vertices using Dijkstra's algorithm.

ALGORITHM

1. Start
2. Obtain no. of vertices and adjacency matrix for the given graph
3. Create cost matrix from adjacency matrix. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity
4. Initialize visited[] to zero
5. Read source vertex and mark it as visited
6. Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex
distance[i]=cost[0][i];
7. Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.
8. Recalculate the shortest distance of remaining vertices from the source.
9. Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v
if(visited[v]==0)
distance[v]=min(distance[v]
distance[w]+cost[w][v])
10. Stop

PROGRAM

```
#include <stdio.h>
#include <conio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX], int n, int startnode);
main()
{
int G[MAX][MAX], i, j, n, u;
clrscr();
printf("Enter no. of vertices: ");
scanf("%d", &n);
printf("Enter the adjacency matrix:\n");
for(i=0; i<n; i++)
for(j=0; j<n; j++)
scanf("%d", &G[i][j]);
printf("Enter the starting node: ");
```

```
scanf("%d", &u);
dijkstra(G, n, u);
}
void dijkstra(int G[MAX][MAX], int n,int startnode)
{
int cost[MAX][MAX], distance[MAX], pred[MAX];
int visited[MAX],count, mindistance, nextnode, i, j;
for(i=0; i<n; i++)
for(j=0; j<n; j++)
if(G[i][j] == 0)
cost[i][j] = INFINITY;
else
cost[i][j] = G[i][j];
for(i=0; i<n; i++)
{
distance[i] = cost[startnode][i];
pred[i] = startnode;
visited[i] = 0;
}
distance[startnode] = 0;
visited[startnode] = 1;
count = 1;
while(count < n-1)
{
mindistance = INFINITY;
for(i=0; i<n; i++)
if(distance[i] < mindistance && !visited[i])
{
mindistance = distance[i];
nextnode=i;
}
visited[nextnode] = 1;
for(i=0; i<n; i++)
if(!visited[i])
if(mindistance + cost[nextnode][i] <
distance[i])
{
distance[i] = mindistance +
cost[nextnode][i];
pred[i] = nextnode;
}
count++;
}
for(i=0; i<n; i++)
if(i != startnode)
{
```

```
printf("\nDistance to node%d = %d", i,
distance[i]);
printf("\nPath = %d", i);
j = i;
do
{
j = pred[j];
printf("<-%d", j);
} while(j != startnode);
}
getch();
}
```

Output

```
Enter no. of vertices: 5
Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 0 60 0
Enter the starting node: 0
Distance to node1 = 10
Path = 1<-0
Distance to node2 = 50
Path = 2<-3<-0
Distance to node3 = 30
Path = 3<-0
Distance to node4 = 60
Path = 4<-2<-3<-0
```

Result

Thus Dijkstra's algorithm is used to find shortest path from a given vertex.

Ex. No: 7.b. **PRIM'S ALGORITHM**
Date:

AIM:

To find the shortest path for the given graph from a specified source to all other vertices using Prim's algorithm.

ALGORITHM

1. Select a starting vertex
2. Select an edge connecting the tree vertex and fringe vertex that has minimum weight.
3. Repeat Steps 3 and 4 until there are fringe vertices
4. Add the selected edge and the vertex to the minimum spanning tree T
5. [END OF LOOP]
6. EXIT

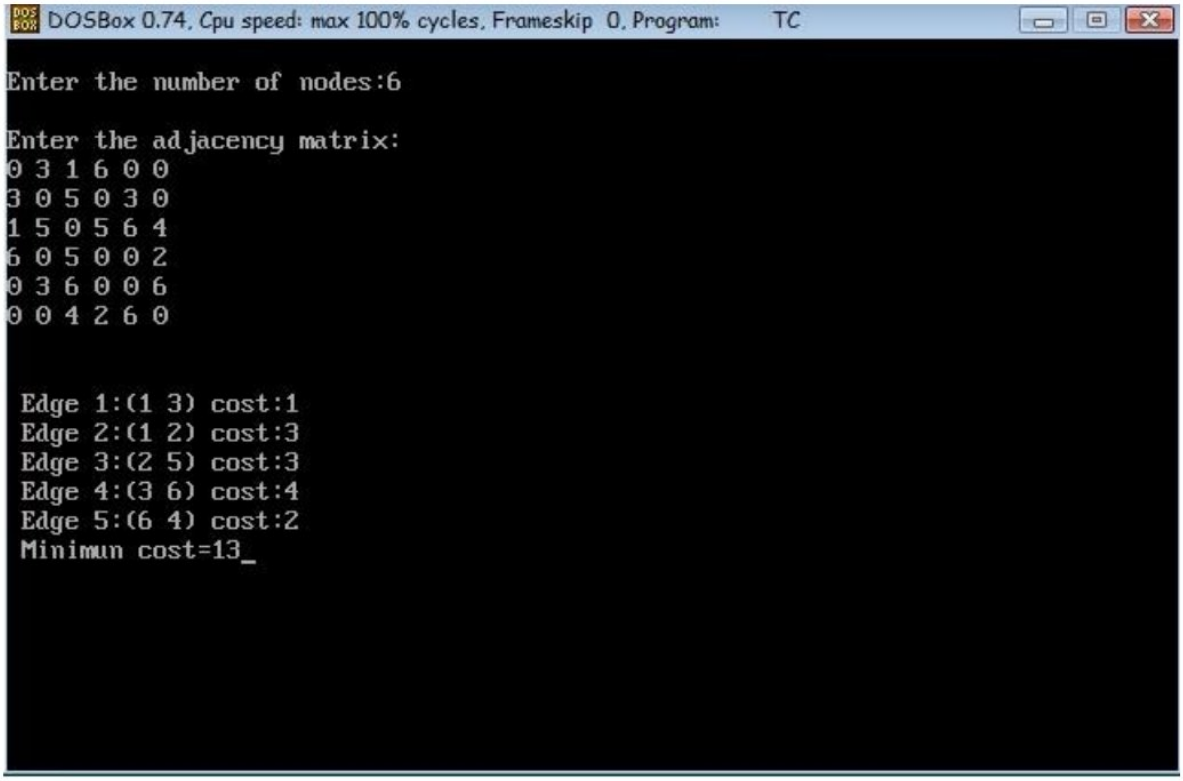
PROGRAM

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
 void main()
 {
        clrscr();
        printf("\nEnter the number of nodes:");
        scanf("%d",&n);
        printf("\nEnter the adjacency matrix:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)

        {
                scanf("%d",&cost[i][j]);
                if(cost[i][j]==0)
                        cost[i][j]=999;
        }
        visited[1]=1;
        printf("\n");
        while(ne < n)
        {
                for(i=1,min=999;i<=n;i++)
                for(j=1;j<=n;j++)
                if(cost[i][j]< min)
                if(visited[i]!=0)
                {
                        min=cost[i][j];
```

```
                    a=u=i;
                    b=v=j;
                }
                if(visited[u]==0 || visited[v]==0)
                {
                    printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                    mincost+=min;
                    visited[b]=1;
                }
                cost[a][b]=cost[b][a]=999;
        }
        printf("\n Minimun cost=%d",mincost);
        getch();
}
```

OUTPUT:



Result

Thus Prim's algorithm is used to find shortest path from a given vertex.

Ex. No: 8 **IMPLEMENTATION OF DEPTH FIRST SEARCH**

Date:

AIM:

To write a C program for implementing the depth first search (DFS) algorithm using Graphs

ALGORITHM:

Step 1: Start by declaring the structure for creating the node with vertex and edges
Step 2.a: Declare function to read the graph and insert values for the graph
Step 2.b: Traversal starts from a vertex by visiting each adjacent vertex by traversing recursively using DFS
Step 3.a: Graph can have cycles so make sure the nodes are not revisited
Step 3.b: If a node is visited, mark it and store in an array and then move on to the next
Step 4: Based on the adjacency matrix, values are searched in the graph and printed
Step 5: Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
typedef struct node {
struct node  *next;
int vertex;
}node;
node *g[20];
int visited[20];
int n;
void read_graph();
void insert(int,int);
void dfs(int);
void main()
{
int i;
clrscr();
read_graph();
for(i=0;i<n;i++)
visited[i]=0;
dfs(0);
getch();
}
void dfs(int i)
{
node *p;
printf("\t %d",i);
p=g[i];
visited[i]=1;
while(p!=NULL) {
i=p->vertex;
if(!visited[i])
```

```
dfs(i);
p=p->next;
}}
void read_graph() {
int i,vi,vj,edges;
printf("\n enter the number of vertices:");
scanf("%d",&n);
for(i=0;i<n;i++) {
g[i]=NULL;
printf("\n enter the number of edges:");
scanf("%d",&edges);
for(i=0;i<edges;i++) {
printf("\n enter edge(u,v):");
scanf("%d %d",&vi,&vj);
insert(vi,vj);
}}}
void insert(int vi,int vj) {
node *p,*q;
q=(node*)malloc(sizeof(node));
q->vertex=vj;
q->next=NULL;
if(g[vi]==NULL)
g[vi]=q;
else {
p=g[vi];
while(p->next!=NULL)
p=p->next;
p->next=q;
}}
```

OUTPUT:

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

RESULT:

Thus a C Program for implementing Depth First Search (DFS) algorithm is implemented successfully

Ex. No: 9.a **IMPLEMENTATION OF LINEAR SEARCH**

Date:

AIM:

To write a C program for implementing the linear search algorithm

ALGORITHM:

Step 1: Start with the first item in the list.
Step 2: Compare the current item to the target
Step 3: If the current value matches the target then we declare victory and stop.
Step 4: If the current value is less than the target then set the current item to be the next item and repeat from 2.
Step 5: Stop.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main
{
int a[20];
int i,size,search;
clrscr();
printf("\n LINEAR SEARCH");
printf("\n enter the total number of elements:");
scanf("%d",&size);
for(i=0;i<size;i++){
printf("\t enter %d elements:",i+1);
scanf("%d",&a[i]); }
printf("\n enter the element to be searched:");
scanf("%d",&search);
for(i=0;i<size;i++){
if(search==a[i]){
printf("\n element exists in list at position:%d",i+1);
break;
}
else{
printf("\n element not found");
break;
}}getch();
}
```

OUTPUT:

Enter the total number of elements: 4
Enter 1 elements: 11 Enter 2 elements: 22
Enter 3 elements: 33 Enter 4 elements: 44
Enter the element to be searched: 33
Element exists in list at position: 3

RESULT: Thus a C program for linear search algorithm is implemented successfully

Ex. No. 9.b **BINARY SEARCH**
Date:

AIM
To locate an element in a sorted array using Binary search method

ALGORITHM
1. Start
2. Read number of array elements, say *n*
3. Create an array *arr* consisting *n* sorted elements
4. Get element, say *key* to be located
5. Assign 0 to *lower* and *n* to *upper*
6. While (*lower* < *upper*)
Determine middle element *mid* = (upper+lower)/2
If key = arr[mid] then
Print mid
Stop
Else if key > arr[mid] then
lower = mid + 1
else
upper = mid – 1
7. Print "Element not found"
8. Stop

PROGRAM

```
/* Binary Search on a sorted array */
#include <stdio.h>
#include <conio.h>
void main()
{
int a[50],i, n, upper, lower, mid, val, found;
clrscr();
printf("Enter array size : ");
scanf("%d", &n);
for(i=0; i<n; i++)
a[i] = 2 * i;
printf("\n Elements in Sorted Order \n");
for(i=0; i<n; i++)
printf("%4d", a[i]);
printf("\n Enter element to locate : ");
scanf("%d", &val);
upper = n;
lower = 0;
found = -1;
while (lower <= upper)
```

```
{
mid = (upper + lower)/2;
if (a[mid] == val)
{
printf("Located at position %d", mid);
found = 1;
break;
}
else if(a[mid] > val)
upper = mid - 1;
else
lower = mid + 1;
}
if (found == -1)
printf("Element not found");
getch();
}
```

OUTPUT

```
Enter array size : 9
Elements in Sorted Order
0 2 4 6 8 10 12 14 16
Enter element to locate : 12
Located at position 6
Enter array size : 10
Elements in Sorted Order
0 2 4 6 8 10 12 14 16 18
Enter element to locate : 13
Element not found
```

RESULT:

Thus a C program for binary search algorithm is implemented successfully

Ex. No: 10.a. **INSERTION SORT**

Date:

AIM:

To sort an array of N numbers using Insertion sort.

ALGORITHM

1. Start
2. Read number of array elements *n*
3. Read array elements *Ai*
4. Sort the elements using insertion sort

In pass p, move the element in position p left until its correct place is found among the first p + 1 elements.
Element at position p is saved in temp, and all larger elements (prior to position p) are moved one spot to the right. Then temp is placed in the correct spot.

5. Stop

PROGRAM

```
#include<stdio.h>

#include<conio.h>

main()

{

int i, j, k, n, temp, a[20], p=0;

printf("Enter total elements: ");

scanf("%d",&n);

printf("Enter array elements: ");

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

scanf("%d", &a[i]);

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

{

temp = a[i];

j = i - 1;

while((temp<a[j]) && (j>=0))
```

```
{
a[j+1] = a[j];
j = j - 1;
}
a[j+1] = temp;
p++;
printf("\n After Pass %d: ", p);
for(k=0; k<n; k++)
printf(" %d", a[k]);
}
printf("\n Sorted List : ");
for(i=0; i<n; i++)
printf(" %d", a[i]);
}
```

OUTPUT

```
Enter total elements: 6
Enter array elements: 34 8 64 51 32 21
After Pass 1: 8 34 64 51 32 21
After Pass 2: 8 34 64 51 32 21
After Pass 3: 8 34 51 64 32 21
After Pass 4: 8 32 34 51 64 21
After Pass 5: 8 21 32 34 51 64
Sorted List : 8 21 32 34 51 64
```

RESULT

Thus array elements was sorted using insertion sort.

Ex. No: 10.b. **SELECTION SORT**

Date:

AIM:

To sort an array of N numbers using selection sort.

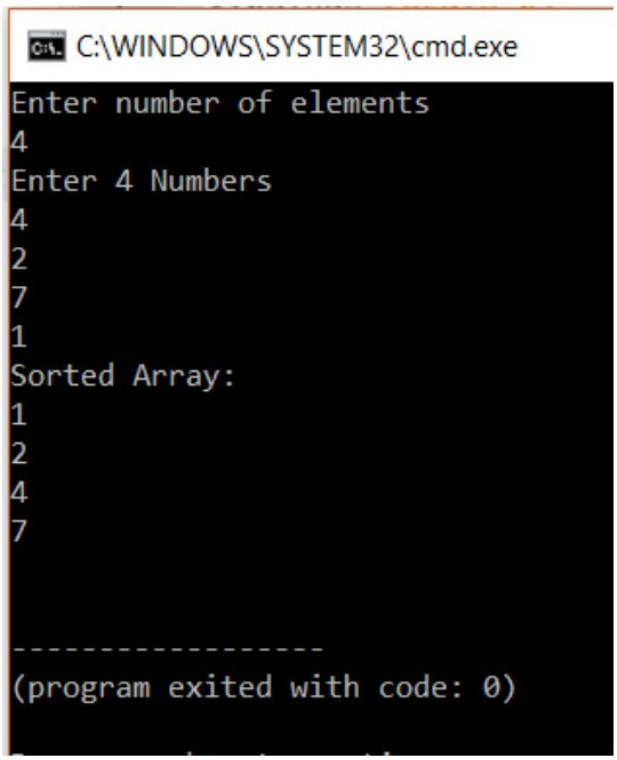
ALGORITHM:

Step 1 − Set min to the first location
Step 2 − Search the minimum element in the array
Step 3 – swap the first location with the minimum value in the array
Step 4 – assign the second element as min.
Step 5 − Repeat the process until we get a sorted array.

PROGRAM:

```
#include <stdio.h>
#include<conio.h>
int main()
{
int a[100], n, i, j, position, swap;
clrscr();
printf("Enter number of elements\n");
scanf("%d", &n);
printf("Enter %d Numbers\n", n);
for (i = 0; i < n; i++)
scanf("%d", &a[i]);
for(i = 0; i < n - 1; i++)
{
position=i;
for(j = i + 1; j < n; j++)
{
if(a[position] > a[j])
position=j;
}
if(position != i)
{
swap=a[i];
a[i]=a[position];
a[position]=swap;
}
}
printf("Sorted Array\n");
for(i = 0; i < n; i++)
printf("%d\n", a[i]);
getch();
return 0;
}
```

OUTPUT



RESULT:
Thus array elements was sorted using selection sort.

Ex. No: 10.c. IMPLEMENTATION OF QUICK SORT

Date:

AIM:

To write a C program for implementing quick sort algorithm

ALGORITHM:

Step 1: Define the function for performing quick sort
Step 2.a: Allocate a pivot element in the list and compare the first and last elements
Step 2.b: If the elements are not in order, swap them and order them in list
Step 3: This is done for all the elements stored in the list
Step 4.a: Declare the number of elements and count them
Step 4.b: Print the ordered/sorted list of elements
Step 5: Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void quick(int num[25],int first,int last){
int i,j,pivot,temp;
if(first<last)
{
pivot=first;
i=first;
j=last;
while(i<j)
{
while(num[i]<=num[pivot]&&i<last)
i++;
while(num[j]>num[pivot])
j--;
if(i<j)
{
temp=num[i];
num[i]=num[j];
num[j]=temp;
}}
temp=num[pivot];
num[pivot]=num[j];
num[j]=temp;
quick(num,first,j-1);
quick(num,j+1,last);
}}
void main()
{
int i,count,num[25];
clrscr();
printf("\n enter the number of elements:");
scanf("%d",&count);
```

```
printf("\n enter %d elements:",count);
for(i=0;i<count;i++)
scanf("%d",&num[i]);
quick(num,0,count-1);
printf("\n quick sorted elements:");
for(i=0;i<count;i++)
printf("%d \t",num[i]);
getch();
}
```

OUTPUT:

```
Enter the number of elements : 6
Enter 6 elements:
99
11
88
22
77
33
Quick sorted elements: 11      22      33      77      88      99
```

RESULT:

Thus a C program for implementing the quick sort is completed successfully

Ex. No: 10.d. **IMPLEMENTATION OF MERGE SORT**

Date:

AIM:

To write a C program for implementing merge sort by combining two sorted lists

ALGORITHM:

Step 1: Define the function for performing merge sort
Step 2.a: Obtain the number of elements for list1 and list2
Step 2.b: Enter the ordered elements for list 1 and list 2
Step 3: Compare the elements of both lists and merge them
Step 4.a: Then perform sorting of two lists after merging them together as one list
Step 4.b: Print the combined or merged and ordered/sorted list of elements
Step 5: Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void merge(int [],int,int [],int, int []);
int main()
{
int a[50],b[50],m,n,c,sorted[100];
clrscr();
printf("\n enter the number of elements in first list:");
scanf("%d",&m);
printf("\n enter %d integers \n",m);
for(c=0;c<m;c++){
scanf("%d",&a[c]);
}
printf("\n enter the number of elements in second list:");
scanf("%d",&n);
printf("\n enter %d integers \n",n);
for(c=0;c<n;c++){
scanf("%d",&b[c]);
}
merge(a,m,b,n,sorted);
printf("\n sorted list is:");
for(c=0;c<m+n;c++){
printf("\n %d\t",sorted[c]);
}
getch();
return 0;
}
void merge(int a[],int m,int b[],int n,int sorted[])
{
int i,j,k;
j=k=0;
for(i=0;i<m+n;){
if(j<m && k<n){
```

```
if(a[j]<b[k]){
sorted[i]=a[j];
j++;
}
else{
sorted[i]=b[k];
k++;
}
i++;
}
else if(j==m){
for(;i<m+n;){
sorted[i]=b[k];
k++;
i++;
}}
else{
for(;i<m+n;){
sorted[i]=a[j];
j++;
i++;
}}}}
```

OUTPUT:

```
Enter the number of elements in the first list: 4
Enter 4 integers
11
55
77
99
Enter the number of elements in the second list: 4
Enter 4 integers
22
44
66
88
Sorted list is:
11      22      44      55      66      77      88      99
```

RESULT:

Thus a C program to implement merge sort by combining two sorted list is completed successfully

Ex. No: 11.a **IMPLEMENTATION OF HASHING TECHNIQUE**

Date:

AIM:

To write a C program for implementing the separate chaining hashing technique

ALGORITHM:

Step 1: Start by creating a list and a hash table using structure
Step 2: Inside the hash table, store the table size and allocate memory for the size of the table
Step 3: Insert the values in the list and its key will be stored in the hash table
Step 4: After insertion, search for the value in the table. The search happens efficiently with the help of the hash table
Step 5: The searched value is printed if found
Step 6: Display the elements in the list by traversing each value in the list with the hashing technique and then print it
Step 6: This is done with the help of while loop and switch case statements
Step 7: Stop

PROGRAM:

```
#include<stdio.h>
#include<malloc.h>
struct list{
int element;
struct list*next;
};
struct hashtable{
int tablesize;
struct list**thelists;
};
int hash(int key,int tablesize){
return(key%tablesize);
}
int nextprime(int x){
int i=x,c;
while(c!=1){
int j;
c=0;
i++;
for(j=2;j<=i;j++)
if(i%j==0)c++;
}
return i;
}
struct  hashtable*initialize(int tablesize){
struct hashtable*H;
int i;
H=malloc(sizeof(struct hashtable));
H->tablesize=nextprime(tablesize);
H->thelists=malloc(sizeof(struct list)*H->tablesize);
```

```
for(i=0;i<H->tablesize;i++){
H->thelists[i]=malloc(sizeof(struct list));
H->thelists[i]->next=NULL;
}
return H;
}
struct list *find(int key,struct hashtable*H){
struct list *p,*l;
l=H->thelists[hash(key,H->tablesize)];
p=l->next;
while(p!=NULL && p->element!=key)
p=p->next;
return p;
}
void insert(int key,struct hashtable * H){
struct list*pos,*newcell,*l;
pos=find(key,H);
if(pos==NULL){
newcell=malloc(sizeof(struct list));
l=H->thelists[hash(key,H->tablesize)];
newcell->next=l->next;
newcell->element=key;
l->next=newcell;
}}
void display(struct hashtable *H){
int i;
struct list *temp;
for(i=0;i<H->tablesize;i++){
temp=H->thelists[i]->next;
while(temp!=NULL){
printf("%d\t",temp->element);
temp=temp->next;
}}}
int main(){
int n,ch,data;
struct hashtable *H;
struct list *temp;
printf("\nEnter the size of table:");
scanf("%d",&n);
H=initialize(n);
do{
printf("\n1.Insert\n2.Search\n3.Display\n4.Exit\nEnter choice:");
scanf("%d",&ch);
switch(ch){
case 1:
printf("\nEnter data to be inserted:");
```

```
scanf("\%d",&data);
insert(data,H);
break;
case 2:
printf("\nEnter data to be searched");
scanf("%d",&data);
temp=find(data,H);
if(temp==NULL)
printf("\nData not found");
else
printf("%d is found",temp->element);
break;
case 3:
display(H);
break;
case 4:
return 0;
}}
while(5);
}
```

OUTPUT:

```
Enter the size of the table: 5
1.Insert
2. Search
3.Display
4.Exit
Enter choice: 1
Enter data to be inserted: 11
Enter choice: 1
Enter data to be inserted: 22
Enter choice: 1
Enter data to be inserted: 33
Enter choice: 3
33      11      22
Enter the data to be searched: 22
22 is found
Enter choice: 4
(program exits)
```

RESULT: Thus a C program for separate chaining hashing technique is implemented successfully

Ex. No.11.b **OPEN ADDRESSING HASHING TECHNIQUE**

Date:

Aim

To implement hash table using a C program.

Algorithm

1. Create a structure, data (hash table item) with key and value as data.
2. Now create an array of structure, data of some certain size (10, in this case). But, the size of array must be immediately updated to a prime number just greater than initial array capacity (i.e 10, in this case).
3. A menu is displayed on the screen.
4. User must choose one option from four choices given in the menu
5. Perform all the operations
6. Stop

Program

```
/* Open hashing */
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
main()
{
int a[MAX], num, key, i;
char ans;
int create(int);
void linearprobing(int[], int, int);
void display(int[]);
printf("\nCollision handling by linear probing\n\n");
for(i=0; i<MAX; i++)
a[i] = -1;
do
{
printf("\n Enter number:");
scanf("%d", &num);
key = create(num);
linearprobing(a, key, num);
printf("\nwish to continue?(y/n):");
ans = getch();
} while( ans == 'y');
display(a);
}
int create(int num)
{
int key;
```

```
key = num % 10;
return key;
}
void linearprobing(int a[MAX], int key, int num)
{
int flag, i, count = 0;
void display(int a[]);
flag = 0;
if(a[key] == -1)
a[key] = num;
else
{
i=0;
while(i < MAX)
{
if(a[i] != -1)
count++;
i++;
}
if(count == MAX)
{
printf("hash table is full");
display(a);
getch();
exit(1);
}
for(i=key+1; i<MAX; i++)
if(a[i] == -1)
{
a[i] = num;
flag = 1;
break;
}
for(i=0; i<key && flag==0; i++ )
if(a[i] == -1)
{
a[i] = num;
flag = 1;
break;
}
}
}
void display(int a[MAX])
{
int i;
printf("\n Hash table is:");
```

```
for(i=0; i<MAX; i++)
printf("\n %d\t\t%d",i,a[i]);
}
```

Output

```
Collision handling by linear probing
Enter number:1
wish to continue?(y/n):
Enter number:26
wish to continue?(y/n):
Enter number:62
wish to continue?(y/n):
Enter number:93
wish to continue?(y/n):
Enter number:84
wish to continue?(y/n):
Enter number:15
wish to continue?(y/n):
Enter number:76
wish to continue?(y/n):
Enter number:98
wish to continue?(y/n):
Enter number:26
wish to continue?(y/n):
Enter  number:199
wish to continue?(y/n):
Enter number:1234
wish to continue?(y/n):
Enter number:5678
hash table is full
Hash table is:
0 1234
1 1
2 62
3 93
4 84
5 15
6 26
7 76
8 98
9 199
```

Result

Thus hashing has been performed successfully.