

DHANALAKSHMI

COLLEGE OF ENGINEERING, CHENNAI

Dr. V.P.R. Nagar, Manimangalam Village, Varadharajapuram Post, Chennai - 601 301
Phone : 2717 8366 ; Website : dce-edu.com ; e-mail: dce-edu@yahoo.com

Name :

Year : Semester :

Branch :

University Register Number

Certificate

Certified that this is a bonafide record of the work done by Mr./Ms.

in the laboratory during the year

Signature of Head of the department

Signature of laboratory in-charge

Internal Examiner

External Examiner

CONTENTS

[illegible]

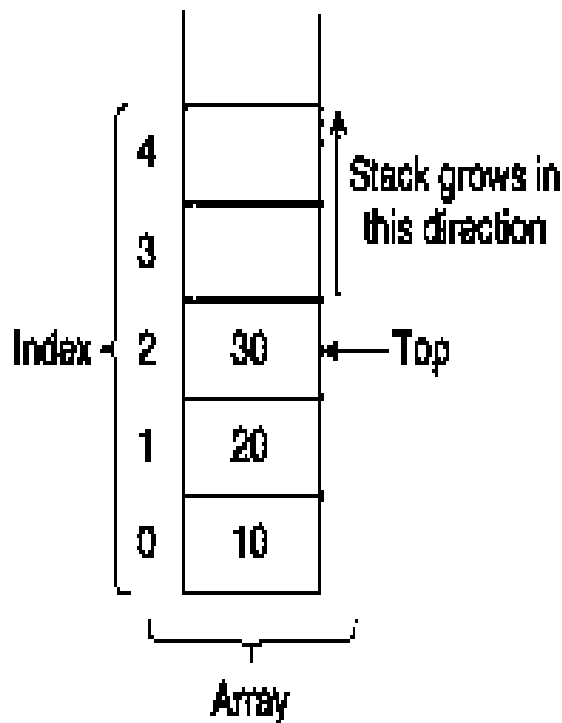
Ex. No: 1A**ARRAY IMPLEMENTATION OF STACK ADT****AIM**

To write a C program to implement Stack operations such as push, pop and display using array.

PRE LAB DISCUSSION

An array is a random access data structure, where each element can be accessed directly and in constant time. A typical illustration of random access is a book - each page of the book can be open independently of others. Random access is critical to many algorithms, for example binarysearch.

A stack data structure can be implemented using one dimensional array. But stack implemented using array, can store only fixed number of data values. This implementation is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using LIFO principle with the help of a variable 'top'. Initially top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.



ALGORITHM

Step 1: Start.

Step 2: Initialize 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 5: Display operation checks that top is not equal to -1 and returns storage[top];

Step 6: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define size 5
int item;
int s[10];
int top;
void display()
{
    int i;
    if(top==-1)
    {
        printf("\nstack is empty");
        return;
    }
    printf("\nContent of stack is:\n");
    for(i=0;i<=top;i++)
        printf("%d\t",s[i]);
}
void push()
{
```

```

if(top==size-1)
{
printf("\nStack is full");
return;
}
printf("\nEnter item:\n");
scanf("%d",&item);
s[++top]=item;
}

void pop()
{
if(top== -1)
{
printf("\nstack is empty");
return;
}
printf("\nDeleted item is: %d",s[top]);
top--;
}

void main()
{
int ch;
top=-1;
clrscr();
printf("\n1.push\t2.pop\n3.display\t4.exit\n");
do{
printf("\nEnter your choice:\n");
scanf("%d",&ch);
switch(ch)
{
case 1:// printf("Enter item:\n");

```



```
//scanf("%d",&item);
push();
break;
case 2: pop();
break;
case 3: display();
break;
case 4: exit(0);
default: printf("\nWrong entry ! try again");
}
}while(ch<=4);
getch();
}
```

OUTPUT

```
1.push 2.pop
3.display    4.exit
Enter your choice:
1
Enter item:
100
Enter your choice:
1
Enter item:
200
Enter your choice:
1
Enter item:
300
Enter your choice:
2
```

Deleted item is: 300

Enter your choice:

3

Content of stack is:

100 200

Enter yourchoice:4

RESULT

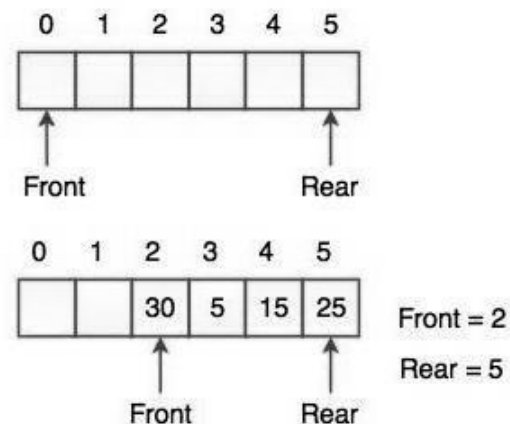
Thus the C program to implement stack using array was executed successfully.

AIM

To write a C program to implement Queue operations such as enqueue, dequeue and display using array.

PRE LAB DISCUSSION

Queue implemented using array can store only fixed number of data values. The implementation of queue data structure using array is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using FIFO (First In First Out) principle with the help of variables 'front' and 'rear'. Initially both 'front' and 'rear' are set to -1. Whenever, we want to insert a new value into the queue, increment 'rear' value by one and then insert at that position. Whenever we want to delete a value from the queue, then increment 'front' value by one and then display the value at 'front' position as deleted element.



- The Front and Rear of the queue point at the first index of the array. (Array index starts from 0).
- While adding an element into the queue, the Rear keeps on moving ahead and always points to the position where the next element will be inserted. Front remains at the first index.

ALGORITHM

Step 1: Start.

Step 2: Initialize front=0; rear=-1.

Step 3: Enqueue operation moves a rear by one position and inserts a element at the rear.

Step 4: Dequeue operation deletes a element at the front of the list and moves the front by one Position.

Step 5: Display operation displays all the element in the list.

Step 6: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define SIZE5      /* Size of Queue*/
int Q[SIZE],f=0,r=-1; /* Global declarations*/
Qinsert(int elem)
{
    /* Function for Insert operation*/
    if(Qfull())
        printf("\n\n Overflow!!!!\n\n");
    else
    {
        ++r;
        Q[r]=elem;
    }
}

int Qdelete()
{
    /* Function for Delete operation*/
    int elem;
    if(Qempty()){ printf("\n\n Underflow!!!!\n\n");
        return(-1); }
    else
    {
        elem=Q[f];
```

```

        f=f+1;
        return(elem);
    }
}

```

```

int Qfull()
{
    /* Function to Check Queue Full*/
    if(r==SIZE-1) return 1;
    return 0;
}

```

```

int Qempty()
{
    /* Function to Check Queue Empty*/
    if(f > r) return 1;
    return 0;
}

```

```

display()
{
    /* Function to display status of Queue*/
    inti;
    if(Qempty()) printf(" \n Empty Queue\n");
    else
    {
        printf("Front->");
        for(i=f;i<=r;i++)
            printf("%d ",Q[i]);
        printf("<-Rear");
    }
}

```

```

void main()
{
    /* Main Program */
}

```

```

int opn,elem;
do
{
    clrscr();
    printf("\n ### Queue Operations using Arrays### \n\n");
    printf("\n Press 1-Insert, 2-Delete,3-Display,4-Exit\n");
    printf("\n Your option ? ");
    scanf("%d",&opn);
    switch(opn)
    {
        case 1: printf("\n\nRead the element to be Inserted ?");
            scanf("%d",&elem);
            Qinsert(elem); break;
        case 2: elem=Qdelete();
            if( elem != -1)
                printf("\n\nDeleted Element is %d \n",elem);
            break;
        case 3: printf("\n\nStatus of Queue\n\n");
            display(); break;
        case 4: printf("\n\n Terminating \n\n"); break;
        default: printf("\n\nInvalid Option !!! Try Again !! \n\n");
            break;
    }
    printf("\n\n\n Press a Key to Continue . . . ");
    getch();
}while(opn != 4);
getch();
}

```

OUTPUT

```

### Queue Operations using Arrays###
Press 1-Insert, 2-Delete,3-Display,4-Exit

```

```
Your option ? 1
Read the element to be Inserted ?100
Press a Key to Continue . . .
#### Queue Operations using Arrays####
Press 1-Insert, 2-Delete,3-Display,4-Exit
Your option ? 1
Read the element to be Inserted ?200
Press a Key to Continue . . .
#### Queue Operations using Arrays####
Press 1-Insert, 2-Delete,3-Display,4-Exit
Your option ? 1
Read the element to be Inserted ?300
Press a Key to Continue . . .
#### Queue Operations using Arrays####
Press 1-Insert, 2-Delete,3-Display,4-Exit
Your option ? 2
Deleted Element is 100
Press a Key to Continue . . .

#### Queue Operations using Arrays####
Press 1-Insert,2-Delete,3-Display,4-Exit
Your option ?3
Status ofQueue
Front->200 300 <-Rear
Press a Key to Continue . . .
```

RESULT

Thus the C program to implement Queue using array was completed successfully.

Ex.No:2

ARRAY IMPLEMENTATION OF LIST ADT

AIM

To write a C program to implement list using an array

PRE LAB DISCUSSION

Array of list is an important data structure used in many applications. It is an interesting structure to form a useful data structure. It combines static and dynamic structure. Static means array and dynamic means linked list used to form a useful data structure. Array elements can be stored in consecutive manner in memory. Insert and delete operation takes more time in array. Array elements cannot be added, deleted once it is declared. In array, elements can be modified easily by identifying the index value. Pointer cannot be used in array. So, it does not require extra space in memory for pointer.

ALGORITHM

Step 1: Start.

Step 2: Declare the necessary functions for implementation.

Step 3: Get the input from the user and store it in an array.

Step 4: In Insertion, half of the elements to be shifted upwards and in deletion half of the elements to be shifted downwards.

Step 5: Display the output using an array.

Step 6: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
void create();
void insert();
void deletion();
void search();
void display();
int a,b[20], n, p, e, f, i, pos;
void main()
{
```

```

//clrscr();
int ch;
char g='y';
do
{
printf("\n main Menu");
printf("\n 1.Create \n 2.Delete \n 3.Search \n 4.Insert \n 5.Display\n 6.Exit \n");
printf("\n Enter your Choice");
scanf("%d", &ch);

switch(ch)
{
case 1:
create();
break;

case 2:
deletion();
break;

case 3:
search();
break;

case 4:
insert();
break;

case 5:
display();
break;

case 6:
exit();
break;

default:
printf("\n Enter the correctchoice:");
}
printf("\n Do u want tocontinue::");
scanf("\n%c", &g);
}
while(g=='y'||g=='Y');
getch();
}

void create()

```

```

{
printf("\n Enter the number of nodes");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\n Enter the Element:",i+1);
scanf("%d", &b[i]);
}
}

void deletion()
{
printf("\n Enter the position u want to delete::");
scanf("%d", &pos);
if(pos>=n)
{
printf("\n Invalid Location::");
}
else
{
for(i=pos+1;i<n;i++)
{
b[i-1]=b[i];
}
n--;
}
printf("\n The Elements after deletion");
for(i=0;i<n;i++)
{
printf("\t%d", b[i]);
}
}

void search()
{
printf("\n Enter the Element to be searched:");
scanf("%d", &e);

for(i=0;i<n;i++)
{
if(b[i]==e)
{
printf("Value is in the %d Position", i);
}
else
{
printf("Value %d is not in the list::", e);
}
}
}

```



```

continue;
}
}
}

void insert()
{
printf("\n Enter the position u need to insert::");
scanf("%d", &pos);

if(pos>=n)
{
printf("\n invalid Location::");
}
else
{
for(i=MAX-1;i>=pos-1;i--)
{
b[i+1]=b[i];
}
printf("\n Enter the element to insert::\n");
scanf("%d",&p);
b[pos]=p;
n++;
}
printf("\n The list after insertion::\n");

display();
}

void display()
{
printf("\n The Elements of The list ADT are:");
for(i=0;i<n;i++)
{
printf("\n\n%d", b[i]);
}
}
}

```

OUTPUT

Main Menu

- 1.Create
- 2.Delete
- 3.Search
- 4.Insert

5.Display

6.Exit

Enter your Choice1

Enter the number of nodes 2

Enter theElement:2

Enter theElement:3

Do u want to continue:::y

Main Menu

1.Create

2.Delete

3.Search

4.Insert

5.Display

6.Exit

Enter your Choice 5

The Elements of The list ADT are:

2

3

Do u want to continue:::y

Main Menu

1.Create

2.Delete

3.Search

4.Insert

5.Display

6.Exit

Enter your Choice 4

Enter the position u need to insert::0

Enter the element to inert::5

The list after insertion::

The Elements of the list ADT are:

5

2

3

Do u want to continue:::y

Main Menu

1.Create

2.Delete

3.Search

4.Insert

5.Display

6.Exit

Enter your Choice 2

Enter the position u want to delete::1

The Elements after deletion

5

3

Do u want to continue:::

RESULT

Thus the C program to implement list using array was completed successfully.

Ex.No: 3A

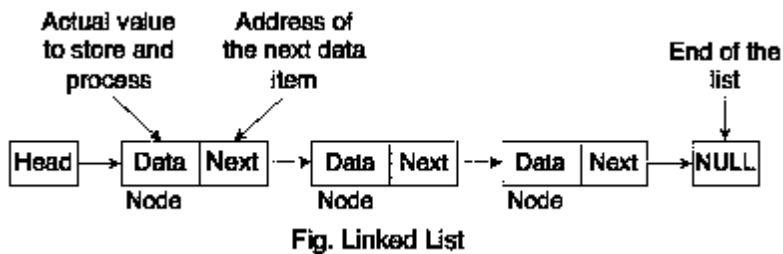
LINKED LIST IMPLEMENTATION OF LIST ADT

AIM

To write a C program to implement singly linked list.

PRE LAB DISCUSSION

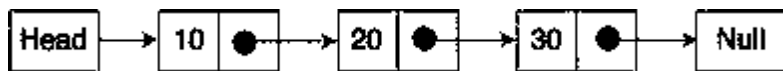
Linked list is a linear data structure. It is a collection of data elements, called nodes pointing to the next node by means of a pointer. In linked list, each node consists of its own data and the address of the next node and forms a chain.



Linked list contains a link element called first and each link carries a data item. Entry point into the linked list is called the head of the list. Link field is called next and each link is linked with its next link. Last link carries a link to null to mark the end of the list.

Linked list is a dynamic data structure. While accessing a particular item, start at the head and follow the references until you get that data item.

Linked list is used while dealing with an unknown number of objects:



Linked list contains two fields - First field contains value and second field contains a link to the next node. The last node signifies the end of the list that means NULL. The real life example of Linked List is that of Railway Carriage. It starts from engine and then the coaches follow. Coaches can traverse from one coach to other, if they connected to each other.

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,*y,*ptr,*pt;

void create( void );
void insert( void );
void delet( void );
void display ( void );
int j,pos,k=1,count;
void main()
{
    int n,i=1,opt;
    clrscr();
    p = NULL;
    printf("%d",sizeof(LIST));
    printf( "Enter the no of nodes :\n " );
    scanf( "%d",&n );
    count = n;
    while( i <= n)
    {
        create();
```

```

i++;
}
printf("\nEnter your option:\n");
printf("1.Insert \t 2.Delete \t 3.Display \t 4.Exit\n");
do
{
scanf("%d",&opt);
switch( opt )
{
case 1:
insert();
count++;
break;
case 2:
delet();
count--;
if ( count == 0 )
{
printf("\n List is empty\n");
}
break;

case 3:
printf("List elements are:\n");
display();
break;
}
printf("\nEnter your option \n");
}while( opt != 4 );
getch();
}

void create ( )
{
if( p== NULL )
{
p = ( LIST * ) malloc ( sizeof ( LIST ) );
printf( "Enter the element:\n" );
scanf( "%d",&p->no );
p->next = NULL;
h = p;
}
else
{
t= ( LIST * ) malloc (sizeof( LIST ));
printf( "\nEnter the element" );
scanf( "%d",&t->no );
t->next = NULL;
p->next = t;

```

```
p = t;
}
}
```

```
void insert()
{
    t=h;
    p = ( LIST * ) malloc ( sizeof(LIST) );
    printf("Enter the element to be inserted:\n");
    scanf("%d",&p->no);
    printf("Enter the position to insert:\n");
    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;
    }
}
```

```
void delet(){
    printf("Enter the position to delete:\n");
    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;
    }
}
```

```
void display()
{
    t= h;
```



```

while( t->next != NULL )
{
printf("\t%d",t->no);
t = t->next;
}
printf( "\t %d\t",t->no );
}

```

OUTPUT

```

Enter the no of nodes: 3
Enter the element: 1
Enter the element 2
Enter the element 3
Enter your option:
1. Insert 2.Delete 3.Display 4.Exit
3
List elements are:
1 2 3
Enter your option 1
Enter the element to be inserted:
12
Enter the position to insert: 1
Enter your option 3
List elements are:
12 1 2 3
Enter your option 1
Enter the element to be inserted:
13
Enter the position to insert: 3
Enter your option 1
Enter the element to be inserted:
14
Enter the position to insert:6
Enter your option 3
List elements are:
12 1 13 2 3 14
Enter your option2
Enter the position todelete:1
Enter your option3
List elements are:
1 13 2 3 14
Enter your option2
Enter the position todelete:3
Enter your option3
List elements are:
1 13 3 14

```

Enter your option2
Enter the position todelete:4
Enter your option3
List elements are:
1 13 3
Enter your option:6

RESULT

Thus the C program to implement List was completed successfully.

Ex. No:3B**LINKED LIST IMPLEMENTATION OF STACK****AIM**

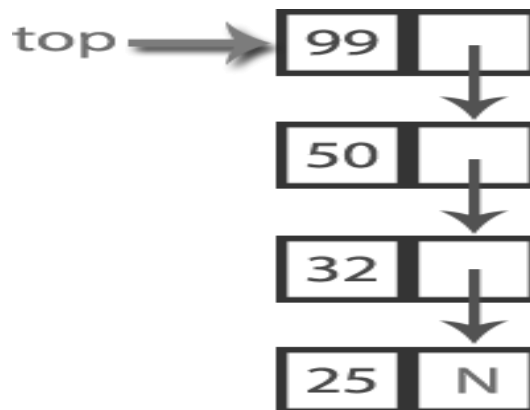
To write a C program to implement Stack operations such as push, pop and display using linked list.

PRE LAB DISCUSSION

The major problem with the stack implemented using array is, it works only for fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using array is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using linked list data structure. The stack implemented using linked list can work for unlimited number of values. That means, stack implemented using linked list works for variable size of data. So, there is no need to fix the size at the beginning of the implementation. The Stack implemented using linked list can organize as many data values as we want.

In linked list implementation of a stack, every new element is inserted as 'top' element. That means every newly inserted element is pointed by 'top'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by 'top' by moving 'top' to its next node in the list. The next field of the first element must be always NULL.

Example



In above example, the last inserted node is 99 and the first inserted node is 25. The order of elements inserted is 25, 32, 50 and 99.

There are two basic operations performed in a Stack:

1. Push(): is used to add or insert new elements into the stack.
2. Pop(): is used to delete or remove an element from the stack.

ALGORITHM

Step 1: Start.

Step 2: push operation inserts an element at the front.

Step 4: pop operation deletes an element at the front of the list;

Step 5: display operation displays all the elements in the list.

Step 6: Stop.

PROGRAM

```
#include<stdio.h>

#include<stdlib.h>
#include<conio.h>
void pop();
void push(int value);
void display();
struct node
{
    int data;
    struct node *link;
};
struct node *top=NULL,*temp;
void main()
{
    int choice,data;
    while(1) //infinite loop is used to insert/delete infinite number of elements in stack
    {
        printf("\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
        printf("\nEnter ur choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: //To push a new element into stack
                printf("Enter a new element :");
```

```

        scanf("%d",&data);
        push(data);
        break;
case 2: // pop the element from stack
        pop();
        break;
case 3: // Display the stack elements
        display();
        break;
case 4: // To exit
        exit(0);
    }
}
getch();
//return 0;
}

void display()
{
    temp=top;
    if(temp==NULL)
    {
        printf("\nStack is empty\n");
    }
    printf("\n The Contents of the Stackare...");
    while(temp!=NULL)
    {
        printf(" %d->",temp->data);
        temp=temp->link;
    }
}

void push(int data)
{

```

```

        temp=(struct node *)malloc(sizeof(struct node)); // creating a space for the new
element.
        temp->data=data;
        temp->link=top;
        top=temp;
display();
}
void pop()
{
    if(top!=NULL)
    {
        printf("The popped element is %d",top->data);
        top=top->link;
    }
    else
    {
        printf("\nStack Underflow");
    }
display();
}

```

OUTPUT

```

1.Push
2.Pop
3.Display
4.Exit
Enter ur choice:1
Enter a new element :10
The Contents of the Stack are... 10 ->

```

```

1.Push
2.Pop

```

3.Display

4.Exit

Enter ur choice:1

Enter a new element :20

The Contents of the Stack are... 20 -> 10 ->

1.Push

2.Pop

3.Display

4.Exit

Enter ur choice:1

Enter a new element :30

The Contents of the Stack are... 30 -> 20 -> 10 ->

1.Push

2.Pop

3.Display

4.Exit

Enter ur choice:2

The popped element is 30

The Contents of the Stack are... 20 -> 10 ->

1.Push

2.Pop

3.Display

4.Exit

RESULT

Thus the C program to implement Stack using linked list was completed successfully.

Ex. No: 3C

LINKED LIST IMPLEMENTATION OF QUEUE

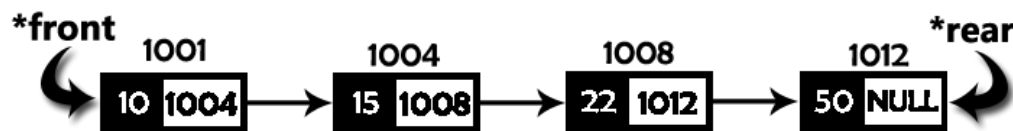
AIM

To write a C program to implement Queue operations such as enqueue, dequeue and display using linked list.

PRE LAB DISCUSSION

The major problem with the queue implemented using array is, It will work for only fixed number of data. That means the amount of data must be specified in the beginning itself. Queue using array is not suitable when we don't know the size of data which we are going to use. A queue data structure can be implemented using linked list data structure. The queue which is implemented using linked list can work for unlimited number of values. That means, queue using linked list can work for variable size of data (No need to fix the size at beginning of the implementation). The Queue implemented using linked list can organize as many data values as we want. In linked list implementation of a queue, the last inserted node is always pointed by '**rear**' and the first node is always pointed by '**front**'.

Example



In above example, the last inserted node is 50 and it is pointed by '**rear**' and the first inserted node is 10 and it is pointed by '**front**'. The order of elements inserted is 10, 15, 22 and 50.

There are two basic operations performed on a Queue.

Enqueue(): This function defines the operation for adding an element into queue. Dequeue(): This function defines the operation for removing an element from queue.

ALGORITHM

Step 1: Start.

Step 2: Enqueue operation inserts an element at the rear of the list.

Step 4: Dequeue operation deletes an element at the front of the list.

Step 5: Display operation display all the element in the list.

Step 6: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct Node
{
    int data;
    struct Node *next;
}*front = NULL,*rear = NULL;

void insert(int);
void delete();
void display();

void main()
{
    int choice, value;
    clrscr();
    printf("\n:: Queue Implementation using Linked List ::\n");
    while(1){
        printf("\n***** MENU *****\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: printf("Enter the value to be insert: ");
                    scanf("%d", &value);
                    insert(value);
                    break;
            case 2: delete(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("\nWrong selection!!! Please try again!!!\n");
        }
    }
}
```

```

    }
}

void insert(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode -> next = NULL;
    if(front == NULL)
        front = rear = newNode;
    else{
        rear -> next = newNode;
        rear = newNode;
    }
    printf("\nInsertion is Success!!!\n");
}

void delete()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else{
        struct Node *temp = front;
        front = front -> next;
        printf("\nDeleted element: %d\n", temp->data);
        free(temp);
    }
}

void display()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else{
        struct Node *temp = front;
        while(temp->next != NULL){

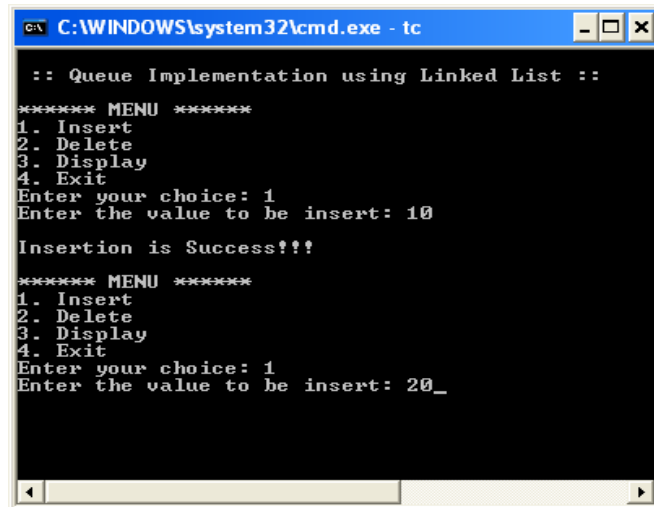
```

```

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

```

OUTPUT



```

C:\WINDOWS\system32\cmd.exe - tc

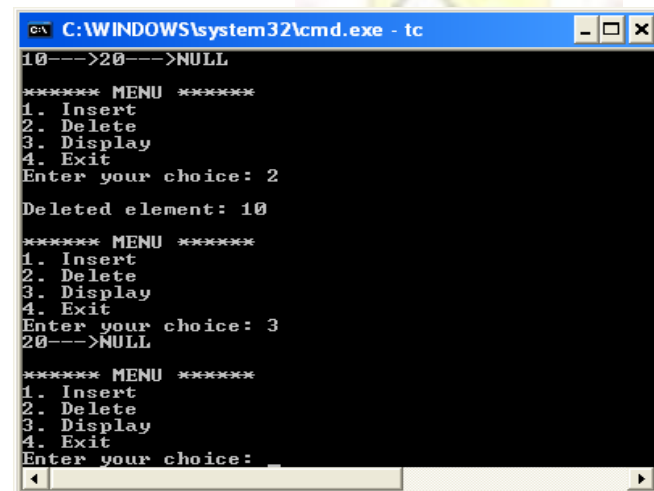
:: Queue Implementation using Linked List ::

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 10

Insertion is Success!!!

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20_

```



```

C:\WINDOWS\system32\cmd.exe - tc

10--->20--->NULL

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2

Deleted element: 10

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
20--->NULL

***** MENU *****
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: _

```

RESULT

Thus the C program to implement Queue using linked list was completed successfully.

Ex.No:4A

APPLICATIONS OF LIST – POLYNOMIAL MANIPULATION

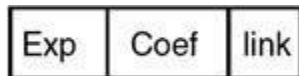
AIM

To write a 'C' program to represent a polynomial as a linked list and write functions for polynomial addition

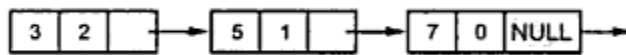
PRE LAB DISCUSSION

A **polynomial equation** is an **equation** that can be written in the form. $ax^n + bx^{n-1} + \dots + rx + s = 0$, where a, b, \dots, r and s are constants. We call the largest exponent of x appearing in a nonzero term of a **polynomial** the degree of that **polynomial**. A Polynomial has mainly two fields Exponent and coefficient.

Node of a Polynomial:



For example $3x^2 + 5x + 7$ will represent as follows.



In each node the exponent field will store the corresponding exponent and the coefficient field will store the corresponding coefficient. Link field points to the next item in the polynomial. Given two polynomial numbers represented by a linked list. Write a function that add these lists means add the coefficients which have same variable powers.

Example:

Input:

$$\text{1st number} = 5x^2 + 4x^1 + 2x^0$$

$$\text{2nd number} = 5x^1 + 5x^0$$

Output:

$$5x^2 + 9x^1 + 7x^0$$

ALGORITHM

Step1:Start the program

Step2:Get the coefficients and powers for the two polynomials to be added.

Step3:Add the coefficients of the respective powers.

Step4:Display the added polynomial.

Step5:Terminate the program.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct polynomial
{
    int coff;
    int pow;
    struct polynomial *link;
}*ptr,*start1,*node,*start2,*start3,*ptr1,*ptr2;
typedef struct polynomial pnl;
int temp1,temp2;

void main()
{
    void create(void);
    void prnt(void);
    void suml(void);
    void sort(void);
    clrscr();
    printf("Enrtter the elements of the first polynomial :");
    node = (pnl *) malloc(sizeof (pnl));
    start1=node;
    if (start1==NULL)
    {
        printf(" Unable to create memory.");
        getch();
        exit();
    }
    create();
    printf("Enter the elements of the second poly :");
    node = (pnl *) malloc(sizeof (pnl));
    start2=node;
    if (start2==NULL)
    {
        printf("Unable to create memory.");
        getch();
        exit();
    }
    create();
    clrscr();
    //printing the elements of the lists
    printf("The elements of the poly first are :");
    ptr=start1;
    print();
    printf("The elements of the poly second are :");
    ptr=start2;
```

```

    print();
    printf("The first sorted list is :");
    ptr=start1;
    sort();
    ptr=start1;
    print();
    printf("The second sorted list is :");
    ptr=start2;
    sort();
    ptr=start2;
    print();
    printf("The sum of the two lists are :");
    suml();
    ptr=start3;
    print();
    getch();
}
/* ..... */
void create()
{
    char ch;
    while(1)
    {
        printf(" Enter the coff and pow :");
        scanf("%d%d",&node->coff,&node->pow);
        if (node->pow==0 )
        {
            ptr=node;
            node=(pnl *)malloc(sizeof(pnl));
            node=NULL;
            ptr->link=node;
            break;
        }
        printf("Do u want enter more coff ?(y/n)");
        fflush(stdin);
        scanf("%c",&ch);
        if (ch=='n' )
        {
            ptr=node;
            node=(pnl *)malloc(sizeof(pnl));
            node=NULL;
            ptr->link=node;
            break;
        }
        ptr=node;
        node=(pnl *)malloc(sizeof(pnl));
        ptr->link=node;
    }
}

```



```

    }
}
/* ..... */
void print()
{
    int i=1;
    while(ptr!=NULL )
    {
        if(i!=1)
            printf("+ ");
        printf(" %dx^%d\n ",ptr->coff,ptr->pow);
        ptr=ptr->link;
        i++;
    }
    //printf(" %d^%d",ptr->coff,ptr->pow);
}
/* ..... */
void sort()
{
    for(;ptr->coff!=NULL;ptr=ptr->link)
        for(ptr2=ptr->link;ptr2->coff!=NULL;ptr2=ptr2->link)
        {
            if(ptr->pow>ptr2->pow)
            {
                temp1=ptr->coff;
                temp2=ptr->pow;
                ptr->coff=ptr2->coff;
                ptr->pow=ptr2->pow;
                ptr2->coff=temp1;
                ptr2->pow=temp2;
            }
        }
}
/* ..... */
void suml()
{
    node=(pnl *)malloc (sizeof(pnl));
    start3=node;

    ptr1=start1;
    ptr2=start2;

    while(ptr1!=NULL && ptr2!=NULL)
    {
        ptr=node;
        if (ptr1->pow > ptr2->pow )
        {

```

```

node->coff=ptr2->coff;
node->pow=ptr2->pow;
ptr2=ptr2->link; //update ptr list B
}
else if ( ptr1->pow < ptr2->pow )
{
node->coff=ptr1->coff;
node->pow=ptr1->pow;
ptr1=ptr1->link; //update ptr list A
}
else
{
node->coff=ptr2->coff+ptr1->coff;
node->pow=ptr2->pow;
ptr1=ptr1->link; //update ptr list A
ptr2=ptr2->link; //update ptr listB
}

node=(pnl *)malloc (sizeof(pnl));
ptr->link=node; //update ptr listC
} //end of while

if(ptr1==NULL) //end of listA
{
while(ptr2!=NULL)
{
node->coff=ptr2->coff;
node->pow=ptr2->pow;
ptr2=ptr2->link; //update ptr list B
ptr=node;
node=(pnl *)malloc (sizeof(pnl));
ptr->link=node; //update ptr list C
}
}
elseif(ptr2==NULL) //end of listB
{
while(ptr1!=NULL)
{
node->coff=ptr1->coff;
node->pow=ptr1->pow;
ptr1=ptr1->link; //update ptr list B
ptr=node;
node=(pnl *)malloc (sizeof(pnl));
ptr->link=node; //update ptr list C
}
}
node=NULL;

```

```
ptr->link=node;
}
```

OUTPUT

```
Enter the elements of the first polynomial : Enter the coff and pow :1 1
Do u want enter more coff?(y/n)y
Enter the coeff and pow :1 0
Enter the elements of the second poly : Enter the coff and pow :1 1
Do u want enter more coff?(y/n)y
Enter the coeff and pow :2 0
The elements of the poly first are :  $1x^1 + 1x^0$ 
The elements of the poly second are :  $1x^1 + 2x^0$ 
The first sorted list is :  $1x^0 + 1x^1$ 
The second sorted list is :  $2x^0 + 1x^1$ 
The sum of the two lists are :  $3x^0 + 2x^1$ 
```

RESULT

Thus the C program to implement Polynomial using linked list was completed successfully.

Ex.No:4B

APPLICATIONS OF STACK – INFIX TO POSTFIX CONVERSION

AIM

To write a C program to implement the conversion of infix to postfix expression using Stack.

PRE LAB DISCUSSION

One of the applications of Stack is in the conversion of arithmetic expressions in high-level programming languages into machine readable form. An arithmetic expression may consist of more than one operator and two operands e.g. $(A+B)*C(D/(J+D))$. These complex arithmetic operations can be converted into polish notation using stacks which then can be executed in two operands and an operator form.

Infix Expression:

It follows the scheme of **<operand><operator><operand>** i.e. an **<operator>** is preceded and succeeded by an **<operand>**. Such an expression is termed infix expression. E.g., **A+B**

Postfix Expression:

It follows the scheme of **<operand><operand><operator>** i.e. an **<operator>** is succeeded by both the **<operand>**. E.g., **AB+**

Steps to convert Infix To Postfix

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1. Push “(“ onto Stack, and add “)” to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered, then:
 1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
 2. Add operator to Stack.[End of If]
6. If a right parenthesis is encountered, then:
 1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
 2. Remove the left Parenthesis.[End of If]
[End of If]
7. END.

ALGORITHM

Step 1: Start.

Step 2: Create a stack to store operand and operator.

Step 3: In Postfix notation the operator follows the two operands and in the infix notation the operator is in between the two operands.

Step 4: Consider the sum of A and B. Apply the operator “+” to the operands A and B and write the sum as A+B is INFIX. + AB is PREFIX. AB+ is POSTFIX

Step 5: Get an Infix Expression as input and evaluate it by first converting it to postfix and then evaluating the postfix expression.

Step 6: The expressions with in innermost parenthesis must first be converted to postfix so that they can be treated as single operands. In this way Parentheses can be successively eliminated until the entire expression is converted.

Step 7: The last pair of parentheses to be opened with in a group of parentheses encloses the first expression with in that group to be transformed. This last-in first-out immediately suggests the use of Stack. Precedence plays an important role in the transforming infix to postfix.

Step 8: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 20
int top=-1;

char pop();
char stack[MAX];
void push(char item);

int prcd(char symbol){
switch(symbol){
case '+':
```

```

    case '-':return 2;
    break;
    case '*':
    case '/':return 4;
    break;

    case '^':
    case '$':return 6;
    break;
    case '(':
    case ')':
    case '#':return 1;
    break;
}
}

int isoperator(char symbol){
    switch(symbol){
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
        case '$':
        case '(':
        case ')':return 1;
        break;
        default:
        return 0;
    }
}

void convertip(char infix[],char postfix[]){
    int i,symbol,j=0;

```

```
stack[++top]='#';
for(i=0;i<strlen(infix);i++)
{
symbol=infix[i];
if(isoperator(symbol)==0)
{
postfix[j]=symbol;
j++;
}
else{
if(symbol=='(')
push(symbol);
else if(symbol==')'){
while(stack[top]!='('){
postfix[j]=pop();
j++;
}
pop();//pop out (.
}
else{
if(prcd(symbol)>prcd(stack[top]))
push(symbol);
else{
while(prcd(symbol)<=prcd(stack[top])){
postfix[j]=pop();
j++;
}
push(symbol);
} //end of else.
} //end of else.
} //end of else.
} //end of for.
```

```

while(stack[top]!='#'){
postfix[j]=pop();
j++;
}
postfix[j]='\0';//null terminate string.
}

void main()
{
char infix[20],postfix[20];
clrscr();
printf("Enter the valid infix string:\n");
gets(infix);
convertip(infix,postfix);
printf("The corresponding postfix string is:\n");
puts(postfix);
getch();
}

void push(char item)

{

top++;
stack[top]=item;
}

char pop()

{

char a;
a=stack[top];
top--;
return a;
}

```


OUTPUT

Enter the valid infix string:

$(a+b)*c$

The corresponding postfix string is:

$ab+c*$

RESULT

Thus the C program to convert infix to postfix expression using Stack was completed successfully.

Ex.No:4C**APPLICATIONS OF QUEUE – FCFS****AIM**

To schedule snapshot of processes queued according to FCFS scheduling.

Process Scheduling

- Ø CPU scheduling is used in multiprogrammed operating systems.
- Ø By switching CPU among processes, efficiency of the system can be improved.
- Ø Some scheduling algorithms are FCFS, SJF, Priority, Round-Robin, etc.
- Ø Gantt chart provides a way of visualizing CPU scheduling and enables to understand better.

First Come First Serve (FCFS)

- Ø Process that comes first is processed first
- Ø FCFS scheduling is non-preemptive
- Ø Not efficient as it results in long average waiting time.
- Ø Can result in starvation, if processes at beginning of the queue have long bursts.

ALGORITHM

1. Define an array of structure *process* with members *pid*, *btime*, *wtime* & *ttime*.
2. Get length of the ready queue, i.e., number of process (say *n*)
3. Obtain *btime* for each process.
4. The *wtime* for first process is 0.
5. Compute *wtime* and *ttime* for each process as:
 - a. $wtime_{i+1} = wtime_i + btime_i$
 - b. $ttime_i = wtime_i + btime_i$
6. Compute average waiting time *awat* and average turnaround time *atur*
7. Display the *btime*, *ttime* and *wtime* for each process.
8. Display GANTT chart for the above scheduling
9. Display *awat* time and *atur*
10. Stop

PROGRAM

```
/* FCFS Scheduling- fcfs.c */
#include <stdio.h>
struct process
{
int pid; int btime; int wtime; int ttime;
} p[10];

main()
```

```

{
int i,j,k,n,ttur,twat; float awat,atur;

printf("Enter no. of process : "); scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("Burst time for process P%d (in ms) : ",(i+1)); scanf("%d", &p[i].btime);
p[i].pid = i+1;
}

p[0].wtime = 0; for(i=0; i<n; i++)
{
p[i+1].wtime = p[i].wtime + p[i].btime; p[i].ttime = p[i].wtime + p[i].btime;
}
ttur = twat = 0; for(i=0; i<n; i++)
{
ttur += p[i].ttime; twat += p[i].wtime;
}
awat = (float)twat / n; atur = (float)ttur / n;

printf("\nFCFS Scheduling\n\n"); for(i=0; i<28; i++)
printf("-");
printf("\nProcess B-Time T-Time W-Time\n"); for(i=0; i<28; i++)
printf("-");
for(i=0; i<n; i++)
printf("\nP%d\t%4d\t%3d\t%2d", p[i].pid,p[i].btime,p[i].ttime,p[i].wtime);
printf("\n"); for(i=0; i<28; i++)
printf("-");
printf("\n\nAverage waiting time: %5.2fms", awat);
printf("\n\nAverage turn around time : %5.2fms\n", atur);
printf("\n\nGANTT Chart\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n");
printf("|"); for(i=0; i<n; i++)
{

```

```

k = p[i].btime/2; for(j=0; j<k; j++)
printf(" "); printf("P%d",p[i].pid); for(j=k+1; j<p[i].btime; j++)
printf(" ");
printf("|");
}
printf("\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n");
printf("0");
for(i=0; i<n; i++)
{
for(j=0; j<p[i].btime; j++) printf(" ");
printf("%2d",p[i].ttime);
}
}

```

Output

\$ gcc fcfs.c

\$./a.out

Enter no. of process : 4

Burst time for process P1 (in ms) : 10 Burst time for process P2 (in ms) : 4 Burst time for process P3 (in ms) : 11 Burst time for process P4 (in ms) : 6

FCFS Scheduling

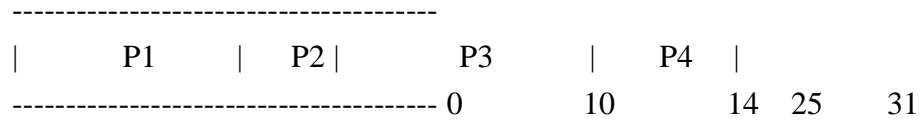
Process B-Time T-Time W-Time

| | | | |
|----|----|----|----|
| P1 | 10 | 10 | 0 |
| P2 | 4 | 14 | 10 |
| P3 | 11 | 25 | 14 |
| P4 | 6 | 31 | 25 |

Average waiting time : 12.25ms

Average turn around time : 20.00ms

GANTT Chart



RESULT

Thus the C program to convert FCFS Scheduling using Queue was completed successfully.

Ex.No:5

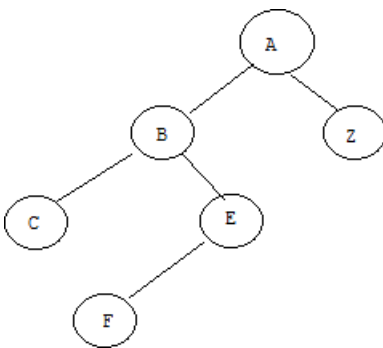
IMPLEMENTATION OF BINARY TREES AND OPERATIONS OF BINARY TREES

AIM

To write a C program to implement the binary trees and its operations.

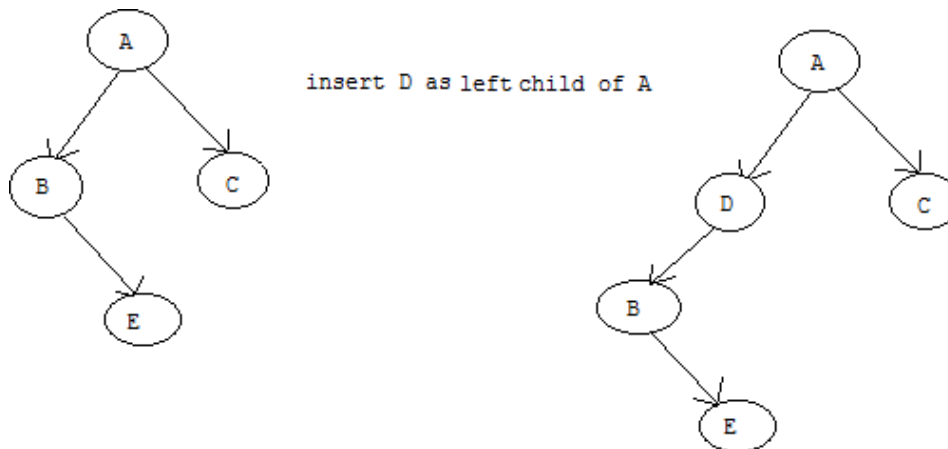
PRE LAB DISCUSSION

A **binary tree** is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.



Insertion:

In binary trees, a new node before insert has to specify 1) whose child it is going to be 2) mention whether new node goes as left/right child. For example(below image),



To add a new node to leaf node, a new node should also mention whether new node goes as left/right child.

Deletion:

For deletion, **only certain nodes** in a binary tree can be removed unambiguously.

Suppose that the node to delete is node A. If A has no children, deletion is accomplished by setting the child of A's parent to null. If A has one child, set the parent of A's child to A's parent and set the child of A's parent to A's child. In a binary tree, a node with two children **cannot** be deleted unambiguously.

ALGORITHM

Step 1: Start.

Step 2: Create a Binary Tree for N elements.

Step 3: Insert an element in binary tree

Step 4: Traverse the tree in inorder.

Step 5: Traverse the tree in preorder

Step 6: Traverse the tree in postorder.

Step 7: Stop

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct node
{
int data;
struct node *rlink;
struct node *llink;
}*tmp=NULL;
typedef struct node NODE;
NODE *create();
void preorder(NODE *);
void inorder(NODE *);
void postorder(NODE *);
void insert(NODE *);
void main()
{
int n,i,m;
clrscr();
do
{
printf("\n\n0.create\n\n1.insert \n\n2.preorder\n\n3.postorder\n\n4.inorder\n\n5.exit\n\n");
printf("\n\nEnter ur choice");
```

```

scanf("%d",&m);
switch(m)
{
case 0:
tmp=create();
break;
case 1:
insert(tmp);
break;
case 2:
printf("\n\nDisplay tree in Preorder traversal\n\n");
preorder(tmp);
break;
case 3:
printf("\n\nDisplay Tree in Postorder\n\n");
postorder(tmp);
break;
case 4:
printf("\n\nInorder\n\n");
inorder(tmp);
break;
case 5:
exit(0);
}
}
while(n!=5);

```

```

getch();
}
void insert(NODE *root)
{
NODE *newnode;
if(root==NULL)
{
newnode=create();
root=newnode;
}
else
{
newnode=create();
while(1)
{
if(newnode->data<root->data)
{
if(root->llink==NULL)
{
root->llink=newnode;

```



```

break;
}
root=root->llink;
}
if(newnode->data>root->data)
{
if(root->rlink==NULL)
{
root->rlink=newnode;
break;
}
root=root->rlink;
}
}
}
}
}
NODE *create()
{
NODE *newnode;
int n;
newnode=(NODE*)malloc(sizeof(NODE));
printf("\n\nEnter the Data ");
scanf("%d",&n);
newnode->data=n;
newnode->llink=NULL;
newnode->rlink=NULL;
return(newnode);
}

```

```

void postorder(NODE *tmp)

```

```

{
if(tmp!=NULL)
{
postorder(tmp->llink);
postorder(tmp->rlink);
printf("%d->",tmp->data);
}
}

```

```

void inorder(NODE *tmp)

```

```

{
if(tmp!=NULL)
{
inorder(tmp->llink);
printf("%d->",tmp->data);
}
}

```

```

inorder(tmp->rlink);
}
}
void preorder(NODE *tmp)
{
if(tmp!=NULL)
{
printf("%d->",tmp->data);
preorder(tmp->llink);
preorder(tmp->rlink);
}
}
}

```

OUTPUT

0.Create

1.insert

2.preorder

3.postorder

4.inorder

5.exit

Enter ur choice 0

Enter the Data 3

Enter ur choice 1

Enter the Data 7

Enter ur choice 1

Enter the Data 8

Enter ur choice 1

Enter the Data 6

Enter ur choice 1

Enter the Data 4

Enter ur choice2

Display tree in Preorder traversal

3->7->6->4->8

Enter ur choice3

Display tree in Postorder 4 -> 6 -> 8 -> 7 ->3

Enter ur choice4

Inorder

3 -> 4 -> 6 -> 7 -> 8

Enter ur choice 5

RESULT

Thus the C program to implement binary tree and its operation was completed successfully.

Ex.No:6

IMPLEMENTATION OF BINARY SEARCH TREES

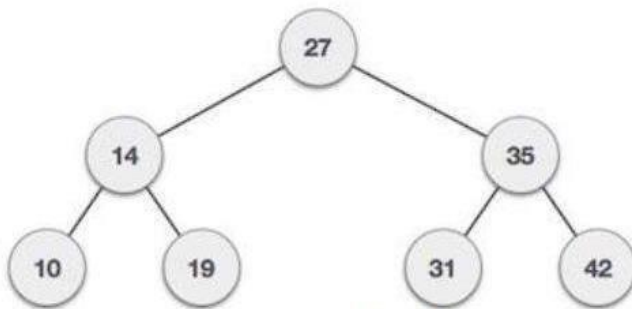
AIM

To write a C program to implement the binary search trees.

PRE LAB DISCUSSION

A **binary search tree** (BST) is a **tree** in which all nodes follows the below mentioned properties

- The left sub-**tree** of a node has key less than or equal to its parent node's key.
 - The right sub-**tree** of a node has key greater than or equal to its parent node's key. Thus, a binary search tree (BST) divides all its sub-trees into two segments; *left* sub-tree and *right* sub-tree and can be defined as
- $$\text{left_subtree (keys)} \leq \text{node (key)} \leq \text{right_subtree (keys)}$$



Following are basic primary operations of a tree which are following.

- **Search** – search an element in a tree.
- **Insert** – insert an element in a tree.
- **Delete** – removes an existing node from the tree
- **Preorder Traversal** – traverse a tree in a preorder manner.
- **Inorder Traversal** – traverse a tree in an inorder manner.
- **Postorder Traversal** – traverse a tree in a postorder manner.

ALGORITHM

- Step 1: Start the process.
- Step 2: Initialize and declare variables.
- Step 3: Construct the Tree
- Step 4: Data values are given which we call a key and a binary search tree
- Step 5: To search for the key in the given binary search tree, start with the root node and Compare the key with the data value of the root node. If they match, return the root pointer.
- Step 6: If the key is less than the data value of the root node, repeat the process by using the left subtree.
- Step 7: Otherwise, repeat the same process with the right subtree until either a match is found or the subtree under consideration becomes an empty tree.
- Step 8: Terminate

PROGRAM

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

struct tree
{
    int data;
    struct tree *lchild;
    struct tree *rchild;
} *t,*temp;

int element;
void inorder(struct tree *);
void preorder(struct tree *);
void postorder(struct tree *);
struct tree * create(struct tree *, int);
struct tree * find(struct tree *, int);
struct tree * insert(struct tree *, int);
struct tree * del(struct tree *, int);
struct tree * findmin(struct tree *);
struct tree * findmax(struct tree *);
void main()
{
    int ch;

    do
    {
        printf("\n\t\t\t\t\tBINARY SEARCH TREE");
        printf("\n\t\t\t\t***** ***** *****");
        printf("\nMain Menu\n");
        printf("\n1.Create\n2.Insert\n3.Delete\n4.Find\n5.FindMin\n6.FindMax");
        printf("\n7.Inorder\n8.Preorder\n9.Postorder\n10.Exit\n");
        printf("\nEnter ur choice :");
        scanf("%d",&ch);
        switch(ch)
        {
```

```

case1:
    printf("\nEnter the data:");
    scanf("%d",&element);
    t=create(t,element);
    inorder(t);
    break;
case2:
    printf("\nEnter the data:");
    scanf("%d",&element);
    t=insert(t,element);
    inorder(t);
    break;
case3:
    printf("\nEnter the data:");
    scanf("%d",&element);
    t=del(t,element);
    inorder(t);
case4: break;

    printf("\nEnter the data:");
    scanf("%d",&element);
    temp=find(t,element);
    if(temp->data==element)
        printf("\nElement %d is at %d",element,temp);
    else
        printf("\nElement is not found");
    break;
case 5:

    temp=findmin(t);
    printf("\nMax element=%d",temp->data);
    break;
case6:
    temp=findmax(t);
    printf("\nMax element=%d",temp->data);
    break;
case7:
    inorder(t);
case8: break;
    preorder(t);
case9: break;
    postorder(t);
    break;
case 10:exit(0);

```

```

    }
    }while(ch<=10);
}

```

```

struct tree * create(struct tree *t, int element)
{
    t=(struct tree *)malloc(sizeof(struct tree));
    t->data=element;
    t->lchild=NULL;
    t->rchild=NULL;
    return t;
}

```

```

struct tree * find(struct tree *t, int element)
{
    if(t==NULL)
        return NULL;
    if(element<t->data)
        return(find(t->lchild,element));
    else
        if(element>t->data)
            return(find(t->rchild,element));
        else
            return t;
}

```

```

struct tree *findmin(struct tree *t)
{
    if(t==NULL)
        return NULL;
    else
        if(t->lchild==NULL)
            return t;
        else
            return(findmin(t->lchild));
}

```

```

struct tree *findmax(struct tree *t)
{
    if(t!=NULL)
    {
        while(t->rchild!=NULL)
            t=t->rchild;
    }
    return t;
}

struct tree *insert(struct tree *t,int element)
{
    if(t==NULL)
    {
        t=(struct tree *)malloc(sizeof(struct tree));
        t->data=element;
        t->lchild=NULL;
        t->rchild=NULL;
        return t;
    }
    else
    {
        if(element<t->data)
        {
            t->lchild=insert(t->lchild,element);
        }
        else
        {
            if(element>t->data)
            {
                t->rchild=insert(t->rchild,element);
            }
            else
            {
                if(element==t->data)
                {
                    printf("element already present\n");
                }
                return t;
            }
        }
    }
}

struct tree * del(struct tree *t, int element)
{
    if(t==NULL)

```



```

        printf("element not found\n");
else
    if(element<t->data)
        t->lchild=del(t->lchild,element);
    else
        if(element>t->data)
            t->rchild=del(t->rchild,element);
        else
            if(t->lchild&& t->rchild)
            {
                temp=findmin(t->rchild);
                t->data=temp->data;
            }
            else
            {

```

```
t->rchild=del(t->rchild,t->data);
```

```
temp=t;
if(t->lchild==NULL)
    t=t->rchild;
    return t;
    }else

}
```

```
if(t->rchild==NULL)
    t=t->lchild; free(temp);
```

```
void inorder(struct tree *t)
{
    if(t==NULL)
        return;
    else
    {
        inorder(t->lchild);
        printf("\t%d",t->data);
        inorder(t->rchild);
    }
}
```

```
void preorder(struct tree *t)
{
    if(t==NULL)
        return;
    else
    {
        printf("\t%d",t->data);
        preorder(t->lchild);
        preorder(t->rchild);
    }
}
```

```
void postorder(struct tree *t)
{
    if(t==NULL)
        return;
    else
    {
        postorder(t->lchild);
```

```

        postorder(t->rchild);
        printf("\t%d",t->data);
    }
}

```

OUTPUT

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :1

Enter the data:10

10

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :2

Enter the data:20

10 20

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :2

Enter the data:30

10 20 30

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :2

Enter the data:25

10 20 25 30

BINARY SEARCH TREE

Main Menu

1.Create
2.Insert
3.Delete
4.Find
5.FindMin
6.FindMax
7.Inorder
8.Preorder
9.Postorder
10.Exit
Enter ur choice :4

Enter the data:25

Element 25 is at 2216

BINARY SEARCH TREE

Main Menu

1.Create
2.Insert
3.Delete
4.Find
5.FindMin
6.FindMax
7.Inorder
8.Preorder
9.Postorder
10.Exit

Enter ur choice :5

Max element=10

BINARY SEARCH TREE

Main Menu

1.Create
2.Insert
3.Delete
4.Find
5.FindMin
6.FindMax
7.Inorder
8.Preorder
9.Postorder
10.Exit

Enter ur choice :6

Max element=30

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :7

10 20 25 30

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :8

10 20 30 25

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :9

25 30 20 10

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :3

Enter the data:10

20 25 30

BINARY SEARCH TREE

Main Menu

- 1.Create
- 2.Insert
- 3.Delete
- 4.Find
- 5.FindMin
- 6.FindMax
- 7.Inorder
- 8.Preorder
- 9.Postorder
- 10.Exit

Enter ur choice :10

RESULT

Thus the C program to implement the binary search trees was completed successfully.

Ex.No:7

IMPLEMENTATION OF AVL TREES

AIM

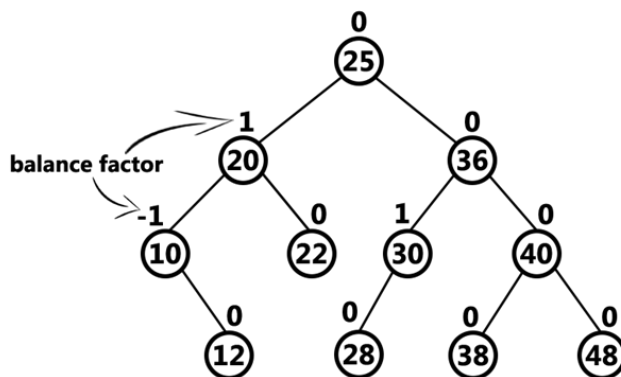
To write a C program to implement AVL trees.

PRE LAB DISCUSSION

AVL tree is a self balanced binary search tree. That means, an AVL tree is also a binary search tree but it is a balanced tree. A binary tree is said to be balanced, if the difference between the heights of left and right subtrees of every node in the tree is either -1, 0 or +1. In other words, a binary tree is said to be balanced if for every node, height of its children differ by at most one. In an AVL tree, every node maintains an extra information known as **balance factor**. An AVL tree is defined as follows...

An AVL tree is a balanced binary search tree. In an AVL tree, balance factor of every node is either -1, 0 or +1.

Balance factor of a node is the difference between the heights of left and right subtrees of that node. The balance factor of a node is calculated either **height of left subtree - height of right subtree** (OR) **height of right subtree - height of left subtree**.



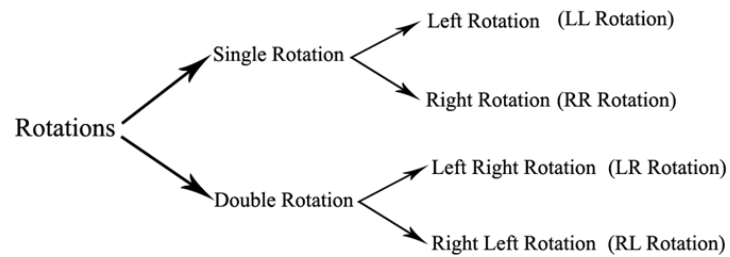
AVL Tree Rotations

In AVL tree, after performing every operation like insertion and deletion we need to check the balance factor of every node in the tree. If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced. We use **rotation** operations to make the tree balanced whenever the tree is becoming imbalanced due to any operation.

Rotation operations are used to make a tree balanced.

Rotation is the process of moving the nodes to either left or right to make tree balanced.

There are **four** rotations and they are classified into **two** types.



ALGORITHM

Step 1: Start

Step 2: Insert the new element into the tree using Binary Search Tree insertion logic.

Step 3: After insertion, check the Balance Factor of every node.

- If the Balance Factor of every node is 0 or 1 or -1 then go for next operation.
- If the Balance Factor of any node is other than 0 or 1 or -1 then tree is said to be imbalanced. Then perform the suitable Rotation to make it balanced. And go for next operation.

Step 4: Compare, the search element with the value of root node in the tree.

If search element is larger, then continue the search process in right subtree.

If we reach to the node with search value, then display "Element is found" and terminate the function.

Step 5: Stop

PROGRAM

```
#include<stdio.h>
#include<malloc.h>
typedef enum { FALSE ,TRUE } ;
struct node
{
    int info;
    int balance;
    struct node *lchild;
    struct node *rchild;
};
struct node *insert (int , struct node *, int *);
```

```

struct node* search(struct node *,int);

struct node* search(struct node *ptr,int info)
{
    if(ptr!=NULL)
        if(info < ptr->info)
            ptr=search(ptr->lchild,info);
        else if( info > ptr->info)
            ptr=search(ptr->rchild,info);
    return(ptr);
}/*End of search()*/

struct node *insert (int info, struct node *pptr, int *ht_inc)
{
    struct node *aptr;
    struct node *bptr;
    if(pptr==NULL)
    {
        pptr = (struct node *) malloc(sizeof(structnode));
        pptr->info =info;
        pptr->lchild = NULL;
        pptr->rchild = NULL;
        pptr->balance = 0;
        *ht_inc = TRUE;
        return (pptr);
    }
    if(info < pptr->info)
    {
        pptr->lchild = insert(info, pptr->lchild, ht_inc);
        if(*ht_inc==TRUE)
        {
            switch(pptr->balance)
            {
                case -1: /* Right heavy */

```

```

        pptr->balance = 0;
        *ht_inc = FALSE;
        break;
    case 0: /* Balanced */
        pptr->balance = 1;
        break;
    case 1: /* Left heavy */
        aptr = pptr->lchild;
        if(aptr->balance == 1)
        {
            printf("Left to Left Rotation\n");
            pptr->lchild = aptr->rchild;
            aptr->rchild = pptr;
            pptr->balance = 0;
            aptr->balance = 0;
            pptr = aptr;
        }
        else
        {
            printf("Left to right rotation\n");
            bptr = aptr->rchild;
            aptr->rchild = bptr->lchild;
            bptr->lchild = aptr;
            pptr->lchild = bptr->rchild;
            bptr->rchild = pptr;

            if(bptr->balance == 1 )
                pptr->balance = -1;
            else
                pptr->balance = 0;
            if(bptr->balance == -1)
                aptr->balance = 1;
            else
                aptr->balance = 0;
            bptr->balance = 0;
            pptr = bptr;
        }
        *ht_inc = FALSE;
    } /*End of switch */
} /*End of if */
} /*End of if */
if(info > pptr->info)
{
    pptr->rchild = insert(info, pptr->rchild, ht_inc);
    if(*ht_inc == TRUE)
    {

```

```

switch(pptr->balance)
{
case 1: /* Left heavy */
    pptr->balance = 0;
    *ht_inc = FALSE;
    break;
case 0: /* Balanced */
    pptr->balance = -1;
    break;
case -1: /* Right heavy */
    aptr = pptr->rchild;
    if(aptr->balance == -1)
    {
        printf("Right to Right Rotation\n");
        pptr->rchild = aptr->lchild;
        aptr->lchild = pptr;
        pptr->balance = 0;
        aptr->balance = 0;
        pptr = aptr;
    }
    else
    {
        printf("Right to Left Rotation\n");
        bptr = aptr->lchild;
        aptr->lchild = bptr->rchild;
        bptr->rchild = aptr;
        pptr->rchild = bptr->lchild;
        bptr->lchild = pptr;
        if(bptr->balance == -1)
            pptr->balance = 1;
        else
            pptr->balance = 0;
        if(bptr->balance == 1)
            aptr->balance = -1;
        else
            aptr->balance = 0;
        bptr->balance = 0;
        pptr = bptr;
    }
    /*End of else*/
    *ht_inc = FALSE;
} /*End of switch */
} /*End of if*/
} /*End of if*/

return(pptr);
} /*End of insert()*/

```

```

void display(struct node *ptr,int level)
{
    int i;
    if ( ptr!=NULL )
    {
        display(ptr->rchild, level+1);
        printf("\n");
        for (i = 0; i < level; i++)
            printf("  ");
        printf("%d", ptr->info);
        display(ptr->lchild, level+1);
    }/*End of if*/
}/*End of display()*/

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

main()
{
    int ht_inc;
    int info ;
    int choice;
    struct node *root = (struct node *)malloc(sizeof(struct node));
    root = NULL;

    while(1)
    {
        printf("1.Insert\n");
        printf("2.Display\n");
        printf("3.Quit\n");
        printf("Enter your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                printf("Enter the value to be inserted : ");
                scanf("%d", &info);
                if( search(root,info) == NULL )
                    root = insert(info, root, &ht_inc);

```

```

        else
            printf("Duplicate value ignored\n");
        break;
case 2:

    if(root==NULL)
    {
        printf("Tree is empty\n");
        continue;
    }
    printf("Tree is :\n");
    display(root, 1);
    printf("\n\n");
    printf("Inorder Traversal is: ");
    inorder(root);
    printf("\n");
case 3: break;

default: exit(1);

    printf("Wrong choice\n");

    }/*End of switch*/
}/*End of while*/
}/*End of main()*/

```

OUTPUT

```
Enter your choice : 1
Enter the value to be inserted : 4
1.Insert
2.Display
3.Quit
Enter your choice : 2
Tree is :
      12
     / 
    9  
   /  
  7   
 /    
4     

Inorder Traversal is: 4  7  9  12
1.Insert
2.Display
3.Quit
Enter your choice : 3_
```

RESULT

Thus the C program to implement AVL tree was completed successfully.

Ex.No:8

IMPLEMENTATION OF HEAPS USING PRIORITY QUEUES

AIM

To write a C program to implement the heaps using priority queues

PRE LAB DISCUSSION

A Priority Queue is an abstract data type to efficiently support finding an item with the highest priority across a series of operations. The basic operations are :

1. Insert
2. Find - minimum (or maximum),and
3. Delete-minimum (or maximum).

A heap is a specialized tree-based data structure that satisfies the heap property: if B is a child node of A, then $\text{key}(A) \geq \text{key}(B)$. This implies that an element with the greatest key is always in the root node, and so such a heap is sometimes called a max-heap. (Alternatively, if the comparison is reversed, the smallest element is always in the root node, which results in a min-heap). Heaps are usually implemented in an array, and don't require pointers between elements.

The operations commonly performed with a heap are :

- ☐ delete-max or delete-min : removing the root node of a max or min heap, respectively.
- ☐ increase-key or decrease-key : updating a key within a max or min heap, respectively.
- ☐ insert : adding a new key to the heap.
- ☐ merge : joining two heaps to form a valid new heap containing all the elements of both.

ALGORITHM

Step 1: Start

Step 2: Declaring the maximum size of the queue

Step 3: Lower bound of the array is initialized to 0

Step 4: Inserts an element in the queue

Step 5: Deletes an element from the queue

Step 6: Display the queue

Step 7 :Stop

PROGRAM

```
#include<stdio.h>
#include<math.h>
#define MAX 100/*Declaring the maximum size of the queue*/
void swap(int*,int*);
main()
{
int choice,num,n,a[MAX],data,s,lb;
void display(int[],int);
void insert(int[],int,int,int);
int del_hi_piori(int[],int,int);
n=0;/*Represents number of nodes in the queue*/
lb=0;/*Lower bound of the array is initialized to 0*/
while(1)
{
printf("\n.....MAINMENU .....");
printf("\n1.Insert.n");
printf("\n2.Delete.n");
printf("\n3.Display.n");
printf("\n4.Quit.n");
printf("\nEnter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1:/*choice to accept an element and insert it in the queue*/
printf("Enter data to be inserted : ");
scanf("%d",&data);
insert(a,n,data,lb);
n++;
break;
case2:
s=del_hi_piori(a,n+1,lb);
if(s!=0)
printf("\nThe deleted value is : %d n",s);
if(n>0)
n--;
break;
case 3:/*choice to display the elements of the queue*/
printf("\n");
display(a,n);
break;
case 4:/*choice to exit from the program*/
return;
default:
printf("Invalid choice.n");
```

```

}
printf("\n\n");
}
}
/*This function inserts an element in the queue*/
void insert(int a[],int heapsize,int data,int lb)
{
int i,p;
int parent(int);
if(heapsize==MAX)
{
printf("Queue Is Full!!\n");
return;
}
i=lb+heapsize;
a[i]=data;
while(i>lb&& a[p=parent(i)]<a[i])
{
swap(&a[p],&a[i]);
i=p;
}
}
/*This function deletes an element from the queue*/
int del_hi_piori(int a[],int heapsize,int lb)
{
int data,i,l,r,max_child,t;
int left(int);
int right(int);
if(heapsize==1)
{
printf("Queue Is Empty!!\n");
return 0;
}
t=a[lb];
swap(&a[lb],&a[heapsize-1]);
i=lb;
heapsize--;
while(1)
{
if((l=left(i))>=heapsize)
break;
if((r=right(i))>=heapsize)
max_child=l;
else
max_child=(a[l]>a[r])?l:r;
if(a[i]>=a[max_child])
break;

```

```

swap(&a[i],&a[max_child]);
i=max_child;
}
return t;
}
/*Returns parent index*/
int parent(int i)
{
float p;
p=((float)i/2.0)-1.0;
return ceil(p);
}
/*Returns leftchild index*/
int left(int i)
{
return 2*i+1;
}
/*Returns rightchild index*/
int right(int i)
{
return 2*i+2;
}
/*This function displays the queue*/
void display(int a[],int n)
{
int i;
if(n==0)
{
printf("Queue Is Empty!!\n");
return;
}
for(i=0;i<n;i++)
printf("%d ",a[i]);
printf("\n");
}
/*This function is used to swap two elements*/
void swap(int*p,int*q)
{
int temp;
temp=*p;
*p=*q;
*q=temp;
}

```

OUTPUT

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 1

Enter data to be inserted : 52

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 1

Enter data to be inserted : 63

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 1

Enter data to be inserted : 45

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 1

Enter data to be inserted : 2

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 1

Enter data to be inserted : 99

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.
Enter your choice : 3
99 63 45 2 52

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 2
The deleted value is : 99

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 3
63 52 45 2

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 2
The deleted value is : 63

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 2
The deleted value is : 52

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 3
45 2

.....MAIN MENU.....
1.Insert.
2.Delete.

3.Display.

4.Quit.

Enter your choice : 4

RESULT

Thus the 'C' program to implement priority queue using heaps was executed successfully.

Ex.No:9A

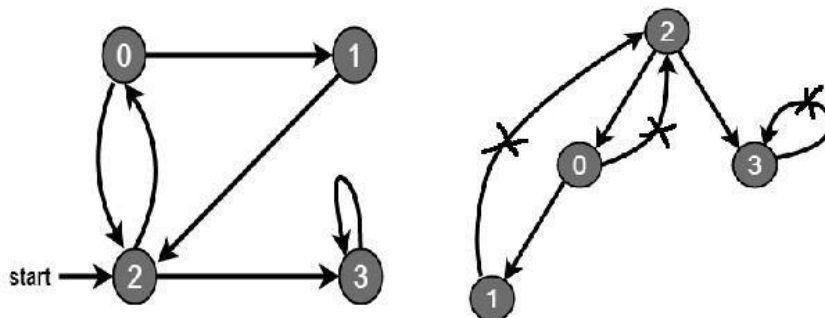
DEPTH FIRST SEARCH

AIM

To write a C program for implementing the traversal algorithm for Depth first traversal

PRE LAB DISCUSSION

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Depth First Traversal of the following graph is 2, 0, 1, 3.



ALGORITHM

- Step 1: Start from any vertex, say V_i .
- Step 2: V_i is visited and then all vertices adjacent to V_i are traversed recursively using DFS.
- Step 3: Since, a graph can have cycles. We must avoid revisiting a node. To do this, when we visit a vertex V , we mark it visited.
- Step 4: A node that has already been marked as visited should not be selected for traversal.
- Step 5: Marking of visited vertices can be done with the help of a global array `visited[]`.
- Step 6: Array `visited[]` is initialized to false (0).

PROGRAM

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

node *G[20];
//heads of linked list
int visited[20];
int n;
void read_graph();
//create adjacency list
void insert(int,int);
//insert an edge (vi,vj) in te adjacency list
void DFS(int);

void main()
{
    int i;
    read_graph();
    //initialised visited to 0
    for(i=0;i<n;i++)
        visited[i]=0;

    DFS(0);
}

void DFS(int i)
{
    node *p;

    printf("\n%d",i);
    p=G[i];
    visited[i]=1;
    while(p!=NULL)
    {
        i=p->vertex;

        if(!visited[i])
            DFS(i);
        p=p->next;
    }
}
```



```

}

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

    scanf("%d",&n);

    //initialise G[] with a null

    for(i=0;i<n;i++)
    {
        G[i]=NULL;
        //read edges and insert them in G[]

        printf("Enter number of edges:");
        scanf("%d",&no_of_edges);

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

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

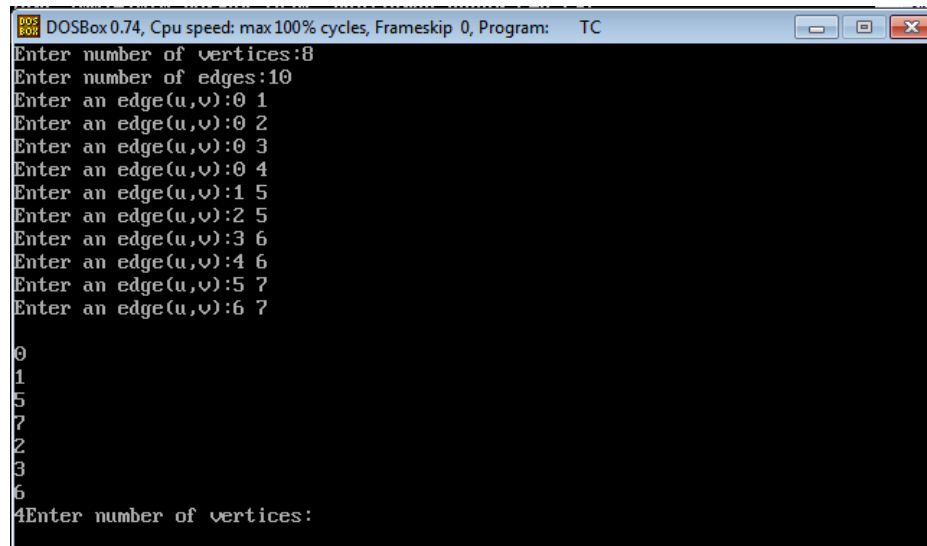
    //acquire memory for the new node
    q=(node*)malloc(sizeof(node));
    q->vertex=vj;
    q->next=NULL;

    //insert the node in the linked list numbervi
    if(G[vi]==NULL)
        G[vi]=q;
    else
    {
        //go to end of the linked list
        p=G[vi];

```

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

OUTPUT



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter number of vertices:8
Enter number of edges:10
Enter an edge(u,v):0 1
Enter an edge(u,v):0 2
Enter an edge(u,v):0 3
Enter an edge(u,v):0 4
Enter an edge(u,v):1 5
Enter an edge(u,v):2 5
Enter an edge(u,v):3 6
Enter an edge(u,v):4 6
Enter an edge(u,v):5 7
Enter an edge(u,v):6 7

0
1
5
7
2
3
6
4Enter number of vertices:
```

RESULT

Thus the C program for implementing the traversal algorithm for Depth first traversal was implemented successfully.

Ex.No:9B

BREADTH FIRST SEARCH

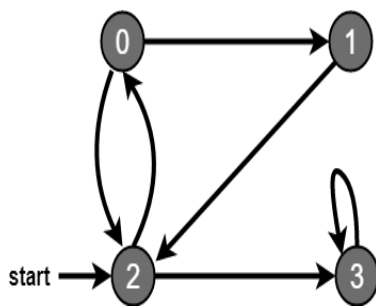
AIM

To write a C program for implementing the traversal algorithm for Breadth first traversal

PRE LAB DISCUSSION

Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.

For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Breadth First Traversal of the following graph is 2, 0, 3, 1.



ALGORITHM

- Step 1: Start from any vertex, say V_i .
- Step 2: V_i is visited and then all vertices adjacent to V_i are traversed recursively using BFS.
- Step 3: avoid revisiting a node. To do this, when we visit a vertex V , we mark it visited.
- Step 4: A node that has already been marked as visited should not be selected for traversal.
- Step 5: Marking of visited vertices can be done with the help of a global array `visited []`.
- Step 6: Array `visited []` is initialized to false (0).

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#define initial 1
#define waiting 2
#define visited 3
```

```

int n;
intadj[MAX][MAX];
intstate[MAX];
void create_graph();
void BF_Traversal();
void BFS(intv);
int queue[MAX], front = -1, rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();
int main()
{
    create_graph();
    BF_Traversal();
    return 0;
}
void BF_Traversal()
{
    int v;

    for(v=0; v<n; v++)
        state[v] = initial;

    printf("Enter Start Vertex for BFS: \n");
    scanf("%d", &v);
    BFS(v);
}
void BFS(int v)
{
    int i;
    insert_queue(v);
    state[v] = waiting;
    while(!isEmpty_queue())
    {
        v = delete_queue( );
        printf("%d ",v);
        state[v] =visited;

        for(i=0; i<n;i++)
        {
            if(adj[v][i] == 1 && state[i] == initial)
            {
                insert_queue(i);
                state[i] = waiting;
            }
        }
    }
    printf("\n");
}

```

```

}

void insert_queue(int vertex)
{
    if(rear == MAX-1)
        printf("Queue Overflow\n");
    else
    {
        if(front == -1)
            front = 0;
        rear = rear+1;
        queue[rear] = vertex ;
    }
}

int isEmpty_queue()
{
    if(front == -1 || front > rear)
        return 1;
    else
        return 0;
}

intdelete_queue()
{
    intdelete_item;
    if(front == -1 || front > rear)
    {
        printf("Queue Underflow\n");
        exit(1);
    }

    delete_item = queue[front];
    front = front+1;
    return delete_item;
}

void create_graph()
{
    int count,max_edge,origin,destin;

    printf("Enter number of vertices : ");
    scanf("%d",&n);
    max_edge = n*(n-1);

    for(count=1; count<=max_edge; count++)
    {
        printf("Enter edge %d( -1 -1 to quit ) : ",count);
    }
}

```

```

scanf("%d %d",&origin,&destin);

if((origin == -1) && (destin == -1))
    break;

if(origin>=n || destin>=n || origin<0 || destin<0)
{
    printf("Invalid edge!\n");
    count--;
}
else
{
    adj[origin][destin] = 1;
}
}
}

```

OUTPUT

```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter number of vertices : 8
Enter edge 1( -1 -1 to quit ) : 0 1
Enter edge 2( -1 -1 to quit ) : 0 2
Enter edge 3( -1 -1 to quit ) : 0 3
Enter edge 4( -1 -1 to quit ) : 0 4
Enter edge 5( -1 -1 to quit ) : 1 5
Enter edge 6( -1 -1 to quit ) : 2 5
Enter edge 7( -1 -1 to quit ) : 3 6
Enter edge 8( -1 -1 to quit ) : 4 6
Enter edge 9( -1 -1 to quit ) : 5 7
Enter edge 10( -1 -1 to quit ) : 6 7
Enter edge 11( -1 -1 to quit ) : -1 -1
Enter Start Vertex for BFS:
0
0 1 2 3 4 5 6 7
Enter number of vertices : _

```

RESULT

Thus the C program for implementing the traversal algorithm for Breadth first traversal was implemented successfully

Ex.No:10

APPLICATIONS OF GRAPHS

AIM

To write a C program to implement the shortest path using dijkstra's algorithm.

PRE LAB DISCUSSION

Dijkstra's algorithm has many variants but the most common one is to find the shortest paths from the source vertex to all other vertices in the graph.

Steps:

- Set all vertices distances = infinity except for the source vertex, set the sourcedistance = 0.
- Push the source vertex in a min-priority queue in the form (distance , vertex), as the comparison in the min-priority queue will be according to verticesdistances.
- Pop the vertex with the minimum distance from the priority queue (at first thepopped vertex =source).
- Update the distances of the connected vertices to the popped vertex in case of "current vertex distance + edge weight < next vertex distance", then push thevertex with the new distance to the priority queue.
- If the popped vertex is visited before, just continue without usingit.
- Apply the same algorithm again until the priority queue isempty.

ALGORITHM

Step 1:Select the source node also called the initial node

Step 2: Define an empty set N that will be used to hold nodes to which a shortest path has been found.

Step 3: Label the initial node with , and insert it intoN.

Step 4: Repeat Steps 5 to 7 until the destination node is in N or there are no more labelled nodes in N.

Step 5: Consider each node that is not in N and is connected by an edge from the newly inserted node.

Step 6. (a) If the node that is not in N has no label then SET the label of the node = the label of the newly inserted node + the length of the edge.

(b) Else if the node that is not in N was already labelled, then SET its new label = minimum

Step 7: Pick a node not in N that has the smallest label assigned to it and add it to N.

PROGRAM

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

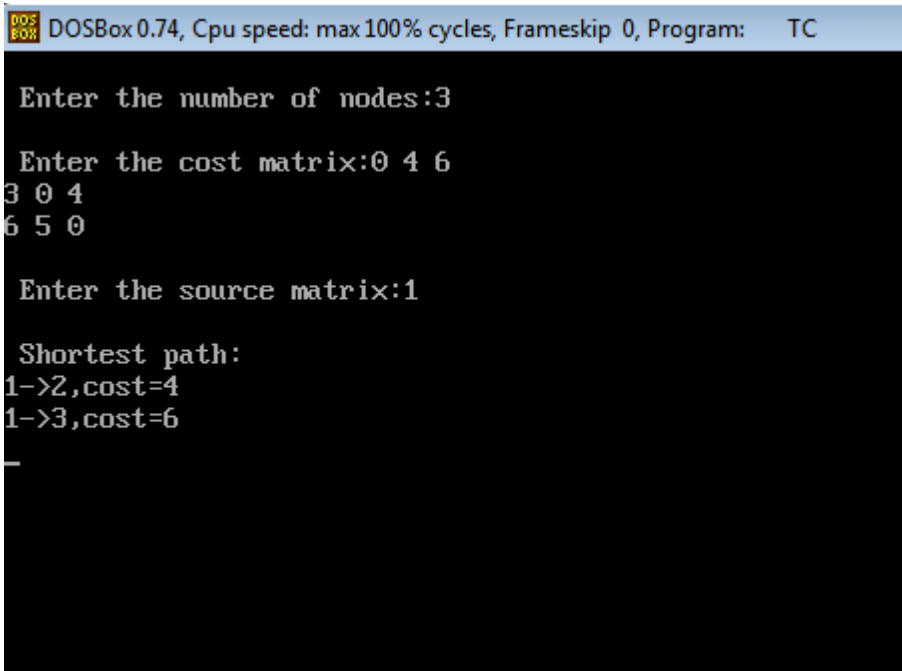
```

#define infinity 999
void dij(int n,int v,int cost[10][10],int dist[])
{
    int i,u,count,w,flag[10],min;
    for(i=1;i<=n;i++)
        flag[i]=0,dist[i]=cost[v][i];
    count=2;
    while(count<=n)
    {
        min=99;
        for(w=1;w<=n;w++)
            if(dist[w]<min && !flag[w])
                min=dist[w],u=w;
        flag[u]=1;
        count++;
        for(w=1;w<=n;w++)
            if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
                dist[w]=dist[u]+cost[u][w];
    }
}

void main()
{
    int n,v,i,j,cost[10][10],dist[10];
    clrscr();
    printf("\n Enter the number of nodes:");
    scanf("%d",&n);
    printf("\n Enter the cost matrix:");
    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]=infinity;
        }
    printf("\n Enter the source matrix:");
    scanf("%d",&v);
    dij(n,v,cost,dist);
    printf("\n Shortest path:\n");
    for(i=1;i<=n;i++)
        if(i!=v)
            printf("%d->%d,cost=%d\n",v,i,dist[i]);
    getch();
}

```


OUTPUT



```
DOS
BOX DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

Enter the number of nodes:3

Enter the cost matrix:0 4 6
3 0 4
6 5 0

Enter the source matrix:1

Shortest path:
1->2,cost=4
1->3,cost=6
-
```

The screenshot shows a DOSBox window with a blue title bar containing the text "DOS BOX DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The main window area is black with white text. The text shows the user inputting values for a shortest path algorithm: the number of nodes (3), a cost matrix (0 4 6; 3 0 4; 6 5 0), and a source matrix (1). The program then outputs the shortest paths from node 1: 1->2 with a cost of 4, and 1->3 with a cost of 6. A single hyphen "-" is printed on the final line.

RESULT

Thus the C program to implement shortest path was completed successfully.

Ex.No:11A

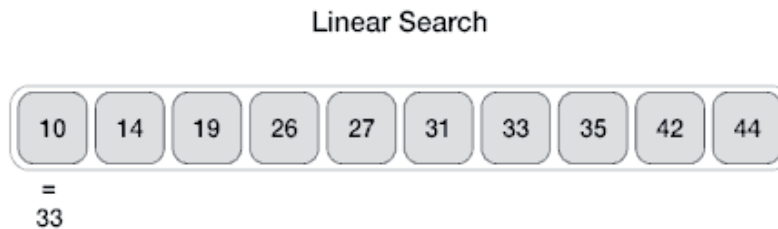
LINEARSEARCH

AIM

To write a C program to implement the concept of linear search.

PRE LAB DISCUSSION

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.



To search the number 33 in the array given below, linear search will go step by step in a sequential order starting from the first element in the given array.

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 arr[20];
int i,size,sech;
printf("\n\t-- Linear Search --\n\n");
printf("Enter total no. of elements : ");
scanf("%d",&size);
for(i=0; i<size; i++)
{
printf("Enter %d element : ",i+1);
scanf("%d",&arr[i]);}
printf("Enter the element to be searched:");
scanf("%d",&sech);
for(i=0; i<size; i++)
{
if(sech==arr[i])
{
printf("Element exits in the list at position : %d",i+1);
break;
}
}getch();
}

```

OUTPUT

-- Linear Search --

```

Enter total no. of elements : 5
Enter 1 element : 10
Enter 2 element :4
Enter 3 element :2
Enter 4 element : 17
Enter 5 element : 100
Enter the element to be searched: 17
Element exits in the list at position : 4

```

RESULT

Thus the C program to implement linear search was executed successfully.

Ex.No:11B

BINARY SEARCH

AIM

To write a C program to implement the concept of binary search.

PRE LAB DISCUSSION

Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

If searching for 23 in the 10-element array:

| | | | | | | | | | | |
|---------------------------------------|---|---|---|----|----|----|----|----|----|----|
| | 2 | 5 | 8 | 12 | 16 | 23 | 38 | 56 | 72 | 91 |
| | L | | | | | | | | H | |
| 23 > 16, take 2 nd half | 2 | 5 | 8 | 12 | 16 | 23 | 38 | 56 | 72 | 91 |
| | | | | | L | | | | H | |
| 23 < 56, take 1 st half | 2 | 5 | 8 | 12 | 16 | 23 | 38 | 56 | 72 | 91 |
| | | | | | | | L | | H | |
| Found 23, Return 5 | 2 | 5 | 8 | 12 | 16 | 23 | 38 | 56 | 72 | 91 |

ALGORITHM

- Step 1: Set the list to be the whole list
- Step 2: Find the middle value of the list
- Step 3: If the middle value is equal to the target then we declare victory and stop.
- Step 4: If the middle item is less than the target, then we set the new list to be the upper half of the old list and we repeat from step 2 using the new list.
- Step 5: If the middle value is greater than the target, then we set the new list to be the bottom half of the list, and we repeat from step 2 with the new list.
- Step 6: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main(){
int n,i,search,f=0,low,high,mid,a[20];
clrscr();
```

```

printf("Enter the nvalue:");
scanf("%d",&n);
for(i=1;i<=n;i++){
printf("Enter the number in ascending order a[%d]=",i);
scanf("%d",&a[i]);
}
printf("Enter the search element:");
scanf("%d",&search);
low=1;
high=n;
while(low<=high){
mid=(low+high)/2;
if(search<a[mid]){
high=mid-1;
}
elseif(search>a[mid]){
low=mid+1;
}
else{
f=1;
printf("obtained in the position %d:",mid);
getch();
exit();
}
}
if(f==0)
printf("not present");
getch();
}

```

OUTPUT

```

Enter the n value:5
Enter the number in ascending order a[1]=10
Enter the number in ascending order a[2]=8
Enter the number in ascending order a[3]=9
Enter the number in ascending order a[4]=24
Enter the number in ascending order a[5]=1
Enter the search element:9
obtained in the position 3:

```

RESULT

Thus the C program to implement the binary search was completed successfully.

Ex.No:11C

BUBBLE SORT

AIM

To write a C program to implement the concept of bubble sort.

PRE LAB DISCUSSION

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example:

First Pass:

(**5** 1 4 2 8) \rightarrow (**1** **5** 4 2 8), Here, algorithm compares the first two elements, and swaps since $5 > 1$.

(1 **5** 4 2 8) \rightarrow (1 **4** **5** 2 8), Swap since $5 > 4$

(1 4 **5** 2 8) \rightarrow (1 4 **2** **5** 8), Swap since $5 > 2$

(1 4 2 **5** 8) \rightarrow (1 4 2 **5** 8), Now, since these elements are already in order ($8 > 5$), algorithm does not swap them.

Second Pass:

(**1** **4** 2 5 8) \rightarrow (**1** **4** 2 5 8)

(1 **4** 2 5 8) \rightarrow (1 **2** **4** 5 8), Swap since $4 > 2$

(1 2 **4** 5 8) \rightarrow (1 2 **4** 5 8)

(1 2 4 **5** 8) \rightarrow (1 2 4 **5** 8)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:

(**1** **2** 4 5 8) \rightarrow (**1** **2** 4 5 8)

(1 **2** **4** 5 8) \rightarrow (1 **2** **4** 5 8)

(1 2 **4** **5** 8) \rightarrow (1 2 **4** **5** 8)

(1 2 4 **5** 8) \rightarrow (1 2 4 **5** 8)

ALGORITHM

Step 1: Start.

Step 2: Repeat Steps 3 and 4 for i=1 to 10

Step 3: Set j=1

Step 4: Repeat while j<=n

(A) if a[i] < a[j]

Then interchange a[i] and a[j]

[End of if]

(B) Set j = j+1

[End of InnerLoop]

[End of Step 1 Outer Loop]

Step 5: Stop.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main(){
int n, i, j, temp , a[100];
printf("Enter the total integers you want to enter (make it less than 100):\n");
scanf("%d",&n);
printf("Enter the %d integer array elements:\n",n);
for(i=0;i<n;i++){
scanf("%d",&a[i]);
}
for(i=0;i<n-1;i++){
for(j=0;j<n-i-1;j++){
if(a[j+1]<a[j]){
temp = a[j];
a[j] = a[j+1];
a[j+1] =temp;
}
}
}
printf("The sorted numbers are:");
for(i=0;i<n;i++){
printf("%3d",a[i]);
}
getch();
}
```

OUTPUT

Enter the total integers you want to enter (make it less than 100):

5

Enter the 5 integer array elements:

99

87

100

54

150

The sorted numbers are: 54 87 99 100 150

RESULT

Thus the C program for the concept of bubble sort was implemented successfully.

Ex.No:11D

INSERTION SORT

AIM

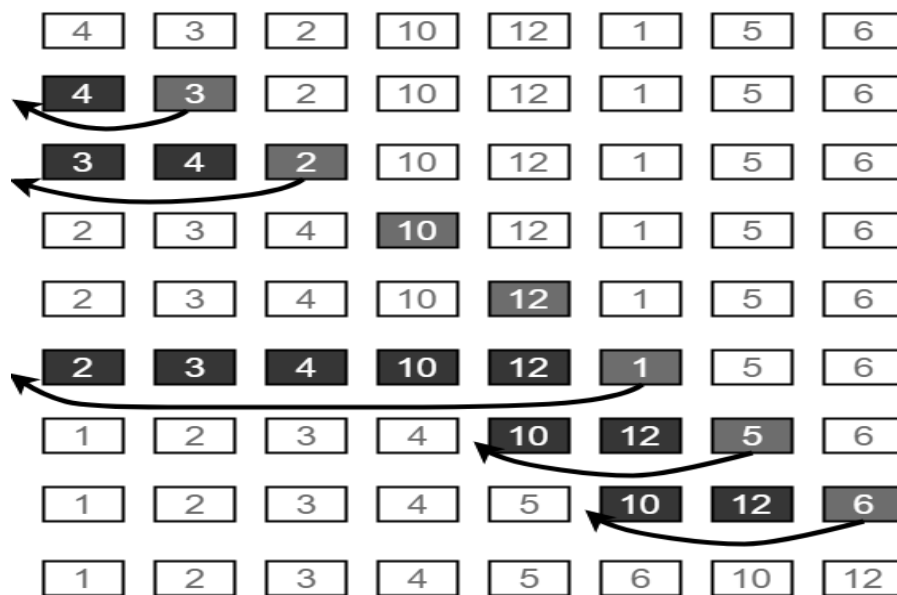
To write a C program to implement the concept of insertionsort.

PRE LAB DISCUSSION

Insertion sort is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be inserted in this sorted sub-list, has to find its appropriate place and then it has to be inserted there.

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Insertion Sort Execution Example



ALGORITHM

- Step 1: Start with an empty left hand [sorted array] and the cards face down on the table [unsorted array].
- Step 2: Then remove one card [key] at a time from the table [unsorted array], and insert it into the correct position in the left hand [sorted array].
- Step 3: To find the correct position for the card, we compare it with each of the cards already in the hand, from right to left.
- Step 4: Stop

PROGRAM

```
#include<stdio.h>
void inst_sort(int[]);
void main()
{
    int num[5],count;
    printf("\n enter the five elements to sort:\n");
    for(count=0;count<5;count++)
        scanf("%d",&num[count]);
    inst_sort(num); /*function call for insertion sort*/
    printf("\n\n elements after sorting:\n"); for(count=0;count<5;count++)
        printf("%d\n",num[count]);
}
void inst_sort(int num[])
{ /* function definition for insertion sort*/
    int i,j,k;
    for(j=1;j<5;j++) { k=num[j];
        for(i=j-1;i>=0&& k<num[i];i--)
            num[i+1]=num[i];
        num[i+1]=k;
    }}
```

OUTPUT

enter the five elements to sort:

5 4 3 2 1

elements after sorting:

1
2
3
4
5

RESULT

Thus the C program to implement insertion sort was completed successfully.

Ex.No:12A

HASHING WITH SEPARATE CHAINING

AIM

To write a C program to implement the concept of hashing using separate chaining.

PRE LAB DISCUSSION

In hashing there is a hash function that maps keys to some values. But these hashing function may lead to collision that is two or more keys are mapped to same value. Chain hashing avoids collision. The idea is to make each cell of hash table point to a linked list of records that have same hash functionvalue.

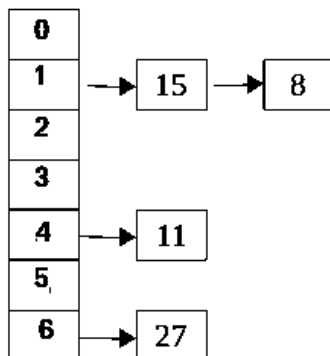
Let's create a hash function, such that our hash table has 'N' number of buckets. To insert a node into the hash table, we need to find the hash index for the given key. And it could be calculated using the hashfunction.

Example: $\text{hashIndex} = \text{key} \% \text{noOfBuckets}$

Insert: Move to the bucket corresponds to the above calculated hash index and insert the new node at the end of the list.

Let's say hash table with 7 buckets (0, 1, 2, 3, 4, 5, 6)

Keys arrive in the Order (15, 11 , 27 , 8)



ALGORITHM

Step 1:Start

Step 2:Create Table size

Step 3: Create hash function

Step 4: To insert a node into the hash table, we need to find the hash index for the given key. And it could be calculated using the hash function.

Step 5: Display hash entry. Step 6: Stop

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 3
struct node
{
    int data;
    struct node *next;
};
struct node *head[TABLE_SIZE]={NULL},*c,*p;
void insert(int i,int val)
{
    struct node * newnode=(struct node *)malloc(sizeof(struct node));
    newnode->data=val;
    newnode->next = NULL;
    if(head[i] == NULL)
        head[i] = newnode;
    else
    {
        c=head[i];
        while(c->next != NULL)
            c=c->next;
        c->next=newnode;
    }
}
void display(int i)
{
    if(head[i] == NULL)
    {
        printf("No Hash Entry");
        return;
    }
    else
```

```

        {
            for(c=head[i];c!=NULL;c=c->next)
                printf("%d->",c->data);
        }
    }
main()
{
    int opt,val,i;
    while(1)
    {
        printf("\nPress 1. Insert\t 2. Display \t3. Exit \n");
        scanf("%d",&opt);
        switch(opt)
        {
            case 1:
                printf("\nenter a value to insert into hash table\n");
                scanf("%d",&val);
                i=val%TABLE_SIZE;
                insert(i,val);
                break;
            case 2:
                for(i=0;i<TABLE_SIZE;i++)
                {
                    printf("\nentries at index %d\n",i);
                    display(i);
                }
                break;
            case 3: exit(0);
        }
    }
}

```

OUTPUT

```

C:\Users\sivaram\Desktop\hash.exe
Press 1. Insert  2. Display  3. Exit
1
enter a value to insert into hash table
21
Press 1. Insert  2. Display  3. Exit
1
enter a value to insert into hash table
18
Press 1. Insert  2. Display  3. Exit
1
enter a value to insert into hash table
31
Press 1. Insert  2. Display  3. Exit
2
entries at index 0
21->18->
entries at index 1
31->
entries at index 2
No Hash Entry

```

RESULT

Thus the C program to implement hashing using separate chaining was completed successfully.

Ex.No:12B

HASHING WITH OPEN ADDRESSING

AIM

To write a C program to implement the concept of hashing using linear probing in open addressing.

PRE LAB DISCUSSION

Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).

Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.

Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.

Delete(k): **Delete operation is interesting.** If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted".

Insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

Open Addressing is done following ways:

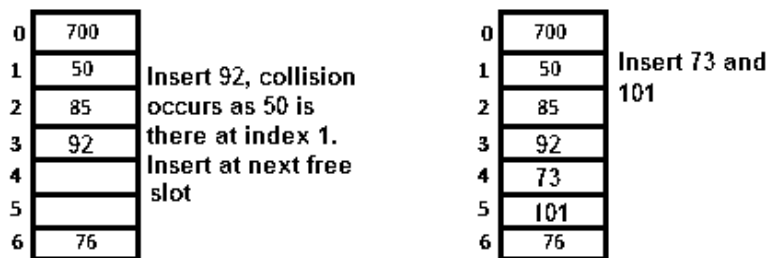
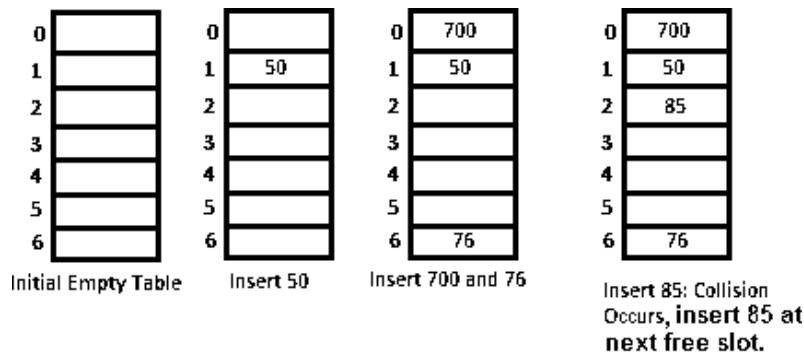
a) Linear Probing: In linear probing, we linearly probe for next slot. For example, typical gap between two probes is 1 as taken in below example also. let **hash(x)** be the slot index computed using hash function and **S** be the table size

If slot $\text{hash}(x) \% S$ is full, then we try $(\text{hash}(x) + 1) \% S$

If $(\text{hash}(x) + 1) \% S$ is also full, then we try $(\text{hash}(x) + 2) \% S$

If $(\text{hash}(x) + 2) \% S$ is also full, then we try $(\text{hash}(x) + 3) \% S$

Let us consider a simple hash function as "key mod 7" and sequence of keys as 50, 700, 76, 85, 92, 73, 101.



Clustering: The main problem with linear probing is clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search an element.

b) Quadratic Probing We look for i^2 th slot in i 'th iteration.
let $\text{hash}(x)$ be the slot index computed using hashfunction.

If slot $\text{hash}(x) \% S$ is full, then we try $(\text{hash}(x) + 1*1) \% S$

If $(\text{hash}(x) + 1*1) \% S$ is also full, then we try $(\text{hash}(x) + 2*2) \% S$

If $(\text{hash}(x) + 2*2) \% S$ is also full, then we try $(\text{hash}(x) + 3*3) \% S$

c) Double Hashing We use another hash function $\text{hash2}(x)$ and look for $i*\text{hash2}(x)$ slot in i 'th iteration.

let $\text{hash}(x)$ be the slot index computed using hash function.

If slot $\text{hash}(x) \% S$ is full, then we try $(\text{hash}(x) + 1*\text{hash2}(x)) \% S$

If $(\text{hash}(x) + 1*\text{hash2}(x)) \% S$ is also full, then we try $(\text{hash}(x) + 2*\text{hash2}(x)) \% S$

If $(\text{hash}(x) + 2*\text{hash2}(x)) \% S$ is also full, then we try $(\text{hash}(x) + 3*\text{hash2}(x)) \% S$

ALGORITHM

Step 1: Start

Step 2: Create hash table and hash function

Step 3: Assigning INT_MIN indicates that cell is empty

Step4: INT_MIN and INT_MAX indicates that cell is empty. So if cell is empty loop will

break and goto bottom of the loop to insert element. If table is full we should break, if not check this, loop will go to infinite loop.

Step 5: If the hash element is empty, the deletion has immediately failed.

Otherwise, check for a match between the target and data key

- If there is a match, delete the hash element (and set the flag)
- If there is no match, probe the table until either:
 - An match is found between the target key and the data's key; the data can be deleted, and the deleted flag set.
 - A completely empty hash element is found

Step 6: If the hash element is empty, the search has immediately failed.

Otherwise, check for a match between the search and data key

- If there is a match, return the data.
- If there is no match, probe the table until either:
 - An match is found between the search and data key
 - A completely empty hash element is found.

Step 7: Stop

PROGRAM

```
#include<stdio.h>
#include<limits.h>
void insert(int ary[],int hFn, int size){
    int element,pos,n=0;
    printf("Enter key element to insert\n");
    scanf("%d",&element);
    pos = element%hFn;
    while(ary[pos]!= INT_MIN)
    {
        // INT_MIN and INT_MAX indicates that cell is empty. So if cell is empty loop will
        break and goto bottom of the loop to insert element
        if(ary[pos]== INT_MAX)
            break;
        pos = (pos+1)%hFn;
        n++;
        if(n==size)
            break;    // If table is full we should break, if not check this, loop will go to infinite
loop.
    }

    if(n==size)
```

```

        printf("Hash table was full of elements\nNo Place to insert this element\n\n");
    else
        ary[pos]=element;    //Insertingelement
    }

void delete(int ary[],int hFn,int size)
{
    /* very careful observation required while deleting. To understand code of thisdelete
function see the note at end of the program*/
    int element,n=0,pos;

    printf("Enter element to delete\n");
    scanf("%d",&element);

    pos = element%hFn;

    while(n++ != size){
        if(ary[pos]==INT_MIN){
            printf("Element not found in hash table\n");
            break;
        }
        else if(ary[pos]==element)
        {
            ary[pos]=INT_MAX;
            printf("Element deleted\n\n");
            break;
        }
        else
        {
            pos = (pos+1) % hFn;
        }
    }
    if(--n==size)
        printf("Element not found in hash table\n");
}

void search(int ary[],int hFn,int size){
    int element,pos,n=0;

    printf("Enter element you want to search\n");
    scanf("%d",&element);

    pos = element%hFn;

    while(n++ != size){
        if(ary[pos]==element){
            printf("Element found at index %d\n",pos);

```

```

        break;
    }
    else
        if(ary[pos]==INT_MAX ||ary[pos]!=INT_MIN)
            pos = (pos+1) %hFn;
    }
    if(--n==size) printf("Element not found in hash table\n");
}

void display(int ary[],int size){
    int i;

    printf("Index\tValue\n");

    for(i=0;i<size;i++)
        printf("%d\t%d\n",i,ary[i]);
}
int main(){
    int size,hFn,i,choice;

    printf("Enter size of hash table\n");
    scanf("%d",&size);

    int ary[size];

    printf("Enter hash function [if mod 10 enter 10]\n");
    scanf("%d",&hFn);

    for(i=0;i<size;i++)
        ary[i]=INT_MIN; //Assigning INT_MIN indicates that cell is empty
    do{
        printf("Enter your choice\n");
        printf(" 1-> Insert\n 2-> Delete\n 3-> Display\n 4-> Searching\n 0-> Exit\n");
        scanf("%d",&choice);

        switch(choice){
            case 1:
                insert(ary,hFn,size);
                break;
            case 2:
                delete(ary,hFn,size);
                break;
            case 3:
                display(ary,size);
                break;
            case 4:
                search(ary,hFn,size);

```

```

        break;
    default:
        printf("Enter correct choice\n");
        break;
    }
}while(choice);

return 0;
}

```

OUTPUT

```

Enter size of hash table
10
Enter hash function [if mod 10 enter 10]
10
Enter your choice
1-> Insert
2-> Delete
3->Display
4->Searching
0->Exit
1
Enter key element to insert
12
Enter your choice
1-> Insert
2-> Delete
3->Display
4->Searching
0->Exit
1
Enter key element to insert
22
Enter your choice
1-> Insert
2-> Delete
3->Display
4->Searching
0->Exit
1
Enter key element to insert
32
Enter your choice
1-> Insert
2-> Delete

```

```

3->Display
4->Searching
0->Exit
3
Index      Value
0          -2147483648
1          -2147483648
2          12
3          22
4          32
5          -2147483648
6          -2147483648
7          -2147483648
8          -2147483648
9          -2147483648
Enter your choice
1-> Insert
2-> Delete
3->Display
4->Searching
0->Exit
2
Enter element to delete
12
Element deleted
Enter your choice
1-> Insert
2-> Delete
3->Display
4->Searching
0->Exit
4
Enter element you want to search
32
Element found at index 4
Enter your choice
1->Insert
2->Delete
3->Display
4->Searching
0->Exit

```

RESULT

Thus the C program to implement hashing using linear probing in open addressing was completed successfully.

