SRM MADURAI

COLLEGE FOR ENGINEERING AND TECHNOLOGY

Approved by AICTE, New Delhi | Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY MANUAL

Sub.Code	: CS3311
Sub.Name	: DATA STRUCTURES LABORATORY
Class	: II CSE
Regulation	: 2021

Prepared By,

Dr.C.Callins Christiyana, Professor & Head, CSE

Ms.G.Aninthitha AP/CSE

Ms.V.Lithika AP/CSE

Verified By,

Dr.C.Callins Christiyana

HoD/CSE

Approved By,

SYLLABUS

1. Array implementation of Stack, Queue and Circular Queue ADTs.
2. Implementation of Singly Linked List.
3. Linked list implementation of Stack and Linear Queue ADTs.
4. Implementation of Polynomial Manipulation using Linked list.
5. Implementation of Evaluating Postfix Expressions, Infix to Postfix conversion.
6. Implementation of Binary Search Trees.
7. Implementation of AVL Trees.
8. Implementation of Heaps using Priority Queues.
9. Implementation of Dijkstra's Algorithm.
10. Implementation of Prim's Algorithm.
11. Implementation of Linear Search and Binary Search.
12. Implementation of Insertion Sort and Selection Sort.
13. Implementation of Merge Sort.
14. Implementation of Open Addressing (Linear Probing and Quadratic Probing).

COURSE OUTCOMES:

At the end of this course, the students will be able to:

CO1: Implement Linear data structure algorithms.

CO2: Implement applications using Stacks and Linked lists.

CO3: Implement Binary Search tree and AVL tree operations.

CO4: Implement graph algorithms.

CO5: Analyze the various searching and sorting algorithms.

LIST OF EXPERIMENTS

Ex:No:1	Implementation of Singly Linked List
Date :	

Aim

To write a C program to perform the list operations using singly linked List.

Algorithm

1. Create a head node of the singly liked list with data part as value -1.
2. Get the user choice to perform insertion, deletion, find, find previous or display the elements in the singly linked list.
3. For insertion, get the value to be inserted (x) and the value (m) after which the new will be inserted. To insert the value at the beginning, give the value of 'm' as -1. The new element is inserted after the specified node of the linked list. It is necessary to traverse the node to reach the specified node after which the new node will be inserted. Assume the specified node is addressed as 'p'. After the insertion of new element, the new element will point the node which was pointed by node 'p' and node 'p' will point the new element.
4. For deletion, the value to be deleted (x) has to be inputted. It is necessary to traverse the linked list to reach the specified node. Assume the specified node as 'p'. It is necessary to identify the previous address of 'p' and the address of the previous is stored in 'pre'. After the deletion of the node 'p', the next of 'pre' will point next of 'p'. If the list is empty, no elements will be deleted from the list.
5. For find operation. element to be searched is inputted. Each element of the list is matched with the given element. If the element is found on any of the location, then location (address) of that element is returned otherwise null is returned.
6. For findprevious operation. element to be searched is inputted. Each element of the list is matched with the given element. If the element is found on any of the location, then the previous location (address) of that element is returned otherwise null is returned.
7. The display operation prints the elements of the linked list in the sequential order.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct snode
{
```

```
int data;
struct snode *next;
};
void insertion(struct snode *,int,int);
void deletion(struct snode *,int);
struct snode *find(struct snode *,int);
struct snode *findprevious(struct snode *,int);
void print(struct snode *);
void main()
{
struct snode *head,*p;
int ch,x,m;
clrscr();
//Creation of header node
head=(struct snode*)malloc(sizeof(struct snode));
head->data=-1;
head->next=NULL;
while(1)
{
printf("\n1.Insertion 2.Deletion 3.find 4.findprevious 5.print 6.exit\n");
printf("Enter the choice for operations\n");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("Enter the element to be inserted\n");
        scanf("%d",&x);
        printf("Enter the element after which new to be inserted\n");
        scanf("%d",&m);
        printf("Before Insertion\n");
        print(head);
        insertion(head,x,m);
        printf("After Insertion\n");
        print(head);
        break;
case 2:printf("Enter the element to be deleted\n");
```

```
        scanf("%d",&x);
        printf("Before Deletion\n");
        print(head);
        deletion(head,x);
        printf("After Deletion\n");
        print(head);
        break;
case 3:printf("enter the element to be searched\n");
        scanf("%d",&x);
        p=find(head,x);
        if(p!=NULL)
        printf("The element is present at the address %d\n",p);
        else
        printf("The element is not present\n");
        break;
case 4:printf("enter the element to be searched\n");
        scanf("%d",&x);
        p=findprevious(head,x);
        if(p!=NULL)
        printf("The previous snode address of the element is %d\n",p);
        else
        printf("The element is not present\n");
        break;
case 5:printf("The elements in the linked list are\n");
        print(head);
        break;
case 6:exit(0);
        break;
}
}
getch();
}
void insertion(struct snode *head,int x,int m)
{
struct snode *newnode,*p;
```

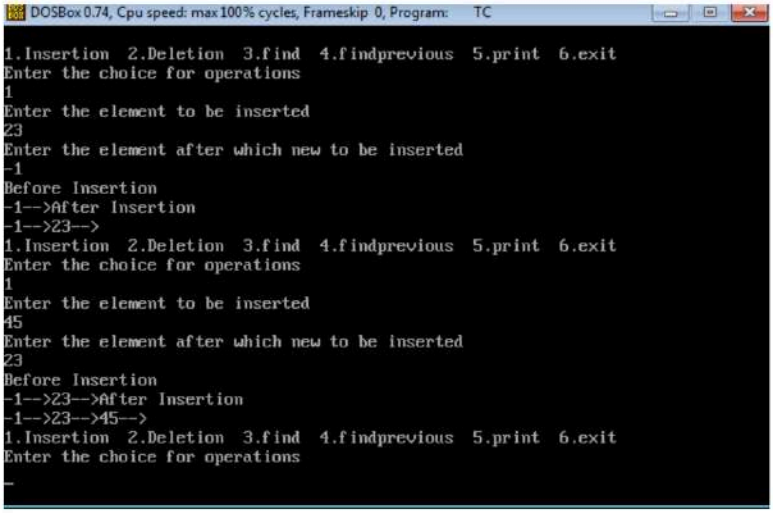
```
newnode=(struct snode*)malloc(sizeof(struct snode));
newnode->data=x;
if(m==-1)
{
newnode->next=head->next;
head->next=newnode;
}
else
{
p=head->next;
while(p!=NULL)
{
if(m==p->data)
{
newnode->next=p->next;
p->next=newnode;
return;
}
else
p=p->next;
}
}
}
void deletion(struct snode *head,int x)
{
struct snode *p,*pre;
pre=head;
p=head->next;
while(p!=NULL)
{
if(x==p->data)
{
pre->next=p->next;
free(p);
return;
```

```
}
else
{
pre=p;
p=p->next;
}
}
}
struct snode *find(struct snode *head, int x)
{
struct snode *p;
p=head->next;
while(p!=NULL)
{
if(x==p->data)
return(p);
else
p=p->next;
}
}
struct snode *findprevious(struct snode *head, int x)
{
struct snode *p,*pre;
pre=head;
p=head->next;
while(p!=NULL)
{
if(x==p->data)
return(pre);
else
{
pre=p;
p=p->next;
}
}
```

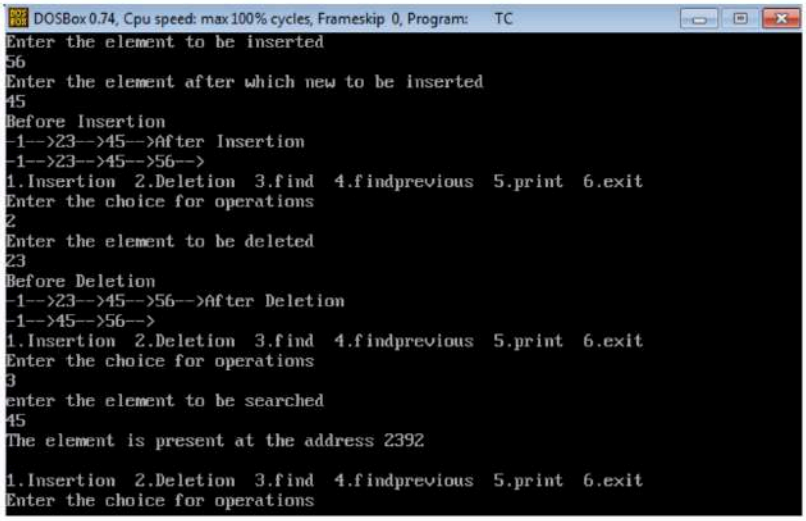
```
}
void print(struct snode *head)
{
struct snode *p;
p=head;
while(p!=NULL)
{
printf("%d-->",p->data);
p=p->next;
}
}
```

Sample Output

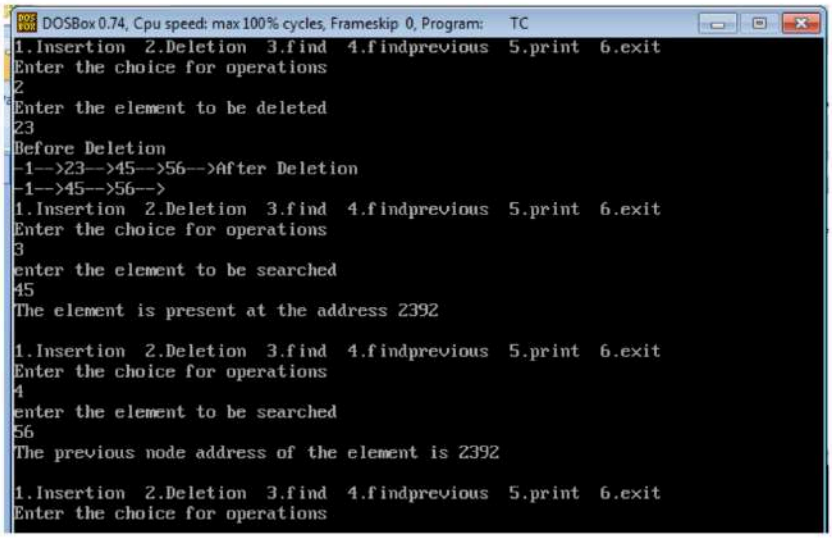
Insertion Operation



Deletion and Find Operation



Find previous operation



Result:

Thus, the singly linked list has been successfully implemented.

Sample Viva Questions:

1. What are the advantages of Singly linked list?
2. How the node structure of SLL has been organized?
3. How the space for singly linked list node has been created?
4. How the data values of the singly linked list have been accessed?
5. When we say the linked list is empty?
6. What are the drawbacks of Singly linked list?

Augmented Experiment:

1. Write a C program to count the number of elements present in the linked list.

Ex:No:2	Implementation of Polynomial Manipulation using Linked list
Date :	

Aim

To write a C program to perform the Polynomial addition and Multiplication operations using Linked List.

Algorithm for Polynomial Addition:

1. Create two polynomial linked list to represent two polynomial equations with two head addresses head1 and head2 respectively. The result of polynomial addition is stored into new polynomial linked list. The resultant polynomial linked list head address is addressed as head3.
2. Create two pointers, p1 and p2, to represent the current nodes of two input polynomial linked lists. Then, we generate the new polynomial nodes based on the exponent values of these two pointers. There are three cases:
    a. **Case 1: p1's power is greater than p2's power:** In this case, we create a new node in the resultant polynomial with p1's coefficient and p1's exponent. After that p1 is moved into the next node.
    b. **Case 2: p2's power is greater than p1's power:** In this case, we create a new node in the resultant polynomial with p2's coefficient and p2's exponent. After that p2 is moved into the next node.
    c. **Case 3: p1 and p2 have the same power:** In this case, the new coefficient is the total of p1's coefficient and p2's coefficient. If the new coefficient is not 0, we create a new node with the same exponent and the new coefficient. After that, both p1 and p2 are moved to the next nodes.
3. The procedure is continued until either one of the linked list is exhausted. Then the remaining nodes of other linked list is directly appended to the resultant polynomial linked list.

Algorithm for Polynomial Multiplication:

1. Create two polynomial linked list to represent two polynomial equations with two head addresses head1 and head2 respectively. The result of polynomial addition is stored into new polynomial linked list. The resultant polynomial linked list head address is addressed as head3.
2. Create two pointers, p1 and p2, to represent the current nodes of two input polynomial linked lists.

3. To multiply two polynomial equations, we have to multiply the 2nd polynomial with each term of 1st polynomial and store the multiplied value in a new linked list.
4. The coefficients of elements having the same power in resultant polynomial linked list will be added after then.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct pnode
 {
    int coeff;
    int exp;
    struct pnode *next;
};
void addition(struct pnode *,struct pnode *);
void multiplication(struct pnode *, struct pnode *);
void print(struct pnode *);
void main()
{
struct pnode *head1,*head2,*p,*newnode;
int no,coeff1,exp1,ch,x,m;
clrscr();
//Creation of header node of polynomial1
head1=(struct pnode *)malloc(sizeof(struct pnode));
```

```
head1->coeff=-1;
head1->exp=-1;
head1->next=NULL;
//Creation of polynomial1
printf("enter the values for polynomial1\n");
while(1)
{
printf("Enter the value for the coefficient\n");
scanf("%d",&coeff1);
printf("Enter the value for the exponent\n");
scanf("%d",&exp1);


newnode=(struct pnode*)malloc(sizeof(struct pnode));
newnode->coeff=coeff1;
newnode->exp=exp1;
newnode->next=NULL;
p=head1;
while(p->next!=NULL)
{
p=p->next;
}
p->next=newnode;
printf("\nDo you want to continue to create the node?if Yes give the value 1\n");
scanf("%d",&no);
if(no!=1)
break;
}
print(head1);
//Creation of header node of polynomial2
head2=(struct pnode *)malloc(sizeof(struct pnode));
```

```
head2->coeff=-1;
head2->exp=-1;
head2->next=NULL;
//Creation of polynomial2
printf("\nenter the values for polynomial2\n");
while(1)
{
printf("Enter the value for the coefficient\n");
scanf("%d",&coeff1);
printf("Enter the value for the exponent\n");
scanf("%d",&exp1);
newnode=(struct pnode*)malloc(sizeof(struct pnode));
newnode->coeff=coeff1;
newnode->exp=exp1;
newnode->next=NULL;
p=head2;
while(p->next!=NULL)
{
p=p->next;
}
p->next=newnode;
printf("\nDo you want to continue to create the node?if Yes give the value 1\n");
scanf("%d",&no);
if(no!=1)
break;
}
print(head2);
while(1)
{
printf("Enter the choice of operation 1.Addition 2.Multiplication 3.exit\n");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nBefore Addition\n");
        printf("polynomial list 1\n");
```

```
        print(head1);
        printf("\npolynomial list 2\n");
        print(head2);
        addition(head1,head2);
        break;
case 2: printf("Before Multiplication\n");
        printf("polynomial list 1\n");
        print(head1);
        printf("polynomial list 2\n");
        print(head2);
        multiplication(head1,head2);
        break;
case 3:exit(0);
    break;
}
}
getch();
}
void print(struct pnode *header)
{
struct pnode *p;
p=header;
while(p!=NULL)
{
printf("%d-",p->coeff);
printf("%d-->",p->exp);
p=p->next;
}
}
void addition(struct pnode *head1,struct pnode *head2)
{
struct pnode *p1,*p2,*head3,*newnode,*p3;
head3=(struct pnode *)malloc(sizeof(struct pnode));
head3->coeff=-1;
head3->exp=-1;
```

```
head3->next=NULL;
p3=head3;
p1=head1->next;
p2=head2->next;
while(p1&&p2)
{
if(p1->exp>p2->exp)
{
newnode=(struct pnode *)malloc(sizeof(struct pnode));
newnode->coeff=p1->coeff;
newnode->exp=p1->exp;
newnode->next=NULL;
p3->next=newnode;
p3=newnode;
p1=p1->next;
}
else if(p2->exp>p1->exp)
{
newnode=(struct pnode *)malloc(sizeof(struct pnode));
newnode->coeff=p2->coeff;
newnode->exp=p2->exp;
newnode->next=NULL;
p3->next=newnode;
p3=newnode;
p2=p2->next;
}
else if(p1->exp==p2->exp)
{
newnode=(struct pnode *)malloc(sizeof(struct pnode));
newnode->coeff=p1->coeff+p2->coeff;
newnode->exp=p1->exp;
newnode->next=NULL;
p3->next=newnode;
p3=newnode;
p1=p1->next;
```

```
p2=p2->next;
}
}
if(p1==NULL)
{
while(p2)
{
newnode=(struct pnode *)malloc(sizeof(struct pnode));
newnode->coeff=p2->coeff;
newnode->exp=p2->exp;
newnode->next=NULL;
p3->next=newnode;
p3=newnode;
p2=p2->next;
}
}
if(p2==NULL)
{
while(p1)
{
newnode=(struct pnode *)malloc(sizeof(struct pnode));
newnode->coeff=p1->coeff;
newnode->exp=p1->exp;
newnode->next=NULL;
p3->next=newnode;
p3=newnode;
p1=p1->next;
}
}
printf("\nAfter Addition\n");
print(head3);
}
void multiplication(struct pnode *head1,struct pnode *head2)
{
struct pnode *p1,*p2,*head3,*newnode,*p3,*p4,*head5,*p5;
```

```
head3=(struct pnode *)malloc(sizeof(struct pnode));
head3->coeff=-1;
head3->exp=-1;
head3->next=NULL;
p3=head3;
head5=(struct pnode *)malloc(sizeof(struct pnode));
head5->coeff=-1;
head5->exp=-1;
head5->next=NULL;
p5=head5;
p1=head1->next;
while(p1)
{
p2=head2->next;
while(p2)
{
newnode=(struct pnode *)malloc(sizeof(struct pnode));
newnode->coeff=p1->coeff*p2->coeff;
newnode->exp=p1->exp+p2->exp;
newnode->next=NULL;
p3->next=newnode;
p3=newnode;
p2=p2->next;
}
p1=p1->next;
}
//Merging
p3=head3->next;
while(p3)
{
p4=p3->next;
newnode=(struct pnode *)malloc(sizeof(struct pnode));
newnode->coeff=p3->coeff;
newnode->exp=p3->exp;
newnode->next=NULL;
```

```
while(p4)
{
if(p3->exp==p4->exp)
{
newnode->coeff=newnode->coeff+p4->coeff;
p4=p4->next;
}
else
{
p4=p4->next;
}
}
p3=p3->next;
p5->next=newnode;
p5=newnode;
if(p5->exp==p3->exp)
p3=p3->next;
}
printf("\nAfter Multiplication\n");
print(head5);
}
```

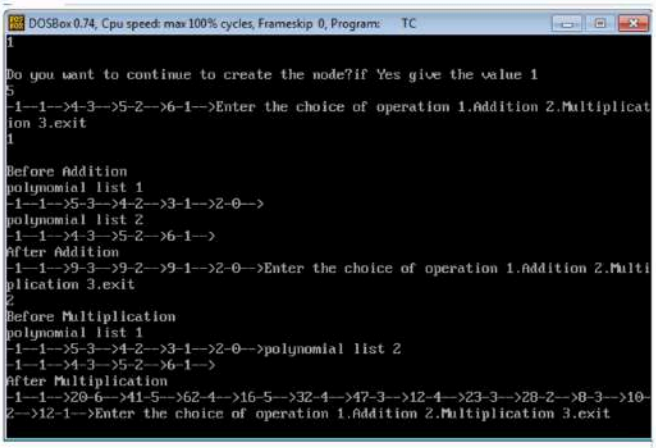
Sample Output :

```
Enter the value for the coefficient
4
Enter the value for the exponent
2

Do you want to continue to create the node?if Yes give the value 1
1
Enter the value for the coefficient
3
Enter the value for the exponent
1

Do you want to continue to create the node?if Yes give the value 1
1
Enter the value for the coefficient
2
Enter the value for the exponent
0

Do you want to continue to create the node?if Yes give the value 1
0
-1--1-->5-3-->4-2-->3-1-->2-0-->
enter the values for polynomial2
Enter the value for the coefficient
```



Result:

Thus, the polynomial addition and multiplication have been implemented successfully.

Sample Viva Questions:

1. What are the fields of polynomial linked list node?
2. What are the conditions to be checked to add two terms of the polynomial equations?
3. What is merging in polynomial multiplication?

Augmented Experiment:

1. Write a C program to perform polynomial subtraction.

Ex:No:3A	Array Implementation of Stack
Date :	

AIM:

To write a program in C to implement the stack operations using array.

ALGORITHM:

1. Create the stack array with maximum capacity and initializes the top pointer with -1..
2. Get the user choice to perform PUSH, POP or display elements.
3. For PUSH, check the stack full condition. If the stack is full, it is not possible to insert the item on to the stack, otherwise the new item is inserted at the top end.
4. For Pop, operation, check the stack empty condition, If the stack is empty, it not possible to delete the item from the stack, otherwise the item which is pointed by the top is deleted from the stack.

5. For display operation, the elements from zero position up to the top position from the stack array will be printed.

Program

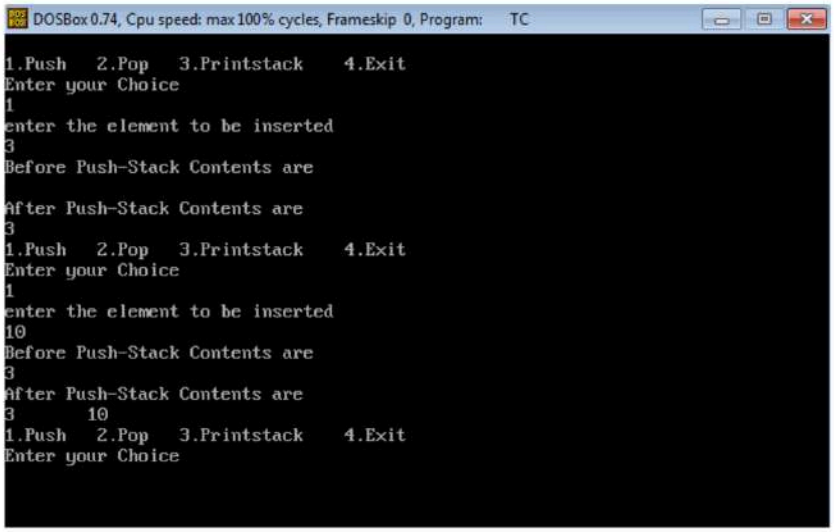
```
#include <stdio.h>
#include <conio.h>
int stack[10];
int top=-1;
void push(int);
void pop();
void print();
void main()
{
int ch,x;
clrscr();
while(1)
{
printf("\n1.Push\t 2.Pop\t 3.Printstack\t 4.Exit\n");
printf("Enter your Choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("enter the element to be inserted\n");
    scanf("%d",&x);
    printf("Before Push-");
    print();
    push(x);
    break;
case 2: printf("Before Pop-");
    print();
    pop();
    break;
case 3:print();
    break;
case 4:exit(0);
    break;
```

```
}
}
getch();
}
void push(int x)
{
if(top==9)
{
printf("Stack is Full, Not possible to push");
return;
}
else
{
top=top+1;
stack[top]=x;
printf("\nAfter Push-");
print();
}
}
void pop()
{
int del;
if(top==-1)
{
printf("Stack is Empty, Not possible to pop");
return;
}
else
{
del=stack[top];
printf("\nThe popped item is %d\n",del);
top=top-1;
printf("After Pop-");
print();
}
```

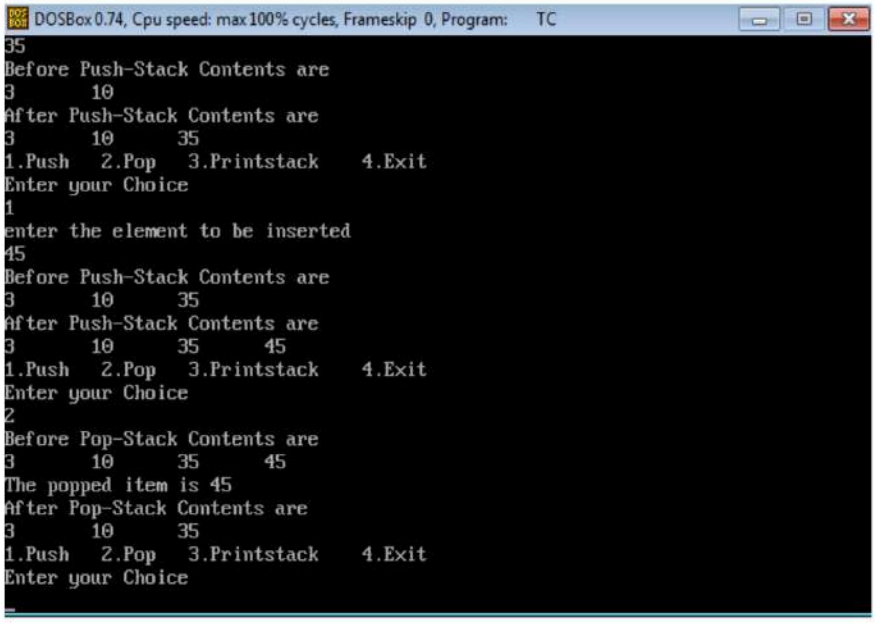
```
}
void print()
{
int i;
printf("Stack Contents are\n");
for(i=0;i<=top;i++)
{
printf("%d\t",stack[i]);
}
}
```

Sample Output:

Push Operation



PoP Operation



Result:

Thus, the stack operations have been implemented successfully.

Sample Viva Questions:

1. What is the concept of stack?
2. What is top pointer?
3. What is the value of top pointer when the stack is empty?
4. What is the condition for full stack array?

Augmented Experiment:

1. Write a C program to push 3 yellow balls, 2 red balls and 4 blue balls on to the stack and display the value of top pointer when the first yellow ball, first red ball and first blue balls are getting popped.

Ex:No:3B	Array Implementation of Queue
Date :	

AIM:

To write a program in C to implement the queue operations using array.

ALGORITHM:

1. Create the queue array with maximum capacity and initializes the rear pointer with -1 value and front pointer as 0.
2. Get the user choice to perform ENQUEUE, DEQUEUE or display elements.
3. For ENQUEUE, check the queue full condition. If the queue is full, it is not possible to insert the item on to the queue, otherwise the new item is inserted at the rear end.
4. For DEQUEUE, operation, check the queue empty condition, If the queue is empty, it not possible to delete the item from the queue, otherwise the item which is pointed by the front is deleted from the queue.
5. For display operation, the elements from zero position up to the rear position from the queue array will be printed.

Program

```
#include <stdio.h>
#include <conio.h>
int queue[10];
int rear=-1;
int front=0;
void enqueue(int);
void dequeue();
```

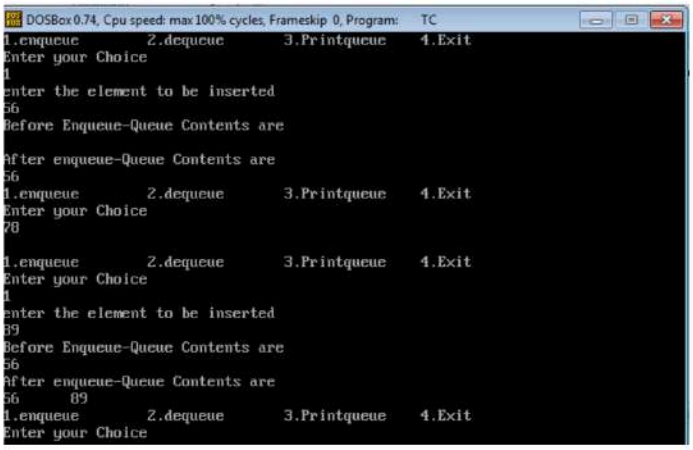
```
void print();
void main()
{
int ch,x;
clrscr();
while(1)
{
printf("\n1.enqueue\t 2.dequeue\t 3.Printqueue\t 4.Exit\n");
printf("Enter your Choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("enter the element to be inserted\n");
      scanf("%d",&x);
      printf("Before Enqueue-");
      print();
      enqueue(x);
      break;
case 2:printf("Before dequeue-");
      print();
      dequeue();
      break;
case 3:print();
      break;
case 4:exit(0);
      break;
}
}
getch();
}
void enqueue(int x)
{
if(rear==9)
{
printf("Queue is Full, Not possible to insert");
```

```
return;
}
else
{
rear=rear+1;
queue[rear]=x;
printf("\nAfter enqueue-");
print();
}
}
void dequeue()
{
int del;
if(rear<front)
{
printf("Queue is Empty, Not possible to delete\n");
return;
}
else
{
del=queue[front];
printf("\nThe deleted item is %d\n",del);
front=front+1;
printf("After dequeue-");
print();
}
}
void print()
{
int i;
printf("Queue Contents are\n");
for(i=front;i<=rear;i++)
{
printf("%d\t",queue[i]);
}
```

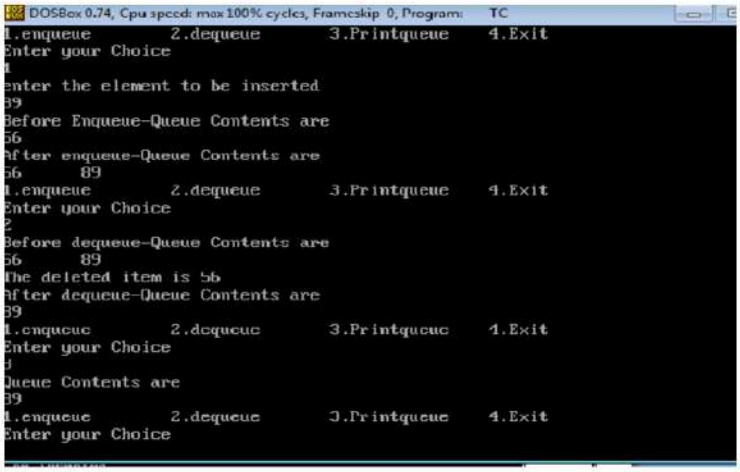
}

Sample Output

Enqueue Operation



Dequeue Operation:



Result:

Thus, the queue operations have been implemented successfully.

Sample Viva Questions:

1. What is the concept of queue?
2. What are rear and front pointers?
3. What is the empty condition for queue array?
4. What is the condition for full queue array?

Augmented Experiment:

1. Insert 10 boxes into the container queue and store 30 files into boxes with 5 files in a box. Write a C program to display the name of the boxes will not get files.

Ex:No:3C	Array Implementation of Circular Queue
Date :	

AIM:

To write a program in C to implement the circular queue operations using array.

ALGORITHM:

1. Create the queue array with maximum capacity and initializes the rear pointer with -1 value and front pointer as 0.
2. Get the user choice to perform ENQUEUE, DEQUEUE or display elements.
3. For ENQUEUE, check the queue full condition. If the queue is full, it is not possible to insert the item on to the queue, otherwise the new item is inserted at the rear end.
4. For DEQUEUE, operation, check the queue empty condition, If the queue is empty, it not possible to delete the item from the queue, otherwise the item which is pointed by the front is deleted from the queue.
5. For display operation, the elements from zero position up to the rear position from the queue array will be printed.

Program

```
#include <stdio.h>
#include <conio.h>
int cqueue[10];
int rear=-1;
int front=0;
void enqueue(int);
void dequeue();
void print();
void main()
{
int ch,x;
clrscr();
while(1)
{
printf("\n1.enqueue\t 2.dequeue\t 3.Printqueue\t 4.Exit\n");
printf("Enter your Choice\n");
scanf("%d",&ch);
switch(ch)
{
```

```
case 1:printf("enter the element to be inserted\n");
      scanf("%d",&x);
      printf("Before Enqueue-");
      print();
      enqueue(x);
      break;
case 2:printf("Before dequeue-");
      print();
      dequeue();
      break;
case 3:print();
      break;
case 4:exit(0);
      break;
}
}
getch();
}
void enqueue(int x)
{
rear=(rear+1)%n;
if(rear==front && cqueue[front] != -1)
printf(“queue is full”);
else
{
cqueue[rear]=x;
printf("\nAfter enqueue-");
print();
}
}
void dequeue()
{
int del;
if(cqueue[front] == -1)
{
```

```
printf(“circular queue is empty”)
}
else
{
del=cqueue[front];
cqueue[front]=-1;
front=(front+1)%n;
printf("\nThe deleted item is %d\n",del);
printf("After dequeue-");
print();
}
}
void print()
{
int i;
printf("Queue Contents are\n");
for(i=front;i<=rear;i++)
{
printf("%d\t",cqueue[i]);
}
}
```

Result:

Thus, the circular queue operations have been implemented successfully.

Sample Viva Questions:

1. What is the advantage of circular queue?
2. How the rear and front pointers are updated in circular queue?
3. When the circular queue is empty?

Augmented Experiment:

1. Assume there are 10 chairs in the bank for the customers. The two new customers arrived to the bank after the three customers left the bank. Can the new customers get the chair?. Write a C program to implement the above scenario.

Ex:No:4A	Linked list Implementation of Stack
Date :	

AIM:

To write a program in C to implement the stack operations using linked list.

ALGORITHM:

1. Create the stack linked list with header node and makes the top points the head.
2. Get the user choice to perform PUSH, POP or display elements.
3. The PUSH operation inserts the new element at the front of the list since the front of the list is pointed by the top pointer. Initially the element after the header node is pointed by the top pointer. After the insertion of new element, the new node will point the top and the new node is pointed by the header node.The new node becomes the new top after insertion.
4. The POP operation deletes element at the top pointer. If the stack is empty, pop cannot be performed. After the deletion, the top pointer will point the next of top and the header node will point the new top.
5. The display operation prints the element of the stack from the Top position to the last node.

Program

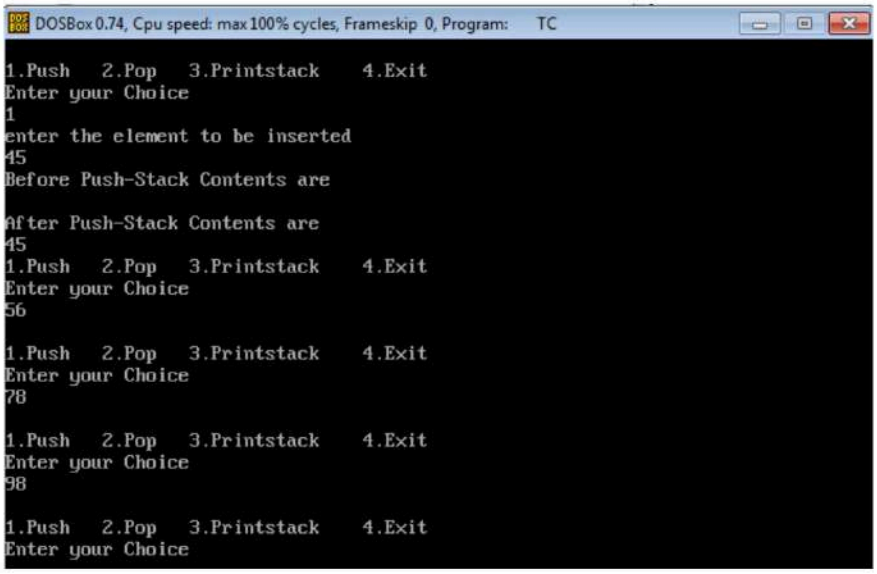
```
#include <stdio.h>
#include <conio.h>
struct stnode
{
int data;
struct stnode *next;
};
struct stnode *head, *top;
void push(int);
void pop();
void print();
void main()
{
int ch,x;
clrscr();
//Creation of head node;
head=(struct stnode *)malloc(sizeof(struct stnode));
```

```
head->data=-1;
head->next=NULL;
top=head;
while(1)
{
printf("\n1.Push\t 2.Pop\t 3.Printstack\t 4.Exit\n");
printf("Enter your Choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("enter the element to be inserted\n");
      scanf("%d",&x);
      printf("Before Push-");
      print();
      push(x);
      break;
case 2:printf("Before Pop-");
      print();
      pop();
      break;
case 3:print();
      break;
case 4:exit(0);
      break;
}
}
getch();
}
void push(int x)
{
struct stnode *newnode;
newnode=(struct stnode *)malloc(sizeof(struct stnode));
newnode->data=x;
newnode->next=head->next;
head->next=newnode;
```

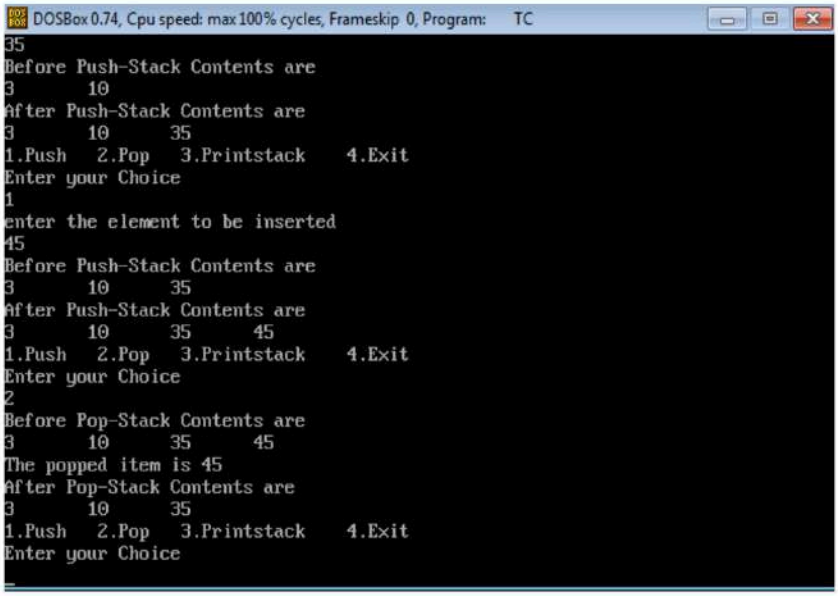
```
top=newnode;
printf("\nAfter Push-");
print();
}
void pop()
{
struct stnode *p;
if(head->next==NULL)
{
printf("Stack is Empty, Not possible to pop");
return;
}
else
{
p=top;
top=top->next;
head->next=top;
printf("\nThe popped item is %d\n",p->data);
free(p);
printf("After Pop-");
print();
}
}
void print()
{
struct stnode *p;
printf("Stack Contents are\n");
p=head->next;
while(p!=NULL)
{
printf("%d\t",p->data);
p=p->next;
}
}
```

Sample Output

Push Operation



Pop Operation and Print Stack



Result:

Thus, the stack operations have been implemented successfully using linked list.

Sample Viva Questions:

1. What is stack linked list?
2. Which node in the stack linked list is pointed by top pointer?
3. What is the empty condition for stack linked list?

Augmented Experiment:

1. Write a C program to push 3 yellow balls, 2 red balls and 4 blue balls on to the stack linked list and display the color of the 4^{th} ball deleted from the stack linked list.

Ex:No:4B	Linked list Implementation of Queue
Date :	

AIM:

To write a program in C to implement the queue operations using linked list.

ALGORITHM:

1. Create the queue linked list with header node and makes the last element in the list is pointed by the rear pointer and first element is identified with the help of head->next.
2. Get the user choice to perform ENQUEUE, DEQUEUE or display elements.
3. The ENQUEUE operation inserts the new element at the at the end of the list since the rear pointer points the end of the list. After the insertion of new element, the new node will be pointed by the rear pointer.
4. The DEQUEUE operation deletes element at the front of the list which is identified by the head->next. Before deletion, we have to check whether the queue is empty or not. If the queue is empty, the deletion is not possible.
5. The display operation prints the element of the queue from the head->next to rear.

Program.

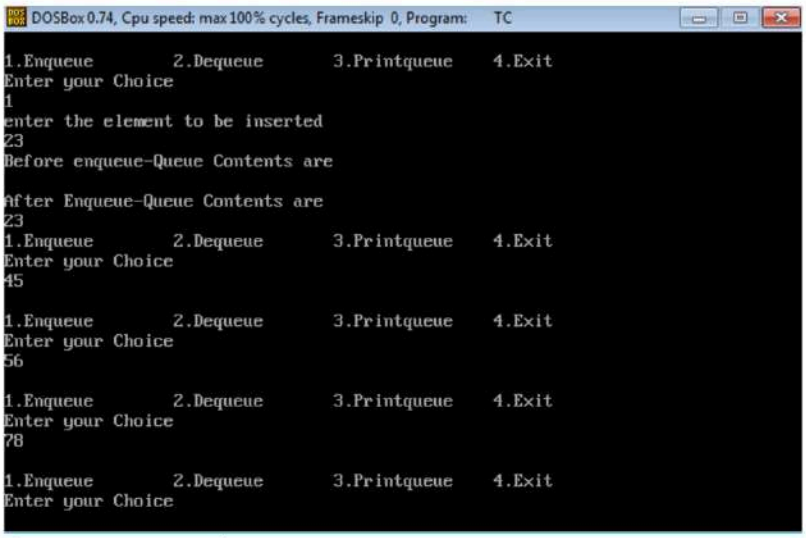
```
#include <stdio.h>
#include <conio.h>
struct qnode
{
int data;
struct qnode *next;
};
struct qnode *head, *rear;
void enqueue(int);
void dequeue();
void print();
void main()
{
int ch,x;
clrscr();
//Creation of head node;
head=(struct qnode *)malloc(sizeof(struct qnode));
head->data=-1;
```

```
head->next=NULL;
rear=head;
while(1)
{
printf("\n1.Enqueue\t 2.Dequeue\t 3.Printqueue\t 4.Exit\n");
printf("Enter your Choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("enter the element to be inserted\n");
      scanf("%d",&x);
      printf("Before enqueue-");
      print();
      enqueue(x);
      break;
case 2:printf("Before dequeue-");
      print();
      dequeue();
      break;
case 3:print();
      break;
case 4:exit(0);
      break;
}
}
getch();
}
void enqueue(int x)
{
struct qnode *newnode;
newnode=(struct qnode *)malloc(sizeof(struct qnode));
newnode->data=x;
newnode->next=NULL;
rear->next=newnode;
rear=newnode;
```

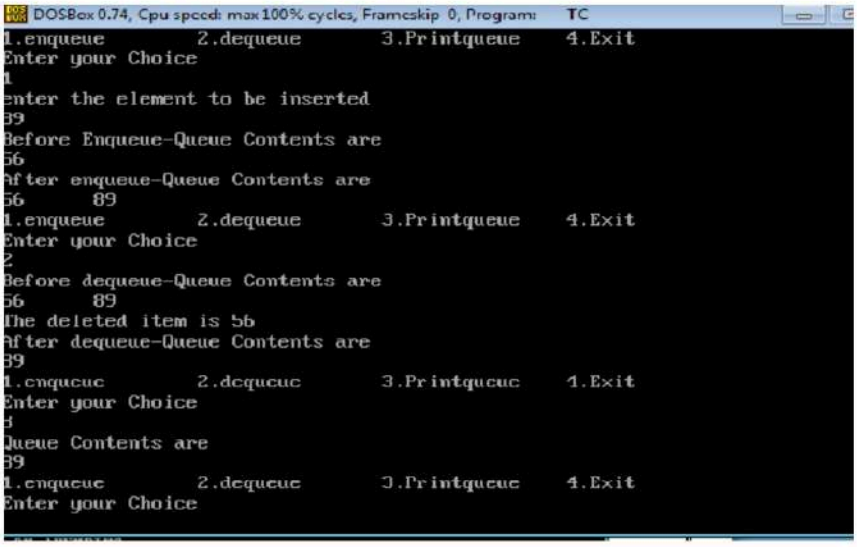
```
printf("\nAfter Enqueue-");
print();
}
void dequeue()
{
struct qnode *del;
if(head==rear)
{
printf("Queue is Empty, Not possible to pop");
return;
}
else if(rear==head->next)
rear=head;
else
{
del=head->next;
printf("\nThe deleted item is %d\n",del->data);
head->next=del->next;
}
printf("After dequeue-");
print();
}
void print()
{
struct qnode *p;
printf("Queue Contents are\n");
p=head->next;
while(p!=NULL)
{
printf("%d\t",p->data);
p=p->next;
}
}
```

Sample Output:

Enqueue Operation



Dequeue operation:



Result:

Thus, the queue operations have been implemented successfully using linked list.

Sample Viva Questions:

1. What is queue linked list?
2. Which node in the queue linked list is pointed by rear pointer?
3. What is front pointer in the queue linked list?
4. What is the empty condition for queue linked list?

Augmented Experiment:

1. Write a C program to implement the token distribution in the hospital using queue linked list.

Ex:No:5	Implementation of conversion of Infix expression into Postfix expression
Date :	

Aim

To write a C program to convert the infix expression to postfix notation.

Algorithm:

Step 1 : Append '#' delimiter at the end of given infix expression.

Step 2 : Push '#' onto the stack.

Step 3 : Read the character 'ch' from Infix expression, one character at a time.

Step 4 : Test 'ch'

Step 4.1: If 'ch' is an operand, append it with Postfix expression.

Step 4.2: If 'ch' is an operator, do the following,

Step 4.2.1 : If the stack contains '#' or '(' on the top, push the 'ch' on to the stack.

Step 4.2.2 : If 'ch' is '(', push it onto the stack.

Srep 4.2.3 : If 'ch' is ')', pop the symbols from the stack and append them onto the postfix expression until '(' is encountered. Then discard both '(' and ')'.

Step 4.2.4 : If the 'ch' has higher priority than the operator in the top of the stack, push the 'ch' on to the stack.

Step 4.2.5 : If the 'ch' has lower priority than the operator in the top of the stack, pop the symbols from the stack and append them onto the postfix expression until 'ch' has lower priority. Then push 'ch' on to the stack.

Step 4.2.6 : If 'ch' has the same priority with the top of the stack then use the associativity rules. If the associativity is from left to right, then pop and append the top of the stack to the postfix and push the 'ch' onto the stack. If the associativity is from right to left, then push 'ch' onto the stack.

Step 4.2.7 : If the 'ch' is '#', then pop all the symbols from the stack and append them onto the postfix expression.

Step 5 : Repeat steps 3-4 until '#' delimiter is encountered in the infix expression.

Step 6 : Print the postfix expression.

Program:

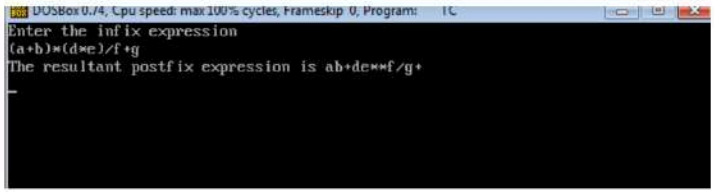
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
struct infixpriority
```

```
{
char ope;
int pri;
};
struct stackpriority
{
char ope;
int pri;
};
struct infixpriority ip[]={{'*',2},{'/',2},{'+',1},{'-',1},{'(',3},{')',0},{'#',0}};
struct stackpriority sp[]={{'*',2},{'/',2},{'+',1},{'-',1},{'(',0},{')',0},{'#',0}};
void main()
{
int i,k,n,top=-1,j=0,ipc,spc;
char stack[50],c,infixex[20],postfix[20],ch;
clrscr();
printf("Enter the infix expression\n");
scanf("%s",infixex);
n=strlen(infixex);
infixex[n]='#';
top=top+1;
stack[top]='#';
for(k=0;k<=n;k++)
{
ch=infixex[k];
if(ch==')')
{
c=stack[top];
top=top-1;
while(c!='(')
{
   postfix[j]=c;
   j=j+1;
   c=stack[top];
   top=top-1;
```

```
}
}
else if(ch=='#')
{
c=stack[top];
top=top-1;
while(c!='#')
{
   postfix[j]=c;
   j=j+1;
   c=stack[top];
   top=top-1;
}
}
else if(ch=='*' || ch=='/' || ch=='+' || ch=='-' ||ch=='(')
{
   c=stack[top];
   top=top-1;
   for(i=0;i<7;i++)
   {
   if(ch==ip[i].ope)
     ipc=ip[i].pri;
   }
   for(i=0;i<7;i++)
   {
   if(c==sp[i].ope)
     spc=sp[i].pri;
   }
   while(spc>=ipc)
   {
   postfix[j]=c;
   j=j+1;
   c=stack[top];
   top=top-1;
   for(i=0;i<7;i++)
```

```
    {
    if(c==sp[i].ope)
     spc=sp[i].pri;
    }
    }
    top=top+1;
    stack[top]=c;
    top=top+1;
    stack[top]=ch;
}
else
 {
    postfix[j]=ch;
    j=j+1;
 }
}
postfix[j]='\0';
printf("The resultant postfix expression is %s\n",postfix);
getch();
}
```

Sample Output:



Result:

Thus, the conversion of infix expression to postfix expression has been successfully implemented.

Sample Viva Questions:

1. What is postfix expression?
2. Why infix expression has to be converted into postfix expression?

Augmented Experiment:

1. Write a C program to evaluate postfix expression using stack.

Ex:No:6	Implementation of Binary Search Trees
Date :	

Aim

To write a C program to implement the operations of Binary Search Tree.

Algorithm:

1. Create the root node for the binary search tree.
2. Get the choice from the user to perform insert, delete, find, find min, find max or display the elements of the binary search tree.
3. The Find operation searches the given element in the binary search tree. If the element is present in the binary search tree, the address of the node containing the element is returned, otherwise NULL is returned.
4. The findmin operation finds the minimum value in the binary search tree and returns the address of the node contains the minimum element. The findmin operation starts the process from the root node and keeps on moving in left direction until there is a left child. The stopping point is the minimum value node.
5. The findmax operation finds the maximum value in the binary search tree and returns the address of the node contains the maximum element. The findmax operation starts the process from the root node and keeps on moving in right direction until there is a right child. The stopping point is the maximum value node.
6. The insertion operation in binary search tree inserts a new element into it at the leaf level. The insertion operation starts the process from the root node. If the new element is less than the root then it is inserted at the left subtree of the root. If it is greater than the root the new element is inserted at the right subtree of the root.
7. The deletion operation in binary search tree deletes the node from binary search tree, if the value to be deleted is present in the tree.
8. The display operation prints the elements in the binary search tree based on any one of the traversal order.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct bstnode
{
int data;
struct bstnode *left;
```

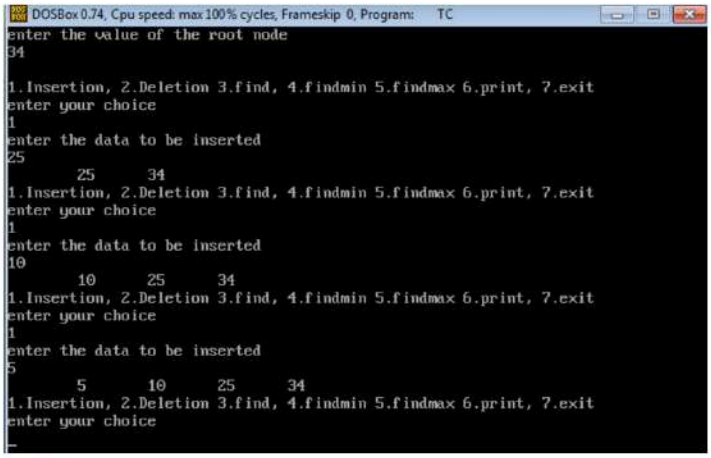
```
struct bstnode *right;
};
struct bstnode *insert(struct bstnode *,int);
struct bstnode *delete1(struct bstnode *,int);
void find(struct bstnode *,int);
struct bstnode *findmin(struct bstnode *);
struct bstnode *findmax(struct bstnode *);
void print(struct bstnode *);
void main()
{
int ch,x;
struct bstnode *t,*t1;
clrscr();
printf("enter the value of the root node\n");
scanf("%d",&t->data);
t->left=NULL;
t->right=NULL;
while(1)
{
printf("\n1.Insertion, 2.Deletion 3.find, 4.findmin 5.findmax 6.print, 7.exit\n");
printf("enter your choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("enter the data to be inserted\n");
      scanf("%d",&x);
      t=insert(t,x);
      print(t);
      break;
case 2:printf("enter the data to be removed\n");
      scanf("%d",&x);
      t=delete1(t,x);
      print(t);
      break;
case 3:printf("enter the data to be searched\n");
```

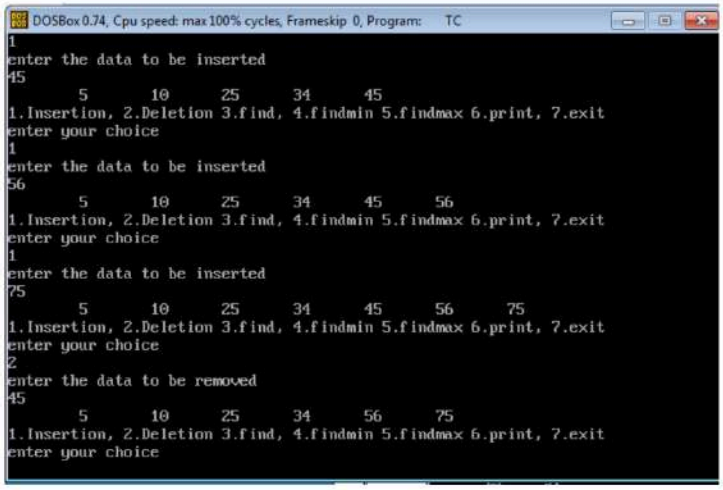
```
        scanf("%d",&x);
        find(t,x);
        break;
case 4:t1=findmin(t);
        printf("The min value is %d\n",t1->data);
        break;
case 5:t1=findmax(t);
        printf("The max value is %d\n",t1->data);
        break;
case 6:print(t);
        break;
case 7:exit(0);
}
}
getch();
}
struct bstnode *insert(struct bstnode *t,int x)
{
if(t==NULL)
{
t=(struct bstnode *)malloc(sizeof(struct bstnode));
t->data=x;
t->left=NULL;
t->right=NULL;
}
else if(x<t->data)
t->left=insert(t->left,x);
else if(x>t->data)
t->right=insert(t->right,x);
return t;
}
struct bstnode *delete1(struct bstnode *t,int x)
{
struct bstnode *tmp;
if(t==NULL)
```

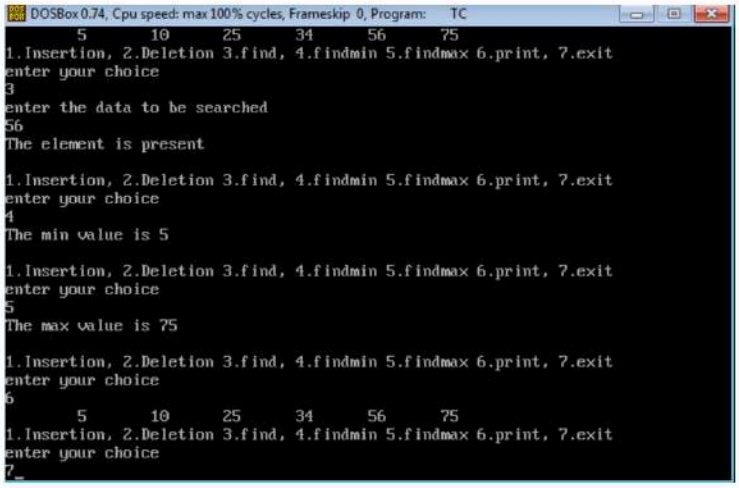
```
printf("The elment is not present in the tree\n");
else if(x<t->data)
t->left=delete1(t->left,x);
else if(x>t->data)
t->right=delete1(t->right,x);
else
{
if(t->left!=NULL &&t->right!=NULL)
{
tmp=findmin(t->right);
t->data=tmp->data;
t->right=delete1(t->right,t->data);
}
else
{
tmp=t;
if(t->left==NULL)
t=t->right;
if(t->right==NULL)
t=t->left;
free(tmp);
}
}
return t;
}
void find(struct bstnode *t,int x)
{
if(t==NULL)
{
printf("The element is not present in the list\n");
return;
}
if(x<t->data)
find(t->left,x);
if(x>t->data)
```

```
find(t->right,x);
else
printf("The element is present\n");
}
struct bstnode *findmin(struct bstnode *t)
{
while(t->left!=NULL)
{
t=t->left;
}
return t;
}
struct bstnode *findmax(struct bstnode *t)
{
while(t->right!=NULL)
{
t=t->right;
}
return t;
}
void print(struct bstnode *t)
{
if(t==NULL)
return;
else
{
print(t->left);
printf("\t%d",t->data);
print(t->right);
}
}
```

Sample Output:







Result

Thus, the C program for binary search tree operations have been implemented successfully.

Sample Viva Questions:

1. What is binary search tree property?
2. What are three cases in the deletion of the binary search tree?

Augmented Experiment:

1. Write a C program to find the median value in the binary search tree.

Ex:No:7	Implementation of AVL Trees
Date :	

Aim

To learn the implementations of AVL tree.

AVL Trees (or) Height Balanced Trees

AVL Trees are introduced by Adelson- Velskii – Landis. AVL trees are also called as height balanced trees.

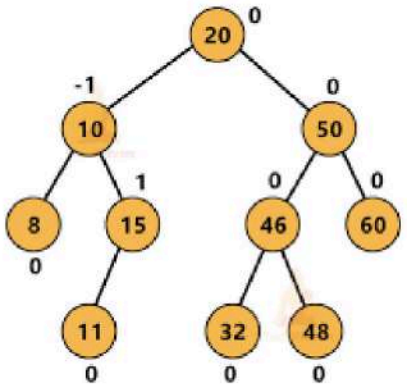
Definition of AVL Tree:

The binary search tree is said to be AVL tree iff all the nodes of the tree have the balance factor either +1, -1 or 0.

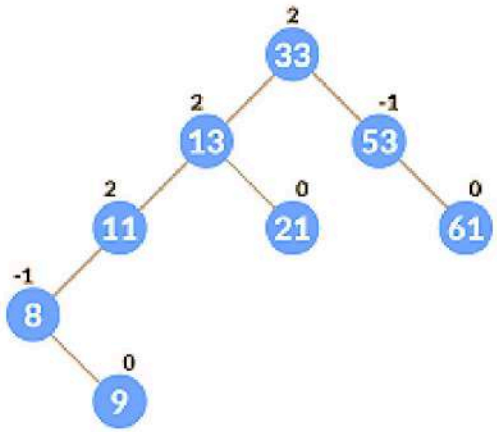
Balance factor of the node:

Balance factor of the node is defined as the difference between the height of the left subtree and height of the right subtree.

Ex:1 : AVL Tree



Ex:2 : Not an AVL Tree



Which makes the tree as unbalanced?

The repeated insertions and deletions make the tree as unbalanced. The AVL rotations are applied over the unbalanced tree to make the tree as balanced.

AVL Rotation:

AVL Rotations are classified into

- Single Rotation
  - Left Left Rotation (LL Rotation)
  - Right Right Rotation (RR Rotation)
- Double Rotation
  - Right Left Rotation (RL Rotation)
  - Left Right Rotation (LR Rotation)

Procedure to verify the tree as unbalanced due to insertion:

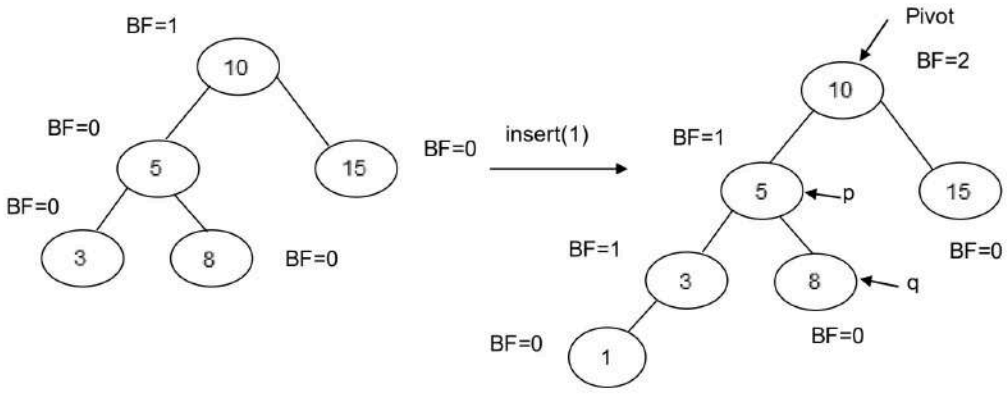
- Select the pivot node in the insertion path.
  - Pivot node is the deepest node in the insertion path whose balance factor is either +1 or -1 before insertion.
- After the insertion of the newnode, recompute the balance factor the pivot node and the below nodes in the tree.
- If the pivot node balance factor is changed from +1 to +2 (or) -1 to -2, then we can say the tree as unbalanced.

Definition of AVL tree node:

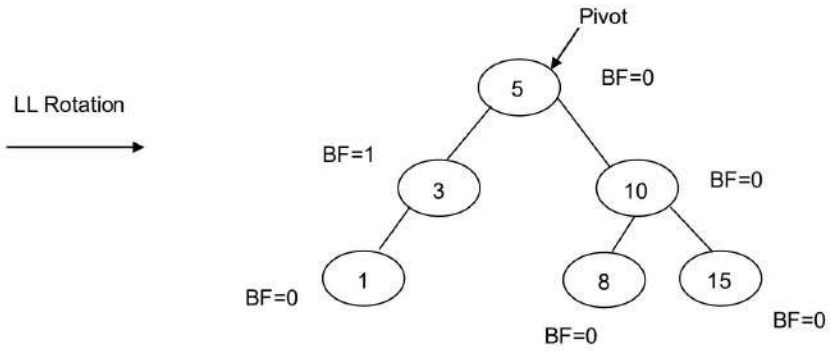
```
struct avlnode
{
int data;
int BF;
struct avlnode *left;
struct avlnode *right;
};
```

Left Left Rotation (LL Rotation):

Left Left rotation is occurred when the new node is inserted at the left subtree of left child of the pivot node.



Before Insertion **After Insertion**



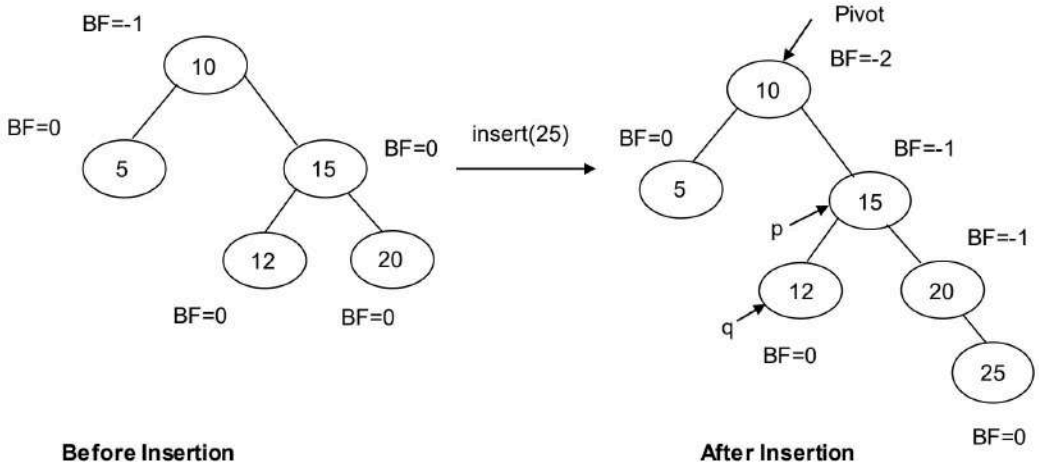
After Rotation

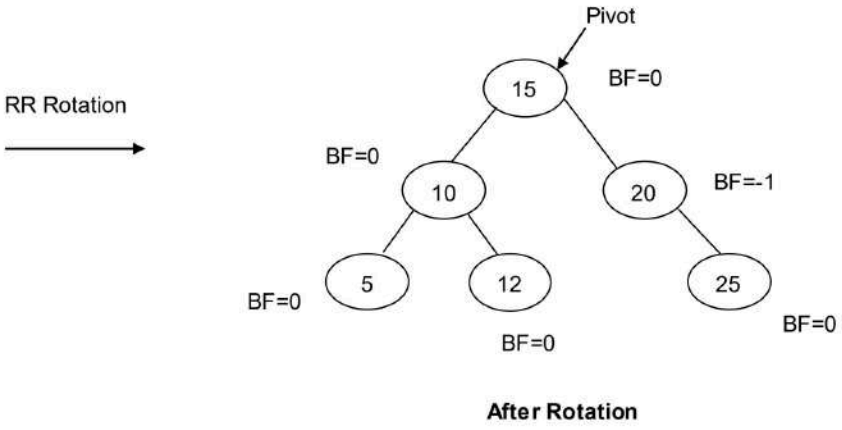
Implementation:

```
void LLrotation (struct avlnode *pivot)
{
struct avlnode *p, *q;
p=pivot->left;
q=p->right;
pivot->left=q;
p->right=pivot;
pivot=p;
pivot->BF=0;
pivot->right->BF=0;
}
```

Right Right Rotation (RR Rotation):

Right Right rotation is occurred when the new node is inserted at the right subtree of right child of the pivot node.





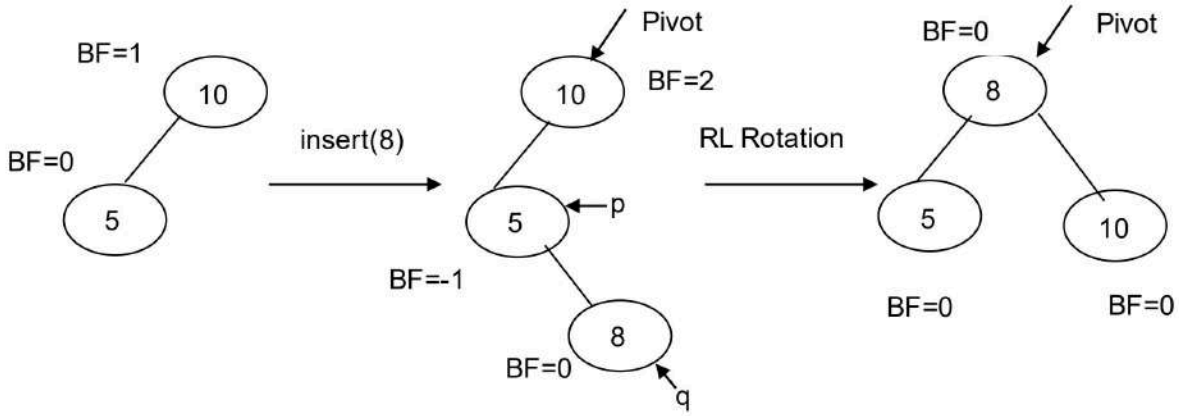
Implementation:

```
void RRrotation (struct avlnode *pivot)
{
struct avlnode *p, *q;
p=pivot->right;
q=p->left;
pivot->right=q;
p->left=pivot;
pivot=p;
pivot->BF=0;
pivot->left->BF=0;
}
```

Right Left Rotation (RL Rotation):

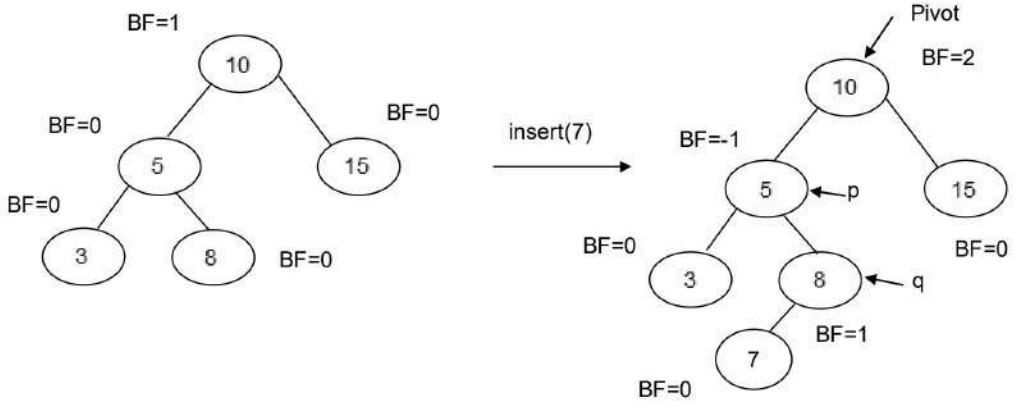
Right Left rotation is occurred when the new node is inserted at the right subtree of left child of the pivot node.

Case 1: (Right subtree of the left child of the pivot is empty)

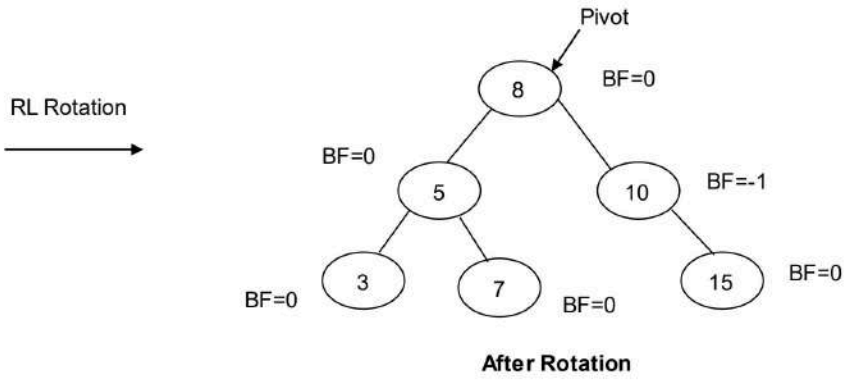


Before Insertion **After Insertion** **After Rotation**

Case 2: (New node is inserted in the left direction of right subtree of the left child of the pivot)

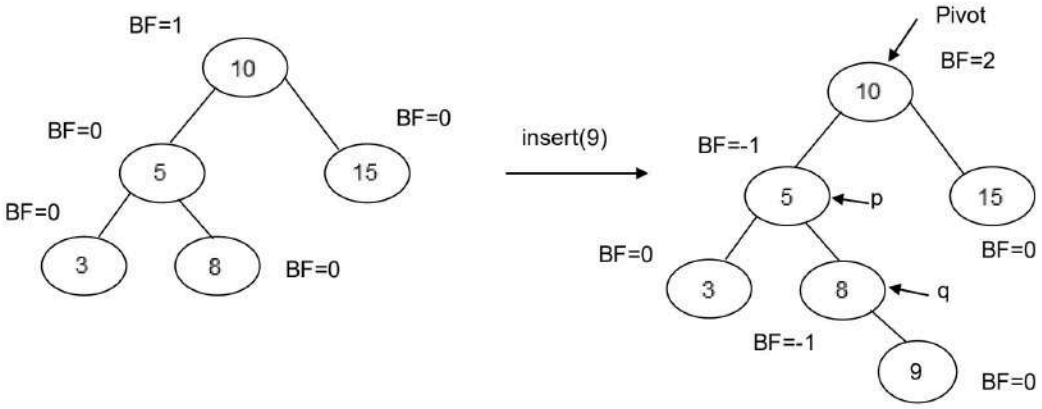


Before Insertion **After Insertion**

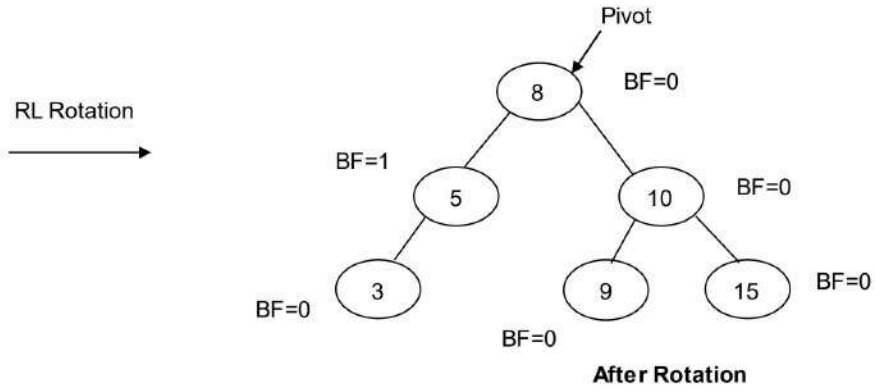


After Rotation

Case 3: (New node is inserted in the right direction of right subtree of the left child of the pivot)



Before Insertion **After Insertion**



After Rotation

Implementation:

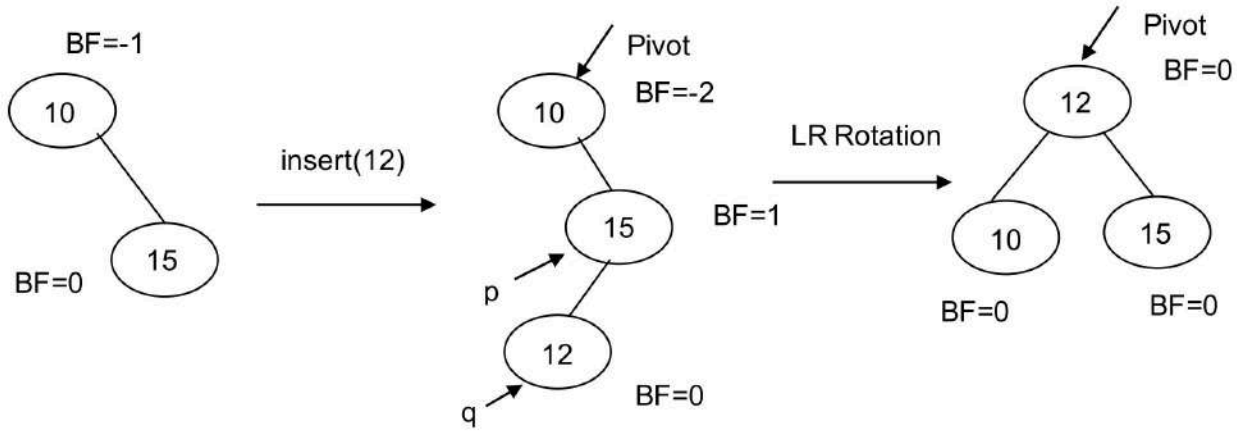
```
void RLrotation (struct avlnode *pivot)
{
struct avlnode *p, *q;
p=pivot->left;
q=p->right;
pivot->left=q->right;
p->right= q->left;
q->left=p;
q->right=pivot;
pivot=q;
if(pivot->BF==0)
{
pivot->left->BF=0;
```

```
pivot->right->BF=0;
}
else if(pivot->BF==1)
{
pivot->BF=0;
pivot->left->BF=0;
pivot->right->BF=-1;
}
else if(pivot->BF==-1)
{
pivot->BF=0;
pivot->left->BF=1;
pivot->right->BF=0;
}
}
```

Left Right Rotation (LR Rotation):

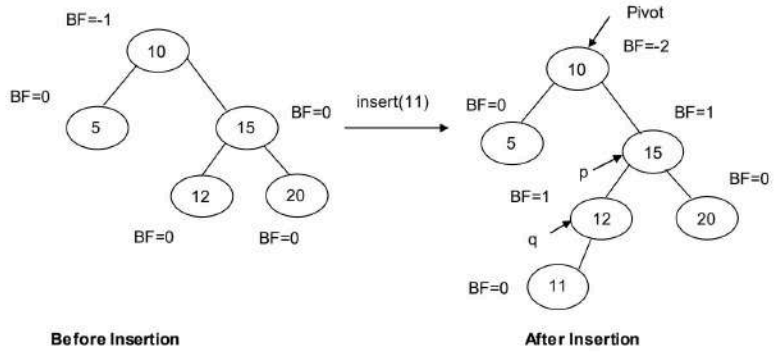
Left Right rotation is occurred when the new node is inserted at the left subtree of right child of the pivot node.

Case 1: (Left subtree of the right child of the pivot is empty)

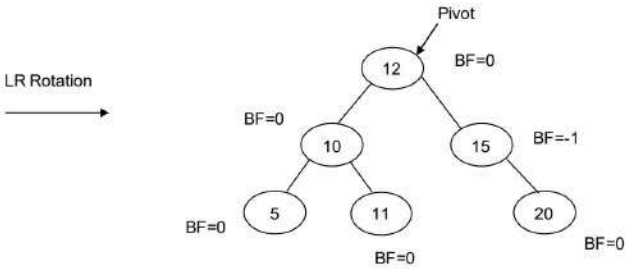


Before Insertion **After Insertion** **After Rotation**

Case 2: (New node is inserted in the left direction of Left subtree of the right child of the pivot)

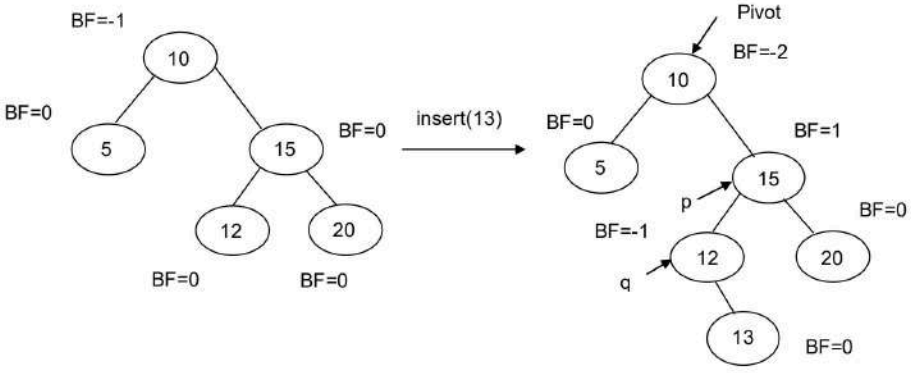


Before Insertion **After Insertion**

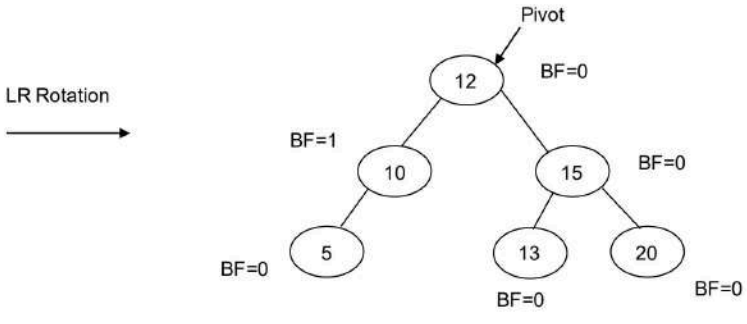


After Rotation

Case 3: (New node is inserted in the right direction of Left subtree of the right child of the pivot)



Before Insertion **After Insertion**



After Rotation

Implementation:

```
void LRrotation (struct avlnode *pivot)
{
struct avlnode *p, *q;
p=pivot->right;
q=p->left;
pivot->right=q->left;
p->left= q->right;
q->left=pivot;
q->right=p;
pivot=q;
if(pivot->BF==0)
{
pivot->left->BF=0;
pivot->right->BF=0;
}
else if(pivot->BF==1)
{
pivot->BF=0;
pivot->left->BF=0;
pivot->right->BF=-1;
}
else if(pivot->BF==-1)
{
pivot->BF=0;
pivot->left->BF=1;
pivot->right->BF=0;
}
}
```

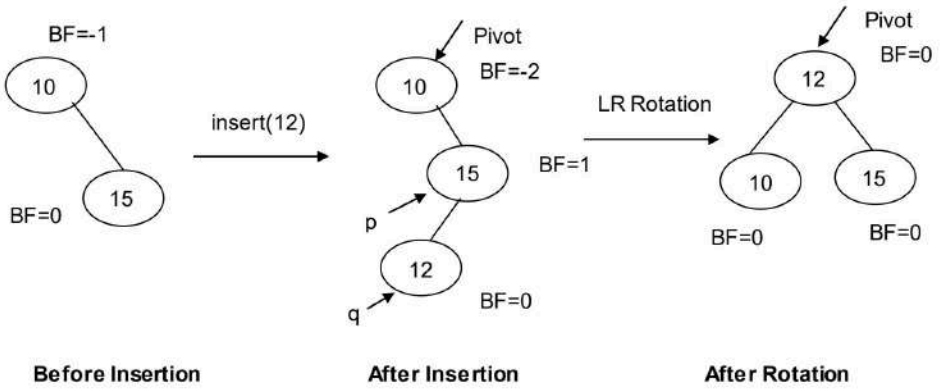
Insertion in AVL Tree:

The insertion procedure inserts the new node in the AVL tree. To insert the newnode the following has to be done.

- The insertion path in the AVL tree has to be identified to insert the newnode.
- After insertion, we have to check the tree whether it is balanced or unbalanced.

- If the tree is unbalanced, we have to apply the appropriate rotation to make the tree as balanced.
- To apply the appropriate rotation the pivot node has to be identified in the insertion path. The pivot node is the deepest node in the insertion path whose balance factor is changed from +1 to +2 or -1 to -2 due to insertion.
- The insertion position of the newnode is identified with respect to the pivot node then the appropriate rotation will be applied to make the tree as balanced.

Ex:



Implementation:

```
void avlinsert(struct avlnode *root, int x)
{
struct avlnode *in,*inp,*pivot,*q,*newnode;
newnode=(struct avlnode *)malloc(sizeof(struct avlnode));
newnode ->data=x;
newnode->BF=0;
newnode->left=NULL;
newnode->right=NULL;
if(root==NULL)
{
root=newnode;
return;
}
in=root;
inp=NULL;
pivot=root;
while(in!=NULL)
{
```

```
if(in->BF!=0)
{
pivot=in;
}
inp=in;
if(x<in->data)
in=in->left;
else
in=in->right;
}
//insertion of newnode
if(x<inp->data)
inp->left=newnode;
else
inp->right=newnode;
//revision of balance factor
q=pivot;
while(q!=newnode)
{
if(x<q->data)
{
q->BF=q->BF+1;
q=q->left;
}
else
{
q->BF=q->BF-1;
q=q->right;
}
}
//balancing the tree
if(pivot->BF>=-1 && pivot->BF<=+1) // tree is balanced
return;
if(x<pivot->data)
{
```

```
if (x<pivot->left->data)
        LLrotation(pivot);
else
        RLrotation(pivot);
}
else
{
if(x<pivot->right->data)
        LRrotation(pivot);
else
        RRrotation(pivot);
}
}
```

Applications of AVL Tree:

- AVL trees are mostly used for implementing dictionaries.
- AVL trees are also used extensively in database applications in which insertions and deletions are fewer but there are frequent lookups for data required.

Result

Thus, the operations on AVL tree and its implementations have been studied.

Sample Viva Questions:

1. What is AVL Tree?
2. What is balance factor?
3. What is AVL rotation?
4. When the rotations to be applied in the AVL tree?
5. What are the types of AVL tree rotations?

Ex:No:8	Array Implementation of Heap
Date :	

Aim

To write a C program to implement Heap using array.

Algorithm:

1. Initialize the heap array with maximum size.
2. Get the choice from the user whether to perform insertion, deletion or display the elements.
3. Insertion in binary heap inserts the new element to the heap. To insert the new element 'x' onto the heap, the hole is created in the next available location on the heap. The new element 'x' is compared with the holes parent. If the parent is minimum, then the new element is placed on the hole. Otherwise, the parent element is slide down to the hole's position and there is the hole at the parent position. This is nothing but, bubbling up the hole towards the root. This process is continued until the new element is placed in the hole. This process is known as percolate up. That is the new element is percolated up until its proper location is found.whether insertion all necessary variables and functions.
4. The deletemin operation deletes the minimum element from the heap. In Minheap, the minimum element is located in the root. If the minimum element is deleted, the hole will be created in the root position. The number of elements in the heap is also reduced by '1'. To satisfy the structure property, the last element 'x' is placed somewhere in the heap. If 'x' is placed in the hole then the deletion is completed.
5. Initially 'x' is compared with the children of the root. If 'x' is smaller than the two children of the root, then 'x' is placed in the hole. Otherwise the smallest child of the hole is slide up to the hole position. This makes the hole at the child position. This process is repeated until 'x' is placed in the hole. This process is called percolate down process.
6. The display operation displays all the elements of the heap.

Program

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int h[20];
int n=0;
```

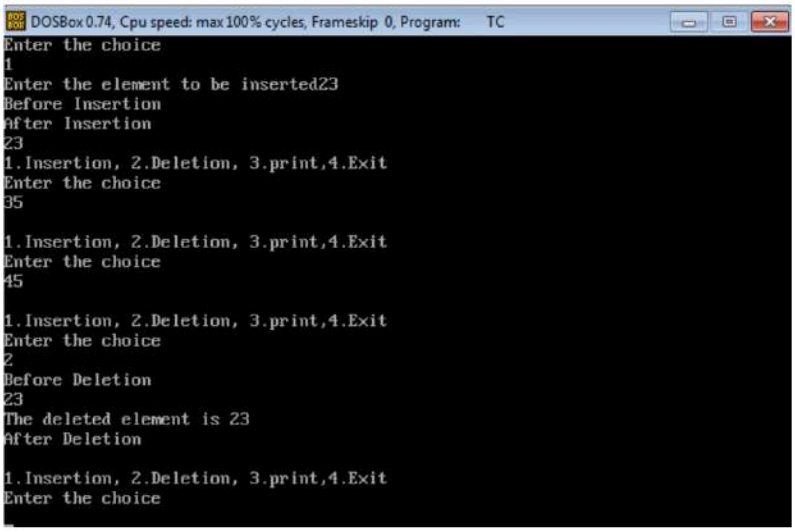
```
void insert(int);
void delete1();
void print();
void main()
{
  int ch,x;
  clrscr();
  while(1)
  {
    printf("\n1.Insertion, 2.Deletion, 3.print,4.Exit\n");
    printf("Enter the choice\n");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1: printf("Enter the element to be inserted");
              scanf("%d",&x);
              printf("Before Insertion\n");
              print();
              insert(x);
              printf("After Insertion\n");
              print();
              break;
      case 2: printf("Before Deletion\n");
              print();
              delete1(x);
              printf("After Deletion\n");
              print();
              break;
      case 3: printf("The elements of the heap\n");
              print();
      case 4: exit(0);
    }

  }
  getch();
```

```
}
void insert(int x)
{
   int i;
   n=n+1;
   for(i=n;i>1;i=i/2)
   {
      if(h[i/2]<x)
      break;
      else
      h[i]=h[i/2];
   }
   h[i]=x;
}
void delete1()
{
   int i,c,del,x;
   del=h[1];
   x=h[n];
   n=n-1;
   i=1;
   c=2*i;
   for(i=1;i*2<=n;i=c)
   {
      if(c<n && h[c]>h[c+1])
      c=c+1;
      if(x<h[c])
      break;
      else
      h[i]=h[c];
   }
   h[i]=x;
   printf("\nThe deleted element is %d\n",del);
}
void print()
```

```
{
   int i;
   for(i=1;i<=n;i++)
   {
      printf("%d\t",h[i]);
   }
}
```

Sample Output:



Result:

Thus, the implementation of heap using array has been successfully executed.

Sample Viva Questions:

1. What is priority heap?
2. What is complete binary tree?
3. What is ordering property of binary heap?
4. What is percolate up?
5. What is percolate down?

Augmented Experiment:

1. Organize the customers details of the bank along with age. When the service requested by the customers in the queue, the service will be availed by the customers based on their age. The older should get the service first. Write a C program to implement the above scenario.

Ex:No:9	Implementation of Dijkstra's Algorithm
Date :	

Aim

To write a C program to find the shortest path from source vertex to all other vertices in the weighted graph using dijikstra's algorithm.

Dijikstra's Algorithm:

Given a weighted graph G=(V,E) and a source vertex 's', the single source shortest path problem finds the shortest distance path from source vertex to every other vertices in the graph.

Dijkstra's algorithm uses distance array 'D' and path array 'P' for all the vertices of the Graph 'G'. The algorithm works as follows.

Algorithmic Steps:

1. For each vertex 'v' in Graph 'G', set D[V]=∞ & Known[v]=False, and P[V]=undefined
2. Set D[source vertex]=0,P[source vertex]=NULL
3. While there are unknown vertices in graph 'G'
    1. Select unknown vertex 'V' with lowest D[V]
    2. Set Known[v]=True
    3. For each adjacent vertex is of 'V' with Weight 'W'

        If D[V]+W<D[U] then Set D[U]=D[V]+W and P[U]=V

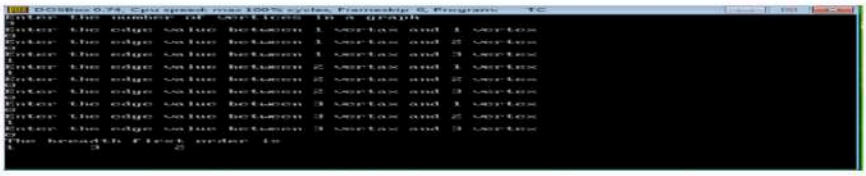
Program

```
#include <stdio.h>
#include <conio.h>
void main()
{
int g[10][10],known[10],d[10],p[10];
int n,i,j,flag,lowpos,small;
clrscr();
printf("Enter the number of vertices in the graph\n");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf("Enter the edge weight between vertex %d and vetex %d\t", i,j);
scanf("%d",&g[i][j]);
```

```
}
}
//Initialization
for(i=1;i<=n;i++)
{
known[i]=0;
d[i]=1000;
p[i]=-1;
}
d[1]=0;
//Processing the vertices in  stages
while(1)
{
flag=0;
for(i=1;i<=n;i++)
{
if(known[i]==0)
flag=1;
}
if(flag==0)
break;
small=1000;
for(i=1;i<=n;i++)
{
if(known[i]!=1)
{
if(d[i]<small)
{
lowpos=i;
small=d[i];
}
}
}
known[lowpos]=1;
for(i=1;i<=n;i++)
```

```
{
if(known[i]!=1)
{
if(g[lowpos][i]!=0)
{
if(d[lowpos]+g[lowpos][i]<d[i])
{
d[i]=d[lowpos]+g[lowpos][i];
p[i]=lowpos;
}
}
}
}
printf("\nV\tKnown\tD\tP\n");
for(i=1;i<=n;i++)
{
printf("%d\t%d\t%d\t%d\n",i,known[i],d[i],p[i]);
}
}
getch();
}
```

Sample Output:



Result:

Thus, the dijikstra's algorithm has been implemented successfully for finding the shortest path from the source vertex to all other vertices in the graph.

Sample Viva Questions:

1. What is shortest path?
2. What is weighted graph?
3. How dijikstra's algorithm works?

Augmented Experiment:

1. Organize the route map of 10 cities in the graph. Find the shortest path from city 1 to all other cities. Write a C program to implement the above scenario.

Ex:No:10	Implementation of Prim's Algorithm
Date :	

Aim

To write a C program to find a minimum spanning tree from the undirected weighted graph using Prim's algorithm.

Algorithm:

Prim's algorithm starts with a single vertex in MST and adds up adjacent vertices one by one by discovering all of the connected edges along the way. Edges with the lowest weights that don't generate cycles are chosen for inclusion in the MST structure.

Algorithmic Steps:

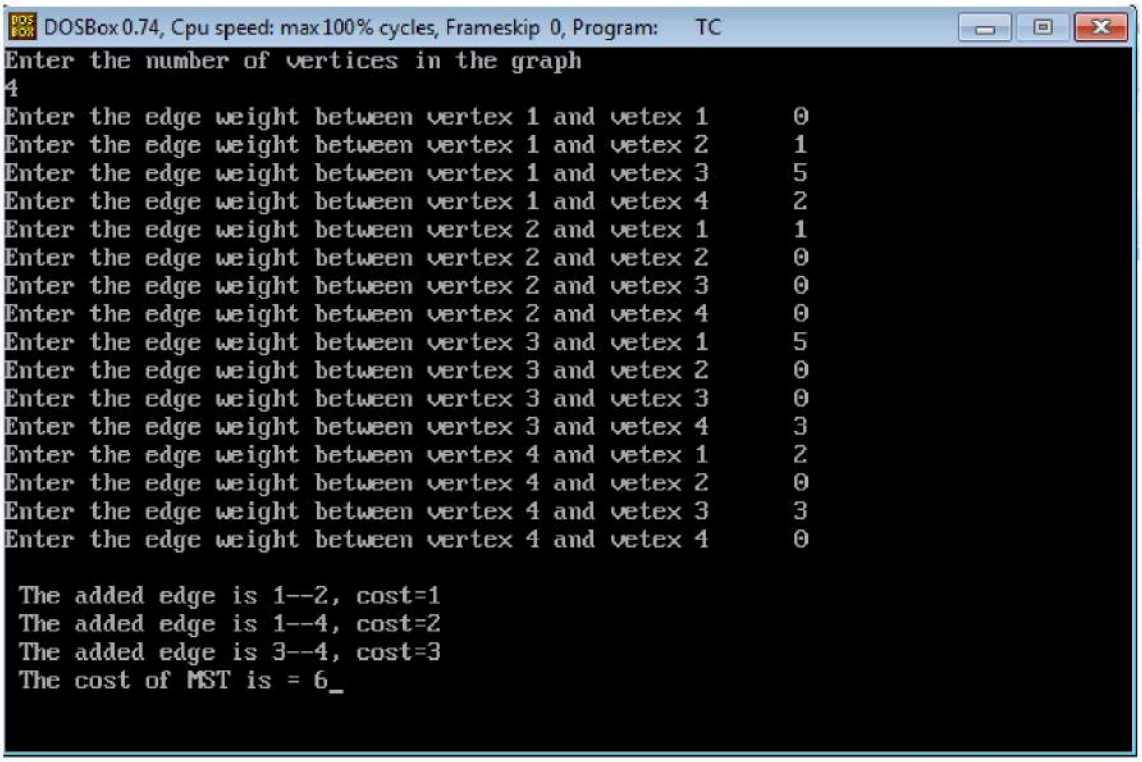
1. Select any vertex as a tree vertex.
2. Select the minimum cost edge which connects any of the tree vertex and graph vertex.
3. Add the selected edge to the MST if it doesn't form a cycle, otherwise discard it.
4. Repeat steps (2) and (3) until all the vertices are connected to the MST.

Program:

```
#include <stdio.h>
#include <conio.h>
void main()
{
int g[10][10],MST[10];
int n,i,j,k,mincost,cost=0,u,v;
clrscr();
printf("Enter the number of vertices in the graph\n");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf("Enter the edge weight between vertex %d and vetex %d\t", i,j);
scanf("%d",&g[i][j]);
}
}
//Initialization
for(i=1;i<=n;i++)
{
```

```
MST[i]=0;
}
MST[1]=1;
//Processing the vertices in  stages
for(k=1;k<=n-1;k++)
{
mincost=9999;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
if(g[i][j] && ((MST[i] && !MST[j]) || (!MST[i] && MST[j])))
{
if(g[i][j]<mincost)
{
mincost=g[i][j];
u=i;
v=j;
}
}
}
}
printf("\n The added edge is %d--%d, cost=%d",u,v,mincost);
MST[u]=1;
MST[v]=1;
cost=cost+mincost;
}
printf("\n The cost of MST is = %d",cost);
getch();
}
```

Sample Output:



Result:

Thus, the C program for determining minimum spanning tree from the undirected weighted graph has been implemented successfully.

Sample Viva Questions:

1. What is spanning Tree?
2. What is minimum spanning tree?
3. What are the algorithms are used to compute the minimum spanning tree from the graph?
4. Is the mimimum spanning tree unique for the graph?

Augmented Experiment:

1. Write a C program to determine minimum spanning tree from the graph using kruskals' algorithm.

Ex:No:11 A	Implementation of Linear Search
Date :	

Aim:

To write a C program to implement linear search algorithm.

Algorithm:

1. The array of elements and the element to be searched are given.
2. The array elements are accessed from the first element.
3. While there are elements in the array, compare the current element of the array with the element to be searched.
    i. If match is found, return the index of the current element of the array, otherwise move to the next element.
4. Return -1 to indicate the unsuccessful search.

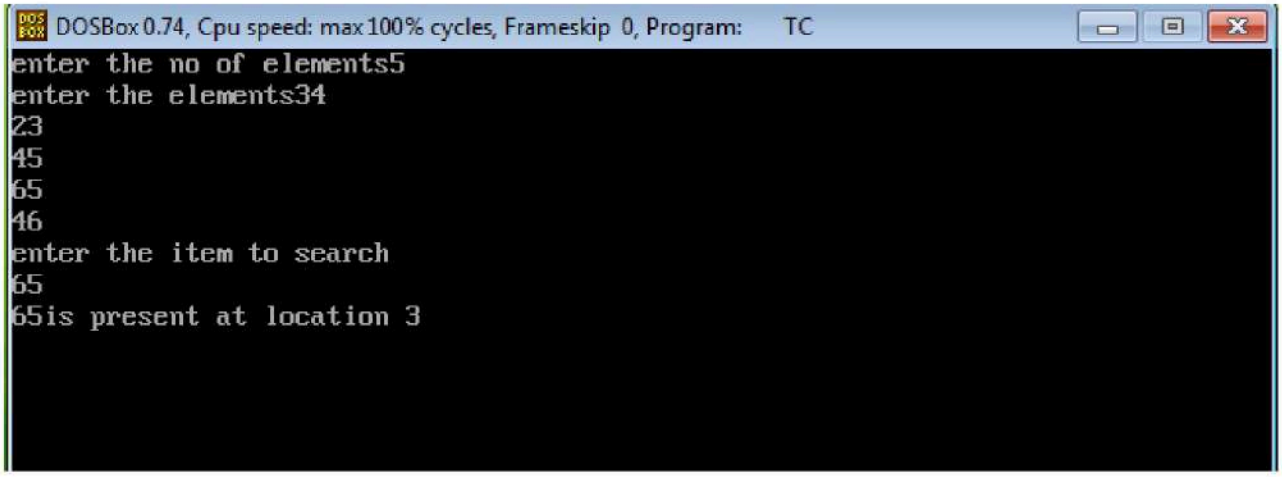
Program :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main ()
{
 int a[20],n,i,x,flag=0;
 clrscr();
 printf ("Enter the number of elements in the array\n");
 scanf("%d",&n);
 printf ("Enter the number of elements in the array one by one\n");
 for(i=0;i<n;i++)
 {
    printf ("Enter the element\n");
    scanf ("%d", &a[i]);
 }
  printf ("Enter the element to be searched\n");
 scanf("%d",&x);
 for(i=0;i<n;i++)
 {
    if(a[i]==x)
    {
```

```
        printf("The element is present at %d position",i);
        flag=1;
        break;
    }

 }
 if(flag==0)
 printf("The element to be searched is not present in the list");
 getch ();
}
```

Sample Output:



Result:

Thus, the C program for linear search algorithm has been implemented successfully.

Sample Viva Questions:

1. What is searching?
2. What are the algorithms available to search the element in the list?
3. What is the logic behind the linear search?
4. Why binary search is efficient?

Augmented Experiment:

1. Write a C program to display the position of the first proper square number in the list.

Ex:No:11 B	Implementation of Binary Search
Date :	

Aim:

To write a C program to implement binary search algorithm.

Algorithm:

1 The sorted array of elements and the element to be searched is given.

2.Set low=0, high=n-1, Where 'n' is the size of the array.

3.If low<=high, go to step 4, otherwise return -1 and halt.

4. i. Calculate mid=low+(high-low)/2.

ii. If the element to be searched is matched with the middle element, then return the position of the middle element and halt.

iii. If the element to be searched is less than the middle element, then use the first half of the array to continue the search by changing high=mid-1. Go to step 3 to continue the search.

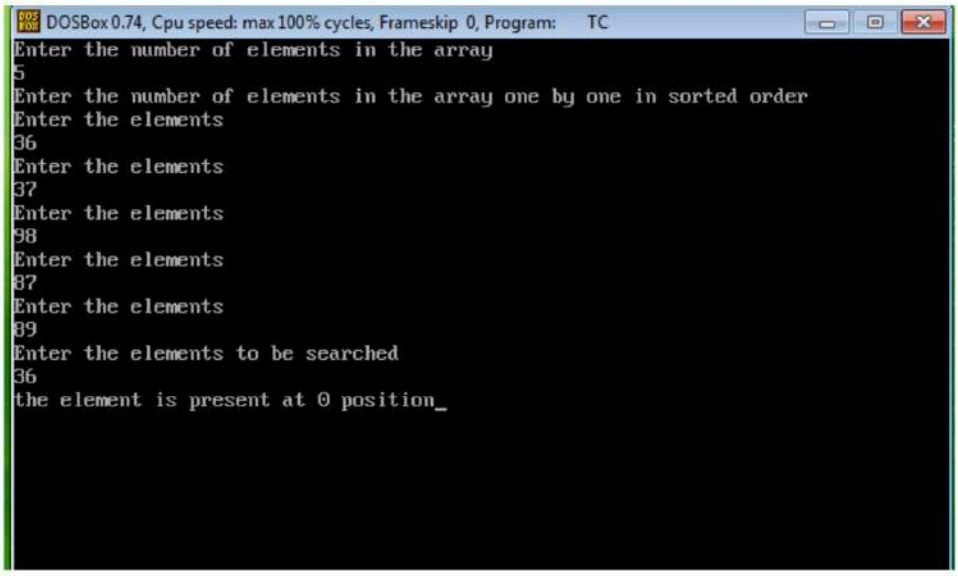
iv. If the element to be searched is greater than the middle element, then use the second half of the array to continue the search by changing low=mid+1. Go to step 3 to continue the search.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
void main ()
{
 int a[20],n,i,x,low,high,mid,flag=0;
 clrscr();
 printf ("Enter the number of elements in the array\n");
 scanf("%d",&n);
 printf ("Enter the number of elements in the array one by one in sorted order\n");
 for(i=0;i<n;i++)
 {
    printf ("Enter the element\n");
    scanf ("%d", &a[i]);
 }
printf ("Enter the element to be searched\n");
```

```
 scanf("%d",&x);
 low=0;
 high=n-1;
 while(low <= high)
 {
    mid=low+floor(((high-low)/2));
    if(x==a[mid])
     {
     printf("The element is present at %d position",mid);
     flag=1;
     break;
     }
     else if(x<a[mid])
     high=mid-1;
     else if(x>a[mid])
     low=mid+1;
 }
 if(flag==0)
 printf("The element to be searched is not present in the list");
 getch ();
}
```

Sample Output:



Result:

Thus, the C program for binary search algorithm has been implemented successfully.

Ex:No:12 A	Implementation of Selection Sort
Date :	

Aim:

To write a C program to implement the selection sort algorithm.

Algorithm:

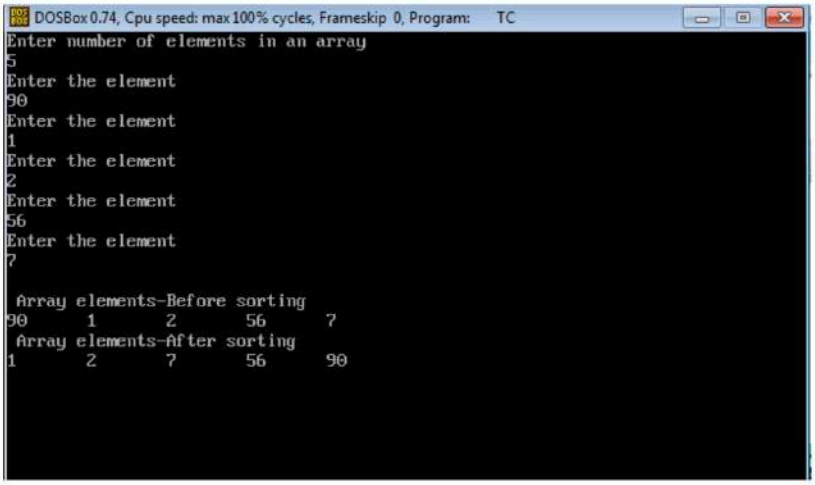
1. Selection sort scans the entire list and finds the smallest element in the list. The smallest element is exchanged with the first element and put the smallest element in its final position in the sorted list.
2. The entire list is scanned again starting from the second element to find the smallest among the last n-1 elements. The smallest element among the last 'n-1' elements is exchanged with the second element putting the second smallest element in its correct place in the sorted list.
3. This process is continued for 'n-1' times to place all the elements in its correct place in the sorted list.

Program:

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a[20],n,i,j,min,t;
clrscr();
printf("Enter number of elements in an array\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the element\n");
scanf("%d",&a[i]);
}
printf("\n Array elements-Before sorting\n");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
for(i=0;i<=n-2;i++)
```

```
{
min=i;
for(j=i+1;j<=n-1;j++)
{
if(a[j]<a[min])
{
min=j;
}
}
//Swapping
t=a[i];
a[i]=a[min];
a[min]=t;
}
printf("\n Array elements-After sorting\n");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
getch();
}
```

Sample Output :



Result:

Thus the C Program for selection sort algorithm has been successfully executed.

Ex:No:12 B	Implementation of Insertion Sort
Date :	

Aim:

To write a C program to implement the Insertion sort algorithm.

Algorithm:

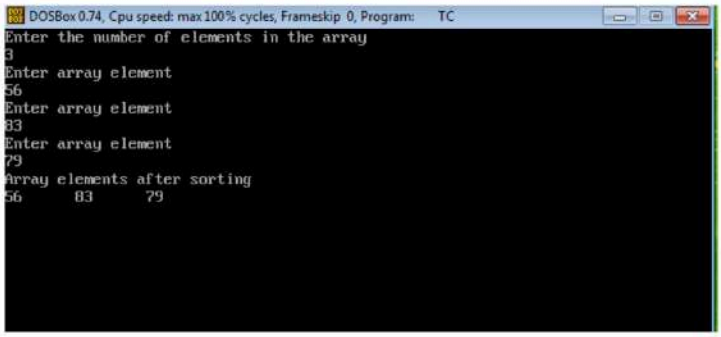
1. If it is the first element, it is already sorted.
2. Pick the next element.
3. Compare with all the elements in sorted sub-list.
4. Shift all the the elements in sorted sub-list that is greater than the value to be sorted.
5. Insert the value.
6. Repeat until list is sorted.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
   int a[10],n,i,p,j,tmp;
   clrscr();
   printf("Enter the number of elements in the array\n");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Enter array element\n");
      scanf("%d",&a[i]);
   }
  for(p=1;p<=n-1;p++)
   {
     tmp=a[p];
     for(j=p-1;j>=0;j--)
     {
        if(a[j]>tmp)
              a[j+1]=a[j];
        else
              break;
```

```
    }
    a[j+1]=tmp;
  }
  printf("Array Elements after sorting\n");
  for(i=0;i<n;i++)
  printf("%d\t",a[i]);
 getch();
}
```

Sample Output:



Result:

Thus the C Program for insertion sort algorithm has been successfully executed.

Sample Viva Questions:

1. What is sorting?
2. List the various sorting algorithms.
3. Compare selection sort with insertion sort.

Augmented Experiment:

1. Write a C program to find the top 10 rank holders of the class.

Ex:No:13	Implementation of Merge Sort
Date :	

Aim:

To write a C program to implement merge sort algorithm.

Algorithm:

1. Divide: Divide the list or array recursively into two halves until it can no more be divided.
2. Conquer: Each subarray is sorted individually using the merge sort algorithm.
3. Merge: The sorted subarrays are merged back together in sorted order. The process continues until all elements from both subarrays have been merged.

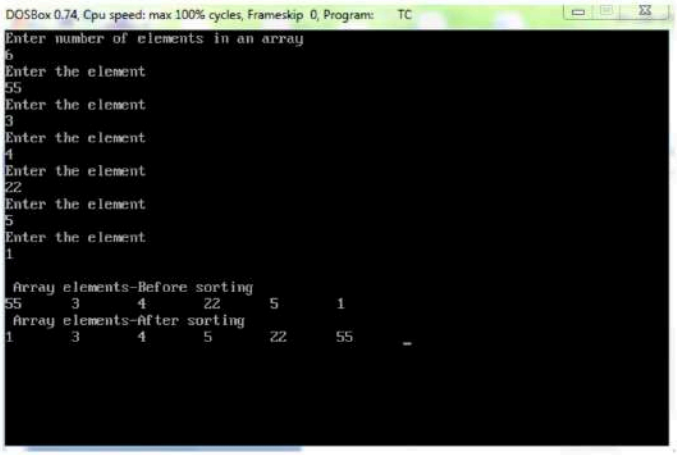
Program:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int a[20],b[20],c[20];
void mersort(int [],int,int);
void merge(int [],int,int,int);
void main()
{
int n,i,j;
clrscr();
printf("Enter number of elements in an array\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the element\n");
scanf("%d",&a[i]);
}
printf("\n Array elements-Before sorting\n");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
mersort(a,0,n-1);
printf("\n Array elements-After sorting\n");
```

```
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
getch();
}
void mersort(int a[],int low,int high)
{
int mid;
if(low<high)
{
mid=low+(high-low)/2;
mersort(a,low,mid);
mersort(a,mid+1,high);
merge(a,low,mid,high);
}
}
void merge(int a[],int low,int mid,int high)
{
int i,j,k,n1,n2;
n1=mid-low+1;
n2=high-mid;
//Copy the first half into array B
for(i=0;i<n1;i++)
b[i]=a[low+i];
//Copy the second half into array C
//j=0;
for(i=0;i<n2;i++)
{
c[i]=a[mid+1+i];
}
//merging
i=0;
j=0;
k=low;
```

```
while(i<n1 && j<n2)
{
if(b[i]<c[j])
{
a[k]=b[i];
i++;
}
else
{
a[k]=c[j];
j++;
}
k++;
}
while(i<n1)
{
a[k]=b[i];
i++;
k++;
}
while(j<n2)
{
a[k]=c[j];
j++;
k++;
}
}
```

Sample Output:



Result:

Thus the C Program to implement merge sort algorithm has been executed successfully.

Sample Viva Questions:

1. What is divide and conquer principle?
2. When the merging of two arrays will be gets over?
3. What is the minimum size of the sub array during divide phase of the merge sort?

Augmented Experiment:

1. Write a C program to list the names of the states of India according to the population.

Ex:No:14A	Implementation of Open Addressing using Linear Probing
Date :	

Aim:

To write a C program to implement hash table using linear probing technique.

Algorithm:

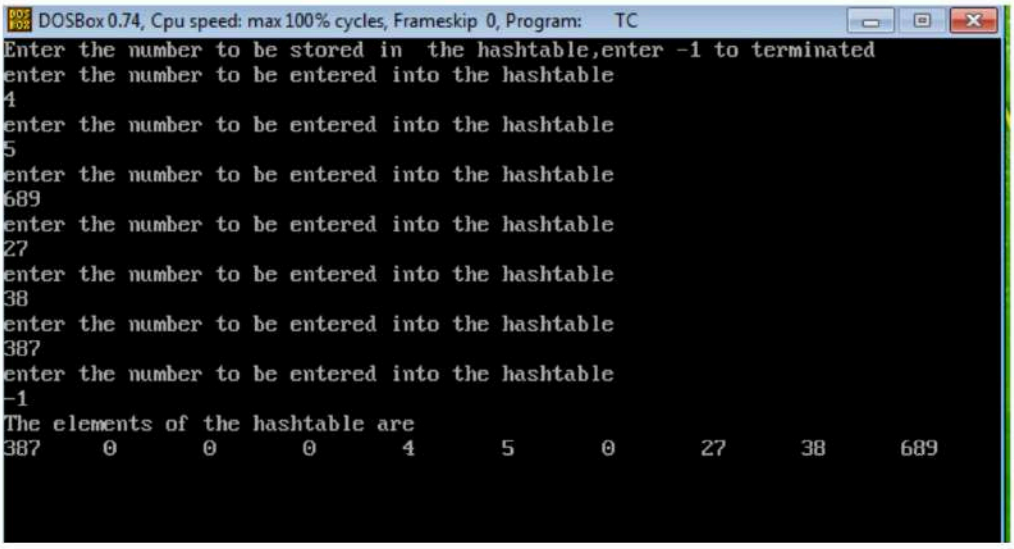
1. Calculate the initial hash value for the key.
2. If the slot is empty, place the key-value pair there.
3. If the slot is occupied, probe linearly by moving to the next slot.
4. Repeat steps 2 and 3 until an empty slot is found, and insert the key-value pair there.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void main()
{
int hashtable[10],i,x,flag;
clrscr();
//Initialization of Hash table
for(i=0;i<10;i++)
hashtable[i]=0;
printf("Enter the numbers to be stored in the hashtable, enter -1 to terminate\n");
while(1)
{
printf("enter the number to be entered into the hashtable\n");
scanf("%d",&x);
if(x==-1)
break;
flag=0;
i=x%10;
while(flag==0)
{
if(hashtable[i]==0)
{
hashtable[i]=x;
flag=1;
```

```
}
else
{
i=(i+1)%10;
}
}
}
printf("The elements of the hashtable are\n");
for(i=0;i<10;i++)
printf("%d\t",hashtable[i]);
getch();
}
```

Sample Output:



Result:

Thus the implementation of hash table using linear probing has been successfully executed.

Ex:No:14B	Implementation of Open Addressing using Quadratic Probing
Date :	

Aim:

To write a C program to implement hash table using quadratic probing technique.

Algorithm:

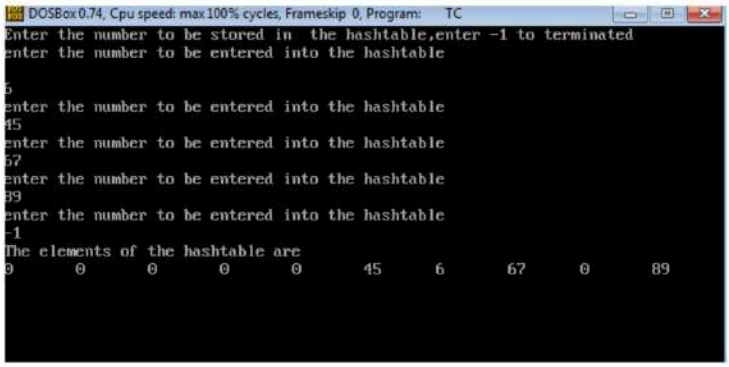
1. Calculate the initial hash position for the key.
2. If the position is occupied, apply the quadratic probing formula to find the next available slot.
3. Repeat this process until an empty slot is found, and insert the data.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void main()
{
int hashtable[10],i,j,k,x,flag;
clrscr();
//Initialization of Hash table
for(i=0;i<10;i++)
hashtable[i]=0;
printf("Enter the numbers to be stored in the hashtable, enter -1 to terminate\n");
while(1)
{
printf("enter the number to be entered into the hashtable\n");
scanf("%d",&x);
if(x==-1)
break;
flag=0;
i=x%10;
k=i;
j=1;
while(flag==0)
{
if(hashtable[i]==0)
{
```

```
hashtable[i]=x;
flag=1;
}
else
{
i=(k+(j*j))%10;
j=j+1;
}
}
}
printf("The elements of the hashtable are\n");
for(i=0;i<10;i++)
printf("%d\t",hashtable[i]);
getch();
}
```

Sample Output:



Result:

Thus the implementation of hash table using Quadratic probing has been successfully executed.

Sample Viva Questions:

1. What is hashing?
2. What is hash function?
3. What is collision in hashing?
4. What are collision resolution techniques?
5. How linear probing and quadratic probing addresses the collision problem?

Augmented Experiment:

1. Write a C program to store the details of the students roll no into the structure so that the record will be searched with fewer attempts.