# DATA STRUCTURE LAB

**(20MCA135)**

# LAB RECORD

Submitted in partial fulfilment of the requirements for the award of the degree of Master of Computer Applications of A P J Abdul Kalam Technological University

**Submitted by:**

**GOKUL BIJU (SJC22MCA-2028)**

## MASTER OF COMPUTER APPLICATIONS

**ST.JOSEPH'S COLLEGE OF ENGINEERING AND TECHNOLOGY, PALAI**

**CHOONDACHERRY P.O, KOTTAYAM**

**KERALA**

**February 2023**

# ST. JOSEPH' S COLLEGE OF ENGINEERING AND TECHNOLOGY, PALAI

**(An ISO 9001: 2015 Certified College)**

**CHOONDACHERRY P.O, KOTTAYAM KERALA**

## CERTIFICATE

This is to certify that the Data Structure Lab Record (20MCA135) submitted by Gokul Biju student of First semester MCA at ST. JOSEPH'S COLLEGE OF ENGINEERING AND TECHNOLOGY, PALAI in partial fulfilment for the award of Master of Computer Applications is a bonafide record of the lab work carried out by him under our guidance and supervision. This record in any form has not been submitted to any other University or Institute for any purpose.

**Prof. Anish Augustine K** **Prof. Anish Augustine**

**Faculty In- Charge** **(HoD In Charge-MCA)**

Submitted for the End Semester Examination held on _____

**Examiner 1:**

**Examiner 2:**

# DECLARATION

I Gokul Biju, do hereby declare that the Data Structure Lab Record (20 MCA 135) is a record of work carried out under the guidance of Mr. Anish Augustine, Asst.Professor, Department of Computer Applications, SJCET, Palai as per the requirement of the curriculum of Master of Computer Applications Programme of A P J Abdul Kalam Technological University, Thiruvananthapuram. Further, I also declare that this record has not been submitted, full or part thereof, in any University / Institution for the award of any Degree / Diploma.

Place: Choondacherry GOKUL BIJU

Date : (SJC22MCA-2028)

# CONTENT

**1. Linear Search Implementation**

**Program :**

```
#include<stdio.h>
void main()  {
        int a[100] ,n ,s;
        printf("Enter the array size");
        scanf("%d" ,&n);
        printf("Enter the array elements");
        for(int i=0;i<n ; i++) {
             scanf("%d",&a[i]);  }
        printf("Enter the element to be searched\n");
        scanf("%d",&s);
        for(int i=0;i<n;i++) {
            if(a[i]==s)   {
                printf("Element found at position\t %d \n",i+1);
                break;    }
             if(i==n){
                  printf("Element not found\n");
             }}}
```

**Output:**

```
Enter the array size5
Enter the array elements1
8
5
3
4
Enter the element to be searched
4
Element found at position       5
```

## 2.Binary Search

**Program :**

```
#include<stdio.h>
void main()  {
        int a[100],n,s,first,last,mid,flag;
        printf("Enter the array size");
        scanf("%d",&n);
        printf("Enter the array elements");
        for(int i=0;i<n;i++)  {
                scanf("%d",&a[i]);
        }
        printf("Enter the element to be searched\n");
        scanf("%d",&s);
        first=0;
        last=n-1;
        flag=0;
        while(first<=last) {
                mid=(first+last)/2;
                if(s==a[mid]) {
                        flag=1;
                        break;
                 }
                else if(s>a[mid]) {
                        first=mid+1;
                 }
                else{
                   last=mid-1;
                 }
```

```
        }
            if(flag==0)
                printf("\n  The number is not found");
            else
                printf("\n  The number is found and its position is: %d\n",mid+1);
}
```

```
Enter the array size5
Enter the array elements12
13
44
55
33
Enter the element to be searched
13

  The number is found and its position is: 2
```

**3. Array Insertion**

**Program :**

```
#include <stdio.h>
void main() {
            int array[100], item, pos,size;
            printf("Enter the size of array\n");
            scanf("%d",&size);
            printf("Enter the elements of array\n");
            for(int i=0;i<size;i++)
               scanf("%d",&array[i]);
            printf("Enter the element to be inserted in the array\n");
            scanf("%d",&item);
            printf("Enter the position element to be inserted in the array\n");
            scanf("%d",&pos);
            size=size+1;
            for(int i=size-1;i>=pos;i--)
```

```
            array[i]=array[i-1];
        array[pos-1]=item;
        printf("new array elements are\n");
        for(int i=0;i<size;i++)
            printf("%d \n",array[i]);
}
```

**Output :**

```
Enter the size of array
4
Enter the elements of array
1
2
3
4
Enter the element to be inserted in the array
5
Enter the position element to be inserted in the array
5
new array elements are
1
2
3
4
5
```

**4.Array Deletion**

**Program:**

```
#include <stdio.h>
void main() {
        int array[100],pos,size;
        printf("Enter the size of array\n");
        scanf("%d",&size);
        printf("Enter the elements of array\n");
        for(int i=0;i<size;i++)
            scanf("%d",&array[i]);
```

```
        printf("Enter the position element to be deleted in the array\n");
        scanf("%d",&pos);
        for(int i=pos-1;i<size-1;i++)
            array[i]=array[i+1];
        printf("array elements after deletion are\n");
        for(int i=0;i<size-1;i++)
            printf("%d \n",array[i]);
}
```

**Output:**

```
Enter the size of array
5
Enter the elements of array
1
2
3
4
5
Enter the position element to be deleted in the array
5
array elements after deletion are
1
2
3
4
```

**5 .To merge two arrays into a third array**

**Program :**

```
#include <stdio.h>
void main(){
        int array1[100],array2[100],array3[100],size1,size2,size3;
        printf("Enter the size of 1st array\n");
        scanf("%d",&size1);
        printf("Enter the elements of 1st array\n");
        for(int i=0;i<size1;i++)
            scanf("%d",&array1[i]);
```

```
        printf("Enter the size of 2nd array\n");
            scanf("%d",&size2);
        printf("Enter the elements of 2nd array\n");
        for(int i=0;i<size2;i++)
            scanf("%d",&array2[i]);
        size3=size1+size2;
        for(int i=0;i<size1;i++)
            array3[i]=array1[i];
        for(int i=0;i<size2;i++)
            array3[i+size1]=array2[i];
        printf("array elements before sorting\n");
        for(int i=0;i<size3;i++)
            printf("%d \n",array3[i]);
        for(int i = 0; i < size3; i++) {
            int temp;
            for(int j = i + 1; j < size3; j++)  {
                if(array3[i] > array3[j]) {
                    temp = array3[i];
                    array3[i] = array3[j];
                    array3[j] = temp;
                }
            }
        }
        printf("array elements after sorting\n");
        for(int i=0;i<size3;i++)
            printf("%d \n",array3[i]);

}
```

**Output:**

```
Enter the size of 1st array
4
Enter the elements of 1st array
1
2
3
4
Enter the size of 2nd array
3
Enter the elements of 2nd array
5
6
7
array elements before sorting
1
2
3
4
5
6
7
array elements after sorting
1
2
3
4
5
6
7
```

**6. Matrix Operations**

**Program :**

```
#include <stdio.h>
void main() {
        int m1[3][3],m2[3][3],m3[3][3];
        printf("Enter the elements of first matrix\n");
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++)
                scanf("%d",&m1[i][j]);
        }
        printf("Enter the elements of second matrix\n");
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++)
                scanf("%d",&m2[i][j]);
```

```
        }
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++)
                m3[i][j]=m1[i][j]+m2[i][j];
        }
        printf("Matrix after addition\n");
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                printf("%d\t",m3[i][j]);
            }
            printf("\n");
        }
}
```

**Output:**

```
Enter the elements of first matrix
1
2
3
4
5
6
7
8
9
Enter the elements of second matrix
9
8
7
6
5
4
3
2
1
Matrix after addition
10      10      10
10      10      10
10      10      10
```

**7.Stack Operation**

**Program:**

```
#include<stdio.h>
void push();
void pop();
void peek();
int isfull();
int isempty();
int stack[100],maxsize,top=-1;
void main() {
        printf("Enter the stack size");
        scanf("%d",&maxsize);
        isempty();
        isfull();
        peek();
        int item;
        printf("Enter the element to be inserted ");
        scanf("%d",&item);
        push(item);
        int term;
        printf("Enter 1 to delete an element from stack\n ");
        scanf("%d",&term);
        if(term==1)
            pop();
}
int isempty() {
        if(top==-1) {
            printf("The stack is empty\n");
```

```
                return 0;
            }
        else
                return 1;
}
int isfull() {
    if(top==maxsize) {
        printf("The stack is full\n");
        return 0;
      }
     else
        return 1;
}
void peek() {
     printf("The peek of stack is %d\t\n",top);
}
void push(int data) {
    if(isfull()==1) {
        top=top+1;
        stack[top]=data;
        printf("The element %d is inserted \n",data);
     }
}
void pop() {
    if(isempty()==1){
        printf("\nThe popped element is %d \n",stack[top]);
        top=top-1;
      }
}
```

**Output:**

```
Enter the stack size
4
The stack is empty
The peek of stack is -1
Enter the element to be inserted 5
The element 5 is inserted
Enter 1 to delete an element from stack
 1

The popped element is 5
```

**8.Queue Operation**

**Program:**

```
#include<stdio.h>
#include <stdlib.h>
#include <string.h>
int cqueue[100], maxsize,front=0, rear=0;
void enqueue();
void dequeue();
void peek();
void main(){
        printf("Enter the circular queue size\n");
        scanf("%d",&maxsize);
        int choice;
        int element,i;
        do{
          printf("1.Enter 1 to see rear and front value of queue\n");
          printf("2.Enter 2 to insert an element in the  queue\n");
          printf("3.Enter 3 to delete an element from the queue\n");
          scanf("%d",&i);
          switch(i){
            case 1:peek();
```

```
                    break;
                case 2:printf("\nEnter the element to be inserted\n");
                    scanf("%d",&element);
                    enqueue(element);
                    break;
                case 3:dequeue();
                    break;
                default:printf("\nEntered the wrong choice\n");
                }
                printf("\nDo you want to continue(1/2)\n");
                scanf("%d",&choice);
            }while(choice==1);
}
void enqueue(int data){
            if(rear==maxsize-1){
            printf("Sorry,The circular queue is full cannot insert\n");
            exit(0);
    }
    else{
             cqueue[rear]=data;
             rear=rear+1;
          }
}
void dequeue(){
            if(front==-1){
                    printf("The circular queue is empty\n");
                    exit(0);
             }
            else{
```

```
                printf("The deleted element is %d \n",cqueue[front]);
                  front=front+1;
            }
}
void peek(){
    printf("The front value is %d \n",front);
    printf("The rear value is %d \n",rear);
}
```

**Output:**

```
sjcet@Z238-UL:~/kishor/data structure$ gcc queue.c
sjcet@Z238-UL:~/kishor/data structure$ ./a.out
Enter the circular queue size
4
1.Enter 1 to see rear and front value of  queue
2.Enter 2 to insert an element in the  queue
3.Enter 3 to delete an element from the  queue
2

Enter the element to be inserted
15

Do you want to continue(1/2)
1
1.Enter 1 to see rear and front value of  queue
2.Enter 2 to insert an element in the  queue
3.Enter 3 to delete an element from the  queue
2

Enter the element to be inserted
20

Do you want to continue(1/2)
1
1.Enter 1 to see rear and front value of  queue
2.Enter 2 to insert an element in the  queue
3.Enter 3 to delete an element from the  queue
1
The front value is 0
The rear value is 2

Do you want to continue(1/2)
1
1.Enter 1 to see rear and front value of  queue
2.Enter 2 to insert an element in the  queue
3.Enter 3 to delete an element from the  queue
3
The deleted element is 15

Do you want to continue(1/2)
```

**9.Circular Queue Operation**

**Program:**

```
#include<stdio.h>
#include <stdlib.h>
#include <string.h>
int cqueue[100], maxsize,front=0, rear=0;
void enqueue();
void dequeue();
void peek();
void main(){
        printf("Enter the circular queue size\n");
        scanf("%d",&maxsize);
        int choice;
        int element,i;
        do{
          printf("1.Enter 1 to see rear and front value of circular queue\n");
          printf("2.Enter 2 to insert an element in the circular queue\n");
          printf("3.Enter 3 to delete an element from the circular queue\n");
          scanf("%d",&i);
          switch(i){
            case 1:peek();
                  break;
            case 2:printf("\nEnter the element to be inserted\n");
                  scanf("%d",&element);
                  enqueue(element);
                  break;
            case 3:dequeue();
                  break;
```

```
                default:printf("\nEntered the wrong choice\n");
            }
            printf("\nDo you want to continue(1/2)\n");
            scanf("%d",&choice);
        }while(choice==1);
}
void enqueue(int data){
        if((front==-1 && rear==maxsize-1)||(rear==front-1)){
                printf("Sorry,The circular queue is full cannot insert\n");
                exit(0);
        }
        else if(rear==maxsize-1 && front!=0){
                rear = 0;
                cqueue[rear]=data;
        }
        else{
                cqueue[rear]=data;
                rear=rear+1;
            }
}
void dequeue(){
        if(front==-1 || rear==front-1){
                printf("The circular queue is empty\n");
                exit(0);
        }
        else{
                printf("The deleted element is %d \n",cqueue[front]);
                front=front+1;
        }
```

```
}
void peek(){
        printf("The front value is %d \n",front);
        printf("The rear value is %d \n",rear);
}
```

**Output:**

```
Enter the circular queue size
4
1.Enter 1 to see rear and front value of circular queue
2.Enter 2 to insert an element in the circular queue
3.Enter 3 to delete an element from the circular queue
2

Enter the element to be inserted
12

Do you want to continue(1/2)
1
1.Enter 1 to see rear and front value of circular queue
2.Enter 2 to insert an element in the circular queue
3.Enter 3 to delete an element from the circular queue
2

Enter the element to be inserted
2

Do you want to continue(1/2)
1
1.Enter 1 to see rear and front value of circular queue
2.Enter 2 to insert an element in the circular queue
3.Enter 3 to delete an element from the circular queue
1
The front value is 0
The rear value is 2

Do you want to continue(1/2)
1
1.Enter 1 to see rear and front value of circular queue
2.Enter 2 to insert an element in the circular queue
3.Enter 3 to delete an element from the circular queue
3
The deleted element is 12

Do you want to continue(1/2)
2
```

**10.Structure Implementation**

**Program:**

```
#include<stdio.h>
#include<string.h>
struct student {
        int rollno;
        char name[20];
        char course[5];
};
struct college {
        char name1[7];
        struct student s1;
};
void main(){
        struct college c1;
        c1.s1.rollno=35;
        strcpy(c1.s1.name,"kishor vinod");
        strcpy(c1.s1.course,"mca");
        strcpy(c1.name1,"sjcet");
        printf("College Name : %s\n",c1.name1);
        printf("Course Name  : %s\n",c1.s1.course);
        printf("Student Name : %s\n",c1.s1.name);
        printf("roll no      : %d\n",c1.s1.rollno);
}
```

**Output:**

```
College Name : sjcet
Course Name  : mca
Student Name : kishor vinod
roll no      : 35
```

**11.Linked list implementation using Stack**

**Program:**

```
#include<stdio.h>
#include<stdlib.h>
void push();
void pop();
void display();
struct stacknode {
        int data;
        struct stacknode *next;
}*top=NULL;
void main() {
        int choice;
        do {
                printf("\n\n SELECT AN OPTION FROM THE BELOW MENU\n");
                printf("\n 1. PUSH OPERATION\n");
                printf("\n 2. POP OPERATION\n");
                printf("\n 3. VIEW LINKED STACK\n");
                printf("\n 4. EXIT\n\n");
                scanf("%d",&choice);
                switch(choice) {
                        case 1: push();
                                break;
                        case 2: printf("\n Poping Out the last item :\n");
                                pop();
                                break;
                        case 3: printf("\n The stack consists of items \n\n");
                                display();
```

```
                        break;
                case 4: exit(0);
                default: printf("\n Invalid Option\n");
            }
        }while(choice!=4);
}
void push() {
        struct stacknode *newnode;
        newnode=(struct stacknode*)malloc(sizeof(struct stacknode));
        printf("\n Enter a value :\n\n");
        scanf("%d",&newnode->data);
        if(top==NULL) {
                newnode->next=NULL;
                top=newnode;
        }
        else {
                newnode->next=top;
                top=newnode;
        }
        printf("\n Now, the items in the stack are\n\n");
        display();
}
void pop() {
        if(top==NULL)
                printf("\n Stack Underflow, Insert atleast one item\n");
        else{
                struct stacknode *temp=top;
                top=temp->next;
                printf("\n the item %d has been poped out...\n",temp->data);
```

```
                free(temp);
                printf("\n Now , the items in the stack are\n");
                display();
        }
}
void display(){
        struct stacknode *temp=top;
        if(top==NULL)
                printf("\n STACK IS EMPTY!\n");
        else{
                while(temp!=NULL){
                        printf("%d\t",temp->data);
                        temp=temp->next;
                }
        }

}
```

**Output:**

```
SELECT AN OPTION FROM THE BELOW MENU
1. PUSH OPERATION
2. POP OPERATION
3. VIEW LINKED STACK
4. EXIT
1
Enter a value :
12
Now, the items in the stack are
12
SELECT AN OPTION FROM THE BELOW MENU
1. PUSH OPERATION
2. POP OPERATION
3. VIEW LINKED STACK
4. EXIT
1
Enter a value :
14
Now, the items in the stack are
14      12
```

```
 SELECT AN OPTION FROM THE BELOW MENU

 1. PUSH OPERATION

 2. POP OPERATION

 3. VIEW LINKED STACK

 4. EXIT

3

 The stack consists of items

14      12

 SELECT AN OPTION FROM THE BELOW MENU

 1. PUSH OPERATION

 2. POP OPERATION

 3. VIEW LINKED STACK

 4. EXIT

2

 Poping Out the last item :

 the item 14 has been poped out...

 Now , the items in the stack are
12

 SELECT AN OPTION FROM THE BELOW MENU

 1. PUSH OPERATION

 2. POP OPERATION

 3. VIEW LINKED STACK

 4. EXIT
```

**12.Doubly Linked List**

**Program:**

```
#include<stdio.h>
#include<stdlib.h>
int count=0;
void insert_begin();
void insert_end();
void insert_pos();
void delete_begin();
void delete_end();
void delete_pos();
void search_key();
void traverse_list();
struct node{
        int data;
        struct node *prev;
        struct node *next;
}*head=NULL;
void main(){
        int opt,item;
        do{
                printf("\n SELECT A VALID OPTION FROM THE MENU\n");
                printf("\n1. INSERTION AT BEGINNING\n");
                printf("\n2. INSERTION AT END\n");
                printf("\n3. INSERTION AT A GIVEN POSITION\n");
                printf("\n4. DELETION AT BEGINNING\n");
                printf("\n5. DELETION AT END\n");
                printf("\n6. DELETION AT A PARTICULAR POSITION\n");
```

```
                printf("\n7. SEARCH FOR AN ITEM\n");
                printf("\n8. DISPLAY LIST\n");
                printf("\n9. EXIT\n");
                scanf("%d",&opt);
                switch(opt) {
                    case 1: insert_begin();
                            break;
                    case 2: insert_end();
                             break;
                    case 3: insert_pos();
                            break;
                    case 4: delete_begin();
                            break;
                    case 5: delete_end();
                             break;
                    case 6: delete_pos();
                             break;
                    case 7: search_key();
                             break;
                    case 8: traverse_list();
                             break;
                    case 9: exit(0);
                    default: printf("\n Invalid Option\n");
                   }
        } while(opt!=9);
}
void insert_begin(){
        int item;
        printf("\n enter a value: ");
```

```
        scanf("%d",&item);
        struct node *newnode;
        newnode=(struct node*)malloc(sizeof(struct node));
        newnode->data=item;
        if(head==NULL) {
                head=newnode;
                newnode->prev=NULL;
                newnode->next=NULL;
                count++;
        }
        else{
                struct node *temp=head;
                temp->prev=newnode;
                newnode->prev=NULL;
                newnode->next=temp;
                head=newnode;
                count++;
        }
       printf("\n the items in the list are:\n");
       traverse_list();
}
void insert_end(){
        int item;
        printf("\n enter a value: ");
        scanf("%d",&item);
        struct node *newnode;
        newnode=(struct node*)malloc(sizeof(struct node));
        newnode->data=item;
        if(head==NULL) {
```

```
                head=newnode;
                newnode->prev=NULL;
                newnode->next=NULL;
                count++;
            }
            else {
                struct node *temp=head;
                while(temp->next!=NULL)
                temp=temp->next;
                temp->next=newnode;
                newnode->prev=temp;
                newnode->next=NULL;
                count++;
        }
        printf("\n the items in the list are\n");
        traverse_list();
}
void insert_pos() {
        int item,pos,i=1;
        struct node *temp=head;
        printf("\n enter a value: ");
        scanf("%d",&item);
        struct node *newnode;
        newnode=(struct node*)malloc(sizeof(struct node));
        newnode->data=item;
        printf("\n Enter the position to which the new node is to be inserted: ");
        scanf("%d",&pos);
        if(pos>count){
                printf("\n invalid position\n");
```

```
        }
        while(temp->next!=NULL&&i!=pos-1){
                temp=temp->next;
                i++;
        }
        if(i==pos-1){
                newnode->next=temp->next;
                temp->next=newnode;
                newnode->prev=temp;
                count++;
        }
        else{
                if(pos==count){
                        while(temp->next!=NULL)
                        temp=temp->next;
                        temp->next=newnode;
                        newnode->next=NULL;
                        newnode->prev=temp;
                        count++;
                }
                else
                        printf("\n POSITION not found in list\n");
            }
        printf("\n the items in the list are\n");
        traverse_list();
}
void delete_begin(){
        struct node *temp=head;
        if(head==NULL)
```

```
                printf("\n doubly linked list is empty\n");
        else{
                if(temp->next==NULL){
                        temp->prev=NULL;
                        head=NULL;
                        printf("\n the item %d has been deleted\n",temp->data);
                        free(temp);
                        count--;
                        traverse_list();
                }
                else{
                        head=temp->next;
                        temp->next->prev=NULL;
                        temp->prev=NULL;
                        temp->next=NULL;
                        printf("\n the item %d has been deleted from beginning\n",temp-
                        >data);
                        free(temp);
                        count--;
                        printf("\n the items in the list are\n");
                        traverse_list();
                }
        }
}
void delete_end(){
        struct node *temp=head;
        if(head==NULL)
                printf("\n doubly linked list is empty\n");
        else if(temp->next==NULL){
                printf("\n the item %d has been deleted\n",temp->data);
```

```
                temp->prev=NULL;
                temp->next=NULL;
                head=NULL;
                free(temp);
                count--;
                printf("\n the items in the list are\n");
                traverse_list();
        }
        else{
                while(temp->next!=NULL)
                temp=temp->next;
                temp->prev->next=NULL;
                temp->prev=NULL;
                printf("\n the item %d has been deleted from end\n",temp->data);
                free(temp);
                count--;
                printf("\n the items in the list are\n");
                traverse_list();
        }
}
void delete_pos(){
        int pos,i=1;
        struct node *temp=head;
        if(head==NULL)
                printf("\n the doubly linked list is empty\n");
        else{
                printf("\n enter the position of node to be deleted: ");
                scanf("%d",&pos);
                if(pos>count)
```

```
                printf("\n position is not within the list\n");
        else{
                while(temp->next!=NULL&&pos!=i){
                        temp=temp->next;
                        i++;
                }
                temp->prev->next=temp->next;
                temp->prev=NULL;
                temp->next=NULL;
                printf("\n the item %d has been deleted",temp->data);
                free(temp);
                count--;
                printf("\n the items in the doubly linked list are\n");
                traverse_list();
        }
    }
}
void traverse_list(){
    struct node *temp=head;
    if(head==NULL)
        printf("\n list is empty\n");
    else{
    while(temp!=NULL){
        printf("%d\t",temp->data);
        temp=temp->next;
    }
    }
}
void search_key(){
```

```
        int item;
        printf("\n enter an item to be searched: \n");
        scanf("%d",&item);
        struct node *temp=head;
        while(temp->data!=item&&temp->next!=NULL)
        temp=temp->next;
        if(temp->data==item)
                printf("\n the item %d found in the list",item);
        else
                printf("\n the item %d not found in the list\n",item);
}
```

Output:

```
 SELECT A VALID OPTION FROM THE MENU
1. INSERTION AT BEGINNING
2. INSERTION AT END
3. INSERTION AT A GIVEN POSITION
4. DELETION AT BEGINNING
5. DELETION AT END
6. DELETION AT A PARTICULAR POSITION
7. SEARCH FOR AN ITEM
8. DISPLAY LIST
9. EXIT
1

 enter a value: 12

 the items in the list are:
12
```

```
 SELECT A VALID OPTION FROM THE MENU

1. INSERTION AT BEGINNING

2. INSERTION AT END

3. INSERTION AT A GIVEN POSITION

4. DELETION AT BEGINNING

5. DELETION AT END

6. DELETION AT A PARTICULAR POSITION

7. SEARCH FOR AN ITEM

8. DISPLAY LIST

9. EXIT
8
12
 SELECT A VALID OPTION FROM THE MENU

1. INSERTION AT BEGINNING

2. INSERTION AT END

3. INSERTION AT A GIVEN POSITION

4. DELETION AT BEGINNING

5. DELETION AT END

6. DELETION AT A PARTICULAR POSITION

7. SEARCH FOR AN ITEM

8. DISPLAY LIST

9. EXIT
9
```

## 13.Binary Search Tree Implementation

**Program:**

```
#include <stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *l;
        struct node *r;
}*root=NULL,*temp=NULL,*t1,*t2;
```

```
void insert();
void create();
void search(struct node *t);
void search1(struct node *t,int data);
void inorder(struct node *t);
void delete();
void delete1();
int smallest(struct node *t);
int largest(struct node *t);
int flag = 1;
int main(){
        int ch;
        printf("\nOPERATIONS ---");
        printf("\n1 - Insert an element into tree\n");
        printf("2 - Traversal\n");
        printf("3 - Delete a node \n");
        printf("4 - Exit\n");
        do {
          printf("\nEnter your choice : ");
          scanf("%d", &ch);
          switch (ch){
                  case 1: insert(); break;
                  case 2:inorder(root);break;
                  case 3:delete();break;
                  case 4:printf("\nInvalid option\n");
                          exit(0);
                 default :
                      printf("Wrong choice, Please enter correct choice ");
                      break;
```

```
            }
        }while(ch<4);
}
void insert(){
        create();
        if (root == NULL)
        root = temp;
        else
        search(root);
}
void create(){
        int data;
        printf("Enter data of node to be inserted : ");
        scanf("%d", &data);
        temp = (struct node *)malloc(1*sizeof(struct node));
        temp->data = data;
        temp->l = temp->r = NULL;
}
void search(struct node *t){
        if ((temp->data > t->data) && (t->r != NULL))
                search(t->r);
        else if ((temp->data> t->data) && (t->r == NULL))
                t->r = temp;
        else if ((temp->data< t->data) && (t->l != NULL))
                search(t->l);
        else if ((temp->data < t->data) && (t->l == NULL))
                t->l = temp;
}
void inorder(struct node *t){
```

```
        if (root == NULL){
                printf("No elements in a tree to display");
                return;
        }
        if (t->l != NULL)
                inorder(t->l);
                printf("%d ->", t->data);
        if (t->r != NULL)
                inorder(t->r);
}
void delete(){
        int data;
        if (root == NULL){
                printf("No elements in a tree to delete");
                return;
        }
        printf("Enter the data to be deleted : ");
        scanf("%d", &data);
        t1 = root;
        t2 = root;
        search1(root, data);
}
void search1(struct node *t, int data){
        if ((data>t->data)){
                t1 = t;
                search1(t->r, data);
        }
        else if ((data < t->data)){
                t1 = t;
```

```
                search1(t->l, data);
        }

        else if ((data==t->data)){
                delete1(t);
        }
}
void delete1(struct node *t){
        int k;
        if ((t->l == NULL) && (t->r == NULL)){
                if (t1->l == t){
                        t1->l = NULL;
                }
                else{
                        t1->r = NULL;
                }
                t = NULL;
                free(t);
                return;
        }
        else if ((t->r == NULL)){
        if (t1 == t){
                root = t->l;
                t1 = root;
        }
        else if (t1->l == t){
                t1->l = t->l;
        }
        else{
```

```
            t1->r = t->l;
          }
            t = NULL;
            free(t);
            return;
    }
    else if (t->l == NULL){
            if (t1 == t){
                    root = t->r;
                    t1 = root;
             }
            else if (t1->r == t)
                    t1->r = t->r;
            else
                    t1->l = t->r;
                    t == NULL;
                    free(t);
                    return;
    }
    else if ((t->l != NULL) && (t->r != NULL)){
            t2 = root;
            if (t->r != NULL){
            k = smallest(t->r);
            flag = 1;
    }
    else{
            k =largest(t->l);
            flag = 2;
    }
```

```
        search1(root, k);
        t->data = k;
        }
}
int smallest(struct node *t) {
        t2 = t;
        if (t->l != NULL){
                t2 = t;
                return(smallest(t->l));
        }
        else
                return (t->data);
        }
int largest(struct node *t){
        if (t->r != NULL){
                t2 = t;
                return(largest(t->r));
        }
        else
                return(t->data);
}
```

Output:

```
OPERATIONS ---
1 - Insert an element into tree
2 - Traversal
3 - Delete a node
4 - Exit

Enter your choice : 1
Enter data of node to be inserted : 12

Enter your choice : 1
Enter data of node to be inserted : 24

Enter your choice : 1
Enter data of node to be inserted : 6

Enter your choice : 2
6 ->12 ->24 ->
Enter your choice : 3
Enter the data to be deleted : 12

Enter your choice : 2
6 ->24 ->
Enter your choice : 4
```

**14.Balanced Binary Search Tree**

**Program:**

```
#include <stdio.h>
#include <stdlib.h>
struct node {
        int item;
        struct node *left;
        struct node *right;
};
struct node *newNode(int item) {
        struct node *node = (struct node *)malloc(sizeof(struct node));
        node->item = item;
        node->left = NULL;
        node->right = NULL;
```

```
        return (node);
}
int checkHeightBalance(struct node *root, int *height) {
        int leftHeight = 0, rightHeight = 0;
        int l = 0, r = 0;
        if (root == NULL) {
                *height = 0;
                 return 1;
        }
        l = checkHeightBalance(root->left, &leftHeight);
        r = checkHeightBalance(root->right, &rightHeight);
       *height = (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;
        if ((leftHeight - rightHeight >= 2) || (rightHeight - leftHeight >= 2))
                return 0;
        else
            return l && r;
}
int main() {
        int height = 0;
       struct node *root = newNode(1);
        root->left = newNode(2);
        root->right = newNode(3);
        root->left->left = newNode(4);
        root->left->right = newNode(5);
        if (checkHeightBalance(root, &height))
                printf("The tree is balanced\n");
         else
                printf("The tree is not balanced\n");
```

```
}
```

**Output:**

```
sjcet@Z238-UL:~/kishor/data structure$ gcc bbst.c
sjcet@Z238-UL:~/kishor/data structure$ ./a.out
The tree is balanced
```

**15.Set Implementation**

**Program:**

```
#include<stdio.h>
void main(){
    int a[10],b[10],i,c[10],d[10],e[10],f[10],j,k=0,n1,l,n2,ch,m=0,n=0,p=0;
    printf("Enter number of element of set A\n");
    scanf("%d",&n1);
    printf("Enter the element of set A \n");
    for(i=0;i<n1;i++)
        scanf("%d",&a[i]);
    printf("Enter number of element of set B\n");
    scanf("%d",&n2);
    printf("Enter the element of set B \n");
    for(i=0;i<n2;i++)
        scanf("%d",&b[i]);
    while(ch!=4){
        printf("\n\nSelect your choice\n");
        printf("\n1.Union of the 2 Sets \n2.Intersection Of The2Sets\n3.Differenence
        between The Sets End\n4.Exit\n");
        printf("\nEnter your choice\n");
        scanf("\n%d",&ch);
        switch(ch){
        case 1: for(i=0;i<n1;i++){
                    for(j=0;j<k;j++){
                        if(c[j]==a[i])
```

```
                        break;
                    }
                    if(j==k){
                        c[k]=a[i];
                        k++;
                    }
                }
        for(i=0;i<n2;i++){
            for(j=0;j<k;j++){
                if(c[j]==b[i])
                    break;
                }
                if(j==k){
                    c[k]=b[i];
                    k++;
                }
            }
            printf("Union of set A and B is:-\n");
            for(i=0;i<k;i++)
            printf("%d ",c[i]);
            break;
        case 2:
            n=0;
            printf("INTERSECTION \n");
            for( i=0;i<n1;i++){
                for(j=0;j<n2;j++){
                    if(a[i]==b[j]){
                        d[n]=a[i];
                        n++;
```

```
                }
            }
        }
        printf("intersection of set A and set B are:-\n");
        for(i=0;i<n;i++)
        printf("%d ",d[i]);
        break;
case 3:
        m=0;
        p=0;
        for( i=0;i<n1;i++){
        for(j=0;j<n2;j++){
                if(b[j]==a[i])
                        break;
                }
                if(j==n2){
                        for(l=0;l<m;l++){
                                if(e[l]==a[i])
                                        break;
                        }
                if(l==m){
                                e[m]=a[i];
                                m++;
                        }
                }
        }
        for( i=0;i<n2;i++){
                for(j=0;j<n1;j++){
                        if(b[i]==a[j])
```

```
                        break;
                }
            if(j==n1){
                for(l=0;l<p;l++){
                    if(d[l]==b[i])
                        break;
            }
            if(l==p){
                d[p]=b[i];
                p++;
                }
              }
            }
            printf("Difference of A-B is:-\n");
            for(i=0;i<m;i++){
                printf("%d ",e[i]);
                }
            printf("\n");
            printf("Difference of B-A is:-\n");
            for(i=0;i<p;i++){
                printf("%d ",d[i]);
                }
            }
        }
}
```

Output:

```
Enter number of element of set A
3
Enter the element of set A
1
2
3
Enter number of element of set B
2
Enter the element of set B
3
4


Select your choice

1.Union of the 2 Sets
2.Intersection Of The2Sets
3.Differenence between The Sets End
4.Exit

Enter your choice
1
Union of set A and B is:-
1 2 3 4

Select your choice

1.Union of the 2 Sets
2.Intersection Of The2Sets
3.Differenence between The Sets End
4.Exit

Enter your choice
2
INTERSECTION
intersection of set A and set B are:-
3
```

```
Select your choice

1.Union of the 2 Sets
2.Intersection Of The2Sets
3.Differenence between The Sets End
4.Exit

Enter your choice
4
```

**16.Disjoint Set Implementation**

**Program:**

```
#include<stdio.h>
struct disjointSet {
        int parent[10];
        int rank[10];
        int n;
}dis;
void makeset(){
         int i;
         for(i=0;i<dis.n;i++)
                  dis.parent[i]=i;
         dis.rank[i]=0;
}
void displayset(){
        int i;
        printf("\nparent array\n");
        for(i=0;i<dis.n;i++){
                  printf("%d",dis.parent[i]);
         }
        printf("\nrank of array\n");
        for(i=0;i<dis.n;i++) {
                printf("%d",dis.rank[i]);
         }
        printf("\n");
}
int find(int x){
         if(dis.parent[x]!=x){
                  dis.parent[x]=find(dis.parent[x]);
```

```
                }
                return dis.parent[x];
}
void Union(int x,int y){
        int xset=find(x) , yset=find(y);
        if(xset==yset)
                return;
        if(dis.rank[xset]<dis.rank[yset]){
                dis.parent[xset]=yset;
                dis.rank[xset]=-1;
        }
        else if(dis.rank[xset]<dis.rank[yset]){
                dis.parent[yset]=xset;
                dis.rank[yset]=-1;
        }
        else{
                dis.parent[yset]=xset;
                dis.rank[xset]=dis.rank[xset]+1;
                dis.rank[yset]=-1;
        }
}
int main(){
        int x,y,n;
        printf("\nenter number of elements :\n");
        scanf("%d",&dis.n);
        makeset();
        int ch,w;
        do{
                printf("\n1.UNION\n2.FIND \n3.DISPLAY");
```

```
            printf("\nenter choice :");
            scanf("%d",&ch);
            switch(ch){
                case 1:
                    printf("\nenter elements to perform union :");
                    scanf("%d %d",&x,&y);
                    Union(x,y);
                    break;
                case 2:
                    printf("\nenter elements to check if connected components :");
                    scanf("%d %d",&x,&y);
                    if(find(x)==find(y))
                        printf("\nconnected components !");
                    else
                        printf("\n no connected components !");
                        break;
                case 3:
                    displayset();
                    break;
            }
            printf("\n do you want to continue ?(1/0)");
            scanf("%d",&w);
        }while(w==1);
        return 0;
}
```

**Output:**

```
enter number of elements :
2

1.UNION
2.FIND
3.DISPLAY
enter choice :1

enter elements to perform union :12
13

 do you want to continue ?(1/0)1

1.UNION
2.FIND
3.DISPLAY
enter choice :2

enter elements to check if connected components :12
14

connected components !
 do you want to continue ?(1/0)
```

**17.Max Heap Implementation**

**Program:**

```
#include <stdio.h>

int size = 0;

void swap(int *a, int *b){
        int temp = *b;
        *b = *a;
        *a = temp;
}

void heapify(int array[], int size, int i){
        if (size == 1){
                printf("Single element in the heap");
        }
```

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

```
        for (i = 0; i < size; i++){
                if (num == array[i])
                        break;
                }
        swap(&array[i], &array[size - 1]);
        size -= 1;
        for (int i = size / 2 - 1; i >= 0; i--){
                heapify(array, size, i);
        }
}
void printArray(int array[], int size){
       for (int i = 0; i < size; ++i)
               printf("%d ", array[i]);
        printf("\n");
}
int main(){
        int array[10];
        insert(array, 3);
        insert(array, 4);
        insert(array, 9);
        insert(array, 5);
        insert(array, 2);
        printf("Max-Heap array: ");
        printArray(array, size);
        deleteRoot(array, 4);
        printf("After deleting an element: ");
        printArray(array, size);
}
```

**Output:**

```
sjcet@Z238-UL:~/kishor/data structure$ gcc maxheap.c
sjcet@Z238-UL:~/kishor/data structure$ ./a.out
Max-Heap array: 9 5 4 3 2
After deleting an element: 9 5 2 3
```

**18.Min Heap Implementation**

**Program:**

```
#include <stdio.h>
#define HEAP_CAPACITY 10
#define SUCCESS_VAL 99999
#define FAIL_VAL -99999
int size = 0;
int i;
int heap[HEAP_CAPACITY];
void swap(int *a,int *b){
        int temp = *b;
        *b = *a;
        *a = temp;
}
void heapify(int i){
        if (size == 1){
                return;
        }
        else{
                int smallest = i;
                int left = 2 * i + 1;
                int right = 2 * i + 2;
                if(left < size && heap[left] < heap[smallest])
                        smallest = left;
```

```
                if(right < size && heap[right] < heap[smallest])
                        smallest = right;
                if (smallest != i){
                        swap(&heap[i], &heap[smallest]);
                        heapify(smallest);
                }
        }
}
int insert(int newNum){
        if(size==0) {
                heap[0] = newNum;
                size += 1;
                return SUCCESS_VAL;
        }
        else if(size < HEAP_CAPACITY){
                heap[size] = newNum;
                size += 1;
                for(i =(size-1)/2;i>=0;i--) {
                   heapify(i);
                }
                return SUCCESS_VAL;
        }
        else{
                printf("Heap capacity reached. Insertion failed.\n");
                return FAIL_VAL;
        }
}
int delete(int number){
        int i,index=-1;
```

```
        if(size <=0){
                printf("Empty min heap");
                return FAIL_VAL;
        }
        for(i=0;i<size;i++) {
                if(number == heap[i]){
                        index = i;
                        break;
                }
        }
        if(index == -1) {
                printf("Key is not found\n");
                return FAIL_VAL;
        }
        swap(&heap[i],&heap[size-1]);
        size -= 1;
        for(i=(size-1)/2; i>=0;i--) {
                heapify(i);
        }
        return SUCCESS_VAL;
}
void printHeap(){
        for( i=0;i<size;++i){
           if(i==0)
              printf("%d(root) ", heap[i]);
           else
              printf("%d(%d's child) ",heap[i],heap[(i-1)/2]);
        }
        printf("\n");
```

```
}
int main(){
        while(1){
                printf("\n___MENU___\n1.Insert Element \n2.Print MinHeap \n3.Delete
                                                                Element \n4.Exit \n");
         int choice;
         scanf("%d",&choice);
         if(choice==1) {
                printf("Enter the element to be inserted\n");
                int item;
                scanf("%d",&item);
                int res=insert(item);
                if(res==SUCCESS_VAL)
                   printf("inserted successfully\n");
         }
         else if(choice==2) {
                printHeap();
         }
         else if(choice==3)  {
                int res = delete(heap[0]);
                if(res==SUCCESS_VAL)
                        printf("Delete Successfully\n");
                else
                        printf("Deleted Unsuccessfully\n");
           }
         else if(choice==4){
                break;
         }
       }
}
```

**Output:**

```
___MENU___
1.Insert Element
2.Print MinHeap
3.Delete Element
4.Exit
1
Enter the element to be inserted
12
inserted successfully

___MENU___
1.Insert Element
2.Print MinHeap
3.Delete Element
4.Exit
1
Enter the element to be inserted
13
inserted successfully

___MENU___
1.Insert Element
2.Print MinHeap
3.Delete Element
4.Exit
2
12(root) 13(12's child)

___MENU___
1.Insert Element
2.Print MinHeap
3.Delete Element
4.Exit
4
```

**19.btree implementation**

**Program:**

```
#include<stdio.h>

#include<stdlib.h>

#define MAX 3

#define MIN 2
```

```
struct BTreeNode {
        int val[MAX + 1], count;
        struct BTreeNode *link[MAX + 1];
};
struct BTreeNode *root;
struct BTreeNode *createNode(int val, struct BTreeNode *child){
        struct BTreeNode *newNode;
        newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
        newNode->val[1] = val;
        newNode->count = 1;
        newNode->link[0] = root;
        newNode->link[1] = child;
        return newNode;
}
void insertNode(int val, int pos, struct BTreeNode *node, struct BTreeNode *child){
        int j = node->count;
        while(j > pos)  {
                node->val[j+1] = node->val[j];
                node->link[j+1] = node-> link[j];
                j--;
        }
        node->val[j+1] = val;
        node->link[j+1] = child;
        node->count++;
}
void splitNode(int val, int *pval, int pos, struct BTreeNode *node, struct BTreeNode *child,
struct BTreeNode **newNode){
        int median, j;
        if(pos > MIN)
                median = MIN + 1;
```

```
        else
                median = MIN;
        *newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
        j = median + 1;
        while(j <= MAX) {
                (*newNode)->val[j-median] = node->val[j];
                (*newNode)->link[j-median] = node->link[j];
                j++;
        }
        node->count = median;
        (*newNode)->count = MAX - median;
        if(pos <= MIN) {
                insertNode(val, pos, node, child);
        }
        else{
                insertNode(val, pos-median, *newNode, child);
        }
        *pval = node->val[node->count];
        (*newNode)->link[0] = node->link[node->count];
        node->count--;
}
int setValue(int val, int *pval, struct BTreeNode *node, struct BTreeNode **child){
        int pos;
        if (!node) {
                *pval = val;
                *child = NULL;
                return 1;
        }
```

```
        if (val < node->val[1]){
                pos = 0;
        }
        else {
          for (pos = node->count;(val < node->val[pos] && pos > 1); pos--);
                  if (val == node->val[pos])
                        {
                                printf("Duplicates are not permitted\n");
                                return 0;
                        }
        }
        if (setValue(val, pval, node->link[pos], child)) {
                if (node->count < MAX){
                                insertNode(*pval, pos, node, *child);
                }
        else{
                  splitNode(*pval, pval, pos, node, *child, child);
                  return 1;
                }
        }
        return 0;
}
void insert(int val){
        int flag, i;
        struct BTreeNode *child;
       flag = setValue(val, &i, root, &child);
        if (flag)
                root = createNode(i, child);
}
```

```
void search(int val, int *pos, struct BTreeNode *myNode){
        if (!myNode){
                return;
        }
        if (val < myNode->val[1]){
                *pos = 0;
        }
        else{
                for (*pos = myNode->count; (val < myNode->val[*pos] && *pos > 1);
                (*pos)--){
                        if (val == myNode->val[*pos]) {
                                printf("%d is found \n", val);
                                return;
                        }
                }
     search(val, pos, myNode->link[*pos]);
      return;
}
void traversal(struct BTreeNode *myNode){
        int i;
        if (myNode){
                for (i = 0; i < myNode->count; i++) {
                        traversal(myNode->link[i]);
                        printf("%d ", myNode->val[i + 1]);
                }
                traversal(myNode->link[i]);
        }
}
```

```
int main(){
        int val, ch;
        insert(8);
        insert(9);
        insert(10);
        insert(11);
        insert(15);
        insert(16);
        insert(17);
        insert(18);
        insert(20);
        insert(23);
        traversal(root);
        printf("\n");
        search(11, &ch, root);
}
```

**Output:**

```
sjcet@Z238-UL:~/kishor/data structure$ gcc btree.c
sjcet@Z238-UL:~/kishor/data structure$ ./a.out
8 9 10 11 15 16 17 18 20 23
11 is found
```

**20.Red Black Tree Implementation**

**Program:**

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

```
        int data, color;
        struct rbNode *link[2];
};
struct rbNode *root = NULL;
struct rbNode *createNode(int data) {
        struct rbNode *newnode;
        newnode = (struct rbNode *)malloc(sizeof(struct rbNode));
        newnode->data = data;
        newnode->color = RED;
        newnode->link[0] = newnode->link[1] = NULL;
        return newnode;
}
void insertion(int data) {
        struct rbNode *stack[98], *ptr, *newnode, *xPtr, *yPtr;
        int dir[98], ht = 0, index;
        ptr = root;
        if (!root) {
                root = createNode(data);
                return;
        }
        stack[ht] = root;
        dir[ht++] = 0;
        while (ptr != NULL){
                if (ptr->data == data) {
                        printf("Duplicates Not Allowed!!\n");
                        return;
                }
                index = (data - ptr->data) > 0 ? 1 : 0;
                stack[ht] = ptr;
```

```
            ptr = ptr->link[index];
            dir[ht++] = index;
        }
        stack[ht - 1]->link[index] = newnode = createNode(data);
        while ((ht >= 3) && (stack[ht - 1]->color == RED)) {
            if (dir[ht - 2] == 0) {
                yPtr = stack[ht - 2]->link[1];
            if (yPtr != NULL && yPtr->color == RED) {
                stack[ht - 2]->color = RED;
                stack[ht - 1]->color = yPtr->color = BLACK;
                ht = ht - 2;
            }
            else {
            if (dir[ht - 1] == 0) {
                yPtr = stack[ht - 1];
            }
             else {
            xPtr = stack[ht - 1];
            yPtr = xPtr->link[1];
            xPtr->link[1] = yPtr->link[0];
            yPtr->link[0] = xPtr;
            stack[ht - 2]->link[0] = yPtr;
            }
            xPtr = stack[ht - 2];
            xPtr->color = RED;
            yPtr->color = BLACK;
            xPtr->link[0] = yPtr->link[1];
            yPtr->link[1] = xPtr;
            if (xPtr == root)
```

```
{
root = yPtr;
} else {
stack[ht - 3]->link[dir[ht - 3]] = yPtr;
}
break;
}
} else
{
yPtr = stack[ht - 2]->link[0];
if ((yPtr != NULL) && (yPtr->color == RED))
{
stack[ht - 2]->color = RED;
stack[ht - 1]->color = yPtr->color = BLACK;
ht = ht - 2;
} else
{
if (dir[ht - 1] == 1)
{
yPtr = stack[ht - 1];
} else
{
xPtr = stack[ht - 1];
yPtr = xPtr->link[0];
xPtr->link[0] = yPtr->link[1];
yPtr->link[1] = xPtr;
stack[ht - 2]->link[1] = yPtr;
}
xPtr = stack[ht - 2];
```

```
                yPtr->color = BLACK;
                xPtr->color = RED;
                xPtr->link[1] = yPtr->link[0];
                yPtr->link[0] = xPtr;
                if (xPtr == root)
                {
                root = yPtr;
                } else
                {
                stack[ht - 3]->link[dir[ht - 3]] = yPtr;
                }
                break;
                }
                }
                }
                root->color = BLACK;
}
void deletion(int data) {
        struct rbNode *stack[98], *ptr, *xPtr, *yPtr;
        struct rbNode *pPtr, *qPtr, *rPtr;
        int dir[98], ht = 0, diff, i;
        enum nodeColor color;
        if (!root)
        {
        printf("Tree not available\n");
        return;
        }
        ptr = root;
        while (ptr != NULL)
```

```
{
if ((data - ptr->data) == 0)
break;
diff = (data - ptr->data) > 0 ? 1 : 0;
stack[ht] = ptr;
dir[ht++] = diff;
ptr = ptr->link[diff];
}
if (ptr->link[1] == NULL)
{
if ((ptr == root) && (ptr->link[0] == NULL))
{
free(ptr);
root = NULL;
} else if (ptr == root)
{
root = ptr->link[0];
free(ptr);
} else
{
stack[ht - 1]->link[dir[ht - 1]] = ptr->link[0];
}
} else
{
xPtr = ptr->link[1];
if (xPtr->link[0] == NULL)
{
xPtr->link[0] = ptr->link[0];
color = xPtr->color;
```

```
xPtr->color = ptr->color;
ptr->color = color;
if (ptr == root)
{
root = xPtr;
} else
{
stack[ht - 1]->link[dir[ht - 1]] = xPtr;
}
dir[ht] = 1;
stack[ht++] = xPtr;
} else
{
i = ht++;
while (1)
{
dir[ht] = 0;
stack[ht++] = xPtr;
yPtr = xPtr->link[0];
if (!yPtr->link[0])
break;
xPtr = yPtr;
}
dir[i] = 1;
stack[i] = yPtr;
if (i > 0)
stack[i - 1]->link[dir[i - 1]] = yPtr;
yPtr->link[0] = ptr->link[0];
xPtr->link[0] = yPtr->link[1];
```

```
yPtr->link[1] = ptr->link[1];
if (ptr == root)
{
root = yPtr;
}
color = yPtr->color;
yPtr->color = ptr->color;
ptr->color = color;
}
}
if (ht < 1)
return;
if (ptr->color == BLACK)
{
while (1)
{
pPtr = stack[ht - 1]->link[dir[ht - 1]];
if (pPtr && pPtr->color == RED)
{
pPtr->color = BLACK;
break;
}
if (ht < 2)
break;
if (dir[ht - 2] == 0) {
rPtr = stack[ht - 1]->link[1];
if (!rPtr)
break;
if (rPtr->color == RED)
```

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

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

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

```
        {
        stack[ht - 2]->link[dir[ht - 2]] = rPtr;
        }
        break;
        }
        }
        ht--;
        }
        }
}
void inorderTraversal(struct rbNode *node) {
        if (node) {
        inorderTraversal(node->link[0]);
        printf("%d  ", node->data);
        inorderTraversal(node->link[1]);
        }
        return;
}
int main() {
        int ch, data;
        while (1) {
                printf("1. Insertion\t2. Deletion\n");
                printf("3. Traverse\t4. Exit");
                printf("\nEnter your choice:");
                scanf("%d", &ch);
                switch (ch) {
                case 1:
                        printf("Enter the element to insert:");
                        scanf("%d", &data);
```

```
                insertion(data);
                break;
            case 2:
                printf("Enter the element to delete:");
                scanf("%d", &data);
                deletion(data);
                break;
            case 3:
                inorderTraversal(root);
                printf("\n");
                break;
            case 4:
                exit(0);
            default:
                printf("Not available\n");
                break;
                }
                printf("\n");
                }
    return 0;
}
```

**Output:**

```
1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:1
Enter the element to insert:12

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:1
Enter the element to insert:13

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:1
Enter the element to insert:14

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:3
12  13  14

1. Insertion    2. Deletion
3. Traverse     4. Exit
Enter your choice:4
```

**21.Prims Algorithm**

**Program:**

```
#include<stdio.h>
#include <string.h>
#include<stdbool.h>
#define INF 9999999
#define V 5
int G[V][V] = {
        {0, 9, 75, 0, 0},
        {9, 0, 95, 19, 42},
        {75, 95, 0, 51, 66},
        {0, 19, 51, 0, 31},
        {0, 42, 66, 31, 0}};
```

```
int main(){
        int no_edge; // number of edge
        int selected[V];
        memset(selected, false, sizeof(selected));
        no_edge = 0;
        selected[0] = true;
        int x;
        int y;
        printf("Edge : Weight\n");
        while (no_edge < V - 1){
                int min = INF;
                x = 0;
                y = 0;
                for (int i = 0; i < V; i++){
                        if (selected[i]){
                                for (int j = 0; j < V; j++){
                                        if (!selected[j] && G[i][j]){
                                                if (min > G[i][j]) {
                                                        min = G[i][j];
                                                        x = i;
                                                        y = j;
                                                }
                                        }
                        }
                }
        }
        printf("%d - %d : %d\n", x, y, G[x][y]);
        selected[y] = true;
        no_edge++;
```

```
        }
        return 0;
}
```

**Output:**