/*Binary Search Tree Program */

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct bst
{
        struct bst *left;
        int key;
        struct bst *right;
}*root=NULL,*new1,*temp,*parent;
void create();
void display();
struct bst* find(int );
void findmax();
struct bst* findmin(struct bst *q);
void insert(struct bst *,struct bst *);
void inorder(struct bst*);
void delete1(struct bst *,int);
void create()
{
        int ele;
        new1=(struct bst*)malloc(sizeof(struct bst));
        new1->left=NULL;
        new1->right=NULL;
        printf("enter data ");
        scanf("%d",&ele);
        new1->key=ele;
        if(root==NULL)
        root=new1;
        else
        insert(root,new1);
}
void insert(struct bst *root,struct bst *new1)
{
        if(new1->key<root->key)
        {
        if(root->left==NULL)
        root->left=new1;
        else
```

```
        insert(root->left,new1);
        }
        else if(new1->key>root->key)
           {
                if(root->right==NULL)
                root->right=new1;
                else
                insert(root->right,new1);
           }
}
void display()
{
        if(root==NULL)
        printf("\n trre is not creayed");
        else
        inorder(root);
}
void inorder(struct bst *temp)
{
        if(temp!=NULL)
        {
        inorder(temp->left);
        printf(" %d",temp->key);
        inorder(temp->right);
        }
}
struct bst* find(int x)
{
        struct bst *par;
        int found=0;
        temp=root;
        if(temp==NULL)
        printf("\n element not found");
        else
        {
                while(temp!=NULL)
                {
                if(temp->key==x)
                {
                        found=1;
```

```
                        break;
                }

                else if(x<temp->key)
                {
                        parent=temp;
                        temp=temp->left;
                }
                else
                {
                        parent=temp;
                        temp=temp->right;
                }
                }
                if(found==1)
                {
                        printf("node present in bst with key=%d",temp->key);
                }
                else
                        printf("node is not present with key=%d",x);
                        return temp;

        }
}
void findmax()
{
        temp=root;
        while(temp->right!=NULL)
        {
                temp=temp->right;
        }
        printf("\n maxium element of bst is%d ",temp->key);
        getch();
}
struct bst * findmin(struct bst *p)
{
        struct bst *par;
        temp=p;
        while(temp->left!=NULL)
        {
```

```
                par=temp;
                temp=temp->left;
        }
        printf("\n minium element of bst is%d ",temp->key);
        getch();

        return par;

}
void delete1(struct bst *temp,int x)
{
        struct bst *p,*q;
        if(temp==NULL)
        printf("bst is empty \n");
         else  if(temp->left==NULL & temp->right==NULL)
           {
                free(temp);
                root=NULL;
            }

            while(temp!=NULL)
            {
                if(temp->key==x)
                break;

                else if(x<temp->key)
                    {
                    parent=temp;
                    temp=temp->left;
                    }
                    else
                    {
                        parent=temp;
                        temp=temp->right;
                    }

                }


                printf("temp =%d",temp->key);
```

```
            printf("parent =%d",parent->key);
            getch();
              /* deleting leaf node */

        if(temp->left==NULL && temp->right==NULL)
        {
                if(parent->left ==temp)
                parent->left=NULL;
                else
                parent->right=NULL;
                free(temp);
        }
        /*deleting non leaf node with one right child*/
        else if(temp->left==NULL && temp->right!=NULL)
        {
                if(temp->key>parent->key)
                parent->right=temp->right;
                else
                parent->left=temp->right;
                free(temp);
        }
        /* deleting non leaf node with one left child*/
        else if(temp->left!=NULL && temp->right==NULL)
        {
                if(temp->key<parent->key)
                parent->left=temp->left;
                else
                parent->right=temp->left;
                free(temp);
        }

/*deleting non leaf node with two children */

        else
        {
        if(temp->left!=NULL && temp->right!=NULL)
        {
        p=findmin(temp->right);
        q=p->left;
        temp->key=q->key;
```

```
        p->left=NULL;
        free(q);
        }

    }
}



void main()
{
        int choice,ele;
        while(1)
        {
        clrscr();
        printf("\t \t  binary search tree");
        printf("\n 1.create &  insert");
        printf("\n 2. display tree");
        printf("\n 3.find");
        printf("\n 4.find max");
        printf("\n 5 find min");
        printf("\n 6.delete  ");
        printf("\n 7.exit");
        printf("\nenter choic");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:create();
        break;
        case 2:display();getch();
        break;
        case 3:printf("\n enter ele to search");
        scanf("%d",&ele);
        temp=find(ele);

        getch();
        break;
        case 4:findmax();
        break;
        case 5:findmin(root);
```

```
        break;
        case 6: printf("\n enter ele to delete");
        scanf("%d",&ele);

        delete1(root,ele);
        break;
        case 7:exit(0);
        }
   }
}
```

/* Heap Sort using MaxHeapify */

```
#include<stdio.h>
#include<conio.h>
int a[10];
void Maxheapify(int,int);
void heapsort(int n);
int main()
{
        int n,i,j;
        clrscr();
        printf("enter the number of elemens]n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
                scanf("%d",&a[i]);
        heapsort(n);
        printf("elemensts after sorted\n");
        for(i=1;i<=n;i++)
                printf("%d ",a[i]);
        getch();
        return 0;
}
void heapsort(int n)
{
        int i,temp;
        for(i=n/2;i>=1;i--)
                Maxheapify(n,i);
        printf("Max Heap elements \n");
        for(i=1;i<=n;i++)
                printf("%d ",a[i]);
        for(i=n;i>=1;i--)
        {
```

```
                temp=a[1];
                a[1]=a[n];
                a[n]=temp;
                n=n-1;
                Maxheapify(n,1);
        }
}

void Maxheapify(int n, int i)
{
        int largest=i,l,r,temp;
        l=2*i;
        r=2*i+1;
        while(l<=n && a[l]>a[largest])
                largest=l;
        while(r<=n && a[r]>a[largest])
            largest=r;
        if(largest!=i)
        {
                temp=a[largest];
                a[largest]=a[i];
                a[i]=temp;
                Maxheapify(n,largest);
        }
}
```