```c
#include <stdio.h>
#include <stdlib.h>

//?????? ??? ????? ?? char.
typedef char BinTreeElementType;

typedef struct BinTreeNode *BinTreePointer;
 struct BinTreeNode {
    BinTreeElementType Data;
    BinTreePointer LChild, RChild;
} ;

typedef enum {
    FALSE, TRUE
} boolean;


void CreateBST(BinTreePointer *Root);
boolean EmptyBST(BinTreePointer Root);
void BSTInsert(BinTreePointer *Root, BinTreeElementType Item);
void BSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found, BinTreePointer *LocPtr);
void BSTSearch2(BinTreePointer Root, BinTreeElementType Item, boolean *Found,BinTreePointer *LocPtr,
BinTreePointer *Parent);
void BSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue);
void InorderTraversal(BinTreePointer Root);
int BSTLevel(BinTreeElementType Item,BinTreePointer Root);

int main()
{
    //?????? ??????????.
    BinTreePointer ARoot;
    int i,j,level;
    boolean found;
    char str[] = "PROCEDURE";

    CreateBST(&ARoot);

    //??????? ??? ??? ???? ?????????? ???? ???? ???.
    for(i=0; str[i] != '\0'; i++)
        BSTInsert(&ARoot,str[i]);

    //???????? ??? level ???? ????????? ??? ??????? ??? ???.
    for(i=0; str[i] != '\0'; i++)
    {
        //??????? ??? ?? ????????? ???? ??? ???? ??? ???? ??????.
        found = FALSE;
        for(j=0; j < i; j++)
        {
            if(str[i] == str[j])
            {
                found = TRUE;
                break;
            }
        }
        if(!found)
        {
            level = BSTLevel(str[i],ARoot);
            printf("Level of %c: %d\n",str[i],level);
        }
    }


    return 0;
}

void CreateBST(BinTreePointer *Root)
```

```
{
    *Root = NULL;
}

boolean EmptyBST(BinTreePointer Root)
{
    return (Root==NULL);
}

void BSTInsert(BinTreePointer *Root, BinTreeElementType Item)
{
    BinTreePointer LocPtr, Parent;
    boolean Found;

    LocPtr = *Root;
    Parent = NULL;
    Found = FALSE;
    while (!Found && LocPtr != NULL) {
        Parent = LocPtr;
        if (Item < LocPtr->Data)
            LocPtr = LocPtr ->LChild;
        else if (Item > LocPtr ->Data)
            LocPtr = LocPtr ->RChild;
        else
            Found = TRUE;
    }
    if (Found)
        printf("To %c EINAI HDH STO DDA\n", Item);
    else {
        LocPtr = (BinTreePointer)malloc(sizeof (struct BinTreeNode));
        LocPtr ->Data = Item;
        LocPtr ->LChild = NULL;
        LocPtr ->RChild = NULL;
        if (Parent == NULL)
            *Root = LocPtr;
        else if (Item < Parent ->Data)
            Parent ->LChild = LocPtr;
        else
            Parent ->RChild = LocPtr;
    }
}

void BSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,BinTreePointer *LocPtr)
{

    (*LocPtr) = Root;
    (*Found) = FALSE;
    while (!(*Found) && (*LocPtr) != NULL)
    {
        if (KeyValue < (*LocPtr)->Data)
            (*LocPtr) = (*LocPtr)->LChild;
        else
            if (KeyValue > (*LocPtr)->Data)
                (*LocPtr) = (*LocPtr)->RChild;
            else (*Found) = TRUE;
    }
}

void BSTSearch2(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,BinTreePointer *LocPtr,
BinTreePointer *Parent)
{
    *LocPtr = Root;
    *Parent=NULL;
    *Found = FALSE;
    while (!(*Found) && *LocPtr != NULL)
    {
```

```
        if (KeyValue < (*LocPtr)->Data) {
            *Parent=*LocPtr;
            *LocPtr = (*LocPtr)->LChild;
        }
        else
            if (KeyValue > (*LocPtr)->Data) {
                *Parent=*LocPtr;
                *LocPtr = (*LocPtr)->RChild;
            }
            else *Found = TRUE;
    }

}

void BSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue)
{

   BinTreePointer
   n,
   Parent,
   nNext,
   SubTree;
   boolean Found;

     BSTSearch2(*Root, KeyValue, &Found , &n, &Parent);
     if (!Found)
        printf("TO STOIXEIO %d DEN EINAI STO DDA\n", KeyValue);
     else {
          if (n->LChild != NULL && n->RChild != NULL)
          {
                 nNext = n->RChild;
                 Parent = n;
                 while (nNext->LChild !=NULL)
                 {
                      Parent = nNext;
                      nNext = nNext->LChild;
                 }
                n->Data = nNext->Data;
                n = nNext;
          }
          SubTree = n->LChild;
          if (SubTree == NULL)
             SubTree = n->RChild;
          if (Parent == NULL)
             *Root = SubTree;
          else if (Parent->LChild == n)
                  Parent->LChild = SubTree;
               else
                   Parent->RChild = SubTree;
          free(n);
     }
}

void InorderTraversal(BinTreePointer Root)
{
    if (Root!=NULL) {
        InorderTraversal(Root->LChild);
        printf("%c ",Root->Data);
        InorderTraversal(Root->RChild);
    }
}

int BSTLevel(BinTreeElementType Item,BinTreePointer Root)
{
    //?????? ??????????.
    boolean Found = FALSE;
```

```
    int level = 1;

    //???? ?????????? ????? ?? ?????? ?? ???????? ? ????? ?? "??????????" ?? ???.
    while(!(Found) && (Root != NULL))
    {
        /*???????? ??? ????????? ?? ???? ??? ?????? ASCII ?????????????? ??? ????
         ????????? ???? ?? BSTSearch.*/
        if(Root->Data == Item)
            Found=TRUE;
        else
        {
            if(Item < Root->Data)
                Root = Root->LChild;
            else
                Root = Root->RChild;

            level++;
        }
    }

    //????????? ????? ??????? ?? ??????? ? ??? ?? ???????? ??? ???.
    if(Found)
        return level;
    else
        return -1;
}
```