```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 MinBSTValue(BinTreePointer Root);
int MaxBSTValue(BinTreePointer Root);
int main()
{
    //?????? ??????????.
    BinTreePointer ARoot;
    int i,min,max;
    char str[] = "PROCEDURE";

    CreateBST(&ARoot);

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


    //??????? ??? ????? ??? ???????????? ??? ??? ??????????? ?? ???????? ??????????.
    min = MinBSTValue(ARoot);
    max = MaxBSTValue(ARoot);

    //???????? ?????????????.
    printf("Min BST Value: %c\n",min);
    printf("Max BST Value: %c\n",max);

    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;
```

```
 66
 67      LocPtr = *Root;
 68      Parent = NULL;
 69      Found = FALSE;
 70      while (!Found && LocPtr != NULL) {
 71          Parent = LocPtr;
 72          if (Item < LocPtr->Data)
 73              LocPtr = LocPtr ->LChild;
 74          else if (Item > LocPtr ->Data)
 75              LocPtr = LocPtr ->RChild;
 76          else
 77              Found = TRUE;
 78      }
 79      if (Found)
 80          printf("To %c EINAI HDH STO DDA\n", Item);
 81      else {
 82          LocPtr = (BinTreePointer)malloc(sizeof (struct BinTreeNode));
 83          LocPtr ->Data = Item;
 84          LocPtr ->LChild = NULL;
 85          LocPtr ->RChild = NULL;
 86          if (Parent == NULL)
 87              *Root = LocPtr;
 88          else if (Item < Parent ->Data)
 89              Parent ->LChild = LocPtr;
 90          else
 91              Parent ->RChild = LocPtr;
 92      }
 93  }
 94
 95  void BSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,BinTreePointer *LocPtr)
 96  {
 97
 98      (*LocPtr) = Root;
 99      (*Found) = FALSE;
100      while (!(*Found) && (*LocPtr) != NULL)
101      {
102          if (KeyValue < (*LocPtr)->Data)
103              (*LocPtr) = (*LocPtr)->LChild;
104          else
105              if (KeyValue > (*LocPtr)->Data)
106                  (*LocPtr) = (*LocPtr)->RChild;
107              else (*Found) = TRUE;
108      }
109  }
110
111  void BSTSearch2(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,BinTreePointer *LocPtr,
BinTreePointer *Parent)
112  {
113      *LocPtr = Root;
114      *Parent=NULL;
115      *Found = FALSE;
116      while (!(*Found) && *LocPtr != NULL)
117      {
118          if (KeyValue < (*LocPtr)->Data) {
119              *Parent=*LocPtr;
120              *LocPtr = (*LocPtr)->LChild;
121          }
122          else
123              if (KeyValue > (*LocPtr)->Data) {
124                  *Parent=*LocPtr;
125                  *LocPtr = (*LocPtr)->RChild;
126              }
127              else *Found = TRUE;
128      }
129
130  }
```

```
131
132 void BSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue)
133 {
134
135    BinTreePointer
136    n,
137    Parent,
138    nNext,
139    SubTree;
140    boolean Found;
141
142      BSTSearch2(*Root, KeyValue, &Found , &n, &Parent);
143      if (!Found)
144         printf("TO STOIXEIO %d DEN EINAI STO DDA\n", KeyValue);
145      else {
146           if (n->LChild != NULL && n->RChild != NULL)
147           {
148                  nNext = n->RChild;
149                  Parent = n;
150                  while (nNext->LChild !=NULL)
151                  {
152                       Parent = nNext;
153                       nNext = nNext->LChild;
154                  }
155                 n->Data = nNext->Data;
156                 n = nNext;
157           }
158           SubTree = n->LChild;
159           if (SubTree == NULL)
160              SubTree = n->RChild;
161           if (Parent == NULL)
162              *Root = SubTree;
163           else if (Parent->LChild == n)
164                   Parent->LChild = SubTree;
165                else
166                    Parent->RChild = SubTree;
167           free(n);
168       }
169 }
170
171 void InorderTraversal(BinTreePointer Root)
172 {
173     if (Root!=NULL) {
174         InorderTraversal(Root->LChild);
175         printf("%d ",Root->Data);
176         InorderTraversal(Root->RChild);
177     }
178 }
179
180
181 int MinBSTValue(BinTreePointer Root)
182 {
183     //???? ?????????? ???? ?? ????????????? ?? ???????? ???????? ????? ?? ??? ??????? ??? ????????(NULL).
184     while(Root->LChild != NULL)
185         Root=Root->LChild;
186
187     //E???????? ??? ????? ??? ?????? ??? ?????????? ????????.
188     return Root->Data;
189 }
190
191 int MaxBSTValue(BinTreePointer Root)
192 {
193     //???? ?????????? ???? ?? ????????????? ?? ???? ???????? ????? ?? ??? ??????? ??? ?????(NULL).
194     while(Root->RChild != NULL)
195         Root=Root->RChild;
196
```

```
197     //E???????? ??? ????? ??? ?????? ??? ?????????? ????????.
198      return Root->Data;
199  }
```