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


typedef int BinTreeElementType;

typedef struct BinTreeNode *BinTreePointer;

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

typedef enum {
    FALSE, TRUE
} boolean;


void CreateBST(BinTreePointer *Root);
boolean BSTEmpty(BinTreePointer Root);
void RecBSTInsert(BinTreePointer *Root, BinTreeElementType Item);
void RecBSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found, BinTreePointer *LocPtr
);
void RecBSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue);
void RecBSTInorder(BinTreePointer Root);
void RecBSTPreorder(BinTreePointer Root);
void RecBSTPostorder(BinTreePointer Root);
int Number_Ceiling(BinTreePointer Root, BinTreeElementType Item);
int Number_Floor(BinTreePointer Root, BinTreeElementType Item);

int main()
{
    //?????? ??????????.
    BinTreePointer ARoot;
    BinTreeElementType Insert_Item,Search_Item;
    int Ceiling,Floor;

    //?????????? ????? ???
    CreateBST(&ARoot);

    /*???? ?????????? ??? ???????? ????????? ??? ??? ????? ?? ????? ????????? ???????.*/
    while(TRUE)
    {
        printf("Enter number to insert: ");
        scanf("%d",&Insert_Item);

        if(Insert_Item < 0)
            break;
        else if(Insert_Item == 0)
            printf("Number must be > 0 or < 0(If you want to stop inserting).\n");
        else
            RecBSTInsert(&ARoot,Insert_Item);
    }

    /*???? ?????????? ??? ????????? ???? ????????? ??? ?????? ??? Ceiling ??? Floor ???
    ????? ?? ????? ????????? ???????.*/
    while(TRUE)
    {
        printf("Enter number to search: ");
        scanf("%d",&Search_Item);

        if(Search_Item < 0)
            break;
        else if(Search_Item == 0)
            printf("Number must be > 0 or < 0(If you want to stop inserting).\n");
        else
```

```
66          {
67              Ceiling = Number_Ceiling(ARoot,Search_Item);
68              printf("Ceiling:%d\n",Ceiling);
69
70              Floor = Number_Floor(ARoot,Search_Item);
71              printf("Floor:%d\n",Floor);
72          }
73      }
74
75      return 0;
76  }
77
78  void CreateBST(BinTreePointer *Root)
79  {
80      *Root = NULL;
81  }
82
83  boolean BSTEmpty(BinTreePointer Root)
84  {
85      return (Root==NULL);
86  }
87
88  void RecBSTInsert(BinTreePointer *Root, BinTreeElementType Item)
89  {
90      if (BSTEmpty(*Root)) {
91          (*Root) = (BinTreePointer)malloc(sizeof (struct BinTreeNode));
92          (*Root) ->Data = Item;
93          (*Root) ->LChild = NULL;
94          (*Root) ->RChild = NULL;
95      }
96      else
97          if (Item < (*Root) ->Data)
98              RecBSTInsert(&(*Root) ->LChild,Item);
99          else if (Item > (*Root) ->Data)
100             RecBSTInsert(&(*Root) ->RChild,Item);
101         else
102             printf("To %d EINAI HDH STO DDA\n", Item);
103 }
104
105 void RecBSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found, BinTreePointer *LocPtr)
106 {
107
108     if (BSTEmpty(Root))
109         *Found=FALSE;
110     else
111         if (KeyValue < Root->Data)
112             RecBSTSearch(Root->LChild, KeyValue, &(*Found), &(*LocPtr));
113         else
114             if (KeyValue > Root->Data)
115                 RecBSTSearch(Root->RChild, KeyValue, &(*Found), &(*LocPtr));
116             else
117                 {
118                     *Found = TRUE;
119                     *LocPtr=Root;
120                 }
121 }
122
123 void RecBSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue)
124 {
125
126    BinTreePointer TempPtr;          //* true AN TO STOIXEIO KeyValue EINAI STOIXEIO TOY DDA *)
127
128     if (BSTEmpty(*Root))          //* ????? ?????? ?? KeyValue ?? ?? ?????? *)
129         printf("to %d DeN BRE8HKe STO DDA\n", KeyValue);
130      else
131          //* ????????? ?????????? ??? ????? ??? ???????? ??? ???? KeyValue ??? ???????? ???
```

```
132           if (KeyValue < (*Root)->Data)
133              RecBSTDelete(&((*Root)->LChild), KeyValue);       //* ???????? ????????? *
134           else
135             if (KeyValue > (*Root)->Data)
136                   RecBSTDelete(&((*Root)->RChild), KeyValue);   //* ????  ????????? *
137             else                                        //* TO KeyValue ??????? ???????? ??? ?????? *)
138                 if ((*Root)->LChild ==NULL)
139                  {
140                       TempPtr = *Root;
141                       *Root = (*Root)->RChild;      //* ??? ???? ???????? ????? *)
142                       free(TempPtr);
143                  }
144                 else if ((*Root)->RChild == NULL)
145                    {
146                         TempPtr = *Root;
147                         *Root = (*Root)->LChild;      //* ???? ???????? ?????, ???? ??? ???? *)
148                         free(TempPtr);
149                    }
150                    else                               //* ???? 2 ?????? *)
151                    {
152                         //* ?????? ??? INORDER ???????? ??? *)
153                         TempPtr = (*Root)->RChild;
154                         while (TempPtr->LChild != NULL)
155                               TempPtr = TempPtr->LChild;
156                         /* ?????????? ??? ???????? ??? ????? ??? ??????????
157                         ??? ?????????? ??? ???????? ??? ???????? ?????? */
158                         (*Root)->Data = TempPtr->Data;
159                         RecBSTDelete(&((*Root)->RChild), (*Root)->Data);
160                    }
161 }
162
163 void RecBSTInorder(BinTreePointer Root)
164 {
165     if (Root!=NULL) {
166         printf("L");
167         RecBSTInorder(Root->LChild);
168         printf("/%d/",Root->Data);
169         printf("R");
170         RecBSTInorder(Root->RChild);
171     }
172     printf("U");
173 }
174
175 void RecBSTPreorder(BinTreePointer Root)
176 {
177     if (Root!=NULL) {
178         printf("/%d /",Root->Data);
179         printf("L");
180         RecBSTPreorder(Root->LChild);
181         printf("R");
182         RecBSTPreorder(Root->RChild);
183     }
184     printf("U");
185 }
186
187 void RecBSTPostorder(BinTreePointer Root)
188 {
189     if (Root!=NULL) {
190         printf("L");
191         RecBSTPostorder(Root->LChild);
192         printf("R");
193         RecBSTPostorder(Root->RChild);
194         printf("/%d /",Root->Data);
195     }
196     printf("U");
197 }
```

```
198
199
200 int Number_Ceiling(BinTreePointer Root, BinTreeElementType Item)
201 {
202     int ceil;
203
204     if(BSTEmpty(Root))
205         return -1;
206     else if (Root->Data == Item)
207         return Item;
208     else if (Item > Root->Data)
209         return Number_Ceiling(Root->RChild,Item);
210     else
211         ceil = Number_Ceiling(Root->LChild,Item);
212
213     if(ceil == -1)
214         return Root->Data;
215
216     return ceil;
217
218 }
219
220 int Number_Floor(BinTreePointer Root, BinTreeElementType Item)
221 {
222     int floor;
223
224     if(BSTEmpty(Root))
225         return -1;
226     else if(Root->Data == Item)
227         return Item;
228     else if(Item < Root->Data)
229         return Number_Floor(Root->LChild,Item);
230     else
231         floor = Number_Floor(Root->RChild,Item);
232
233     if(floor == -1)
234         return Root->Data;
235
236     return floor;
237 }
238
```