

```

1: #include<stdio.h>
2: #include<stdlib.h>
3:
4: struct  node* t=NULL;
5:
6: struct node{
7:     int data;
8:     struct node* left;
9:     struct node* right;
10:    struct node* parent;
11:
12: };
13:
14: struct node* cnode(int v){
15:     struct node* temp=(struct node*)malloc(sizeof(struct node));
16:     temp->data=v;
17:     temp->left=temp->right=NULL;
18:     temp->parent=NULL;
19:     return(temp);
20: }
21:
22: /*int maxof(int a,int b){
23:     if(a>b)return a;
24:     return b;
25: }
26:
27: void rotate_right(struct node* t){
28:     int x=t->data;
29:     int y=t->left->data;
30:     struct node* tll=t->left->left;
31:     struct node* tlr=t->left->right;
32:     struct node* tr=t->right;
33:
34:     t->data=y;
35:     t->right=t->left;
36:     t->right->data=x;
37:     t->left=tll;
38:     t->right->left=tlr;
39:     t->right->right=tr;
40: }
41:
42: void rotate_left(struct node* t){
43:     int y=t->data;
44:     int z=t->right->data;
45:     struct node* tll=t->left;
46:     struct node* tlrl=t->right->left;
47:     struct node* tlrr=t->right->right;
48:
49:     t->data=z;
50:     t->left=t->right;
51:     t->left->data=y;
52:     t->left->left=tll;
53:     t->left->right=tlrl;
54:     t->right=tlrr;

```

```

55: }
56:
57: int slope(struct node* t){
58:     return(t->left->ht-t->right->ht);
59: }
60:
61: void rebalance(struct node* t){
62:     if(slope(t)==2){
63:         if(slope(t->left)==-1)
64:             rotate_left(t->left);
65:         rotate_right(t);
66:     }
67:     if(slope(t)==-2){
68:         if(slope(t->right)==1)
69:             rotate_right(t->right);
70:         rotate_left(t);
71:     }
72:     return;
73: }*/
74: struct node* min(struct node* t){
75:     struct node* temp=t;
76:     while(temp->left!=NULL){
77:         temp=temp->left;
78:     }
79:     return(temp);
80: }
81: struct node* max(struct node* t){
82:     struct node* temp=t;
83:     while(temp->right!=NULL){
84:         temp=temp->right;
85:     }
86:     return(temp);
87: }
88: struct node* find(struct node* t,int v){
89:     if(v==t->data)
90:         return t;
91:     else if(v<t->data){
92:         find(t->left,v);
93:     }
94:     else{
95:         find(t->right,v);
96:     }
97: }
98: struct node* succ(struct node* t,int v){
99:     struct node* temp=find(t,v);
100:    if(temp->right!=NULL){
101:        return(min(temp->right));
102:    }
103:    struct node* temp2=temp->parent;
104:    while(temp2->parent!=NULL&&temp2->right==temp){
105:        temp=temp2;
106:        temp2=temp2->parent;
107:    }
108:    }

```

```

109:     if(temp2->parent==NULL&&temp2->right==temp){
110:         return temp2->parent;
111:     }
112:     return(temp2);
113:
114:
115: }
116: struct node* pred(struct node* t,int v){
117:     struct node* temp=find(t,v);
118:     if(temp->left!=NULL){
119:         return(max(temp->left));
120:     }
121:     struct node* temp2=temp->parent;
122:     while(temp2->parent!=NULL&&temp2->left==temp){
123:         temp=temp2;
124:         temp2=temp2->parent;
125:     }
126:     if(temp2->parent==NULL&&temp2->left==temp){
127:         return(temp2->parent);
128:     }
129:     return(temp2);
130: }
131:
132: struct node* insert(struct node* t,int v){
133:     if(t==NULL){
134:         t=cnode(v);
135:         return t;
136:     }
137:     else if(v<t->data){
138:
139:         t->left=insert(t->left,v);
140:         //rebalance(t->left);
141:         // t->ht=1+maxof(t->left->ht,t->right->ht);
142:         t->left->parent=t;
143:     }
144:     else if(v>t->data){
145:         t->right=insert(t->right,v);
146:         //rebalance(t->right);
147:         // t->ht=1+maxof(t->left->ht,t->right->ht);
148:         t->right->parent=t;
149:     }
150:     return t;
151: }
152:
153: struct node* del(struct node* t,int v){
154:     if(t==NULL)
155:         return t;
156:     if(v<t->data){
157:         t->left=del(t->left,v);
158:     }
159:     else if(v>t->data){
160:         t->right=del(t->right,v);
161:     }
162:     else{

```

```

163:         if(t->left==NULL){
164:             struct node* temp=t->right;
165:             free(t);
166:             return temp;
167:         }
168:         else if(t->right==NULL){
169:             struct node* temp=t->left;
170:             free(t);
171:             return temp;
172:         }
173:         else if(t->left!=NULL&& t->right!=NULL){
174:             struct node* p=min(t);
175:             t->data=p->data;
176:             t->right=del(t->right,t->data);
177:         }
178:     }
179: }
180: }
181: void inorder(struct node* t){
182:     if(t!=NULL){
183:         inorder(t->left);
184:         printf("%d:\n",t->data);
185:         inorder(t->right);
186:     }
187: }
188: int main(){
189:     t=insert(t,5);
190:     t=insert(t,3);
191:     t=insert(t,10);
192:     t=insert(t,1);
193:     t=insert(t,2);
194:     t=insert(t,4);
195:     t=insert(t,11);
196:     inorder(t);
197:     int p;
198:     printf("\n");
199:     printf("For which value you want to know SUccessor: ");
200:     scanf("%d",&p);
201:     struct node* k=succ(t,p);
202:
203:     if(k!=NULL){
204:
205:         printf(" SUCCESSOR od %d IS: %d \n",p,k->data);
206:     }
207:     else{
208:         printf("NO SUCCESSOR EXIST for %d!",p);
209:     }
210:     k=NULL;
211:     printf("\n");
212:     printf("For which value you want to know predeccor: ");
213:     scanf("%d",&p);
214:     k=pred(t,p);
215:     if(k!=NULL){
216:

```

```
217:         printf("\npredeccsor of %d IS: %d \n",p,k->data);
218:     }
219:     else{
220:         printf("\nNO predescor EXIST!");
221:     }
222:
223:     printf("\n");
224: }
```