

```

1 #include <iostream>
2 #include <fstream>
3 using namespace std;
4
5 class Node{
6     public:
7
8     Node* parent;
9     int key1;
10    int key2;
11    Node* child1;
12    Node* child2;
13    Node* child3;
14
15    // Constructor for internal node
16    Node(Node* newParent, int newKey1, int newKey2, Node* newChild1, Node* newChild2, Node* newChild3){
17        parent = newParent;
18        key1 = newKey1;
19        key2 = newKey2;
20        child1 = newChild1;
21        child2 = newChild2;
22        child3 = newChild3;
23    } //Node()
24
25    // Constructor for leaf node
26    Node(Node* newParent, int data){
27        parent = newParent;
28        key1 = -1;
29        key2 = data;
30    } //Node()
31
32    void printNode(fstream& outstream){
33        outstream << "(parent.key1 = ";
34
35        if(parent!=NULL)
36            outstream << parent->key1;
37        else
38            outstream << "0";
39
40        outstream << ", key1 = " << key1;
41        outstream << ", key2 = " << key2 << ", child1 = ";
42
43        if(child1 != NULL)
44            outstream << child1->key1;
45        else
46            outstream << "-1";
47
48        outstream << ", child2 = ";
49
50        if(child2 != NULL)
51            outstream << child2->key1;
52        else
53            outstream << "-1";
54
55        outstream << ", child3 = ";
56
57        if(child3 != NULL)
58            outstream << child2->key1;
59        else
60            outstream << "-1";
61
62        outstream << ")" << endl;
63    } //printNode()
64 } //Node class
65
66 class Tree{
67     public:
68
69     Node* root;
70
71     Tree(){
72         root = new Node(NULL, -1, -1, NULL, NULL, NULL);
73     } //Tree()
74
75     void updateKeys(Node* spot){
76         // If we're past the root of the tree (this function traverses up the tree),
77         // we are done
78         if(spot==NULL)
79             return;
80
81         if(spot->child2 != NULL && spot->child2->key1 == -1)
82             spot->key1 = spot->child2->key2;
83
84         else if(spot->child2 == NULL)
85             spot->key1 = -1;

```

```

86     else if(spot->child2 != NULL && spot->child2->key1 != -1){
87         //Find the smallest node in the second subtree
88         Node* p = spot->child2;
89         while(p->key1 != -1)
90             p = p->child1;
91         spot->key1 = p->key2;
92     }//else if
93
94     if(spot->child3 != NULL && spot->child3->key1 == -1)
95         spot->key2 = spot->child3->key2;
96
97     else if(spot->child3 == NULL)
98         spot->key2 = -1;
99
100     else if(spot->child3 != NULL && spot->child3->key1 != -1){
101         Node* p = spot->child3;
102         while(p->key1 != -1)
103             p = p->child1; //while
104         spot->key2 = p->key2;
105     }//else if
106     updateKeys(spot->parent);
107 }//updateKeys()
108
109 void insert(Node* spot, Node* newData){
110     cout << "In insert();" << endl <<
111         "newData->key1 = " << newData->key1 << ", newData->key2 = " << newData->key2 << endl;
112
113     // Case 0: Spot has 0 children
114     if(spot->child1 == NULL && spot->child2 == NULL && spot->child3 == NULL){
115         cout << "Case 0" << endl;
116         newData->parent=spot;
117         spot->child1 = newData;
118     }
119
120     // Case 1: Spot has 1 child
121     else if(spot->child1 != NULL && spot->child2 == NULL && spot->child3 == NULL){
122         cout << "Case 1" << endl;
123         newData->parent=spot;
124         if(spot->child1->key2 <= newData->key2){
125             spot->child2 = newData;
126             cout << "spot->child2 = newData" << endl;
127         }
128         else {
129             spot->child2 = spot->child1;
130             spot->child1 = newData;
131         }//else
132     }
133
134     // Case 2: Spot has 2 children
135     else if(spot->child1 != NULL && spot->child2 != NULL && spot->child3 == NULL){
136         cout << "Case 2" << endl;
137         newData->parent=spot;
138         // Modified bubble sort from beginning of semester
139         // to put children in ascending order
140         // It's O(n^2), but we only have 4 items max
141         Node* children[3] = {spot->child1, spot->child2, newData};
142         int begin=0, end=2, swapflag=1, walker;
143
144         while(end>begin && swapflag>0){
145             walker=begin;
146             if(swapflag>=1)
147                 swapflag=0;
148             while(walker < end){
149                 if(children[walker]->key2 > children[walker+1]->key2){
150                     //swap
151                     Node* temp=children[walker];
152                     children[walker]=children[walker+1];
153                     children[walker+1]=temp;
154                     swapflag++;
155                     walker++;
156                 }//if
157             }
158             end--;
159         }//while
160         spot->child1 = children[0];
161         spot->child2 = children[1];
162         spot->child3 = children[2];
163     }//if
164
165     // Case 3: Spot has 3 children
166     else if(spot->child1 != NULL && spot->child2 != NULL && spot->child3 != NULL){
167         cout << "Case 3" << endl;
168         Node* children[4] = {spot->child1, spot->child2,

```

```

171     spot->child3, newData};
172     int begin=0, end=3, swapflag=1, walker;
173     cout << "Done building array; starting sort" << endl;
174
175     while(end>begin && swapflag>0){
176         swapflag=0;
177         walker=begin;
178         while(walker < end){
179             cout << "In while(); walker = " << walker << endl;
180             if(children[walker]->key1 > children[walker+1]->key1){
181                 cout << "In if();" << endl;
182                 cout << "children[" << walker << "]==" << children[walker] << endl;
183                 //swap
184                 Node* temp=children[walker];
185                 cout << "Node* temp=children[walker];" << endl;
186                 children[walker]=children[walker+1];
187                 cout << "children[walker]=children[walker+1];" << endl;
188                 children[walker+1]=temp;
189                 cout << "children[walker+1]=temp;" << endl;
190                 swapflag++;
191                 walker++;
192             }//if
193             end--;
194         }//while
195     }//while
196     cout << "Done sorting" << endl;
197
198     cout << "Printing contents of 4 nodes" << endl;
199     for(int i=0; i<4; i++){
200         cout << "children[" << i << "].key2 = " << children[i]->key2 << endl;
201     }
202
203     spot->child1 = children[0];
204     spot->child2 = children[1];
205
206     Node* newNode = new Node(NULL, -1, -1, NULL, NULL, NULL);
207     Node* newChild1 = children[2];
208     Node* newChild2 = children[3];
209     newNode->child1 = newChild1;
210     newNode->child2 = newChild2;
211     newNode->child1->parent = newNode;
212     newNode->child2->parent = newNode;
213
214     updateKeys(spot);
215     updateKeys(newNode);
216
217     //cout << "About to make recursive call of insert()" << endl;
218
219     if(spot->parent==NULL){
220         root = spot->parent = new Node(NULL, -1, -1, spot, newNode, NULL);
221         newNode->parent = spot->parent;
222     }//if
223     insert(spot->parent, newNode);
224 }//else if
225
226
227     // Update the keys
228     updateKeys(spot);
229 }//insert()
230
231 void printTree(Node* nodeToPrint, fstream& outstream){
232     if(nodeToPrint==NULL)
233         return;
234     else{
235         nodeToPrint->printNode(outstream);
236         printTree(nodeToPrint->child1, outstream);
237         printTree(nodeToPrint->child2, outstream);
238         printTree(nodeToPrint->child3, outstream);
239     }//else
240 }//printTree()
241
242 Node* findSpot(Node* currentNode, int data){
243     if(currentNode->child1 == NULL || currentNode->child1->key1 == -1)
244         return currentNode;
245
246     else if(data < currentNode->key1)
247         return findSpot(currentNode->child1, data);
248
249     else if((data >= currentNode->key1 && data > currentNode->key2) || currentNode->key2 == -1)
250         return findSpot(currentNode->child2, data);
251
252     else if(data >= currentNode->key2 && currentNode->key2 == -1)
253         return findSpot(currentNode->child3, data);
254 }//findSpot()
255

```

```

256 };//Tree class
257
258 int main(){
259     fstream infile;
260     infile.open("infile.txt", fstream::in);
261
262     if(!infile.is_open()){
263         cout << "Error opening infile.txt, quitting." << endl;
264         return 0;
265     }//if
266
267     fstream outfile;
268     outfile.open("outfile.txt", fstream::out | fstream::trunc);
269
270     if (!outfile.is_open()) {
271         cout << "Error opening outfile.txt, quitting." << endl;
272         return 0;
273     }//if
274
275     Tree* tree = new Tree();
276
277     while(!infile.eof()){
278         int input;
279         infile >> input;
280         Node* spot = tree->findSpot(tree->root, input);
281         Node* newNode = new Node(NULL, input);
282         outfile << "Tree with " << input << " inserted:" << endl;
283         tree->insert(spot, newNode);
284         tree->printTree(tree->root, outfile);
285         outfile << "-----" << endl;
286     }//while
287     cout << "Completed insert operation" << endl;
288     return 0;
289 }//main()

```