```cpp
#include <iostream>          //  cout
using namespace std;
struct Node{
    char ID;
    Node *left, *right;
    Node(const char id, Node * l , Node *r){
      ID= id;
      left = l;
      right = r;
    }
};

Node* gRoot = nullptr; ///< Rotpeker til hele treet
bool gKomplettTre = true,   ///< Er treet komplett eller ei.
    gNivaaOpp = false ; ///< Gått opp ett nivå en gang eller ei.
int gDybde = 0 , ///< Aktuell max.dybde å sjekke noder opp mot.
    gNivaa = -1; ///< Aktuelt nivå node/nullptr er på.

/**
 * 1. traverserer treet fra høyre barn til venstre. dvs høyre barn først
 *    dermed seg selv også venstre barn.(reverse inorder)
 *
 * 2. Finner aller først den høyrenoden lengst til høyre. dvs første node
 *    uten høyre barn helt til høyre. Når denne blir funnet setter den gDybde
 *    til å være gNivaa.(gNivaaOpp må være false for at sjekken skal foretas)
 *
 * 3. Dersom den finner en node på samme nivaa som gDybde og har enten kun
 *    venstre barn eller begge barna.
 *    Vil den nye gDybden være gNivaa +1 og gNivaaOpp vil være true.
 *    (gNivaaOpp må være false for at sjekken skal foretas). Den har nå
 *    funnet maksdybden et komplett tre kan være.
 *
 * Krav for ett fullt tre
 * 4. Noder med gNivaa < gDybde må ha ett fult sett med barn.
 *
 * 5. Dersom gNivaaOpp er true dvs. et komplett tres maks dybde er funnet.
 *    Da må alle noder ha nivå lik eller under gDybde.
 *
 * 6. Dersom en node har ett høyre barn men ingen venstre barn er det ikke et
 *    komplett tre(uansett).
 */
void erKomplettTre(Node * node){
  if(node && gKomplettTre){
    gNivaa++;
    //sjekker om krav for ett komplett tre er brutt
    if((gNivaa < gDybde && !(node->left && node->right))//Er nivået til noden
                                                        //lavere enn gDybde
                                                        //og har den ikke
                                                       //fullt barn?(punkt 4)

        || (gNivaaOpp && gNivaa > gDybde)//Har gDybde gått opp et nivå
                                         //og er nivået til noden større enn
                                         //denne dybden? (punkt 5)
        || (node->right && !node->left)){//Har noden ett høyre barn uten ett
                                          //venstre barn? (punkt 6)
          gKomplettTre = false; //Hvis ja, da er det ikke et komplett tre.
          return; //De neste linjene vil ikke kjøres.
    }
    erKomplettTre(node->right);//kaller seg selv rekursivt med høyre barn.
```

```
                                //Høyre barn lengst til høyre vil da bli funnet
                                //først. (punkt 1)
    // leter etter maks dybde
    if(!gNivaaOpp && !node->right){//Har gdybden ikke gått opp et nivå en
                               //gang før og er det ingen høyre barn? (punkt 2)
      gDybde = gNivaa;//Ja, da har høyrebarn lengst til høyre blitt funnet.
    }
    //Er gNivaaOpp false og noden sitt nivå det samme som dybden?
    //Har noden minst ett nodebarn? (tilfellet der noden kun har
    //høyre barn er ikke reelt siden det blir fanget opp i punkt 6) (punkt 3)
    if(!gNivaaOpp && gNivaa == gDybde && (node->left || node->right)){
      gDybde = gNivaa +1;//setter ny gDybde en mer enn node nivå
      gNivaaOpp = true;//Denne inkrementering vil skje kun en gang.
    }
    erKomplettTre(node->left);
    gNivaa--;
  }
}

// teste funksjoner
/**
 * Printing a binary tree
 */
//https://stackoverflow.com/questions/36802354/print-binary-tree-in-a-pretty-way-using-c
void printBT(const std::string &prefix, const Node *node, bool isLeft)
{
    if (node != nullptr)
    {
        std::cout << prefix;
        std::cout << (isLeft ? "|--" : "L--");
        // print the value of the node
        std::cout << (int)node->ID << std::endl;
        // enter the next tree level - left and right branch
        printBT(prefix + (isLeft ? "|   " : "    "), node->right, true);
        printBT(prefix + (isLeft ? "|   " : "    "), node->left, false);
    }
}


void printBT(const Node *node)
{
    printBT("", node, false);
}

void nullStill(){
  gKomplettTre = true;
  gNivaaOpp = false;
  gDybde = 0;
  gNivaa = -1;
}

Node* byggCase1Tre(){
  Node *n1 = new Node(1,nullptr,nullptr);
  return n1;
}

Node* byggCase2Tre(){
  Node *n1 = new Node(1,nullptr,nullptr),
       *n2 = new Node(2,nullptr,n1 );
```

```
118
119   return n2;
120 }
121
122 Node* byggCase3Tre(){
123   Node *n1 = new Node(1,nullptr,nullptr),
124        *n2 = new Node(2,n1,nullptr );
125
126   return n2;
127 }
128
129 Node* byggCase4Tre(){
130   Node *n1 = new Node(1,nullptr,nullptr),
131        *n2 = new Node(2,nullptr,nullptr ),
132        *root = new Node(3,n1,n2);
133   return root ;
134 }
135
136 Node* byggCase5Tre(){
137   Node *n0 = new Node(0,nullptr,nullptr),
138         *n1 = new Node(1,n0,nullptr),
139        *n2 = new Node(2,nullptr,nullptr ),
140        *root = new Node(3,n1,n2);
141   return root ;
142 }
143
144 Node* byggCase6Tre(){
145   Node *n0 = new Node(0,nullptr,nullptr),
146         *n3 = new Node(3, nullptr,nullptr),
147         *n1 = new Node(1,n0,n3),
148        *n2 = new Node(2,nullptr,nullptr ),
149        *root = new Node(3,n1,n2);
150   return root ;
151 }
152
153 Node* byggCase7Tre(){
154
155   Node    *n5 = new  Node(5, nullptr,nullptr),
156     * n2 = new Node(2, n5, nullptr),
157   *n3 = new Node(3, nullptr,nullptr),
158         *n1 = new Node(1,n3,nullptr ),
159        *root = new Node('r',n1,n2);
160   return root ;
161 }
162
163 Node* byggCase8Tre(){
164   Node * n1 = new Node(1,nullptr,nullptr),
165         *n2 = new Node(2,nullptr,nullptr),
166         *n3 = new Node(3,nullptr,nullptr),
167         *n4 = new Node(4, n1,n2),
168         *n5 = new Node(5, n3,nullptr),
169         *n6 = new Node(6, n5,nullptr),
170         *n7 = new Node(7,nullptr,n6),
171         *n8 = new Node(8,n4,n7);
172   return n8;
173 }
174
175 Node* byggCase9Tre(){
176   Node *n1 = new Node(1,nullptr,nullptr),
177       *n5 = new Node(5, nullptr,nullptr),
```

```
      *n2 = new Node(2, n1,nullptr),
      *n3 = new Node(3, n2, nullptr),
      *n4 = new Node(4,n3, n5);
    return n4;
}

Node * byggCase10Tre(){
  Node * n1 = new Node(1,nullptr,nullptr),
        *n3 = new Node(3,nullptr,nullptr),
        *n2 = new Node(2,n1,nullptr),
        *n4 = new Node(4,n3,n2);
        return n4;
}

Node* byggCase11Tre(){
  Node * n1 = new Node(1,nullptr,nullptr),
      * n2 = new Node(2,nullptr,nullptr),
      * n3 = new Node(3,nullptr,nullptr) ,
      * n4 = new Node(4,n1,nullptr),
      * n5 = new Node(5,n4,n2),
      * n6 = new Node(6,n3,nullptr),
      * n7 = new Node(7,n5,n6);
  return n7;
}

Node * byggCase12Tre(){
  Node* n1 = new Node(1,nullptr,nullptr),
      * n2 = new Node(2,nullptr,nullptr),
      * n3 = new Node(3,nullptr,nullptr),
      * n4 = new Node(4,nullptr,nullptr),
      * n5 = new Node(5,nullptr,nullptr),
      * n6 = new Node(6,n2,n3),
      * n7 = new Node(7,n4,n5),
      * n8 = new Node(8,n6,n7),
      * n9 = new Node(9,n1,n8);
    return n9;

}

Node * byggCase13Tre(){
  Node* n6, * n8, * n11a, * n11b, * n12, * n13a, * n13b,
        * n17, * n28, * n31, * n33, * n34, * n35a, * n35b, * n39, * n72;

    n8 = new Node(8, nullptr, nullptr);
    n11b = new Node(11, nullptr, nullptr);
    n13b = new Node(13, nullptr, nullptr);
    n31 = new Node(31, nullptr, nullptr);
    n35b = new Node(35, nullptr, nullptr);
    n34 = new Node(34,n35b, nullptr);
    n12 = new Node(12, n11b, nullptr);
    n39 = new Node(39,nullptr, nullptr);
    n72 = new Node(72, nullptr, nullptr);
    n6 = new Node(6, n72, nullptr);
    n13a = new Node(13, n12, nullptr);
    n28 = new Node(28, nullptr, nullptr);
    n35a = new Node(35, n34, n39);
    n11a = new Node(11, n6, n13a);
    n33 = new Node(33, n28, n35a);
    n17 = new Node(17, n11a, n33);
    gRoot = n17;
```

```
238   return gRoot;
239 }
240
241 Node * byggCase14Tre( ){
242   Node * n1 = new Node(1, nullptr,nullptr),
243        * n2= new Node(2,nullptr,nullptr),
244        * n3= new Node(3,nullptr,nullptr),
245        * n4= new Node(4,nullptr,nullptr),
246        * n5= new Node(5,nullptr,nullptr),
247        * n6= new Node(6,nullptr,nullptr),
248        * n7= new Node(7,nullptr,nullptr),
249        * n8= new Node(8,nullptr,nullptr),
250        * n9= new Node(9,n1,n2),
251        * n10= new Node(10,n3,n4),
252        * n11= new Node(11,n5,n6),
253        * n12= new Node(12,n7,n8),
254        * n13= new Node(13,n9,n10),
255        * n14= new Node(14,n11,n12),
256        * n15= new Node(15,n13,n14);
257     return n15;
258 }
259
260 // https://cplusplus.com/forum/beginner/4639/
261 typedef Node* (*AlleByggeFunksjoner)( );
262
263 int main(int argc, char const *argv[])
264 {
265   AlleByggeFunksjoner byggeFunksjoner[] = {
266     byggCase1Tre,
267     byggCase2Tre,
268     byggCase3Tre,
269     byggCase4Tre,
270     byggCase5Tre,
271     byggCase6Tre,
272     byggCase7Tre,
273     byggCase8Tre,
274     byggCase9Tre,
275     byggCase10Tre,
276     byggCase11Tre,
277     byggCase12Tre,
278     byggCase13Tre,
279     byggCase14Tre
280   };
281
282   const int antallCase = sizeof(byggeFunksjoner)
283                          /sizeof(AlleByggeFunksjoner);
284   int sukksessArray[antallCase] = { 1,0,1,1,1, 1,0,0, 0,0,0 ,0,0,1};
285   int antallSukkses = 0;
286
287   for(int i = 0; i < antallCase; i ++){
288     cout << "case " << (i+1) << " tre" << endl;
289     string komplettEllerIkke = sukksessArray[i] ? "Et " : "Ikke et " ;
290     cout << komplettEllerIkke<< "komplett tre " << endl;
291     gRoot = byggeFunksjoner[i]( );
292     printBT(gRoot);
293     erKomplettTre(gRoot);
294     if(gKomplettTre == sukksessArray[i]){
295       cout << "sukkses" <<endl;
296       antallSukkses++;
297     }  else {
```

```
298       cout << "feil" << endl;
299     }
300     cout << " - - - - - - - - - - \n" << endl;
301     nullStill();
302   }
303 
304   cout << "Testet " << antallCase << " tilfeller." << endl;
305   if(antallCase == antallSukkses){
306     cout << "Alle testene var en sukkses. " << endl;
307   } else {
308     cout << antallSukkses << "/" << antallCase
309          << " tester var sukksesfulle." << endl;
310   }
311   return 0;
312 }
```