```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <cstring> // for string tokenizer and c-style string processing
#include <algorithm> // max function

using namespace std;

class BTNode{

    private:
        int nodeid;
        int data;
        int levelNum;
        BTNode* leftChildPtr;
        BTNode* rightChildPtr;

    public:

        BTNode(){}

        void setNodeId(int id){
            nodeid = id;
        }

        int getNodeId(){
            return nodeid;
        }

        void setData(int d){
            data = d;
        }

        int getData(){
            return data;
        }

        void setLevelNum(int level){
            levelNum = level;
        }

        int getLevelNum(){
            return levelNum;
        }

        void setLeftChildPtr(BTNode* ptr){
            leftChildPtr = ptr;
        }

        void setRightChildPtr(BTNode* ptr){
            rightChildPtr = ptr;
        }

        BTNode* getLeftChildPtr(){
            return leftChildPtr;
        }

        BTNode* getRightChildPtr(){
            return rightChildPtr;
        }

        int getLeftChildID(){
            if (leftChildPtr == 0)
                return -1;
```

```
65
66              return leftChildPtr->getNodeId();
67          }
68
69          int getRightChildID(){
70              if (rightChildPtr == 0)
71                  return -1;
72
73              return rightChildPtr->getNodeId();
74          }
75  };
76
77
78
79  class Node{
80
81      private:
82          int data;
83          Node* nextNodePtr;
84          Node* prevNodePtr;
85
86      public:
87          Node(){}
88
89          void setData(int d){
90              data = d;
91          }
92
93          int getData(){
94              return data;
95          }
96
97          void setNextNodePtr(Node* nodePtr){
98                  nextNodePtr = nodePtr;
99          }
100
101         Node* getNextNodePtr(){
102             return nextNodePtr;
103         }
104
105         void setPrevNodePtr(Node* nodePtr){
106                 prevNodePtr = nodePtr;
107         }
108
109         Node* getPrevNodePtr(){
110             return prevNodePtr;
111         }
112
113 };
114
115 class Queue{
116
117     private:
118         Node* headPtr;
119         Node* tailPtr;
120
121
122     public:
123         Queue(){
124             headPtr = new Node();
125             tailPtr = new Node();
126             headPtr->setNextNodePtr(0);
127             tailPtr->setPrevNodePtr(0);
128         }
```

```
129
130         Node* getHeadPtr(){
131             return headPtr;
132         }
133
134         Node* getTailPtr(){
135             return tailPtr;
136         }
137
138         bool isEmpty(){
139
140             if (headPtr->getNextNodePtr() == 0)
141                 return true;
142
143             return false;
144         }
145
146
147         void enqueue(int data){
148
149             Node* newNodePtr = new Node();
150             newNodePtr->setData(data);
151             newNodePtr->setNextNodePtr(0);
152
153             Node* lastNodePtr = tailPtr->getPrevNodePtr();
154
155             if (lastNodePtr == 0){
156
157                 headPtr->setNextNodePtr(newNodePtr);
158                 newNodePtr->setPrevNodePtr(0);
159
160             }
161             else{
162
163                 lastNodePtr->setNextNodePtr(newNodePtr);
164                 newNodePtr->setPrevNodePtr(lastNodePtr);
165
166             }
167
168             tailPtr->setPrevNodePtr(newNodePtr);
169
170         }
171
172
173         int dequeue(){
174
175             Node* firstNodePtr = headPtr->getNextNodePtr();
176             Node* nextNodePtr = 0;
177
178             int poppedData = -100000; //empty queue
179
180             if (firstNodePtr != 0){
181                 nextNodePtr = firstNodePtr->getNextNodePtr();
182                 poppedData = firstNodePtr->getData();
183             }
184             else
185                 return poppedData;
186
187             if (nextNodePtr != 0){
188                 nextNodePtr->setPrevNodePtr(0);
189                 headPtr->setNextNodePtr(nextNodePtr);
190             }
191             else{
192                 headPtr->setNextNodePtr(0);
```

```
193                 tailPtr->setPrevNodePtr(0);
194             }
195 
196             return poppedData;
197 
198         }
199 
200 
201         int peek(){
202 
203             Node* firstNodePtr = headPtr->getNextNodePtr();
204 
205             if (firstNodePtr != 0)
206                 return firstNodePtr->getData();
207             else
208                 return -100000; //empty queue
209 
210         }
211 
212 
213 };
214 
215 
216 
217 
218 
219 class BinaryTree{
220 
221     private:
222         int numNodes;
223         BTNode* arrayOfBTNodes;
224 
225     public:
226 
227         BinaryTree(int n){
228             numNodes = n;
229             arrayOfBTNodes = new BTNode[numNodes];
230 
231             for (int id = 0; id < numNodes; id++){
232                 arrayOfBTNodes[id].setNodeId(id);
233                 arrayOfBTNodes[id].setLevelNum(-1);
234                 arrayOfBTNodes[id].setLeftChildPtr(0);
235                 arrayOfBTNodes[id].setRightChildPtr(0);
236             }
237         }
238 
239         void setLeftLink(int upstreamNodeID, int downstreamNodeID){
240 
                arrayOfBTNodes[upstreamNodeID].setLeftChildPtr(&arrayOfBTNodes[downstreamNode
                ID]);
241         }
242 
243         void setRightLink(int upstreamNodeID, int downstreamNodeID){
244 
                arrayOfBTNodes[upstreamNodeID].setRightChildPtr(&arrayOfBTNodes[downstreamNod
                eID]);
245         }
246 
247         void printLeafNodes(){
248 
249             for (int id = 0; id < numNodes; id++){
250 
251                 if (arrayOfBTNodes[id].getLeftChildPtr() == 0 &&
                    arrayOfBTNodes[id].getRightChildPtr() == 0)
```

```
252                 cout << id << " ";
253             }
254 
255             cout << endl;
256         }
257 
258 
259         bool isLeafNode(int nodeid){
260 
261             if (arrayOfBTNodes[nodeid].getLeftChildPtr() == 0 &&
                arrayOfBTNodes[nodeid].getRightChildPtr() == 0)
262                 return true;
263 
264             return false;
265         }
266 
267         int getNodeHeight(int nodeid){
268 
269             if (nodeid == -1)
270                 return -1;
271 
272             if (isLeafNode(nodeid) )
273                 return 0;
274 
275             int leftChildID = arrayOfBTNodes[nodeid].getLeftChildID(); // -1 if not exist
276             int rightChildID = arrayOfBTNodes[nodeid].getRightChildID(); // -1 if not
                exist
277 
278             return max(getNodeHeight(leftChildID), getNodeHeight(rightChildID)) + 1;
279 
280         }
281 
282 
283         int getTreeHeight(){
284             return getNodeHeight(0);
285         }
286 
287 
288 
289         void assignLevelNumbers(){
290 
291             Queue queue;
292             queue.enqueue(0);
293             arrayOfBTNodes[0].setLevelNum(0);
294 
295             while (!queue.isEmpty()){
296 
297                 int firstNodeInQueue = queue.dequeue();
298 
299                 int leftChildID = arrayOfBTNodes[firstNodeInQueue].getLeftChildID();
300                 if (leftChildID != -1){
301                     queue.enqueue(leftChildID);
302 
                        arrayOfBTNodes[leftChildID].setLevelNum(arrayOfBTNodes[firstNodeInQue
                        ue].getLevelNum()+1);
303                 }
304 
305                 int rightChildID = arrayOfBTNodes[firstNodeInQueue].getRightChildID();
306                 if (rightChildID != -1){
307                     queue.enqueue(rightChildID);
308 
                        arrayOfBTNodes[rightChildID].setLevelNum(arrayOfBTNodes[firstNodeInQu
                        eue].getLevelNum()+1);
309                 }
```

```
310
311             }
312
313
314         }
315
316         int getDepth(int nodeid){
317             return arrayOfBTNodes[nodeid].getLevelNum();
318         }
319
320 };
321
322
323
324 int main(){
325
326     string filename;
327     cout << "Enter a file name: ";
328     cin >> filename;
329
330     int numNodes;
331     cout << "Enter number of nodes: ";
332     cin >> numNodes;
333
334     BinaryTree binaryTree(numNodes);
335
336     ifstream fileReader(filename);
337
338     if (!fileReader){
339         cout << "File cannot be opened!! ";
340         return 0;
341     }
342
343     int numCharsPerLine = 10;
344
345     char *line = new char[numCharsPerLine];
346     // '10' is the maximum number of characters per line
347
348     fileReader.getline(line, numCharsPerLine, '\n');
349     // '\n' is the delimiting character to stop reading the line
350
351     while (fileReader){
352
353         char* cptr = strtok(line, ",: ");
354
355         string upstreamNodeToken(cptr);
356         int upstreamNodeID = stoi(upstreamNodeToken);
357
358         cptr = strtok(NULL, ",: ");
359
360         int childIndex = 0; // 0 for left child; 1 for right child
361
362         while (cptr != 0){
363
364             string downstreamNodeToken(cptr);
365             int downstreamNodeID = stoi(downstreamNodeToken);
366
367             if (childIndex == 0 && downstreamNodeID != -1)
368                 binaryTree.setLeftLink(upstreamNodeID, downstreamNodeID);
369
370             if (childIndex == 1 && downstreamNodeID != -1)
371                 binaryTree.setRightLink(upstreamNodeID, downstreamNodeID);
372
373             cptr = strtok(NULL, ",: ");
```

```
374                 childIndex++;
375             }
376
377             fileReader.getline(line, numCharsPerLine, '\n');
378
379         }
380
381
382         binaryTree.assignLevelNumbers();
383
384         for (int id = 0; id < numNodes; id++)
385             cout << "Depth of Node " << id << " : " << binaryTree.getDepth(id) << endl;
386
387
388
389
390
391
392         return 0;
393     }
```

```
Enter a file name: binaryTreeFile_1.txt
Enter number of nodes: 10
Depth of Node 0 : 0
Depth of Node 1 : 1
Depth of Node 2 : 1
Depth of Node 3 : 2
Depth of Node 4 : 2
Depth of Node 5 : 2
Depth of Node 6 : 3
Depth of Node 7 : 3
Depth of Node 8 : 3
Depth of Node 9 : 4
```