Ryan Ellis
COSC 320 --- 001 Lab 3 2/20/2025

Lab report: In this lab, we were given header files for the tnode class structure, and from those files two functions were constructed to be implemented and tested. The countOneChild function was written and implemented to traverse a given binary tree and count the number of nodes that have just one child. The max function was written and implemented to traverse a given binary tree and return the max value in the tree. This lab took me about 3 hours to complete cumulatively.

Pre-lab:

1. Review what we learned about tree and binary tree.
2. Read and understand d_tnode.h
3. Read d_tnode1.h and understand functions: buildTree, depth, deleteTree, clearTree, countLeaf

Lab 3.1

Exercise 1:

Header File

```
//
============================================================================
================================================
// Name        : countOneChild.h
// Author      : Ryan C. Ellis
// Creation Date: 2/18/2025
// Last Update: 2/20/2025
// Description: Header file that includes the tnode class structure to build Binary tree, and
countOneChild function to
// traverse tree and count nodes that have 1 child.
//
============================================================================
================================================

#ifndef COUNTONECHILD_H
#define COUNTONECHILD_H

#include <iostream>
#include "d_tnode.h"


using namespace std;

template <class T>
void countOneChild(tnode<T> *, int&);   //prototypes to count and print
template <class T>
void printTree(tnode<T> *, int, int);

#endif
```

```
//
=============================================================================
===
// Description: This function will traverse a given binary tree, with the root
// and variable to count by reference as parameters. No return values, just
// updating the variable passed to reflect nodes that have 1 child.
//
=============================================================================
===
template <class T>
void countOneChild(tnode<T> *nodeptr,int &nodeCount){

   if(nodeptr != NULL){
      if((nodeptr->left == NULL || nodeptr->right == NULL) && (nodeptr->left || nodeptr->right))
         nodeCount++;
      countOneChild(nodeptr->left,nodeCount );
      countOneChild(nodeptr->right, nodeCount);
   }


}

//=============================================================
// Description : This function recursively prints the tree contents to the
// console using a reverse inorder traversal with indenting.
//=============================================================
template <class T>
void printTree(tnode<T> *t, int Indent, int Level) {
  if (t != nullptr) {
    printTree(t->right, Indent, Level + 1);
    for(int i = 0; i <(Indent * Level); i++)
      cout<< " ";
    cout << t->nodeValue << "\n";
    printTree(t->left, Indent, Level + 1);
  }
}
```

Implementation File (main program)

```
//
=============================================================================
==============================================
// Name        : Lab03_01.cpp
// Author      : Ryan C. Ellis
// Creation Date: 2/18/2025
// Last Update: 2/20/2025
// Description: Main function to build a Binary tree and test the countOneChild function.
```

```
//
================================================================================
================================================

#include <iostream>
#include <cstddef>
#include "countOneChild.h"
#include "d_tnode.h"


using namespace std;

tnode<char>* buildTree();       //prototypes to build tree and print div line
void div();

int main(){

   tnode<char> *root;      //declare root of tree and counter variable for 1 child
   int nodeCount = 0;

   root = buildTree();         //set root to built tree

   countOneChild(root, nodeCount );    //call countOneChild on root

   printTree(root, 4, 0);          //print binary tree from left side orientation
   div();

  cout<<"There are "<<nodeCount<< " nodes with 1 child."<<endl;        //output findings


   return 0;
}
void div(){
   cout<<"\n====================================="<<endl;      //div line function
}
tnode<char>* buildTree(){
   tnode<char> *root, *b, *c, *d, *e, *f, *g;      //build tree function built from bottom up

   g = new tnode<char>('G');
   f = new tnode<char>('F');
   e = new tnode<char>('E', (tnode<char>*)NULL, g);
   d = new tnode<char>('D');
   c = new tnode<char>('C', e, f);
   b = new tnode<char>('B', d, (tnode<char>*)NULL);
   root = new tnode<char>('A', b, c);

   return root;                        //return root node
}
```

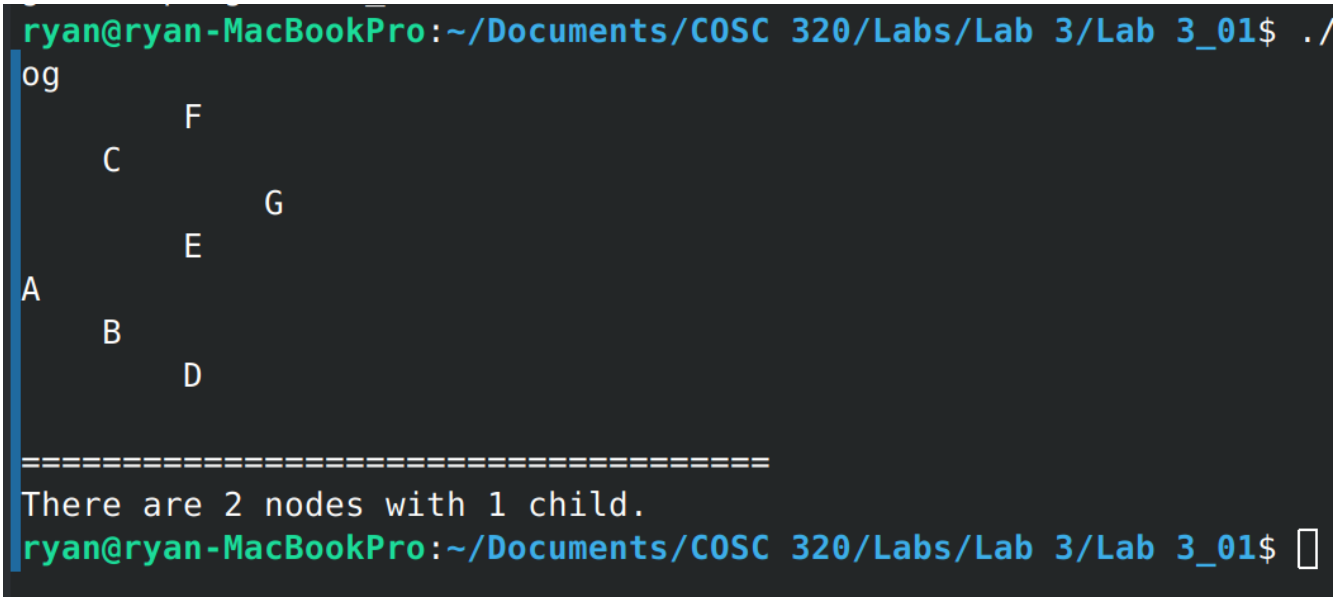
Output:

```
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 3/Lab 3_01$ ./p
og
        F
            G
    C
            G
        E
A
    B
        D

====================================
There are 3 nodes with 1 child.
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 3/Lab 3_01$
```

```
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 3/Lab 3_01$ ./p
og
        F
    C
            G
        E
A
    B
        D

====================================
There are 2 nodes with 1 child.
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 3/Lab 3_01$
```



Lab 3.2

Exercise 2:

Header File

```
//
//============================================================================
//============================================
// Name        : max.h
// Author      : Ryan C. Ellis
// Creation Date: 2/20/2025
// Last Update:
// Description: Header file that includes the tnode class structure to build
//Binary tree, and max function to traverse tree and return max value
//
//============================================================================
//============================================

#ifndef MAX_H
#define MAX_H

#include "d_tnode.h"
#include <iostream>

using namespace std;

template <class T>
T max(tnode<T> *);              //prototypes to find max and print
template <class T>
void printTree(tnode<T> *, int, int);
```

```
#endif

//
//=========================================================================
===
// Description: This function will traverse a given binary tree, with the root
// as the parameter, and return the max value in the tree.
//
//=========================================================================
===
template <class T>
T max(tnode<T> *nodeptr){
   T maxVal = nodeptr->nodeValue;
   if(nodeptr != NULL){

      //************************
      //first attempt returns seg fault

      // if(nodeptr->nodeValue < nodeptr->right->nodeValue)
      //     maxVal = nodeptr->right->nodeValue;
      // if(nodeptr->nodeValue <nodeptr->left->nodeValue)
      //     maxVal = nodeptr->left->nodeValue;
      // max(nodeptr->left);
      // max(nodeptr->right);

      //************************

      if(nodeptr->right != NULL){     //check right side, initialize right max calling function to right
      T maxR = max(nodeptr->right);   //set right max value to max if greater
         if(maxR >maxVal)
            maxVal = maxR;
      }

      if(nodeptr->left != NULL){      //check left side, initialize left max calling function to left
         T maxL = max(nodeptr->left);    //set left max value to max if greater
         if(maxL >maxVal)
            maxVal = maxL;
      }

   return maxVal;          //return max value

   }

   return T();     //return default T value

}
//=============================================================
// Description : This function recursively prints the tree contents to the
```

```
// console using a reverse inorder traversal with indenting.
//============================================================
template <class T>
void printTree(tnode<T> *t, int Indent, int Level) {
  if (t != nullptr) {
    printTree(t->right, Indent, Level + 1);
    for(int i = 0; i <(Indent * Level); i++)
      cout<< " ";
    cout << t->nodeValue << "\n";
    printTree(t->left, Indent, Level + 1);
  }
}
```

Implementation File (main program)

```
//
============================================================================
================================================
// Name        : Lab03_02.cpp
// Author      : Ryan C. Ellis
// Creation Date: 2/20/2025
// Last Update:
// Description: Main function to build a Binary tree and test the max function.
//
============================================================================
================================================

#include <iostream>
#include <cstddef>
#include "max.h"
#include "d_tnode.h"

using namespace std;

tnode<int>* buildIntTree();     //prototypes to find max and print div line
void div();

int main(){
    tnode<int> *root;       //declare root

    root = buildIntTree();      //set root to built int tree

    printTree(root,4 ,0 );      //print tree
    cout<<"Searching for max value ...";
    div();
    cout<<"Max value: "<<max(root)<<endl;   //find max and output results

    return 0;
}
void div(){         //div line
```

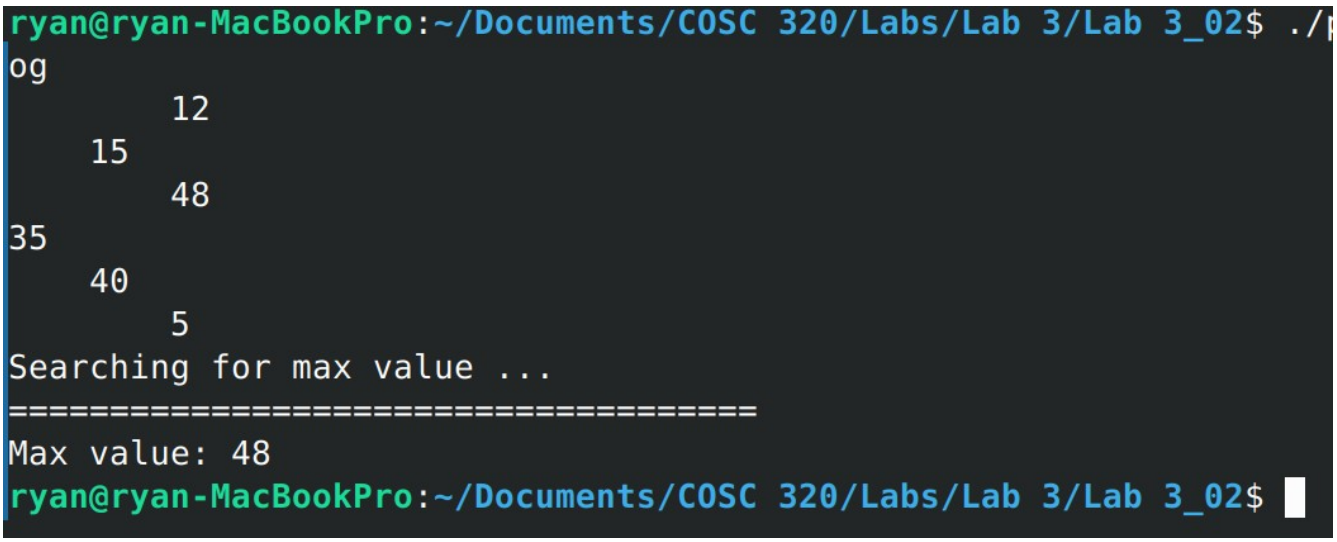
```
    cout<<"\n====================================="<<endl;
}
tnode<int>* buildIntTree(){
    tnode<int> *root, *b, *c, *d, *e, *f;      //build tree function built from bottom up


    f = new tnode<int>(12, (tnode<int>*)NULL, (tnode<int>*)NULL);
    e = new tnode<int>(48, (tnode<int>*)NULL, (tnode<int>*)NULL);
    d = new tnode<int>(5);
    c = new tnode<int>(15, e, f);
    b = new tnode<int>(40, d, (tnode<int>*)NULL);
    root = new tnode<int>(35, b, c);

    return root;                        //return root node
}
```

Output:

```
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 3/Lab 3_02$ mak
g++ -g -Wall -std=c++11 -c Lab03_02.cpp
g++ -o prog Lab03_02.o
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 3/Lab 3_02$ ./p
og
        72
    15
        48
35
    40
        5
Searching for max value ...
=====================================
Max value: 72
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 3/Lab 3_02$
```


ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 3/Lab 3_02$ ./p
og
12
15
48
35
40
5
Searching for max value ...
==========
Max value: 48
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 3/Lab 3_02$