```cpp
/*********************************************************************
Name: Kevin Davis
Date: Sep-17-2019
Class: CMPS 3013

Description: This program uses an algorithm that can read a file of random
numbers and insert them into a tree in such a manner that it will end
up being full.

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

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <algorithm>


//http://www.webgraphviz.com/

using namespace std;

struct node
{
    int data;
    node *left;
    node *right;
    node()
    {
        data = -1;
        left = NULL;
        right = NULL;
    }
    node(int x)
    {
        data = x;
        left = NULL;
        right = NULL;
    }
};

class BSTree
{
private:
    node * root;

    int count(node *root)
    {
        if (!root)
        {
            return 0;
        }
```

```
53          else
54          {
55              return 1 + count(root->left) + count(root->right);
56          }
57      }
58
59      void insert(node *&root, node *&temp)
60      {
61          if (!root)
62          {
63              root = temp;
64          }
65          else
66          {
67              if (temp->data < root->data)
68              {
69                  insert(root->left, temp);
70              }
71              else
72              {
73                  insert(root->right, temp);
74              }
75          }
76      }
77
78      void print_node(node *n, string label = "")
79      {
80          if (label != "")
81          {
82              cout << "[" << label << "]";
83          }
84          cout << "[[" << n << "][" << n->data << "]]\n";
85          if (n->left)
86          {
87              cout << "\t|-->[L][[" << n->left << "][" << n->left->data << "]]\n";
88          }
89          else
90          {
91              cout << "\t\\-->[L][null]\n";
92          }
93          if (n->right)
94          {
95              cout << "\t\\-->[R][[" << n->right << "][" << n->right->data << "]]
                  \n";
96          }
97          else
98          {
99              cout << "\t\\-->[R][null]\n";
100         }
101     }
102
103     /**
```

```
104     * type = ['predecessor','successor']
105     */
106     node *minValueNode(node *root)
107     {
108         node *current = root;
109
110         if (root->right)
111         {
112             current = root->right;
113             while (current->left != NULL)
114             {
115                 current = current->left;
116             }
117         }
118         else if (root->left)
119         {
120             current = root->left;
121             while (current->right != NULL)
122             {
123                 current = current->right;
124             }
125         }
126
127         return current;
128     }
129
130     int height(node *root)
131     {
132         if (!root)
133         {
134             return 0;
135         }
136         else
137         {
138             int left = height(root->left);
139             int right = height(root->right);
140             if (left > right)
141             {
142                 return left + 1;
143             }
144             else
145             {
146                 return right + 1;
147             }
148         }
149     }
150
151     /* Print nodes at a given level */
152     void printGivenLevel(node *root, int level)
153     {
154         if (root == NULL)
155             return;
```

```
156         if (level == 1)
157         {
158             print_node(root);
159         }
160         else if (level > 1)
161         {
162             printGivenLevel(root->left, level - 1);
163             printGivenLevel(root->right, level - 1);
164         }
165     }
166     //************************************************************************
167     // Method to help create GraphViz code so the expression tree can
168     // be visualized. This method prints out all the unique node id's
169     // by traversing the tree.
170     // Recivies a node pointer to root and performs a simple recursive
171     // tree traversal.
172     //************************************************************************
173     void GraphVizGetIds(node *nodePtr, ofstream &VizOut)
174     {
175         static int NullCount = 0;
176         if (nodePtr)
177         {
178             GraphVizGetIds(nodePtr->left, VizOut);
179             VizOut << "node" << nodePtr->data
180                 << "[label=\"" << nodePtr->data << "\\n"
181                 //<<"Add:"<<nodePtr<<"\\n"
182                 //<<"Par:"<<nodePtr->parent<<"\\n"
183                 //<<"Rt:"<<nodePtr->right<<"\\n"
184                 //<<"Lt:"<<nodePtr->left<<"\\n"
185                 << "\"]" << endl;
186             if (!nodePtr->left)
187             {
188                 NullCount++;
189                 VizOut << "nnode" << NullCount << "[label=\"X
                    \",shape=point,width=.15]" << endl;
190             }
191             GraphVizGetIds(nodePtr->right, VizOut);
192             if (!nodePtr->right)
193             {
194                 NullCount++;
195                 VizOut << "nnode" << NullCount << "[label=\"X
                    \",shape=point,width=.15]" << endl;
196             }
197         }
198     }
199
200     //************************************************************************
201     // This method is partnered with the above method, but on this pass it
202     // writes out the actual data from each node.
203     // Don't worry about what this method and the above method do, just
204     // use the output as your told:)
205     //************************************************************************
```

```
206     void GraphVizMakeConnections(node *nodePtr, ofstream &VizOut)
207     {
208         static int NullCount = 0;
209         if (nodePtr)
210         {
211             GraphVizMakeConnections(nodePtr->left, VizOut);
212             if (nodePtr->left)
213                 VizOut << "node" << nodePtr->data << "->"
214                 << "node" << nodePtr->left->data << endl;
215             else
216             {
217                 NullCount++;
218                 VizOut << "node" << nodePtr->data << "->"
219                     << "nnode" << NullCount << endl;
220             }
221
222             if (nodePtr->right)
223                 VizOut << "node" << nodePtr->data << "->"
224                 << "node" << nodePtr->right->data << endl;
225             else
226             {
227                 NullCount++;
228                 VizOut << "node" << nodePtr->data << "->"
229                     << "nnode" << NullCount << endl;
230             }
231
232             GraphVizMakeConnections(nodePtr->right, VizOut);
233         }
234     }
235
236 public:
237     BSTree()
238     {
239         root = NULL;
240     }
241     ~BSTree()
242     {
243     }
244
245     int count()
246     {
247         return count(root);
248     }
249
250     void insert(int x)
251     {
252         node *temp = new node(x);
253         insert(root, temp);
254     }
255
256     void minValue()
257     {
```

```
258         print_node(minValueNode(root), "minVal");
259     }
260
261     int height()
262     {
263
264         return height(root);
265     }
266
267     int top()
268     {
269         if (root)
270             return root->data;
271         else
272             return 0;
273     }
274
275     /* Function to line by line print level order traversal a tree*/
276     void printLevelOrder()
277     {
278         cout << "Begin Level Order==================\n";
279         int h = height(root);
280         int i;
281         for (i = 1; i <= h; i++)
282         {
283             printGivenLevel(root, i);
284             cout << "\n";
285         }
286         cout << "End Level Order==================\n";
287     }
288
289     //*********************************************************************
290     // Recieves a filename to place the GraphViz data into.
291     // It then calls the above two graphviz methods to create a data file
292     // that can be used to visualize your expression tree.
293     //*********************************************************************
294     void GraphVizOut(string filename)
295     {
296         ofstream VizOut;
297         VizOut.open(filename);
298         VizOut << "Digraph G {\n";
299         GraphVizGetIds(root, VizOut);
300         GraphVizMakeConnections(root, VizOut);
301         VizOut << "}\n";
302         VizOut.close();
303     }
304 };
305 //function header
306 void split(vector<int> &vec, BSTree &tree, int low, int high);
307
308 int main()
309 {
```

```
310     ifstream Data;
311     vector<int> holder;
312     int temp, size;
313     BSTree B;
314
315     Data.open("input.dat");
316     //while there are files to be read insert them into the vector
317     while (Data >> temp)
318     {
319         holder.push_back(temp);
320     }
321     //sort the vector
322     sort(holder.begin(), holder.end());
323     size = holder.size();
324     //test printout after sorting
325     for (int i = 0; i<size; i++)
326     {
327         cout << holder[i] << " ";
328     }
329     cout << endl;
330
331     split(holder, B, 0, size - 1);
332
333     B.printLevelOrder();
334     B.GraphVizOut("before.txt");
335
336     Data.close();
337     return 0;
338 }
339 //use the high and low positions to find the midpoint
340 //after finding the midpoint insert its value
341 //then call split using mid-1 as the high (to  go left)
342 //and call split again using mid+1 as the low (to go right)
343
344 void split(vector<int> &vec, BSTree &tree, int low, int high)
345 {
346     if (low <= high)
347     {
348         int mid = (high + low) / 2;
349         tree.insert(vec[mid]);
350
351         split(vec, tree, low, mid - 1);
352         split(vec, tree, mid + 1, high);
353     }
354 }
355
```