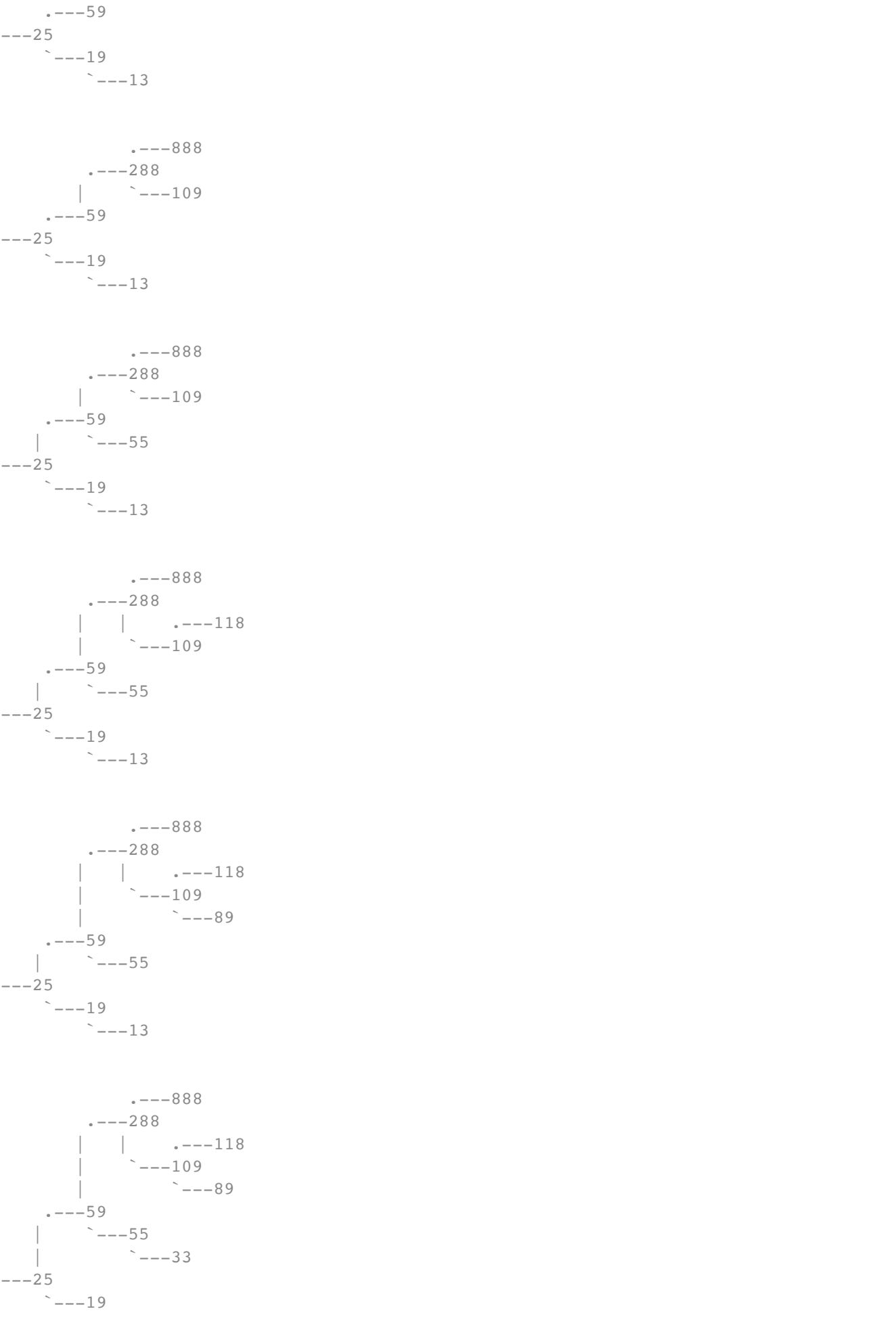
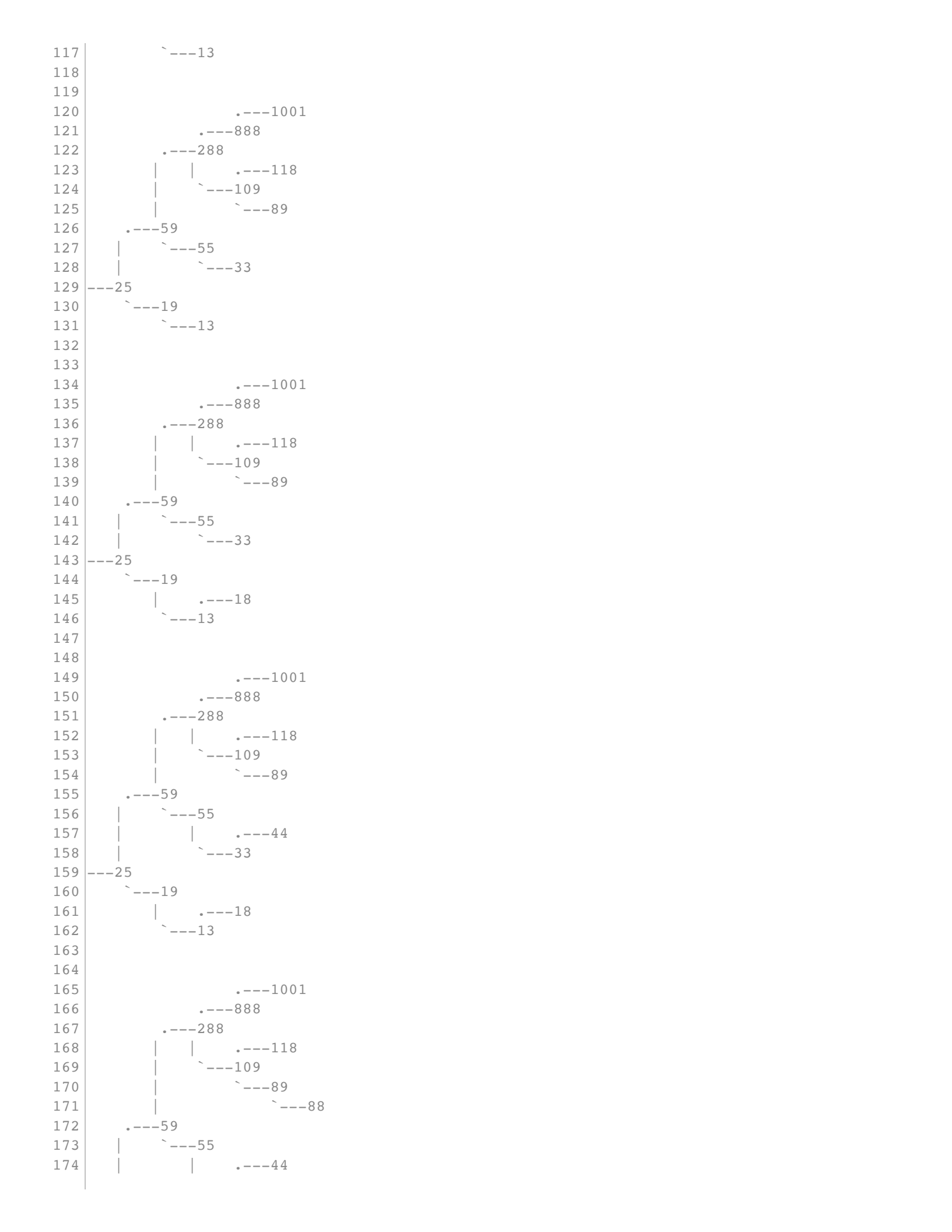
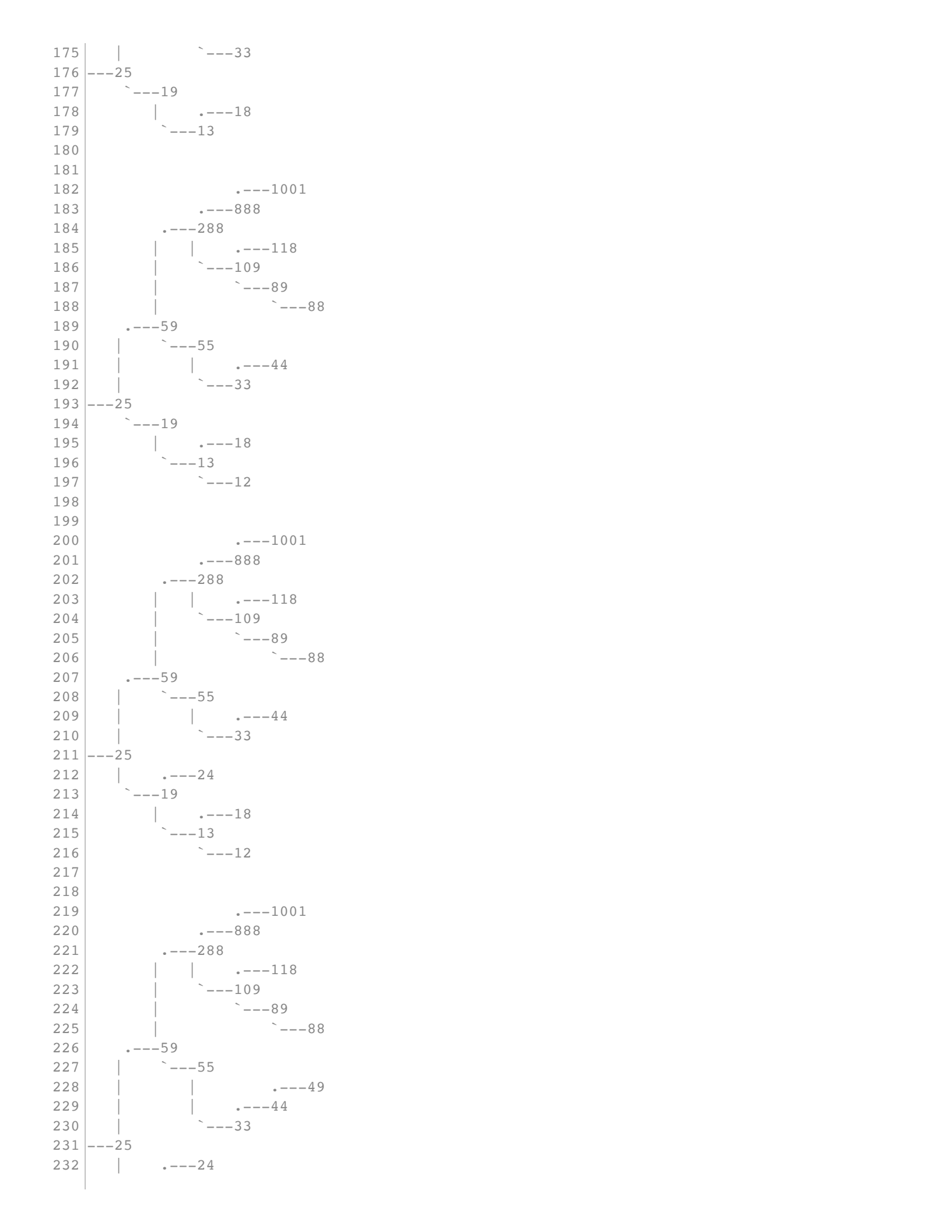
```
*********************************************************************************
* PROGRAMMED BY : Nick Reardon
* CLASS         : CS1D
* SECTION       : MW - 2:30p
* Assignment #5 : Binary Trees
*********************************************************************************

                          Assignment #5 - Binary Trees

Implement a binary tree using an array, vector or linked list.
(Note: duplicates are allowed in a binary tree)

Store the following elements using the properties of a binary
search tree.

25, 59, 288, 19, 13, 888, 109, 55, 118, 89, 33, 1001, 18, 44, 88,
12, 24, 49, 9,

Perform the in-order, post-order, pre-order, breadth-first
traversals.

In addition to the traversals, print out the binary tree by
level. Show the parent-child relationship for all the nodes of
the tree.


Due February 19th


*********************************************************************************

---25


    .---59
---25


        .---288
    .---59
---25


        .---288
    .---59
---25
    `---19


        .---288
    .---59
---25
    `---19
        `---13


            .---888
        .---288
```


.---59
---25
`---19
`---13
.---888
.---288
|
`---109
.---59
---25
`---19
`---13
.---888
.---288
|
`---109
.---59
|
`---55
---25
`---19
`---13
.---888
.---288
|
|
.---118
|
`---109
.---59
|
`---55
---25
`---19
`---13
.---888
.---288
|
|
.---118
|
`---109
|
`---89
.---59
|
`---55
---25
`---19
`---13
.---888
.---288
|
|
.---118
|
`---109
|
`---89
.---59
|
`---55
|
`---33
---25
`---19


`---13
.---1001
.---888
.---288
| | .---118
| `---109
| `---89
.---59
| `---55
| `---33
---25
`---19
`---13
.---1001
.---888
.---288
| | .---118
| `---109
| `---89
.---59
| `---55
| `---33
---25
`---19
| .---18
`---13
.---1001
.---888
.---288
| | .---118
| `---109
| `---89
.---59
| `---55
| | .---44
| `---33
---25
`---19
| .---18
`---13
.---1001
.---888
.---288
| | .---118
| `---109
| `---89
| `---88
.---59
| `---55
| | .---44


`---33
---25
`---19
.---18
`---13
.---1001
.---888
.---288
.---118
`---109
`---89
`---88
.---59
`---55
.---44
`---33
---25
`---19
.---18
`---13
`---12
.---1001
.---888
.---288
.---118
`---109
`---89
`---88
.---59
`---55
.---44
`---33
---25
.---24
`---19
.---18
`---13
`---12
.---1001
.---888
.---288
.---118
`---109
`---89
`---88
.---59
`---55
.---49
.---44
`---33
---25
.---24

```
    `---19
       |    .---18
       `---13
            `---12


                .---1001
            .---888
        .---288
       |   |    .---118
       |    `---109
       |        `---89
       |            `---88
    .---59
   |    `---55
   |        |        .---49
   |        |    .---44
   |        `---33
---25
   |    .---24
    `---19
       |    .---18
       `---13
            `---12
                `---9


                .---1001
            .---888
        .---288
       |   |    .---118
       |    `---109
       |        `---89
       |            `---88
    .---59
   |    `---55
   |        |        .---49
   |        |    .---44
   |        `---33
---25
   |    .---24
    `---19
       |    .---18
       `---13
            `---12
                `---9


Level By Level:
25
19  59
13  24  55  288
12  18  33  109  888
9  44  89  118  1001
49  88

In Order Traversal:
9  12  13  18  19  24  25  33  44  49  55  59  88  89  109  118  288  888  1001
```

```
291
292 Post Order Traversal:
293 9  12  18  13  24  19  49  44  33  55  88  89  118  109  1001  888  288  59  25
294
295 Pre Order Traversal:
296 25  19  13  12  9  18  24  59  55  33  44  49  288  109  89  88  118  888  1001
297
298 Breadth First Traversal:
299 25  19  59  13  24  55  288  12  18  33  109  888  9  44  89  118  1001  49  88
300
301 sh: pause: command not found
302 Process exited with status 0
303 logout
304 Saving session...
305 ...copying shared history...
306 ...saving history...truncating history files...
307 ...completed.
308
309 [Process completed]
310
311
```

```cpp
/*****************************************************************************
 * AUTHOR           : Nick Reardon
 * Assignment #5    : Binary Trees
 * CLASS            : CS1D
 * SECTION          : MW - 2:30p
 * DUE DATE         : 02 / 19 / 20
 *****************************************************************************/
#include "main.h"

using std::cout; using std::endl;
#include <stdio.h>

int main()
{

    /*
     * HEADER OUTPUT
     */
    PrintHeader(cout, "Prompt.txt");

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

    LinkedBinaryTree<int> bTree;

    std::ifstream iFile;
    iFile.open("Input.txt");


    int temp;
    while (iFile >> temp)
    {
        bTree.insert(temp);

        bTree.printTree(cout);

    }
    iFile.close();



    bTree.printTree(cout);

    bTree.PrintLevelByLevel(cout);

    bTree.Traversal_InOrder(cout);

    bTree.Traversal_PostOrder(cout);

    bTree.Traversal_PreOrder(cout);

    bTree.Traversal_BreadthFirst(cout);


    system("pause");
    return 0;
}

```

```
/*****************************************************************************
 * AUTHOR           : Nick Reardon
 * Assignment #5    : Binary Trees
 * CLASS            : CS1D
 * SECTION          : MW - 2:30p
 * DUE DATE         : 02 / 19 / 20
 *****************************************************************************/
#ifndef _MAIN_H_
#define _MAIN_H_

//Standard includes
#include <iostream>
#include <iomanip>
#include <string>
#include "PrintHeader.h"

//Program Specific
#include "LinkedBinaryTree.h"

#endif // _HEADER_H_


```

```
/*****************************************************************************
 * AUTHOR           : Nick Reardon
 * Assignment #5    : Binary Trees
 * CLASS            : CS1D
 * SECTION          : MW - 2:30p
 * DUE DATE         : 02 / 19 / 20
 *****************************************************************************/
#ifndef _LINKEDBINARYTREE_H_
#define _LINKEDBINARYTREE_H_
#include <exception>
#include <sstream>
#include <string>
#include <queue>
#include "Except.h"

enum ERROR_TYPE
{
    DEFUALT,
    FULL,
    EMPTY
};

struct Trunk
{
    Trunk* prev;
    std::string str;

    Trunk(Trunk* prev, std::string str)
    {
        this->prev = prev;
        this->str = str;
    }
};

template <class Type>
struct Node
{
    Type value;

    Node<Type>* parent;

    Node<Type>* left;
    Node<Type>* right;

    Node<Type>(const Type& newValue, Node<Type>* parentNode, Node<Type>* leftNode,
Node<Type>* rightNode)
    {
        value = newValue;

        parent = parentNode;

        left = leftNode;
        right = rightNode;
    }

    Node<Type>(const Type& newValue, Node<Type>* leftNode, Node<Type>* rightNode)
    {
        value = newValue;
```

```
58
59          left = leftNode;
60          right = rightNode;
61      }
62
63  };
64
65  template <class Type>
66  class LinkedBinaryTree
67  {
68  private:
69      Node<Type>* root;
70
71      int capacity;
72      int currentSize;
73
74
75  protected:
76      void insertRecursion(const Type& newValue, Node<Type>* node)
77      {
78          if (newValue == node->value)
79          {
80              Node<Type>* tempPtr = new Node<Type>(newValue, node, node->left, nullptr);
81
82              if (node->left != nullptr)
83              {
84                  node->left->parent = tempPtr;
85              }
86
87              node->left = tempPtr;
88
89              tempPtr = nullptr;
90
91          }
92          else if (newValue > node->value)
93          {
94              if (node->right == nullptr)
95              {
96                  node->right = new Node<Type>(newValue, node, nullptr, nullptr);
97              }
98              else
99              {
100                 insertRecursion(newValue, node->right);
101             }
102         }
103         else
104         {
105             if (node->left == nullptr)
106             {
107                 node->left = new Node<Type>(newValue, node, nullptr, nullptr);
108             }
109             else
110             {
111                 insertRecursion(newValue, node->left);
112             }
113         }
114     }
115
```

```
116     void destroyRecursion(Node<Type>* node)
117     {
118         if (node != nullptr)
119         {
120             destroyRecursion(node->left);
121             destroyRecursion(node->right);
122             delete node;
123         }
124     }
125 
126     Type* searchRecursion(const Type& searchValue, const Node<Type>* node) const
127     {
128         if (searchValue == node->value)
129         {
130             return node;
131         }
132         else
133         {
134             if (searchValue > node->value)
135             {
136                 if (searchValue == node->right->value)
137                 {
138                     return node->right;
139                 }
140                 else
141                 {
142                     searchRecursion(searchValue, node->right);
143                 }
144             }
145             else
146             {
147                 if (searchValue == node->left->value)
148                 {
149                     return node->left;
150                 }
151                 else
152                 {
153                     searchRecursion(searchValue, node->left);
154                 }
155             }
156         }
157     }
158 
159     void InOrder_Recursion(const Node<Type>* node, std::ostream& output) const
160     {
161         if (root == nullptr)
162         {
163             throw(Except_runtime_error_class("Tree is empty - Nothing to print", EMPTY,
    5));
164         }
165         else if(node != nullptr)
166         {
167 
168             InOrder_Recursion(node->left, output);
169 
170             output << node->value << "   ";
171 
172             InOrder_Recursion(node->right, output);
173 
```

```
174         }
175     }
176 
177     void PostOrder_Recursion(const Node<Type>* node, std::ostream& output) const
178     {
179         if (root == nullptr)
180         {
181             throw(Except_runtime_error_class("Tree is empty - Nothing to print", EMPTY,
    5));
182         }
183         if (node != nullptr)
184         {
185             PostOrder_Recursion(node->left, output);
186 
187             PostOrder_Recursion(node->right, output);
188 
189             output << node->value << "   ";
190         }
191 
192     }
193 
194     void PreOrder_Recursion(const Node<Type>* node, std::ostream& output) const
195     {
196         if (root == nullptr)
197         {
198             throw(Except_runtime_error_class("Tree is empty - Nothing to print", EMPTY,
    5));
199         }
200         if (node != nullptr)
201         {
202             output << node->value << "   ";
203 
204             PreOrder_Recursion(node->left, output);
205 
206             PreOrder_Recursion(node->right, output);
207 
208         }
209     }
210 
211     void BreadthFirst_Recursion(std::queue<Node<Type>*>& queue, const Node<Type>* node,
    std::ostream& output) const
212     {
213         if (root == nullptr)
214         {
215             throw(Except_runtime_error_class("Tree is empty - Nothing to print", EMPTY,
    5));
216         }
217         if (node != nullptr)
218         {
219             output << node->value << "   ";
220 
221             if (node->left != nullptr)
222             {
223                 queue.push(node->left);
224             }
225 
226             if (node->right != nullptr)
227             {
228                 queue.push(node->right);
```

```
229             }
230 
231             queue.pop();
232 
233             if (!queue.empty())
234             {
235                 BreadthFirst_Recursion(queue, queue.front(), output);
236             }
237         }
238     }
239 
240 
241 
242     void Print2DUtil(const Node<Type>* branch, int space, std::ostream& output) const
243     {
244         // Base case
245         if (branch == NULL)
246             return;
247 
248         // Increase distance between levels
249         space += 10;
250 
251         // Process right child first
252         Print2DUtil(branch->right, space, output);
253 
254         // Print current node after space
255         // count
256         output << '\n';
257         for (int i = 10; i < space; i++)
258         {
259             output << " ";
260         }
261 
262         output << branch->value << "\n";
263 
264         // Process left child
265         Print2DUtil(branch->left, space, output);
266     }
267 
268     // Helper function to print branches of the binary tree
269     void showTrunks(Trunk* p, std::ostream& output)
270     {
271         if (p == nullptr)
272             return;
273 
274         showTrunks(p->prev, output);
275 
276         output << p->str;
277     }
278 
279     // Recursive function to print binary tree
280     // It uses inorder traversal
281     void printTreeHelper(Node<Type>* node, Trunk* prev, bool isLeft, std::ostream& output)
282     {
283         if (node == nullptr)
284             return;
285 
286         std::string prev_str = "    ";
```

```
287         Trunk* trunk = new Trunk(prev, prev_str);
288 
289         printTreeHelper(node->right, trunk, true, output);
290 
291         if (!prev)
292             trunk->str = "---";
293         else if (isLeft)
294         {
295             trunk->str = ".---";
296             prev_str = "    |";
297         }
298         else
299         {
300             trunk->str = "`---";
301             prev->str = prev_str;
302         }
303 
304         showTrunks(trunk, output);
305         output << node->value << '\n';
306 
307         if (prev)
308             prev->str = prev_str;
309         trunk->str = "    |";
310 
311         printTreeHelper(node->left, trunk, false, output);
312     }
313 
314 
315 public:
316 
317     LinkedBinaryTree<Type>()
318     {
319         root = nullptr;
320         currentSize = 0;
321     }
322 
323     //LinkedBinaryTree<Type>(const LinkedBinaryTree<Type>& otherTree);
324 
325     ~LinkedBinaryTree()
326     {
327         destroy();
328     }
329 
330     void destroy()
331     {
332         destroyRecursion(root);
333         root = nullptr;
334         currentSize = 0;
335     }
336 
337     //bool empty() const;
338     //bool full() const;
339     //int size() const;
340 
341     void insert(const Type& newValue)
342     {
343         if (root == nullptr)
344         {
```

```
345             root = new Node<Type>(newValue, nullptr, nullptr, nullptr);
346         }
347         else
348         {
349             insertRecursion(newValue, root);
350         }
351 
352         currentSize++;
353     }
354 
355     Type* search(const Type& searchValue) const
356     {
357         search(searchValue, root);
358     }
359 
360     void printAll(std::ostream& output) const;
361 
362     void Traversal_InOrder(std::ostream& output) const
363     {
364         output << "In Order Traversal:" << '\n';
365         InOrder_Recursion(root, output);
366         output << "\n\n";
367     }
368 
369 
370     void Traversal_PostOrder(std::ostream& output) const
371     {
372         output << "Post Order Traversal:" << '\n';
373         PostOrder_Recursion(root, output);
374         output << "\n\n";
375     }
376 
377     void Traversal_PreOrder(std::ostream& output) const
378     {
379         output << "Pre Order Traversal:" << '\n';
380         PreOrder_Recursion(root, output);
381         output << "\n\n";
382     }
383 
384     void Traversal_BreadthFirst(std::ostream& output) const
385     {
386         std::queue<Node<Type>*> queue;
387 
388         queue.push(root);
389 
390         output << "Breadth First Traversal:" << '\n';
391         BreadthFirst_Recursion(queue, root, output);
392         output << "\n\n";
393     }
394 
395     void Print2D(std::ostream& output) const
396     {
397         Print2DUtil(root, 0, output);
398     }
399 
400 
401 
402     void printTree(std::ostream& output) const
```

```
403     {
404         printTreeHelper(root, nullptr, false, output);
405         output << "\n\n";
406     }
407 
408     void PrintLevelByLevel(std::ostream& output) const
409     {
410         if (root == nullptr)
411         {
412             throw(Except_runtime_error_class("Tree is empty - Nothing to print", EMPTY,
    5));
413         }
414 
415         output << "Level By Level:" << '\n';
416 
417         std::queue<Node<Type>*> queue;
418 
419         int levelNodes = 0;
420 
421         queue.push(root);
422 
423         while (!queue.empty())
424 {
425 
426             levelNodes = queue.size();
427 
428             while (levelNodes > 0)
429             {
430                 Node<Type>* temp = queue.front();
431 
432                 output << temp->value << "   ";
433 
434                 if (temp->left != nullptr)
435                 {
436                     queue.push(temp->left);
437                 }
438 
439                 if (temp->right != nullptr)
440                 {
441                     queue.push(temp->right);
442                 }
443 
444                 queue.pop();
445 
446                 levelNodes--;
447             }
448             output << '\n';
449         }
450 
451         output << "\n";
452     }
453 
454 
455 
456 };
457 
458 
459 #endif // !_LINKEDBINARYTREE_H_
460 
```

```
/*****************************************************************************
 * AUTHOR           : Nick Reardon
 * Assignment #5    : Binary Trees
 * CLASS            : CS1D
 * SECTION          : MW - 2:30p
 * DUE DATE         : 02 / 19 / 20
 *****************************************************************************/
#ifndef _EXCEPT_H_
#define _EXCEPT_H_

#include <string>
#include <exception>
#include <sstream>

/**
 * @class   Except_runtime_error_class Except_runtime_error_class.h Except_runtime_error_class.h
 *
 * @brief   Generic Exception class with basic output setup
 *
 * @author  Nick Reardon
 * @date    12/09/2020
 */
class Except_runtime_error_class : virtual public std::runtime_error {

protected:

    int error_number;               ///< Error number
    int error_offset;               ///< Error offset

public:

    /**
     * @fn  explicit Except_runtime_error_class::Except_runtime_error_class(const std::string& msg, int err_num, int err_off)
     *      : std::runtime_error(msg)
     * @brief   Constructor
     *
     * @param   msg     The error message.
     * @param   err_num Error number.
     * @param   err_off Error offset.
     */
    explicit
        Except_runtime_error_class(const std::string& msg, int err_num, int err_off) :
        std::runtime_error(msg)
    {
        error_number = err_num;
        error_offset = err_off;

    }

    /** Destructor.
     *  Virtual to allow for subclassing.
     */
    virtual ~Except_runtime_error_class() throw () {}

    /** Returns error number.
     *  @return #error_number
```

```
57      */
58     virtual int getErrorNumber() const throw() {
59         return error_number;
60     }
61 
62     /**Returns error offset.
63      * @return #error_offset
64      */
65     virtual int getErrorOffset() const throw() {
66         return error_offset;
67     }
68 
69     virtual void outputError(std::ostream& output) const throw()
70     {
71         output << "Exception - Error number " << error_number
72             << ":" << std::string(error_offset, ' ') << what() << '\n';
73     }
74 
75 };
76 
77 
78 #endif // !_EXCEPT_H_
```

```
/*****************************************************************************
 * AUTHOR           : Nick Reardon
 * Assignment #5    : Binary Trees
 * CLASS            : CS1D
 * SECTION          : MW - 2:30p
 * DUE DATE         : 02 / 19 / 20
 *****************************************************************************/
#ifndef _PRINTHEADER_H_
#define _PRINTHEADER_H_

#include <iostream>
#include <iomanip>
#include <ostream>
#include <string>
#include <fstream>

/*****************************************************************************
* PrintHeader
* ---------------------------------------------------------------------------
* This function will output a class header through the use of ostream.
* It also will output the program description
* ---------------------------------------------------------------------------
*  Call
* ------
* The function call requires 1 parameters. The following example uses an
* output file in the ostream parameter. Ex:
*
*       PrintHeader (oFile);
*
* ---------------------------------------------------------------------------
*  Output
* --------
*   The function will output as follows. Ex:
*
*       ********************************************************
*       * PROGRAMMED BY : Parsa Khazravi and Nick Reardon
*       * CLASS         : CS1B
*       * SECTION       : MW: 7:30pm
*       * Lab #3        : Functions - GCD
*       ********************************************************
*
* ---------------------------------------------------------------------------
* CONSTANTS
* ---------------------------------------------------------------------------
* OUTPUT - USED FOR CLASS HEADING
* ---------------------------------------------------------------------------
* PROGRAMMER        : Name(s) of programmer(s)  - Nick Reardon
* SECTION           : Class times               - MW - 7:30p
* CLASS             : Class label               - CS1B
* PROGRAM_NUM       : # of the program
* PROGRAM_NAME      : Title of the program
* PROGRAM_TYPE      : Type of program - Lab, Assignment, etc.
*
* ---------------------------------------------------------------------------
* MAX_OUTPUT        : Max movies to be output at once
*****************************************************************************/
const std::string PROGRAMMER = "Nick Reardon";
const std::string SECTION = "MW - 2:30p";
```

```cpp
const std::string CLASS = "CS1D";
const int PROGRAM_NUM = 5;
const std::string PROGRAM_NAME = "Binary Trees";
const std::string PROGRAM_TYPE = "Assignment";


void PrintHeader(std::ostream &output, std::string inputText)
{
    std::string typeNum = PROGRAM_TYPE + " #" + std::to_string(PROGRAM_NUM);

    output << std::left
        << std::string(76, '*')
        << std::endl
        << "* PROGRAMMED BY : " << PROGRAMMER << std::endl
        << "* " << std::setw(14) << "CLASS" << ": " << CLASS << std::endl
        << "* " << std::setw(14) << "SECTION" << ": " << SECTION << std::endl
        << "* " << std::setw(14) << typeNum << ": " << PROGRAM_NAME << std::endl
        << std::string(76, '*')
        << std::endl << std::endl
        << std::string(((76 - typeNum.length() - PROGRAM_NAME.length() ) / 2), ' ')
        << typeNum + " - " + PROGRAM_NAME
        << std::endl << std::endl
        << std::ifstream(inputText).rdbuf()
        << std::endl
        << std::string(76, '*')
        << std::endl << std::endl;

}

#endif //_PRINTHEADER_H_
```