```
****************************************************************************
* 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
```

```

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

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

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

sh: pause: command not found
Process exited with status 0
logout
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]


```

```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;
}
```

```cpp
/******************************************************************************
 * 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;

        left = leftNode;
        right = rightNode;
    }

};

template <class Type>
class LinkedBinaryTree
{
private:
    Node<Type>* root;

    int capacity;
    int currentSize;


protected:
    void insertRecursion(const Type& newValue, Node<Type>* node)
    {
        if (newValue == node->value)
        {
            Node<Type>* tempPtr = new Node<Type>(newValue, node, node->left, nullptr);

            if (node->left != nullptr)
            {
                node->left->parent = tempPtr;
            }

            node->left = tempPtr;

            tempPtr = nullptr;

        }
        else if (newValue > node->value)
        {
            if (node->right == nullptr)
            {
                node->right = new Node<Type>(newValue, node, nullptr, nullptr);
            }
            else
            {
                insertRecursion(newValue, node->right);
            }
        }
        else
        {
            if (node->left == nullptr)
            {
                node->left = new Node<Type>(newValue, node, nullptr, nullptr);
            }
            else
            {
                insertRecursion(newValue, node->left);
            }
        }
    }

    void destroyRecursion(Node<Type>* node)
    {
        if (node != nullptr)
        {
            destroyRecursion(node->left);
            destroyRecursion(node->right);
            delete node;
        }
    }

    Type* searchRecursion(const Type& searchValue, const Node<Type>* node) const
    {
        if (searchValue == node->value)
        {
            return node;
        }
        else
        {
            if (searchValue > node->value)
            {
                if (searchValue == node->right->value)
                {
                    return node->right;
                }
                else
                {
                    searchRecursion(searchValue, node->right);
                }
            }
            else
            {
                if (searchValue == node->left->value)
                {
                    return node->left;
                }
                else
                {
                    searchRecursion(searchValue, node->left);
                }
            }
        }
    }

    void InOrder_Recursion(const Node<Type>* node, std::ostream& output) const
    {
        if (root == nullptr)
        {
            throw(Except_runtime_error_class("Tree is empty - Nothing to print", EMPTY, 5));
        }
        else if(node != nullptr)
        {

            InOrder_Recursion(node->left, output);

            output << node->value << "  ";

            InOrder_Recursion(node->right, output);

```

```cpp
        }
    }

    void PostOrder_Recursion(const Node<Type>* node, std::ostream& output) const
    {
        if (root == nullptr)
        {
            throw(Except_runtime_error_class("Tree is empty - Nothing to print", EMPTY, 5));
        }
        if (node != nullptr)
        {
            PostOrder_Recursion(node->left, output);

            PostOrder_Recursion(node->right, output);

            output << node->value << "  ";
        }

    }

    void PreOrder_Recursion(const Node<Type>* node, std::ostream& output) const
    {
        if (root == nullptr)
        {
            throw(Except_runtime_error_class("Tree is empty - Nothing to print", EMPTY, 5));
        }
        if (node != nullptr)
        {
            output << node->value << "  ";

            PreOrder_Recursion(node->left, output);

            PreOrder_Recursion(node->right, output);

        }
    }

    void BreadthFirst_Recursion(std::queue<Node<Type>*>& queue, const Node<Type>* node, std::ostream& output) const
    {
        if (root == nullptr)
        {
            throw(Except_runtime_error_class("Tree is empty - Nothing to print", EMPTY, 5));
        }
        if (node != nullptr)
        {
            output << node->value << "  ";

            if (node->left != nullptr)
            {
                queue.push(node->left);
            }

            if (node->right != nullptr)
            {
                queue.push(node->right);
            }

            queue.pop();

            if (!queue.empty())
            {
                BreadthFirst_Recursion(queue, queue.front(), output);
            }
        }
    }



    void Print2DUtil(const Node<Type>* branch, int space, std::ostream& output) const
    {
        // Base case
        if (branch == NULL)
            return;

        // Increase distance between levels
        space += 10;

        // Process right child first
        Print2DUtil(branch->right, space, output);

        // Print current node after space
        // count
        output << '\n';
        for (int i = 10; i < space; i++)
        {
            output << " ";
        }

        output << branch->value << "\n";

        // Process left child
        Print2DUtil(branch->left, space, output);
    }

    // Helper function to print branches of the binary tree
    void showTrunks(Trunk* p, std::ostream& output)
    {
        if (p == nullptr)
            return;

        showTrunks(p->prev, output);

        output << p->str;
    }

    // Recursive function to print binary tree
    // It uses inorder traversal
    void printTreeHelper(Node<Type>* node, Trunk* prev, bool isLeft, std::ostream& output)
    {
        if (node == nullptr)
            return;

        std::string prev_str = "    ";
        Trunk* trunk = new Trunk(prev, prev_str);

        printTreeHelper(node->right, trunk, true, output);

        if (!prev)
            trunk->str = "---";
        else if (isLeft)
        {
            trunk->str = ".---";
            prev_str = "    |";
        }
        else
        {
            trunk->str = "`---";
            prev->str = prev_str;
        }

        showTrunks(trunk, output);
        output << node->value << '\n';

        if (prev)
            prev->str = prev_str;
        trunk->str = "    |";

        printTreeHelper(node->left, trunk, false, output);
    }


public:

    LinkedBinaryTree<Type>()
    {
        root = nullptr;
        currentSize = 0;
    }

    //LinkedBinaryTree<Type>(const LinkedBinaryTree<Type>& otherTree);

    ~LinkedBinaryTree()
    {
        destroy();
    }

    void destroy()
    {
        destroyRecursion(root);
        root = nullptr;
        currentSize = 0;
    }

    //bool empty() const;
    //bool full() const;
    //int size() const;

    void insert(const Type& newValue)
    {
        if (root == nullptr)
        {
            root = new Node<Type>(newValue, nullptr, nullptr, nullptr);
        }
        else
        {
            insertRecursion(newValue, root);
        }

        currentSize++;
    }

    Type* search(const Type& searchValue) const
    {
        search(searchValue, root);
    }

    void printAll(std::ostream& output) const;

    void Traversal_InOrder(std::ostream& output) const
    {
        output << "In Order Traversal:" << '\n';
        InOrder_Recursion(root, output);
        output << "\n\n";
    }


    void Traversal_PostOrder(std::ostream& output) const
    {
        output << "Post Order Traversal:" << '\n';
        PostOrder_Recursion(root, output);
        output << "\n\n";
    }

    void Traversal_PreOrder(std::ostream& output) const
    {
        output << "Pre Order Traversal:" << '\n';
        PreOrder_Recursion(root, output);
        output << "\n\n";
    }

    void Traversal_BreadthFirst(std::ostream& output) const
    {
        std::queue<Node<Type>*> queue;

        queue.push(root);

        output << "Breadth First Traversal:" << '\n';
        BreadthFirst_Recursion(queue, root, output);
        output << "\n\n";
    }

    void Print2D(std::ostream& output) const
    {
        Print2DUtil(root, 0, output);
    }



    void printTree(std::ostream& output) const
    {
        printTreeHelper(root, nullptr, false, output);
        output << "\n\n";
    }

    void PrintLevelByLevel(std::ostream& output) const
    {
        if (root == nullptr)
        {
            throw(Except_runtime_error_class("Tree is empty - Nothing to print", EMPTY, 5));
        }

        output << "Level By Level:" << '\n';

        std::queue<Node<Type>*> queue;

        int levelNodes = 0;

        queue.push(root);

        while (!queue.empty())
{

            levelNodes = queue.size();

            while (levelNodes > 0)
            {
                Node<Type>* temp = queue.front();

                output << temp->value << "  ";

                if (temp->left != nullptr)
                {
                    queue.push(temp->left);
                }

                if (temp->right != nullptr)
                {
                    queue.push(temp->right);
                }

                queue.pop();

                levelNodes--;
            }
            output << '\n';
        }

        output << "\n";
    }



};


#endif // !_LINKEDBINARYTREE_H_

```

```
/*****************************************************************************
 * 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
     */
    virtual int getErrorNumber() const throw() {
        return error_number;
    }

    /**Returns error offset.
     * @return #error_offset
     */
    virtual int getErrorOffset() const throw() {
        return error_offset;
    }

    virtual void outputError(std::ostream& output) const throw()
    {
        output << "Exception - Error number " << error_number
            << ":" << std::string(error_offset, ' ') << what() << '\n';
    }

};


#endif // !_EXCEPT_H_
```

```cpp
/******************************************************************************
 * 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";
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_
```