```
//*****************************************************************************************
**
//
//              File:                                   studentList.cpp
//
//              Student:                                Sean Herrick
//
//              Assignment:                             Program #09
//
//              Course Name:                    Data Structures II
//
//              Course Number:                  COSC 3100-01
//
//              Due:                                    November 7th, 2022
//
//
//              This program is an example of a Binary Search Tree's operations
//
//              Other files required:
//                      1.      Node.h
//                      2.      BST.h
//                      3.      student.h
//
//*****************************************************************************************
**

#include <iostream>
#include <fstream>
#include <new>
#include <iomanip>

using namespace std;

#include "BST.h"
#include "student.h"


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

void buildList ( BST <Student> & studentList );
void process ( BST <Student> & studentList );
char getChoice ( );
void addStudent ( BST <Student> & studentList );
void removeStudent ( BST <Student> & studentList );
void findStudent ( BST <Student> & studentList );
void updateStudent ( BST <Student> & studentList );
void listInformation ( BST <Student> & studentList );
void printStudents ( BST <Student> & studentList );
void displayOneStudent ( Student );

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

int main ( )
{
        BST <Student> studentList;

        buildList ( studentList );
        process ( studentList );

        return 0;
}
```

```
//*****************************************************************************************
**

void buildList ( BST <Student>& studentList )
{
        Student students;
        ifstream studentData;

        studentData.open ( "studentFile.txt" );

        while ( studentData >> students.id )
        {
                studentData.ignore ( );
                studentData.getline ( students.name, 50 );
                studentData.getline ( students.citystate, 50 );
                studentData.getline ( students.phone, 12 );
                studentData >> students.gender >> students.year >> students.credits
                                        >> students.gpa;
                studentData.ignore ( );
                studentData.getline ( students.major, 6 );
                studentList.insert ( students );
        }
}

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

void process ( BST <Student> & studentList )
{
        char choice;

        do
        {
                choice = getChoice ( );

                switch ( choice )
                {
                case 'A':
                        addStudent ( studentList );
                        break;
                case 'S':
                        findStudent ( studentList );
                        break;
                case 'U':
                        updateStudent ( studentList );
                        break;
                case 'D':
                        removeStudent ( studentList );
                        break;
                case 'P':
                        printStudents ( studentList );
                        break;
                case 'X':
                        listInformation ( studentList );
                        break;
                case 'Q':
                        break;
                }
        }
        while ( choice != 'Q' );
}

//*****************************************************************************************
**
```

```
char getChoice ( )
{
        char choice = ' ';
        bool valid;

        cout << "======== MENU ========\n"
                 << "A:    Add a new Student\n"
                 << "S:    Search for a Student Record\n"
                 << "U:    Update a Student Record\n"
                 << "D:    Delete a Student Record\n"
                 << "P:    Print Student Records\n"
                 << "X:    List General Information\n"
                 << "Q:    Quit\n"
                 << "Enter a choice: ";

        do
        {
                cin >> choice;
                choice = toupper ( choice );

                switch ( choice )
                {
                case 'A':
                        valid = true;
                        break;
                case 'S':
                        valid = true;
                        break;
                case 'U':
                        valid = true;
                        break;
                case 'D':
                        valid = true;
                        break;
                case 'P':
                        valid = true;
                        break;
                case 'X':
                        valid = true;
                        break;
                case 'Q':
                        valid = true;
                        break;
                default:
                        valid = false;
                        cout << "\ainvalid choice\n" << "Please try again: ";
                        break;
                }
        }
        while ( ! ( valid ) );

        return choice;
}

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

void addStudent ( BST <Student> & studentList )
{
        Student students;
        bool success = false;

        cout << "Enter new student ID: ";
        cin >> students.id;
```

```
        cin.ignore ( );
        cout << "Enter new student name: ";
        cin.getline ( students.name, 50 );

        cout << "Enter new student city and state: ";
        cin.getline ( students.citystate, 50 );

        cout << "Enter new student phone number: ";
        cin.getline ( students.phone, 12 );

        cout << "Enter new student year: ";
        cin >> students.year;

        cout << "Enter new student gender: ";
        cin >> students.gender;

        cin.ignore ( );
        cout << "Enter new student major: ";
        cin.getline ( students.major, 6 );

        cout << "Enter new student credits: ";
        cin >> students.credits;

        cout << "Enter new student gpa: ";
        cin >> students.gpa;

        if ( studentList.insert ( students ) )
        {
                cout << "Student added!" << endl << endl;
                cout << "ID" << setw ( 10 ) << "Name"  << setw ( 25 ) << "GPA" << setw ( 11 )
                 << "Credits" << setw ( 9 ) << "Major" << endl;

                displayOneStudent ( students );
        }

        else
        {
                cout << "Student, " << students.name << " could not be added." << endl << endl;
        }
}

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

void findStudent ( BST <Student> & studentList )
{
        Student students;

        cout << "Enter the student id you want to find: ";
        cin >> students.id;

        if ( studentList.retrieve ( students ) )
        {
                cout << endl << "Student successfully retrieved!" << endl << endl;
            cout << "ID" << setw ( 10 ) << "Name"  << setw ( 25 ) << "GPA" << setw ( 11 )
                 << "Credits" << setw ( 9 ) << "Major" << endl;

                displayOneStudent ( students );
        }

        else
        {
                cout << "Student was not found." << endl << endl;
        }
}
```

```
//*****************************************************************************************
**

void updateStudent ( BST <Student> & studentList )
{
        Student students;

        cout << "Enter the student id you want to update: ";
        cin >> students.id;

    cin.ignore ( );
        cout << "Enter the student name to be updated: ";
        cin.getline ( students.name, 50 );

        cout << "Enter the student city and state to be updated: ";
        cin.getline ( students.citystate, 50 );

        cout << "Enter the student phone number to be updated: ";
        cin.getline ( students.phone, 12 );

        cout << "Enter the student year to be updated: ";
        cin >> students.year;

        cout << "Enter the student gender to be updated: ";
        cin >> students.gender;

        cin.ignore ( );
        cout << "Enter the student major to be updated: ";
        cin.getline ( students.major, 6 );

        cout << "Enter the student credits to be updated: ";
        cin >> students.credits;

        cout << "Enter the student gpa to be updated: ";
        cin >> students.gpa;

        if ( studentList.update ( students ) )
        {
                cout << students.name << "'s record has been updated!" << endl << endl;
            cout << "ID" << setw ( 10 ) << "Name"  << setw ( 25 ) << "GPA" << setw ( 11 )
                 << "Credits" << setw ( 9 ) << "Major" << endl;

                displayOneStudent ( students );
        }

        else
        {
                cout << "Failed to updated, " << students.name << "'s record." << endl << endl;
        }
}


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

void removeStudent ( BST <Student>& studentList )
{
        Student students;

        cout << "Enter the student ID that you want to remove: ";
        cin >> students.id;

        if ( studentList.remove ( students ) )
        {
```

```
                cout << "The student record below has been removed!" << endl;
            cout << "ID" << setw ( 10 ) << "Name"  << setw ( 25 ) << "GPA" << setw ( 11 )
                 << "Credits" << setw ( 9 ) << "Major" << endl;

                displayOneStudent ( students );
        }

        else
        {
                cout << "Could not remove " << students.name << "." << endl << endl;
        }

}

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

void printStudents ( BST <Student> & studentList )
{
        cout << "ID" << setw ( 10 ) << "Name"  << setw ( 25 ) << "GPA" << setw ( 11 )
                 << "Credits" << setw ( 9 ) << "Major" << endl;
        studentList.inorderTraverse ( displayOneStudent );
}

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

void displayOneStudent ( Student students )
{
        cout << left << setw ( 8 ) << students.id << setw ( 26 )  << students.name
                 << setw ( 6 ) << fixed << setprecision ( 2 ) << setw ( 7 ) << students.gpa << setw (
11 )
                 << students.credits << students.major << endl;
}

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

void listInformation ( BST <Student> & studentList )
{
        int students = 0;
        int height = 0;

        students = studentList.getCount ( );
        height = studentList.getHt ( );

        cout << "There are " << students << " student(s) in the list" << endl;
        cout << "The height of the tree is " << height << endl;

        if ( studentList.isEmpty ( ) )
        {
                cout << "The tree is empty" << endl;
        }

        else
        {
                cout << "The tree is not empty" << endl;
        }

        if ( studentList.isFull ( ) )
        {
                cout << "The tree is full" << endl;
        }

        else
```

```
        {
                cout << "The tree is not full" << endl;
        }

        if ( studentList.isComplete ( ) )
        {
                cout << "The tree is a complete tree" << endl;
        }

        else
        {
                cout << "The tree is not a complete tree" << endl;
        }

        if ( studentList.isBalanced ( ) )
        {
                cout << "The tree is a balanced tree" << endl;
        }

        else
        {
                cout << "The tree is not a balanced tree" << endl;
        }
}

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

/*
ID      Name                    GPA    Credits    Major
3930    Leibniz, Gottfried W    1.95   13         MATH
4454    Atanasoff, Eniac C      1.88   14         CPSC
4559    Shyster, Samuel D       1.95   13         SOCI
4777    Gauss, Carl F           4.00   41         MATH
5316    GotoDijkstra, Edgar G   4.00   15         CPSC
5430    Nightingale, Florence K 3.15   15         NURS
5710    Busch, Arch E           2.75   74         ENGR
5873    Psycho, II,  Prunella E 2.99   120        PSYC
7107    Shoemaker, Imelda M     3.15   15         POLS
7448    Roosevelt, Rose Y       2.95   135        POLS
7844    Aardvark, Anthony A     2.79   43         ENGR
7885    Fibonacci, Leonard O    3.25   115        MATH
9463    Hochschule, Hortense C  2.70   100        EDUC
9743    Johnson, James L        3.15   15         ENGR
10236   Andrews, Peter J        2.78   42         CPSC
10304   Deutsch, Sprechen Z     3.05   14         GERM
11688   Kronecker, Leo P        2.75   77         MATH
11749   Issacson, Jacob A       2.99   25         RELI
11951   Mouse, Michael E        1.99   87         EDUC
13511   Pitt, Stew              0.21   12         GNED
14674   Rockne, Newton K        1.98   116        PE
14815   Tchaikovsky, Wolfgang A 2.75   79         MUSC
15052   Einstein, Alfred M      2.78   41         ENGR
15671   Rembrandt, Roberta E    2.20   77         ART
15755   VandenVander, Vanessa V 3.74   110        HIST
15802   Pascal, Blaze R         1.98   15         CPSC
15889   Gazelle, Gwendolyn D    2.78   43         PE
16183   Kuts, Cole              3.98   105        FOOD
16540   Weerd, Dewey L          2.99   115        PHIL
16622   Issacson, Esau B        2.98   25         RELI
17376   Scrooge, Ebenezer T     3.25   118        SOCI
17424   Nakamura, Toky O        1.95   12         SOCI
18213   Marx, Karl Z            2.75   78         ECON
18264   Lucky, Lucy L           2.29   66         HIST
19077   Medes, Archie L         3.10   80         ENGR
```

```
19918   Virus, Vera W             3.25   115        CPSC
20454   Chicita, Juanita A        2.66   95         BIOL
20991   Augusta, Ada B            3.83   46         CPSC
21144   Pasteur, Louise A         3.10   16         BIOL
22277   Principal, Pamela P       1.75   14         EDUC
22447   Zylstra, Zelda A          1.95   16         ENGL
23314   Macdonald, Ronald B       2.99   15         CPSC
23497   Fault, Paige D            2.95   55         CPSC
23544   Gestalt, Gloria G         2.48   42         PSYC
23750   Vespucci, Vera D          2.29   89         GEOG
24237   Euler, Lennie L           3.83   15         MATH
25377   Porgy, Bess N             2.78   44         MUSI
25831   Santamaria, Nina P        1.77   15         HIST
26316   Custer, General G         1.95   40         HIST
27503   Fahrenheit, Felicia O     3.85   40         CHEM
28658   Cicero, Marsha            2.87   77         LATI
29583   Yewliss, Cal C            2.99   76         MATH
30268   Newmann, Alfred E         0.99   115        EDUC
30280   Dewey, Johanna A          3.83   41         EDUC
30381   Elba, Able M              3.40   77         SPEE
30655   Angelo, Mike L            3.74   117        ART
30749   Mendelssohn, Mozart W     2.87   76         MUSC
30878   Lewis, Clark N            3.37   114        GEOG
31631   Aristotle, Alice A        3.10   78         PHIL
32598   Xerxes, Art I             3.25   119        GREE
32631   Freud, JR, Fred E         1.85   15         PSYC

There are 61 student(s) in the list
The height of the tree is 12
The tree is not empty
The tree is not full
The tree is not a complete tree
The tree is not a balanced tree
*/
```

```
//*************************************************************************************************
**
//
//              File:                                   student.h
//
//              Student:                                Sean Herrick
//
//              Assignment:                             Program #09
//
//              Course Name:                    Data Structures II
//
//              Course Number:                  COSC 3100-01
//
//              Due:                                    November 7th, 2022
//
//
//              This program is an example of a Binary Search Tree's operations
//
//              Other files required:
//                      1.      BST.h
//                      2.      studentList.cpp
//                      3.      Node.h
//
//*************************************************************************************************
**

#ifndef STUDENT_H
#define STUDENT_H

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

struct Student
{
        int id,
                year,
                credits;
        char name [ 50 ],
                citystate [ 50 ],
                phone [ 12 ],
                gender,
                major [ 6 ];
        float gpa;

        Student ( );
        friend ostream & operator << ( ostream & out, const Student& data );
        bool operator == ( const Student & rhs ) const;
        bool operator == ( int value ) const;
        bool operator != ( const Student & rhs ) const;
        bool operator != ( int value ) const;
        bool operator < ( const Student & rhs ) const;
        bool operator < ( int value ) const;
        bool operator > ( const Student & rhs ) const;
        bool operator > ( int value ) const;
        bool operator <= ( const Student & rhs ) const;
        bool operator <= ( int value ) const;
        bool operator >= ( const Student & rhs ) const;
        bool operator >= ( int value ) const;
        int operator % ( int value ) const;
        Student & operator = ( int value );

};

//*************************************************************************************************
```

```
**

Student :: Student ( )
{
        id = 0;
}

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

ostream & operator << ( ostream & out, const Student & data )
{
        out << data.id << "/";

        for ( int i = 0; i < 6; i++ )
        {
                out << data.name [ i ];
        }

        return out;
}

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

bool Student :: operator == ( const Student & rhs ) const
{
        return ( this->id == rhs.id );
}

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

bool Student :: operator == ( int value ) const
{
        return ( this->id == value );
}

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

bool Student :: operator != ( const Student & rhs ) const
{
        return ( this->id != rhs.id );
}

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

bool Student :: operator != ( int value ) const
{
        return ( this->id != value );
}

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

bool Student :: operator < ( const Student & rhs ) const
{
        return ( this->id < rhs.id );
}

//*************************************************************************************************
**
```

```
bool Student :: operator < ( int value ) const
{
        return ( this->id < value );
}

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

bool Student :: operator > ( const Student & rhs ) const
{
        return ( this->id > rhs.id );
}

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

bool Student :: operator > ( int value ) const
{
        return ( this->id > value );
}

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

bool Student :: operator <= ( const Student & rhs ) const
{
        return ( this->id <= rhs.id );
}

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

bool Student :: operator <= ( int value ) const
{
        return ( this->id <= value );
}

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

bool Student :: operator >= ( const Student & rhs ) const
{
        return ( this->id >= rhs.id );
}

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

bool Student :: operator >= ( int value ) const
{
        return ( this->id >= value );
}

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

Student & Student :: operator = ( int value )
{
        this->id = value;

        return *this;
}

//*************************************************************************************************
**
```

```
int Student :: operator % ( int value ) const
{
        return ( this->id % value );
}

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

#endif
```

```
//*****************************************************************************************
**
//
//              File:                                   Node.h
//
//              Student:                                Sean Herrick
//
//              Assignment:                             Program #09
//
//              Course Name:                    Data Structures II
//
//              Course Number:                  COSC 3100-01
//
//              Due:                                    November 7th, 2022
//
//
//              This program is an example of a Binary Search Tree's operations
//
//              Other files required:
//                      1.      studentList.cpp
//                      2.      BST.h
//                      3.      student.h
//
//*****************************************************************************************
**

#ifndef NODE_H
#define NODE_H

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

template <typename TYPE>
class Node
{
public:
    TYPE data;

    union
    {
        Node <TYPE>* next;
        Node <TYPE>* left;
    };

    union
    {
        Node <TYPE>* prev;
        Node <TYPE>* right;
    };

    Node ( );
    Node ( const TYPE& d, Node <TYPE>* n = nullptr, Node <TYPE>* p = nullptr );

};

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

template <typename TYPE>
Node <TYPE>::Node ( )
{
    data = 0;
    next = nullptr;
    prev = nullptr;
```

```
}

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

template <typename TYPE>
Node<TYPE>::Node ( const TYPE& d, Node <TYPE>* n, Node <TYPE>* p )
{
    data = d;
    next = n;
    prev = p;
}

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

#endif
```

```
//*************************************************************************************************
**
//
//              File:                                   BST.h
//
//              Student:                                Sean Herrick
//
//              Assignment:                             Program #09
//
//              Course Name:                    Data Structures II
//
//              Course Number:                  COSC 3100-01
//
//              Due:                                    November 7th, 2022
//
//
//              This program is an example of a Binary Search Tree's operations
//
//              Other files required:
//                      1.      Node.h
//                      2.      student.h
//                      3.      studentList.cpp
//
//*************************************************************************************************
**

#ifndef BST_H
#define BST_H

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

#include "Node.h"

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

template <typename TYPE>
class BST
{
private:
    Node <TYPE>* root;
    void _destruct ( Node <TYPE>* pRoot );
    Node <TYPE>* _retrieve ( Node <TYPE>* pRoot, const TYPE & dataOut ) const;
    Node <TYPE>* _insert ( Node <TYPE>* pRoot, const TYPE & dataIn );
    Node <TYPE>* _remove ( Node <TYPE>* pRoot, const TYPE & dataOut );
    void _inorderTraverse ( Node <TYPE>* pRoot, void ( * process ) ( TYPE x ) ) const;
    int _getCount ( Node <TYPE>* pRoot ) const;
    int _getHt ( Node <TYPE>* pRoot ) const;
    bool _isBalanced ( Node <TYPE>* pRoot ) const;
    bool _isComplete ( Node <TYPE>* pRoot ) const;

public:

    BST ( );
    ~BST ( );
    bool insert ( const TYPE & dataIn );
    bool retrieve ( TYPE & dataOut ) const;
    bool remove ( TYPE & dataIn );
    bool update ( const TYPE & dataIn );
    void inorderTraverse ( void ( * process ) ( TYPE x ) ) const;
    int getCount ( ) const;
    int getHt ( ) const;
    bool isBalanced ( ) const;
```

```
    bool isComplete ( ) const;
    bool isEmpty ( ) const;
    bool isFull ( ) const;

};

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

template <typename TYPE>
BST <TYPE>::BST ( )
{
    root = nullptr;
}

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

template <typename TYPE>
BST <TYPE>::~BST ( )
{
    _destruct ( root );
}

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

template <typename TYPE>
void BST <TYPE>:: _destruct ( Node <TYPE>* pRoot )
{
    if ( pRoot != nullptr )
    {
        _destruct ( pRoot->left );
        _destruct ( pRoot->right );
        delete pRoot;
    }
}

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

template <typename TYPE>
bool BST <TYPE>:: retrieve ( TYPE & dataOut ) const
{
    bool success = false;
    Node <TYPE>* pFound;

    pFound = _retrieve ( root, dataOut );

    if ( pFound != nullptr )
    {
        dataOut = pFound->data;
        success = true;
    }

    return success;
}

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

template <typename TYPE>
Node <TYPE>* BST <TYPE>:: _retrieve ( Node <TYPE>* pRoot, const TYPE & dataOut ) const
{
    if ( pRoot != nullptr )
```

```
    {
        if ( pRoot->data > dataOut )
        {
            pRoot = _retrieve ( pRoot->left, dataOut );
        }

        else if ( pRoot->data < dataOut )
        {
            pRoot = _retrieve ( pRoot->right, dataOut );
        }
    }

    return pRoot;
}

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

template <typename TYPE>
bool BST <TYPE>:: insert ( const TYPE & dataIn )
{
    bool success = false;
    Node <TYPE>* pFound;

    pFound = _retrieve ( root, dataIn );

    if ( pFound == nullptr )
    {
        root = _insert ( root, dataIn );
        success = true;
    }

    return success;
}

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

template <typename TYPE>
Node <TYPE>* BST <TYPE>:: _insert ( Node <TYPE>* pRoot, const TYPE & dataIn )
{
    if ( pRoot != nullptr )
    {
        if ( pRoot->data > dataIn )
        {
            pRoot->left = _insert ( pRoot->left, dataIn );
        }

        else
        {
            pRoot->right = _insert ( pRoot->right, dataIn );
        }
    }

    else
    {
        pRoot = new Node <TYPE> ( dataIn );
    }

    return pRoot;
}

//*************************************************************************************************
**
```

```
template <typename TYPE>
bool BST <TYPE>:: remove ( TYPE & dataIn )
{
    bool success = false;
    Node <TYPE>* pFound;

    pFound = _retrieve ( root, dataIn );

    if ( pFound != nullptr )
    {
        dataIn = pFound->data;
        root = _remove ( root, dataIn );
        success = true;
    }

    return success;
}

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

template <typename TYPE>
Node <TYPE>* BST <TYPE>:: _remove ( Node <TYPE>* pRoot, const TYPE & dataOut )
{
         Node<TYPE>* pDel;
         Node<TYPE>* pMax;

         if ( pRoot != nullptr )
         {
                 if ( pRoot->data > dataOut )
                 {
                         pRoot->left = _remove( pRoot->left, dataOut );
                 }

                 else if ( pRoot->data < dataOut )
                 {
                         pRoot->right = _remove( pRoot->right, dataOut );
                 }

                 else
                 {
             //2 children remove
                         if ( ( pRoot->left != nullptr ) && ( pRoot->right != nullptr ) )
                         {
                                 pMax = pRoot->left;

                                 while ( ( pMax != nullptr ) && ( pMax->right != nullptr ) )
                                 {
                                         pMax = pMax->right;
                                 }

                                 pRoot->data = pMax->data;
                                 pRoot->left = _remove ( pRoot->left, dataOut );
                         }

             //1 and none children remove
                         else
                         {
                                 pDel = pRoot;

                                 if ( pRoot->left != nullptr )
                                 {
                                         pRoot = pRoot->left;
                                 }
```

```
                                else
                                {
                                        pRoot = pRoot->right;
                                }

                                delete pDel;
                        }
                }

        }

        return pRoot;
}

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

template <typename TYPE>
bool BST <TYPE>:: update ( const TYPE & dataIn )
{
    bool success = false;
    Node <TYPE>* pFound;

    pFound = _retrieve ( root, dataIn );

    if ( pFound != nullptr )
    {
        pFound->data = dataIn;
        success = true;
    }

    return success;
}

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

template <typename TYPE>
void BST <TYPE>:: inorderTraverse ( void ( * process ) ( TYPE x ) ) const
{
    _inorderTraverse ( root, process );
}

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

template <typename TYPE>
void BST <TYPE>:: _inorderTraverse ( Node <TYPE>* pRoot, void ( * process ) ( TYPE x ) ) const
{
    if ( pRoot != nullptr )
    {
        _inorderTraverse ( pRoot->left, process );
        process ( pRoot->data );
        _inorderTraverse ( pRoot->right, process );
    }

}

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

template <typename TYPE>
int BST <TYPE>:: getCount ( ) const
{
    return ( _getCount ( root ) );
```

```
}

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

template <typename TYPE>
int BST <TYPE>:: _getCount ( Node <TYPE>* pRoot ) const
{
    int count = 0;

    if ( pRoot != nullptr )
    {
        count = ( ( 1 ) + _getCount ( pRoot->left ) + _getCount ( pRoot->right ) );
    }

    return count;
}

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

template <typename TYPE>
int BST <TYPE>:: getHt ( ) const
{
    return ( _getHt ( root ) - 1 );
}

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

template <typename TYPE>
int BST <TYPE>:: _getHt ( Node <TYPE>* pRoot ) const
{
    int ht = 0;

    if ( pRoot != nullptr )
    {
        ht = 1 + max ( _getHt ( pRoot->left ), _getHt ( pRoot->right ) );

    }

    return ht;
}

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

template <typename TYPE>
bool BST <TYPE>:: isEmpty ( ) const
{
    bool success = true;

    if ( root != nullptr )
    {
        success = false;
    }

    return success;
}

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

template <typename TYPE>
bool BST <TYPE>:: isFull ( ) const
```

```
{
    bool success = true;
    Node <TYPE>* pNew;

    pNew = new ( nothrow ) Node <TYPE>;

    if ( pNew != nullptr )
    {
        success = false;
        delete pNew;
    }

    return success;
}

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

template <typename TYPE>
bool BST <TYPE>:: isComplete ( ) const
{
    return ( _isComplete ( root ) );
}

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

template <typename TYPE>
bool BST <TYPE>:: _isComplete ( Node <TYPE>* pRoot ) const
{
    bool complete = true;

    if ( pRoot != nullptr )
    {
        if ( ( _getHt ( pRoot->left ) == _getHt ( pRoot->right ) ) )
        {
            complete = ( _isComplete ( pRoot->left ) && _isComplete ( pRoot->right ) );
        }

        else
        {
            complete = false;
        }
    }

    return complete;
}

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

template <typename TYPE>
bool BST <TYPE>:: isBalanced ( ) const
{
    return ( _isBalanced ( root ) );
}

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

template <typename TYPE>
bool BST <TYPE>:: _isBalanced ( Node <TYPE>* pRoot ) const
{
    bool balanced = true;
```

```
    if ( pRoot != nullptr )
    {
        if ( abs ( _getHt ( pRoot->left ) - _getHt ( pRoot->right ) ) <= 1 )
        {
            balanced = ( _isBalanced ( pRoot->left ) && _isBalanced ( pRoot->right ) );
        }

        else
        {
            balanced = false;
        }
    }

    return balanced;
}

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

#endif
```