Project 3: Binary Search Trees

1.1 Information

Topics: Trees, Binary Trees

Turn in: All source files (.cpp and .hpp).

Starter files: Download on GitHub or D2L.

Penalties: The following items may negatively impact your score.

- **Program doesn't build**
  Your program should always build. Programs turned in that don't build will automatically receive a grade of 50%. Additionally, I build your code from the command line in Linux; Your code should be portable. Certain features are allowed in Visual Studio or Windows but don't work for all compilers.
  Avoid: `#pragma once`, `system("pause")`, ignoring filename cases

- **Missing source files**
  If your .hpp or .cpp files are missing, they cannot be graded and will result in a 0%. Always double-check to make sure you're submitting all your files.

- **Visual Studio files**
  I don't want these. I ONLY want your .hpp and .cpp files. I won't count off if you turn it in, but do me a favor (and help me grade quickly) by not turning in junk files.

- **Zipped files**
  I don't want this. Just submit your source files. I won't count off if you turn in a zip, but when I download assignments they're already zipped so it just makes more work for me.

1.1.1 About

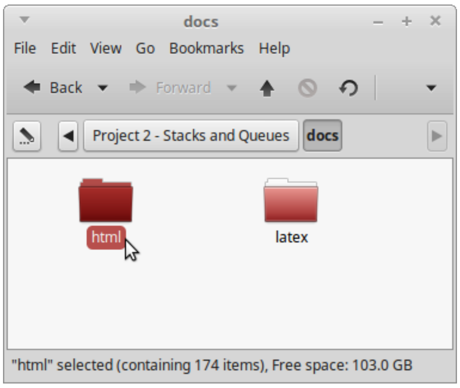
```
Project 3 - Binary Search Trees/
 |__CodeBlocks Projet - Program/ ...
 |  Contains the CB project for the program
 |__CodeBlocks Projet - Tester/ ...
 |  Contains the CB project for unit tests
 |__docs/ ...  Contains documentation pages
 |__main.cpp ...  Contains main()
 |__BinarySearchTree.hpp ...  Contains the declaration
 |                            for the Binary Tree
 |__BinarySearchTree.cpp
 |__Employee.hpp ...  Has functionality dealing with
 |                    Employees
 |__EmployeeManager.hpp ...  The virtual Airport
 |__EmployeeManager.cpp
 |__Tester.hpp ...  Unit tests file
 |__Timer.hpp ...  Timer to record duration
 |__employee-list.txt ...  Input file with employee data
```

For this program you will only be implementing functions in the Binary Search Tree.

1.2 Doxygen documentation

Once again, this project has documentation generated with **doxygen**. All the comments above functions are contained in these documentation pages, so you can consult it in either location.

To read the documentation, go to the **docs** folder, then the **html** folder.

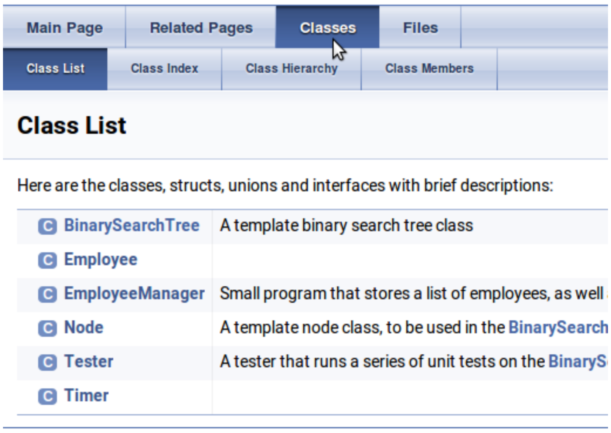

docs
File Edit View Go Bookmarks Help
Back
Forward
Project 2 - Stacks and Queues
docs
html
latex

"html" selected (containing 174 items), Free space: 103.0 GB

Figure 1.1: Inside the docs folder

Then, open **index.html**.

From the Index page, click on the **Classes** tab. This will open up a list of all the classes with documentation.



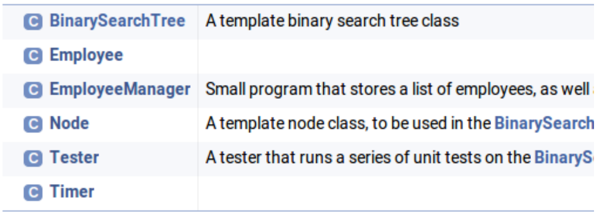
Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

BinarySearchTree	A template binary search tree class
Employee	
EmployeeManager	Small program that stores a list of employees, as well
Node	A template node class, to be used in the BinarySearch
Tester	A tester that runs a series of unit tests on the BinaryS
Timer	

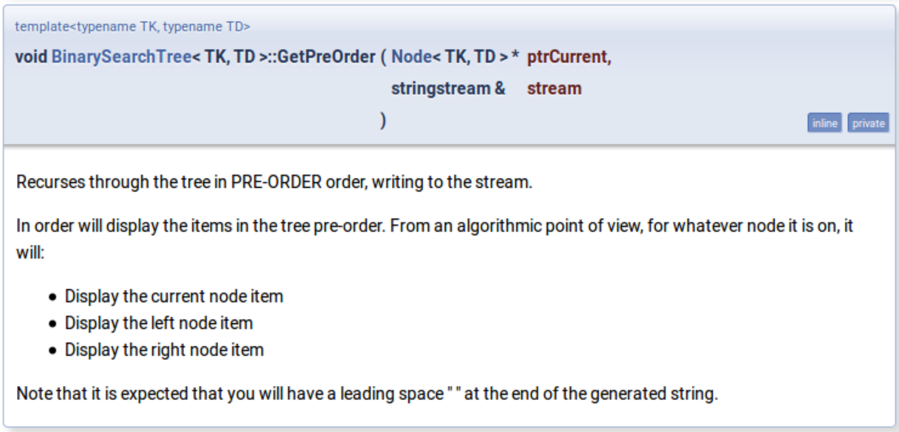
Figure 1.2: The Classes tab

Click on a class to view all its functions and the function specs.



BinarySearchTree	A template binary search tree class
Employee	
EmployeeManager	Small program that stores a list of employees, as well
Node	A template node class, to be used in the BinarySearch
Tester	A tester that runs a series of unit tests on the BinaryS
Timer	

Figure 1.3: The list of classes in the documentation



```
template<typename TK, typename TD>
void BinarySearchTree< TK, TD >::GetPreOrder ( Node< TK, TD > *  ptrCurrent,
                                                stringstream &    stream
                                              )
```

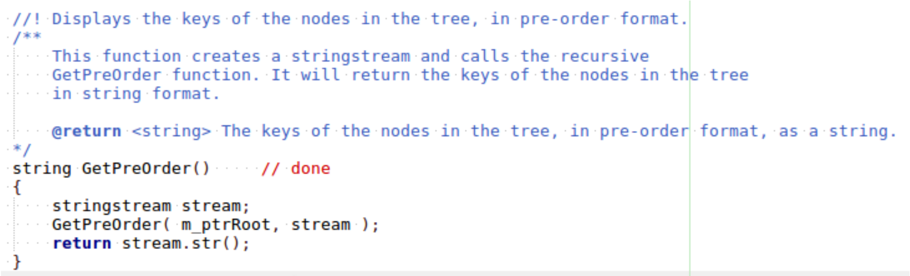
Recurses through the tree in PRE-ORDER order, writing to the stream.

In order will display the items in the tree pre-order. From an algorithmic point of view, for whatever node it is on, it will:

- Display the current node item
- Display the left node item
- Display the right node item

Note that it is expected that you will have a leading space " " at the end of the generated string.

Figure 1.4: A function's documentation



```
//! Displays the keys of the nodes in the tree, in pre-order format.
/**
    This function creates a stringstream and calls the recursive
    GetPreOrder function. It will return the keys of the nodes in the tree
    in string format.

    @return <string> The keys of the nodes in the tree, in pre-order format, as a string.
*/
string GetPreOrder()     // done
{
    stringstream stream;
    GetPreOrder( m_ptrRoot, stream );
    return stream.str();
}
```

Figure 1.5: Doxygen comments in the code

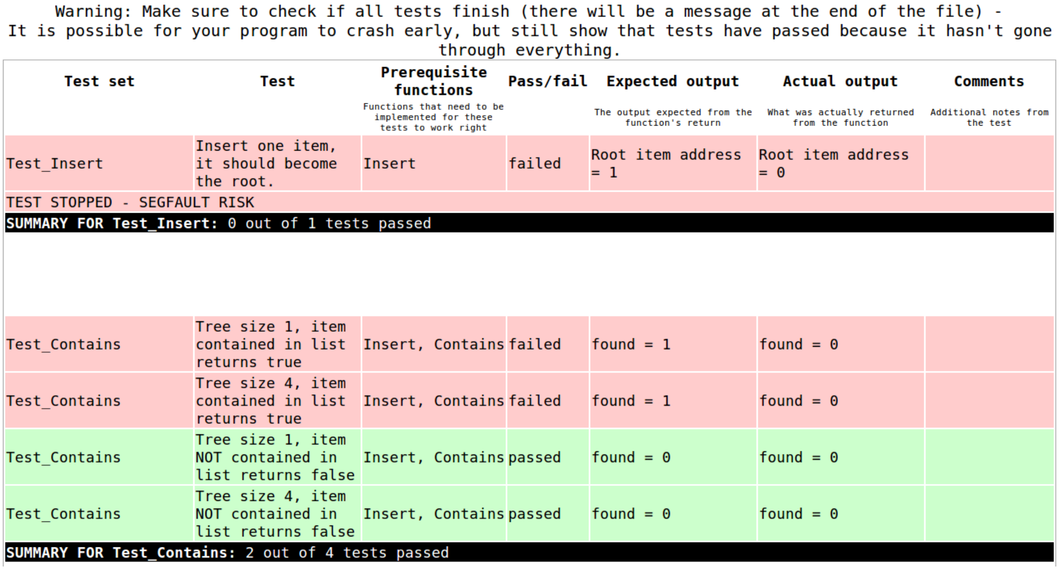
1.3 Testing

Before you start working with the actual program itself, you should develop your Binary Search Tree from within a project that uses unit tests. If you're using Code::Blocks, a CBP file is available in the **CodeBlocks Project - Tester** folder.

If you're making your own project, the files you need to do testing are:

- test_main.cpp
- BinarySearchTree.hpp
- Tester.hpp
- cuTEST/
  - TesterBase.hpp
  - TesterBase.cpp
  - StringUtil.hpp

When you run the test, it will generate an output file called **test_result.html** At first, its output will look like this:

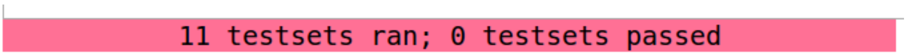


Warning: Make sure to check if all tests finish (there will be a message at the end of the file) - It is possible for your program to crash early, but still show that tests have passed because it hasn't gone through everything.

Test set	Test	Prerequisite functions Functions that need to be implemented for these tests to work right	Pass/fail	Expected output The output expected from the function's return	Actual output What was actually returned from the function	Comments Additional notes from the test
Test_Insert	Insert one item, it should become the root.	Insert	failed	Root item address = 1	Root item address = 0	
TEST STOPPED - SEGFAULT RISK						
SUMMARY FOR Test_Insert: 0 out of 1 tests passed						
Test_Contains	Tree size 1, item contained in list returns true	Insert, Contains	failed	found = 1	found = 0	
Test_Contains	Tree size 4, item contained in list returns true	Insert, Contains	failed	found = 1	found = 0	
Test_Contains	Tree size 1, item NOT contained in list returns false	Insert, Contains	passed	found = 0	found = 0	
Test_Contains	Tree size 4, item NOT contained in list returns false	Insert, Contains	passed	found = 0	found = 0	
SUMMARY FOR Test_Contains: 2 out of 4 tests passed						

Figure 1.6: Most unit tests fail

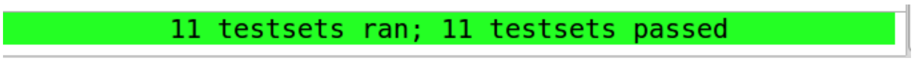
At the bottom of the page is the summary of all tests run:



```
11 testsets ran; 0 testsets passed
```

Figure 1.7: Most unit tests fail

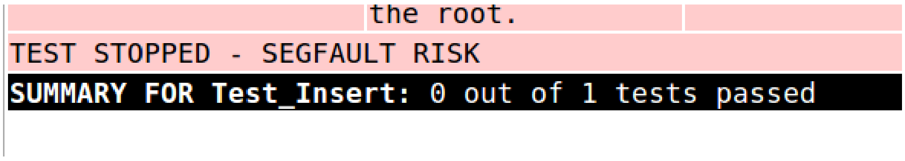
Once everything passes, it will look like this:



```
11 testsets ran; 11 testsets passed
```

Figure 1.8: Successful status at end

If there's a risk of a **segfault** (due to something returning null when it shouldn't), the test set will also cancel, not running all the tests:



```
the root.
TEST STOPPED - SEGFAULT RISK
SUMMARY FOR Test_Insert: 0 out of 1 tests passed
```

Figure 1.9: Segfault risk causes tests to exit early

Make sure your tests all pass before getting into the program portion!

1.4 Program

The program loads a list of employees from a text file.

```
hamadi muruaga
    mission-critical-outside-the-box
    hotel-management-salesperson
    happy-alligator-754@sonnet.info

aboudou herppich
    synergistic-scaffolding
    legal-trainee
    faint-king-penguin-1574@telekinetic.com

kittie chamica
    synergistic-bandwidth
    metal-fabrication-director
    playful-centipede-1400@spiderweb.co.uk
```

The employees are stored in the **EmployeeManager** in three separate trees so that we can see how the indexing works for different types of keys. It also stores the employees in an unsorted **vector**.

When you do a search for a specific employee, it the program will tell you how long it took to find them in the binary tree vs. how long it takes in the vector. For example:

```
LINEAR SEARCH TIME: 2.9768e-05 (microseconds) ... 3.1186e-05 (ticks)
TREE SEARCH TIME:   2.912e-06 (microseconds) ... 3.373e-06 (ticks)
```

1.4.1 Example output

Program startup:

```
10000 employees loaded
id index generated, size 10000
name index generated, size 10000
email index generated, size 10000


1. Search by ID
2. Search by name
3. Search by email
4. Quit

>>
```

Searching by ID:

```
SEARCH BY ID
id: 30

EMPLOYEE FOUND:
EMPLOYEE ID: 30
     FIRST NAME: nissrine
     LAST NAME:  millet
     JOB TITLE:  culinary-engineer
     COMPANY:    immersive-deep-dive
     EMAIL:      stark-x-ray-tetra-2799@cobweb.edu

     LINEAR SEARCH TIME: 3.023e-06 (microseconds) ... 4.37e-06 (ticks)
     TREE SEARCH TIME:   0.000152533 (microseconds) ... 0.000152954 (ticks)
```

Searching by name:

```
SEARCH BY NAME
lastname-firstname: nunn-mi

EMPLOYEE FOUND:
EMPLOYEE ID: 4
     FIRST NAME: mi
     LAST NAME:  nunn
     JOB TITLE:  recording-arts-analyst
     COMPANY:    empowering-outreach
     EMAIL:      helpless-french-bulldog-1820@cobweb.info

     LINEAR SEARCH TIME: 2.9768e-05 (microseconds) ... 3.1186e-05 (ticks)
     TREE SEARCH TIME:   2.912e-06 (microseconds) ... 3.373e-06 (ticks)
```

Searching by email:

```
SEARCH BY EMAIL
email: playful-centipede-1400@spiderweb.co.uk

EMPLOYEE FOUND:
EMPLOYEE ID: 2
     FIRST NAME: kittie
     LAST NAME:  chamica
     JOB TITLE:  metal-fabrication-director
     COMPANY:    synergistic-bandwidth
     EMAIL:      playful-centipede-1400@spiderweb.co.uk

     LINEAR SEARCH TIME: 3.906e-06 (microseconds) ... 4.905e-06 (ticks)
     TREE SEARCH TIME:   2.7e-06 (microseconds) ... 3.101e-06 (ticks)
```

1.5 Class declarations

Employee

```
struct Employee
{
    Employee();
    Employee( int newId, string newFirstName,
                string newLastName, string newCompany,
                string newJobTitle, string newEmail );
    void Display();

    int     id;
    string  firstName;
    string  lastName;
    string  company;
    string  jobTitle;
    string  email;
};

```

EmployeeManager

```
class EmployeeManager
{
public:
    EmployeeManager();

    void SearchById();
    void SearchByName();
    void SearchByEmail();

    void MainMenu();

private:
    void LoadEmployees();

    Employee* SearchById_Tree( int index );
    Employee* SearchByName_Tree( string name );
    Employee* SearchByEmail_Tree( string email );

    Employee* SearchById_Linear( int index );
    Employee* SearchByName_Linear( string name );
    Employee* SearchByEmail_Linear( string email );

    void GenerateIdIndex();
    void GenerateNameIndex();
    void GenerateEmailIndex();

    int GetIntInput( int min, int max );

private:
    vector<Employee> m_employeeList;

    BinarySearchTree<int, Employee*> m_idIndex;
    BinarySearchTree<string, Employee*> m_nameIndex;
    BinarySearchTree<string, Employee*> m_emailIndex;
};

```

Node

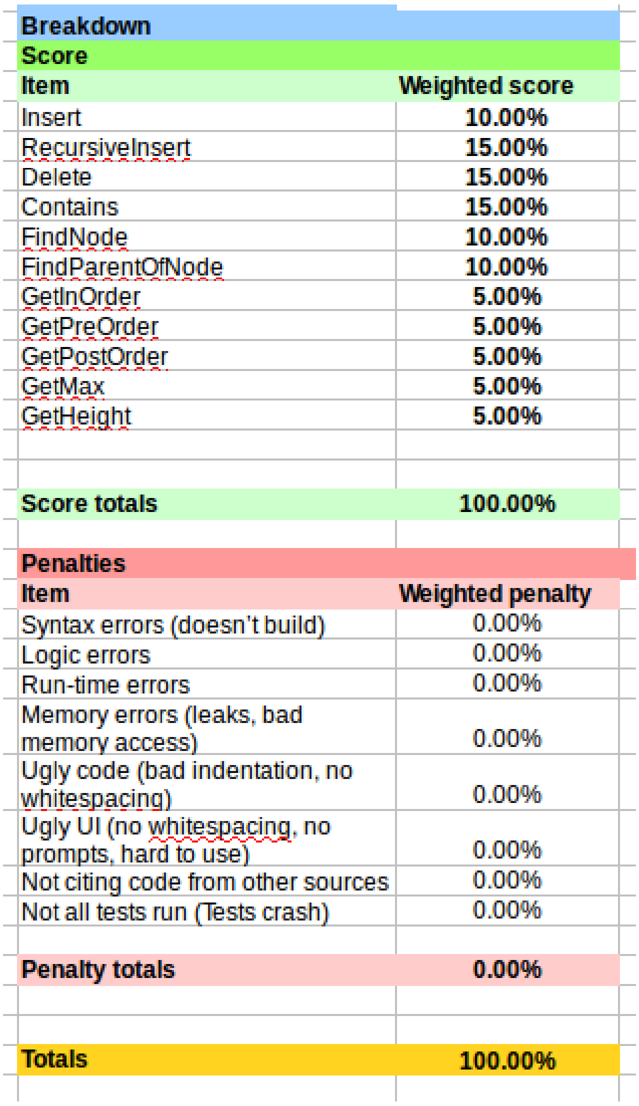
```
        template <typename TK, typename TD>
        class Node
        {
        public:
            Node();
            ~Node();
            Node<TK, TD>* ptrLeft;
            Node<TK, TD>* ptrRight;
            TD data;
            TK key;
        };

```

BinarySearchTree

```
template <typename TK, typename TD>
//! A template binary search tree class
class BinarySearchTree
{
public:
    BinarySearchTree();
    ~BinarySearchTree();

    void Insert( const TK& newKey, const TD& newData );
    void Delete( const TK& key );
    bool Contains( const TK& key );
    string GetInOrder();
    string GetPreOrder();
    string GetPostOrder();
    TK* GetMax();
    int GetCount();
    int GetHeight();
    TD* GetData( const TK& key );

private:
    Node<TK, TD>* FindNode( const TK& key );
    Node<TK, TD>* FindParentOfNode( const TK& key );
    void RecursiveInsert( const TK& newKey,
        const TD& newData, Node<TK, TD>* ptrCurrent );
    void GetInOrder( Node<TK, TD>* ptrCurrent,
        stringstream& stream );
    void GetPreOrder( Node<TK, TD>* ptrCurrent,
        stringstream& stream );
    void GetPostOrder( Node<TK, TD>* ptrCurrent,
        stringstream& stream );
    TK* GetMax( Node<TK, TD>* ptrCurrent );
    int GetHeight( Node<TK, TD>* ptrCurrent );

private:
    Node<TK, TD>* m_ptrRoot;
    int m_nodeCount;

friend class Tester;
};

```

Grading Breakdown

Features



Breakdown	
Score	
Item	**Weighted score**
Insert	**10.00%**
RecursiveInsert	**15.00%**
Delete	**15.00%**
Contains	**15.00%**
FindNode	**10.00%**
FindParentOfNode	**10.00%**
GetInOrder	**5.00%**
GetPreOrder	**5.00%**
GetPostOrder	**5.00%**
GetMax	**5.00%**
GetHeight	**5.00%**
Score totals	**100.00%**
Penalties	
Item	**Weighted penalty**
Syntax errors (doesn't build)	0.00%
Logic errors	0.00%
Run-time errors	0.00%
Memory errors (leaks, bad memory access)	0.00%
Ugly code (bad indentation, no whitespacing)	0.00%
Ugly UI (no whitespacing, no prompts, hard to use)	0.00%
Not citing code from other sources	0.00%
Not all tests run (Tests crash)	0.00%
Penalty totals	**0.00%**
Totals	**100.00%**