CS515 – Lab 7

The purpose of this lab is to review BST's and to start working on threaded BST's. You will implement some tree traversal operations (depth-first traversal and breath-first traversal) and the tree destructor.

Note: some of the code found in this lab can be used with some modification in assignment 9P—You'll be at a disadvantage if you skip it!

Download all files from `~cs515/public/7L`

The header file includes a header file `set.h`, a partially implemented `set.cpp` and a test driver `settest0.cpp`. The Set ADT is now implemented with a binary search tree. The tree is threaded and has a dummy root node in order to support efficient in-order traversals and iterator operations (though you don't need to work on Set::Iterator in this lab.) To start, you may compile the test driver `settest0.cpp` and see how a threaded binary search tree is built and displayed.

Your task is to implement two tree-traversal methods in set.cpp (**`depthFirstInOrder`** and **`breadthFirst`**) and the destruction method **`destructCode`** that used by the Set destructor.

1) **`depthFirstInOrder`**

There are three basic ways to traversal a BST: in-order, pre-order, and post-order. These three methods are called **depth-first** traversals since we try to go deeper in the tree before exploring the sibling nodes. Depth-first traversals can be implemented using a recursive algorithms or an iterative algorithms. You are asked to write an iterative version for depth-first traversal in this lab.

2) **`breadthFirst`**

Breadth-first traversal is another way to explore a tree. It is done level-by-level, such that we explore the full width of the tree at a given level, before going *deeper*. (Hint: you need to use a helper data structure (STL) in order to implement the breath-first traversal.)

3) **`destructCode`**

The `destructCode` method is a helper method that should free tree nodes recursively. Its argument is a pointer passed by reference, so that any modification to this pointer (formal parameter) within the method is reflected to the actual parameter in the calling method.

Once you are done implementing the three methods, uncomment the method calls in `settest0.cpp`. The program output should look like the following:

```
$./a.out
                          35
                    33
              21
      19
              17
14
      9
              3
Depth First In Order Traverse:
3 9 14 17 19 21 33 35
Breadth First Traverse:
14 9 19 3 17 21 33 35
```

You must run the program using Valgrind to make sure there are no memory leaks nor any other memory error in your programs, thus confirming your `destructCode` works.

Important: You must not modify the header file set.h, and any other existing methods.

Submission:

- Submit the file **`set.cpp`** and a **README** file.
- You may change the makefile locally, but we will be testing with our own.
- You should also fill out the README file and submit it as well. To submit the files in Mimir, follow the link for this lab in MyCourses. Here is a list of files you are expected to submit:

```
set.cpp README
```

- As always, do not turn in executables or object code, and make sure your submission compiles successfully on Agate. Programs that produce compile time errors or warnings will receive a zero mark (even if it might work perfectly on your home computer.) To check this, use the make command with the provided makefile, and check that your submission passes tests on Mimir.

Important:

You must include the standard comment block in each of your source files, including your name, section, date and collaboration details. You must also finish the README file along with your programs.

You should also include detailed collaboration declaration information in your comments. If you worked in pairs in this lab, each of you must include the partner's name in your program comment. Both you and your partner must complete an individual submission in order to earn a grade. If you work by yourself, you must indicate in the program comments that you have worked on your own independently.

You can resubmit as needed—your last submission will be graded. Here is a tentative grading scheme:

set test run 1 in-order depth-first	20
set test run 2 in-order depth- first	20
set test run 1 breadth-first	20
set test run 2 breadth-first	20
Valgrind check	20