# ICT283 Lab 10 (not assessed now but must be attempted)

**Objectives:**

* More practice with use of STL map.
* Continue with the data structures requirements of assignment 2.
* Start work on specific assignment 2 requirements shown in this lab.
* Experiment with sorting and do empirical comparisons of their performance.
* Make use of the pointer to function concept.
* Think critically about the problem-solving approaches that you use and the relevant theory/concepts.
* Consider using **Git** for Version Control of your code.

It is very important not to fall behind with these exercises. You should complete this lab to prepare for assignment 2.

**Demonstrate exercises d and e to your tutor. These constitute part of the progress demonstrations required for Assignment 2.**

**Specific Assignment 2 requirements start in this lab.**

**Exercises d and e are the specific requirements of Assignment 2. The assignment 2 specification will refer back to these exercises.**

**Data structure for bonus marks for the assignment is also in this document.**

## Exercises

**a.**

This question enables you to practice using the STL map if you feel you need more programming practice with the STL map (see Lab 7). The map (or variants like multimap) data structure is needed in assignment 2. For this exercise, you would need to refer back to the practice exercises from lab 2 onwards.

Use an STL map to relate students with their results. The student id is the map key. Compare your approach taken in Lab 7, exercise 1 where id was associated with name.

You should also consider wrapping (encapsulating) std::map in your own **class** **Map**. See the data structures section of exercise e below. The assignment has bonus marks for those who are able to do it as specified in exercise e below.

Write a test program to demonstrate that your map works. Multiple students and multiple results are involved.

**b.**

Create two arrays of 1000 (try 10,000 too) numbers each. Populate both arrays with random numbers or use your data reading routine from the assignment to read in one of the numerical data columns. Use the library routine *qsort* (<http://www.cppreference.com/wiki/c/other/qsort>) and *sort* (<http://www.cppreference.com/wiki/stl/algorithm/sort>) in turn to sort both the arrays. You will need to write a function for doing comparisons. The function is passed to *qsort* as a function pointer. The library routine *qsort* will “callback” your function when it needs to compare two items in the array.

Use a profiler to time how long it takes to do the sort. Try a few other sort routines and time them. If you have a compiler IDE which provides profiling tools, you might try it out. Otherwise, you may want to use the very sleepy profiler (<http://www.codersnotes.com/sleepy/>). Run your program and then run the profiler and select your programs process to profile. If the program runs too fast because you have got a fast processor, increase the size of the arrays. Alternatively, you can do the timing yourself. See the example code “*Timing Fibonacci.cpp*” which uses <http://en.cppreference.com/w/cpp/chrono>. Or see <http://en.cppreference.com/w/cpp/chrono/c/time> or <http://en.cppreference.com/w/cpp/chrono/c/clock> (has code example).

Put each part in separate subroutines, so that profiler can tell how much time was spent in each subroutine. See how the Fibonacci subroutine is timed in lab 9. Consider creating a ***Stopwatch*** class [[1]](#footnote-1)that provides a nicer interface for timing code executions than that provided in the file *Timing Fibonacci.cpp.*

Implement the merge algorithm (merge of sorted containers) given in PowerPoint file *Lect-25.ppt*. The two arrays to be merged are the ones from above. They need to be pre-sorted using one of the sorting algorithms (*sort*, *qsort*) above. Profile your code. You can use the timing routines as used in the sample code *Timing Fibonacci.cpp* in your new ***Stopwatch*** class[[2]](#footnote-2).

Consider how the BST insert and traversals routines take in function pointers to functions that you write. You will be implementing function pointers in this lab (exercise below) for use in Assignment 2. See the textbook chapter on *Binary Trees*, section on *Binary Tree Traversal Algorithms and Functions as Parameters*. The example file “*funcPtr.cpp*” explains and expands the textbook example.

**c.**

In the example code “*Timing Fibonacci.cpp*” plot the time taken to calculate each Fibonacci number. An easy way to do this, is to modify the *doFibn(unsigned n)* routine so that the pair of values of *n* and *elapsed\_seconds.count()* are printed to a file as CSV, with each pair on a line. Then use the spreadsheet’s plot tool to plot the graph.

What observation can you make about the relationship between *n* and the running time for this recursive implementation of Fibonacci. How does the time complexity[[3]](#footnote-3) increase as n is increased? Is it possible to make this routine more efficient?

**d. (needed for assignment 2)**

Convert your BST from laboratory 9 into a minimal and complete template BST. Use function pointers in the traversal routines. The file “*funcPtr.cpp*” shows how function pointers work. *Timing Fibonacci.cpp* uses function pointers and functions as parameters. Use the approach covered in the example source code files and textbook. You **cannot** use ***std::function*** (or any other similar routine from *std namespace*) for this lab and Assignment 2. You can use *std::function* in later units, but now you have to learn what is involved from the ground-up.

You may see more uses for function pointers, and you can use as you see fit and justify the usage. See the files *testTree.cpp* and *BinarySearchTree.h*. The file *testTree.cpp* shows how data from a tree traversal can be collected by client code. In this example, there is *class CollectU* that acts as a collector of *Unit* objects that are kept by the client code when the *BST* does a traversal. The code for collecting *Unit* objects is found in the routine *test2()* in *testTree.cpp*. The *test1()* routine does a print of the returned values and it is similar to the textbook example. You need to understand the textbook example of function pointer usage before you look at the routine *test2()* in *testTree.cpp.* So please review the textbook material first. Then review the code provided in the file *funcPtr.cpp*.

The routine *test2()* in the file *testTree.cpp* takes the idea further so that it can be used in the assignment.

When putting data into the BST, what methods or operators must exist for the data itself?

**Test** the BST to make sure all methods work. In particular, make sure that you can pass the BST to other routines by *value*, *const reference* and *reference*. Check that data values are as expected on pass/return to/from routines. Without making sure of that the BST passes these tests, your BST is not ready for use in Assignment 2.

**e. (needed for assignment 2)**Use the modified form of assignment 1 that reads from multiple data files. (Exercise. 4 of Lab 8).

Do not attempt to code this question until you have completed exercise 4 of Lab 8 and exercise d above. Your template BST from lab 9 must be completed and fully tested before you make the required changes to the BST in this lab.

The major design/programming requirements of Assignment 2 are as shown below. **Completion of the requirements below would mean that you have completed the bulk of assignment 2**. You need to show[[4]](#footnote-4) progress on Assignment 2 to your tutor.

Most of the assignment menu options from assignment 1 are reused. **Option 3** is replaced to make use of Lab 9, exercise c, the C++ function called ***sPCC*** that calculates the sample Pearson Correlation Coefficient. So instead of total solar radiation, ***sPCC*** calculation is carried out for a user specified month and combinations of two data fields.

The combinations are:

1. Average Wind Speed (S) and Ambient Air Temperature (T). This is called S\_T.
2. Average Wind Speed (S) and Solar Radiation (R). This is called S\_R.
3. Ambient Air Temperature (T) and Solar Radiation (R). This is called T\_R.

Before using ***sPCC*** in the assignment, check the calculation using a spreadsheet. Advice is provided here: <https://support.microsoft.com/en-us/office/pearson-function-0c3e30fc-e5af-49c4-808a-3ef66e034c18>

Check that your ***sPCC*** gives a similar result. This is easier than manual calculation as you are validating your ***sPCC*** routine against a validated formula.

Prompt the user for the month as a numeric value. So, for January, user will enter 1. So, if the user enters 1 at the Option 3 prompt, The output will be as shown below:   
Sample Pearson Correlation Coefficient for January   
S\_T: n.nn   
S\_R: n.nn   
T\_R: n.nn

where

n.nn is the actual calculated value for each sPCC.

Note that for this Option 3, the selected month applies to all years that have been loaded. So multiple years’ data for the user specified month.

**Data Structures for Assignment 2**

Think of how you can use your **BST with function pointers** **and map (*your own Map* or STL map variants, which include *multimap*)** in the modified assignment 1 to store data for processing as required by the Assignment menu options. The use of BST and map is \*mandatory\* for assignment 2, so think carefully and do not start coding until you have thought through the issues.

You are not told how to incorporate the BST or the STL map (or any of its variants) in your assignment 1 to make it into assignment 2, so whatever approach you come up with will require written justification including advantages and disadvantages of your approach. \*Vector is optional\*. If you decide to use Vector as well, write justification for why the Vector, BST and std::map (or map variants) together will work best in your estimation.

The data structures can be used in many ways and so we want to know the thinking behind your choice of usage. Write down what you are thinking and the relevant theory you have used to guide your thinking. Consider the advantages/disadvantages of your approach and other potential approaches.

Hint: There are issues that you must resolve before you incorporate the BST and map data structures. The write up of these issues and your resolution is part of the rationale for taking your particular approach. Ideal or perfect solutions may not be possible with various combinations of the required data structures as trade-offs would be needed. You need to explain what you are trading-off and why.

Wrapping the std::map (or variants) in your own minimal but complete **class Map** will lead to **bonus** **marks**. Examine the Stack and Queue classes that have been provided. You would have practice wrapping the std::vector in your own class Vector without exposing everything in std::vector.

Your own **class Map** must be minimal but complete, and you will need to defend your choice of public methods for the Map otherwise no bonus marks will be given. The primary behaviour of Map is to associate a key with its value. You get to decide and justify if you allow duplicate keys in your design. A duplicate key means the key has more than one distinct value.

When designing class Map, consider if client code would have to change if instead of wrapping std::map, you had to later change to your Vector inside your class Map.

Do not provide unnecessary Map methods, as **class Map** is a small class in terms of code that you write. You should be aware that the std::map is **not** a good guide for what is minimal but complete design.

Once you have made a decision to use **class Map**, std::map must not be used in part of application code.

Although not required, you can use STL data structures like stack, queue, linked list if they are wrapped in your own data structure classes[[5]](#footnote-5). Only your BST and std::map (or variants or **Map**) are mandatory data structures required for Assignment 2.

In lab 9 recursive routines were asked for in the BST, but in Assignment 2 both recursive and iterative implementations are possible. So, you can choose, but you need to explain why you took a particular approach by actual testing for suitability. Do not make things up as you go along as you now have access to sufficient theory/concept to guide the thinking process needed for assignment 2. Cite the theory/concept in your rationale for your approach. For deciding on recursive or iterative approaches, provide explanations using actual testing that you did when loading the Date objects into your BST.

Add the BST (and Map, if used) to the UML for assignment 1 to see how your design looks “on paper”. Doxygen will be used to extract your actual design from the final code.

If anyone does come up with a properly justified “perfect solution” for incorporating both BST and the map into Assignment 2, I will have no choice but to create a new improved assignment that does not have a perfect solution for next time.

Stress test your assignment by loading many copies of each of the data files. The file *data/data\_source.txt* will have the same file name listed many times for each file. This stress test would be having all the csv data files in *data* listed multiple times in *data/data\_source.txt.* Check both iterative and recursive implementations of your BST.

Write a test plan to test the various menu options. Actual expected output has to be calculated using a spreadsheet. This expected output is to be in the test plan table. See example of test plan in Topic 8.

Make you sure you have manually checked the expected output. This manual checking applies for all the menu options.

**f.**

Please read the Question and Answer (QandA) files for both assignments 1 and 2 (when available). The QandA file for assignment 2 can get updated very close to submission time, if someone asks a question then. All past QandA files, including ones for the labs are also relevant.

A good approach to take would be to always have a running version of the assignment. It may not fulfil all the requirements of the assignment, but at any point in time, some requirements would be completed, while other parts are stubbed. Remember the incremental approach covered in the workshops at the start of semester. So, use stubs for functionality that is incomplete.

If you have questions, ask early during class (lab/workshops/clinics). For email queries, it is not likely that turnaround time will be quick in the last week of semester/trimester, so all questions need to be asked by the end of the second last week of semester/trimester. If you attempt parts d and e of this lab now, you will come up with the questions to ask about assignment 2 early.

**g.**

Review the example Coding Standards document provided. Various organisations (or projects) have different coding standards. This particular one is for a safety-critical system. **What is Cyclomatic complexity? Work out the Cyclomatic complexity of your largest routine/method.**

Review the section on General Design (section 3) and relate to the approaches you are taking in the labs and assignment. See file *JSF-AV-C++rules.pdf.*

There are other software development advice/principles like **DRY**. How do such principles relate to SOLID? See question in exam revision pack. For a quick overview with worked example <https://mattilehtinen.com/articles/4-most-important-software-development-principles-dry-yagni-kiss-and-sine/>. If you have not been applying these ideas, you need to start immediately as advice was given in Lab 5.

**h.**

To manage code changes as code evolves, consider using a Version Control tool like **Git**. The folder *Git\_usage* has advice from the textbook’s author how to get started. The example used is for a web page (index.html), but you can use a simple main.cpp file to get started. See the file *Git Installing and Using Malik.pdf*. If you want a little background, chapter 1 of the file *Git ebook.pdf* would help. Chapter 2 covers most of the common usage you will need for a while. Start using *Git* now. You will not regret it as it is an investment in your career. It will also save you time when developing code in the assignment.

1. See discussion here <https://codereview.stackexchange.com/questions/186727/stopwatch-class-in-c.> It can be handy in timing other areas of your code. [↑](#footnote-ref-1)
2. See an example here <https://codereview.stackexchange.com/questions/186727/stopwatch-class-in-c> [↑](#footnote-ref-2)
3. See the section on “*Asymptotic Notation: Big-O Notation*” in the textbook chapter on *Searching and Sorting Algorithms*. [↑](#footnote-ref-3)
4. For the assignment 2 mark to be credited to you, your tutor needs to record that you have shown your work in person to your tutor prior to the submission of the assignment. [↑](#footnote-ref-4)
5. Queue and Stack have already been provided in previous labs. [↑](#footnote-ref-5)