ICT283 Assessed exercise 2

Objectives:

To learn

- BST concepts
- To construct a simple Binary Search Tree (BST)
- Demonstrate recursive vs iterative approaches
- Preparation for assignment 2.
- Exercise b must be submitted. It is assessed by personal demonstration/defence like Lab 4.

Lab 8 question 4 needs to be completed first.

It is very important not to fall behind with these exercises.

Exercise b is assessed by personal demonstration/defence.

Submit the exercise b into LMS. Demonstrate exercise b to your tutor during the following week's lab session. This is an in-person defence of your work.

Exercises

For this lab, use **recursive routines** for dealing with the tree. See PowerPoint notes. The textbook uses non-recursive (iterative) routines but for the lab, you need to write recursive routines. You may want to use the unit reference book *Introduction to Algorithms* by Cormen et al.

Assignment 2 can use iterative and/or recursive routines. You will need to justify your choice.

In the assignment, you would normally be asked to provide a rational for your data structures. In this video for BST justification one particular example is used. https://www.youtube.com/watch?v=9Jry5-82I68&index=5&list=PLUl4u3cNGP61Oq3tWYp6V_F-5jb5L2iHb. In the video, the tree algorithm is modified to cater for a new requirement. This approach is not acceptable – see Open Closed Principle. Think of a better solution. Other than that, the video explains the BST and its use very well.

a. (Preparation: before you start on exercise b)

Implement a simple (**non-templated**) Binary Search Tree (BST) class called **int*Bst***, which stores integers. Provide the following recursive operations: Insert, Delete tree, Search, inOrder traversal, preOrder traversal, postOrder traversal. Would Delete tree be public or private? Methods that are essential for the correct operation of the tree in terms of managing memory must be provided. You should go through the lecture notes (and the textbook for ideas).

Processing a node simply involves printing out the contents of the node, which in this case is the integer stored in the node.

Data should come from a data file containing integers. You decide on the format. The main program should open the data file and insert into the tree and demonstrate other tree operations.

The point of this exercise is to demonstrate that you understand the BST. There is no need to go overboard with it and put in operations not asked for that you will normally find in the textbook and the reference book, Introduction to Algorithms.

```
class intBst{…  //intBST.h

   /// declaration of methods and data members of class intBst with doxygen
        comments

};

// The implementation can go into intBST.cpp but for convenience only, keep the
```

```
// rest of the code in intBST.h with normal (non-doxygen) code comments
// The integer BST will be changed later to a template BST, and that is the reason
// for having everything in inBST.h now – convenience for the changeover to
// template BST.
```

To use your BST in your test program, you will have the following declaration in the main program's routine main().

```
intBst intTree;     // declaration of non-templated integer tree
                    // other code follows the above:
                      // insert various integer values
                      // test the tree methods
```

Make sure that the BST is tested properly. So that would involve passing it to subroutines by value and reference.

b. (demonstrating in class the following week – start work on it at home)**

Complete exercise *a* above before attempting this exercise for assessment. Recursive operations are needed. Please do not overreach and skip exercise a above. You would not be saving time.

This exercise is preparation for assignment 2. It does not mean what follows is how a Binary Search Tree (BST) is going to be used in your assignment. The purpose of this exercise is to get you started with a basic template Binary Search Tree needed for the assignment. Later labs will provide additional modifications.

Convert your BST from exercise *a* above into a **template Binary Search Tree (BST)**. Note that recursive operations are required for this assessed lab work.

You may use non-recursive versions in assignment 2 but you must know the recursive versions as well so that you can justify pros/cons of iterative vs recursive algorithms. Justification has to be from actual experience using the data provided in this unit, so provide actual comparison timings and explanation related to the data used.

Your *Bst.h* file for this exercise will now have:

```
template <class T>
class Bst{…

   /// declaration of methods and data members of class Bst with doxygen comments

};

// rest of the code in Bst.h with normal (non-doxygen) code comments
```

As this is a template class, there is **no *Bst.cpp***.

As *Node* stores the data, *Node* will also be implemented as a template.

Test your template BST with the integer data file from exercise a. Change the declaration in your test program (main) from exercise **a** above to:

```
Bst<int>  intTree;
        // rest of the code is the same as exercise a in the main program.
```

This is the only change. If further changes are needed, take note to evaluate the changes from both a design perspective and implementation perspective.

A read subroutine would read and load *dateTree* data structure shown below. Data you need is in the *data* folder that was provided in Lab 8. Your program will read all the files listed in *data/met_index.txt*. **Lab 8 question 4** must be completed to be able to do this exercise. Test your template BST by inserting just the Date values.

```
Bst<Date> dateTree; // Date is the date class from assignment 1.

Make sure you test all the BST operations on the dateTree
```

Would the Date class from assignment 1 work with the BST, or do you need to modify the Date class or provide additional routines/methods? What else is needed and why? Should the routines be methods of the Date class or helper routines in your Date namespace? Would the strategy design pattern[1] be useful? Explain your reasoning.

Create a separate text data file, *date.txt* that has only dates using the same date format as assignment 1. Use this data file to test your *dateTree*. The assignment data files are likely to have dates that are in sequentially increasing order. This new data file, *date.txt*, will have dates that are not in any order. Check that your tree works.

When submitting, there will be one solution (work space) with 3 projects in Codeblocks. All three projects will use the same BST.h containing your **template BST class**. There must be only one copy of BST.h in your solution.

The first project is the integer tree, *intTree,* the second project is the *dateTree* using Lab 8's data file. The third project would be *dateTree* using *date.txt*.

For the assessment, your tutor would normally ask you to demonstrate/explain details of your approach with the BST and its use in the *dateTree* data structure (2nd and/or 3rd project in the solution). You may also be asked to explain the first project using *intTree*.

Not assessed in this lab but assessed in later work including Assignment 2.
What would you have to do if the user (client) of your tree data structure would like to do something else instead of just printing the values? See textbook section on "Binary Tree Traversal and Functions as Parameters" in the chapter on Binary Trees. See the code file funcPtr.cpp after you have gone through the textbook section as the code builds on what is in the textbook.

Further practice with Recursion

1. Fibonacci sequence

   Fibonacci numbers are a series of numbers defined as shown below.

   - F(1) = 1 // first number
   - F(2) = 2 // second number
   - F(n) = F(n-1) + F(n-2) // subsequent numbers

   These numbers have been used in Mathematics and Computer Science, and some have argued that these numbers represent some patterns in nature (https://en.wikipedia.org/wiki/Patterns_in_nature). For a quick background to these numbers and their applications, see https://en.wikipedia.org/wiki/Fibonacci_number. The unit textbook, like most Computer Science texts, covers Fibonacci numbers in the chapter on recursion. The unit reference book Introduction to Algorithms has good coverage of algorithms and various issues including parallel algorithm versions of Fibonacci.

   For this exercise, two versions of routine F are needed.
   One version of the routine is **iterative** (uses loops). You need to implement this version.
   The second version of the routine is **recursive**. This recursive version is provided to you in the file *Timing Fibonacci.cpp*. Examine the code in this file.

[1] This pattern was required reading before.

i. Compare the algorithms of each version. Which version of the algorithm best matches the definition of Fibonacci numbers as given above? Review their algorithmic performance.

ii. Compare the runtime performance of each routine F, when n is large. How long would it take to calculate F(100) on your machine? On the machine in the lab?

iii. Plot the output n vs time taken. The log files would have the output data for the run. Compare the experimental results with the analysis of each algorithm.

2. Tower of Brahma (aka Tower of Hanoi)

Edouard Lucas, who is said to have coined the term "Fibonacci sequence", introduced a legend and invented an associated puzzle in 1883. The legend goes as follows:
"*In the great temple of Brahma in Benares, on a brass plate under the dome that marks the center of the world, there are 64 disks of pure gold that the priests carry one at a time between these diamond needles according to Brahma's immutable law: no disk may be placed on a smaller disk. In the beginning of the world, all 64 disks formed the Tower of Brahma on one needle. Now, however, the process of transfer of the tower from one needle to another is in mid-course. When the last disk is finally in place, once again forming the Tower of Brahma but on a different needle, then the end of the world will come and all will turn to dust.*" (Reproduced from an exercise in *Algorithms and Data Structures: The Basic Toolbox*[2] by Mehlhorn and Sanders). Our unit textbook also has this example in the chapter on recursion. This puzzle got mathematicians, and later computer scientists very interested. Consequently, algorithms for solving this puzzle have been widely studied.

The following can be done after you have completed all assessments in the unit.

- Create the puzzle using everyday items (buttons, bottle caps …etc.) as described at https://www.scientificamerican.com/article/the-tower-of-hanoi/ and work out an algorithm for solving the puzzle. Use the rule as described in the legend above.
- Compare your algorithm with the worked example in the unit textbook (chapter on Recursion)
- Implement the algorithm in C++ and try solving the puzzle for a few disks.
- If the priests in the story had a computer such as yours and attempted to solve the problem in simulation, how long would it take to transfer all 64 disks from one needle to one of the other needles?
- If the priests had to move each disk manually at the rate of one disk per second with no breaks, how will long will it take them? Compare this time with the age of the universe as we know it.
- What can you say about the complexity of this problem using recursion? Consider what happens when there are only 3 disks in the problem, and then as the number of disks is increased to 64. Would it be a polynomial time problem?

If you are after some "light" reading during the inter-semester break, try The Tower of Hanoi – Myths and Maths by Hinz, Klavzar, Milutinovi and Petr. You may discover the origin of some of the concepts you have studied before and will study later.

[2] It is a good book.