Linköping University
Department of Science and Technology/ITN
Aida Nordman

Data Structures
TND004
Lab 3

Goals

- To implement a **binary search tree class** supporting bi-directional iterators.

Prologue

In this lab exercise, you are requested to make some modifications and add some extra functionality to the (template) class **`BinarySearchTree`** presented in the course book, section 4.3. The code for this class can be downloaded from the course website.

As in lab 1, you are going to use again smart pointers.

Preparation

You can find below a list of tasks that you need to do before the **HA** lab session on week 19.

- Review Fö 6, where binary search trees were introduced.
- Read sections 4.1 to 4.3 of the book. Pay special attention to section 4.3 (you can safely skip reading section 4.3.6 for this lab).
- Review the notes of lesson 3.
- Downloaded the code for the (template) class **`BinarySearchTree`** from the from the course website. This class is described in section 4.3 of the course book.
- Do exercise 1. The lab assistant can give feedback on your solution in the beginning of the lab, since the remaining exercises of this lab build upon this one.

Presenting solutions and deadline

You should demonstrate your solution orally during your **RE** lab session on **week 20**. After week 20, if your solution for lab 3 has not been approved then it is considered a late lab. Note that a late lab can be presented provided there is time in a **RE** lab session.

The necessary requirements for approving your lab are given below.

- Use of global variables is not allowed, but global constants are accepted.
- Readable and well-indented code. Note that complicated functions and over-repeated code make programs quite unreadable and prone to bugs.
- There are no memory leaks. Recall that you can use one of the tools suggest in lab 1.
- The code does not generate compiler warnings[1].

[1] **`g++`** is the reference compiler.

- The only modifications allowed to the public interface of class **`BinarySearchTree`** are the ones explicitly described in this exercise. However, you are allowed to add extra private members to the class, if needed.
- The use of built-in pointers, as well as the **`new`** and **`delete`** operations, are strictly forbidden.
- Bring a written answer to the question in exercise 5, with indication of the name plus LiU login of each group member. You'll need to discuss your answer with the lab assistant who in turn will give you feedback. Hand written answers that we cannot understand are simply ignored.

If you have any specific question about the exercises, then send us an e-mail. Be short and concrete, otherwise you won't get a quick answer. You can write your e-mail in Swedish. Add the course code to the e-mail's subject, i.e. "**TND004:** ...".

Finally, presenting your solution to the lab exercises can only be done in the **RE** sessions.

Exercise 1: replace built-in pointers by smart pointers

- Replace all built-in pointers in the code for the given **`BinarySearchTree`** class, by smart pointers. And by the way, since we are using smart pointers the destructor and member function **`makeEmpty`** become simpler, right?
- Add to each node a pointer to the parent node. Make the necessary changes in the given code for the insertion/removal of a value in the tree. The private member function **`clone`**[2] needs also to be modified.
- Add a private member function to display the tree using pre-order[3] with indentation, as shown in the example **`test1_out.txt`**. This function should then be called from the public member function **`printTree`**.
- Add a (test) public member function **`get_parent`** that, given a value x, returns the value stored in the parent of the node storing x, if a parent node is found. Otherwise, **`T()`** is returned, where **`T`** is the type of x.
- Test your code with the program given in **`test1.cpp`**. The expected output is provided in the file **`test1_out.txt`**.

Exercise 2: find the predecessor and the successor of x

- Add to the **`BinarySearchTree`** class a member function **`find_pred_succ`** that, given a value x, returns two values **`a`** and **`b`**, such that **`a`** is the largest value stored in the tree smaller than x and **`b`** is the smallest value stored in the tree larger than x.
- Test your code with the program given in **`test2.cpp`**. The expected output is provided in the file **`test2_out.txt`**.

[2] The private member function **clone** is called by the copy constructor to perform the actual copying work.
[3] The given code uses an in-order traversal of the tree and, consequently, it displays all values stored in the tree in increasing order.

Exercise 3: add a bi-iterator class

- Add a class **`BiIterator`** that represents (simplified) bi-iterators for the class **`BinarySearchTree`**. This class should overload the usual iterator operators: **`operator*`**, **`operator->`**, **`operator==`**, **`operator!=`**, pre and pos-increment **`operator++`**, and pre and pos-decrement **`operator--`**. The class **`BiIterator`** should also have a default constructor and a constructor to create a bi-iterator given a (smart) pointer to a tree's node.

  Define **`BiIterator`** as a public nested class of **`BinarySearchTree`** class.

- Add two member functions of class **`BinarySearchTree`**, **`begin`** and **`end`**. Function **`begin`** returns a **`BiIterator`** to the node storing the smallest value of the tree, while **`end`** returns **`BiIterator()`**.

- Modify the member function **`BinarySearchTree::contains`** such that it returns a **`BiIterator`** (instead of a **`bool`**).

- Test your code with the program given in **`test3.cpp`**. The expected output is provided in the file **`test3_out.txt`**.

Exercise 4: build a frequency table of words

In this exercise, you are requested to write a program that displays a frequency table for the words in a given text file. For simplicity, you can assume that all words in the file consist of lower-case words and there are no punctuation signs.

You must use a binary search tree, i.e. an instance of class **`BinarySearchTree`**, to represent the frequency table and use bi-iterators (i.e. instances of class **`BinarySearchTree::BiIterator`**) to traverse the tree.

Each row of the frequency table contains a key (**`string`**) and a counter. Thus, you need first to define a class that represents each row of the table. This class should also overload **`operator<`**.

Test your code with the file **`words.cpp`**. The expected frequency table is available in the file **`frequency_table.txt`**.

Exercise 5

The answer to the following question must be expressed in your own words.

- Indicate the time complexity of the operations **`BiIterator::operator++`** and **`BiIterator::operator--`**, in the best and worst cases. Use Big-Oh notation and **motivate clearly** your answer.

Remember to write your answer in paper, preferably computer typed, and do not forget to indicate the name plus LiU login of each group member. Deliver your written answer to the lab assistant, when you present the lab.

Lycka till!