CSC2040 Data Structures and Algorithms, and Programming Languages

Practical 8

Wednesday 6 December 2017

Binary Trees and Recursion

The learning outcome of this practical is to be confident in using a standard data structure (a tree node class), to be able to add new facilities to the library class, and to be able to use recursion to perform calculations.

Create a new project called Practical8, with the file test.cpp for your main function. Include the supplied TreeNode.h class into your project.

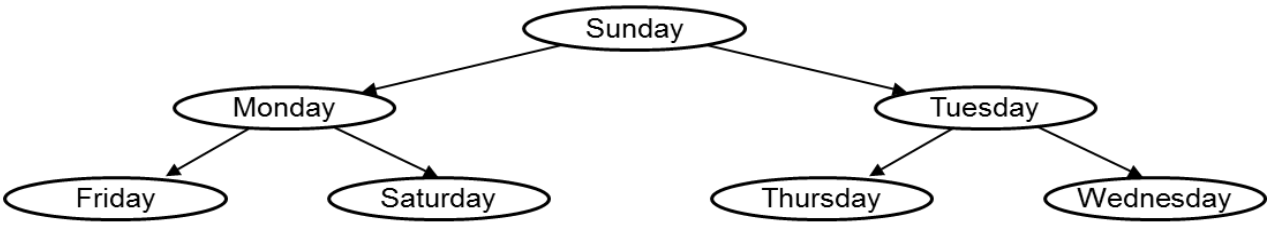
Part 1: Extending the TreeNode class

Add a function to your TreeNode class (in TreeNode.h):

```
template <typename T>
bool TreeNode<T>::searchTree(T key)
```

which searches the tree for the given item 'key', and returns 'true' if the key was found in the tree, and 'false' if it wasn't. Assume the initial tree has at least one item.

In your test program, construct a binary tree of week days as below (you can construct it in one rather lengthy statement, if you like):



Test your function by searching the above tree for a number of cases:
- search for all seven days
- search for several "days" which aren't in the tree. Try to break it!

For each search, print out its details: what you were searching for, and whether or not you found it.

Part 2: Recursion

1) The series of so-called Fibonacci numbers goes like this:

Fib = 1 1 2 3 5 8 13 21 …

First, define this series using a function Fib(N) to define the N^{th} number in the series. Don't use the '…' shorthand – be precise.

Now in the test.cpp write a function

int **Fib** (int N)

which returns the value of the N^{th} Fibonacci number. Base your code on your definition. Don't use a loop – use recursion.

Test your function by calling it with lots of different values and printing the result each time.

2) In the test.cpp write a template function:

```
template<typename T>
int depth (TreeNode<T>* tree)
```

which returns the depth of a binary tree 'tree'. Remember that the tree may not be balanced. The depth of an empty tree is 0.

Note, you can make **depth** to be a friend function inside the TreeNode class by including the following:

```
template <typename T>
friend int depth(TreeNode<T>* tree);
```

to grant **depth** the access to the class's private members (e.g., left, right). See C++ Lecture 3 for friend functions.

Test your function using the trees you constructed in Part 1, and some others.

3) In test.cpp write a function:

int **sigma** (TreeNode<int>* tree)

which returns the sum of all the (int) values in a binary tree of integers. Again, you can make **sigma** to be a friend function of the TreeNode class to grant it the access to the class's private members.

Test your function by using properly constructed integer trees.

Practical Test 3

Practical Test 3 will take place during the practical class on Wednesday 13 December. You will be asked to develop C++ programs to solve a given set of problems, and then to use the Assignment Tool in Queens Online to upload your answers. The test is open book, but with restrictions. See the Course Information for the "Format for practical tests and final examination". Pen and paper can be used for working out / rough work.