CSC2059 Data Structures and Algorithms

Practical 6: Binary Trees and Recursion

Wednesday 17th November 2021

Launch Visual Studio. Create a new Solution Practical6 and name the first project Recursion.

Project 1. Recursion

The sum of the reciprocals of square numbers produces a convergent series (the Basel problem), with a limit equalling $\pi^2/6$ (≈ 1.64493):

$$\frac{1}{1}+\frac{1}{4}+\frac{1}{9}+\frac{1}{16}+\frac{1}{25}+\frac{1}{36}+\cdots=\frac{\pi^2}{6}.$$

The following shows a function to calculate the sum of the first n reciprocals by using a loop:

```
double rsquare(int n)
{
	double sum = 0.;
	for (double i = 1; i <= n; i++)
		sum += 1. / (i * i);
	return sum;
}
```

You are asked to implement the calculation by using a recursion. In the Recursion project create a test program test1.cpp. Write your recursive function in test1.cpp and test it in the main function by calling it with variable values for n.

In the following, we will try to implement all the operations on tress by using recursions.

Project 2. Extending the TreeNode class

In Solution Practical6 add a new project Tree1. Include the provided binary-tree class TreeNode.h into the project.

(1) Add a member function to the TreeNode class (in TreeNode.h):

```
bool search(T key);
```

which searches the binary 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 the project, create a test program test2.cpp. In the program construct a binary tree of week days as shown in Fig.1 below:

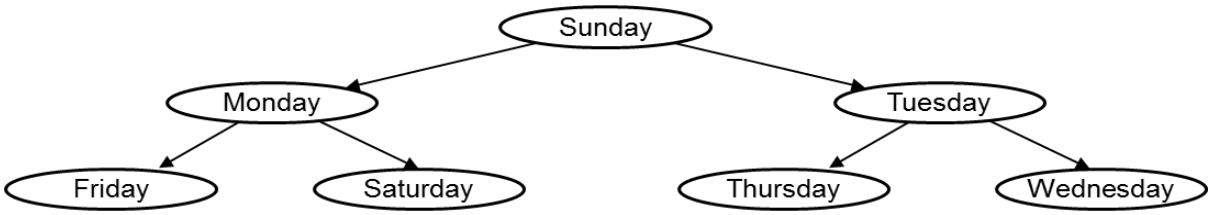

Fig.1 A week-day binary tree

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.

(2) Add a member function to the TreeNode class:

```
int depth();
```

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

In test2.cpp test your function using the trees you constructed in (1), and some others (e.g., below).

Project 3. Binary search trees

In Solution Practical6 add a new project Tree2. In the project, create a test program test3.cpp.

(1) In test3.cpp write a function:

```
TreeNode<int>* bst();
```

Inside this function you construct a balanced binary search tree (BST) using dynamic allocation, for the random number sequence: 6, 23, 12, 69, 17, 29, 7. The function returns the pointer to the tree. You should use the "search with insertion" method described in the lecture notes to construct this tree.

(2) In test3.cpp write a function:

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

The function is called with a binary integer tree. It takes the tree and calculates the sum of all the (int) node values inside the tree; it returns the sum after finishing execution.

(3) To further your experience with tree recursions, in test3.cpp write a function:

```
int maxnum(TreeNode<int>* tree);
```

The function takes a binary integer tree and finds the maximum number inside the tree. It returns the number found after finishing execution.

Finally, in test3.cpp write a main function to test all the three functions. You first call the bst function to construct and obtain the BST. Then you use this BST to test the other two functions sigma and maxnum by printing out the results returned from the functions.

If you have any questions, ask a demonstrator or a teaching staff face to face in the labs, or online by using MS Teams, during the timetabled practical session time. At any other time outside the timetabled session time, you can send your questions by using the Ticketing System, or by using emails.