Object-Oriented Programming in C++

Lab Exercise 1 (2%)

Objective

The objective of this exercise is to get up-to-speed with C++, including declaring and defining classes, operator overloading, and input/output processing.

You can work on this exercise in a group of two. Hand in the code for part one at the end of the lab, and part two by the start of tomorrow's lab.

Description

Your task is to implement a class for manipulating the states of a *Towers-of-Hanoi* puzzle, and write a program for solving it.

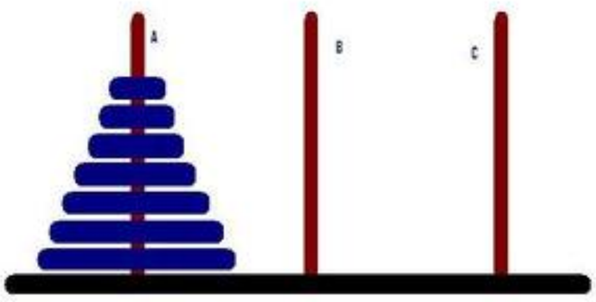

Image from shaunhall.co.uk

PART I

a) Create a class *TowersOfHanoi.* Use a separate header file (*toh.h*) for the class declarations and a C++ file (*toh.cpp*) for the class declarations. Also create a main program (*main.cpp*) for testing the class.

b) Add a constructor to the class. The constructor takes three arguments, the first indicating the number of *pegs*, the second the number of *disks*, and the third the *number of the peg* to place the disks on. The disks will be placed in a successively decreasing size on the indicated peg. The last argument is optional and, if omitted, the disks will be placed on the leftmost peg.

c) Write a method for returning the number of pegs in the puzzle (int get_num_pegs()) and the top-most disk on a peg (int get_topmost_disk(int peg)). For the latter method, if there are no disks on the peg the value nodisk (which you should defined as *-1*) should be returned.

d) Write a method for moving a top-most disk from one peg to another one, for example, `move(0, 2)`, would move the top-most disk from peg 0 to peg 2. Note that for this method there is no restriction on the size of the moving disk, that is, a larger disk can be placed on the top of a smaller one. However, the method should be used legitimately, that is, the peg indices should be in the range *0* to *number of pegs minus one*, and there should be a disk on the source peg. If not, the method should assert.

e) Overload the == operator. Two objects are the same if they have the same number of pegs with an identical disk arrangement.

f) Overload the << operator (as a friend). The output format should be as follows:

```
[ 6 5 4 3 2 1 0 ] [ ] [ ]
```

The output should be in a single line (no newline at the end), showing the pegs in a left-to-right order, with the disks on each of the pegs listed within square brackets. The disks are numbered from *0* to *number of disks minus one*, 0 being the smallest disk. The bottom-most disk on the peg is listed first and so on. Note: The output must be exact: disks are separated by a single space, pegs by two spaces, and there is one space between empty brackets.

Finish this part of the exercise in the lab and submit via *MySchool* (*toh.h* and *toh.cpp*).

PART II

The goal when solving a Tower of Hanoi puzzle is to get all disks from the left-most peg to the right-most one. Only one peg can be moved at a time, and a larger disk *cannot* be placed on top of a smaller one.

a) Write a brute-force program (e.g., *iterative depth-first-search*[*]) for solving small puzzle instances, using 3 pegs and a small number of disks (try your program with 1, 2, 3, and 4 disks). Take the number of pegs and disks as a command-line argument. The program should print out the *start* and *final* states of the puzzle, the moves the solution is composed of, the total number of moves your algorithm performed in total (not only for the solution, but all tries), and the time (in milliseconds) it took to solve the puzzle. An example output for a 2-disk puzzle could be as follows:

```
[ 1 0 ] [ ] [ ]
[ ] [ ] [ 1 0 ]
(0 1) (0 2) (1 2)
Number of moves explored: 8
Time in milliseconds: 20
```

b) Write a more intelligent program for solving the puzzle. This part is optional with no extra points, but *a matter of pride and glory!*

You have until 11pm to finish part II of the exercise and submit via *MySchool*.

[*] https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search