ITP 365: Managing Data in C++

Homework 3: Tower of Hanoi   Due: Friday, 9/29/2017 @ 11:59PM

Goal

Use the graphical capabilities of the Stanford C++ library to animate a recursive solution to the Tower of Hanoi problem. You will find a description, and a small animation, of the Tower of Hanoi in Wikipedia. The link is http://en.wikipedia.org/wiki/Tower_of_Hanoi.

Lab Setup

- Download the starter project from Blackboard.
- All your code files must begin with comments in the following format (replace the name/email with your actual information):
```
// ITP 365 Fall 2017
// HW03 – Towers of Hanoi
// Name: Tommy Trojan
// Email: ttrojan@usc.edu
// Platform: Mac
```
- The project already has a **main.cpp** file with almost no code in it. For this project you will need to complete **main.cpp** and create two C++ classes (you will need to write both the .h and the .cpp files):
    - **Disk** – This class will represent an individual Disk in the simulation
    - **Peg** – This class will represent an individual Peg in the simulation
- The Stanford library uses pixels to measure all graphics. Therefore, almost everything on the screen is measured as whole numbers (or integers). Be sure to use integers when dealing with graphics and on screen objects.

Part 1: Disk class

- We will be moving Disks around or window for this assignment. For our purposes, a Disk is just a rectangle.
- Your Disk class must have at least the following member variables. You may store more data with each Disk, but the following list is required at a minimum:
    - The Disks x and y coordinates. It is a GoodIdea™ to have coordinates be the bottom center of the Disk. This will make centering different size Disks much easier. The size is in pixels.
    - The width and height of the Disk, in pixels
    - The color of the Disk (stored as a string).
- Your Disk class is to have at least the following member functions. You may have more, but this list is the minimum:
    - A default constructor and a parameterized constructor that sets x, y, width, and height for the Disk.
    - Setter functions for the x and y coordinates.
    - Getter function for the height of a Disk.

    - A *draw* function that accepts a GWindow (by reference). This function will use the GWindow and the member variables to create the colored rectangles that represent your Disks.
- It is a GoodIdea® to test your Disk class by creating a few Disks around the main window. Then call the *draw* function to have them drawn. Make sure your Disk class and *draw* function work properly before going to part 2. You do not need to keep this test code in main once you start part 2.

Part 2: Peg class

- Pegs are the other visual component of our simulation. A Peg consists of a thin, black, vertical rectangle with zero, or more Disks.
- A Peg must have at least the following member variables. You may store more data with each Peg, but the following list is required at a minimum:
    - Vector of Disks member variable which will contain all the Disk objects that are "on" that Peg.
    - The Peg's x and y coordinates. It is a GoodIdea© to have coordinates be the bottom center of the Peg. The Peg is a tall, skinny rectangle.
    - The width and height of the Peg, in pixels
    - The color of the Peg (stored as a string).
- Your Peg class is to have at least the following member functions. You may have more, but this list is the minimum:
    - A default constructor and a parameterized constructor that sets the x, y, width, and height for the Peg.
    - A *draw* function that draws the Peg and all of the Disks that are "on" that Peg. The Disks are to be drawn centered on their Peg. To display the Peg, create the rectangle using the GWindow to represent the Peg. Then call the *draw* function for each Disk on the Peg (using the items in the Disk Vector)
    - An *addDisk* member function. This function takes a Disk object as input. The Disk object that is passed into this function is to be added at the end of the Vector (for performance). This means that the first Disk in a Vector (at index 0) is the bottom Disk on that Peg. The Disk object that is last in the Vector is the top Disk on that Peg.
    - A *removeDisk* member function. This function is to remove the last Disk on that Peg and return it.
- As with Part 1, it is highly recommended that you modify main to test your Peg class code. Create a few Peg objects, add Disks to them, remove Disks from them, drawing your Peg objects after each change. Make sure your code works before continuing the next step.

Part 3: Towers of Hanoi setup

- Before we can implement the recursive solution, we must prompt the user for input and setup the objects. We will always have three Pegs, but the user can specify the number of Disks, as well as the starting and ending Pegs.
- Inside of the main file, create the following functions:

    - *promptForDisks*. This function uses std::cout to prompt the user for the number of Disks to start with. The valid range is 2 to 10 (inclusive) Disks. Anything else is to be rejected. Continue to prompt the user until they enter a valid number of Disks. Use std::cin to read user input. It will return the number of Disks the user requested.
    - *promptForPegs*. It receives 2 integers (by reference) as input. Using std::cout, prompt the user for a starting Peg and ending Peg numbers. The valid values are 1, 2, or 3. You are to reject anything else. You are also to ensure that the starting and ending Peg numbers are not the same. Use std::cin to read user input. Store the starting Peg value in the 1st input and the ending Peg value in the 2nd input. The function returns nothing.
    - *draw*. It accepts a GWindow (by reference) and the vector of Pegs (by reference). Use the *clear* function from GWindow to clear the graphics currently in the window. Then write your *draw* function to draw all the Pegs (and therefore all the Disks on the Pegs). You might want to use the *pause* function (which accepts the number of milliseconds to wait) at the end of draw. This can simulate animation, eventually.
- Once you have read in all the user input, create the GWindow with the 3 Peg objects. It is a GoodIdea™® to store them in a Vector, as it makes drawing all of them easier than having three separate variable names.
- Create the appropriate number of Disk objects and place them on the correct starting Peg.
- Test your code before going to Part 4. Now your code in main will not be replaced in Part 4, so testing at this point isn't "a waste of time". It doesn't matter if your recursive solution works if you can't draw Pegs and Disks properly.

Part 4: Implement the recursive solution

- The key to solving this puzzle is to recognize that it can be solved by breaking the problem down into a collection of smaller problems and further breaking those problems down into even smaller problems until a solution is reached. One description of the solution might be:
  1.1. Label the Pegs "start", "temp", "end" — these labels may change at different steps
  1.2. Let N be the total number of Disks
  1.3. Number the Disks from N-1 (smallest, topmost) to 0 (largest, bottommost)
  1.4. Move N Disks from "start" to "end":
      1.4.1. Transfer N−1 Disks from "start" to "temp". This leaves Disk 0 alone on "start"
      1.4.2. Move Disk 0 from "start" to "end"
      1.4.3. Transfer N−1 Disks from "temp" to "end" so they sit on Disk 0
- Create a *moveDisk* function that accepts the GWindow object (by reference), the Peg collection (by reference), the starting Peg (represented with an int), and the destination Peg (represented with an int). It must remove the top most Disk from the start Peg, add it to the destination Peg, and then call the *draw* function from main.
- Create a *towerSolver* function in main.cpp. This function is to take five arguments: the GWindow object (by reference), the Peg collection (by reference), the starting Peg (represented with an int), the destination Peg (represented with an int), and the number of disks to move.
- The *towerSolver* is to use recursion and the *moveDisk* function to recursively move all the Disks from the source Peg to the target Peg.
- Call *towerSolver* one time in main to start the recursive solution.

A Note on Style

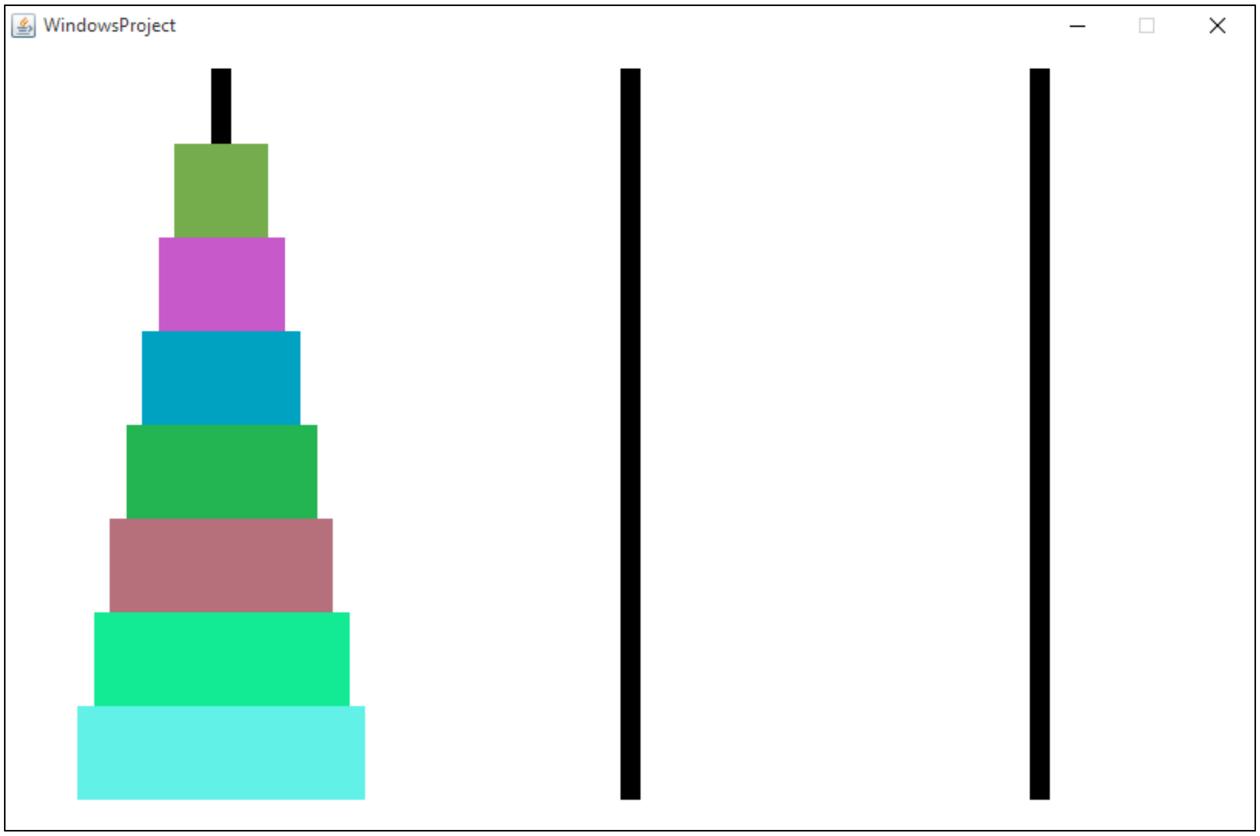
Be sure to comment your code.

As we discussed in lecture, it is extremely important that your code is properly indented as it greatly adds to readability. Because of this, if you submit a code file that is not reasonably indented, you will have points deducted.

Likewise, you will lose points if your variable names are not meaningful. Make sure you use variable names that correspond to what you are actually storing in the variables.

Sample output

A sample execution of the assignment, with 7 Disks with a start Peg of 3 and an end Peg of 1 might look like this. NOTE: For full credit, you do not have to color each disk a different color – but it's nice to look at this way, isn't it?



Deliverables

1. A compressed **Hw03** folder containing ***all of your .h and .cpp files***. It must be submitted through Blackboard.

Grading

Item	Points
Part 1: Create a Disk Class	15
Part 2: Create a Peg Class	25
Part 3: Initial Setup for Tower of Hanoi	20
Part 4: Implement Recursive Solution	30
Comments and Coding Style	10
Total*	**100**

* Points will be deducted for poor code style, or improper submission.