American University of Beirut
Department of Electrical and Computer Engineering

EECE 330 Data Structures
Fall 2014-2015
Midterm – October 22, 2014
4:00 – 6:00 pm

Course Instructor: Prof. Hazem Hajj
Lab Instructors: Zaher Kanafani, Ali Moukalled, and Youssef Jaffal.

Name :_____

ID:_____

Section:_____

Instructions:

- Write your name on the question sheet
- Place your IDs on the table.
- You are allowed ONLY textbook material as posted on Moodle.
- You can find all the needed base code on Moodle.
- Phones are STRICTLY FORBIDDEN!!!
- When done, make sure you have all the files uploaded to Moodle.
- Give yourself sufficient time to upload the files at the end. You will need to make two uploads:
    - Upload 1: The programming part upload for questions 1 to 5. Make sure you include ALL the .cpp and .h files in a .Zip folder. Name this folder with your full name and upload it into the slot called midterm programming part upload.
    - Upload 2: The written part upload. Please upload the word file into the slot called midterm written part upload.
- Do NOT forget any file, and do NOT upload a wrong file!! You are responsible for correctness of the uploads.

1. (50 pts + 10 pts Bonus) PROBLEM 1: BST and related…

In this problem you are asked to create and manipulate a dictionary of novels. You will need to implement the following functions (specified a-g). You will also need to test each of them. As you progress through the implementations, update the main() program to test the implemented functions: instantiating a class *novel*, incrementing price of a novel, filling a dictionary of novels using a BST, computing BST height, computing BST count of nodes, copying BST to an array list of novels, and conducting a search for a novel based on title.

a. (15 pts) Declare a **class novel** with private number: ISBN (string), title (string), price (double). Include:
   - A constructor with default values. Use "NA" as default string for title and ISBN = -1.
   - Set functions for all parameters
   - Readers for all parameters.
   - Overload the output operator.

b. (6 pts) Write **comparator classes,** to help search the BST according to title.

c. (15 pts) Write a function **"fill_bst"** to create a dictionary of novels, using a BST data structure and the BinNode (and BinNodePtr) structures discussed in class. The function inputs are: file name, the BST to be populated. The function returns Boolean indicating success (or failure) in opening and reading the file. A text file "novels.txt" is provided on moodle to test the function and fill the BST with the specified novels. A sample entry for novel in the file looks as follows. The first line is the ISBN, the second line is the novel title and the third line is the price. You can assume no errors in entries, but we don't know how many entries we have. Here is a sample entry:

   0439708184
   Harry Potter and the Sorcerer's Stone
   16.04

Note: Make sure you account for reading the title having spaces. Remember to clear the buffer to avoid any read problem from file.

d. (8 pts) Write a function **bstHeight()** to compute the height of a BST tree.

e. (6 pts) Write a function **bstCount()** to compute the number of nodes in a BST tree.

f. (10 pts – **Bonus Challenge Question)** Write a function "copy toAList" that takes as input a BST, and fills an arraylist (AList) with the nodes in the tree in sequence starting. Hint: You will need the BST height and the BST count of nodes.

2. (50 pts) PROBLEM 2: Stacks and … Generalized Tower of Hanoi Game:

For this problem, your job is to generalize the Hanoi algorithm so it can deal with any stack of items. See appendix for the original Hanoi problem. If you are not familiar with the tower of Hanoi, you can read more about in the appendix.
We will assume we can have any list of items (not just disks), and that can be sorted (e.g. sorting novels based on price). The puzzle starts with the items in a

neat stack in ascending order of size in one list, the smallest at the top, thus making the equivalent of a conical shape.

The objective of the puzzle is to move the entire stack to another list, obeying the following simple rules:

- Only one item can be moved at a time.
- Each move consists of (popping) taking the top item from one of the stacks and (pushing) placing it on top of another stack i.e. an item can only be moved if it is the top of the stack.
- No item may be placed on top of a smaller item in the stack.

You are asked to write the code to create and test the generalized Hanoi game. You will use the class novel and "novels.text" to test your implemented functions. As you write the functions, update main to test the functions.

a. (15 pts) Write a function **fill_stack()** that takes as input the file "novels.text" (same one mentioned in other problem), and fills in a stack of novels in descending order of price, with the top of the stack being the cheapest.

b. (7 pts) Write a function **move_OneItem()** that as input two stacks (by reference) of any types, and moves one item from the top of one stack to the top of the other stack. Test the function by having two stacks of novels, and moving one items from one stack to another.

c. (20 pts) Implement the function **general_hanoi()** for the generalized Hanoi game. Hint: Here is an illustration of what the generalized recursive function would do.

Function: **general_hanoi( )**

Input: Three stacks by reference: A, B, and C. Initially, A has the items. B and C are empty.
Output: Does not return a value (returns void). Moves the items from A to B.

Algorithm:

```
1. if(n> 0) // If there is more than one item in the stack
2.  {
3.      Call the general_hanoi (n-1,stack_A, stack_B, stack_C);
4.        Call the move_OneItem() function to move one item from stack A to stack C.
5.      Call the general_hanoi (n-1,stack_B, stack_C, stack_A);
6.  }
```

To move *n* items from Stack_A to Stack_C:
Step 1: move *n*−1 items from A to B. This leaves item *n* alone in stack A

Step 2: move item *n* from A to C
Step 3: move *n*−1 items from B to C so they sit on item *n*

d. (8 pts) **Analyze** the performance based on the original Hanoi algorithm, and derive the resulting performance. Make your main program print out:
    a. The size of the initial stack, and the number of steps it took to solve the game.
    b. Your conclusion on the asymptotic performance of the generalized Hanoi function.
    c. Your justification for the answer by first stating the equation for the performance of the algorithm (T(n) in terms of T(n-1)), then expanding to conclude.

For your reference (if needed), useful equations:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}. \quad (2.1)$$

$$\sum_{i=1}^{n} i^2 = \frac{2n^3+3n^2+n}{6} = \frac{n(2n+1)(n+1)}{6}. \quad (2.2)$$

$$\sum_{i=1}^{\log n} n = n \log n. \quad (2.3)$$

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1-a} \text{ for } 0 < a < 1. \quad (2.4)$$

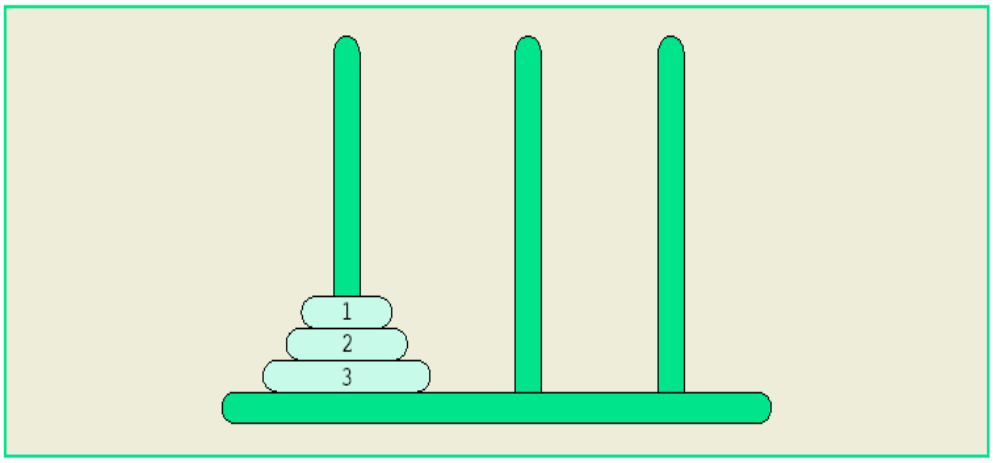
$$\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1} \text{ for } a \neq 1. \quad (2.5)$$

Divide and Conquer:

- $T(n) = a\ T(n/b) + c\ n^k\ ; \quad T(1) = c$
- Assuming: $n = b^m$; By expanding, we get:
- $T(n) = c\ a^m \sum^m_{i=0} (b^k/a)^i$
  - If $a > b^k$ $T(n) = \theta(n^{\log_b a})$;
  - If $a = b^k$ $T(n) = \theta(n^k \log n)$; (e.g. Merge sort: k = 1, a = b = 2.)
  - If $a < b^k$ $T(n) = \theta(n^k)$;
  - Proof (p. 480-82); 3rd edition.

Appendix - Original Tower of Hanoi Problem

The Tower of Hanoi is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes, and which can slide onto any rod as shown in the figure.



Tower of Hanoi problem with three disks

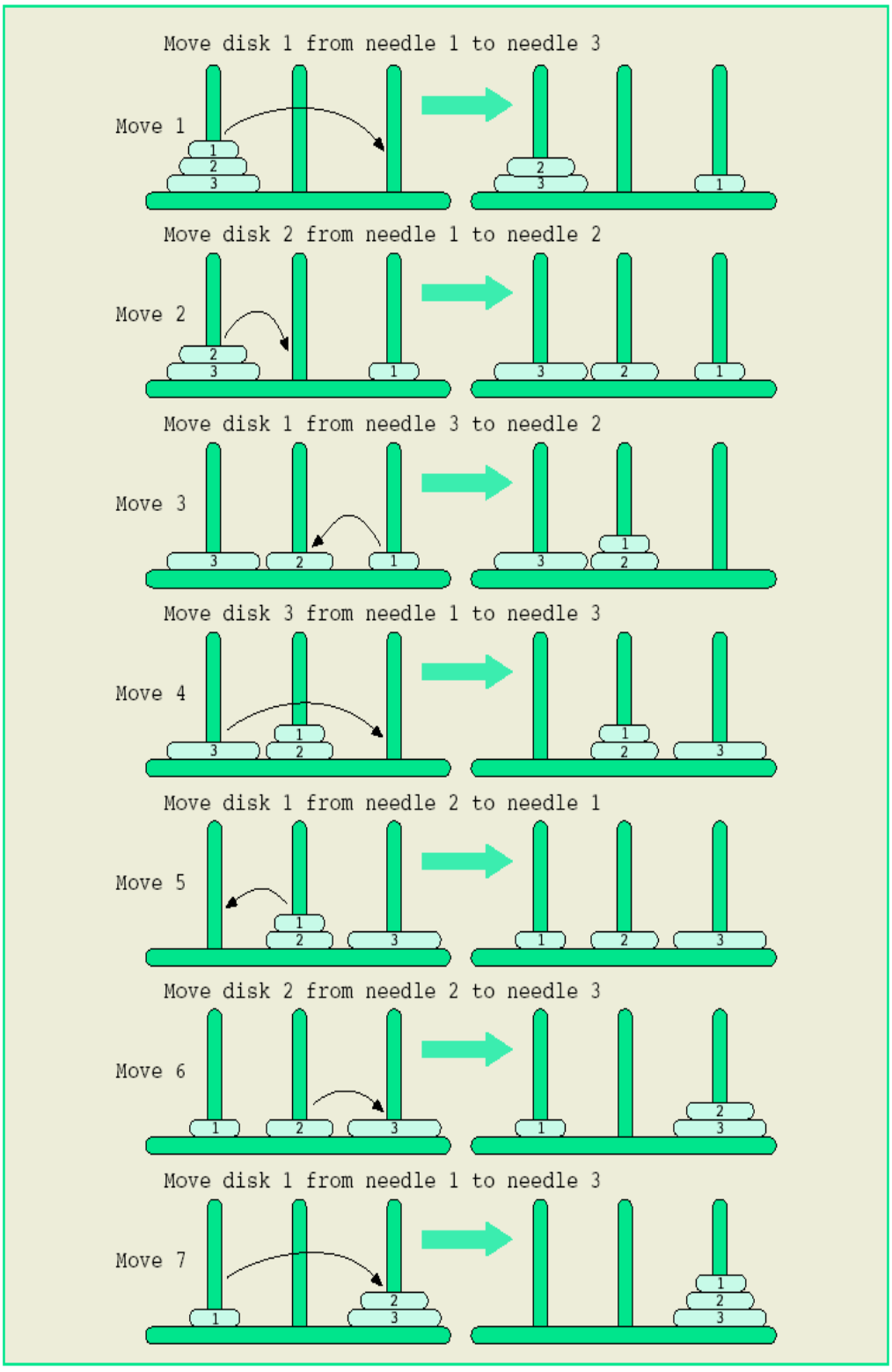
The objective of the puzzle is to move the entire stack to another rod (needle), obeying the following simple rules:

- Only one disk can be moved at a time
- The removed disk must be placed on one of the needles
- A larger disk cannot be placed on top of a smaller disk

The recursive function translates into the following:

```
void moveDisks(int count, int needle1, int needle3, int needle2)
{
   if (count > 0)
   {
       moveDisks(count - 1, needle1, needle2, needle3);
       cout << "Move disk " << count << " from " << needle1
            << " to " << needle3 << "." << endl;
       moveDisks(count - 1, needle2, needle3, needle1);
   }
}
```

Here is an illustration of how the game works with three disks:



Solution of Tower of Hanoi problem with three disks