CSCI 2110 Data Structures and Algorithms
Laboratory No. 7
Week of October 30- Nov 3, 2023

Due: Sunday, November 5, 11.59 PM

Recursion

The objective of this lab is to help you to continue to get familiar with recursion by writing small programs with recursive methods.

Marking Scheme
Each exercise carries 10 points.
Your final score will be scaled down to a value out of 10. For example, if there are three exercises and you score 9/10, 10/10 and 8/10 on the three exercises, your total is 27/30 or 9/10.
Error checking: Unless otherwise specified, you may assume that the user enters the correct data types and the correct number of input entries, that is, you need not check for errors on input

Submission Requirements:

- No submission other than a single ZIP file will be accepted.
- You MUST SUBMIT .java files that are readable by the TA. If you submit files that are unreadable such as .class file, the lab will be marked 0.
- Please additionally comment out package specifiers.

Late Submission Penalty: The lab is due on Sunday at 11.59 PM. Late submissions up to 5 hours (4.59 AM on Monday) will be accepted without penalty. After that, there will be a 10% late penalty per day on the mark obtained. For example, if you submit the lab on Monday at 12 noon and your score is 8/10, it will be reduced to 7.2/10. Submissions past two days (48 hours) after the grace submission time, that is, submission past 4.59 AM Wednesday will not be accepted.

Exercise 1 (Recursive Review)

This exercise presents a brief review of recursion. You will write a series of short methods to implement the recursive programs discussed in the lectures. Each program is presented in pseudo code. Implement all 3 methods as static methods and test them in the main method in a file called Exercise1.java.

a) Factorial of an integer n:
if n == 0, then factorial(n) = 1 //base case
if n > 0, then factorial(n) = n * factorial(n-1) //glue case

Write a recursive method that calculates the factorial of a non-negative integer *n*. Call this method in your main method, using a loop to print the factorials of the integers 1 through 10. Your method header should be:
public static int factorial(int n)

b) Fibonacci series:
if n==0, then fib(n) = 0 //base case
if n==1, then fib(n) = 1 //base case
if n>1, then fib(n) = fib(n-1) + fib(n-2) //glue case

Write a recursive method that finds the *n*th number in the Fibonacci series. Call this method in your main method, using a loop to print the first 20 numbers in the Fibonacci series. Your method header should be:
public static int fib(int n)

c) Exponentiation (*x* to the power *n*):
if n == 0, then power(x,n) = 1 //base case
if n>0, then power(x,n) = power(x,n-1)*x //glue case

Write a recursive method that calculates *x* to the power *n*, where *x* and *n* are both positive integers. Call this method in your main method, **prompting a user to enter *x* and *n***, and print *x* to the power of *n*. Your method header should be:
public static int power(int x, int n)

Test your program to ensure its proper operation. Save data from one test run in a text file.

A sample test run might look like this:
Factorial of a number
Enter a positive integer: 4
The factorial of 4 is 24

Fibonacci numbers
The first 20 numbers in the Fibonacci series are
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181

Power of a number
Enter a positive integer x: 2
Enter another positive integer: 3
2 to the power of 3 is 8

Exercise 2: Write a recursive method called *countDown* that takes a single positive integer *n* as parameter and prints the numbers *n* through 1 followed by 'BlastOff!'. Your method header should be:
public static void countDown(int n)

countDown(10) would print:
10 9 8 7 6 5 4 3 2 1 BlastOff!

Write a program called Exercise2.java. Your program should **accept input from a user** (specifying an integer *n*) and count down to 1 from *n* before printing 'BlastOff!'. Test your program with a variety of inputs to ensure its proper operation. Do not hard code inputs. Save data from one test run in a text file.

Exercise3 (Countdown Part 2)
Modify the *countDown* method you wrote in the previous exercise so that it prints only even numbers when *n* is even, and only odd numbers when *n* is odd. Your main method and the header for your *countDown* method can remain unchanged.

countDown(10) would print:
10 8 6 4 2 BlastOff!

countDown(9) would print:
9 7 5 3 1 BlastOff!

Write a program called Exercise3.java. Your program should **accept input from a user** (specifying an integer *n*) and count down from *n* before printing 'BlastOff!'. Test your program with a variety of inputs to ensure its proper operation. Do not hard code inputs. Save data from one test run in a text file.

Exercise4 (Multiples)
Write a recursive method called *multiples* to print the first *m* multiples of a positive integer *n*. Your method header should be:
public static void multiples(int n, int m)

multiples(2, 5) would print:
2, 4, 6, 8, 10

multiples(3, 6) would print:
3, 6, 9, 12, 15, 18

Write a program called Exercise4.java. Your program should **accept input from a user** (specifying integers *n* and *m*) and print multiples. Test your program with a variety of inputs to ensure its proper operation. Do not hard code inputs. Save data from one test run in a text file.
Note: You may print multiples in either ascending or descending order.

Exercise5 (Write Vertical)
Write a recursive method called *writeVertical* that takes that takes a single positive integer *n* as a parameter and prints the digits of that integer vertically, one per line. Your method header should be:
public static void writeVertical(int n)

writeVertical(1234) would print:
1
2
3
4

Write a program called Exercise5.java. Your program should **accept input from a user** (specifying an integer *n*) and print its digits vertically. Test your program with a variety of inputs to ensure its proper operation. Do not hard code inputs. Save data from one test run in a text file.

Exercise6 (Sum of Squares)

Write a recursive method called *squares* that takes that takes a single positive integer *n* as a parameter and calculates the sum of the squares of all digits 1 through *n*. Your method header should be:
public static int squares(int n)

squares(4) would return:
30

Write a program called Exercise6.java. Your program should **accept input from a user** (specifying an integer *n*) and return the sum of the squares of all digits 1 through *n*. Test your program with a variety of inputs to ensure its proper operation. Do not hard code inputs. Save data from one test run in a text file.

Exercise7 (Towers of Hanoi)

Rewrite the recursive solution to the Towers of Hanoi problem discussed in recent lectures, and modify it to return the number of moves required to solve the problem for a given number of discs. Your recursive method, called *solve,* should take a single positive integer *n* as a parameter and return the number of moves required for an optimal solution. Your method header will likely take two forms:
public static long solve(int n)
and
public static long solve(int n, int start, int end, int tmp)

Write a program called Exercise7.java. Your program should **accept input from a user** (specifying an integer *n*) and return the number of moves required to optimally solve the Towers of Hanoi problem with *n* discs. Determine experimentally how the execution time for solving the Towers of Hanoi problem grows as the value of *n* increases. You can use the following code to capture the execution time:

```
long startTime, endTime, executionTime;
startTime = System.currentTimeMillis();

//code segment

endTime = System.currentTimeMillis();
executionTime = endTime – startTime;
```

Test your program with inputs of 8, 12, 16, 20, 24, 28, and 32. Save data from your test runs in a text file showing *n*, the number of moves, and execution time in three columns.
Note: Calculating the number of moves required to solve the Towers of Hanoi problem for a given number of discs is straightforward. The formula is (2^n)-1. Do not use this formula in your solution. Your *solve* method must return this value upon exit.

What to submit:
One zip file containing the following:
1. All .java files
2. One text/word/pdf document containing sample inputs/outputs. Please ensure your name and Banner ID is on this document.
You MUST SUBMIT .java files that are executable by your TAs. TAs will run your program on the RandomNames.txt file and check your solution. If you submit files that are unreadable such as .class, you will lose points. Please additionally comment out package specifiers.