Justin Pytel-Meyer

Aaron Tran-Huyhn

**CS 116 Fall 2022 Lab #10**

Due: **Tuesday, November 29, 2022 11:00 PM CST**

Points: **20**

**Instructions:**

1. Use this document template to report your answers and create separate java files for your classes. Enter all lab partner names at the top of first page.
2. You don’t need to finish your lab work during the corresponding lab session.
3. ZIP your Java files and lab report into a single file. Name the file as follows:

LastName\_FirstName\_CS116\_Lab10\_Report.zip

1. Submit the final document to Blackboard Assignments section before the due date. No late submissions will be accepted.
2. ALL lab partners need to submit a report, even if it is the same document.

**Objectives:**

1. (6 points) Demonstrate your understanding of recursion,
2. (14 points) Design and implement recursive method.

**Problem 1 [6 points]:**

**1.** **[3 pts]** Consider the following method.

// precondition: x >= 0

public void mystery(int x) {

System.out.print(x % 10);

if ((x / 10) != 0) mystery(x / 10);

System.out.print(x % 10);

}

Which of the following is printed as a result of the call mystery(1234) ?

1. 1441
2. 3443
3. 12344321
4. 43211234
5. Many digits are printed due to infinite recursion.

|  |
| --- |
| Your answer: |
| D |

**2.** **[3 pts]** Consider the following recursive method

public static int mystery(int n) {

if (n == 0) return 1;

else return 3 \* mystery(n - 1);

}

What value is returned as a result of the call mystery(5)?

1. 0
2. 3
3. 81
4. 243
5. 6561

|  |
| --- |
| Your answer: |
| D |

**Problem 2 [14 points]:**

**1. [7 pts]** You are given the following problem:

Give a string of length n, check if it is a palindrome.

1. **[2 out of 7 pts]** Define the GENERAL (the subproblem to solve) and BASE recursive case:

|  |
| --- |
| **GENERAL recursive case:** |
| If it is true that the first and last two characters are the same then call the method again with those characters removed until the string is smaller the two characters. |
| **BASE recursive case:** |
| If the string is of length 1 or less then it is either not a string or is already a palindrome. |

1. **[3 out of 7 pts]** **Write a Java implementation** (an application class with recursive method and main method that is testing it),

|  |
| --- |
| **Your code:** |
| **public** **static** **void** main(String[] args) {  String testStr = "not a palindrome";  String testStr2 = "tacocat";  System.***out***.println(*palindrome*(testStr));  System.***out***.println(*palindrome*(testStr2));  }  **public** **static** **boolean** palindrome(String inputStr) {  **if** (inputStr.length() < 2) {  **return** **true**;  }  **return** inputStr.substring(0, 1).equals(inputStr.substring(inputStr.length() - 1))  & *palindrome*(inputStr.substring(1, inputStr.length() - 1));  } |

1. **[2 out of 7 pts] Draw a diagram of all recursive calls** (along with method arguments and return values) that illustrates the method behavior for a small input problem. You can use lecture slides to get an idea how this diagram should look.

|  |
| --- |
| **Your diagram:** |
|  |

**2. [7 pts]** You are given the following problem:

Given two inputs:

* a (which is of type double), and
* t (which is of type double).

approximate the value of the square root of the number a to within the degree of accuracy (+/- t). specified by tolerance t. A recurrence relation for the square root of number a is:

X(n+1) = (1/2)\*(X(n) + a/X(n)) (1)

and:

X(0) = a/2 (2)

That is, the next approximation X(n+1) is computed using the previous approximation X(n) according to the formula (1). Starting with the first approximation of a/2, apply the formula until the square of the new approximation is within the specified tolerance of a.

|  |
| --- |
| HINT: This is more “growing” than “reducing” situation.  Keep increasing the value of n. |

1. **[2 out of 7 pts]** Define the GENERAL (the subproblem to solve) and BASE recursive case:

|  |
| --- |
| **GENERAL recursive case:** |
| If the current approximation value squared is smaller than the target value and in range of the specified, return that all the way up the stack  Else return the recurrence relation ship up the stack to calculate another approximation. |
| **BASE recursive case:** |
| If N is zero return a/2 |

1. **[3 out of 7 pts]** **Write Java implementation** (an application class with recursive method and main method that is testing it),

|  |
| --- |
| **Your code:** |
| **public** **static** **void** main(String[] args) {    **double** a = 200.0;  **double** t = 100.0;    System.***out***.print(*squareRoot*(a,a,t));        }    **public** **static** **double** squareRoot(**double** currIteration, **double** targetNum, **double** approxRange) {      **if**(currIteration == 0.0) {  **return** targetNum / 2;  }    **double** currentApprox = *squareRoot*(currIteration - 1,targetNum,approxRange);    **if**(Math.*pow*(currentApprox, 2.0) < targetNum && Math.*pow*(currentApprox, 2.0) > targetNum - approxRange ) {  System.***out***.println(currentApprox);  **return** currentApprox;  }    **return** 0.5 \* (currentApprox + (targetNum / currentApprox));  } |

1. **[2 out of 7 pts] Draw a diagram of all recursive calls** (along with method arguments and return values) that illustrates the method behavior for a small input problem. You can use lecture slides to get an idea how this diagram should look.

|  |
| --- |
| **Your diagram:** |
|  |