# CSC 205 Lab 9 : Recursion

## Goals

After completing this lab, you should be able to:

* Identify recursive methods, and understand the role and power of recursion
* Trace through recursive methods using tree diagrams that display all method calls and the values of the current parameters at each call
* Write your own recursive method in place of an iterative method

## Lab Startup

Change into your Labs directory, and let's create and change into a Lab9 directory.

Now, let's copy over some files by typing : cp /pub/digh/CSC205/Lab9/\* .

**Identifying Recursion**

Label each of the following methods as recursive or iterative.

**1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

private static int fib (int n) {

if ( (n == 0) || (n == 1) )

return 1;

else

return ( fib(n-1) + fib(n-2) );

}

**2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

private static int product (int n) {

if (n == 0)

return 1;

else

{

int x = 1;

for (int i = 1; i <= n; i++)

x++;

return x;

}

}

**3. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

private static int count (int[] a, int key) {

int num = 0;

for (int i = 1; i <= 20; i++)

if (a[i] == key)

num++;

return num;

}

**4. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

private static boolean member (int n, int start, int finish, int[] a) {

if (start > finish)

return false;

else if (a[start] == n)

return true;

else

return (member(n, start+1, finish, a));

}

## Tracing through Recursive Methods and Producing Tree Diagrams

Trace through the recursive method below after the call decToBinary(25), and determine the output. On a separate sheet of paper, show a recursive tree diagram like we did in class.

private static void decToBinary (int num)

{

if (num > 0)

{

decToBinary (num / 2);

System.out.print(num % 2);

}

}

After tracing through this method, compile and test the file DecToBinary.java to check your answer found in your trace.

* What is the base case of this method? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* What exactly does this method do? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* What would be the output of the method above if the two lines of code within the if statement were reversed? That is, if the print statement came first. Try it if you are unsure. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Now, let’s trace through the recursive method below after the call peeps(2) and determine the output. On a separate sheet of paper, show a recursive tree diagram.

private static void peeps (int n)

{

System.out.println(n + " Peeps");

if (n == 5)

return;

else

peeps ( n + 1 );

System.out.println(n + " Peeps");

}

Compile and test the file Peeps.java to check your answer found in your trace.

Now, what value would be returned by the recursive method below if it is called as follows from the main method:

System.out.println(puzzle(9));

Show all calls of your recursive trace and what each is equal to.

private static int puzzle (int n)

{

if ( (n % 3) == 2 )

return 1;

else if ( (n % 3) == 1 )

return ( puzzle (n + 1) + 2 );

else

return ( puzzle (n / 3) + 1 );

}

You can check your answer by compiling and test the program Puzzle.java.

## Writing a Program with Infinite Recursion

Create a program named blowUp that calls the main method infinitely. Your call will have to include args as a parameter. Include a counter in your program which you print out before each call to the main method. This counter will need to be declared as global, public, and static prior to the main method. Send your output to an external file using file re-direction.

(e.g., java blowUp > myOutput).

Check your output file for the last number recorded. (Go into vim and then hit a capital G to go to the end of the file.) How many times can one call main infinitely on cobra before it crashes? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Writing a Program with an Iterative and Recursive Solution

Write a value-returning iterative method named iterativeSum that takes

an integer *n* as a parameter and finds the sum of the expression  as *k* ranges from 1 through *n*. That is,



Name your program file mySums and include a call to your method from main to test it.

You will need to use the pow method from the Math class. Since the pow method returns a double, let's have our method return a double. One sample test case you can use is an *n* of 4. This should return a 34.

Add another method to your program named recurSum which is a recursive version of your iterative method. Include a call to it from main. You should of course get the same answer for this method. Use the steps we learned in class for writing a recursive method.

* What is the base case of this method? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

##### What is the recursive case of this method? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

##### 

On your separate sheet of paper, show a trace of your recursive method when *n = 3* using a recursive tree diagram.

**The PrintMe method**

Run the program PrintCall in your directory. You should get a printout like the following.

This was written by call number 2.

This was written by call number 3.

This was written by call number 4.

This ALSO was written by call number 4.

This ALSO was written by call number 3.

This ALSO was written by call number 2.

This ALSO was written by call number 1.

Complete the body of the printMe method in the PrintCall program in your directory so that it will print the same thing. You need to add an if statement with a recursive call, and an additional if statement for the printing of the ALSO statements with the proper indentation.

**Extra Credit**

*The following section is optional. You will earn extra credit by completing it.*

**Rewriting Your Recursive Method So That It Prints an Indented**

**Tree Diagram**

Modify your Puzzle program so that the recursive method includes output statements to trace its action as it executes. For example, the tree-like trace displayed for puzzle(9)

should have a form exactly like the following.

puzzle(9) = puzzle(3) + 1

puzzle(3) = puzzle(1) + 1

puzzle(1) = puzzle(2) + 2

puzzle(2) = returns 1

puzzle(1) = returns 3

puzzle(3) = returns 4

puzzle(9) = returns 5

You will need to change your call in main to just puzzle(9); and then add appropriate output statements to your recursive method. You will have output statements prior to recursive calls, and then output statements once the base case has been reached, and the answers are being sent back to the prior calls.

You will need to mix in a loop to handle the tabbing over. Place the loop below

both before and after your conditional statement.

for (int i=1; i <= 4-n; i++)

System.out.print("\t");

You’re also going to need to set up a local variable that your answers can be assigned to. Your method will still return an int.