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. Recursive

```
private static int fib (int n) {
    if ( (n == 0) || (n == 1) )
          return 1;
     else
          return ( fib(n-1) + fib(n-2) );
}
```

2. Iteration

```
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. Iteration

```
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. Recursion — Linear search

```
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.

(dB)

```
private static void decToBinary (int num)
{
        if (num > 0)
        {
                decToBinary (num / 2);
                System.out.print(num % 2);
        }
}
```

dB(25) = (25/2 = 12, remainder 1) = 1
↳ dB(12) = (12/2 = 6, remainder 0) = 0
↳ dB(6) = (6/2 = 3, remainder 0) = 0
↳ dB(3) = (3/2 = 1, remainder 1) = 1
↳ dB(1) = (1/2 = 0, remainder 1) = 1
↳ Base case reached. Rewind
Sysout = 11001 →

% remainder operator (remainder after the division)

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? if(num ≤ 0)
- What exactly does this method do? Prints binary of number
- 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.

Print out the binary without the rewind. Sysout: 10011

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.

peeps (2)

```
private static void peeps (int n)
{
        System.out.println(n + " Peeps");

        if (n == 5)
                return;
        else
                peeps ( n + 1 );

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

(2+1=3) (3+1=4) (4+1=5)

2 peeps
3 peeps
4 peeps
5 peeps (Rewind)
4 peeps
3 peeps
2 peeps

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

Puzzle.java

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 );
}
```

puzzle(9) = puzzle(9/3)+1
= puzzle(3)+1

puzzle(3) = puzzle(3/3)+1
= puzzle(1)+1

puzzle(1) = puzzle(1+1)+2
= puzzle(2)+2

puzzle(2) = 1

puzzle(9) = puzzle(3)+1
puzzle(3) = puzzle(1)+1
puzzle(1) = puzzle(2)+2
puzzle(2) = 1
Base case reached.
Rewind.
puzzle(2) = 1
puzzle(1) = 1+2=3
puzzle(3) = 3+1=4
puzzle(9) = 4+1=5

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? 10819

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 $2^k + 1$ as k ranges from 1 through n. That is,

sigma (sum of)

$$\sum_{k=1}^{n} 2^k + 1$$

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? `if (n < 1) { return 0; }`
- What is the recursive case of this method? `return Math.pow(2,n) + 1 + recurSum(n-1);`

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

return Math.pow(2,3)+1+recurSum(2); 8+1+8=17
" " " (2,2)+1+recurSum(1); 4+1+3=8
" " " (2,1)" " " (0); 2+1+0=3
return (0);

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.