Advanced Object Oriented Programming

EECS 2030

Fall 2020 :: **Section B**

Matthew Kyan

Advanced Object Oriented Programming

EECS 2030

Lecture 5 :: **Recursion II**

Matthew Kyan

- Tracing a recursive method (video)
  - in Eclipse (video)
  - by hand (representations for worksheet)
    - program stack (compact version)
    - recursive tree
- More examples
  - palindrome

Tracing a method in Eclipse

- Using debug mode
- See video on echo360

- Describes:
  - Setting breakpoints
  - Running in debug mode
  - The debug perspective
  - Tracking the stack & variables in stack memory

EECS2030B - EECS2030_F2020_B/src/week02/recursion1/RecursiveMethods1.java - Eclipse IDE

Debug | Project Explorer | Servers

- RecursiveMethods1 [Java Application]
  - week02.recursion1.RecursiveMethods1 at localhost:519
    - Thread [main] (Suspended)
      - RecursiveMethods1.countZeros(long) line: 99
      - RecursiveMethods1.countZeros(long) line: 96
      - RecursiveMethods1.main(String[]) line: 153
    - /Library/Java/JavaVirtualMachines/jdk1.8.0_261.jdk/Cont

RecursiveMethods1.java

```
76      /**
77       * A recursive method to count the number of digits
78       *
79       * @param n a long value
80       * @return the number of zero digits found in n
81       *
82       */
83      public static int countZeros(long n) {
84
85          if (n == 0L) {          // base case 1
86              return 1;
87          }
88          else if (n < 10L) {     // base case 2
89              return 0;
90          }
91
92          boolean lastDigitIsZero= (n % 10L == 0);
93          final long m = n / 10L;
94
95          if(lastDigitIsZero) {
96              return 1 + countZeros(m);
97          }
98          else {
99              return countZeros(m);
100         }
101
102     }
103
```

Variables | Breakpoints | Expressio

Name	Value
no method return value	
n	80041
lastDigitIsZero	false
m	8004

Console | Problems | Debug Shell

RecursiveMethods1 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_261.jdk/Contents/Home/bin/java (Sep. 22, 2020, 9:17:50 p.m.)

countZeros(800410L):

When a method is invoked, the line number that the current method is up to is stored on the stack (so the method knows where to keep executing from after it returns)

YORK UNIVERSITÉ UNIVERSITY

line numbers

line number at which current countZeros(..) is suspended while a new countZeros(..) is invoked and pushed to the stack

```
1   public static int countZeros(long n) {
2
3       if(n == 0L) {           // base case 1
4               return 1;
5       }
6       else if(n < 10L) {      // base case 2
7               return 0;
8       }
9
10      boolean lastDigitIsZero = (n % 10L == 0);
11      final long m = n / 10L;
12
13      if(lastDigitIsZero) {
14              return 1 + countZeros(m);
15      }
16      else {
17              return countZeros(m);
18      }
19  }
```

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L );
```
// called from main, line 5

program stack:

numzeros=?, line=5

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L );
```

```
countZeros( 800410L )
```

program stack:

numzeros=?, line=5
n=800410L, return=

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L );
```
// called from main, line 5

```
countZeros( 800410L )
```

```
countZeros( 80041L )
```

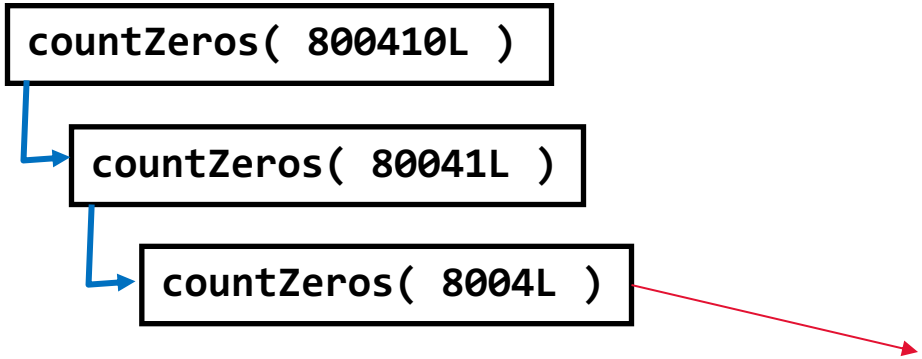
program stack:

numzeros=?, line=5
n=800410L, return=1+?, line=14
n=80041L, return=

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L ); // called from main
```



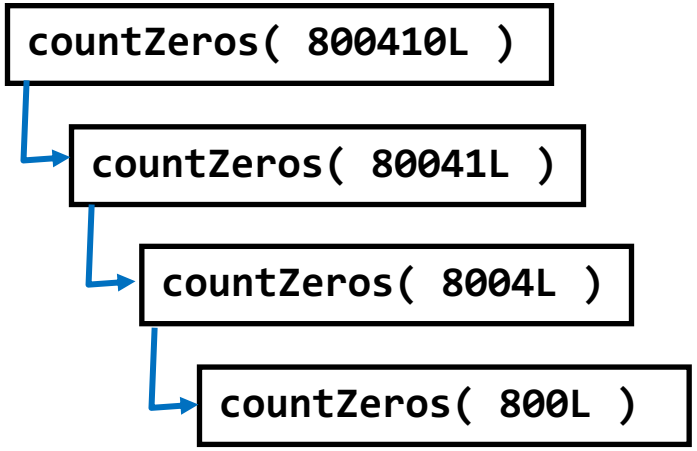
program stack:

numzeros=?, line=5
n=800410L, return=1+?, line=14
n=80041L, return=?, line=17
n=8004L, return=

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L ); // called from main
```



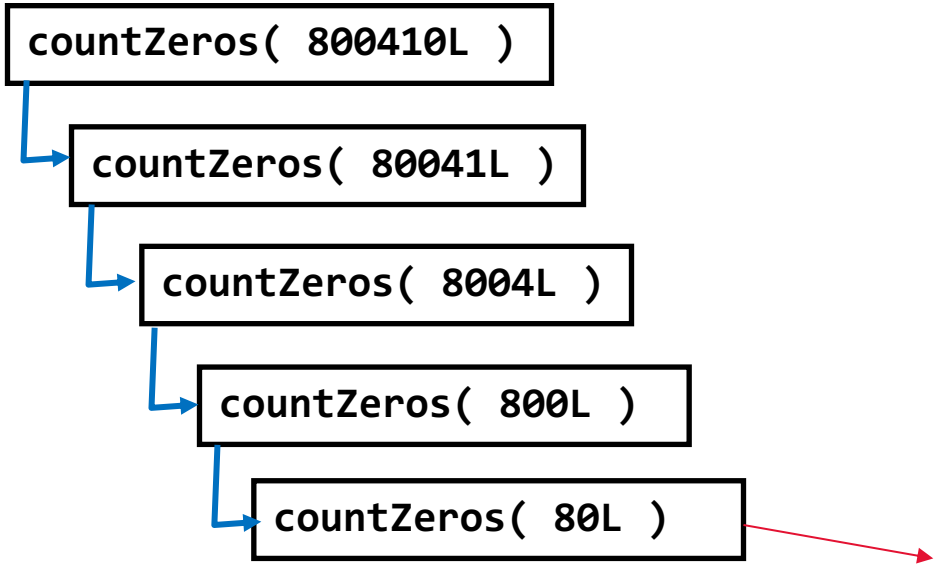
program stack:

numzeros=?, line=5
n=800410L, return=1+?, line=14
n=80041L, return=?, line=17
n=8004L, return=?, line=17
n=800L, return=

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L ); // called from main
```



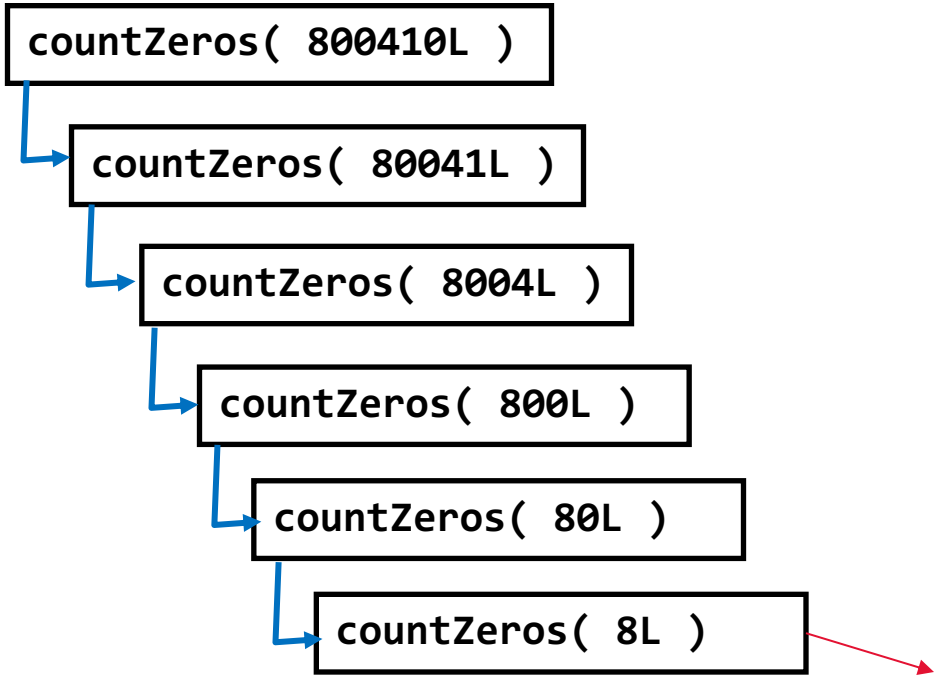
program stack:

numzeros=?, line=5
n=800410L, return=1+?, line=14
n=80041L, return=?, line=17
n=8004L, return=?, line=17
n=800L, return=1+?, line=14
n=80L, return=

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L );  // called from main, line 5
```



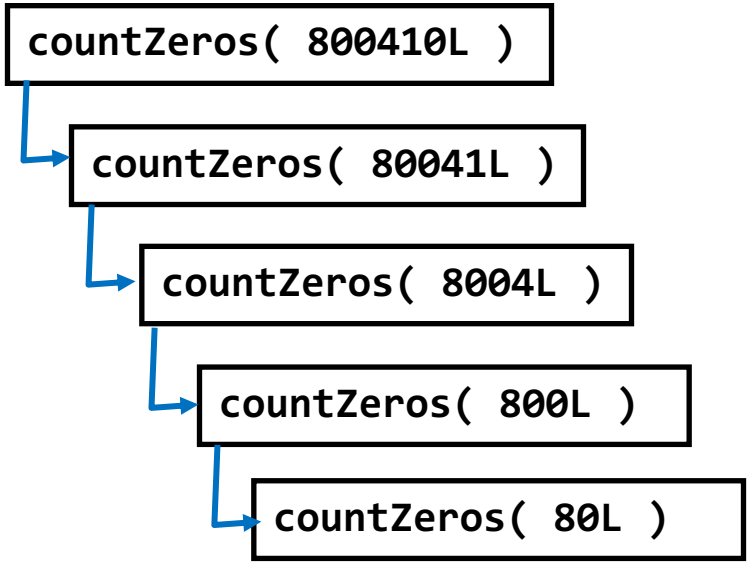
program stack:

numzeros=?, line=5
n=800410L, return=1+?, line=14
n=80041L, return=?, line=17
n=8004L, return=?, line=17
n=800L, return=1+?, line=14
n=80L, return=1+?, line=14
n=8L, return=0

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L ); // called from main
```



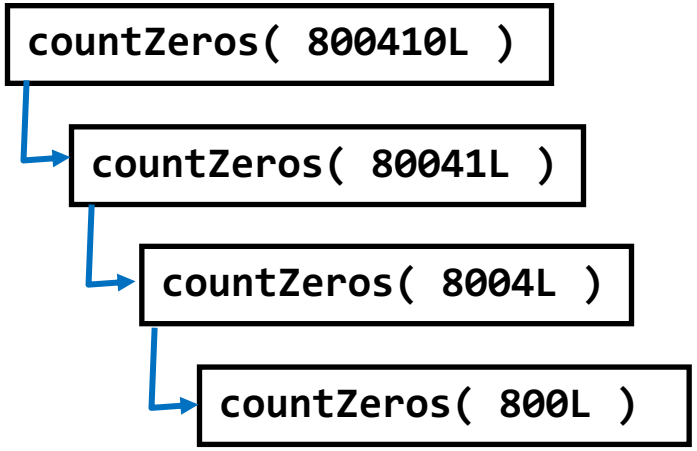
program stack:

numzeros=?, line=5
n=800410L, return=1+?, line=14
n=80041L, return=?, line=17
n=8004L, return=?, line=17
n=800L, return=1+?, line=14
n=80L, return=1+0, line=14

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L ); // called from main
```



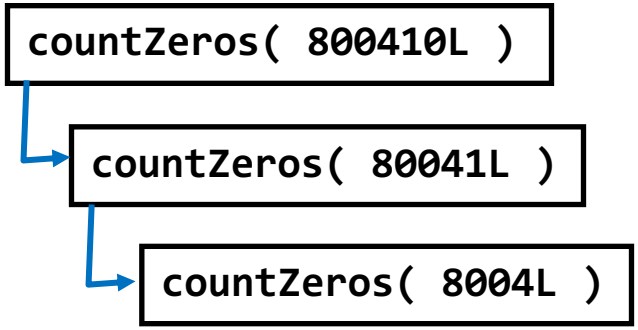
program stack:

numzeros=?, line=5
n=800410L, return=1+?, line=14
n=80041L, return=?, line=17
n=8004L, return=?, line=17
n=800L, return=1+1, line=14

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L ); // called from main
```



program stack:

numzeros=?, line=5
n=800410L, return=1+?, line=14
n=80041L, return=?, line=17
n=8004L, return=2, line=17

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L ); // called from main
```

```
countZeros( 800410L )
    countZeros( 80041L )
```

program stack:

numzeros=?, line=5
n=800410L, return=1+?, line=14
n=80041L, return=2, line=17

program stack (compact representation)

:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L ); // called from main
```

```
countZeros( 800410L )
```

program stack:

numzeros=?, line=5
n=800410L, return=1+2, line=14

program stack (compact representation)

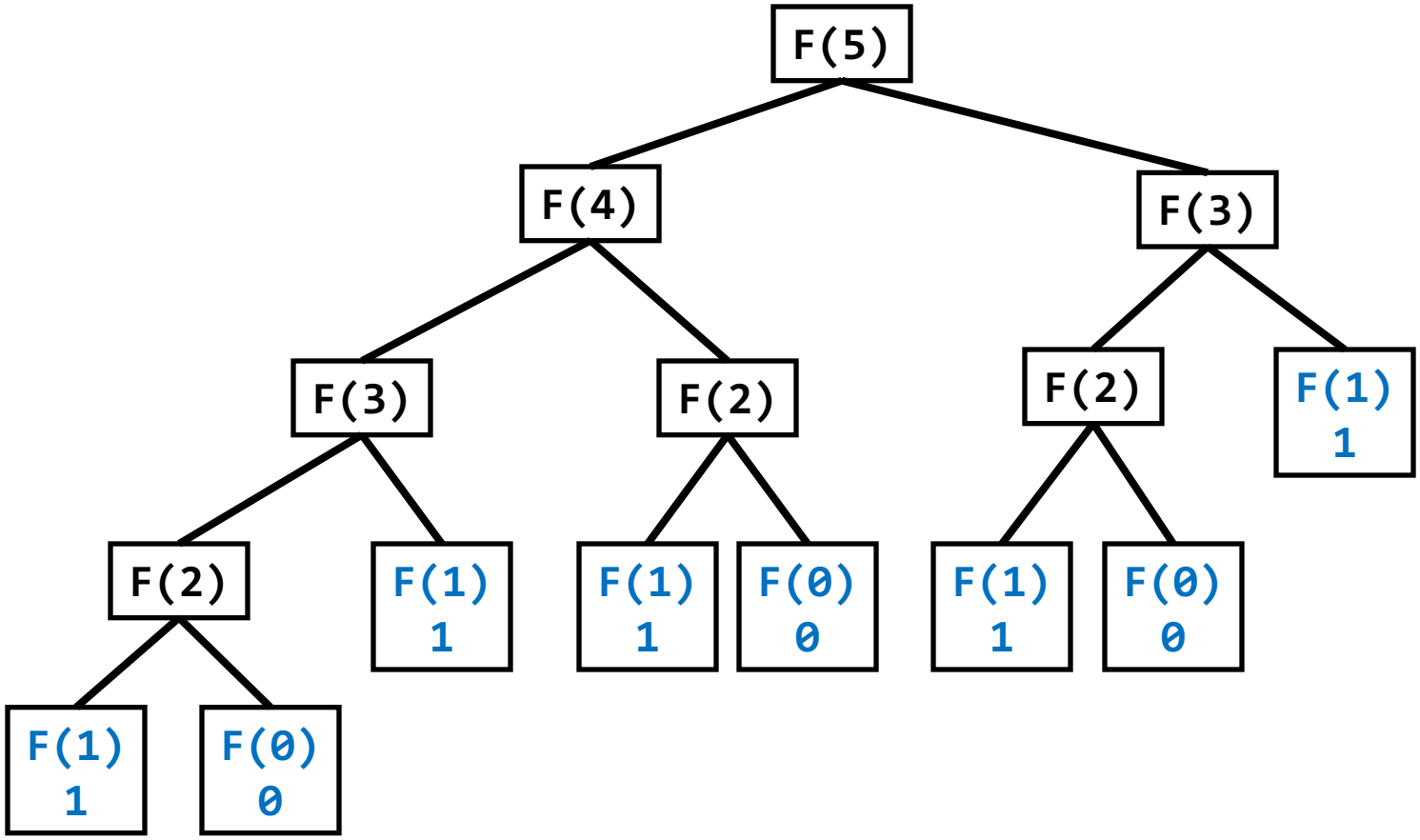
:: show args, return value, and line number (if suspended)

```
int numzeros = countZeros( 800410L ); // called from main
```

program stack:

numzeros=3

recursive tree (e.g. Fibonacci)



recursive tree (alternative manual trace)

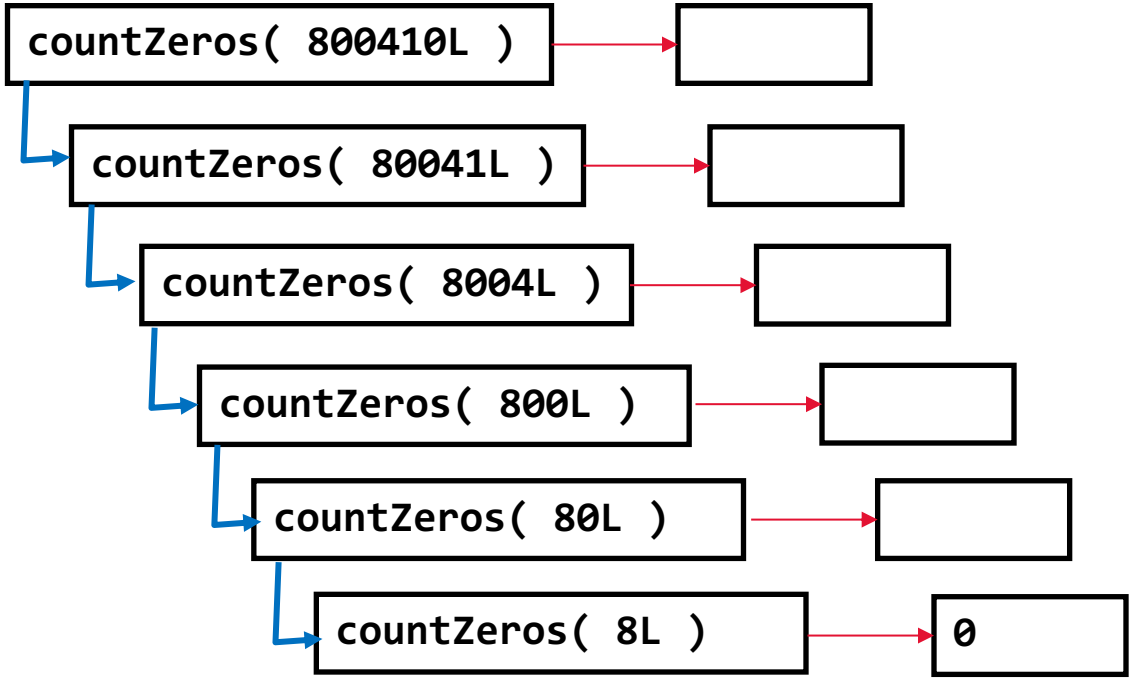
:: show method calls (indent to calling method), show return value (if appropriate)

```
int numzeros = countZeros( 800410L );
```

recursive tree (alternative manual trace)

:: show method calls (indent to calling method), show return value (if appropriate)

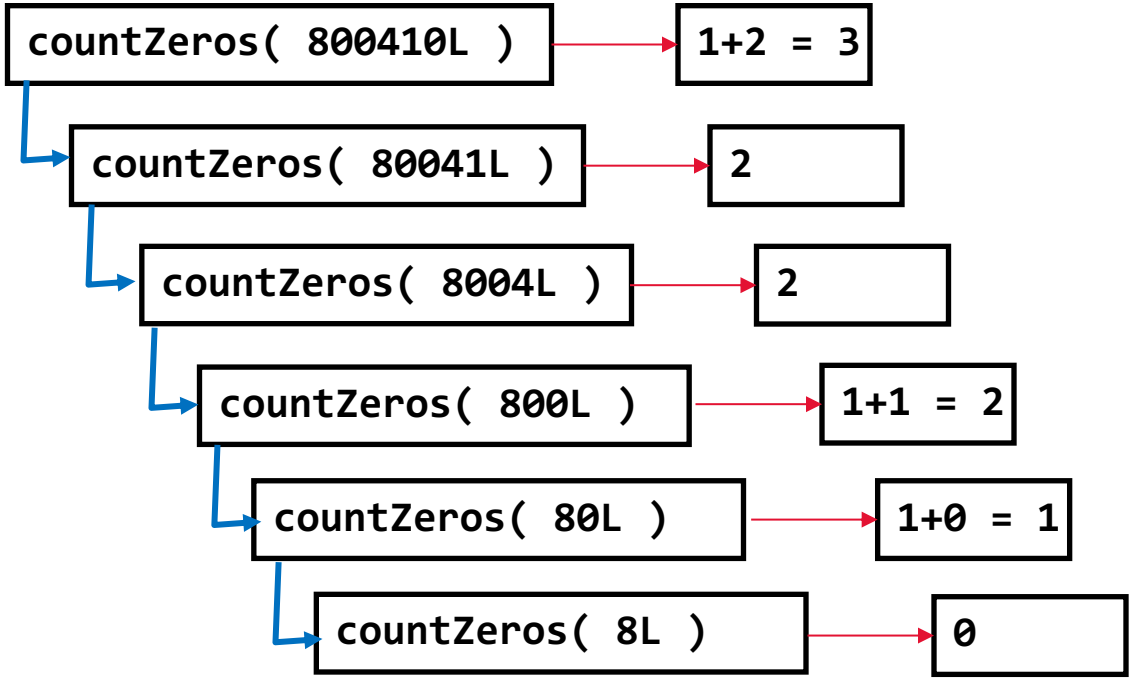
```
int numzeros = countZeros( 800410L );
```



recursive tree (alternative manual trace)

:: show method calls (indent to calling method), show return value (if appropriate)

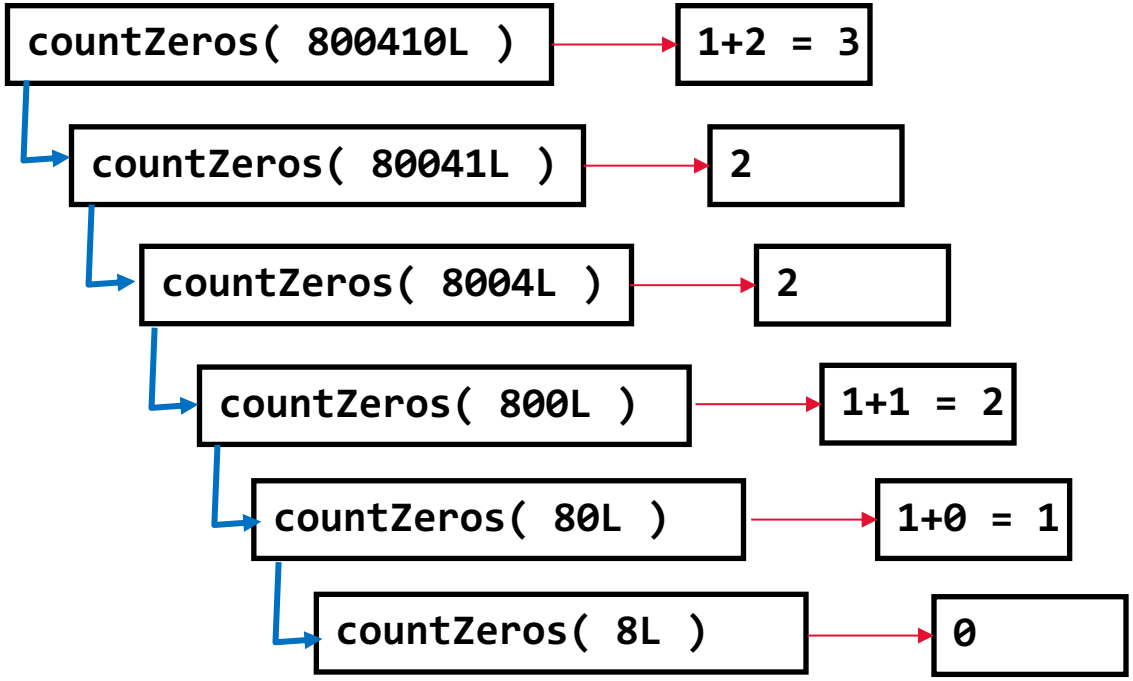
```
int numzeros = countZeros( 800410L );
```



recursive tree (alternative manual trace)

:: show method calls (indent to calling method), show return value (if appropriate)

```
int numzeros = countZeros( 800410L );
```



Other examples

- Palindrome
- Find maximum (array)
- Reverse a list

Palindrome

- A palindrome = a string that if reversed, is equal to itself
- E.g.
  - “radar” → when reversed = “radar” → is a palindrome
  - “racecar” → is a palindrome

  - “modem” ← not a palindrome

isPalindrome(String str)

```
public static boolean isPalindrome(String input) {

    boolean palindrome = false;

    if (input.length() <= 1) {
        palindrome = true;
    }
    else {
        // strategy:
        //      string is palindrome first and last char's are same, and
        //      middle part of string is also a palindrome

        char first = input.charAt(0);
        char last = input.charAt(input.length()-1);
        String middle = input.substring(1, input.length()-1);

        palindrome = ( (first==last) && isPalindrome(middle) );
    }

    return palindrome;
}
```

findMax(int[] input)

// typical iterative solution

```
public static int findMax(int[] input) {

    int max = input[0];

    for (int index=1; index<input.length; index++) {
        if (input[index]>max)
            max = input[index];
    }

    return max;
}
```

findMax(int[] input)

// lets assume you are asked to implement the following
// method to do the same thing, using recursion…

```
public static int findMax(int[] input) {


    // recursive implementation?


}
```

How to traverse array recursively?

- Method does not have arguments we need?
- No problem, create a helper method …

```
public static int findMaxHelper(int[] input, int index, int max);
```

findMax(int[] input)

// lets assume you are asked to implement the following
// method to do the same thing, using recursion…

```
public static int findMax(int[] input) {

    return findMaxHelper(input, 0, input[0]);

}

public static int findMaxHelper(int[] input,
                                int index, int max) {

    // implementation not shown
}
```

index → used to traverse array
max → used to track max so far

```
public static int findMaxHelper(int[] input,
                                int index, int max) {
    if (index<input.length) {
    // not at the end of the array yet

        if (input[index]>max)
          max = findMaxHelper(input, index+1, input[index]);
        else
          max = findMaxHelper(input, index+1, max);
    }

    return max;

}
```

index → used to traverse array
max → used to track max so far

```
public static int findMaxHelper(int[] input,
                                int index, int max) {
    if (index<input.length) {
    // not at the end of the array yet

        if (input[index]>max)
          max = findMaxHelper(input, index+1, input[index]);
        else
          max = findMaxHelper(input, index+1, max);
    }

    return max;

}
```

findMax using next index;
update max

index → used to traverse array
max → used to track max so far

```
public static int findMaxHelper(int[] input,
                                int index, int max) {
    if (index<input.length) {
    // not at the end of the array yet

        if (input[index]>max)
          max = findMaxHelper(input, index+1, input[index]);
        else
          max = findMaxHelper(input, index+1, max);
    }

    return max;

}
```

findMax using next index;
keep current max

Traversing (recursively)

- Strings: use substring() method
- Arrays: use index variable

- Lists (e.g. Arraylist)
  - we haven't considered collections yet so if not familiar don't worry
    - Use subList() method
    - Or .. can remove items from a list as they are processed

Approach to recursive design?

- What are the base cases? (smallest solutions we know)
  - Think about smallest possible input, and its solution
- What is the solution for an n-sized problem, in terms of a reduced version of the problem (e.g. n-1, n-2, etc.)?
- Combine both of the above into your solution
  - ensure base cases come first!

Homework questions (to think about)

```
/** write method to recursively sum elements in array*/
public static int arraySum(int[] input, int index) {
        // to implement
}


/** find number of elements in an array that are greater
        than a given number */
public static int countGreater(int[] input, int number,
                                                int index) {
        // to implement
}
```