Recursion

EMBRY-RIDDLE Aeronautical University

Like Algorithms, recursion starts in mathematics. You may recall recursive mathematical definitions in many of your past courses including CS 222

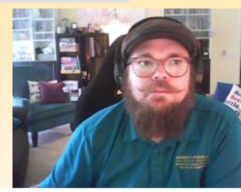
Recursive Definition - Webster-Merriam

"The determination of a succession of elements (as numbers or functions) by operation on one or more preceding elements according to a rule or formula involving a finite number of steps."

- Two key elements:
  - Base case
  - Progress / set creation case
- Uses:
  - Defines a sequence for representing members of a set
  - A condition for determining if an element belongs to a set

$$n! = \begin{cases} 1 & if(n == 1) \\ n*(n-1)! & if\ (n > 0) \end{cases}$$

Example: 3! = 3*2! = 3*2*1! = 3*2*1*1 = 6



CS 315 College of Engineering, Daytona Beach, FL

1

Recursive Functions

EMBRY-RIDDLE Aeronautical University

Webster-Merriam

A function that "calls itself one or more times until a specified condition is met at which time the rest of each repetition is processed from the last one called to the first"

```
public int factorial(int n) {
         if (n==0) return 1;
         else  return n*factorial(n-1);
}
```

factorial(4)
4 * factorial(3)
3*factorial(2)
2*factorial(1)
1
2
6
24

- Each recursive call creates a new activation record on the program stack
- Once a recursive call returns,
  - its activation record is popped from the program stack,
  - return value passed to the previous activation record,
    - i.e. the method that called it
  - Restored activation record continues executing where it left off before the recursive call

CS 315 College of Engineering, Daytona Beach, FL 2

2

The Program Stack

EMBRY-RIDDLE Aeronautical University

A stack is a data structure that stores the most recent record at the top.

The Java program stack stores the activation records for each function call.

When it completes, it is popped from the stack, its results (if applicable) are returned to the previous activation record, which resumes where it left off prior to the recursive call.

Program starts	Main AR
main calls f1	F1 AR Main AR
f1 calls f2	F2 AR F1 AR Main AR
f2 calls f3	F3 AR F2 AR F1 AR Main AR

3

Example: Recursive Binary Search

EMBRY-RIDDLE Aeronautical University

- Previous video demonstrated binary search using a while loop
  - With each iteration, the middle index is compared and then lo and hi are adjusted to narrow the search on the left or right subarray
- How do we make this recursive?
  - Recursive method:
    - RecursiveBinarySearch(int [] a, int key, int lo, int hi)
  - Base Cases?
    - If hi > lo, we have reached the case where we have run out of things to search.
    - If a[mid] == key, we found the key in the array
  - Progress case
    - If key > a[mid], then search right hand side of array
    - If key < a[mid], then search left hand side of the array

4

Recursive Binary Search

EMBRY-RIDDLE Aeronautical University

```java
//Kickoff method for recursive algorithm
// - Implements the same API as BinarySearch
// - returns the results of a call to the recursive case with lo=0 and hi=length-1
public static int indexOf(int[] a, int key) {
    return indexOf(a, key, 0, (a.length-1));
}


//Recursive implementation of the binary search algorithm. Set to private since only accessible
// through the public indexOf method, which kicks off the search.
private static int indexOf(int[] a, int key, int lo, int hi) {
    if (lo > hi)
        return -1; //Base case - key not found
    else {
        int mid = lo + (hi - lo) / 2;
        if (key < a[mid]) {
            return indexOf(a, key, lo, (mid-1)); //Progress - Recurse Left
        }
        else if (key > a[mid]) {
            return indexOf(a, key, (mid+1), hi); //Progress - Recurse Right
        }
        else
            return mid; //Base Case - key == a[mid]
    }
}
```

```
D:\GitHub\cs315-alg-and-ds-java\cs315-supplement\src\m0-binarysearch>java -classpath build edu.princeton.cs.algs4.BinarySearch tinyW.txt < tinyT.txt
50
99
13
```


Example: Integer Print by Digit

EMBRY-RIDDLE Aeronautical University

Given:

```java
void print(int n)
{
        if (n <=0) return;
        else if (n < 10)
                System.out.println(n);
        else {
                print((int) n/10);
                System.out.println(n%10);
        }
}
```

Grab a piece of paper and trace out the execution.

Trace through the execution with the following inputs:

1. 2
2. 12
3. 352
4. 1441

Side Note: "(int) n/10" divides the number by 10 and casts the result into an integer. This is not necessary in more modern versions of Java, but included to simply highlight an awareness of operator vs. data type.

Example: Integer Print by Digit

EMBRY-RIDDLE Aeronautical University

Given:
```
void print(int n)
{
        if (n <=0) return;
        else if (n < 10)
                System.out.println(n);
        else {
                print((int) n/10);
                System.out.println(n%10);
        }
}
```

Trace through the execution with the following inputs:

1. 2
2. 12
3. 352
4. 1441

Trace:

1.print(2) -> "2"

2. print(12) -> print(1)
-> "1"
-> "2"

3. print(352) -> print(35)
-> print(3)
-> "3"
-> "5"
-> "2"

4. print(1441) -> print(144)
-> print(14)
-> print(1)
-> "1"
-> "4"
-> "4"
-> "1"

Types of Recursion

EMBRY-RIDDLE Aeronautical University

- Tail Recursion – Recursive function call is the last executed operation of the function call
  - Recursion could be replaced by a loop
  - For example, our RecursiveBinarySearch method replaced a while loop
- Non-Tail Recursion
  - Cannot be replaced by a simple loop
- Indirect Recursion

Tail recursion may be used for implementing recursive methods in code because it explicitly shows the original definition

```
public void fact(int n) {
        if (n <= 1) return 1;
        else return n*fact(n-1);
}
```

Alternatively, with a loop it is less concise:
```
public void fact(int n) {
        int factorial = 1;
        for (int i=1, i <= n; i++) {
                factorial = factorial*i;
        }
        return factorial;
}
```

Example: Indirect Recursion

EMBRY-RIDDLE Aeronautical University

- A method or definition that calls itself indirectly
  - Chain of method calls
  - Multiple chains may exist
- Examples:
  - f()→g() → h() → f()
  - Message Decoding:
    - receive → decode → store → receive
- Some base case must exist that terminates the chain

Example:

- Progress cases:
  - sin(x) = sin(x/3)*(3-tan^2(x/3))/(1+tan^2(x/3))
  - cos(x) = 1 – sin(x/2)
  - tan(x) = sin(x)/cos(x)
- Base case:
  - If x is sufficiently small, sin(x) = x – (x^3/6)

Non-Tail Example

EMBRY-RIDDLE Aeronautical University

```
public void Mystery(int Num)
{
   if (Num < 0)
   {
      Mystery(Num * -1);
      System.out.print("+");
   }
   else if (Num < 10)
      System.out.print(Num);
   else
   {
      System.out.print(Num % 10);   // Num modulo
10
      Mystery(Num/10);
   }
}

public static void main(String []args)
{
   Mystery(-734);
}
```

Trace the execution of the program starting at main on a separate sheet. Use your results to answer the following questions.

a. How many calls to function Mystery will occur when this program is executed?

a) 734
b) 4
c) 3
d) 5
e) none of the above

b. What will this program output when is executed?

a) 734+
b) +437
c) 437+
d) an infinite number of '+' characters
e) none of the above

Non-Tail Example

```
public void Mystery(int Num)
{
   if (Num < 0)
   {
      Mystery(Num * -1);
      System.out.print("+");
   }
   else if (Num < 10)
      System.out.print(Num);
   else
   {
      System.out.print(Num % 10);   // Num modulo
10
      Mystery(Num/10);
   }
}

public static void main(String []args)
{
   Mystery(-734);
}
```

Trace the execution of the program starting at main on a separate sheet. Use your results to answer the following questions.

a. How many calls to function Mystery will occur when this program is executed?

a) 734
b) 4
c) 3
d) 5
e) none of the above

b. What will this program output when is executed?

a) 734+
b) +437
c) 437+
d) an infinite number of '+' characters
e) none of the above

11

Non-Tail Example

```
public void Mystery(int Num)
{
   if (Num < 0)
   {
      Mystery(Num * -1);
      System.out.print("+");
   }
   else if (Num < 10)
      System.out.print(Num);
   else
   {
      System.out.print(Num % 10);   // Num modulo
10
      Mystery(Num/10);
   }
}

public static void main(String []args)
{
   Mystery(-734);
}
```

a. How many calls to function Mystery will occur when this program is executed?

a) 734
b) <u>4</u>
c) 3
d) 5
e) none of the above

b. What will this program output when is executed?

a) 734+
b) +437
c) <u>437+</u>
d) an infinite number of '+' characters
e) none of the above

12

Warning – Excess Recursion

EMBRY-RIDDLE Aeronautical University

```
int Fib(int n) {
        if (n < 2) return n;
        else return (Fib(n-2) + Fib(n-1));
    }
```

Trace:

Fib(6) = Fib(4) + Fib(5)
= Fib(2) + Fib(3) + Fib(5)
= Fib(0) + Fib(1) + Fib(3) + Fib(5)
= 0 + 1 + Fib(3) + Fib(5)
= 0 + 1 + Fib(1) + Fib(2) + Fib(5)
= …

25 calls for 7th number
Near ¼ million calls for 26th
Near 3 million calls for 31st
Lots of repeated function calls

Avoid Recursive calls where you are repeating the same recursive call multiple times. Fibonacci is a classic example where novices create code that excessively slow!