Recursion

Intro

- Objects must "exist" before being used in other objects.
- Infinite sets.
- Two parts:
  - base/anchor/ground case.
  - recursive/inductive step.
- In sets: Generate and Test.
- Factorial.
- Functions.
- Trees, Grammar, BNF, etc.

Recursion and method calls

- Remember where it was called (so we can return there).
- Also local variables, values for parameters, return address, link to caller's activation record and return values.
- Runtime stack...
- Activation records or stack frames.

Recursive call anatomy

- Power function.

```
1 double power(double x, int n)
2 {
3         if(n == 0)
4                 return 1.0;//base case/anchor
5         else
6                 return x*power(x,n-1);//inducti
7
8 }
```

- Recursion vs iteration?

Tail recursion

- 1 recursive call as last statement in a method.
- Can easily be replaced by loops.
- Logic languages don't have explicit loops.

Non tail

- Iterative version usually requires an explicit stack.
- Reverse LL:

Indirect recursion

- Methods calling each other recursively.
- Chains of method calls...

Nested recursion

- Functions defined in terms of themselves AND accepts as a parameter an additional function call to itself.

- Ackermann:

```
1 Acker(m,n)
2 if  (n == 0)
3             m+1
4 else  if(n > 0 && m == 0)
5              Acker(n-1,1)
6 else
7              Acker(n-1,Acker(n,m-1))
```

Excessive recursion

- Recursion slows down execution (slightly).
- Recursion running too deep? Stack overflow.
- Multiple recursive calls to calculate final answer...
- Fibonacci:

```
1 //Recursive
2 int Fibo(int n)
3         if(n < 2)
4                 return n
5         else
6                 return Fibo(n-2) + Fibo(n-1)
```

```
1 //Iterative
2 int FibIter(n)
3 if(n < 2)
4         return n
5 int first = 0
6 int second = 1
7 int num
8 for(int i = 2; i <= n, ++i)
9         num = second+first
10        first = second
11        second = num
12 return num
```

```
void printBack()
{
        if (head == null)
            return ;
        Node last = head ;
        Node pred = null ;
        while(last.next != null)
        {
            pred = last ;
            last = last.next ;
        }
        while (last != null)
        {
            print(last.element);
            last = pred ;
            Node tmp = head ;
            pred = null ;
            while (tmp != last )
            {
                pred = tmp ;
                tmp = tmp.next ;
            }
        }
}
```

```
void prinBack()
{
        printList(head);
}


void printBack(Node node)
{
        if(node == null)
                return;
        printList(node.next);
        print(node.element);
}
```