CMPE 12b

SPRING 2005

- READ CHAPTER 1: PRINCIPLES OF SOFTWARE ENG.

WE will Follow THE TEXT FAIRLY closely AND COVER ROUGHLY CHAPTERS 2 - 10

CHAPTER 3: RECURSION

RECURSION IS A TECHNIQUE FOR PROBLEM SOLVING WHICH CONSISTS OF BREAKING A PROBLEM INTO SMALLER SUBPROBLEMS OF THE SAME TYPE. THESE SUBPROBLEMS ARE BROKEN DOWN FURTHER, AND FURTHER, UNTIL ALL SUBPROBLEMS ARE SO SMALL THAT THEIR SOLUTION IS EITHER OBVIOUS OR IS READILY COMPUTED BY SOME OTHER METHOD.

RECURSIVE ALGORITHMS (OR FUNCTIONS, METHODS, ETC.) CALL THEMSELVES.

RECURSIVE SOLUTIONS TO PROBLEMS ARE OFTEN VERY ELEGANT, TO THE POINT OF BEING SOMEWHAT DIFFICULT TO COMPREHEND.

UNFORTUNATELY, SUCH SOLUTIONS ARE SOMETIMES VERY INEFFICIENT.

EX THE FACTORIAL FUNCTION

$$n! = \begin{cases} n(n-1)(n-2)\cdots 3\cdot 2\cdot 1 & \text{IF } n>0 \\ 1 & \text{IF } n=0. \end{cases}$$

```
Public static int fact (int n)   //Pre: n≥0
{
    if (n > 0)
        return  n * fact(n-1);
    else
        return  1;
}
```

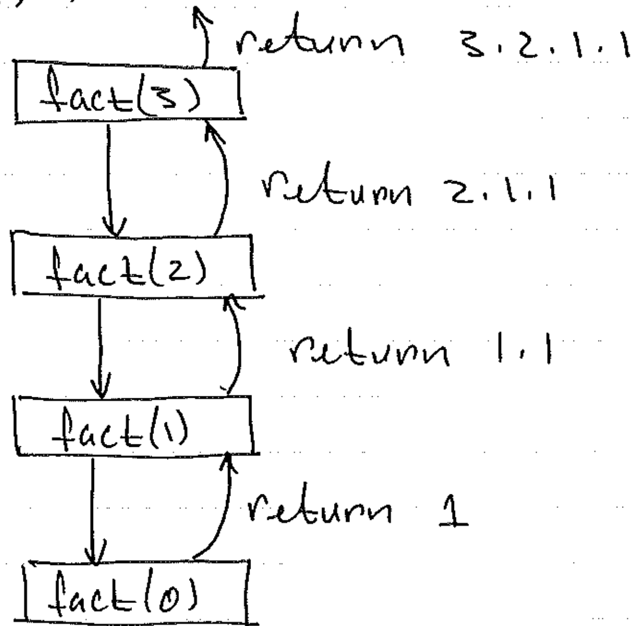
THIS METHOD SHOULD BE SURROUNDED BY A CLASS:

```
Public class example1 {

    Public static int fact(int n) {
            // ABOVE CODE
    }

    Public static void main(String[] args) {
        System.out.println( fact ( 3));
    }
}
```

HERE IS A <u>BOX TRACE</u> OF THE FUNCTION CALL fact(3) :



THE SUBPROBLEM fact(0) IS WHERE THE RECURSION "BOTTOMS OUT" SO TO SPEAK. THIS IS ALSO SOMETIMES CALLED THE <u>BASE CASE</u> OF THE RECURSION.

EVERY RECURSIVE ALGORITHM (OR FUNCTION, OR METHOD) MUST EVENTUALLY REACH A BASE CASE, i.e. A SUBPROBLEM INSTANCE WHOSE SOLUTION CAN BE OBTAINED NON-RECURSIVELY.

WHAT HAPPENS IF fact( ) ABOVE IS CALLED ON A NEGATIVE NUMBER ?

<u>ANSWER</u>: INFINITE REGRESS, PRACTICALLY THIS MEANS THE PROGRAM WOULD HALT WHEN A SYSTEM DEFINED RECURSION LIMIT IS REACHED.
<u>EXERCISE</u>: TRY THIS.

<u>Ex.</u> BINOMIAL COEFFICIENTS

$C(n,k)$ = # k-ELEMENT SUBSETS OF AN n-ELEMENT SET WHERE $0 \le k \le n$.

e.g.) $S = \{1, 2, 3\}$

3-ELEMENT SUBSETS OF S: S
2-ELEMENT SUBSETS OF S: $\{1,2\}, \{1,3\}, \{2,3\}$
1-ELEMENT SUBSETS OF S: $\{1\}, \{2\}, \{3\}$
0-ELEMENT SUBSET OF S: $\emptyset$.

$\therefore$ $C(3,3) = 1$, $C(3,2) = 3$, $C(3,1) = 3$, $C(3,0) = 1$.

~~LET S BE ANY n-ELEMENT SET, AND LET 0 ≤ k ≤ n, OBSERVE THAT THE k-ELEMENT SUBSETS OF S FALL IN~~

LET S BE AN n-ELEMENT SET, AND $0 \le k \le n$. LET x BE ANY ELEMENT OF S. THEN OBSERVE THAT THE k-ELEMENT SUBSETS OF S FALL INTO 2 CATEGORIES:

- THOSE WHICH CONTAIN x: $C(n-1, k-1)$
  (PICK $k-1$ ELEMENTS FROM $S - \{x\}$, ADD x.)
- THOSE WHICH DO NOT CONTAIN x: $C(n-1, k)$
  (PICK k ELEMENTS FROM $S - \{x\}$.)

THUS $C(n,k) = C(n-1, k-1) + C(n-1, k)$

THIS IS KNOWN AS PASCAL'S IDENTITY.
INCLUDING BASE CASES, AND OUT OF BOUNDS CASES WE HAVE

$$C(n,k) = \begin{cases} 0 & \text{if } k > n \\ 1 & \text{if } k=0 \text{ or } k=n \\ C(n-1,k-1)+C(n-1,k) & \text{if } 0<k<n \end{cases}$$

THIS IDENTITY GENERATES PASCAL'S TRIANGLE!

$n=0$ 1
$n=1$ 1 1
$n=2$ 1 2 1
$n=3$ 1 3 3 1
$n=4$ 1 4 6 4 1
⋮

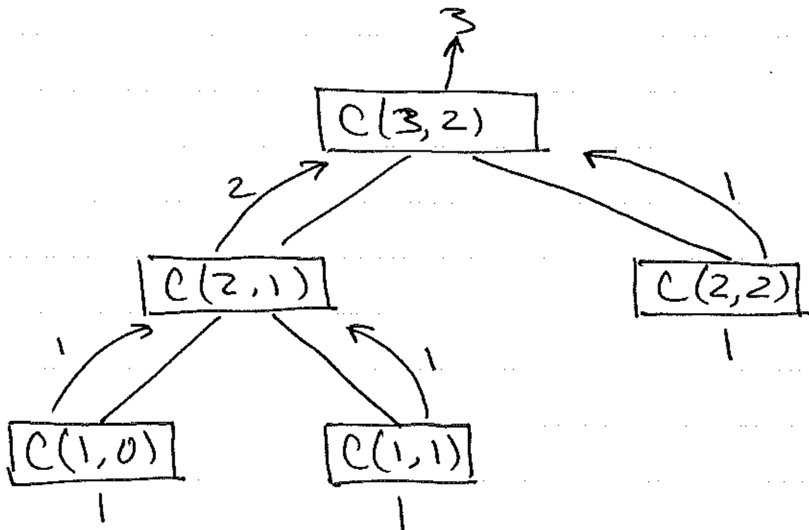
THIS IDENTITY ALSO PROVIDES A RECURSIVE SOLUTION TO THE PROBLEM OF DETERMINING THE NUMBER OF K-ELEMENT SUBSETS OF AN N-ELEMENT SET.

```
Public static int C(int n, int k) {
// Pre: n ≥ 0, k ≥ 0

    if (k > n)
        return 0;
    else if (k == 0 || k == n)
        return 1;
    else
        return C(n-1, k-1) + C(n-1, k)
}
```

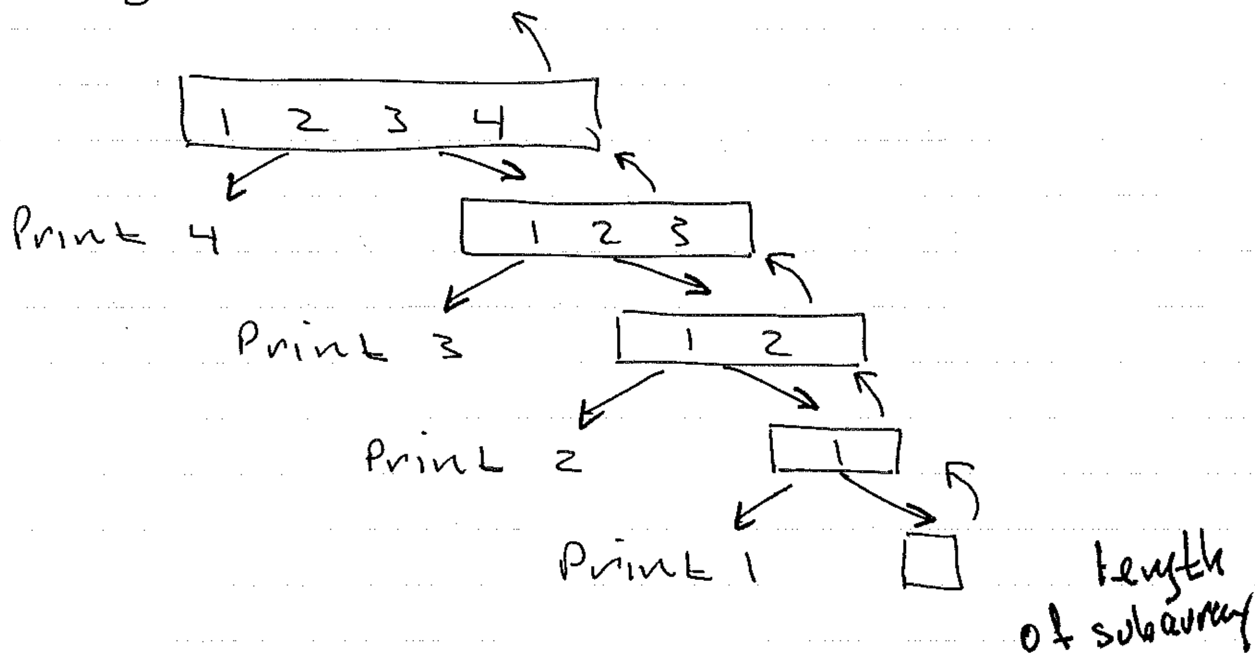
EXERCISE: ENCLOSE THIS IN A CLASS, AND CALL C(n,k) FOR SEVERAL VALUES OF n & k (INCLUDING SOME ILLEGAL ONES.)

BOX TRACE OF THE CALL: C(3, 2)



EX. PRINT OUT AN int ARRAY IN REVERSE ORDER.

e.g.) length n = 4




```
// Reverse.java
Public class Reverse {
    Public static void reverseArray(int[] A, int n){
    // if n==0 do NOTHING else
      if (n > 0) {
          System.out.Print(A[n-1] + " ");
          reverseArray(A, n-1);
      }
    }

    Public static void main(String[] args) {
       int[] B = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

       reverseArray(B, 10);
       System.out.Println();
    }
}
```

Ex.

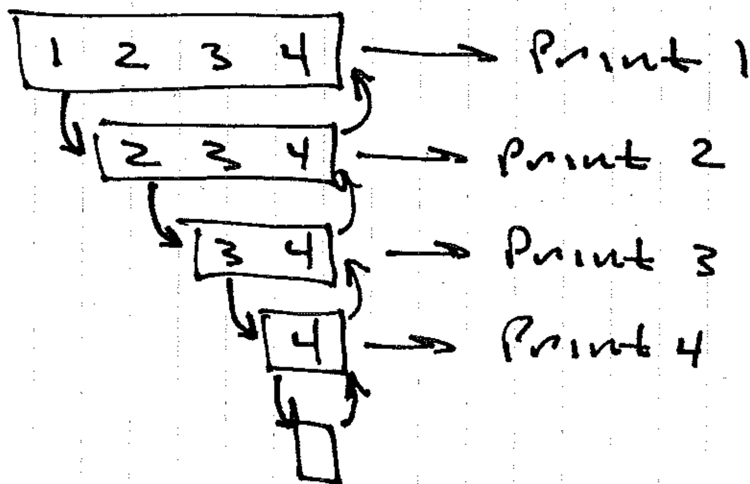
Length of subarray TO BE REVERSED

```
Public static void reverseArray(int[] A, int n){
    // if n == 0 do nothing.
    if (n > 0) {
        reverseArray(A, n-1);
        System.out.print(" " + A[A.length - n]);
    }
}
```

~~In main~~: Same main.

TRACE:

n	index
4	0
3	1
2	2
1	3
0	



IN THIS EXAMPLE, EACH RECURSIVE INVOCATION OF FUNCTION reverseArray( ) DOES TWO THINGS.

- PEEL OFF THE 'LAST' ELEMENT OF A AND PRINT IT OUT FOLLOWED BY A SPACE
- RECURSIVELY CALL ITSELF ON THE SUB-ARRAY A[1....(n-1)].

EACH RECURSIVE INVOCATION REDUCES THE SUBPROBLEM SIZE BY 1.

Ex. BINARY SEARCH: SEARCH A SORTED ARRAY OF INTS FOR A SPECIFIED TARGET. RETURN THE INDEX AT WHICH THE TARGET IS FOUND, OR -1 IF TARGET IS NOT FOUND.

STRATEGY: COMPARE TARGET T TO ELEMENT IN THE 'MIDDLE', IF IT MATCHES, FINE. IF NOT RECURSIVELY CALL ON EITHER 'LEFT OR RIGHT SUBARRAY, DEPENDING ON COMPARISON.

e.g.)