CSC 143

Recursion

Recursion

- A recursive definition is one which is defined in terms of itself.
- Example:
  - Sum of the first n positive integers
    - if n>1, equal to n + sum of the first n-1 positive integers
    - if n=1, the sum is 1 (base case)
  - Palindrome
    - if the number of characters is >1, a piece of text is a palindrome if it starts and ends with the same letter and what is in between is a palindrome.
    - a word with 0 or 1 character is a palindrome (base case)

Motivation

- Divide and conquer
  - Express the problem in terms of a simpler problem
- Factorial n
  - with a loop n!=1*2*3*…*(n-1)*n
  - ≠ with recursion

    n!=n*(n-1)! if n>1

    1!=1

n! with a loop

- Compute n!=1*2*…*(n-1)*n

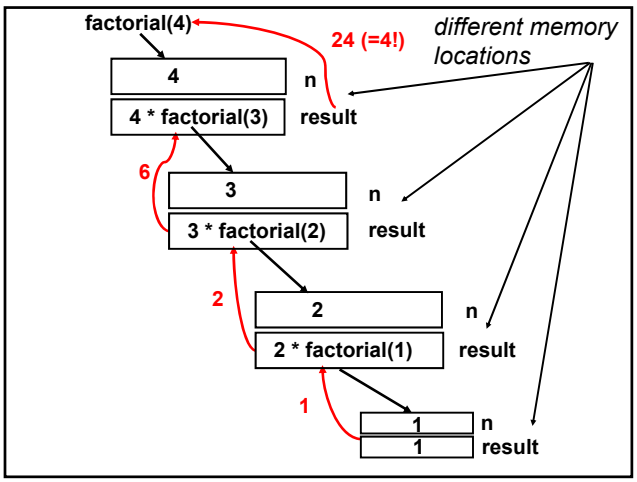
```
public long factorial(int n)
{
  // with a  loop
  long result=1;
  while(n>1){
     result*=n;
     n--;
  }
  return result;
}
```

n! with recursion

- Write n!=n*(n-1)!

```
public long factorial(int n)
{
  // with recursion
  long result;
  if (n>1)
    result=n*factorial(n-1);
  else
    result=1; // base case
  return result;
}
```

- How does it work? Recall that a method call is done by value in java.



Activation Records

- Recall that local variables and parameters are allocated when a method is entered, deleted when the method exits (automatic storage).
- Whenever a method is called, a new activation record is pushed on the call stack, containing:
  - a separate copy of all local variables
  - control flow info (e.g. return address)
- Activation record is popped at end of method
- A recursive method call is handled the same way
  - Each recursive call has its own copy of locals

Infinite Recursion

- Make sure that recursion will eventually end with one of the base cases.

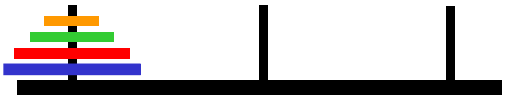
```
public long factorial(int n) {
   long result = factorial(n-1); //oups!
   if ( n==1) // the control flow never gets
              // to this line
     result=1;
   else
     result *=n;
  return result;
}
```

- What happens? Whatever the value of n, factorial(n-1) is called. It ends with a stack overflow error.
- Make sure that you test for the base case before calling the method again.

Recursion versus loops

- Any recursive algorithm can be rewritten as an iterative algorithm
- What is best?
  - Some problems are more elegantly solved with recursion. Others with iterations.
  - Recursion is slightly more expensive. An activation record is pushed on and later popped from the stack for each recursive call

The towers of Hanoi (1)



- Move the tower of disks from the left to the right peg.
- A larger disk can't be placed on top of a smaller disk. Solution?
- Beautifully solved with recursion. More difficult with a loop.

The towers of Hanoi (2)

- Recursive algorithm: move n disks in terms of moving n-1 disks
  - Base case: 1 disk
  - To move n disks from the left to the right peg,
    - move the n-1 top disks from the left to the middle peg
    - move the one remaining disk on the left peg to the right peg
    - move the n-1 disks on the middle peg to the right peg.

The towers of Hanoi (3)

```
public void move
(int n,int peginit,int pegfinal, int pegtemp) {
 // move n disks from peginit to pegfinal, using
 // pegtemp as a temporary holding area
 if (n ==1)
  {System.out.println("move top disk from "+
   peginit+" to " + pegfinal);}
 else
  {
   //move n -1 disks to pegtemp
   move(n-1,peginit,pegtemp,pegfinal);
   //move the remaining disk to pegfinal
   move(1,peginit,pegfinal,pegtemp);
   //move n -1 disks to pegfinal
   move(n-1,pegtemp,pegfinal,peginit);
 }
}
```