# ECE 2574: Data Structures and Algorithms - Recursion Part I

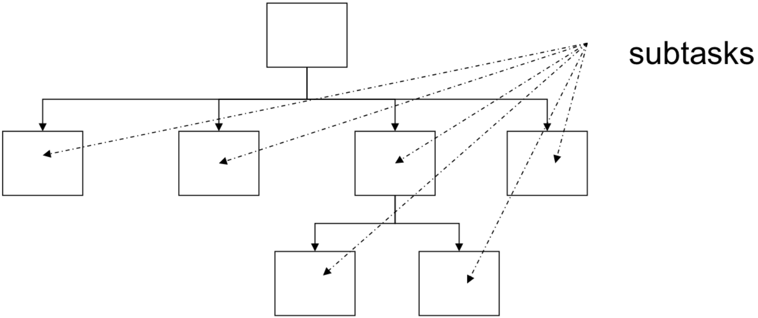
C. L. Wyatt

# Today we will introduce the notion of recursion, look at some examples, and see how to implement them in code.

- Introduction to recursion
- Warmup
- Examples
- Exercise: Recursive Egyptian Powers

# Top-down Design

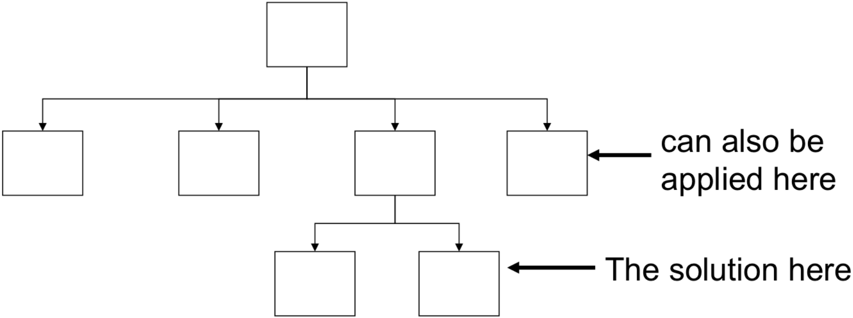
Top down design divides problems into smaller and easier sub-problems.



The hope is, at each of these successive levels, these sub-problems are easier to solve.

# Recursion

In some cases, the solution to lowest level sub-problem can be applied at the higher level.



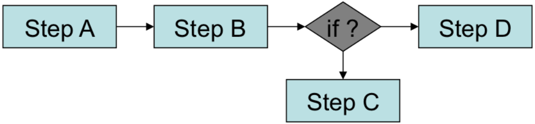
can also be applied here

The solution here

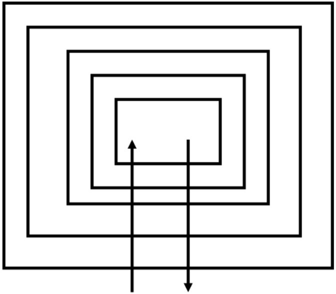
This type of solution is called *recursive*.

# A graphical view

Graphical depictions of algorithms show connected boxes.



# Recursive solutions form nested boxes.



- The inner-most box solves the lowest-level problem,
- The next box solves the next level above.
- In a recursive solution the algorithm is the same at each level.

# Example: n!, the factorial

Simple iterative solution to compute the factorial

```
int factorial (int n)
{
  int result = n;
  do
  {
    n -= 1;
    result = result*n;
  } while (n > 1);

  return result;
};
```

How large an n will this work for in a real programming languages?

# We can break the factorial solution into a recursive solution.

```
n! = 1*2*3*4*......*n
```

grouping terms

```
n! = 1*2*3*4*.......*(n -1)*n
n! = (     (n -1)!        )*n
```

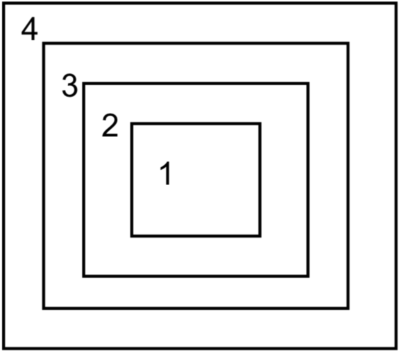
This is an example of a recurrence relation.

# A recursive solution to the factorial

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

# A recursive solution to the factorial

Graphically each box takes the output of the box inside it and multiplies by the integer in that box.
Example: 4! = 4*3*2*1 = (4*(3*(2*(1))))



# Formal definition of recursion

- A recursive procedure is one whose evaluation at (non-initial) inputs involves invoking the procedure itself at another input.
- In the case of the factorial this involves invoking the procedure at (n-1) when the input is n: n! = n*(n-1)!
- Recursion is a very powerful tool in the design and analysis of algorithms.
- Often complex problems have very simple recursive solutions.

# What makes recursive procedures work ?

* At each invocation, the solution must get closer to a known solution
i.e. $0! = 1! = 1$

- The procedure calls must terminate in a finite number, that is the function must not endlessly call itself. Otherwise the recursion is *infinite*

# Recursive version of the GCD algorithm

Recall the GCD algorithm

- A.0 If m $<$ n, swap m and n.
- A.1 Divide m by n and let r be the remainder.
- A.2 If r $=$ 0, terminate; n is the answer.
- A.3 Set m to n, n to r, and go back to step A.1.

A recursive solution (after step 0).

```
int gcd(int m, int n)
{
  if( n == 0 ) return m;
  else return gcd(n, m%n);
}
```

# Recursive version of the GCD algorithm

The recurrence relation for GCD

```
gcd(m,n) = gcd(n, m mod n)
```

The stopping condition (base case) is n = 0
Example:

```
gcd(131,62) -> gcd(62,7) -> gcd(7,6)
-> gcd (6,1) -> gcd(1,0)
```

# Warmup #1

Which of the following C++ functions correctly computes the sum from 1 to n using recursion?

```
int function1(int n){
  int sum = 0;
  for(int i = 1; i <= n; ++i){
    sum += i;
  }
  return sum;
}
```

Incorrect. (15%).

# Warmup #1

Which of the following C++ functions correctly computes the sum from 1 to n using recursion?

```
int function2(int n){
  if(n == 1) return 1;
  return n + function2(n-1);
}
```

Correct (81%).

# Warmup #1

Which of the following C++ functions correctly computes the sum from 1 to n using recursion?

```
int function3(int n){
  return n*(n+1)/2;
}
```

Incorrect. (3%).

# Warmup #2

What would happen if function1 in the previous question was called with an argument of -1?

```
int function1(int n){
  int sum = 0;
  for(int i = 1; i <= n; ++i){
    sum += i;
  }
  return sum;
}
```

The correct answer is “it would return 0” (63%).

# Warmup #3

What would happen if function2 in the previous question was called with an argument of -1?

```
int function2(int n){
  if(n == 1) return 1;
  return n + function2(n-1);
}
```

The correct answer is “A run-time error would occur” (73%).

# Warmup #4

What would happen if function3 in the previous question was called with an argument of -1?

```
int function3(int n){
  return n*(n+1)/2;
}
```

The correct answer is “It would return 0” (85%).

# Another Example: Exponentiation

- The Egyptian Powers algorithm computes x to the power n by repeated squaring.
- The recurrence relation for computing $x^n$ for any positive integer n:

$$x^n = \begin{cases} (x \cdot x)^{n/2} & n \text{ even} \\ x(x \cdot x)^{(n-1)/2} & n \text{ odd} \end{cases}$$

# Exercise: write a recursive function and a set of tests implementing the Egyptian Powers algorithm.

In pseudo-code

```
function RecPowers (x, n)
Input: a real number x and positive integer n
Output: x raised to power n

    if (n == 1) // initial condition
        pow = x;
    else
        if even(n) then
            pow = RecPowers(x*x,n/2)
        else
            pow = x*RecPowers(x*x,(n-1)/2)
        endif
    endif
    return (pow)
endfunction
```

# Next Actions and Reminders

- Read CH pp. 67-87
- Warmup before noon on Monday.
- Program 0 due tonight by 11:59 PM.