Recursion (1)

- *def:* *Recursion* is a technique for defining a problem in terms of one or more smaller versions of the same problem. The solution to the problem is built out of the results from the smaller versions.

  Example: Let $n \in \mathbf{N}$. Then the following functions are *recursive* functions.

  1. The factorial of n
  $$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n > 0 \end{cases}$$
  2. The exponential function
  $$a^n = \begin{cases} 1 & \text{if } n = 0 \\ a \cdot a^{n-1} & \text{if } n > 0 \end{cases}$$ where a is any non–zero real number.
  3. The *Fibonacci sequence*
  $$F(n) = \begin{cases} 1 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n \geq 2 \end{cases}$$

Recursion (2)

- Self-referential definitions can be dangerous if we are not careful to avoid circularity.
- Problems that lend themselves to recursive solutions have the following characteristics:
  1. The problem can be redefined in terms of one or more subproblems, identical in nature to the original problem but in some sense smaller in size.
  2. One or more base cases of the problem have direct or known solutions.
  3. By applying this redefinition process to ever smaller subproblems, eventually the problem is reduced entirely to the base cases.
  4. The base case solutions can be used in some way to build the solution to the whole problem.

Recursion (3)

- **A base case** is an instance of the problem whose solution requires no further recursive definition.
  - It acts as a terminating condition.
  - It is the building block of the complete solution of the problem.
- **Key concepts:** When we attempt to construct a recursive solution of a problem we should keep in mind the following four questions:
  1. How can we define the problem in terms of one or more smaller problems?
  2. What instances of the problem can serve as the base cases?
  3. As the problem size diminishes will we reach these base cases?
  4. How are the solutions from the smaller problems used to build a correct solution to the larger problem?

If expressions

The recursive definition of functions generally consists of an if statement with the following form:

if (this is a base case)
    then solve it directly
    else if (this is not a base case)
        redefine the problem using recursion
end if

- *def:* An *if expression* is an expression of the form:

  if *pred* **then** *expr1* **else** *expr2* **end**

  - **if** $x \geq 0$ **then** x **else** $-x$ **end** $[abs(x)]$
  - **if** $n \geq m$ **then** n **else** m **end** $[max(x, y)]$

- If expressions can be nested as follows:

if $pred_1$
    then $expr_1$
    else **if** $pred_2$ **then** $expr_2$
                else $expr_3$
            end
end

Explicit definition of functions

- The absolute value function was defined as

  $f : \mathbf{R} \rightarrow \mathbf{R}$ such that

  $$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

- The *signature* and the expression defining the result of the function $f(x)$ in terms of the argument x can be merged into one definition called an *explicit function definition.* For example:

  $f : \mathbf{R} \rightarrow \mathbf{R}$
  $f(x) =$
  if $(x \geq 0)$
  then x
  else $-x$
  end
  end f

  where the variable x is called the *formal parameter* of f and takes any value from the domain $\mathbf{R}$.

An example: Factorial function (1)

The problem is to compute the factorial of a natural number n recursively.

Design: The familiar iterative definition of the factorial of n (or $n!$) is
$n! = n \cdot (n-1) \cdot (n-2) \cdots 1 \qquad \text{for } \; n > 0$
$0! = 1$

- How can we define the problem in terms of smaller problems of the same type? We need to define $n!$ in terms of the factorial of a smaller number:

  $n! = n \cdot (n-1)! \qquad\qquad$ *for* $n > 0$

- What instance of the problem can serve as the base case? The smallest number for which we can calculate the factorial is 0. Thus

  $n = 0$ *is the natural choice for our base case.*
  $0! = 1$ *by definition.*

An example: Factorial function (2)

- As the problem size diminishes will we reach this base cases?

  Since each application of the function reduces the parameter n *by 1, and* n *is non-negative, we will always reach the base case* $n = 0$ *eventually.*

$$
\begin{array}{l}
factorial : \mathbf{N} \rightarrow \mathbf{N} \\
factorial(n) = \\
\qquad\qquad \textbf{if} \;\; (n = 0) \\
\qquad\qquad\quad \textbf{then} \;\; 1 \\
\qquad\qquad\quad \textbf{else} \;\; n * factorial(n-1) \\
\qquad\qquad \textbf{end} \\
\textbf{end} \;\; factorial
\end{array}
$$

- How is the solution from the smaller problem used to build a correct solution to the larger problem? *The result returned from the call to* $factorial(n-1)$ *is multiplied by* n *to obtain* $factorial(n)$.

An example: The Towers of Hanoi (1)

The *Towers of Hanoi* problem is a classic case study in recursion. It involves moving a number of different size disks, stacked on a peg in order of decreasing size, from one tower to another using a third tower as an auxiliary.

Problem: Move n disks from peg A to peg C, using peg B as needed, according to the following rules:

1. Only one disk may be moved at a time.
2. This disk must be the top disk on a peg.
3. A larger disk can never be placed on top of a smaller disk.

An example: The Towers of Hanoi (2)

- *The key to the problem is to concentrate our attention on the hardest step, i.e. moving the bottom disk to peg C. There is no way to move the bottom disk until the top* $n-1$ *disks have been moved. We now have* $n-1$ *disks on peg B which must be moved to peg C using peg A.*

  So the problem is equivalent to:

  - *move* $n-1$ *disks from peg A to peg B using peg C*
  - *move the n'th disk from peg A to peg C.*
  - *move* $n-1$ *disks from peg B to peg C using peg A*

- *If* $n=1$ *then the problem consists of moving 1 disk from a given peg to another, which we can solve immediately.*

- *Since each call to the function reduces the parameter* n *by 1, and* n *is positive, we always reach the base case* $n=1$.

- *When all the three smaller problems are finished the larger problem is completed.*

An example: The Towers of Hanoi (3)

- Let Peg be the set of the three pegs, $Tower$ be a set of well-defined towers in the sense that any tower consists of three pegs with disks stacked smaller ones on larger ones in some way, and let

$$take\_put: \; Peg \times Peg \times Tower \rightarrow Tower$$

be a function which moves the top disk from one peg to another in a given tower. Then

$move: \; \mathbf{N} \times Peg \times Peg \times Peg \times Tower \rightarrow Tower$

$move(n, from, to, use, t) =$

  if $(n = 1)$

    then $take\_put(from, to, t)$

    else

        $move(n-1, use, to, from,$

                $take\_put(from, to,$

                        $move(n-1, from, use, to, t)))$

  end

end $move$