Control

Conventional imperative languages such as Fortran, Cobol and Pascal rely heavily on the use of iterative constructs for evaluation. Functional programming style relies heavily on the use of recursion. Almost everyone has met the factorial function:-

```
    fact n         is defined as       {Factorial}
                             if n = 0
                             then 1
                             else n * fact (n-1)
```

Other examples are:-

```
    add x y        is defined as       {Addition}
                             if x = 0
                             then y
                             else add (x-1) (y+1)

    fib n          is defined as       {Fibonacci function}
                             if n = 0 then 1
                             else if n = 1 then 1
                                  else fib (n-1) + fib (n-2)

    mult x y       is defined as       {Multiplication}
                             if x = 0
                             then 0
                             else y + mult (x-1) y

    div  x y       is defined as       {Integer division}
                             if x < y
                             then 0
                             else 1 + div (x-y)  y

    rem  x y       is defined as       {Remainder}
                             if x < y
                             then x
                             else rem (x-y) y

    gcd  x y       is defined as      {Greatest-Common-Divisor}
                             if x = y
                             then x
                             else if x > y
                                  then gcd (x-y) y
                                  else gcd y x
```

All of the examples given so far are of self-recursion. In this form of recursion a function may call itself. It is also possible for functions to be mutually recursive. Mutually recursive functions are functions which call each other. To illustrate this, modify the gcd function:-

```
    gcd1 x y       is defined as
                             if x = y then x
                             else gcd2 x y

    gcd2 x y       is defined as
                             if x > y
                             then gcd1 (x-y) y
                             else gcd2 y x
```

In this example, gcd1 and gcd2 are mutually recursive. Note also that gcd2 is also self-recursive.

In many programming languages allowing recursion it is necessary to define mutually recursive functions in a special way. This is essential if functions can only be defined in terms of already existing functions. In the gcd1/gcd2 example, it is not possible to define gcd1 first because gcd2 is not defined - similarly it is not possible to define gcd2 first. Simultaneous definition seems to be the only way out! In ML functions which are to be defined simultaneously are defined with the 'and' keyword separating them:-

```
    fun gcd1 x y = if x = y
                   then x
                   else gcd2 x y

    and gcd2 x y = if x >y
                   then gcd1 (x-y) y
                   else gcd2 y x;
```

In some languages, definitions of self-recursive functions will also need special treatment, as a function definition may only be made available to the language system for use once it has been defined.

It is important to note that recursive functions must include at least one conditional test within the recursive function set in order to avoid a non-terminating computation:

```
    funny1 x       is defined as       1 + (funny2 (x+1))

    funny2 x       is defined as       2 * (funny1 (x*3))
```

Attempting to use either funny1 or funny2 will cause a failure due to non-termination.