Functional Programming – Miranda

History:

Miranda was developed in 1985-86 by David Turner, and is now currently being marketed by Research Software Ltd. of England. Miranda was the inheritor of the functional languages SASL and KRC. With Miranda, the main purpose was to produce a commercial version of a standard non-strict purely functional language. To make Miranda commercially possible, the development environment had to be made very flexible and easy-to-use.

Application:

Miranda is an advanced functional programming system which runs under the UNIX operating system. The aim of the Miranda system is to provide a modern functional language, set in an ‘industrial quality' programming environment. It is now being used at a growing number of sites for teaching functional programming and as a vehicle for the rapid prototyping of software.

Two Basic Data Structures:
**Lists** - defined as a set of homogeneous typed values. The list data structure is an extremely powerful feature of Miranda. It allows quick and simple list processing and also allows infinite lists.

The list in Miranda is written with square brackets and commas, eg:

```haskell
week_days = ["Mon","Tue","Wed","Thur","Fri"]
days = week_days ++ ["Sat","Sun"]
```

**Tuple** - a non-homogenous sequence of values which can be used to form enumerated data types or even form complex data structures such as records.

A tuple is written using parentheses instead of square brackets. Example:

```haskell
employee = ("Jones",True,False ,39)
```

**Feature**

**Non-Strict**

In Non-Strict functional languages, the arguments to a function are not evaluated until they are actually required within the functions being called. Therefore, any parameter can be passed to a function and until it is needed in that function, this parameter is not evaluated. This is also known as lazy evaluation, and the main advantage of using this method is
that it allows for passing infinite element data structures to a function.

**Purely Functional**

Pure functional languages perform all computation using function application. "Side-effect" features such as destructive assignments and looping are not even provided within the language, and therefore all programs have to strictly adhere to the functional approach of programming.

**The order of execution is irrelevant**

The order in which the equations are given is not in general important. For example, there is no requirement for the definition of an entity to precede its first use. Here is a very simple example of a Miranda script:

\[
\begin{align*}
  z &= \text{sq } x \div \text{sq } y \\
  \text{sq } n &= n \times n \\
  x &= a + b \\
  y &= a - b \\
  a &= 10 \\
  b &= 5
\end{align*}
\]

**Infinite Lists:**

The other main consequence of lazy evaluation is that it makes it possible to write down definitions of infinite data structures.

There is a shorthand notation using ".." for lists whose elements form
an arithmetic series. Here for example are definitions of the factorial
function, and of a number "result" which is the sum of the odd numbers
between 1 and 100:

\[
\text{fac } n = \text{product } [1..n]
\]

\[
\text{result} = \text{sum } [1,3..100]
\] (sum and product are library functions)

Here are some examples of Miranda definitions of infinite lists:

\[
\text{nats} = [0..]
\]

\[
\text{odds} = [1,3..]
\]

**Polymorphic strong typing**

Miranda is strongly typed. That is, every expression and every
sub-expression has a type, which can be deduced at compile time, and
any inconsistency in the type structure of a script results in a compile
time error message.

There are three basic types, called **num**, **bool**, and **char**. The type
num contains integer and floating point numbers. There are two values of
type bool, called True and False. The type char contains the ASCII
character set - character constants are written in single quotes, using C
escape conventions, e.g. 'a', '$', '\n' etc.
If $T$ is type, then $[T]$ is the type of lists whose elements are of type $T$.

For example $[[1,2],[2,3],[4,5]]$ is of type $[[\text{num}]]$, that is it is a list of lists of numbers. String constants are of type $[\text{char}]$, in fact a string such as "hello" is simply a shorthand way of writing ['h','e','l','l','o'].

If $T_1$ to $T_n$ are types, then $(T_1,...,T_n)$ is the type of tuples with objects of these types as components. For example (True,"hello",36) is of type (bool,[char],num).

Reference: