Eiffel Origins

Eiffel, the Tower,
constructed by Gustav Eiffel (1832-1923) in 1888

Eiffel, the Language,
contructed by Bertrand Meyer in 1986

Influences on Eiffel

Algol 60 ← Simula ← Eiffel
(← ‘derived from’)

Hoare remarked about Algol 60,
“that it was such an improvement over
most of its successors”

- Algol 60 -- Naur Report '60 & '63
  America / Europe committee

- Simula 67
  Dahl, Myhrhaug & Nyard (Norway)
  Inheritance, encapsulation, information hiding.
  Simulation language but also general purpose

- Smalltalk
  Alan Kay Xerox PARC 1980
  influenced by Seymor Papert and LOGO.
  Used to develop OS/2

- Eiffel
  Meyer 1986   Eiffel Studio .NET   2002

Eiffel Implementations

- Eiffel Studio (www.eiffel.com)
  (Eiffel Software Inc. USA)
  by Bertrand Meyer
  -- On Order for TCD
  Personal Eiffel (Graphical, Win 95/NT)
  -- Free version available

- SmartEiffel (Completely Free)
  http://smarteiffel.loria.fr/index.html
  Installed on TCD PCs (AP 0.13)

- Visual Eiffel   (Visual Eiffel Lite -- Free)
  from Object Tools (http://www.object-tools.com)
  (Used in DIT)

Eiffel Control Instuctions

Assignment

```
x := e
```

Selection -- **if _ then _ else**

```
if b then
    S1
else
    S2
end
```

Sequencing

```
S1;
S2;
 ...
Sn
```

Semi-colons are optional. (‘; ’ is a separator)

Iteration (loop command)

```
from
        <Init>
until
        <Boolean Condition>
loop
        <Body of loop>
end
```

Routines (Functions or Procedures)

Functions

```
fname ( f1 : T1;  f2 : T2;  ...  fn : Tn) : T is
local
   <Local declarations>
do
    <Body of function >
    result := expr  -- must be included
end -- fname
```

Function call

e.g. x := fname (a1, a2, .. an)

x must be of type T, the type returned by the function.

Example:

```
product(m,n : INTEGER) : INTEGER is
    -- returns product m * (m+1) ... * n, if m ≤ n
    local
        k, r : INTEGER
    do
        from
            k := m
            r := 1
        until
            k > n
        loop
            r := r*k
            k := k+1
        end
        result := r
    end -- product
```

Note:

1. If m > n then product(m,n) = 1
2. product(1,n) returns n! (factorial), if n > 0

Procedures

```
pname ( f1 : T1; f2 : T2; ... fn : Tn) is
    local
        <local declarations>
    do
        <Body of procedure>
    end -- pname
```

Procedure call: pname (a1,a2 .. an)

```
sort (a: ARRAY[STRING]; low, high: INTEGER) is
        local
            k: INTEGER;
            bs: BINARY_SEARCH[STRING]
        do
            from
                !! bs;
                k := low + 1
            until
                k > high
            loop
                bs.search(a, low, k-1, a.item(k));
                insert(a.item(k),bs.index+1,a, low, k-1);
                k := k + 1
            end
        end ;
```