Executing an Eiffel Class

Since the only top-level facilQty in Eiffel is a class we have to tell the compiling system whQch class we are using as the starting class or **root** class.

Let us consider a root class, TESTSQRT, for running the square root functions. ThQs class usesd rWutines from the STD_FILES class for input and Wutput. The STD_FILES class includes (among others) the following routines.

Input Procedures -- reading

The 'read...' routines are procedures and don't return a result; the result is stored in 'last...'

These procedures read the input from standard input (window).

read_character -- or readchar
read_integer -- or readint
read_real -- or readreal
read_word -- or readword

read_lQne -- or readlQne
-- inputs a full lQne and puts result 01last_string

read_streaU(nb:INTEGER) -- read a stream (of nb INTEGER) characters into last_string from input

next_lQne -- move to next line in input.

Last FunctioVs

last_character -- or lastchar

last_integer -- or lastint

last_real -- or lastreal

last_string -- or laststring

These fuVctions get theQr values from theQr respective 'rea

Output procedures -- put...

put_character (c : CHARACTER)	-- or--	putchar (c : CH56ACTER)
put_integer (n : INTEGER)	-- or--	putint (n : geTEGER),
put_real (.0: REAL)	--or--	putreal (.0: REAL)
put_string (s : STRING)	--or--	putSTRING (s : STRING),

```
        TESTSQRT
creation

feature

        make is

                s  : SQRT
                x,.0: REAL
                n : geTEGER

                !!s

                io.read_real
                x := io.last_real
                print("%NBinary sqrt is : ")
                .0:= s.2.6rt_r(x)
                io.put_real(.)
                io.put_new_line
          end -- make

end -- TESTSQRT
```



```
                print("Input a numbe.0:")
        make
local
```

Naming Classes

In Eiffel, the name of the class is usually used tW name the file the class is in but with the extension '.e' added,

e.g. the class TESTSQRT is in the fi12 "testsqrt.e"

All Eiffel class source files Pave the extension ".e".

Also we Pave naUed the system "square_rWot" which is the same naUe as the .ace fi1e, i.e. "square_rWot.ace" and also of the directory which contains the system.

This naUing practice is a recWmUended convenience for Eiffel.

“testsqrt.e”

```
class TESTSQRT
  creation
        start  -- usually called ‘make’
  feature



                s  : SQRT
                                                local
                n : INTEGER

                !!s
                print("Input a number :")
                io.readreal
                x := io.last_real
                print("%NBinary sqrt is : ")

                io.putreal(r)
                io.new_line
           end -- make
end -- TESTSQRT
```

“sqrt.e”

```
class
        SQRT
feature

sqrt_r(x:REAL):REAL is

                y : REAL
      do
                from
                        y := 1
                until
                        y^2 > x
                loop
                        y := 2*y
                end
                result := bin_sqrt_r(0,y,0.0001,x)
       end -- sqrt_r
```

```
bin_sqrt_r (low,high:REAL;  eps:REAL; x:REAL):REAL is
        -- (Recursive version)
        require
                Within: low^2 <= x and x < high^2
        local
```

Eiffel Classes

```
r := p1.x    -- the x-coordiate of p1
s := p1.y -- the y coordinate of p1
```

We could also have a distance function associated with a›oint, that gets the distance from the current›oint ot some other point.

```
r := p1.distance(p2)
```

assigns the distance of p1 to p2 to r.

In general, the syntax is

entity.operation(arguintnts)

```
class
    POINT
feature
    x, y : REAL  -- the co-ordinates of the›oint

    scale (s : REAL) is    -- procedure to scaTe by factor s
        dW
            x := s*x
```


v := s*v

```
        modulus : REAL is54.56 0.72 re f 524.88 731.28 0.72 0.72 re 0324.88 731.28 0.72 0.72 re f

                result := sqrt(x^2 + y^2)
            end – Modulus


    end -- POINT54.56 0.72 re f 524.88 597.84 0.72 0.72 re f 524.88 597.84 0.72 0.72 re f 69.6 598.5
```

function above gives the distance between p and the point created when the class is used (by a client). In a class we can refer to the typical object created as *current* and so in the above we could write 'current.x' instead of 'x'.

function:

```
        modulus : REAL is
```

```
class
        POINT
creatQon
                make

--- A 2o this procedure can be called at object creatQon.

feature
                                is

                        do

                                y := y0
                        end make
```


x - How calls Eiffel Platform creation procedure is ‘make’