Operator Evaluation Order - I

- **Precedence**
  - **Operator precedence rules define the order in which operators of different precedence levels are evaluated:**

Parenthetical groups	(...)
Exponeniation	**
Mult & Div	* , /
Add & Sub	+ , -
Assignment	:=

A * B + C ** D / E - F = ((A * B) + ((C ** D) / E) - F)

Operator Evaluation Order - II

- **Associativity**
  - **Operator associativity rules define the order in which adjacent occurrences of operators with the same precedence levels are evaluated:**

Left associative	* , / , + , -
Right associative	**

B ** C ** D - E + F * G / H =

(((B ** (C ** D)) - E) + ((F * G) / H))

Side Effects and Order of Operand Evaluation

- **A functional side effect occurs when a fucntion either changes one of its parameters or a global variable**

Consider:

```
Procedure sub 1 (...);
    var a : integer;
        function fun (x : integer) : integer;
            x :=  a + 2;
            return (x)
        end;
    a := 7;
    b := a + fun ( a );
    print ( a , b );
end;
```

w/o SE:	7 , 16	
w/ SE:	9 , 16	a , fun(a)
w/ SE:	9 , 18	fun(a) , a

Overloading

An operator or function is overloaded if its meaning depends on the number or types of its arguments

- **+ (real, integer)**
- **- (unary, binary)**

Also called ad-hoc polymorphism

Some languages permit overloading of user-defined functions.

Good idea??

Three kinds of overloading:

- literal
- operator
- subprogram

(These are covered in three different sections in Sebesta)

Coercion

- **Implicit, or automatic, type coercion of operators.**

```
var x: integer;
    y, z: real;
     ...
y := x + z;
```

  ➡ **"mixed-mode" expression**

- **Coercion is different from explicit type conversion:**

```
int x,y
...
(double) x / (double) y
```

However if we treat (double) as a function reference for explicit type conversion then there still remains the derivation of the type of the / operator to double.

Coercion Extremes (Algol 68)

```
int    i;
real   r;
[1:10] int row i;
ref int ref i;
union (int,real)  ir;
proc int p;
  ...
r := i/r;          -- widening (of i)
ir := i;           -- uniting
i := refi;         -- dereferencing
i := p;            -- deproceduring
row i := 5         -- rowing
```

Relational and Boolean Expressions

- **Relational operators compare two operands and return a boolean**

  = < > <= >= <>

- **Lower precedence than arithmetic operators**

  a + b < c + d ≡

  (a + b) < (c + d)

- **Boolean values: true, false**

- **Boolean operators: and, or, xor, not, =**

- **True boolean values are helpful.**

  - **In C, use integers:**

    0 = false

    other = true

  - **a > b > c is legal!**

Short-Circuit Evaluation

- **Get a result without evaluating entire expression.**

  - **x and y ° if x then y else false**

  - **x or y ° if x then true else y**

  - **(x and y are arbitrary boolean expressions)**

- **Same as regular evaluation in the absence of side-effects.**

- **May have a choice, e.g. Ada:**

  - **x or else y**

  - **x and then y**

  - **Otherwise, need to know if it's used.**