Arithmetic Expression

Operator Evaluation Order

(1) Precedence Rules

Ruby	Ada	C-based	PL/I	Apl
**	**, abs	postfix ++--	**,unary+-,¬	same
unary +-	*./,mod,rem	prefix++--,unary+-	*,/	
*,/,% rem	unary+-, not	*,/,%	+,-	
+,-	+,-	+,-	<,≤,=,..	

(2) Associativity

- usually left-to-right

Fortran, Ruby: ** right-associative
C: **, ++, --, unary+, unary -
Ada: ** non-associative

Conditional Expression

C: ? : (ternary)

```
if (count == 0)   avg = 0;
          else    avg = sum / count;
```

➔ avg = (count == 0) ? 0 : sum / count;

Operand Evaluation Order

- order matters because of side-effect

Ex.
```
a = 20;                    //  fun (a)
b = a + fun(a);            //      a = a + 10;
                           //      return a;
```

1. a → fun(a): 50
2. fun(a) → a: 60

Ex.
```
int a = 5;
int fun( ) {
    a = 17;
    return 3;
}
void main ( ){
    a = a + fun( );
}
```

a is either 8 or 20 depending on the order of evaluation.

Solution.

1. Disallow side-effect – restrict flexibility
   Note. Sometimes global variables are convenient.
2. Left-to-right order – restrict code optimization.

Note.

No side-effect on pure functional programming languages.

Operator Overloading

Criteria - readability or reliability

Examples.

+: integer addition, floating-point addition, matrices addition (O.K.)

& in C

1. bitwise AND: x = a &y // no error check on x = &y
2. address: x = &y

Division (/)

```
                        (7)     (2)
C++      avg_ =  sum  /  count;      // avg = 3.0  not  3.5
         float   int     int
```

Pascal

```
avg:= sum / count:          // avg = 3.5
avg:= sum div count;        // avg:= 3
```

PHP

```
7 / 2 → 3.5   // float
8 / 2 → 4     // integer
```

Type Conversion

Narrowing conversion
- int → byte
- may lose data

Widening conversion
- int → float
- relatively safe
- may lose accuracy

(1) Coercion: Implicit type conversion
- reliability vs. flexibility

Example.

Ada	Java
A: Integer; B, C, D: Float; : C := B * A	int a; float b, c, d; : c = b * a;
(X)	(O)

(2) Cast: Explicit type conversion

Float (sum)

o Errors in Expressions
- type error
- coercion error
- overflow, underflow
- division by zero

Relational Expression

- usually overloaded
- lower precedence than arithmetic operators

a + 4 > b * 2 → (a + 4) > (b * 2)

Operation	*Ada*	*C-based Languages*	*Fortran 95*
Equal	=	==	.EQ. or ==
Not equal	/=	!=	.NE. or <>
Greater than	>	>	.GT. or >
Less than	<	<	.LT. or <
Greater than or equal	>=	>=	.GE. or >=
Less than or equal	<=	<=	.LE. or >=

Boolean Expressions

- AND, OR, NOT (XOR)

Precedence

Ada AND = OR

Others AND > OR

C-based: arithmetic > relational > Boolean

Short-Circuit Evaluation

Ex. Searching

Java:

```
index = 0;
while ((index < listlen) && (list[index]  !=  key))
    index = index + 1
```

Without short-circuit, out-of-range exception may occur.

Ada:

and then
or else

```
Index := 1;
while (Index <= Listlen) and then (List(Index) /= Key)
  loop
    Index := Index + 1;
  end loop;
```

Assignment Statement

- workhorse in imperative languages

Syntax: <var> <assignment operator> <expression>

assignment operator: (=) or (:=)

C-based

- conditional targets

```
flag ? count1 : count2 = 0;
```

- compound assignment operator

```
sum += value;
```

- unary assignment operators

```
sum = ++ count;
count++;
```

- assignment as an expression

```
while ((ch = getchar( )) != EOF) {...}
```

Perl

- list assignment

```
($first, $second, $third) = (10, 20, 30);
($first, $second) = ($second, $first);
```