

Expressions

Fundamental means of expressing computations in a programming language

- Syntax (seen this before)
- Semantics

Expression semantics :

- orders of operator and operand evaluation
- type mismatches
- coercions
- short-circuit evaluation

Arithmetic Expressions

Characteristics inherited from from conventions that had eveolved in mathematics

Contain operator, operands, paranthesis and function calls

Operator :

- Types
 - unary
 - binary
 - ternary
- Evaluation order specified by
 - Precedence
 - Associativity
 - Paranthesis

Precedence : Order in which operators are evaluated; different levels for different operators.

- nearly the same for all common imperative languages
- APL has a single level of precedence

Associativity

Associativity :- Order in which operators of the same level are evaluated.

- Usually left : *, /, +, -, postfix ++, postfix -
- Right : prefix ++, prefix -, unary -, **
(FORTRAN)
- Non : ** (Ada)
- APL : order of evaluation determined by associativity; right to left

Paranthesis :- Alter precedence and associativity rules.

- Parenthesized part has precedence over its adjacent unparenthesized part.
- APL use paranthesis to give higher precedence; bad readability.

Operand Evaluation Order

Becomes important because of *side-effects*

Side Effect (functional):– Occurs when a function changes either one of its parameters or a global variable.

- Example :

```
int a;
main()
{
  a = 12;
  b = a + side_effect(7);
}
```

```
int side_effect(int x)
{
  a = 0;
  return (x+a);
}
```

- The value of b depends on the order of evaluation.

Side Effects

Ways to get around side effects :

- Specify the operands are evaluated left to right.
- Disallow functional side-effects
- Functions in expressions are valid only if functions do not change the value of operands in the expressions (FORTRAN 77)

Functions without side effects are called pure functions.

Conditional Expressions

In C (or C++),

```
value = (x == 12) ? 10 : z + y;
```

- ? is a ternary operator
- First operand : boolean test
- Second operand : then part
- Third operand : else part

Overloaded and Coersion

Overloading : When an operator can be used for more than one purpose

- Example : +, &
- Usually errors undetected by compilers.
- Compromises readability and reliability

Distinct operator symbols are helpful

Coersions : It is based on language rules that govern performing operations with mixed types.

- Narrow conversion :– convert to a type that cannot include all of the values of the original type.
- Wide conversion :– convert to a type that includes at least approx. of all values of the original type.
- Useful in languages that allow *mixed-mode expressions*.
- Can have explicit and implicit.

Relational and Boolean Expressions

Relational : An expression with a relational operator and 2 operands

- Have lower precedence than arithmetic operators

Boolean : Expressions contain booleans variables, boolean constants, relational expressions and boolean operators

- NOT, AND and OR
- Usually lower precedence than relational operators (in Pascal its the opposite)
- C has no boolean type.
- Implicit initialization.

Short-Circuit Evaluation: It is one in which the result is determined without evaluating all of the operands and/or operators.

- $(x > 7)$ and $(y < 10)$; if $(x > 7)$ is false why evaluate the whole expression ?
- What if second part has an important side effect ? : $(x > 7) \ \&\& \ ((y = a/12.5) < 30.0)$
- Sometime need to do it.
- Ada allows programmer to specify.

Assignment

Simple :

- = (FORTRAN, BASIC, C, PL/I)
- := (Ada, Pascal, Algol, Modula-2)
- $A = B = C$; It is really $A = (B \text{ equals } C)$ which is a boolean (PL/I)

Multiple Targets : Assigning expression value to more than one location.

- $\text{Sum, Total} = 0$ (PL/I)

Conditional Targets : This is found in C++.

- $\text{flag} ? \text{count1} : \text{count2} = 0$

Compound Assignment : Shorthand for specifying commonly used versions of assignment. Introduced in Algol 68, adopted by C, C++

- $C += D$
- $E *= F$

Unary Assignment

Assignment as Operands