Expression, Assignment

국민대학교 컴퓨터공학부
강 승 식

Topics

- Arithmetic Expressions
  - Overloaded Operators
  - Type Conversions
- Relational and Boolean Expressions
  - Short-Circuit Evaluation
- Assignment Statements
  - Mixed-Mode Assignment

Arithmetic expressions

*a[3]
우선순위는
a[3]이 우선
그다음 *
(*a)[3] 이렇게해야함

- Arithmetic expressions consist of
  - Operators: unary, binary, ternary
  - Operands
  - Parentheses
  - Function calls
- Design issues for arithmetic expressions
  - Operator precedence rules
  - Operator associativity rules
  - Order of operand evaluation
  - Operand evaluation side effects
  - Operator overloading
  - Type mixing in expressions

Operator Precedence, Associativity

- Typical associativity rules
  - Left to right, except **, which is right to left
  - Sometimes unary operators associate right to left (e.g., in FORTRAN)
- APL is different
  - All operators have equal precedence
  - All operators associate right to left
- Precedence and associativity rules can be overriden with parentheses

대부분의 언어는 관계가 없는경우는 좌에서 우임 그러나 APL(행렬,벡터) 모든 operator들이 우에서 좌임

Ruby Expressions 루비는 완전 객체지향 자바처럼

- All operators, are implemented as methods
  - Arithmetic, relational, and assignment operators, as well as array indexing, shifts, and bit-wise logic
  - Operators can all be overriden by application programs

Conditional Expressions

- C families

```
average = (count == 0)? 0 : sum / count
```

Operand Evaluation

- Variables
  - fetch the value from memory
- Constants
  - sometimes a fetch from memory; sometimes the constant is in the machine language instruction
- Parenthesized expressions
- Function call
  - *Functional side effect*
    - a function changes a two-way parameter or a non-local variable
    - `a = 10; b = a + fun(&a);`

요 a는 10일까 fun(&a) 일까??
이순서는 a임
만약 fun(&a) + a 이면 a는 10이 아닌 fun(&a)값이 나온다.
컴파일러 구현 입장에서 편한방법으로 함. (확실힌 모르겠음)

c언어는 evaluation 정해져있지 않아서 컴파일러맘
자바는 좌에서 우로해야된다고 딱 정해져 있음

Solutions to Functional Side Effects

1. Disallow functional side effects 이것을 없애려면?
   - No two-way parameters in functions
   - No non-local references in functions

이것을 없애려면?

2. Operand evaluation order be fixed
   - Java requires that operands appear to be evaluated in left-to-right order 자바는 왼쪽에서 오른쪽으로 정해짐 따라서 functional side effects 발생안함

Overloaded Operators

- Some are common (e.g., + for `int` and `float`)
- Some are potential trouble (e.g., `*` in C and C++)

- User-defined overloaded operators: C++, C#
  - Users can define nonsense operations
  - Readability may suffer, even when the operators make sense

Type Conversions

- *narrowing conversion* 허용 안하는경우가 많음 손실이 있기때문에 그러나 C는 예외
  - `float` to `int`
- *widening conversion* 손실이 없음
  - `int` to `float`

- Mixed-mode expression
  - *coercion* is an implicit type conversion
    - In most languages, all numeric types are coerced in expressions, using widening conversions
    - In Ada, there are virtually no coercions in expressions
- Explicit Type Conversions: type casting
  - `C: (int)angle`
  - `Ada: Float(Sum)`

Errors in Expressions

- Inherent limitations of arithmetic
  - e.g., division by zero
- Limitations of computer arithmetic
  - e.g. overflow

- Often ignored by the run-time system

Relational and Boolean Expressions

- Operator symbols for “not equal” are
  - `!=`, `/=`, `~=`, `.NE.`, `<>`, `#`
- `===` and `!==` in JavaScript and PHP
  - Do not coerce their operands
- Boolean Expressions

FORTRAN 77	FORTRAN 90	C	Ada
.AND.	and	&&	and
.OR.	or	||	or
.NOT.	not	!	not
	xor		

Relational and Boolean Expressions

- One odd characteristic of C's expressions:
  - **`a < b < c`** is a legal expression

a가 b보다 작으면 0이나 1로 먼저 변함 즉 c라는 값이 2 이상이면 무조건 참이다

- Short Circuit Evaluation
  - **`(a < b) && (b < c)`**
  - **`(a < b) || (b++ < c)`** 이렇게 코딩하는건 좋은 습관이 아니다 명확해야함
  - **`(13 * a) * (b++ / 13 - 1)`** → If a is zero

a가 만약 0이라면 곱하기 할 필요가 없음

secure coding- 안전한 코딩
strcpy(s1,s2) 못씀- 이거때문에 해킹이 남

Assignment Statements

- Assignment operator
  - = FORTRAN, BASIC, the C-based languages
  - := ALGOLs, Pascal, Ada
- = can be bad when it is overloaded for equality
  - That's why the C-based languages use == as the relational operator.
- Conditional Targets in Perl
  - `($flag ? $total : $subtotal) = 0`

- Compound Operators
  - a += b;

- Unary Assignment Operators

```
sum = ++count
sum = count++
-count++
```

사람을 헷갈리게 하므로 좋은 습관이 아님 // 고수인 척 하는 사람
괄호를 해줘야함

- Assignment can be used as operands
  - `while ((ch = getchar())!= EOF){…}`

- List assignments in Perl and Ruby

```
($first, $second, $third) = (20, 30, 40);
```

- Mixed-Mode Assignment

32->16비트로 가면 데이터 손실 될수 있으니 허용하면안됨

  - Fortran, C, and C++

자유분방 와이드닝이던 네로리딩이던 다허용

    - any numeric type value can be assigned to any numeric type variable
  - Java

와이드닝만 허용

    - only widening assignment coercions are done
  - Ada

허용 x

    - there is no assignment coercion

Summary

- Expressions
- Operator precedence and associativity
- Operator overloading
- Mixed-type expressions
- Various forms of assignment