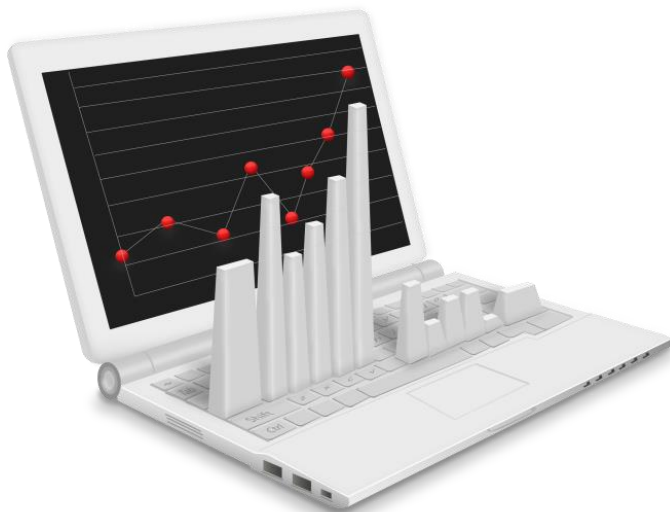


프로그래밍 언어론

Expression과 Assignment

컴퓨터공학과

조은선



Expression 와 Assignment

학 습 목 표

- 프로그래밍 언어의 구문요소 중, expression 과 assignment 에 대해 배운다.

학 습 내 용

- arithmetic expression (산술식, 수식)
- relational expression (관계식)
- logical expression (논리식)
- assignment (지정문)



목 차

- 들어가기
- 학습하기
 - Expression (표현식)
 - arithmetic expression (산술식)
 - relational expression (관계식)
 - logical expression (논리식)
 - Assignment (지정문)
- 평가하기
- 정리하기



알고가기



다음 중 좌측결합법칙도 우측 결합법칙도 성립하지 않는 식은?

(1) $x + y + z$

(2) $x - y - z$

(3) $x < y < z$

(4) $x = y = z$

확인



| Expression 와 Assignment

Expression (표현식)

→ 프로그래밍 언어에서 계산을 나타내는 기본적인 수단

arithmetic expression
(산술식)

relational expression
(관계식)

logical expression
(논리식)

→ **expression**의 설계의 쟁점

- 연산자 우선순위 규칙은 어떻게 할 것인가?
- 연산자 결합 규칙은 어떻게 할 것인가?
- 피연산자의 계산 순서는?
- 피연산자 계산의 side effect (부수효과)는 어떻게 할 것인가?
- 사용자의 의한 연산자 재정의 허용할 것인가?

expression에서 어떤 타입의 혼합을 허용할 것인가....



| Arithmetic Expression (산술식, 수식)

Arithmetic expression (산술식, 수식)

- ➔ 괄호, 연산자(operator), 피연산자(operand), 함수호출등으로 구성됨
- ➔ 최초의 프로그래밍언어 개발의 동기 중 하나임

연산자(operator)의 분류

- ➔ 피연산자(operand)의 개수에 따라

$2 + 53$ -826 $-x$

$a + b$

$(a == b) ? x : y$ $foo(a, b, c)$

$goo(a1, a2, ..., an)$

- ➔ 단항 (unary) 연산자
- ➔ 2항 (binary) 연산자
- ➔ 3항 (ternary) 연산자
- ➔ n항 (n-ary) 연산자

피연산자(Operand)의 계산

- ➔ 상수 일 때: 명령어 내에 존재하면 바로 사용하거나 아니면 메모리에서 가져옴
- ➔ 변수 일 때 : 값을 가져옴
- ➔ 함수호출 결과 일 때 : side effect 때문에 계산 순서가 매우 중요함
- ➔ 괄호식일 때 : 이 피연산자를 먼저 계산함



| 함수의 Side Effect

함수의 Side effect

- 함수가 인자나 전역변수 값을 변경하는 것
- 수식이 호출하는 함수가 그 수식의 다른 피연산자를 변경

예 `inf fun (int *x) { .. *x = 5; ... return 4;}`
`int a = 3,b; b= a + fun(&a);`

- 함수가 global 변수를 바꿀 때

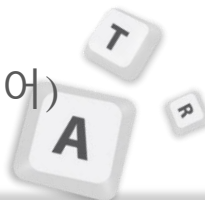
예 `int n;`
`...`
`foo (...) {...; n = 8; ...}`
`main () { n=4; ... = n+foo(...); ...}`

- 함수 결과가 아니더라도 부분 수식이 전체 수식의 다른 피연산자를 결정

예 `k= 5; b = k+ (k++) + (++k);`

문제 : 예측하지 못한 결과 발생 가능성

> 좀 불편한 것을 감수하더라도 side effect를 아예 금지시키기도 함 (함수형언어)



| Type 변환

변환 방향에 따라

➔ widening (대개 정보손실 없음)

`3 → 3.0`

`char → short → int → float → double`

➔ narrowing (정보 손실 가능)

`3.14 → 3`

`double → float → int → short → char`

피연산자의 타입 변환

➔ **명시적 (explicit) 변환** : type cast라 불리며, 프로그래머에 의해 수동적으로 변환됨

`float speed; ... (int) speed;`

➔ **암묵적 (implicit), 자동 변환** : 서로 다른 타입의 operand 들이 혼합된 수식에 편리하게 사용됨.

➔ 대부분의 언어는 widening을 자동 실행

`int a, b, c; float x;`

`a = b* x ; // 문제: c 대신 x가 잘못 들어간 경우 컴파일러에러 검출 불가!`



| 우선순위

우선순위

→ 다른 우선순위를 갖는 인접 연산자들을 계산할 수 있게 순서를 정의한다.

인접연산자 : 최대 하나의 피연산자에 의해 분리되는 연산자

$a + b * c$

→ 우선순위 규칙은 프로그래밍 언어마다 다를 수 있다.

전형적인 우선순위 레벨

1. 괄호
2. 단항 연산자
3. ** (지수: 언어가 지원한다면)
4. *, /
5. +, -

→ 괄호를 통해 우선순위를 정할 수 있다.

참고 : APL은 모든 연산자가 같은 우선순위를 가진다.



| 결합규칙

결합규칙

→ 같은 우선순위를 갖는 인접 연산자들은 순서대로 계산된다

$$a + b + c$$

→ 전형적인 결합 규칙들

→ 연산자의 결합규칙에 따라

> $a-b-c$ 는 $(a-b)-c$ 와 동일하다

→ 좌측결합

> $a=b=c$ 는 $a=(b=c)$ 와 동일하다

→ 우측결합

→ 보통 좌측결합이나 $**$, 단항 연산자등은 오른쪽에서 왼쪽

$$a ** b ** c \quad (\text{의미 } a^{b^c})$$

→ 특이한 예) APL은 모든 연산자들은 오른쪽에서 왼쪽으로 결합한다.

→ 괄호를 통해 우선순위를 정할 수 있다.

$$a * b + c$$

$$a * (b + c)$$



| 연산자 오버로딩(Operator Overloading)

- ➔ 동일한 연산자(operator)가 서로 다른 동작을 하도록 허용.
- ➔ 이미 제공되는 연산자의 기능을 바꿀 수도 있음.

예 C : 다른 의미로 사용되는 연산자 존재

$a \ \& \ b$ 는 bit 연산자이나 $\&b$ 는 주소 참조

$3+4$ 와 $3.2+4.5$ 는 다른 의미 (정수덧셈 \neq 실수 덧셈)

C++ : 사용자 정의 연산자 오버로딩(user-defined operator overloading) 가능

이항 : $+, -, *, /, \%, <, <=, ==, !=, >=, >, <<, >>, \&, |, ^$

단항 : $+, -, !, \sim, ++$

- ➔ 문제점 : 컴파일에러를 잘 못 찾고 가독성이 떨어질 수 있음.

예 $a \ \& \ b$ 를 잘못 입력해서 $\&b$ 로 하면 에러 검출 안됨

➔ 가능한 해결책

OCaml 예> $3+4, \ 3.0 \ +. \ 4.1$ (+와 +. 로 명시적 구분)



| 관계식 (Relational Expression)과 논리식 (Logical Expression)

Relational Expression (관계식)

- ➔ 관계 연산자와 다양한 타입의 피연산자를 사용
- ➔ 진리값을 결과로 준다 --- `true`, `false`
- ➔ 연산자 기호는 언어들에서 다양하게 사용된다

Not equal: `!=`, `/=`, `.NE.`, `<>`, `#`

Logical Expression (논리식)

- ➔ 피연산자와 결과가 진리값이다
- ➔ 연산자들:

Fortran 77	Fortran 90	C	Ada
<code>.AND.</code>	<code>and</code>	<code>&&</code>	<code>and</code>
<code>.OR.</code>	<code>or</code>	<code> </code>	<code>or</code>
<code>.NOT.</code>	<code>not</code>	<code>!</code>	<code>not</code>



| C의 관계식 (Relational Expression)과 논리식 (Logical Expression)

진리값을 표현하는 타입이 없다

→ int 타입 사용

→ false: 0

→ true : nonzero

문제점 : 원하지 않는 결과가 나올 수 있다.

$a < b < c$ 가 허용된다면..

$-8 < -5 < -3 = (-8 < -5) < -3$

$= \text{true} < -3$

$\text{/* nonzero} < -3 \text{ */}$

$= 1 < -3$

$= \text{false}$

단축 계산 (Short Circuit)

→ 논리 연산자의 피연산자 계산을 멈춤
(결과를 미리 알 때)

$(a \geq 0) \ \&\& \ (b < 10) \quad ; \text{ if } a = -2$

$(a \geq 0) \ || \ (b < 10) \quad ; \text{ if } a = +2$

→ 수식에서의 부수효과의 잠재적인 문제

$(a > b) \ || \ (b++ / 3)$

; 단축계산과 일반계산이 서로 다른 결과

→ 예

→ C, C++, Java: $\&\&, ||$

→ Ada: 프로그래머가 지정 가능: and [then], or [else]

→ Pascal: 단축계산 없음.

$\text{index} := 1;$

$\text{while } (\text{index} \leq \text{length}) \text{ and } (\text{L}[\text{index}] \neq \text{value}) \text{ do}$

$\text{index} := \text{index} + 1$

>> 배열 인덱싱 에러가 일어날 수 있음.

| 지정문 (Assignment)

➡ 명령형 언어에서 중심적인 구문요소

➡ 프로그램의 의미가 연속된 assignment의 실행 결과로 나타내어짐

'=' Fortran, Basic, PL/I, C, C++, Java

같다는 (equality check) 연산과 혼동 가능 예) **A = B = C** PL/I

":=" Algol, Pascal, Ada

➡ Assignment 와 관련한 여러 표현

➡ 다수의 저장 장소 (in PL/I) : **A, B = 10**

➡ 조건부 assignment (in C, C++, Java, C#) : **flag ? count1 : count2 = 0;**

➡ 복합 assignment (in Algol 68, C, ... : **sum += value;**

➡ 단항 assignment 들 (assign 될 값이 없음) : **count++; ++count;**

➡ '=' 은 이항연산자 로 쓰임 (C등 Algol 60 계열)

a = b = c = 3; // assignment 자체도 결과값을 내어줌

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

➡ side effect 로 실행 순서에 따라 다른 결과가 발생 가능

b = 3; a = b * (c = d / (b += 1));



평가하기

마지막으로 내가 얼마나 이해했는지를 한번 확인해 볼까요?
총 3문제가 있습니다.

START



평가하기 1

1. 다음은 assignment (지정문)에 대한 설명이다. 옳은 것은?

- ① assignment를 수식의 일부로 사용하는 것은 side effect를 가진 expression 으로서 프로그램 성능을 높인다.
- ② assignment 은 명령형 언어에서는 없어도 되는 구문요소이다.
- ③ 복합 assignment 을 자주 사용해야 오류의 가능성을 줄일 수 있다.
- ④ assignment를 if문의 조건식에 사용하는 언어는 없다.

확인



평가하기 2

2. 다음은 side effects가 발생하는 상황에 대한 설명이다.
가장 관련이 없는 것은?

- ① 어떤 수식에서 호출된 함수가 그 수식의 다른 피연산자의 값을 바꾸는 경우
- ② 함수가 global변수의 값을 바꿀 때
- ③ 어떤 수식에서 부분식이 그 수식의 다른 피연산자의 값을 바꿀 때
- ④ 함수 안의 local 변수 값을 바꿀 때

확인



평가하기 3

3. 연산자 오버로딩 (operator overloading) 에 대한 설명 중 가장 먼 것은?

- ① Operator overloading 은 프로그래밍 언어에 애초에 없던 operator(연산자)를 새로이 정의하는 것이다.
- ② C에서 `a & b` 의 `&`는 bit 연산자이나 `&b`에서는 주소참조로 쓰인 것도 한 예이다.
- ③ Operator overloading 을 사용함으로써 컴파일러가 에러를 찾지 못하는 상황이 발생할 수 있다.
- ④ 다수의 언어에서 정수, 실수에 대한 사칙연산자에 operator overloading 을 사용하여 제공하고 있다.

확인



정리하기

- ➡ arithmetic expression의 evaluation 은 우선순위, 결합성, 피연산자 계산, side effect 등을 고려하여 설계한다.
- ➡ 타입변환에는 widening과 narrowing이 있는데, 이중 widening은 정보 유실이 없기 때문에, 많은 언어에서 프로그래머의 명시 없이 자동으로 (implicit) 수행되도록 지원하고 있다.
- ➡ 관계식은 다양한 타입의 피연산자에 대하여 관계 연산 결과를 진리값으로 주고, 논리식은 진리값들의 논리 연산 결과를 진리값으로 준다.
- ➡ assignment 은 명령형 언어의 핵심적인 구문요소이다.
- ➡ assignment 을 수식의 일부로 사용하거나 복합 assignment과 단항 assignment의 남용은 오류 가능성을 높인다.