07장 자료 유형

【7.3.3】 with문

파스칼의 with문은 이름을 붙인 레코드의 항목이 보통의 변수처럼 보이는 중첩된 유효 범위를 도입한다. 그 레코드를 펼친다고 말한다. ©(심화학습에 있는) 그림 3.20에서 봤듯이 레코드형을 나타내는 항목을 심볼 테이블 유효 범위 스택 위로 푸쉬해 with문을 컴파일러 내에서 구현할 수 있다.

with문은 완전히 제한된 이름의 일부를 어떠한 모호성도 발생하지 않으면 생략할 수 있게 허용하는 언어 기능인 코볼과 PL/I의 생략 참조의 형식화다. 예 7.42의 코볼에서 name of elements(1) of chemical_composition of ruby를 name of elements(1) of ruby로 생략할 수 있을 것이다. ruby는 chemical_composition 항목 이외의 어떤 것 안에서도 elements라는 이름의 항목을 가지고 있지 않기 때문이다. 하지만 현재 유효 범위에서 ruby와 동일한 유형의 다른 레코드가 있다면 참조의 나머지가 필요하다. 그리고 그 유형 내의 element 배열이 한 개 이상의 원소를 포함한다.

예 7.122
코볼과 PL/I에서의 생략 참조

생략 참조는 항목 이름의 유일함에 무조건적으로 의존하기 때문에 코드를 해석하기 어렵게 할 수 있다.

어떤 with문은 생략 정보를 좀 더 명시적으로 지정해 오독을 감소시키게 한다. 예 7.42를 다시 보면 다음과 같다.

예 7.123
파스칼의 with문(반복)

```
with ruby.chemical_composition.elements[1] do begin
name := 'Al';
atomic_number := 13;
atomic_weight := 26.98154;
```

```
metallic := true
end;
```

하지만 파스칼의 with문은 여전히 다음과 같은 문제로부터 고통을 받는다.

1. 동일한 유형의 두 레코드의 필드를 동시에 조작하는(예를 들어 한 레코드의 항목 중 일부를 다른 것의 일치하는 항목에 복사하는 것) 쉬운 방법은 없다. 레코드 중 하나를 열기 위해서 with문을 사용할 수 있지만 나머지 레코드에 대해서는 그렇지 않다.
2. 펼친 레코드의 필드 중 어느 것이나 지역 객체와 동일한 이름을 가지고 있으면 이름 충돌이 발생한다. with문이 중첩된 유효 범위이기 때문에 지역 객체는 일시적으로 접근할 수 없게 된다.
3. 긴 with문에서나 다른 유형을 가진 레코드를 펼친 중첩된 with문에서 필드명과 그것이 속한 레코드 간의 대응은 불확실하게 된다.

예 7.124
모듈라 3의 with문

모듈라 3은 좀 더 일반적인 형태로 with문을 재정의해 이 문제를 다룬다. 레코드를 펼치는 대신에 모듈라 3의 with문은 복잡한 수식에 대해 한 개 이상의 별칭을 도입한다. 모듈라 3 양식의 예를 개정해보면 다음과 같이 작성할 수 있다.

```
WITH e = ruby.chemical_composition.elements[1] DO
e.name := "Al";
e.atomic_number := 13;
e.atomic_weight := 26.98154;
e.metallic := true;
END;
```

여기서 e는 ruby.chemical_composition.element[1]에 대한 별칭이다. 레코드의 필드는 직접적으로 보이지 않는다. 하지만 필드명 앞에 "e."를 붙임으로써 쉽게 접근할 수 있다.

예 7.125
다중 객체 with문

2개 이상의 레코드에 접근하기 위해 다음과 같이 작성할 수 있다.

```
WITH e = whatever, f = whatever DO
e.field1 := f.field1;
e.field3 := f.field3;
e.field7 := f.field7;
END;
```

예 7.126
레코드가 아닌 with문

레코드 이외의 객체에 대해서도 별칭을 만들기 위해 모듈라 3의 with문을 사용할 수 있다. 그것을 두 번 작성하지 않고 복잡한 수식을 검사하고 나서 사용하기 위한 아래 예가 있다.

```
WITH d = complicated expression DO
```

```
IF d <> 0 THEN val := n/d ELSE val := 0 END;
END;
```

설계와 구현

with문

펼친 레코드나 별칭을 붙인 레코드에 대한 숨겨진 포인터를 생성함으로써 파스칼과 모듈라 3 모두의 with문을 목적 프로그램에서 구현한다. with문 내에 있는 레코드의 모든 사용은 숨겨진 포인터로부터의 오프셋을 통해 필드를 효율적으로 접근한다. 보통 with문 없이도 동일한 효율성을 얻을 수 있다. 하지만 컴파일러가 ©(심화학습에 있는) 15.4절로 설명을 미룬 코드 향상의 특별한 형태 중 하나인 전역 공통 하위 수식 분석을 구현할 때만 그렇다.

예 7.127

스킴에서 with를 모방하기

물론 특별한 구성소가 없는 대다수의 언어에서도 비슷한 효과를 얻을 수 있다. 예를 들면 대부분의 함수형 언어는 중첩된 유효 범위를 도입하는 let문을 가지고 있다. 스킴에서 다음의 코드는 ©(심화학습에 있는) 예 7.126의 코드와 거의 동일한 효과를 가진다.

```
(let ((d complicated expression))
(if (not (= d 0)) (/ n d) 0))
```

©(심화학습에 있는) 예 7.125는 비함수형 언어 특징의 사용을 필요로 할 것이다.

예 7.128

C에서 with를 모방하기

C에서 아래와 같이 쓸 수도 있다.

```
{
    my_struct *e = &whatever
    my_struct *f = &whatever
    e->field1 = f->field1;
    e->field3 = f->field3;
    e->field7 = f->field7;
}

{
    double d = complicated_expression
    val = (d ? n/d : 0);
}
```

이 코드는 중첩된 블록 내의 변수를 선언하고 힙에 있지 않는 객체로의 포인터를 생성하는 C 프로그래머의 능력에 의존한다. 파스칼은 중첩된 선언을 허용하지 않는다. 파스칼과 모듈라 3은 모두 힙에 있지 않는 포인터를 허용하지 않는다. C의 예에서 모듈라 3의 with문의 훨씬 더 가까운 근사를 생산하기 위해 포인터 대신 8.3.1절에서 소개할 것인 C++의 참조형을 사용할 수 있다.

✓ 확인문제

64. with문은 무엇인가? 어떤 목적을 위해 사용하나?

65. 생략 참조란 무엇인가?

66. 파스칼에서 깨달은 것처럼 with문의 한계는 무엇인가? 모듈라 3에서 이 한계를 어떻게 극복하는가? 스킴과 같은 언어에서 이를 어떻게 피하는가?

67. with문은 순수하게 표기적인 편리함인가? 그렇지 않으면 실용적인 의미도 가지고 있는가?