06장 제어 흐름

【6.5.4】 아이콘의 생성기

아이콘에서는 클루와 마찬가지로 열거 제어형 반복을 위해 생성기를 사용할 수 있다. 아이콘에서 기본적인 for 루프 예제를 다음과 같이 작성할 수 있다.

예 6.95
아이콘의 간단한 생성기

```
every i := first to last by step do {
    ...
}
```

여기서 ...to...by...는 고유 "혼합위" 생성기다.

문자열 조작을 위해 아이콘을 만든 것이기 때문에 고유 생성기의 대부분은 문자열을 연산한다. 예를 들면 find(substr, str)는 하위 문자열 substr의 발생을 발견할 수 있는 문자열 str 내의 위치(인덱스)를 생성한다. upto(chars, str)는 chars 내 어떤 문자 하나라도 나타나는 문자열 str 내의 위치를 발생시킨다(find에 첫 인자는 큰따옴표로 구분되는 문자다. upto에 첫 인자는 작은 따옴표가 구분하는 cset[문자 집합]이다). 전위 연산자 !는 피연산자의 모든 원소를 생성한다. 이 피연산자는 문자열, 리스트, 레코드, 파일이나 테이블일 수 있다.

하지만 클루와 비교하면 아이콘의 생성기는 언어의 의미에 좀 더 깊게 내장돼있다. 수식을 예상하는 문맥 어디서나 생성자를 사용할 수 있다. 그러면 더 큰 문맥은 여러 개의 결과를 생성시킬 수 있다. 아래 코드는 s 안에서 공백 다음의 모든 위치를 출력한다.

예 6.96
수식 내의 생성기

```
every i := 1 + upto(' ', s) do {
    write(i)
}
```

심지어 다음과 같이 작성할 수 있다.

```
every write(1 + upto(' ', s))
```

아이콘에서 반복을 위해서뿐만 아니라 역추적(backtracking)으로 구현된 목표 유도(goal-directed) 탐색을 위해서도 생성기를 사용한다(역추적은 프롤로그에서 중요하다. 이에 대해서는 11장에서 공부할 것이다). 대부분의 언어는 선택과 논리 제어형 루프를 제어하기 위해 논리형 수식을 사용하지만 아이콘은 성공과 실패라는 좀 더 일반적인 개념을 사용한다. 조건 2<3이 참이기 때문이 아니라 비교 2<3이 성공하기 때문에 다음과 같은 문장을 실행한다고 말한다.

예 6.97

성공을 찾기 위한 생성

```
if 2 < 3 then {
    ...
}
```

이런 차이는 그들 중 하나가 주변 문맥을 성공하게 할 때까지 (혹은 더 이상의 결과가 만들어지지 않을 때까지) 결과를 반복적으로 만들어낼 수 있는 생성기에서 중요하다. 예를 들면 문자열 "abc"가 s의 6번째가 넘어서 나타나야 다음 문장 내의 if문의 몸체를 실행할 것이다.

```
if (i := find("abc", s)) > 6 then {
    ...
}
```

find가 그 결과를 순서대로 생성하기 때문에 (있다면) i는 그런 첫 위치를 나타낸다. 실행 모델은 다음과 같다. find는 s 내에서 "abc"가 나타나는 모든 위치를 찾을 수 있다. 그런 첫 발생 위치가 2라고 가정하자. 그러고 나서 i에 값 2를 대입한다. 하지만 비교 2>6은 실패한다. 실패한 수식 안에 생성자가 있기 때문에 아이콘은 그 생성자를 재개하고 다음에 발생하는 값에 대한 수식을 재계산할 것이다. 비교가 성공할 때까지나 그 생성기가 값을 모두 발생해버릴 때까지 그것은 이 재계산을 계속할 것이다. 이 경우 그것(그 생성기)은 실패하고 그 전체 계산도 완전히 실패한다. 그리고 if의 몸체를 뛰어넘는다.

실패한 수식이 2개 이상의 생성기를 포함한다면 가능한 모든 값을 체계적으로 탐색할 것이다. 예를 들면 어떤 x가 s와 t 모두의 동일한 위치에 나타나는 경우에만 아래 if의 몸체를 실행하고 i는 일치하는 첫 번째 위치를 나타낼 것이다.

예 6.98

여러 개의 생성기를 이용한 역추적

```
if (i := find("x", s)) = find("x", t) then {
    ...
}
```

일치하는 위치가 없다면 i는 s에서 마지막 x의 위치가 되지만 루프의 몸체를 건너뛸 것이다. 모든 검사가 실패하는 경우에 있어 i를 변경하는 것을 프로그래머가 피하기

바란다면 가역 대입 연산자 `<-`를 `:=` 대신 사용할 수 있다. 아이콘이 여태까지의 가역 대입을 역추적한다면 그것은 원래 값을 복구한다.

아이콘에서 `return expr` 대신 `suspend expr` 문장을 사용한다면 모든 사용자 정의 서브루틴은 생성기가 될 수 있다. 아이콘의 `suspend`는 `yield`와 동일하다. `suspend` 뒤에 오는 수식이 생성기의 호출을 포함한다면 서브루틴은 생성된 각 값에 대해 한 번씩 되풀이되어 중지될 것이다.

✓ 확인문제

48. 아이콘의 생성자가 클루의 반복자나 유클리드, C++, 자바, C#의 반복자 객체와 어떻게 다른지를 설명하라.
49. 아이콘의 성공과 실패의 개념을 논하라.
50. 역추적이란 무엇인가? 그것이 왜 유용한가?
51. 아이콘을 제외하고 역추적이 중요한 역할을 하는 언어의 이름을 대라.