중첩된 반복문에서 흐름 제어를 위한 개선된 문법구조

An Improved Syntax for the Control Flow of Nested Loops

최 문 호, 서 성 채, 나 인 섭, 이 성 호*
전남대학교, 제노테크(주)*

Mun-Ho Choi, Seong-Chae Seo, In-Seop Na, Seong-Ho Lee*
Chonnam National Univ., GenoTech Co. Ltd.*

요약

본 논문에서는 중첩된 반복문에서 레이블을 사용하지 않고 프로그램의 흐름을 제어하는 개선된 문법구조를 제안한다. 또한 명시적으로 반복문의 종료 상태를 확인하는 방법을 제안한다. 제안된 방법을 사용하면 흐름 파악이 쉬운 프로그램을 작성할 수 있다.

Ⅰ. 서론

프로그램의 흐름을 바꾸는 기본적인 명령인 goto 문은 프로그램의 특정 부분에서 행번호나 레이블(label)이 있는 다른 부분으로 건너뛸 때 사용하는 명령이다. goto 문은 베이직, C, C++, 파스칼, 펄 등 많은 고급언어에서 사용되고 있다. 그러나 goto 문이 과도하게 사용되면 이해하기 어렵고 유지하기 힘든 스파게티 코드[1]가 나오기 쉽기 때문에 많은 비판의 대상이 되어왔다.

이러한 goto 문의 단점을 해소하기 위해 C언어를 비롯한 구조적 프로그래밍 언어에서는 반복문 내의 흐름 제어를 위해 break문과 continue문을 도입하였다. goto 문의 사용이 나쁜 습관이긴 하지만, 많은 프로그래밍 언어에서는 goto 문을 사용하지 않고는 중첩된 반복문을 벗어나야 하거나 예외가 발생하여 프로그램의 특정 부분으로 제어흐름을 변경하는 경우가 간단히 처리되지 않는다. 후자의 경우는 많은 고급언어에서 예외처리(exception handling)[2] 방법을 도입함으로써 해결하였으나 전자의 경우는 프로그래머에게 그 해결을 부담지우거나 goto 문과 유사하게 점프와 레이블을 사용하는 해결책[3]이 제시되기도 하였다. 그러나 이러한 해결책은 goto 문의 문제점을 그대로 답습하고 있다. 또한 break와 continue의 도입으로 goto 문과 레이블의 사용은 줄었지만 반복문이 반복조건을 완료하여 일반적인 종료를 한 경우와 break로 반복문을 벗어난 경우를 검사하여 확인하는 과정은 여전히 남아있다.

본 논문에서는 중첩된 반복문에서 레이블을 사용하지 않고 자연스럽게 프로그램의 흐름을 제어하는 개선된 문법 구조를 제안한다. 아울러, 반복문의 종료 조건을 확인하여 반복후 처리를 효과적으로 할 수 있는 방안을 제시한다. 본 연구에서 제시한 방법을 사용하면 흐름 파악이 쉬운 프로그램을 작성할 수 있다.

Ⅱ. 기존 중첩 구문과 흐름 제어의 문제점

C, C++, Java, C# 등을 비롯한 구조적 프로그래밍 언어는 반복문 내에서의 효과적인 흐름 제어를 위해 break 문과 continue 문을 제공한다. 이러한 문법구조는 중첩되지 않은 반복문이나 switch 문에서는 무난하게 작동한다. 그러나 그림 1과 같이 중첩된 구문에서는 중첩된 반복문을 한꺼번에 벗어날 수 없어서 7번 줄 및 16~18번 줄과 같이 부가적인 변수와 조건 검사 및 break 사용이 필요하다.

```
1   int[][] a = {
2       { 1, 3, 5, 7, 9},
3       { 2, 4, 6, 8, 10},
4       { 2, 3, 5, 7, 11}
5   };
6   int searchFor = 8;
7   Boolean found = false;
8
9   for (int i = 0; i < a.length; i++) {
10      for (int j = 0; j < a[i].length; j++) {
11          if (a[i][j] == searchFor) {
12              found = true;
13              break;
14          }
15      }
16      if (found) {
17          break;
18      }
19  }
20  if (found) {
21      System.out.println(searchFor + " is in the list");
22  }
23  else {
24      System.out.println(searchFor + " is not found");
25  }
```

▶▶ 그림 1. 중첩된 반복문의 예

이와 같은 문제점을 해결하기 위해 자바에서는 그림 2와 같이 반복문의 시작 지점에 레이블을 지정할 수 있게 하고, 'break 레이블'과 같은 문법을 사용하여 내포된 구

문에서 바깥 반복문까지 한꺼번에 벗어날 수 있는 방법을 도입하였다. 그러나 이 방법은 break문이 어떤 반복문을 벗어나는 지 확인하기 위해 레이블이 찾아봐야 한다는 goto 문의 고질적인 문제를 답습하고 있다. 또한 반복문이 일반적 종료인지, break에 의한 종료인지를 확인하기 위한 변수의 사용을 필요로 한다.

```
1   int[][] a = { /* 그림 1의 예제와 같음 */ };
2   int searchFor = 8;
3   Boolean found = false;
4
5   search:
6   for (int i = 0; i < a.length; i++) {
7       for (int j = 0; j < a[i].length; j++) {
8           if (a[i][j] == searchFor) {
9               found = true;
10              break search;
11          }
12      }
13   }
14  if (found) {
15      System.out.println(searchFor + " is in the list");
16  }
17  else {
18      System.out.println(searchFor + " is not found");
19  }
```

▶▶ 그림 2. 레이블 지정 break를 사용한 중첩된 반복문

Ⅲ. 개선된 문법구조

앞에서 언급된 중첩 구문의 흐름 제어의 문제점을 개선하기 위하여 본 논문에서는 그림 3과 같은 문법구조를 제안한다.

```
반복문1 (조건1) {
    반복문2 (조건2) {
        반복문3 (조건3) {
            벗어나고 싶은 반복 구분 개수만큼 나열
            break ... break;
        }
    }
}

반복문1 (조건1) {
    반복문2 (조건2) {
        반복문3 (조건3) {
            벗어나고 싶은 반복 구분 개수만큼 나열
            break ... continue;
        }
    }
}
```

▶▶ 그림 3. 개선된 제어 흐름 문법구조

예를 들어 이중 반복문을 한꺼번에 벗어나고 싶은 경우 “break break”를 사용하고, 이중 반복문에서 내포된 반복문을 벗어난 후 바깥 반복문의 다음 반복을 지속하고자 하는 경우 ”break continue“를 사용하면 된다. 이와 같은 구조는 반복문 안에 switch 문이 내포된 경우에도 동일하게 적용된다.

위에서 제시한 구문을 사용하면 break 문이 더 이상 레이블을 필요로 하지 않기 때문에 return 문과 같이 break 문도 수식을 갖는 문법구조를 만들 수 있다. 이의 문법 구조를 이용하여 그림 4와 같이 반복문의 일반적 종료 여부를 검사할 수 있다.

```
반복문(조건) {
    break default = value1;
    .
    .
    .
    break value2;
}
if ( break_수식 ) {
    ...
}
```

▶▶ 그림 4. 반복문의 종료 상태 검사를 위한 문법구조

위의 그림에서 break_수식은 break를 일반 변수처럼 사용한 수식(그림 5 참조)을 말한다. “break default”에 대한 값의 대입은 반복문이 일반적인 종료일 경우 “break”가 가지는 값이다. break가 가지는 값의 자료형은 타입 추론에 의해 결정 가능하다. 자바와 C# 같은 언어는 타입 추론을 지원한다. 그림 5는 개선된 문법구조를 사용한 예제이다.

```
1   int[][] a = { /* 그림 1의 예제와 같음 */ };
2   int searchFor = 8;
3
4   for (int i = 0; i < a.length; i++) {
5       break default = false;
6       for (int j = 0; j < a[i].length; j++) {
7           if (a[i][j] == searchFor) {
8               break break true;
9           }
10      }
11   }
12  if (break == true) {
13      System.out.println(searchFor + " is in the list");
14  }
15  else {
16      System.out.println(searchFor + " is not found");
17  }
```

▶▶ 그림 5. 개선된 문법구조를 사용한 중첩된 반복문

위의 그림의 예제는 그림 1과 2의 예제에 비해 불가피하게 도입되었던 부가적인 변수와 레이블 및 break 사용을 피했다.

■ 감 사 의 글 ■

본 연구는 미래창조과학부와 한국산업기술진흥원의 “실감미디어산업 R&D 기반구축및성과확산사업”의 지원을 받아 수행된 연구결과임.

■ 참 고 문 헌 ■

[1] Dijkstra, “Go To Statement Considered Harmful,” Communications of the ACM 11:3 (1968), 147-148.
[2] Wikipedia, “Exception Handling,” http://en.wikipedia.org/ wiki/Exception_handling, 2014.
[3] James Gosling, Bill Joy, et. al., Blocks and Statements, The Java Language Specification, Java SE 7 Edition, Oracle, 2013.