0313_2255_go to문은 해롭다

(대략적인 흐름만 번역한 것입니다.)

번역에 참여한 분

nassol

Go To 구문은 해롭다.

Edsger W. Dijkstra
http://www.u.arizona.edu/~rubinson/copyright_violations/Go_To_Considered_Harmful.html

프로그래머가 go to 구문을 많이 쓸수록 실력이 낮다고 수년간 생각은 해왔다.

그러다가 go to 구문을 쓰면 왜 끔찍한 결과로 이어지는지 그 이유를 알아냈다. 그래서 "higher level" 프로그래밍 언어에서는 go to 구문을 쓰지 말아야 한다고 확신하게 되었다. (기계어 등의 'low level'언어를 제외하고..)

그런데 그 때는 내가 발견한 이유를 그닥 중요히 여기지 않았다.

아주 최근에 논의를 하다가 이 주제에 대해 이야기하기 되었는데, 그 논의를 계기로 나는 이것에 대한 생각을 밝히
첫번째, 프로그래머의 역할은 프로그램을 돌아가게 다 짜는 것으로 끝난다고 여길지도 모르지만, 실제로 중요한 것은 프로그램의 저변에서 돌아가는 프로세스이다. 왜냐하면 이 프로세스야말로, 원래 의도했던 목적을 실현시켜야 하기 때문이다. 그리고 이 프로세스야말로, 원래 기능명세에다 작성했던 것을 역동적인 방식으로 실현해내야 하기 때문이다.

그러나, 일단 프로그램이 일단 만들어지고 나면, 이 프로세스를 만드는 역할은 컴퓨터에게 위임된다.

두번째, 인간의 지적 능력은 정적인 관계를 잘 이해하는 데 적합하게 설계되어 있으며, 시간의 진행에 따른 프로세스를 시각화하는 능력은 비교적 덜 발달하였다.

그러한 이유로, (자신의 한계에 대해 인지하고 있는 현명한 프로그래머인) 우리들은 정적인 프로그램과 동적인 프로세스간의 괴리를 최대한 줄여야만 한다. 그래서 공간에서 구현된 프로그램과 시간상에서 진행되는 프로세스가 일치하도록 해야 한다. (요부분 쪼매... 확실치 않네요.. 이해를 못한듯 ;;)

자, 이제는 프로세스가 진전되는 정도를 어떻게 나타낼지에 대해 생각해보자. 아주 구체적으로 생각해보자.

어떤 행위들이 시간상 순차적으로 이루어지는 프로세스가 하나 있다. 어떤 행위가

일어난 후에, 이 프로세스가 멈추었다고 하자.

바로 이 지점까지 프로세스를 다시 진행시키려면 어떤 데이터를 바꿔야 할까?

논의를 간단하게 하기 위해서, 프로그램 코드가, 개별적인 작업을 수행하는 구문이 나열된 방식으로 짜여졌다고 하자.

이 경우에, 두 개의 연속하는 행위를 기술한 것 사이의 지점을 가리키면 된다.

(마지막 세 단어, 두 개의 연속하는 행위(two successive action descriptions)는 두 가지로 해석이 가능하다.

"연속하는 (행위를 기술한 것)"으로 해석하는 경우에,
이는 텍스트 상에서 순차적으로 있다는 의미가 되고,

"(연속하는 행위)를 기술한 것"으로 해석하는 경우에는,
이 행위가 시간 상 순차적으로 이루어진다는 의미가 된다.

코드 내에서 어느 위치에 해당하는지를 가리키는 것을 "textual index"라고 부르자.

코드에다가 다음과 같은 것들을 삽입하면
(if B then A),
(if B then A1 else A2)
case [i] of (A1, A2, ..., An)
switch..

하나의 textual index로 프로세스 진행정도를 나타낼 수 있다.

하지만 하나의 textual index만으로는 충분치 않다는 점을 인정해야 한다.

textual index가 프로시져(procedure body)의 내부를 가리키는 경우에,
동적인 진전도(progress)를 나타내려면, 지금 가리키는 프로시져가 몇 번째 실행된 것인지도 알아야 한다.

프로시져를 포함시켰을 때에는,
textual index의 sequence를 사용해서 프로세스의 진행정도를 나타낼 수 있다. 이 sequence의 길이는 프로시져를 호출한 dynamic depth와 일치한다. (요 뒷부분은 이해를 못하겠군요..)

이번에는 반복문에 대해 살펴보자. 예를 들면 while B repeat A나 repeat A until B 같은 것 말이다.

재귀 프로시져를 사용하면 반복을 사용할 수 있기 때문에, 요즘은 이런 반복문을 너무나 많이 사용하고 있다. 현실적인 이유때문에, 이런 반복문을 아예 쓰지 말자고 주장하지는 않겠다:

한편으로는 finite equipment를 사용해서 편하게 반복문(repetition clauses)을 구현할 수 있다.

다른 한편으로는 "연역적 사고" 방식을 하면, 반복문으로 인해 생겨나는 프로세스를 우리는 잘 이해할 수 있다.

반복문이 포함되면, textual indices만으로는 더 이상 프로세스의 동적인 진행정도를 나타낼 수 없다.

하지만 반복문이 한 번 실행될 때마다, "동적인 지수(dynamic index)"를 사용할 수 있다. 반복되는 수를 센다.

반복문도 겹쳐서 사용할 수 있으므로,
textual indices와 동적인 indices를 혼합하면
프로세스의 진행정도를 고유하게 나타낼 수 있다.

중요한 것은, 프로세스의 진행도를 가리키는 지수의 값이
프로그래머의 권한 밖에 있다는 것이다.

이 지수들은 프로그래머가 원하건 원하지 않건 값이 만들어진다.

이 지수들은, 프로세스의 진행정도를 나타내는 독립적인 좌표를 알려준다.

그런데 이런 독립적인 좌표가 왜 필요할까?

그 이유는, 프로세스의 진행정도와 연관지어야먄,
어떤 변수의 값을 해석할 수 있기 때문이다.

방에 있는 사람의 수 n를 세려고 할 때, 우리는 사람이 방에 들어올 때 n을 증가시키는 방법을 쓸 수 있다. 누군가 들어오는 것을 봤는데 아직 n을 증가시키지 않았다면, n의 값은 현재 방 안에 있는 사람의 수 보다 1만큼 작을 것이다.

go to 구문을 썼을 때, 어떤 해악성이 있을까?
go to 구문을 쓰면, 프로세스의 진행정도를 나타내는 좌표모음을 구하는 것이 엄청나게 어려워진다.

보통 사람들은 어떤 변수의 값을 고려할지를 신중하게 판단한다.
그러나 이것은 무의미하다.
왜냐하면, 프로세스의 진행정도를 고려하여 값의 의미를 이해해야 하기 때문이다.

물론 go to 구문을 사용하더라도, 프로세스의 진행정도를 나타낼 수는 있다. 예를 들면, 프로그램이 시작된 이후로, 어떤 행위가 수행될 때마다 1씩 증가시키면서 세는 방법으로 말이다.

하지만 이런 식으로 만들어진 좌표는 고유할지는 몰라도, 도움이 안된다.

이런 방식으로 좌표를 만들면, n이 방안에 있는 사람수보다 1작은 프로세스의 지점을 정의하는 것이 무지 어려워진다.

Go to 구문을 쓰는 것은 너무나 원시적이다; go to 구문을 사용하면, 프로그램을 엉망으로 어질러놓기가 너무나 쉬워진다.

clauses가 완벽하다고 주장하려는 것은 아니다.
어떤 clauses를 쓰던지 간에, 프로그래머의 영향을 받지 않는 좌표 체계, 프로세스의 진행정도를 나타내는 좌표 체계를 유지할 수 있도록 해야 한다.

요기까지 번역했어요**~ 0313**

It is hard to end this with a fair acknowledgment.

Am I to judge by whom my thinking has been influenced?

It is fairly obvious that I am not uninfluenced by Peter Landin and Christopher Strachey.

Finally I should like to record (as I remember it quite distinctly)
how Heinz Zemanek at the pre-ALGOL meeting
in early 1959 in Copenhagen
quite explicitly expressed his doubts
whether the go to statement

should be treated
on equal syntactic footing
with the assignment statement.

To a modest extent I blame myself for not having then drawn the consequences of his remark.

The remark about the undesirability of the go to statement is far from new.

I remember having read the explicit recommendation to restrict the use of the go to statement to alarm exits, but I have not been able to trace it;

presumably, it has been made by C. A. R. Hoare. In [1, Sec. 3.2.1.]

Wirth and Hoare together make a remark in the same direction in motivating the case construction:

"Like the conditional,
it mirrors the dynamic structure of a program more clearly
than go to statements and switches,

and it eliminates
the need for introducing a large number of labels
in the program."

In [2] Guiseppe Jacopini seems to have proved the (logical) superfluousness of the go to statement.

The exercise
to translate an arbitrary flow diagram
more or less mechanically
into a jump-less one, however,
is not to be recommended.

Then the resulting flow diagram cannot be expected to be more transparent than the original one.