1 Introducción

En un programa escrito en lenguaje máquina, el control de flujo se lleva a cabo, esencialmente, a través de una instrucción `goto`. Según el relato de Knuth [1974, p. 264], el Dr. Eiichi Gotō (後藤英一) soportaba con humor la mala prensa que estaba cobrando su apellido en su campo profesional: «he cheerfully complained that he was always being eliminated».

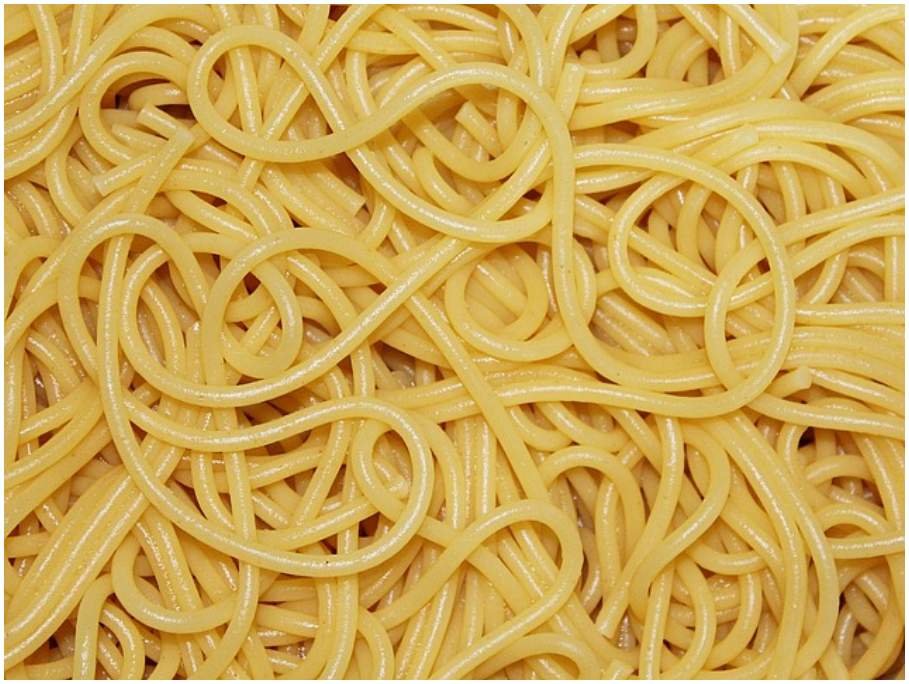

Ilustración 1: Un plato de espaguetis sin estructurar

Durante la década de los sesenta, con el asentamiento del concepto de la *programación estructurada*, se fue forjando un rechazo al recurso a esa intrucción de los lenguajes de programación de alto nivel. En su célebre carta, Dijkstra [1968] se declara convencido de que «the go to statement should be abolished from all "higher level" programming languages (i.e. everything except—perhaps—plain machine code)». Ese *perhaps* parecía estar pidiendo innovaciones radicales en la aquitectura de procesadores.

La propuesta de privación del recurso a los «gotos» no gozó del asentimiento general: «Dijkstra once told me that he actually received "a torrent of abusive letters" after publication of his article» [Knuth, 1974, p. 265].

Sin embargo, a pesar de lo contundente de algunas expresiones de la carta, no debería interpretarse esta, seguramente, como una condena absoluta sobre los saltos (Knuth [1974] reproduce estas palabras de Dijkstra: «please don't fall into the trap of believing that I am terribly dogmatical about»), ni menos aún deducir que la eliminación de los «gotos» es la fórmula mágica para lograr un programa bien estructurado.

Knuth planteaba como objetivo la definición de uno o varios juegos de estructuras, como el manejo de excepciones, para ponerlas a disposición del programador. Auguraba un lenguaje de programación

consensuado para el año 1984: «Will UTOPIA 84 or, perhaps we should call it NEWSPEAK, contain **go to** statements?».

Parece escéptico sobre su desaparición completa: «... several languages have appeared in which the designers proudly announced that they have abolished the **go to** statement. [...] which originally replaced **go to's** by eight so-called "escape" statements. And the eight weren't even enough; the authors wrote, "Our mistake was in assuming that there is no need for a label once the **go to** is removed," and they later [...] added a new statement "**leave** ⟨label⟩ **with** ⟨expression⟩" which goes to the place *after* the statement identified by the ⟨label⟩. Other **go to**-less languages [...] and language designers still feel the need for some euphemism that "goes to" without saying **go to**».

Añade: «... we shouldn't merely remove **go to** statements because it's the fashionable thing to do; the presence or absence of **go to** statements is not really the issue. [...] undisciplined **go to** statements make program structure harder to perceive, and they are ofter symptoms of a poor conceptual formulation». But there has been far too much emphasis on **go to** elimination instead of the really important issues; people have a natural tendency to set up an easily understood quantitative goal like the abolition of jumps, instead of working directly for a qualitative goal like a good program structure» [Knuth, 1974].

Si la controversia polarizada es ingrediente imprescindible para el interés general, la respuesta de Rubin [1987] abanderó la posición favorable a los «gotos», achacándole dos faltas a la carta de Dijkstra: ser académica y poco convincente. Según él, la identificación entre programación estructurada y sin «gotos» es lo que «has caused incalculable harm to the field of programming, which has lost an efficacious tool». Se queja de que «some people have devised program complexity metrics penalizing **GOTOs** so heavily that any program with a **GOTO** is *ipso facto* rated more complex than even the clumsiest **GOTO**-less program».

Rubin propone, para mostrar la utilidad del «goto», la tarea de programar la búsqueda de la primera fila no nula de una matriz cuadrada.

con

```
for (i = 0; i < n; i++){
  for (j = 0; j < n; j++)
    if (x[i][j] != 0) goto no_nula;
  printf("La primera fila nula es "
         "%d.\n", i);
  break;
 no_nula:;
 }
```

sin

```
i = 0;
do{
  j = 0;
  while (j < n && x[i][j] == 0) j++;
  i++;
 } while (i < n && j < n);
if (j == n)
  printf("La primera fila nula es "
         "%d.\n", i - 1);
```

Sin encontrar satisfactorias las contestaciones que se opusieron a la carta de Rubin [Moore et al., 1987], Dijkstra [1987] vuelve a la carga. Imbuido tal vez del ánimo de la disputa, entra a degüello, sin tolerarle a Rubin una vacilación entre mayúsculas y minúsculas siquiera. Rechaza, por ejemplo, el uso de las construcciones `and` (y `or`) condicionales, como las que aparecen en la versión «sin».

Dijkstra está a otros menesteres y no le duelen prendas en añadir variables: «a competent professional programmer [...] should not waste his time in pointing out that the boolean variable `d` is superfluous».

Referencias

E. W. Dijkstra. GO TO statement considered harmful. *Commun. ACM*, 11(3):147–148, 1968. http://doi.org/10.1145/362929.362947

E. W. Dijkstra. On a somewhat disappointing correspondence, 1987. http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1009.PDF

D. E. Knuth. Structured programming with **go to** statements. *ACM Comput. Surv.*, 6(4):261–301, 1974. http://doi.org/10.1145/356635.356640

D. E. Knuth y R. W. Floyd. Notes on avoiding 'go to' statements. *Information processing letters*, 1 (4):177, 1972. http://doi.org/10.1016/0020-0190(71)90018-4

D. Moore, C. Musciano y M. J. Liebhaber. "'GOTO considered harmful' considered harmful" considered harmful? *Commun. ACM*, 30(5):350–355, 1987. http://doi.org/10.1145/22899.315729

F. Rubin. "GOTO considered harmful" considered harmful. *Commun. ACM*, 30(3):195–196, 1987. http://doi.org/10.1145/214748.315722