On Goto Killing You in Your Sleep

1707981

1 __Introduction:;

In March 1968, the notorious 'Go to statement considered harmful' [1] letter was penned by Edsger W. Dijkstra, an influential pioneer of computer science during its early days – particularly if 'influential' were a measurement of the vast swathes of 'considered harmful' papers one could instantly inspire across acedemia (CITE), plus a revolution for structured programming pushing exhaustive efforts to deprecate the goto statement [2, 3]. His paper can be summarised as a criticism that using goto statements causes the flow of the program's execution to diverge greatly from the flow of text, by allowing the process to jump to various areas in code in an unstructured manner, with no obvious indicators from where it came. Due to the compromise in human readability he proposed that the use of goto statements be avoided outside machine level code.

Dijkstra's words were bold on their cricitism of what was once common practice in the era[CITE]. While his motivation was apparently just a negative correlation between a programmer's skill quality and the number of goto statements in their code [1, 4], the observent would recognise Dijkstra as the creator of the popular pathfinding algorithm now known by his name [5], and by its complexity could conclude that his objection to the goto statement was motivated because the ability to simply 'go to' some code is far too easy. Dijkstra's subsequent collision-free descent into madness, that pulled many programmers and academics along with him in the cold grasps of the "considered harmful" phenomenon, in the ever deeping existential crisis illustrating how virtually everything in programming is flawed in some way as our souls are meanwhile sold to the megacorporations of tomorrow [6], began around here.

For the next part **goto "To Pascal"**

2 Conclusion

If you arrived here before reading the rest of the essay please restart the program.

Today it is truly a wonder whether the goto statement has a right to exist, similar to the enigma of whether programmers have a right to exist with it. This appears to be why the goto statement is still around today; while perhaps only 10% of uses [7] go backwards, nobody can truly predict how the volatile community of programmers would react to a stripped feature. Thus it seems that goto is here to stay, at least until the next revolutionary programming

language comes along. With the dawn of quantum computing, there is still hope yet; until then, we still live primarily with programming languages standing at around 30 to 50 years old [CITE], and it seems we'll hesitate to 'go to' anywhere anytime soon.

3 "Goto" in Practice:;

To see whether a backwards goto statement should be useful, it is worth discovering where goto is typically used in the first place. An extensive study [8] of the world's largest open code repository [7], GitHub, found that most goto statements in modern C code are used for error handling. This is usually characterised by a function, sometimes one which loads various error-dependent resources, followed by a function end returning success, then followed by an 'error:' label of such, beneath which any loaded resources are freed before returning. This is popular, but does not necessitate a backwards-capable goto statement as the error condition is usually placed at the end of the function [8].

```
bool DeliberatelyCorruptThisEssay() {
  FILE *file1 = NULL, *file2 = NULL, *file3 = NULL;

  file1 = fopen("essay.tex", "w");
  [...] // do malicious stuff
  file2 = fopen("excuse_for_no_submission.txt", "w")
  [...] // do suspicious stuff
  if (!file2)
    goto Error; // don't worry I got this brah
  file3 = fopen("allies_in_crime.txt", "w");
  if (!file3)
    goto Error; // goto power corruption intensifies
  [...] // do duplicitous stuff
  Success:
  return true; // essay corrupted and defense constructed
  Error:
  [...] // cleanup for any number of unclosed files
  return false; // uh oh, guess we need to hand this in
}
```

Listing 1: An example of goto used for error handling (C++)

The study similarly concluded that 90% of goto uses were forward only, and believed to be used for reasonable purposes. It can thus be safely assumed that the programmer may in fact have a grasp of the devil's powers after all, and that goto can be used for good, and it is perhaps us who are evil.

It is likely this is the reason goto still exists: it is useful, and seemingly, the bigger concern was never the statement itself, but its habitual use: that was the inspiration behind Dijkstra's paper [1]. Indeed [CITE] it is clear that . It has however nurtured a culture of avoiding goto at all costs, even in potentially useful scenarios. [CITE]

4 OOP considered harmful?

5 To Pascal:;

Dijkstra's disdain was not unique. His paper itself found inspiration from the words of N. Wirth, who with C. A. R. Hoare [9] advised the demotion of labels and goto statements whilst proposing changes to the ALGOL language in

1965, with a similar rationale that the structure of a program would be clearer with structured 'case' statements and loops. In their proposal, 'goto' would be stripped of all complexities, leaving no more than a functional jump statement should the ability to jump anywhere in code be ever needed.

For a trivial side remark about the proposed case statements, set **x** to here, then **goto** "Switch statements".

It is arguable that it was Wirth who took the more active role in demoting goto in favour of structured programming. When he and his team's proposals for ALGOL were rejected in favour of a more complex overhaul (to become ALGOL 68) [10], Wirth went on to produce ALGOL W, whilst the Working Group created ALGOL 68. Soon after, ALGOL 68 failed, creating a new hole in the market through which Wirth threaded in 1970 with his new language, Pascal [10].

Wirth's goal in creating Pascal was to advance programming languages – despite the widespread adoption of far more popular, yet stagnated ones – resulting in a language with improved structure and an easier learning curve [11]. He focussed on encouraging structured programming, explained in great detail in a book partly authored by Dijkstra himself [12], wherein code is written in a tree structure, flowing downward roughly proportionally to the internal process.

Yet contrary to all those developments, for some reason, goto statements still existed, even implemented into Pascal itself [11]. **goto "Prevalence of the goto feature"**.

6 Prevalence of the goto feature:;

Like a forbidden fruit or a cigarette box, goto somehow still exists in programming languages, with a large 'DO NOT USE' written on its packet. In fact, one of the most popular programming languages of today, C, has a functional goto statement, described by D. Ritchie as infinitely abusable and unnecessary [13] as he gleefully implemented it into the compiler. It calls to question, if goto were truly harmful, why is it still included in modern programming languages? Even Microsoft's modern interpretation of C++, C#, features a version of goto that is in fact expanded to also support switch labels [14]. Why plant evil in the garden, water it and call it a weed?

There are valid uses for goto, some of which were illustrated in F. Rubin's article [15], disputing Dijkstra's original paper (and considered disappointing [16]). A popular use is when breaking out of deeply embedded loops under a particular condition [15]. This is in fact also described in the C programming book itself [13], raising the question; hypothetically speaking, is there not an alternative way to achieve this? When the issue is that we need to jump to a different area of code, namely a different scope, a jumping statement seems like a reasonable solution. And in order to break out of a scope into a different particular one, would it not be best to refer to it by a name–a label–so that the level of scope can change in the future? Is that not a goto statement?

```
void ImInTooDeep() {
  for (Friend* f = friends[0]; f < &friends[numFriends]; f++) {
```

```
    for (Friend* ff = f->friends[0]; ff < &f->friends[f->numFriends]; ff++) {
      for (Friend* fff = ff->friends[0]; fff < &ff->friends[ff->numFriends]; ff++) {
        goto FoundSomeone; // no other way to break all loops at once
      }
    }
  }

  printf("No-one relevant to your life likes your Pikachu plushie\n");
  FoundSomeone:
  printf("There is a glimmer of hope. Pika!");
  // ...but then they never returned. for this was a void function
}
```

Listing 2: An example of goto used for breaking loops (C)

An article by W. Wulf [3] addresses one particular solution to the above problem, featured in the Bliss programming language: to include a 'leave' statement to escape from a loop and jump to a label. This is safer than a goto statement with its risk of making code unreadable (which is goto's fault [CITE MANY], and definitely not the programmer's), since it can only jump forward. This is a very logical solution, but calls to question whether it is really worthy to replace goto, as merely a restricted form of the same thing? A deeper question would be whether there is any use left for a goto statement that goes backwards in a world wherein many believe a backwards goto in any text is explicitly evil[CITE]?

To find out **goto "Goto in practice;'**

It may be that 'goto' is included just to ensure the feature is available for special cases. However, given the extreme unpopularity of the goto statement[CITE], programmers are likely to avoid using it, sometimes at the expense of line count and computation time.

7 Switch statements:;

Interestingly, the 'case' statement proposal was intended to replace the 'switch' statement of ALGOL 60, which at the time was effectively a declaration of an array of labels to be used with a dynamic 'goto' statement [17]. What makes this interesting is the contrast to the modern switch statement of the C family today, which essentially defines 'switch' as what was once Pascal's 'case', and 'case' as what was once the labels, but restricted within the scope of the switch statement. As C was inspired by the ALGOL family, including Pascal, it is likely that the creator of C considered this small reinterpretation as a more correct terminology.

[put a figure here illustrating the different uses of the switch statement]

8 Stupid stuff

Granted, Dijkstra's pathfinding algorithm was in fact free. And it actually worked. But programmers work in mysterious ways. This author, for one, has paid their institution 9250 so that they can write essays for their institution. And frankly they make no sense at all.

And I for one would find it humourous that the A* algorithm was originally written entirely in goto statements.

Motive to subtly incorporate his pathfinding algorithm into every programming language on the market, where to write a program you had to place instructions in a complex maze with walls made of 0xCC, for the CPU to find and execute without colliding with the break instructions.

(From Pascal) If the divisor is a constant of $c = 2^n$, where *n* is *n*-ything, the compiler optimises this to a right bitshift.

9 Journal – personal thoughts on the paper itself

The opening paragraph of this paper is very bold. It's littered everywhere with 'go to' statements; frankly I feel that this article is extremely inefficient and looks like spaghetti code. Furthermore, it makes bold claims that 'go to' should be abolished from all programming languages. **Explore:** Uses of go to in the modern age, particularly in C++. Introduction of structured programming languages. Etc

In paragraph 3, he makes a point to better visualise the dynamic process of the program, noting that code is a static set of instructions representing a dynamic set of processes that is hard for the human brain to process. He wants to create a simpler link between the program (in text) and the process (spread over time).

In paragraph 4 he begins to make things complicated by introducing 'textual indices', where you can point to an area in the code with rough equivalence to the point of the process being performed by a computer. In paragraph 5 if and else statements are illustrated as similarly simple. No doubt, this will all become much more complicated when goto statements come into play.

In paragraph 5, procedures come in. Interestingly these are kind of like areas for goto statements, too. And I can speak from personal experience that writing my code in a messy, linear fashion; despite the cost of maintainability, actually makes it much easier to debug because all the code is written, linearly, in one place. This is one thing I can't wrap my head around in OOP. When things become moduler I struggle to wade through it all. But that's another story.

By paragraph 6, he explains what is essentially the stack. So long as you keep a list of the textual indices, referring to the calling points from which you jumped into the next procedure, you can characterise your progress through the program.

Paragraph 7 introduces while loops. He remarks that we could already theoretically create loop with functions that call themselves (not mentioning the inevitable stack overflow, mind). He remarks that textual indices are no longer sufficient. We've gone too deep. It's over. But not quite. Introducing the dynamic index, counting the number of repetition. With more nested loops, more textual and dynamic indices come in. But the process is still alive, and it's okay.

Paragraph 8 he finally explains why he took us on that perilous journey. The indices are markers of progress. They're vital, but always invisible to the programmer. Except when the programmer's debugging, but we don't want to do that before the bugs appear.

The remainder of his argument is a well-written elucidation of common programming sense. Go to lets you jump around haphazardly with no indications of the program's progress other than the state of the program itself.

10 Journal – thoughts on the context

It's 1968. Computers aren't even home computers yet. TODO: Read more about the historical context.

11 Journal – thoughts on related papers

The programming language pascal: Written with structured programming in mind. Made as a simple starting language. Supported data structures (compare to ALGOL?). Compiler for Pascal was written in Pascal.

12 OOP considered harmful?

Explore OOP, such as C++, and how it shares the confusing properties of goto.

References

[1] E. W. Dijkstra, "Letters to the editor: Go to statement considered harmful," *Commun. ACM*, vol. 11, pp. 147–148, Mar. 1968.

[2] O. Lecarme, "Structured programming, programming teaching and the language pascal," *SIGPLAN Not.*, vol. 9, pp. 15–21, July 1974.

[3] W. A. Wulf, "A case against the goto," *SIGPLAN Not.*, vol. 7, pp. 63–69, Nov. 1972.

[4] E. W. Dijkstra, "Programming considered as a human activity," pp. 213–217, May 1965.

[5] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, pp. 269–271, Dec. 1959.

[6] E. W. Dijkstra, "How do we tell truths that might hurt?," *SIGPLAN Not.*, vol. 17, pp. 13–15, May 1982.

[7] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, (New York, NY, USA), pp. 92–101, ACM, 2014.

[8] M. Nagappan, R. Robbes, Y. Kamei, E. Tanter, S. McIntosh, A. Mockus, and A. E. Hassan, "An empirical study of goto in c code from github repositories," in *Proceedings of the 2015 10th Joint Meeting on Foundations*

of Software Engineering, ESEC/FSE 2015, (New York, NY, USA), pp. 404–414, ACM, 2015.

[9] N. Wirth and C. A. R. Hoare, "A contribution to the development of algol," *Commun. ACM*, vol. 9, pp. 413–432, June 1966.

[10] N. Wirth, "History of programming languages—ii," ch. Recollections About the Development of Pascal, pp. 97–120, New York, NY, USA: ACM, 1996.

[11] N. Wirth, "The programming language pascal," *Acta Informatica*, vol. 1, pp. 35–63, Mar 1971.

[12] O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, eds., *Structured Programming*. London, UK, UK: Academic Press Ltd., 1972.

[13] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. 1978.

[14] A. H. et al., *The C# Programming Language*. 4 ed., Oct. 2010.

[15] F. Rubin, "Go to considered harmful considered harmful," *Commun. ACM*, vol. 30, pp. 195–196, Mar. 1987.

[16] E. W. Dijkstra, "On a somewhat disappointing correspondence (ewd1009)," May 1987.

[17] H. Bottenbruch, "An introduction to algol 60," *Journal of the ACM (JACM)*, vol. 9, pp. 161–221, 1962.