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. He helped push a revolution for structured programming [12], effectively sparking exhaustive efforts to deprecate the goto statement [2, 3] across the community. 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 in their criticism of what was once common practice [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 observant would recognise Dijkstra as the creator of the popular collision-avoiding pathfinding algorithm of the same name [5], and could conclude that his objection to the goto statement was motivated because the ability to 'go to' anywhere is far too easy. Dijkstra's subsequent collision-free descent into madness, which pulled many programmers and academics along with him in the cold, harmful grasps of the "considered harmful" phenomenon, in his ever deepening existential crisis illustrating how virtually everything in programming is flawed in some way as our souls are meanwhile sold to the terrible programmers in the megacorporations of tomorrow [6], began here.

For an illustration of Dijkstra's pathfinding algorithm, **go to "Pathfinding"**.

Else for the next section of this essay, **go to "To Pascal"**

5 "Conclusions":;

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 fully intact 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 or modified feature. Thus it seems that goto is here to stay until the next revolutionary programming language comes

along. Until then, we work our traditional programming languages standing at around 30 to 50 years old [CITE], hardly escaping the goto controversy in their days, as if in an infinite loop. While a goto-free world may be in our future, we hesitate to goto that place anytime soon.

Go to "Introduction"

4 "Goto in Practice":;

To see whether a backwards goto statement is useful, it is worth exploring where goto is typically used. 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 loading various error-dependent resources; followed by a function ending, 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 is likely these are the reasons goto still exists: it has fairly common uses; and in retrospect, the primary concern was never the statement itself, but its habitual use – such was the first inspiration within Dijkstra's paper [1]. It can thus be safely assumed that the programmer does in fact have control of the devil's powers after all, and that goto can be used for good; and it is perhaps us who are evil. As such, it is amusing that the statement itself has gained so much criticism compared to those who abuse it–and perhaps Wirth deserves more credit for actively promoting structured habits with Pascal [9], [10].

Go to "Conclusions"

2 "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 [11] 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 should be tree-oriented. 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 ever be needed.

As an interesting side remark, this required a restructuring of ALGOL's switch statement, to be replaced by the case statement [11]. The original switch statement was considerably different to its modern incarnation. For an example, **go to "Switch statements"**.

In fact compared to Dijkstra, 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 began work on 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 application of far more popular, yet stagnated ones – resulting in a language with improved structure and an easier learning curve [9]. 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 he originally [1] described.

Yet contrary to all those developments, for some reason, goto statements still existed, even implemented into Pascal itself [9].

Go to "Prevalence of the goto feature"

3 "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 [14] 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 [15]. Why plant evil in the garden and call it a weed?

On first glance, it appears that 'goto' is included just to ensure the feature is available for backward compatibility. However, the extreme unpopularity of the goto statement ([1], [3]) creates strong reasoning to remove it. As it stands, programmers are likely to avoid using it purely for sentimental reasons– sometimes at the expense of a larger line count and computation time [16], which is one consideration for its raison d'tre.

Indeed, there are valid uses for goto, some of which were illustrated in F. Ru-

bin's letter [17], disputing Dijkstra's one (and considered 'disappointing' [18]). A popular use is when breaking out of deeply embedded loops under a particular condition [17]. This is in fact also described in the C programming book itself [14], raising the question; hypothetically speaking, is there not an alternative way to achieve this? When one needs 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 into a shallower scope, would it not be best practice to refer to it by a name? 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++) {
        if (fff->LikesYourPlushie())
          goto FoundSomeone; // no other way to break all loops at once
      }
    }
  }

  printf("No-one relevant to your life likes your Pikachu plushie\n");
  return; // end function here if goto not occured
  FoundSomeone:
  printf("There is a glimmer of hope. Pika!");
  return; // here returning from the Void is optional, but we'll do it for health
       and safety
}
```

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

An article by W. Wulf [3] addressed 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 potentially more readable than a goto statement, since it can only jump forward. This is a logical solution, but calls to question whether it is truly worthy to replace goto, as seemingly merely a restricted form of it? On the other hand, is there ever anything helpful about a goto statement that goes backwards?

To find out **go to "Goto in practice;"**

2 "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 [13]. Heavily contrast to the modern switch statement of the C family today, which essentially defines 'switch' as what was Pascal's 'case', and 'case' as what was once the labels, except restricted within the scope of the statement. As C was inspired by the ALGOL family [CITE], it is likely that the creator of C considered this small reinterpretation as a more correct terminology.

Go to "To Pascal"

1 “Pathfinding”:;

See below for an implementation of the Dijkstra’s algorithm [5], remarkable in that it actually compiles. *tiles* represents an 8x8 array of enumerations of either Tile::W or Tile::E where Tile::W is a wall. Visiting left and right nodes may wrap around to a different row, and exceeding the bottom left and top right corner will crash; these are special features for extra usability. It is unclear whether it actually functions, yet it does; unfortunately suggesting the computer has a better knowledge than the programmer who should be controlling it. Perhaps more goto statements would help. [?!]

```
std::vector<int> vertices, distance(8 * 8), previous(8 * 8);
int source = startY * 8 + startX, destination = endY * 8 + endX; // hack
int i = 0;
LoopStart: // there has to be a better way!
    if (i >= sizeof (tiles) / sizeof (tiles[0]) /* hack */)
        goto LoopDone;

    if (tiles[i] == Tile::W) { // wall
        i++; // increment i
        goto LoopStart; // hack
    }
    distance[i] = INT_MAX; // INT_MAX hack!
    previous[i] = -1; /* also -1 hack */
    // optimise[i] = i * i / (i % 8) * 32 + 42 / 0 // what did this do again?
    vertices.push_back(i);
    i++; // increment i... again
    goto LoopStart;
LoopDone:
// initialise distance to source from source, which is probably a product of 0 and infinity
distance[source] = 0;
LoopStart2: {
    if (vertices.empty())
        goto LoopDone2;
    // pick the shortest-distance vertex in vertices
    auto closestItem = vertices.begin();
    int closestDist = distance[*closestItem];
    auto item = vertices.begin();
    LoopStart3: {
        if (item == vertices.end())
            goto LoopDone3;
        if (distance[*item] < closestDist) {
            closestItem = item;
            closestDist = distance[*item];
        }
        item++;
        goto LoopStart3;
    } LoopDone3:
    // find neighbours of this item in vertices
    item = vertices.begin();
    LoopStart4: {
        if (item == vertices.end())
            goto LoopDone4;
        if (item == closestItem) {
            item++;
            goto LoopStart4;
        }
        // todo: make code readable todo2: make code work
        if (*item == *closestItem + 1 || *item == *closestItem - 1 || *item == *closestItem + 8 || *item == *closestItem
- 8){
            int newDistance = distance[*closestItem] + 1;

            if (newDistance < distance[*item]) {
                distance[*item] = newDistance;
                previous[*item] = *closestItem;
            }
            if (*item == destination)
                goto Success; // found it! todo make rest of the app
        }

        item++;
        goto LoopStart4;
    } LoopDone4:
    // remove it from vertices
    vertices.erase(closestItem);
    goto LoopStart2;
} LoopDone2:
```

Fig. 1: A C++ implementation of Dijkstra’s algorithm [5] in Comic Sans

Go to "To Pascal"

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," in *Proceedings of the IFIP Congress 1965*, 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 *et al.*, "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 *et al.*, "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, "The programming language pascal," *Acta Informatica*, vol. 1, pp. 35–63, Mar. 1971.

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

[12] O. Dahl *et al.*, *Structured Programming*. London, UK: Academic Press Ltd., 1972.

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

[14] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. Upper Saddle River, NJ, USA: Prentice Hall, 1978.

[15] A. Heljsberg *et al.*, *The C# Programming Language*. Addison-Wesley Professional, 4 ed., Oct. 2010.

[16] M. E. Hopkins, "A case for the goto," in *Proceedings of the ACM Annual Conference - Volume 2*, ACM '72, (New York, NY, USA), pp. 787–790, ACM, 1972.

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

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