კომპიუტერული რევოლუცია ჯერ არ მომხდარა, კოდის ზომით არის რაღაც და goto-ს შეზღუდვა არ არის გამოსავალი

არჩილ ლთარ ბოლქვაძე

ბოლო წლების განმავლობაში მე ვმუშაობდი ახალი სამ განზომილებიანი კომპოზიციის ერთეულის (შემდგომში არადანი) შექმნაზე. საბოლოოდ დავარღვიე არსებული დოგმებადქცეული წესები და კანონები და შევიცანი არადანი. რომის პრეზენტაცია მოვახერხინე ჩემი youtube-ს არხის საშუალებით. შედეგად ჩემთვის ცნობილი გახდა რომ იდეის არსის დანახვა ვერ მოხერხდა და დავიწყე გამოსავალის ძებნა. გადავწყვიტე ედსგერ ვიბე დიკსტრას[1] go-to-ს შესახებ წერილის წაკითხვა. წერილმა ძალიან ჩამითრია, კარგად ვერ გავიგე მის მიერ განსაზღვრული ტექსტური ინდექსის იდეა, მაგრამ მერე გავიაზრე რომ ყველა იმ ძირითად საკითხს რომელსაც დაიქსტრა შეეხო თავის წერილში, არადანი სრულად განსაზღვრავს მათ და გამოსავალი ხელთ მაქვს.

გადავწყვიტე ამ წერილში წარმოგიდგინოთ ედსგერ ვიბე დიკსტრას მოსაზრებები, ციტირების სახით და შემდეგ შევავსო ჩემი განმარტებებით. ვისაც პროგრამირება უყვარს, ისმინოს!

Go To Statement Considered Harmful

Edsger W. Dijkstra

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. More recently I discovered why the use of the go to statement has such disastrous effects, and I became convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

ავღნიშნოთ ის გარემოება, რომ ედსგერ ვიბე დიკსტრამ წერილი დაიწყო განაცხადებით, იმის შესახებ რომ go-to-ს გაუქმებით პროგრამისტების ხარისხი გაუმჯობესდება. შეგახსენებთ ეს წერილი 1968 წელს დაიწერა.

1 Edsger Wybe Dijkstra (/ˈdaɪkstrə/ DYKE-strə; Dutch: [ˈɛtsxər ˈʋibə ˈdɛikstra] (About this soundlisten); 11 May 1930 – 6 August 2002) was a Dutch computer scientist, programmer, software engineer, systems scientist, science essayist, and pioneer in computing science. A theoretical physicist by training, he worked as a programmer at the Mathematisch Centrum (Amsterdam) from 1952 to 1962. A university professor for much of his life, Dijkstra held the Schlumberger Centennial Chair in Computer Sciences at the University of Texas at Austin from 1984 until his retirement in 1999. He was a professor of mathematics at the Eindhoven University of Technology (1962–1984) and a research fellow at the Burroughs Corporation (1973–1984). In 1972, he became the first person who was neither American nor British to win the Turing Award.

ჩემი მხრიდან დაკვირვების შედეგად კი დავამატებდი რომ პროგრამირების არსი დღემდე ვერ შევიცანით. ასევე დავიმოწმებდი ალან კურტის კეის[2] თავისი მოხსენებით OOPSLA 1997 "კომპიუტერული რევოლუცია ჯერ არ მომხდარა", Leslie B. Lamport[3]-ს მოხსენებით 2014 Microsoft Research "Thinking Above the Code" და ყველა იმ პროგრამისტს რომელსაც გისმენთ და გიცნობთ.

My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to accomplish the desired effect; it is this process that in its dynamic behavior has to satisfy the desired specifications.

ჭეშმარიტად! პროგრამის კონტროლის ქვეშ მიმდინარე პროცესების მართვა მეტად მნიშვნელოვანი ამოცანაა. ვიდრე პროგრამის ტექსტის შედგენა. ამას ადასტურებს ის გარემოებაც რომ პროცესების სამართავად დღეს ჩვენ გვიწევს დინუქსპროცესისის, კონტეინერების, ვირტუალური მანქანების, და ა.შ. გამოყენება. აქ პრობლემა ის არის რომ აღნიშნული ინსტრუმენტები თავის მხრივ არიან კომპლექსურობით აღსავსე პროგრამები და შესაბამისად ჩვენ როგორც პროგრამისტებს, აღარაფერი დაგვრჩენია იმის გარდა, რომ კოდერები გავხდეთ და აქა იქ სკრიპტებში გაწერ-გამოწერილი კოდების ხარჯზე გამოვიყენოთ ჩვენთვის ბოძებული მართვის მექანიზმები.

რასაკვირველია ღრუბლების ხანაში ღრუბელი იძლევა პროგრამირებად ინტერფეისს (ენას) აღნიშნული პრობლემის მოგვარებისთვის, მაგრამ დაბალ დონეზე როგორც ასთან ახლოს (საწყის ენასთან) პრობლემა მოუგვარებელია.

Yet, once the program has been made, the "making' of the corresponding process is delegated to the machine.

აქ ჩვენ შეგვიძლია უფრო დავკონკრეტდეთ, პროცესის "დამზადება" (პროგრამის ტექსტის მეხსიერებაში ჩატვირთვა მისი სტატიკური/დინამიური გადაბმა ოპერაციულ სისტემასთან და დინამიურ ბიბლიოთეკებთან) და მისი მართვაც (შეჩერება, "პრეემტივ", "დებუგერიდან" ან სხვა შემთხვევებში) გადაეცემა ოპერაციულ სისტემას. შესაბამისად ჩვენი, როგორც პროგრამისტების, მიზანი უნდა იყოს გაცილებით მარტივი მართვის მექანიზმის შემუშავება ვიდრე დინუქსის პროცესია.

---

2 Alan Curtis Kay (born May 17, 1940) is an American computer scientist. He has been elected a Fellow of the American Academy of Arts and Sciences, the National Academy of Engineering, and the Royal Society of Arts. He is best known for his pioneering work on object-oriented programming and windowing graphical user interface (GUI) design. He was awarded the Turing award in 2003.

3 Leslie B. Lamport (born July 2, 1941) is an American computer scientist. Lamport is best known for his seminal work in distributed systems, and as the initial developer of the document preparation system LaTeX and the author of its first manual. Leslie Lamport was the winner of the 2013 Turing Award for imposing clear, well-defined coherence on the seemingly chaotic behavior of distributed computing systems, in which several autonomous computers communicate with each other by passing messages. He devised important algorithms and developed formal modeling and verification protocols that improve the quality of real distributed systems. These contributions have resulted in improved correctness, performance, and reliability of computer systems.

კვლავ დავიმცწმებ აღან კურტის კეის თავისი მცხსენებით OOPSLA 1997 "კცმპიუტრტული რევცღუცია ჯერ არ მცმხდარა".

My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

არადანში გვაქვს სამი საწყისი ტექსტის (შემდგცმში T), ახაღი ტექსტის (შემდგცმში O) და გუღგუღის (შემდგცმში S) განზცმიღებები. რცმღებიც ერთიანდებიან სრულმნიშვნელღვან სიტყვაში (შემდგცმში I).

I არის T განზცმიღების ერთ-ერთი მისამართი. I-იდან T განზცმიღებაში T ენაზე სიტყვების თანამიმდევრცბა წერია. ნებისმიერი რამ go-to-ს ჩათვღით, მაგრამ მთღიანცბაში განსაზღვრავენ I-ს. რაც იმას ნიშნავს რცმ ნებისმიერი T-ნაბიჯი შეიძღება ახდენდეს O ანდა S - ის მცდიფიკაციას სინტაქსის მიხედვით. ბცღც T-ნაბიჯი წარმცადგენს I-ის ღცგიკურ არჩევანს თუ რცმელი კავშირით ("არა", "და" და "ან") გაადგრძელდეს წინადადება S-ის საშუაღებით.

O და S იმპღემენტაციის დცნეზე არის stack-ივით სტრუქტურა მეხსიერებაში ერთმანეთისკენ მიმართული.

O - ზე მარცხნიდან მარჯვნივ იწერება I-ები და რიცხვითი მნიშვნელცბები, წინადადების სინტაქსური გადაბმის გათვაღისწინებით. არადანში ეს სინტაქსი მათემატიკური ღცგიკის პრინციპებს ექვემდებარება კავშირებით "არა", "და", და "ან". S-ი კი წარმცადგენს O-ზე დაწერიღი წინადადების შესრულების მექანიზმს, რცმღის მართვა ჩვენს - S-ის ხელშია, და მთელ ამ პრცცესს ასრულებს T-ს შემსრულებელი (fetch, decode, execute, write loop) რცმელიც სულ წინ მიდის.

ყცველ T ნაბიზე ჩვენ შეგვიძღია შევხედთ/შევცვაღთ O და S -ის მდგცმარეცბები. შესაბამისად ჩვენ შეგვიძღია ვმართთ შესრულების პრცცესი სრულყცფილად. და ბუნებრივად ვღებულცბთ პრცგრამის ქცევის დინამიურ "ხაზს".

მხცღდ ამ შემთხვევაშია შესაძღებელი, რცმ დრც გავასტატიკურთ და გამცვიყენთ ჩვენი უნარები ეფექტურად და არსებული ტღანქი მექანიზმი "ღინუქპრცცესით დაწყებული" გავხადთ ისეთივე მსუბუქი რცგრციც C ენის ფუნქციაა.

Let us now consider how we can characterize the progress of a process. (You may think about this question in a very concrete manner: suppose that a process, considered as a time succession of actions, is stopped after an arbitrary action, what data do we have to fix in order that we can redo

the process until the very same point?) If the program text is a pure concatenation of, say, assignment statements (for the purpose of this discussion regarded as the descriptions of single actions) it is sufficient to point in the program text to a point between two successive action descriptions. (In the absence of go to statements I can permit myself the syntactic ambiguity in the last three words of the previous sentence: if we parse them as "successive (action descriptions)" we mean successive in text space; if we parse as "(successive action) descriptions" we mean successive in time.) Let us call such a pointer to a suitable place in the text a "textual index."

ჩვენ ასევე შეგვიძლია, რომ პროცესი დროში ზუსტად განვსაზღვროთ არადანის საშუალებით.

პროგრამის ტექსტი წარმოვიდგინოთ როგორც 0-ზე დაწერილი წინადადება მაგ: “ერთი ერთი და შეკრიბე და.” და დინამიური პროცესი კი - დაწერილი სიტყვების შესრულების შედეგების (0 და S ზე დანახული ინფორმაციის) თანმიმდევრობაა. ტექსტური ინდექსი მცემულ მაგალითში განისაზღვრება წინადადების სინტაქსური აგებულებიდან გამომდინარე მაგ: “i0 ერთი i1 ერთი i2 შეკრიბე i3”.

When we include conditional clauses (if B then A), alternative clauses (if B then A1 else A2), choice clauses as introduced by C. A. R. Hoare (case[i] of (A1, A2,···, An)),or conditional expressions as introduced by J. McCarthy (B1 -> E1, B2 -> E2, ···, Bn -> En), the fact remains that the progress of the process remains characterized by a single textual index.

გავიაზროთ ის გარემოება რომ ჩვენ ახლა ორი ტექსტური ინდექსი გვაქვს T-ტექსტის და Π-ტექსტის ინდექსები, რომელთანაც ბოლოს კონტროლი ჩვენ ხელთ არის. შესაბამისად ჩვენ გვაქვს საშუალება თავად გადავწყვიტოთ რამდენი, ასევე რომელი T-სიტყვები (T-ინდექსი) გამოვიყენოთ ერთი Π-ის განსამარტებლად.

As soon as we include in our language procedures we must admit that a single textual index is no longer sufficient. In the case that a textual index points to the interior of a procedure body the dynamic progress is only characterized when we also give to which call of the procedure we refer. With the inclusion of procedures we can characterize the progress of the process via a sequence of textual indices, the length of this sequence being equal to the dynamic depth of procedure calling.

Π - ტექსტის/წინადადების შესრულების მექანიზმი იმგვარად მუშაობს (return - ის გარეშე), რომ შესრულიების პროცესი სწორხაზოვანია. სხვა სიტყვებით მისი დინამიური სიღრმე სწორხაზოვნად იზრდება.

Let us now consider repetition clauses (like, while B repeat A or repeat A until B). Logically speaking, such clauses are now superfluous, because we can express repetition with the aid of recursive procedures. For reasons of realism I don't wish to exclude them: on the one hand, repetition clauses can be implemented quite comfortably with present day finite equipment; on the other hand, the reasoning pattern known as "induction" makes us well equipped to retain our intellectual grasp on the processes generated by repetition clauses. With the inclusion of the

repetition clauses textual indices are no longer sufficient to describe the dynamic progress of the process. With each entry into a repetition clause, however, we can associate a so-called "dynamic index," inexorably counting the ordinal number of the corresponding current repetition. As repetition clauses (just as procedure calls) may be applied nestedly, we find that now the progress of the process can always be uniquely characterized by a (mixed) sequence of textual and/or dynamic indices.

ჩემი მსაზრებით “სტრუქტურული პრგრამირება” ზღუდავს ჩვენს წარმსახვით უნარებს ხელთ არსებული პრბლემის გადასაწყვეტად. ჩვენ რგრც პრგრამისტები, უნდა ვცდილბდეთ პრბლემის აღსაწერი ენის შექმას (სპეციფიკაციის მსგავსი რამ თან ზუსტად განსაზღვრული T-სივრცეში I-ების საშუალებით) და პრბლემის აღწერას ამ ენაში. ჩვენ უნდა ვიყთ პრფესინალები გრამატიკაში (pro in grammer). ჩვენ უნდა შევიცნთ ენის და სიტყვის ძალის არსი!

The main point is that the values of these indices are outside programmer's control; they are generated (either by the write-up of his program or by the dynamic evolution of the process) whether he wishes or not. They provide independent coordinates in which to describe the progress of the process.

მთავარი ის არის, რმ პრგრამისტის ხელთ მცქცა ის რაც აქამდე რკინაში, ან კმპილაციის დრს იყ გასაკეთებელი და გახდა დინამიური და მართვადი.

Why do we need such independent coordinates? The reason is - and this seems to be inherent to sequential processes - that we can interpret the value of a variable only with respect to the progress of the process. If we wish to count the number, n say, of people in an initially empty room, we can achieve this by increasing n by one whenever we see someone entering the room. In the in-between moment that we have observed someone entering the room but have not yet performed the subsequent increase of n, its value equals the number of people in the room minus one!

არადანში გვაქვს ზუსტად განსაზღვრული ტექსტის ინდექსი, შესაბამისად გვაქვს განსაზღვრული ის მმენტი რდესაც ცვლადის მნიშვნელბის ინტერპრეტაციაა შესაძლებელი. თავად არადანი სამ არხიან ბმას წარმადგენს. ჩვენ შეგვიძლია ვიფიქრთ უფრ მეტ ან ნაკებ არხიან კნსტრუქციებზეც.

პრგრამისტის გადასაწტვეტი იყს რა სრთული T-ტექსტის ინდექსი ექნება. იყს პრგრამისტის გემცნებაზე. ის ხმ I-ებს ქმნის. და ვენდთ მის I-ებს.

არჩიე ცთარ ბერქვაძე

ე. ხარაძის სახელბის აბასთუმნის ასტრფიზიკული ბსერვატრია.

Alan Kay at OOPSLA 1997 - The computer revolution hasnt happened yet - https://www.youtube.com/watch?v=oKg1hTOQXoY&t=368s

Leslie Lamport: Thinking Above the Code - https://www.youtube.com/watch?v=-4Yp3j_jk8Q

The unbridled use of the go to statement has an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress. Usually, people take into account as well the values of some well chosen variables, but this is out of the question because it is relative to the progress that the meaning of these values is to be understood! With the go to statement one can, of course, still describe the progress uniquely by a counter counting the number of actions performed since program start (viz. a kind of normalized clock). The difficulty is that such a coordinate, although unique, is utterly unhelpful. In such a coordinate system it becomes an extremely complicated affair to define all those points of progress where, say, n equals the number of persons in the room minus one!

The go to statement as it stands is just too primitive; it is too much an invitation to make a mess of one's program. One can regard and appreciate the clauses considered as bridling its use. I do not claim that the clauses mentioned are exhaustive in the sense that they will satisfy all needs, but whatever clauses are suggested (e.g. abortion clauses) they should satisfy the requirement that a programmer independent coordinate system can be maintained to describe the process in a helpful and manageable way.

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.

References:

Wirth, Niklaus, and Hoare C. A. R. A contribution to the development of ALGOL. Comm. ACM 9 (June 1966), 413-432.

BÖhm, Corrado, and Jacopini Guiseppe. Flow diagrams, Turing machines and languages with only two formation rules. Comm. ACM 9 (May 1966), 366-371.

Edsger W. Dijkstra
Technological University
Eindhoven, The Netherlands