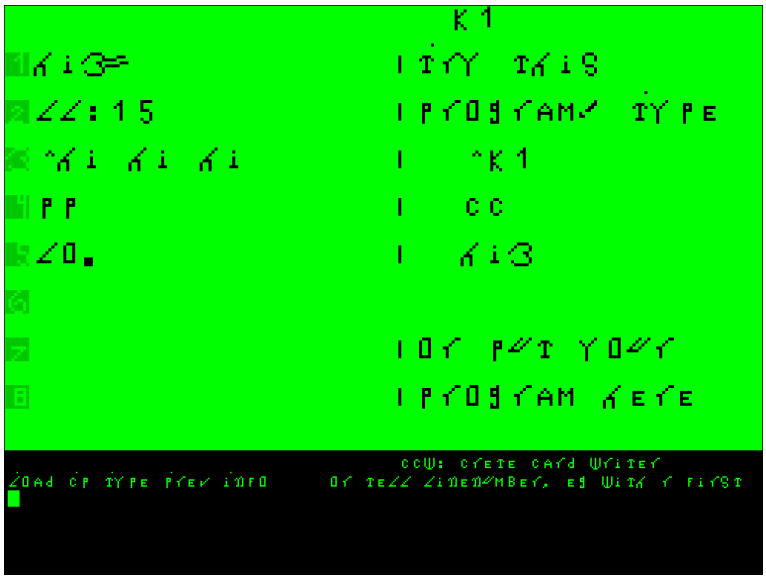
G15 PMN: A GENERAL PROGRAMMING LANGUAGE GOING ALONG WITH A CONCEPT OF A COMPUTER AND ITS OS

by Aristo Tacoma



A programming language for a digital computer has, as one of its criteria for existence, that it is meaningful to the human mind. In contrast, the series of bits, whether assembled into larger constructs like numbers in some number-system, or not, which represent coding in a digital CPU, are much less meaningful.

A programming language, in contrast to a menu system for modifying an existing application on a computer, is something which can program the very computer itself. That's also one of its criteria for being a programming language. While it is typical in most languages that new programs can rely on libraries of earlier

programs in the same language, eg by common names of functions or variables or whatever, if there is a 'block' of something entirely different between the programming language and the computer it is tempting to regard it more as 'application language' and not a general programming language.

I have seen diagrams which describes the existence of a programming language as existing not only on top of a CPU (whether multicore or not), but also on top of a block called, innocently, a 'file system'. Underneath the file system are the disks and such. Also, on these diagrams, we see the RAM, and similar.

This innocent label, 'file system', usually, in these days, involves not only some indices here and there over what may be on such and such sector of such and such disk, but it is a database of searchable and hierarchically ordered, dynamically changable files and folders which have almost nothing to do, conceptually, with the sequential sectors of the disks or what goes for the disks. In fact, a file system in these days is a giant application, requiring an immense set of programs to bridge the simplistic CPU instructions with the simplistic disk sectors in a manner that allows a vast hiearchy of folders and files to be sorted and reorganized and used in all sorts of ways. While it can be conceded that in many circumstances such extreme database structures are practical, they stand out as something foreign to the computer in its core essence and it is not just strange, but fairly absurd, to erect programming languages that _presumes_ something so utterly complex in the nature of what has to be programmed _before_ this programming language.

While a language such as C can be seen to exist, eg in the form of libraries, in the build-up of the extremely complex databases called 'file systems', and therefore can be argued to exist in a manner that does not exactly presume the file system, it is nevertheless true that in any typical C application, a file system is presumed also for the C programming effort, and it is more theoretical that C can be said to be part also of the underlying database. In most cases, for most programming languages, they are nowhere near being part of the underlying giant extremely complex database. They simply assume that it exists as part of the basal and simple facts of what a computer is all about--and this leads me to suggest that most so-called general programming languages aren't general at all. They are more properly 'application languages'.

With G15 PMN, there is no assumption at all of any underlying file system. The CPU is assumed to have less than 300 instructions, all

of which are either simple or, given some explanation and thought, fairly simple. The G15 assembly refers to this CPU and to a dozen disks or so which are divided into one unit only, which it-- similar to the early FORTH implementations in this regard-- calls 'cards'. These are sequentially laid out and identified by a number from one and up to around two hundred million.

A programming language inevitably involves the use of numbers. These numbers can often be put in some nameable structure like a variable, but they should, like the programming language as a whole and in all its parts, make sense to the human mind. Once a programming language is being rebuilt to be independent on the quantity of bits in its numbers, it is also being rebuilt so it no longer fulfills the essential criterion for it to be a programming language, as stated initially-- namely, that it makes sense to the human mind. A number like 1,234,567,890, which is somewhat above one billion makes sense, and fits with the human psychological attention span which, in the brief 'cartoon' form as psychologists sometimes put it, is seven plus-minus two items to focus on. The psychologists can use this to explain why a digit series like

    3,839,239,378,282,488,882,878,183,500

doesn't look like a number but rather like a digit series. It's because it is not meaningful unless one devotes half one's brain to work on a daily basis with 64-bit numbers. The 32-bit numbers, on the other hand, go conveniently up to plus minus two billion, which is, in terms of digits, seven plus two equals nine digits and make perfect sense.

While there is a fascination in humans for technologies that can 'do more', it is also clear that computers, with the immense power they have, ought to be under human control and we should not pride ourselves on efforts that make constructions in the extension of what was originally computers which one could program in a meaningful way to monster structures which defy understanding and so, whether by chaos or by targeted program elements inside such monster structures, can become a danger to humanity.

So understanding a computer is part of what contributes to natural ethical lawful use of the computer. This understanding involves appreciating the power, but also the limit, of the 32-bit number. Around year 2000, in what I have earlier called the y2000-compliant Personal Computer, we found the 32-bit Intel 586 chip at the core of marvellous developments in the software industry. About two decades earlier, IBM researchers had concluded that humans work best with bright green monitors. I have taken these concepts together with--true, what is not yet a real G15 CPU, only a virtual CPU that relies on the assumption of a hierarchical file

system and another type of CPU underneath for the time being--but it is at least a principled concept. When Turing introduced the computer idea, he did it by means of a principled CPU. It took him a lot of time to actually build one. But each and every instruction could be talked about and they could even be programmed with in thought experiments.

In the practical virtual implementation of the G15 CPU, which I also call a 'PVI', with green-screen tuned to meaningful parameters in green tones so as to render both artistic beauty and crisp-clear good text to work with, we have a whole set of programs all programmed in the G15 assembly and the PMN compiler/interpreter written in G15 assembly on top of it (with very approximately half the inspiration coming from FORTH). Some of these programs move cards from here to there or scan cards. But it is fundamentally a less hierarchical approach, also to RAM. For with a language like Lisp, or with most socalled object- oriented programming language, RAM is assumed to be a bunch of 'blocks' which can be created, expanded, and dissolved effortlessly. There is no such RAM like that in actual physical computers. RAM is even simpler than disks--it is generally just laid out in one big sequence, each addressed by (what we presume) is a meaningful number. In the 32-bit computer concept, which we regard as the most general computer because it has adequate complexity for human meaningful tasks of an enormous variety while not being too complex for human understanding in any bit of it, RAM is addressed by a 32-bit number. A programming language should be near the computer structure also in that it treats 'computer memory' the way it is, rather than the way it ideally could have been given the ideosyncratic leanings of the makers of the programming languages. Here, FORTH did it right, and most languages haven't done it nearly so well, but Rust have picked up some points from FORTH. And G15 PMN treats RAM in a straight-forward way, pretty much like FORTH.

When it comes to keyboard and font, again the human criterion must come afore: a keyboard is a natural interface with a computer, and the mouse an ideal companion to the keyboard. With both hands on the keyboard, moving rapidly as 4-5 finger on each hand clicks around, it follows that a keyboard of the QWERTY-type at least as size and quantity of keys go, expanded with some extra useful function keys here and there, makes most sense. When it comes to language, a programming language should be near the computer structure also in the sense that one click on the keyboard should, for visible characters, result in one visible character rather than each character being built up by several keyclicks. Most human languages can be rendered with some precision to a Latin-

like alphabet and English, with few accent marks and an extremely large vocabulary of intercultural and international definition, has been found working as an intercultural language to a larger extent than other proposed alternatives. It is therefore a good approach for a core computer concept with an essential programming language concept to have the notion of something like the US/Ascii 7-bit keyboard and set of characters and typical core set of computer-relevant words as part of its definition. The 8-bit byte structure is however more attuned to the way CPUs are designed--here, especially since 4 times 8 equals 32-bit numbers--so there is room for non-visible characters representing such as function keys in the 8-bit region. This was the approach of most computers leading up to the IBM PC that sort of set the standard for this.

For a programming language to be meaningful, its characters must serve the purpose of clarity during programming. When we take the typical fonts as Times New Roman and apply to programming, we find that some of the characters are ambigious or not as clear as they should be, in a context where the programming must be exact. We do not consider it part of the computer concept that there is a huge application translating presumed mistakes on the programmer's part into correct programs. We want the programming language to actually have control of the computer. In this regard, the font must be helpful to the programmer's mind, it must guide attention useful. In going from a font like Times New Roman to one like Courier New, we see that some of the ambiguities get cleared up, such as a sharper distinction between the digit 1 and the noncapital letter l. However, for intense programming, it can be helpful to have yet more contrast. And since this is part of the work with the computer even at assembly level, in G15 PMN there is a core font, called RBOTFNT, which is entirely free from ambiguities once one gets used to it, and which is defined by fairly few pixels and which is ultra-sharp. The CAR card editor, as written in G15 assembly in its tiny freeware core, just as the CCW Crete Card Writer written in G15 PMN, uses this font to emphasize that which has to be emphasized for programming to work out smoothly. Similar efforts but more tuned to text processing with human language have gone into the shaping of a Courier and Courier Italic (as for numbers) font called the B9Font, which is also part of the G15 OS. Other fonts can be made with fair ease, of course, but these two run through the typical G15 PMN applications.

G15 PMN is a programming language that comes along with its own concept of CPU and disks, and which stays near to the physical idea of the computer and its human-meaningful limits. In this approach, we have also a formal language, in which theories can

get formal illustrations of some features of them, because the assumptions are not hidden inside assumed structures of application-size but rather the whole 'box' is pretty much defined and knowable from the core and up to all its open source.

Through this whole configuration, the computer, with its keyboard, mouse, display, CPU, RAM, disks, and programming language, permits the control with input, output, or input/output, relative to of all suitable peripheral units, including but not limited to: backup devices, network modems, printers, other types of keyboards/displays/mouse pointer devices [also combined], other types of keyboards, robotic motors, servos, sensors, and cameras.

It is the proposal of me as author of it that we have with G15 PMN what is in practice a _more_ general programming language than those languages which have been made in an attempt to 'abstractify away' the RAM, the disks, the number sizes, and the CPUs, because it is well-defined relative to a domain which is also well-defined, and well-definedness allows it to go into the future in a way which has maximalized meaningfulness.

Article text written: August 2025

The language can be installed
from www.g15pmn.com and this
text is also found at
www.yoga6dserver.org and at
yoga6d.org/library
The author, Aristo Tacoma,
can be contacted at berlinib@aol.com.

Earlier pen names for same
author includes Stein Reusch
Weber, Stein von Reusch, and
relates to the formal name
Stein Henning Reusch. Aristo
Tacoma is the active artist
name and editor name.