Variables in Programming

Brian Rabern[*]

January 3, 2025

"The variable is perhaps the most distinctively mathematical of all notions; it is certainly also one of the most difficult to understand" (Russell 1903, §86)

"Once a person has understood the way in which variables are used in programming he has understood the quintessence of programming" (Dijkstra 1972)

In computer science, it is often said that a variable is a "memory location" or a "container that stores a value". It is useful to think of a variable as a box that can be filled with different things during the execution of a program. However, while it's common to talk about variables in terms of memory, it's important to recognize that this way of speaking potentially introduces an ambiguity between the expression in the programming language (e.g., 'x') and the memory address it may be associated with (e.g., `0x7fffd3f6aabc`). Consider the famous quip:

> "Naming variables is one of the hardest problems in computer science"[1]

To my ear the locution "naming a variable" sounds like a confusion of some sort. It reminds me of the confusion that ensues when the Knight in *Through the Looking-Glass* says to Alice: "The name of the song is called "Haddocks' Eyes". Of course, we *can* give names to names. For example, we can name my surname 'Sam', so that Sam is spelled R-a-b-e-r-n. But if one wishes to speak of "naming variables" in this sense, then one should be clear that the name of the variable is not the variable itself.[2] The "variable" in this sense is

[*]`brian.rabern@gmail.com`, https://brianrabern.net

[1]The origin of this quote is a bit murky. It's been attributed to the computer scientist Edsger Dijkstra. But I can't verify this. The software developer Phil Karlton is credited with the related famous quip: "There are only two hard things in Computer Science: cache invalidation and naming things".

[2]That is, don't make a use/mention error. Remember the riddle: What is orange and rhymes with 'parrot'? (Hint: The answer is not 'carrot'! Words aren't orange and veggies don't rhyme.)

a memory location that can be accessed and modified through the "name" in the code. That would make sense.

In contrast to this way of speaking, though, it is also often said that a variable is a *reference* to a memory location. The term "reference" might be a bit misleading, as it suggests that a variable refers to a memory location in the same way that your name refers to you. But a variable in this sense isn't simply a name for a memory location; it is an expression in a programming language that can be evaluated to produce a value. The value of a variable is the value stored at the memory location it "references". In this case, the value can be accessed and modified through the variable, but the variable itself is not the memory location.

Both of these ways of speaking are ubiquitous in the context of computer science and programming.[3] But to speak both ways is to equivocate. I'd prefer to reserve the term "variable" for the expression in the programming language and then talk about the memory location it is associated with. But since the idea that a "variable" is a memory location is so entrenched we can make a distinction between the variable qua expression and the variable qua memory location.

> **Symbolic Variable**: An expression in a programming language that represents a value by being associated with a memory location containing that value. It is the bit of code that the programmer interacts with to access and manipulate the value stored in memory.[4]
>
> **Storage Variable**: A memory location where the value of a symbolic variable is stored. This may refer to a physical or abstract location, depending on the programming language's implementation. The storage variable is what the machine operates on to store or retrieve the value during execution.

[3]It might be that the former way of speaking is more common in low-level programming languages like C, where the programmer is more concerned with the memory locations and how they are accessed, while the latter way of speaking is more common in high-level programming languages like Python, where the programmer is more concerned with the values and how they are manipulated. But that's just speculation.

[4]These are sometimes called "variable identifiers".

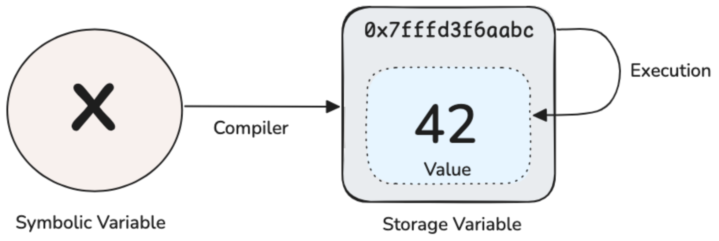

Figure 1: The variable ‘x’ is associated with location `0x7fffd3f6aabc`, which contains the value 42.

Modern programming depends on these two concepts of variables: the symbolic variable that the programmer works with and the storage variable that the machine manipulates. Focusing on the distinction between these two concepts helps clarify the role of “variables” in programming.

0.1 Historical digression

This distinction even exists in the earliest days of computing in connection to the Analytical Engine, designed by Charles Babbage in the 1830s, which is considered the first plans for a general-purpose computer.

Babbage’s Engine introduced the idea of a symbolically accessible, changeable memory location in the form of its “store”. Variables were associated with physical columns on the machine, where each column was essentially a stack of rotatable dials, each corresponding to a different place value in a number. Imagine a giant mechanical lock where each column’s dials represent different place values: one dial for the ones place, another for the tens, hundreds, and so on. By adjusting these dials, the engine could change the number stored in a column. This mechanical design was a precursor to modern computing variables. The columns of rotatable discs have, of course, been replaced by electronic memory cells, where binary digits are used to encode values. Yet, while the physical implementation has evolved dramatically and the abstraction layers involved in programming languages have increased, the underlying concept of associating a variable with a changeable memory location remains foundational.

This was an important advancement, as earlier computing devices did not provide memory locations that could be accessed symbolically and dynamically during computation.[5] The symbolic reference to memory locations and the

[5]Earlier devices like the abacus or the slide rule did not have memory locations that could be accessed symbolically and dynamically during computation. And even early devices such as Leibniz’s stepped reckoner or Pascal’s calculator did not have memory locations that could be accessed symbolically and dynamically during computation.

ability to change and re-use these values were groundbreaking and formed the foundation for modern variables in programming.

In fact, Ada Lovelace's program to calculate Bernoulli numbers in her famous Note G, often considered the first computer program, was an important precursor.[6] By abstracting the Analytical Engine's mechanical operations into symbolic, algorithmically manipulable entities, Lovelace made use of variables ($V_1 \ldots V_{24}$)—symbols representing changeable data. This innovation bridged the gap between raw machine operations and symbolic computation.

Lovelace clearly makes the distinction between the symbolic variables and the actual memory locations in her notes on the Analytical Engine:

> "The V's are for the purpose of convenient reference to any column, either in writing or speaking, and are consequently numbered. The reason why the letter V is chosen for the purpose in preference to any other letter, is because these columns are designated the *Variables*, and sometimes the *Variable columns*, or the *columns of Variables*. The origin of this appellation is, that the values on the columns are destined to change, that is to *vary*, in every conceivable manner.…In order to prevent the possibility of confusion, we have, both in the translation and in the notes, written Variable with a capital letter when we use the word to signify a *column of the engine*, and variable with a small letter when we mean the *variable of a formula*" (Lovelace 1843, Note B)

0.2 The Meaning of a Variable

There are two distinct steps in associating a value with a variable. The first step is to associate a symbolic variable with a storage variable (i.e., a slot in memory). The second step is to associate the storage variable with a value. This distinction is not commonly made in mathematics or logic, where variables (the expressions) are typically assigned a value directly (by the variable assignment). But, perhaps unsurprisingly, it is common in the semantics of programming languages.[7] As R.D. Tennent writes:[8]

> "…it is necessary to structure their semantic models to incorporate both a textually determined environment and a dynamically

[6]Ada Lovelace (1843) "Translator's Notes to M. Menabrea's Memoir", *Scientific Memoirs* 3. Whether or not this was actually the first published computer program is a matter of some debate. Babbage himself clearly wrote "programs" for his Engine, but Lovelace's Note G involved a description that was significantly more organized and complex, e.g., with loops and conditional branching, etc.

[7]Dana Scott and Christopher Strachey (1971), "Toward a mathematical semantics for computer languages", Oxford Programming Research Group Technical Monograph, PRG-6, 1971.

[8]R.D. Tennent (1976), "The denotational semantics of programming languages", Communications of the ACM, 19(8), 437–453.

> changing abstract *store*. This complication arises because it is necessary to distinguish between *identifiers* (program variables, symbolic names, formal parameters) which are syntactic entities, and *locations* (storage variables, references, L-values, addresses, pointers) which are semantic. [...] A semantic model which allows such distinctions to be made is obtained by interpreting expressions and commands relative to both an environment and a store" (Tennent 1976: 444–45)

The variable gets assigned a value in two steps—the compiler does its work, and then the program executes—the first step involves the *variable environment*, while the second step involves the *variable store*.

> **Variable Environment**: A mapping from symbolic variables to storage variables. The variable environment associates each symbolic variable with the storage variable that holds its value.
>
> **Variable Store**: A mapping from storage variables to values. The variable store is the actual memory locations where the values of symbolic variables are stored. It maps each location to a value.

0.3 Russell's antinomy and Dynamic indexing

Russell (1903) suggested that a variable such as 'x' expresses a denoting concept *the variable*, which ambiguously denotes every entity in its range. But he quickly pointed out that "...this can hardly be strictly maintained, for different variables may occur in a proposition"—that is, 'y' can't also express *the variable* or else the distinction between 'Rxy' and 'Rxx' would be lost. The threat is that "variables have a kind of individuality". This tension between the sameness of variables and their distinctness is known as Russell's antinomy of the variable.

Of interest, at least to me, is that the distinction between the variable environment and the variable store bears a striking resemblance to an important feature of a proposed solution to Russell's antinomy. Picking up on a suggestion from Tarski, Pickel and Rabern propose that the enumeration of variables should be dynamic rather than stipulative.[9] They then introduce a dual-parameter system to account for the semantic values of variables. They distinguish between two different parameters in their semantic system: the *discourse context* and the *evaluation sequence*.

The *discourse context* is a dynamic mapping from variables to numbers (positions in sequences). It is initially empty and gets built up as the formula is processed. When quantifiers introduce new variables, they get assigned to

[9] Pickel and Rabern (2016). The Antinomy of the Variable: A Tarskian Resolution. Journal of Philosophy 113 (3):137–170.

new positions. The key innovation is that this context evolves as the sentence is processed. This is clearly analogous to the variable environment in programming languages.

The *evaluation sequence* is an sequence of individuals from the domain. It maps numbers onto objects and is used to give values to variables based on their positions assigned by the discourse context. This is clearly analogous to the variable store in programming languages.

The distinction between the discourse context and the evaluation sequence allows Pickel and Rabern to maintain the semantic difference between distinct variables in a sentence while accommodating the semantic sameness of alphabetic variants. They summarize their solution as follows:

> "The underlying idea is that the semantic contribution of a variable maps a context to a position in a sequence. On the semantics we offer, x and y will be associated with distinct functions from contexts into positions in sequences. Nonetheless, if x and y occur in corresponding positions in sentences that are alphabetic variants, then (in context) they will be correlated with the same position in a sequence. Thus, we offer a sense in which x and y have the same semantic role and a sense in which they don't, thereby resolving the antinomy" (Pickel and Rabern 2016: 155).

This proposal could perhaps gain support from the analogy with the variable environment and the variable store in programming languages.[10]

[10] The quotes in the epigraph are from: Bertrand Russell (1903), *Principles of Mathematics*, W. W. Norton & Company; and E.W. Dijkstra (1972), "Notes on structured programming", in *Structured Programming*, Academic Press, p. 1–82.