A brief history of the stack

Sten Henriksson
Computer Science Department, Lund University, Lund, Sweden.

Since 1968[1], millions of computer science students have taken a course on algorithms and data structures, typically the second course after the initial one introducing programming. One of the basic data structures in such a course is the stack.

The stack has a special place in the emergence of computing as a science, as argued by Michael Mahoney, the pioneer of the history of the theory of computing: "Between 1955 and 1970, a new agenda formed around the theory of automata and formal languages, which increasingly came to be viewed as foundational for the field as a whole".[2] In this process, interest arose in "devices with more generative power than finite automata, and more special structure than Turing machines."[3] The push-down automaton which is based on a stack is such a device.

1. What is a stack?

A *stack*, also known as a *last-in-first-out (LIFO) list* or a *pushdown store,* is a simple memory mechanism usually implemented as an array or as a linked list. By using a formal, axiomatic definition in describing it, we can avoid dependence on the implementation. Let **isempty** be a predicate investigating a stack, let **push** be a function changing a stack by putting a value on it, let **pop** be a function eliminating a value from the stack, let **top** give us the value from the top of the stack without changing it. Finally, let S be a stack and x a value, while **create** is a function that constructs an empty stack. Then:

Axiom 1: **isempty**(**create()**); *A freshly created stack is in fact empty.*

Axiom 2: ¬ **isempty**(**push**(x, S)); *A stack on which something has been put is not empty.*

Axiom 3: **pop**(**push**(x, S) = S; *If we put a value on a stack and then remove it, the stack is unchanged.*

Axiom 4: **top**(**push**(x, S)) = x; *If we put a value on a stack, the value is there when we inspect the stack.*

Axiom 5: ¬**isempty**(S) ⇒ (**push**(**top**(S),**pop**(S))=S); *If a stack is not empty, we may remove the top element and put it back leaving the stack unchanged.*

Looking up the word "stack" in the Oxford English Dictionary[4], we find

> 1. a. A pile, heap or group of things, esp. such a pile or heap with its constituents arranged in an orderly fashion.
> b. fig. A quantity, a 'pile'. Also in pl. and as advb. ellipt., a pile of money. colloq. (orig. U.S.).
> c. to swear on a stack of Bibles (see quot. 1909). U.S. colloq.
> d. A set of shelving on which books are arranged for storage, esp. in a library.
> e. Aeronaut. A series of aeroplanes circling at different altitudes and awaiting landing instructions.
> f. In a computer or calculator, a set of registers or storage locations which store data in such a way that the most recently stored item is the first to be retrieved; also, a list of items so stored, a push-down list.
> 2. A pile of grain in the sheaf, of hay, straw, fodder, etc., gathered into a circular or rectangular form, and usually with a sloping thatched top to protect it from the weather.

3. a. A pile of sticks, faggots, firewood, poles, etc.
b. A pyre or burial pile.
c. A measure of volume for wood and coal, usually 4 cubic yds.
4. Brick-making. = CLAMP
5. a. A number of chimneys, flues, or pipes, standing together in one group.
b. A chimney of a house, factory, etc.; the chimney or funnel of a locomotive or steamship; also, = stack-furnace.
c. In fig. phr. to blow one's stack = to blow one's top {dag}
6. A set (of corn mills).
7. A columnar mass of rock, detached by the agency of water and weather from the main part of a cliff, and rising precipitously out of the sea.
8. attrib. and Comb

ADDITIONS SERIES 1993
stack, n.
Add: [1.] g. A vertical arrangement of public-address or hi-fi equipment.
[5.] [b.] Before 'also' read: a vertical overhead exhaust-pipe on a diesel-powered truck or similar vehicle (slang, orig. U.S.).

The etymology given by OED is

*ON. stakk-r haystack (MSw. stakker, Sw. stack, Da. stak, Norw. dial. stakk):{em}OTeut. type *stakko-z, prob.:{em}pre-Teut. *stogno-s: cf. Russian stog haystack.*

The word seems to be part of the Teutonic legacy and its age may explain why it has come to be utilized in so many contexts. But it is only in the 1 f) description that we find any indication in what way new items are added to or removed from a stack. In a 5 b) stack (=blast furnace), ore and coke is added on the top and molten iron removed from the bottom, making this a queue arrangement, as in the 1 e) case where airplanes are circling at various levels above an airport waiting for their turn to land. For 1a) and 3 a) we have some kind of a pile where new items are added on top and possibly removed from the top as well, but no order is stated. Several of the definitions have in common that only the top element is visible, which may explain why *stack* was used for the computer science stack. *Stack* became the dominating terminology thanks to Dijkstra (see below) but before that *pushdown store*, *LIFO list* and *cellar storage* were used.

2. A reverse history

The importance of the stack in the development of computer science has been explained by Chomsky and Miller in terms of the *push-down storage automaton* (PDA). This is an automaton with one read-only input tape moving in one direction only, and one storage tape, movable in both directions. Symbols are written and read on the storage tape, which is used as a push-down storage/ stack with only one symbol visible at a time. It has been said that "…the theory of push-down automata is, in fact, essentially another version of the theory of context-free grammar"[5] The PDA automaton was introduced by Newell et al in 1959.[6]

The languages defined by (accepted by) the PDA are exactly the context-free languages, while finite automata without a stack define regular languages. If we let a finite automaton have two stacks, it can simulate the tape moves of a Turing machine and thus have the same computational power as a Turing machine. The languages accepted by TM:s are called *recursively enumerable*. If we restrict the TM:s to those who always halt, the corresponding languages are called *recursive*. This is the case corresponding to our notion of a practical algorithm. The trouble here is the ambiguous use of the word recursive. As we know, stacks are used to implement recursive functions, which we informally would say are functions defined in terms of themselves. But recursive is, as demonstrated, also used as a synonym for *decidable*. In the context of Gödel's work there was a need to denote functions simple enough to be represented by recursive functions (in the original sense) which always finish. For an

illuminating discussion see Hopcroft-Ullman.[7] As a final contribution to the terminological difficulties we have the *stack automaton* (SA) which is like a PDA except that it can read its input tape in both directions. Further its push down store is no longer a stack but allows for reading at any place in the interior of the tape. The SA is more capable than the PDA, but not up to the level of a TM.

The property of the PDA to recognize/parse context-free languages of course made it extremely useful in the theory and practice of parsing programming languages. Natural language in human speakers and listeners also involves parsing, e.g. of embedded clauses. Victor H. Yngve, working in computational linguistics in 1960, proposed that a speaker's short term memory functions as a stack with limited depth: successive left branching parts are in turn stored on a stack only to be removed when they are matched as the speaker is uttering their corresponding later parts toward the end of the sentence.[8] However, Miller and Chomsky reject Yngve's hypothesis by discussing the situation when a sentence has nested, self-embedded parts as in

The salmon that the man that the dog chased smoked tasted bad.[9]

This is grammatically correct English and we may speak or understand such a sentence with a conscious effort, but we cannot do it easily. Our difficulties in parsing such a sentence show that our personal stack has a very limited depth, much less than the classic 7±2 result for short-time memory. It is certainly an indication that the stack is not a suitable model for short-time memory.

The source for the first occurrence of the computer science stack in the OED entry is a 1960 paper by E.W. Dijkstra.[10] Dijkstra was a pioneer of compiler construction with the first Algol 60 compiler. Writing about its history, Dijkstra describes his contribution[11]:

> During my 1959 summer holiday in Paterswolde I had given my first thoughts to the question how to implement recursion: in the early months of 1960 we discovered how, in combination with that, to do justice to the scope rules of Algol 60. The definition of Algol makes extensive use of recursive productions. A run-time system for a language like Algol allowing unrestricted procedure calls has to contain some kind of a stack mechanism. If a subroutine/procedure/ method calls itself recursively it is necessary to store return addresses and local variables in such a way that they are not overwritten when the next call of the subroutine is executed. Correspondingly what has been stored must be made available on return from the previous deeper level. Also, stacks are put to use when dealing with the block structure of Algol 60, where local variables in one block have to be stored while we are working within another one. Finally, because Algol is defined using recursion, we can easily translate the productions into a compiler using recursive procedures.

As mentioned, the term recursive, in the sense decidable was used extensively by those working in the aftermath of Gödel's theorems on undecidability. It seems to have returned to computing from this direction, according to John Backus. In a session on the history of Algol 60 he was asked where the metalanguage in the definition of Algol 60 came from and whether it was influenced by linguists like Chomsky. John Backus answered that he had attended a class of Martin Davis[12] where the work of Emile Post and the idea of productions was presented. Backus used these ideas in his 1959 paper on Algol. The metalanguage was further developed and simplified by Peter Naur in the classic Algol 60 report[13] (which is why it is called BNF, Backus-Naur Form).

The importance of the stack for programming languages was stressed by Alan Perlis at the same history conference[14]:

> Algol 60 would have been impossible to adequately process in a reasonable way without the concept of stacks. Though we had stacks before, only in Algol 60 did stacks come to take a central place in the design of processors.

As for the procedure calls, Donald Knuth gives several historic references[15] :

> In 1947 A.M. Turing developed a stack, called Reversion Storage, for use in subroutine linkage. No doubt simple uses of stacks kept in sequential memory locations were common in computer programming from the earliest days, since a stack is such a simple and natural concept. The programming of stacks in linked form appeared first in IPL, as stated above; the name ”stack” stems from IPL terminology (although ”pushdown list” was the more official IPL wording) and it was also independently introduced in Europe by E.W. Dijkstra.

Following Knuth, we find in a 1947 paper by Harry D. Huskey of NBS a description of the situation when a number of subroutines are called upon:

> One approach to the problem of keeping track of position as one drops from one hierarchy of routine to another is by a process called reversion storage, In this method a so-called queue is established which stores in reverse order the addresses one must return to after completion of the respective subroutines in order to proceed with the problem.[16]

So the stack is a queue operating backwards! A footnote says “This approach was developed at NPL See note 1” referring to a group headed by A.M. Turing at the National Physical Laboratory, England. Interestingly enough, Huskey was for a couple of months a visitor to the Dijkstra group in Amsterdam at the time of Dijkstra’s use of a stack in 1960, working on a compiler. Could it have been that he mentioned his 1947 paper?

The language IPL, referred to by Knuth, was used in an interpretative system for list processing by Newell, Simon and Shaw, developed at Rand Corporation in 1956.[17] In list processing, recursion is everywhere. The IPL language has a stack, the use of which has to be explicitly described when a subroutine/procedure is called upon. John McCarthy brought the process to its completion in LISP (1958) with a built-in (sequentially implemented) stack. [18]

3. Algebraic expressions and tax avoidance

The stack was invented independently by F. L. Bauer and K. Samuelson in another context.[19] Working on the fundamentals of mathematical notation, they were interested in the structure of algebraic expressions. The parenthesis-free Polish notation of Lukasiewicz was introduced in 1929 and interest arose in formulating rules for well-formed expressions. Bauer had a continuing interest in mathematical logic since the 1930’s. He wanted to set up a relay realization of a propositional formula and designed a mechanism where intermediate results are stored in the reverse order of their later use. He called the storage principle “Kellar” (cellar). In 1955 he put the principle to use in the analysis of arithmetic formulas, improving the $O(n^2)$ method used by Rutishauser to O(n). In Bauer’s method: “…not only would intermediate results be pushed down, but the operation symbols which are to be postponed would be pushed down as well.” Infix notation could in this way easily be translated to postfix notation. Bauer and his collaborator K. Samelson applied for a patent for the procedure in 1957. The patent was granted much later, too late for being relevant for the advent of handheld calculators allowing infix input.[20] A hardware implementation of a stack was included in 1955 in the plans for the PERM II computer, which however never was built.

Charles.L.Hamblin, Australian philosopher, logician and computer scientist, developed Lukasiewicz’s notation into what is now called “reverse Polish notation” where the operands are placed before the operators. It has the advantage that the operators have the same order as in infix notation, which makes the compiler’s translation into machine instructions much simpler. Hamblin in 1957 independently invented the stack (which he called “a running accumulator”) for this purpose.[21] His ideas were taken up by the British computer manufacturer English Electric and realized in the hardware stack of the KDF9 computer. A few years later the Burroughs B5000 computer became admired for its inclusion of hardware stacks to facilitate the execution of Algol 60 programs. Processors with hardware stacks have been around since then as discussed by Eric LaForest[22]. The VAX computer implemented the CISC instructions *pop* and *push* for the handling of stacks.

It seems that Bauer–Samelson's work with algebraic expressions and a hardware stack to analyze them influenced the language design in which they were involved. The cellar principle was available, hence there would be no problem in extending recursive definitions to a language. "Compiler design and programming language design went hand in hand."[23] Algol 60 became a prime example of this.

But following Knuth, we encounter the use of the stack principle in an earlier and rather unexpected direction:

> Perhaps the first people to recognize that the concepts "stack" (last-in-first-out) and "queue" (first-in-first-out) are important objects of study were cost accountants interested in reducing income tax assessments; for a discussion of the "LIFO" and "FIFO" methods of pricing inventories, see any intermediate accounting textbook, e.g. C.F. and W.J. Schlatter, Cost Accounting (New York, Wiley 1957), Chapter 7.

In accounting, this is a situation where a company has an inventory of identical articles which have been bought or manufactured at different costs. When the value of the inventory is to be declared to the tax authorities, what values should be used? The values for those sold are used in the order from those last bought, i.e. the LIFO method is used. With rising prices/inflation, which has been the most common situation, it means that those still left in the inventory have a lower value, which is advantageous from the point of view of tax reduction. The method was accepted in U.S. tax laws in 1939.

4. The implicit stack

But of course the stack principle is even older. The situation where we have to postpone one action because another one has to be completed is a basic human experience, and can thus be expected to show up in folk literature. So here it is, in a fairy-tale[24].

> **The Old Woman and Her Pig**
>
> An old woman was sweeping her house, and she found a little crooked sixpence. "What," said she, "shall I do with this little sixpence? I will go to market, and buy a little pig."
>
> As she was coming home, she came to a stile: but the piggy wouldn't go over the stile.
>
> She went a little further, and she met a dog. So she said to the dog: "Dog! bite pig; piggy won't go over the stile; and I shan't get home to-night." But the dog wouldn't.
>
> She went a little further, and she met a stick. So she said: "Stick! stick! beat dog! dog won't bite pig; piggy won't get over the stile; and I shan't get home to-night." But the stick wouldn't.
>
> etc. etc.
>
> She went a little further, and she met a cat. So she said: "Cat! cat! kill rat; rat won't gnaw rope; rope won't hang butcher; butcher won't kill ox; ox won't drink water; water won't quench fire; fire won't burn stick; stick won't beat dog; dog won't bite pig; piggy won't get over the stile; and I shan't get home to-night." But the cat said to her, "If you will go to yonder cow, and fetch me a saucer of milk, I will kill the rat." So away went the old woman to the cow.
>
> But the cow said to her: "If you will go to yonder hay-stack, and fetch me a handful of hay, I'll give you the milk." So away went the old woman to the haystack and she brought the hay to the cow.
>
> As soon as the cow had eaten the hay, she gave the old woman the milk; and away she went with it in a saucer to the cat.

> As soon as the cat had lapped up the milk, the cat began to kill the rat; the rat began to gnaw the rope; the rope began to hang the butcher; the butcher began to kill the ox; the ox began to drink the water; the water began to quench the fire; the fire began to burn the stick; the stick began to beat the dog; the dog began to bite the pig; the little pig in a fright jumped over the stile, and so the old woman got home that night.

Each incident of the tale is put on top of a stack and the current stack content is retold after each new addition. The turning point comes when the woman herself goes into action by fetching hay for the cow. [25] In the last paragraph, the stack is emptied and the tale is brought to its conclusion.

A story with such a general structure is called a *cumulative tale*. Tales of this type are found in many cultures. Another well-known Anglo-Saxon one is "This Is the House That Jack Built".[26] Cumulative tales and more generally chain tales (*Kettenmärchen*, German) are the object of a 1929 dissertation by the Finnish scholar Martti Haavio[27]. In his search for the origin and purpose of cumulative tales he found Johan Christopher Wagenseil, a scholar in Königsberg 1699 interested in Jewish-German language and literature. He documented two cumulative songs, “Een Zicklein” and “Eins”. The first one starts with a goat kid which is attacked by a cat, which is bitten by a dog, etc. “Eins” is simply the first in a series of the natural numbers: there is one God, there are two boards (or tables), etc. Both songs are sung in connection with the Jewish Passover celebrations, however not in a religious context but in general merriness and high spirits. There seem to be two main explanations for the texts: one dealing with mystic, kabbalistic interpretations, the other one calling them children songs (“Jewish wisches-wasches suitable for making babies fall asleep “) “Eins” is believed to be of Hebrew origin while the other one is traced to Caldeic (Babylonia).

Haavio classified a number of different patterns in chain tales and he found such tales and songs spread over Africa and India before Western colonization. Unfortunately we do not know the memory implementation of the execution of these cumulative tales. But we do not have to subscribe to any particular theory of human memory to be convinced that our mind has to emulate a stack when such tales are told.

So this is where the search for the origin of the stack ends: the stack is a part of human culture. It s indeed “a simple and natural concept” and we should not be surprised that it has found its way into computers.

Notes and references

[1] Donald Knuth, *The Art of computer programming, vol I, Fundamental algorithms,* Addison-Wesley, New York 1968. Chapter 2 of this book rapidly became the basis for a course in computer science departments and was central in transforming a set of programming techniques into a branch of science.

[2] Michael S. Mahoney , ”What Was the Question? The Origins of the Theory of Computation”. *Using History to Teach Computer Science and Related Disciplines*. Selected Papers from a Workshop, ed. Atsushi Akera and William Aspray (Washington, D.C.: Computer Research Association, 2004), pp. 225-232

[3] Noam Chomsky, "On Certain Formal Properties of Grammars". *Information and Control* 2(1959), 139., after Mahoney ibid.

[4] Oxford English Dictionary, www.oed.com, Nov. 2008.

[5] N. Chomsky and G.A.Miller, ”Introduction to the Formal Analysis of Natural Languages”, *Handbook of Mathematical Psychology Vol II*, p.340, John Wiley and Sons.(1963)

[6] Newell, A., Shaw, J.C., & Simon, H.A. ”Report on a general problem-solving program in Information Processing”, *Proc. International Conference, UNESCO* Paris 1959

[7] John E. Hopcroft, Rajeev Motwani, Jeffery D. Ullman, *Automata Theory, Languages and Computation*..p.385, 3rd Ed. 2007 Pearson – Addison Wesley.

[8] Yngve, V.H. (1960). "A model and an hypothesis for language structure". *Proceedings of the American Philosophical Society, 104*, 444-466.

[9] G.A. Miller and N. Chomsky, "Finitary Models of Language Users" in *Handbook of Mathematical Psychology Vol II*, p.419, John Wiley and Sons.(1963).

[10] E.W. Dijkstra, *Numerische Math*. II. p. 312 (1960).

[11] E.W. Dijkstra, "A programmer's early memories" in *A history of computing in the twentieth century: a collection of essays* / edited by N. Metropolis, New York, Academic Press, 1980.

[12] Martin Davis, b. 1928. Professor Emeritus of New York University, with important contributions to the foundations of mathematics, student of Alonzo Church.

[13] *Revised report on the algorithmic language ALGOL 60* by J.W. Backus, edited by Peter Naur, Copenhagen, 1962.

[14] Presentation by A.J. Perlis p.146 in *History of programming languages, proceedings of the History of programming languages conference, Los Angeles, Calif., June 1-3, 1978, New York, Academic Press, 1981.*

[15] Ibid. Knuth, p. 58.

[16] Harry D. Huskey, "Semiautomatic instruction on the Zephyr," *Proceedings of a second symposium on Large-scale digital calculating machinery,* ,pp. 83-90. Cambridge Mass, Harvard 1949.

[17] IPL-II in A. Newell, C. Shaw and H. Simon, "The Logic Theory Machine", *IRE Transactions on Information Theory IT-2,* p. 61-70.

[18] John McCarthy, "History of LISP" in *History of programming languages II,* New York ACM Press,. 1996, p.178.

[19] F.L. Bauer, "The Cellar Principle of State Transition and Storage Allocation", *Annals of the History of Computing*. Vol 12, No. 1 (1990) Springer International.

[20] US Patent Office, 3,047,228, patented July 31,1962.

[21] C. L. Hamblin [1957]: "Computer Languages." The Australian Journal of Science, 20: 135-139. Reprinted in The Australian Computer Journal, 17(4): 195-198 (November 1985).

[22] Eric LaForest, "Second-Generation Stack Computer Architecture", M. Sc. thesis, http://is.uwaterloo.ca/Eric_LaForest_Thesis.pdf. March 2009.

[23] Ibid. Bauer.

[24] *English fairy tales,* collected by Joseph Jacobs, London, 1890.

[25] This actually makes the tale circular!

[26] The corresponding lyric variant is *the cumulative song*, with examples like "Old McDonald Had a Farm " and "Alouette" . The complexity of songs is analyzed in D. Knuth, "The complexity of songs", CACM Vol. 27 , 4 (1984). In the spirit of Knuth's groundbreaking work, we may investigate the complexity of "The Old Woman and Her Pig". In each paragraph we have a series of increasing linear successions leading to a total length of O (N^2), N being the number of paragraphs. But the tale may be described with a text of size N by just listing the addition of each paragraph plus a rule for their repetitions. Further, we cannot have a complete song with less than the N incidents. Hence the space complexity of a description of the tale is O(N).

[27] Martti Haavio, *Kettenmärchenstudien I*, Helsinki 1929. Haavio (1899 -1973) was a Finnish poet, mythologist and professor of folklore studies at the University of Helsinki 1949-1956.