

Factor (programming language)

Factor is a stack-oriented programming language created by Slava Pestov. Factor is dynamically typed and has automatic memory management, as well as powerful metaprogramming features. The language has a single implementation featuring a self-hosted optimizing compiler and an interactive development environment. The Factor distribution includes a large standard library.

Contents

History

Description

Implementation and libraries

References

External links


History

Slava Pestov created Factor in 2003 as a scripting language for a video game.^[1] The initial implementation, now referred to as JFactor, was implemented in Java and ran on the Java Virtual Machine. Though the early language resembled modern Factor superficially in terms of syntax, the modern language is very different in practical terms and the current implementation is much faster.

The language has changed significantly over time. Originally, Factor programs centered on manipulating Java objects with Java's reflection capabilities. From the beginning, the design philosophy has been to modify the language to suit programs written in it. As the Factor implementation and standard libraries grew more detailed, the need for certain language features became clear, and they were added. JFactor did not have an object system where you could define your own classes, and early versions of native Factor were the same; the language was similar to Scheme in this way. Today, the object system is a central part of Factor. Other important language features such as tuple classes, combinator inlining, macros, user-defined parsing words and the modern vocabulary system were only added in a piecemeal fashion as their utility became clear.

The foreign function interface was present from very early versions to Factor, and an analogous system existed in JFactor. This was chosen over creating a plugin to the C part of the implementation for each external library that Factor should communicate with, and has the benefit of being more declarative, faster to compile and easier to write.

Factor

	
Paradigm	multi-paradigm: <u>functional</u> , <u>concatenative</u> , <u>stack-oriented</u>
Developer	Slava Pestov
First appeared	2003
Stable release	0.98 / July 31, 2018
Typing discipline	<u>strong</u> , <u>dynamic</u>
OS	<u>Windows</u> , <u>macOS</u> , <u>Linux</u>
License	<u>BSD license</u>
Website	<u>factorcode.org</u> (<u>http://factorcode.org/</u>)
Influenced by	
<u>Joy</u> , <u>Forth</u> , <u>Lisp</u> , <u>Self</u>	

The Java implementation initially consisted of just an interpreter, but a compiler to Java bytecode was later added. This compiler only worked on certain procedures. The Java version of Factor was replaced by a version written in C and Factor. Initially, this consisted of just an interpreter, but the interpreter was replaced by two compilers, used in different situations. Over time, the Factor implementation has grown significantly faster.^[2]

Description

Factor is a dynamically typed, functional and object-oriented programming language. Code is structured around small procedures, called words. In typical code, these are 1–3 lines long, and a procedure more than 7 lines long is very rare. Something that would idiomatically be expressed with one procedure in another programming language would be written as several words in Factor.^[3]

Each word takes a fixed number of arguments and has a fixed number of return values. Arguments to words are passed on a data stack, using reverse Polish notation. The stack is used just to organize calls to words, and not as a datastructure. The stack in Factor is used in a similar way to the stack in Forth; for this, they are both considered stack languages. For example, below is a snippet of code that prints out "hello world" to the current output stream:

```
"hello world" print
```

`print` is a word in the `io` vocabulary that takes a string from the stack and returns nothing. It prints the string to the current output stream (by default, the terminal or the graphical listener).^[3]

The factorial function $n!$ can be implemented in Factor in the following way:

```
: factorial ( n -- n! ) dup 1 > [ [1,b] product ] [ drop 1 ] if
```

Not all data has to be passed around only with the stack. Lexically scoped local variables let you store and access temporaries used within a procedure. Dynamically scoped variables are used to pass things between procedure calls without using the stack. For example, the current input and output streams are stored in dynamically scoped variables.^[3]

Factor emphasizes flexibility and the ability to extend the language.^[3] There is a system for macros, as well as for arbitrary extension of Factor syntax. Factor's syntax is often extended to allow for new types of word definitions and new types of literals for data structures. It is also used in the XML library to provide literal syntax for generating XML. For example, the following word takes a string and produces an XML document object which is an HTML document emphasizing the string:

```
: make-html ( string -- xml )
  dup
  <XML
    <html>
      <head><title><-></title></head>
      <body><h1><-></h1></body>
    </html>
  XML> ;
```

The word `dup` duplicates the top item on the stack. The `<->` stands for filling in that part of the XML document with an item from the stack.

Implementation and libraries

Factor includes a large standard library, written entirely in the language. These include

- A cross-platform GUI toolkit, built on top of OpenGL and various windowing systems, used for the development environment.^[4]
- Bindings to several database libraries, including PostgreSQL and SQLite.^[5]
- An HTTP server and client, with the Furnace web framework.^[6]
- Efficient homogeneous arrays of integers, floats and C structs.^[7]
- A library implementing regular expressions, generating machine code to do the matching.^[8]

A foreign function interface is built into Factor, allowing for communication with C, Objective-C and Fortran programs. There is also support for executing and communicating with shaders written in GLSL.^{[3][9]}

Factor is implemented in Factor and C++. It was originally bootstrapped from an earlier Java implementation. Today, the parser and the optimizing compiler are written in the language. Certain basic parts of the language are implemented in C++ such as the garbage collector and certain primitives.

Factor uses an image-based model, analogous to many Smalltalk implementations, where compiled code and data are stored in an image.^[10] To compile a program, the program is loaded into an image and the image is saved. A special tool assists in the process of creating a minimal image to run a particular program, packaging the result into something that can be deployed as a standalone application.^{[3][11]}

The Factor compiler implements many advanced optimizations and has been used as a target for research in new optimization techniques.^{[3][12]}

References

1. Pestov, Slava. "Slava Pestov's corner of the web" (<http://factorcode.org/slava/>).
2. "Concatenative.org wiki: Factor/Implementation History" (<http://concatenative.org/wiki/view/Factor/Implementation%20history>).
3. Pestov, Sviatoslav; Ehrenberg, Daniel (2010). "Factor: a dynamic stack-based programming language". *ACM SIGPLAN Notices*. ACM. **45** (12): 43–58. doi:10.1145/1899661.1869637 (<https://doi.org/10.1145%2F1899661.1869637>).
4. Pestov, Slava. "Factor documentation: UI framework" (<http://docs.factorcode.org/content/article-ui.html>).
5. Coleman, Doug. "Factor documentation: Database library" (<http://docs.factorcode.org/content/article-db.html>).
6. Pestov, Slava. "Factor documentation: HTTP server" (<http://docs.factorcode.org/content/article-http.server.html>).
7. Pestov, Slava. "Factor documentation: Specialized arrays" (<http://docs.factorcode.org/content/article-specialized-arrays.html>).
8. Coleman, Doug; Ehrenberg, Daniel. "Factor documentation: Regular expressions" (<http://docs.factorcode.org/content/article-regexp.html>).
9. Pestov, Slava (28 July 2010). "Overhauling Factor's C library interface" (<http://factor-language.blogspot.com/2010/07/overhauling-factors-c-library-interface.html>).
10. Pestov, Slava (10 January 2010). "Factor's bootstrap process explained" (<http://factor-language.blogspot.com/2010/01/factors-bootstrap-process-explained.html>).

11. Pestov, Slava (5 July 2008). "On shaking trees" (<http://factor-language.blogspot.com/2008/07/on-shaking-trees.html>).
12. Ehrenberg, Daniel (2010). "Closure elimination as constant propagation" (<https://web.archive.org/web/20110726044425/http://factorcode.org/littledan/abstract.pdf>) (PDF). Archived from the original (<http://factorcode.org/littledan/abstract.pdf>) (PDF) on 2011-07-26.

External links

- [Official website \(http://factorcode.org\)](http://factorcode.org)
 - Slava Pestov (October 27, 2008). *Factor: An Extensible Interactive Language* (https://www.youtube.com/watch?v=f_0QlhYIS8g) (flv) (Tech talk). [Google](#).
 - Zed Shaw (2008). *The ACL is Dead* (<https://vimeo.com/2723800>) (flv) (CUSEC 2008). CUSEC. – a presentation written in Factor which mentions and praises Factor
-

Retrieved from "[https://en.wikipedia.org/w/index.php?title=Factor_\(programming_language\)&oldid=1035390575](https://en.wikipedia.org/w/index.php?title=Factor_(programming_language)&oldid=1035390575)"

This page was last edited on 25 July 2021, at 11:13 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.