Joy (programming language)

The **Joy programming language** in computer science is a purely functional programming language that was produced by Manfred von Thun of La Trobe University in Melbourne, Australia. Joy is based on composition of functions rather than lambda calculus. It has turned out to have many similarities to Forth, due not to design but to a sort of parallel evolution and convergence. It was also inspired by the function-level programming style of John Backus's FP.[1]

Joy	
Paradigm	multi-paradigm: functional, concatenative, stack-oriented
Designed by	Manfred von Thun
Developer	Manfred von Thun John Cowan
First appeared	2001
Stable release	March 17, 2003 / March 17, 2003
Typing discipline	strong, dynamic
Major implementations	
Joy0, Joy1, "Current Joy", "John Cowan's Joy", "JoyJ (Joy in jvmm)"	
Influenced by	
Scheme, FP, Forth	
Influenced	
Factor, Cat, V, Trith	

Contents

How It Works

Joy is unusual (except for function-level programming languages and some esoteric ones, such as unlambda) in its lack of a lambda operator, and therefore lack of formal parameters. To illustrate this with a common example, here is how the square function might be defined in an imperative programming language (C):

```c
int square(int x)
{
    return x * x;
}
```

The variable x is a parameter which is replaced by the argument to be squared when the function is called.

In a functional language (Scheme), the same function could be defined:

```scheme
(define square
  (lambda (x)
    (* x x)))
```

This is different in many ways, but it still uses the parameter x in the same way.

In Joy, the square function is defined:

```
DEFINE square == dup * .
```

In Joy, everything is a function that takes a stack as an argument and returns a stack as a result. For instance, the numeral '5' does not represent an integer constant, but instead a short program that pushes the number 5 onto the stack.

- The **dup** operator simply duplicates the top element of the stack by pushing a copy of it.
- The ***** operator pops two numbers off the stack and pushes their product.

So the square function makes a copy of the top element, and then multiplies the two top elements of the stack, leaving the square of the original top element at the top of the stack, with no need for a formal parameter. This makes Joy concise, as illustrated by this definition of quicksort:

```
DEFINE qsort ==
  [small]
  []
  [uncons [>] split]
  [enconcat]
  binrec.
```

"binrec" is one of Joy's many recursive combinators, implementing binary recursion. It expects four quoted programs on top of the stack which represent:

- the termination condition (if a list is "small" (1 or 0 elements) it is already sorted),
- what to do if the termination condition is met (in this case nothing),
- what to do by default (split the list into two halves by comparing each element with the pivot), and finally
- what to do at the end (insert the pivot between the two sorted halves).

Mathematical purity

In Joy, the meaning function is a homomorphism from the syntactic monoid onto the semantic monoid. That is, the syntactic relation of concatenation of symbols maps directly onto the semantic relation of composition of functions. It is a homomorphism rather than an isomorphism, because it is onto but not one-to-one; that is, no symbol has more than one meaning, but some sequences of symbols have the same meaning (e.g. "dup +" and "2 *").

Joy is a concatenative programming language: "The concatenation of two programs denotes the composition of the functions denoted by the two programs".[2]

Its library routines mirror those of ISO C, though the current implementation is not easily extensible with functions written in C.

See also

- RPL
- Concatenative programming language

References

1. Manfred von Thun (December 12, 2003). "A Conversation with Manfred von Thun" (http://www.nsl.com/papers/interview.htm). Retrieved May 31, 2013. " In the early 1980s I came across the famous Backus paper "Can programming be liberated from the von Neumann style," and I was immediately intrigued by the higher level of programming in his FP."
2. "Mathematical Foundations of Joy" (https://web.archive.org/web/20111007025556/http://www.latrobe.edu.au/phimvt/joy/j02maf.html). Archived from the original (http://www.latrobe.edu.au/phimvt/joy/j02maf.html) on October 7, 2011.

External links

- Official Joy Programming Language Website (La Trobe University) (http://www.latrobe.edu.au/humanities/research/research-projects/past-projects/joy-programming-language)
- Joy homepage mirror (http://www.kevinalbrecht.com/code/joy-mirror/index.html)
- Compiled informative collection on Joy (http://joy-lang.org/)
- immediately executable Joy (https://github.com/Wodan58/Joy) (GitHub-Archiv)
- Freneger, Paul (August 2003). "The JOY of forth" (http://portal.acm.org/citation.cfm?id=944579.944583). *ACM SIGPLAN Notices*. **38** (8).
- von Thun, Manfred; Thomas, Reuben (October 9, 2001). "Joy: Forth's Functional Cousin" (http://www.complang.tuwien.ac.at/anton/euroforth/ef01/thomas01a.pdf) (PDF). *Proceedings of the 17th EuroForth Conference*.
- Christopher Diggins (December 31, 2008). "What is a Concatenative Language" (http://www.drdobbs.com/architecture-and-design/228701299). Dr. Dobbs.
- Apter, Stevan. "Functional Programming in Joy and K" (https://web.archive.org/web/20080828115345/http://www.vector.org.uk/archive/v214/joy214.htm). *Vector*. Archived from the original (http://www.vector.org.uk/archive/v214/joy214.htm) on 2008-08-28. Retrieved 2011-02-28.
- mjoy, an interpreter in Delphi for machine drawing (https://github.com/Fpstefan/mjoy) (Subset of Joy)