Macros: Menace to the Modern HLL

Jeremy H. Brown
jhbrown@alum.mit.edu

All Else
nipulate the data

Macros Considered Harmful

Jeremy H. Brown
jhbrown@alum.mit.edu

Macros: They're for Nazis

Jeremy H. Brown
jhbrown@alum.mit.edu

Macros are for Dirty Hippies

Jeremy H. Brown
jhbrown@alum.mit.edu

Macros are Just Like Goto

Jeremy H. Brown
jhbrown@alum.mit.edu

GOTO is Good, Right?

- Gurus can do amazing things with GOTO
- If, switch, for, and while aren't enough
- No language provides all control-flow patterns
- Good programmers won't make mistakes very often
- Coding guidelines make it safe
- GOTO makes up for deficiencies in base language!

[redacted]

You love C++ Parameter Passing

```
int x;
foo (x);  // int foo(int &arg) ?
```

[redacted]

Static Syntax: Feel the Love

Recently, I found myself needing to deal with a "convenience" macro, which quoted several of its arguments for me before passing them along to the real function. Unfortunately, only the macro was exported from the library, and I was unable to access the base function.

```
(define-syntax convenient-function
  (syntax-rules ()
    ((_ arg1 arg2) (much-harder-function 'arg1
'arg2))))
```

How useful. To save me a handful of quotes, I lose the ability to programmatically generate my arguments.

-- Will Donnelly, willdonnelly.wordpress.com
"Fixing broken macros with eval and quasiquote"

[redacted]

EVAL-WHEN EXPLAINED

...the only prerequisite to understanding EVAL-WHEN is an understanding of how the two functions LOAD and COMPILE-FILE interact... There are three possible *situations*... To explain the meaning of the three situations, I'll need to explain a bit about how COMPILE-FILE... goes about compiling a file. To explain how COMPILE-FILE compiles EVAL-WHEN forms, I need to introduce a distinction between compiling top-level forms and compiling non-top-level forms... There are two ways you're most likely to use EVAL-WHEN. One is if you want to write macros that need to save some information at compile time... The other time... is if you want to put the definition of a macro and helper functions it uses in the same file as code that uses the macro.

--- Peter Siebel, *Practical Common Lisp*

[redacted]

You Shouldn’t Need ‘em

- For “derived” constructs in language core, use the compiler
- Replace common language “extensions” with reusable idioms and interfaces (e.g. Python: for+iterators, with+context guards)
- For embedded DSLs, use distinctive syntax and write a tiny compiler/interpreter
- For non-embedded DSLs, write a compiler
- Don’t force readers to learn a new language!

Precisely Stated

Macros encourage people who are not good at language design to do something equivalent to language design, using tools that don't help, and with effects that are too powerful. This makes code unreadable to people joining later and for the authors after time has passed. Well designed macros are well documented, but this doesn't happen much.

-- Richard P. Gabriel,
personal communication

Imprecisely Stated

Q: What do you get when amateurs do language design?

A: PHP.

[redacted]

SQL Embedding

- Bad:

```
(select * from addresses where city = (get-city o))
```

  - unheralded macro-driven transition
  - "regular" lisp evaluation at irregular points
  - not programmatically composable

- Better:

```
(sql-query
 `(select * from addresses where city = ,(get-city o)))
```