Dramatic Before after @ LLTV

Real Macro Metaprogramming on C

Genuine macr

o breakthrough
abstraction
limit

(false macro
of! cpp)

Subject

File	LOC
ls.h	38
extern.h	twenty one
ls.c	526
cmp.c	73
print.c	319
util.c	164
Total	1141

- Anyone knows the ls (1) command

- Source is (Free, no-dependency) from FreeBSD

- A straightforward and easy-to-read code

- Honors **as a C language source**

However…

Three

problems

Problem 1 - Redundant code

cmp.c

```
 int namecmp (const
FTSENT * a, const
FTSENT * b)
```

```
 {
     return (strcoll
(a -> fts_name, b ->
fts_name));
 }

 int revnamecmp
(const FTSENT * a,
const FTSENT * b)
 {
     return (strcoll
(b -> fts_name, a ->
fts_name));
 }
```

Problem 1 -

Redundant code (cont' d)

`util.c`

- Four functions of the same structure `while ((clen = mbrtowc (& wc, ...))! = 0) {`
- `if (clen == (size_t) -1) {...}`
- `else if (clen == (size_t) -2)) {...}`

```
    if (iswprint
  (wc)) {...}
    else {...}
 }
```

- In C, it is difficult to enclose common structure
  - `It` is necessary to refer to the outside environment from within the `while`
  - But I can not use closures!

Problem 2 -

Nonessential information

```
 int i;
    :
 for (i = 0; i
<(int) clen; i ++)
     putchar
((unsigned char) s
[i]);
```

I wish I could write `for (int i = [0..clen]) {putchar (...)}`

```
 FTSENT * p;
     :
 for (p = dp ->
list; p; p = p ->
fts_link) {
     :
 }
```

I wish I could write `foreach (FTSENT * p in dp -> list) {...}`

Problem 3 - Distribution

of information

```
ls.h

 extern int
f_accesstime; / *
use time of last
access * /
ls. c (decl)

 int f_accesstime; /
* use time of last
access * /
ls. c (main)
```

```
 case 'c':
      f_statustime =
1;
      f_accesstime =
0;
 case 'u':
      f_accesstime =
1;
      f_statustime =
0;
```

Takumi's policy

The point of renovation: The

syntax is decorative

- `printf ("% d:% s \ n", n, msg);`
- `(printf "% d:% s \ n" n msg)`
- `if (is_foo (x)) {do_foo (x);} else {do_bar (x);`
- `(if (is _ foo x) (do _ foo x) (do_bar x))`

CiSE (C in S-

Expression)

```
 int
 main (int argc,
char * argv [])
 {
      static char dot
[] = ".", * dotav []
= {dot, NULL};
      struct winsize
win;
      int ch,
fts_options,
notused;
      char * p;
```

```
      (void)
setlocale (LC_ALL,
"");
      if (isatty
(STDOUT_FILENO)) {
            termwidth =
80;
            if ((p =
getenv ("COLUMNS"))!
= NULL && * p! = '\
0')

termwidth = atoi
(p);
            else if
```

```
(ioctl
(STDOUT_FILENO,
TIOCGWINSZ, & win)!
= -1 &&

win.ws_col> 0)

termwidth =
win.ws_col;
          f_nonprint
= 1;
```

CiSE (C in S-Expression)

```
 (define-cfn main
(argc :: int argv ::
char **) :: int
   (setlocale LC_ALL
"")
   (cond [(isatty
STDOUT_FILENO)
          (=
termwidth 80)
          (let *
([p :: char *
(getenv "COLUMNS")]]

[win: :( struct
winsize)])
              (cond
```

```
[(and p (! = (* p) #
\ null)) (=
termwidth (atoi p))]

[(and (! = (ioctl
STDOUT_FILENO
TIOCGWINSZ (& win))
-1)

(> (ref win ws _
color 0))

(= termwidth (ref
win ws _ color))])
                (=
f_nonprint 1))]
```

CiSE (C in S-Expression) (cont' d)

- The C ABI, semantics remain intact
  - You can use the runtime as it is
  - on the bare metal
  - Just a slight change in appearance
- You can use genuine macros
  - Any source code

conversion possible
before compiling

- Unofficial support at Gauche

macro

Syntax tree substitution

```
 (define - cise -
stmt (when test.
body)
   `(if, test
(begin, @ body)))

   :
```

```
 (when (is-- foo x)
(do-- this) (do--
that))
     ↓
 (if (is _ foo x)
(begin (do_this)
(do_that)))

 ; if (is_foo (x))
{do_this (); do_that
();}
```

Macro

(cont'd)

Pattern extraction / concealment

```
 (dotimes [i (strlen
s)] (printf "%
02x" (aref si)))
    ↓
 (let * ([i :: int
0] [cise__ 213 ::
int (strlen s)])
   (for [() (<i
cise__ 213) (inc!
i)]
```

```
      (printf "%
02x" (aref si)))))

 ;; {
 int i = 0; int
cise__ 213 = strlen
(s);
 ; for (; i <cise__
213; i ++) {
 ;; printf ("% 02x",
s [i]);
 ;;}
 ;;}
```

Macro

(cont'd)

Global compile time calculation

```
 (define-enum
FooMode (MODE_X
MODE_Y))
    :
 (gen-printer
FooMode)
```

↓

```
 enum FooMode
{MODE_X, MODE_Y};
```

```
    :
 void FooMode_print
(enum FooMode m)
 {
    switch (m) {
      case MODE_X:
puts ("MODE_X");
break;
      case MODE_Y:
puts ("MODE_Y");
break;
    }
 }
```

Modification 1

- removal of repeating pattern

cmp.sc

```
 (define-cmpfn
namecmp
   (return (strcoll
(-> a fts_name) (->
b fts _ name)))))
 (define-cmpfn-stat
modcmp st_mtime)
```

```
 (define-cmpfn-stat
acccmp st_atime)
 (define-cmpfn-stat
statcmp st_ctime)
 (define - cmpfn -
stat sizecmp st -
size)
```

Modification 1 - Removal of repeating pattern (cont'

d)

```
 FTSENT * p;
     :
 for (p = dp ->
list; p; p = p ->
fts_link) {
     ...
 }
```

↓

```
 (do-ftsent [p (->
dp list)] ...)
```

Modification 2

- Lifting essence

```
util.sc

 (define-cfn
prn_normal (s: :
( const char *)) ::
int
   (let * ([n :: int
0])
      (make-printer
       (putchar (cast
(unsigned char) (*
```

```
s))); ilseq (clen ==
-1)
        (+ = n (printf
"% s" s));
incomplete (clen ==
- 2)
        (default-
print); nonprintable
        (begin
(default-print) (+ =
n (wcwidth wc))));
printable
      (return n)))
```

Original C function: 29 LOC

Modification 2 - Start of essence (cont'd)

util.sc

```
 (define-cfn
prn_octal (s: :
( const char *)) ::
int
    (let * ([len ::
```

```
int 0]
            [esc: :(.
array (static const
char) (())) "\ \ \ \
\" \ "\ aa \ bb \ f
n \ rr \ tt \ vv"])
      (make-printer
       (octal-print
1); ilseq (clen ==
-1)
       (octal-print
(strlen s));
incomplete (clen ==
-2)
       (esc-print);
nonprint
```

```
        (if (and (! =
wc (cast wchar_t # \
")); print
                    (! =
wc (cast wchar_t # \
\))))
          (begin
(default-print) (+ =
len (wcwidth wc)))
          (esc-
print)))
      (return len)))
```

Original C function: 50 LOC

Modification 3

- DSL

```
 (define-flag int
f_accesstime "u"
"cU"); use time of
last access
 (define-flag int
f_statustime "c"
"uU"); use time of
last mode change
 (define-flag int
f_longform "l" "1
Cxm"); long listing
format
 (define-flag int
```

```
f_nonprint "q"
"Bbw"); show
unprintables as?
```

- Generate variable definition on the spot
- Generate option handler in option parser
- Generate variable declaration in another file
- 13 lines of macro code

Comparison

	Before	After
cmp.c	73	15
ls.c	526	277
print.c	319	183
util.c	164	73
Total	1082	548

- The code is about half (effect is larger if the original code is bigger)
- Aggregate meanings → easy to add functions of the same line
- Pattern reusability

CAUTION

!!Caution!!

- It is not a story that "CiSE / Macro can be used"

- Macro is a powerful medicine → Be familiar with usage
- CiSE is an experimental stage

CAUTION (cont' d)

`` Macro Club has two rules, plus one exception ''

1. Do not write macros.

2. If it's the only way to encapsulate the pattern, write a macro
3. Compared to equivalent functions, if the caller becomes easy, you can write macros

Stuart Halloway ("Programming Clojure")

Message from the Takumi

You are a programmer, rather than being used for a versatile god language, play with the language using

language