I almost compiled it, so I compiled it. Factor and image were finished successfully but it did not move on X. When I checked the libraries taken in ldd, I used as many as 75 surprising fire foxes. There were only 9 people on the Arch side. X relationship is largely missing. I put a library that is necessary compared with that of Ubu. In other words, I do not know the package name from the library name, so I used a search package called pkgfile. Cuddy, search data must be imported in advance with pkgfile -u though.

```
[sakae @ arch ~] $ pkgfile -s libgdkglext-x11-1.0.so.0
extra / gtkglext
[sakae @ arch ~] $ pkgfile -s libpangox-1.0.so.0
extra / pangox-compat
```

With this way, when I put various X relationships, I started working on X as well. Whew.

```
[sakae @ manjaro ~] $ sudo pkgfile -u
:: Updating 5 repos ...
warning: download failed: http://spiralinear.org/manjaro/repo/basis/i686/basis.files [HTTP 404] err
warning: download failed: http://spiralinear.org/manjaro/repo/addon/i686/addon.files [HTTP 404] err
warning: download failed: http://spiralinear.org/manjaro/repo/extra/i686/extra.files [HTTP 404] err
warning: download failed: http://spiralinear.org/manjaro/repo/community/i686/community.files [HTTP
warning: download failed: http://spiralinear.org/manjaro/repo/platform/i686/platform.files [HTTP 40
```

The branch manjaro, unfortunately, was not attentive enough, and no source for pkgfile was offered. Is it better to report this for the future?

CiSE

Previously, I pulled the article again to use gauche's engine from C. Next time, there was an article translating the Scheme S expression into the C language, so I will try again.

It is said to be CiSE (C In S - Expression). Since it is easy to use that part is not written also in the manual, there are tacit acknowledgment everywhere.

The converter is a module, so define the driver. (Ah, I write it so, but it is a quote.)

```
[sakae @ secd ~ / t] $ cat toC.scm
(use gauche.cgen)
(use gauche.cgen.cise)
(use gauche.parseopt)

(define (main args)
  (let-args (cdr args)
    ((infile "i = s" #f)
     (outfile "o = s" # f))

    (unless (and infile outfile)
      (display # `" usage: gosh, (car args) - i 'input - file' - o 'output - file' \ n ")
      (exit -1))

    (call-with-input-file infile
       (^ (in)
          (call-with-output-file outfile
             (^ (out)
                (cise-translate in out)))))))
```

This is the famous FizzBuzz 's gauche dialect.

```
[sakae @ secd ~ / t] $ cat FizzBuzz.scm
(.include <stdio.h>)
```

```
(define-cfn main (argc :: int argv :: char **) :: int
  (dotimes (i 30)
    (case (% (+ i 1) 15)
      ((0) (printf "FizzBuzz \ n")))
      ((369 12) (printf "Fizz \ n")))
      ((5 10) (printf "Buzz \ n")))
      (else (printf "% d \ n" (+ i 1))))))
  (return 0))
```

We will use this as below.

```
[sakae @ secd ~ / t] $ gosh toC.scm - i FizzBuzz.scm - o test.c
[sakae @ secd ~ / t] $ gcc test.c
[sakae @ secd ~ / t] $ ./a.out
1
2
Fizz
Four
:
29
FizzBuzz
```

Well, normally, this is a million years old, but I was looking at test.c because I was deeply sorry I was converted to C source. Then a lot of # line what was done was included. Is it for debug? Because I can not stand a little watching, I tried to shape it. (Note: #include statements have been deleted)

```
[sakae @ secd ~ / t] $ fgrep -v '#' test.c | indent

int
main (int argc, char ** argv)
{{
                {
                        int i = 0;
                        int cise__ 702 = 30;
                        for (; (i) <(cise__ 702); (i) ++) {
                                switch (((i) + (1))% (15)) {
                                case 0: {
                                                printf ("FizzBuzz \ n");
                                                break;
                                        }
                                case 3:
                                case 6:
                                case 9:
                                case 12: {
                                                printf ("Fizz \ n");
                                                break;
                                        }
                                case 5:
                                case 10: {
                                                printf ("Buzz \ n");
                                                break;
                                } default: {
                                                printf ("% d \ n", (i) + (1));
                                                break;
                                        }
                                }
                        }
                }
                return (0);
}
}
```

If you hide it in the shell file including the above driver, the application will be completed from the Scheme file suddenly. I wonder if the name is goshc.

Inside of cgen.cise

In the above, where is the heart of toC.scm, cge.cise? I'd like to see the source. It must be worth more than getting three sentences to watch.

Well, the source of the source is FreeBSD,

```
[sakae @ secd /usr/local/share/gauche-0.9/0.9.3.3/lib/gauche/cgen]$ ls
cise.scm precomp.scm tmodule.scm unit.scm
literal.scm stub.scm type.scm
```

I wonder if cise.scm of this will be a front end to associate with users. Other than that, it belongs to people gathering on github. I will look forward.

```
;; If true, include # line directive in the output.
(define cise-emit-source-line (make-parameter # t))
```

Hmm, this is the switch that will issue / not show out what you translated for debug. By default, it is supposed to come out.

```
;;
;; cise-translate inp outp & key enviroment
;;
;; External interface to translate entire CiSE file into C.
;; CiSE expressions are read from INP and the resulting C code
;; is written to OUTP.
;;
;; If CISE-TRANSLATE encounters a form (.static-decls),
;; it expands the rest of CiSE into into temporary string,
;; then emits the forward declarations of static functions
;; into outp, followed by the accumulated C code. With this
;; you do not need to write forward declarations in CiSE source.

(define (cise-translate inp outp
                        : key (environment (make-module # f))
                             (context (cise-context-copy (cise-context)))))
      :
```

Here, I will celebrate your honorable guest. And its merit,

```
;; ============================================================ =============
;; Built-in macros
;;

;; ------------------------------------------------ --------------
;; C function definition
;;
(define-cise-macro (define-cfn form env)
  (define (argchk args)
    (match (canonicalize-vardecl args)
      [() '()]
      [((var ':: type). rest) `((, var., type), @ (argchk rest))]
      [(var. rest) `((, var. ScmObj), @ (argchk rest))]))
   :
```

It appears in this way. Oh, thankfully, thankful.