TR-AC-0037

C07

プログラムソースコードのコメント操作ツール
（FD付）

新上 和正　下川 信祐　北川 美宏 (ATR-I)

2000. 1.20

ATR環境適応通信研究所

目次

1. はじめに

まったくコメント文のないプログラムソースコードでは可読性に難がありますが、実際のソースコード編集においては、処理や計算の流れを追うのに大量のコメント文が目障りになる場面もあります。特に他人の書いたソースコードを改編する場合には、オリジナルのコメント文に加え、オリジナルのソースコードもコメント化することが多々あり、エディタのウィンドウ一面がほとんどコメント文になってしまうことさえあります。

今回、このような場面での利便性を考えて、プログラムソースコードにおけるコメント文の操作ツールを試作しました。本ツールでは、GNU Emacs 系のエディタ環境を前提に、Fortran ソースコードのコメント操作機能を実現しています。

本ツールは、Emacs 用の Lisp コマンドとしてインプリメントされていますので、Emacs 系のエディタをお使いの方には簡単にご利用いただけます。本ツールを開発したグループ内でも有効に活用されており、Fortran ソースコード編集の生産性向上に寄与しています。

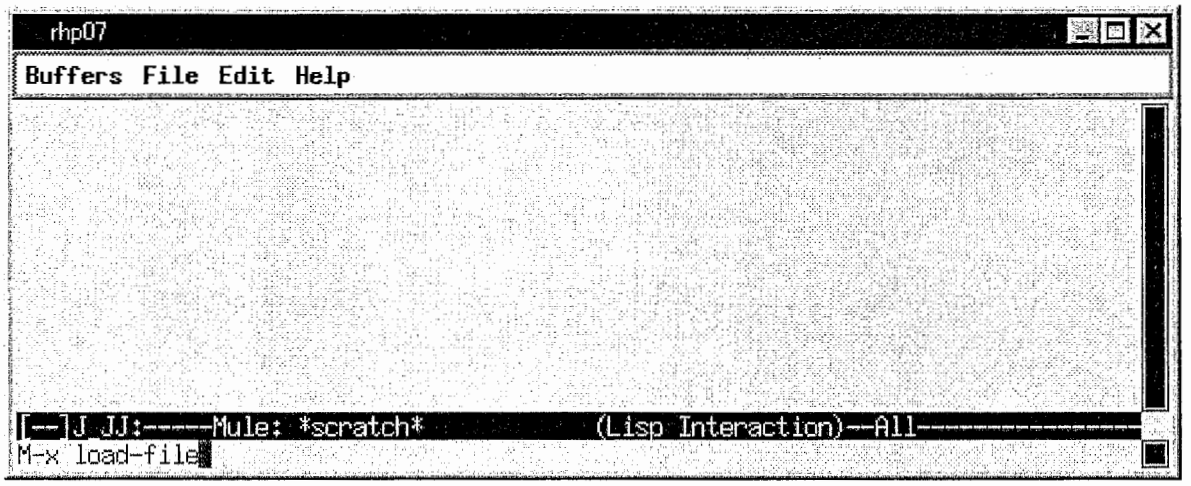
2. 使用法

2.1. 準備

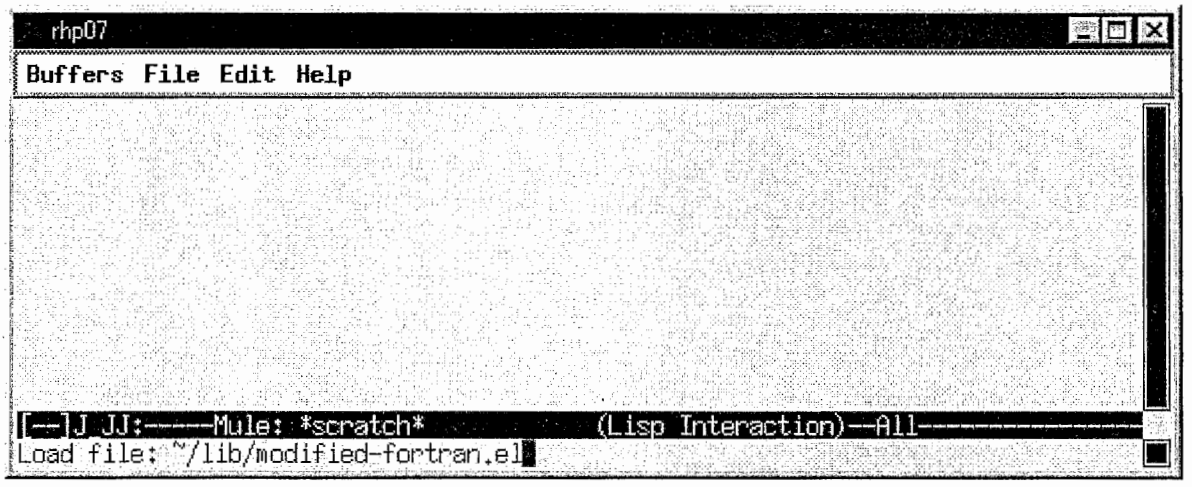
本ツールは、Emacs コマンド形式のLisp プログラム（modified-fortran.el）です。

適当なディレクトリにmodified-fortran.el を置き、Emacs 等からファイルをロードしてください。以下では、ホームディレクトリ内のlib/ディレクトリにファイルがあるものとします。

(1) Emacs にてコマンドload-file を実行します。



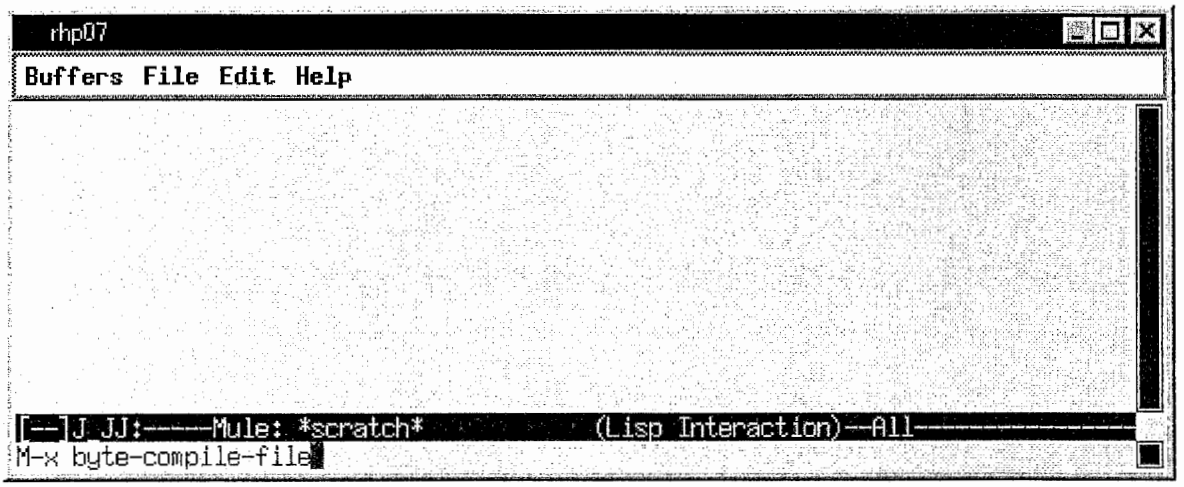
(2) ファイル名 ~/lib/modified-fortran.el を入力します。



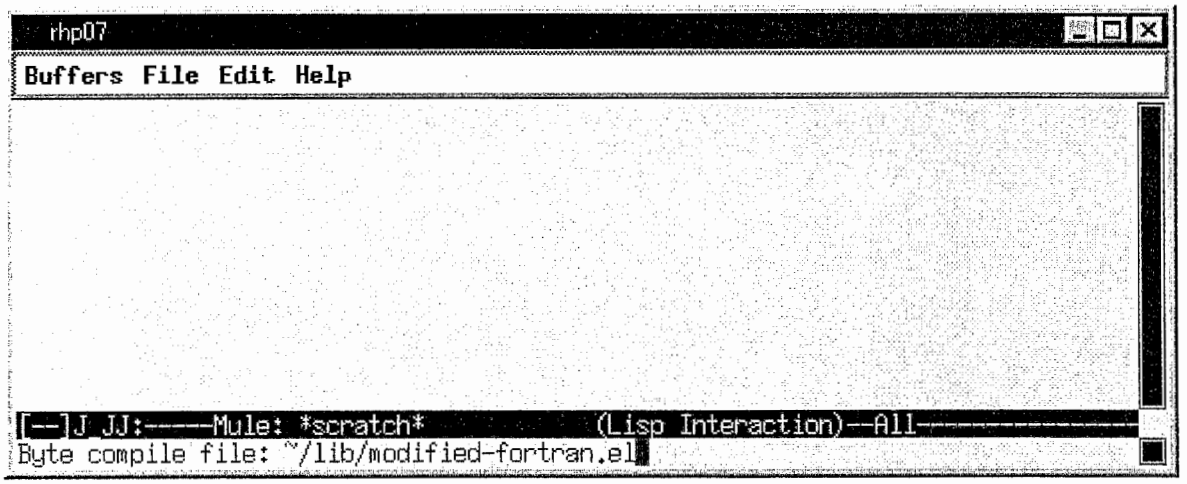
以上でFortran ソースコードのコメント文を表示／非表示するコマンドが使用可能になります。

頻繁に使用される方には、コマンドの実行速度を向上させるため、コマンドのバイトコンパイルをお勧めします。また、.emacs 等の初期設定ファイルにロードの設定をしておくと便利です。

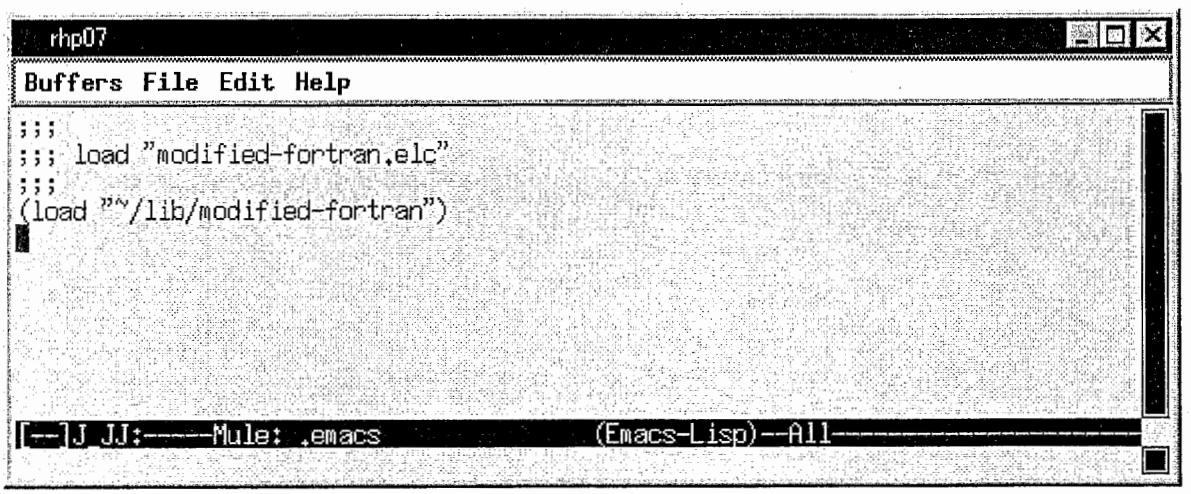
(1) Emacs にてコマンドbyte-compile-file を実行します。



(2) ファイル名 ~/lib/modified-fortran.el を入力します。



(3) コンパイルが終了すると、ファイル ~/lib/modified-fortran.elc が出力されます。これを Emacs 起動時にロードするよう~/.emacs ファイル等に設定します。



以上により、Emacs 起動時にファイル ~/lib/modified-fortran.elc が読み込まれ、バイトコンパイルされたコマンドが使用可能になります。

2.2. 使用法

Fortran ソースコードのコメント文を操作するコマンドとして、4つの関数が、modified-fortran.elに定義されています。

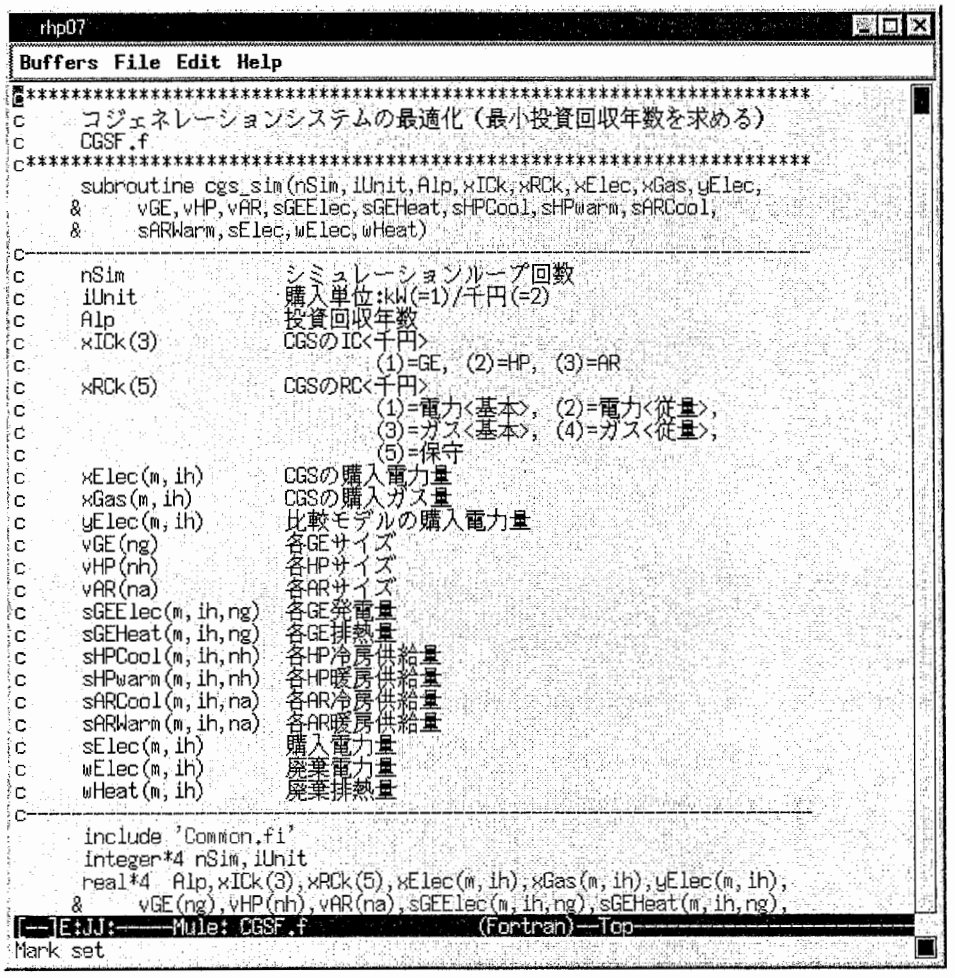
関数名	機能	ファンクションキー※
fortran-erase-comments-region	コメント非表示（範囲指定）	[F1]キー
fortran-recover-comments-region	コメント表示（範囲指定）	[F2]キー
fortran-erase-comments	コメント非表示（全体）	[F3]キー
fortran-recover-comments	コメント表示（全体）	[F4]キー

※ modified-fortran.el中で各関数がファンクションキーにマップされています。
[F1]～[F4]キーについては、modified-fortran.elをロードすることにより、
通常のFortranモードのキーマップが変更されます。

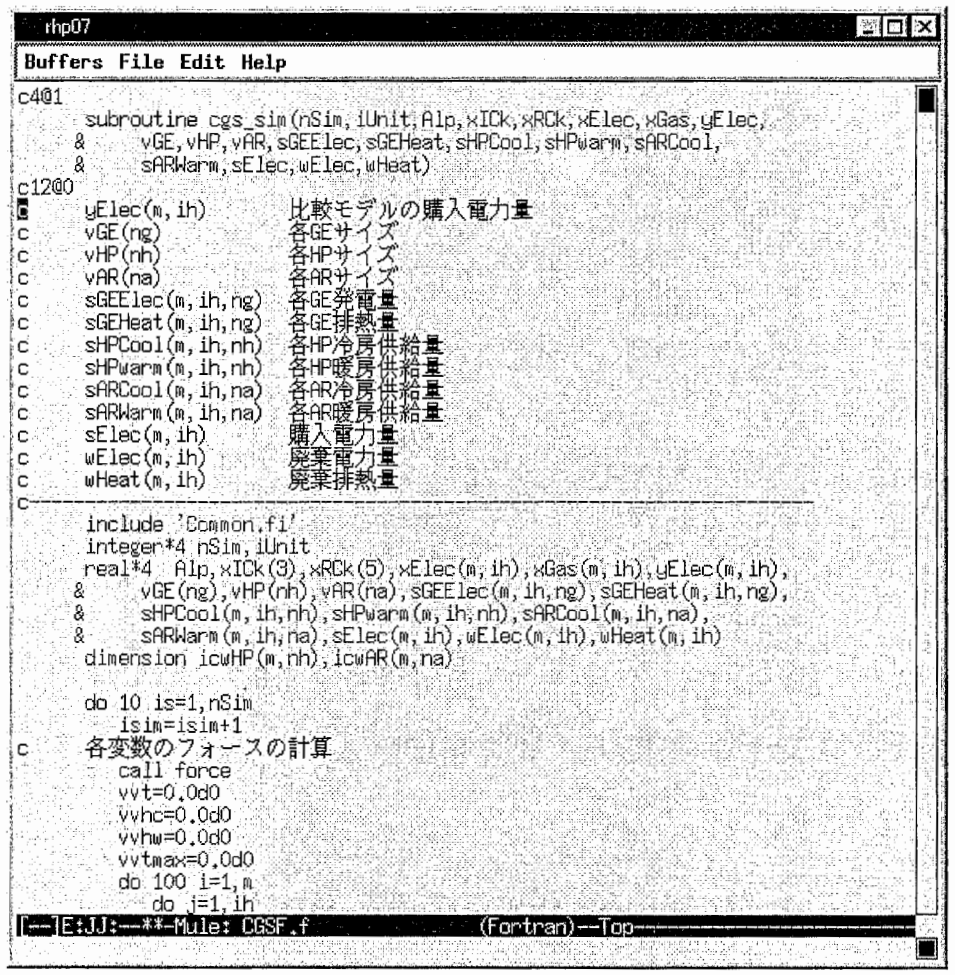
(F1) 範囲を指定してコメント非表示

指定した範囲内で、連続するコメント文を1行の記号列に変換します。

まず、範囲(region)を指定するため、[Ctrl]+[SPACE]キーにより範囲の先頭にマークを設定します。



次に、指定する範囲までカーソルを移動し、[F1]キーを押します。



指定した範囲内で、連続したコメント文が、各々まとめられて１行の記号列に変換されます。

cX@Y

X: 変換されたコメント文の行数
Y: コマンドが管理用に付けるユニークな番号

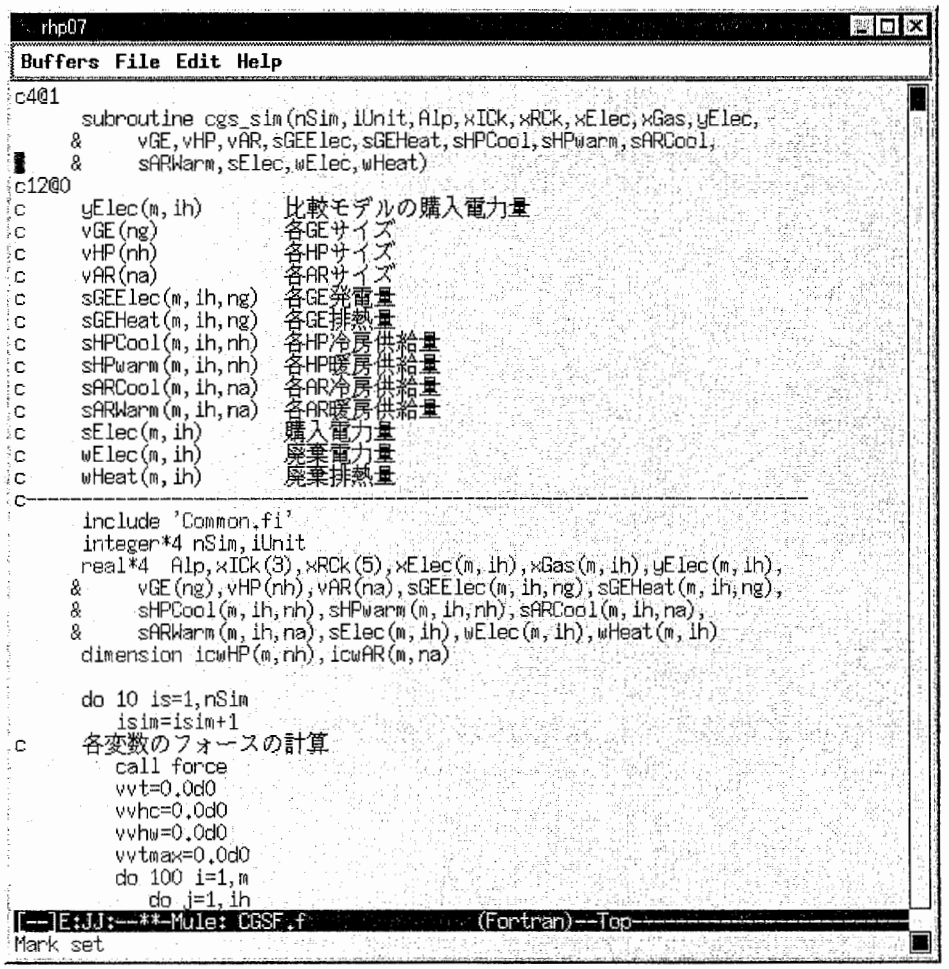
変換されたコメント文は、後述のコメント表示コマンドを使って元に戻すことができます。変換された状態の記号列を削除すると、それに相当するコメント文がまとめて削除されます。なお、記号列の一部を変更した場合、動作は保証されません。

ファイル保存時には、変換されたコメント文は元に戻して保存されます。

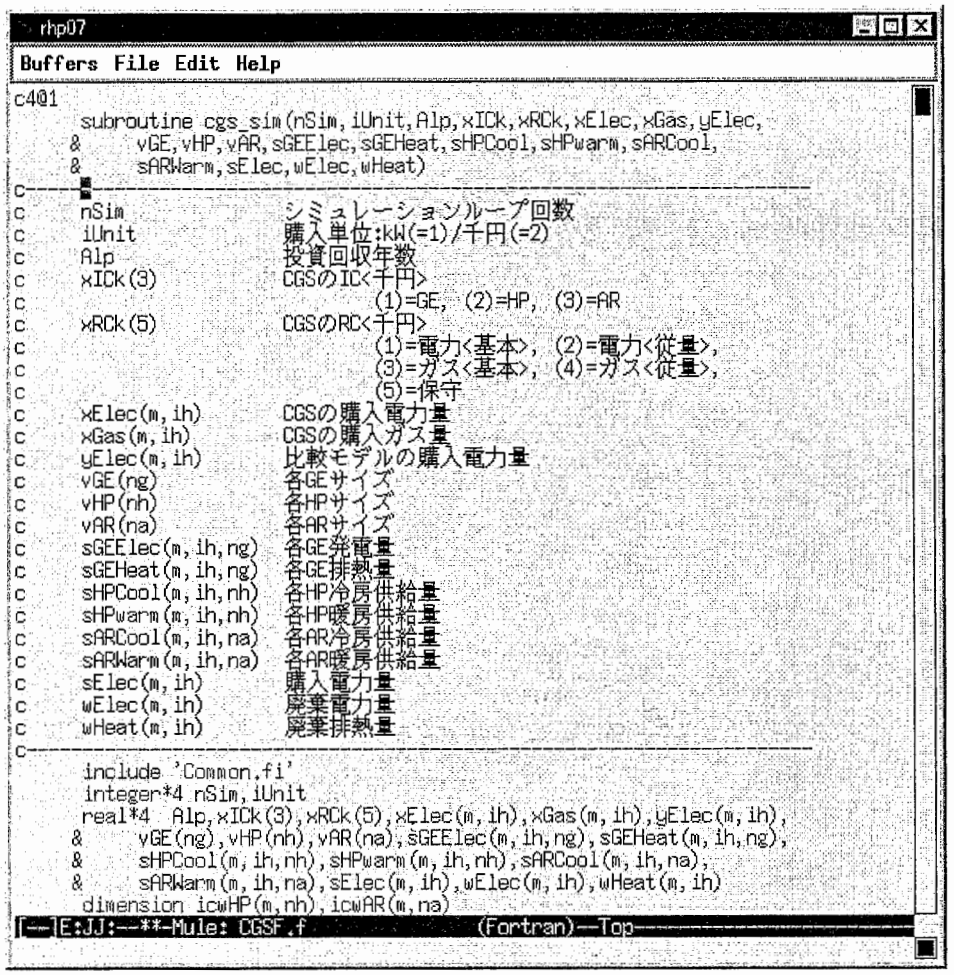
(F2) 範囲を指定してコメント表示

指定した範囲内で、記号列を元のコメント文に戻します。

まず、範囲(region)を指定するため、[Ctrl]+[SPACE]キーにより範囲の先頭にマークを設定します。



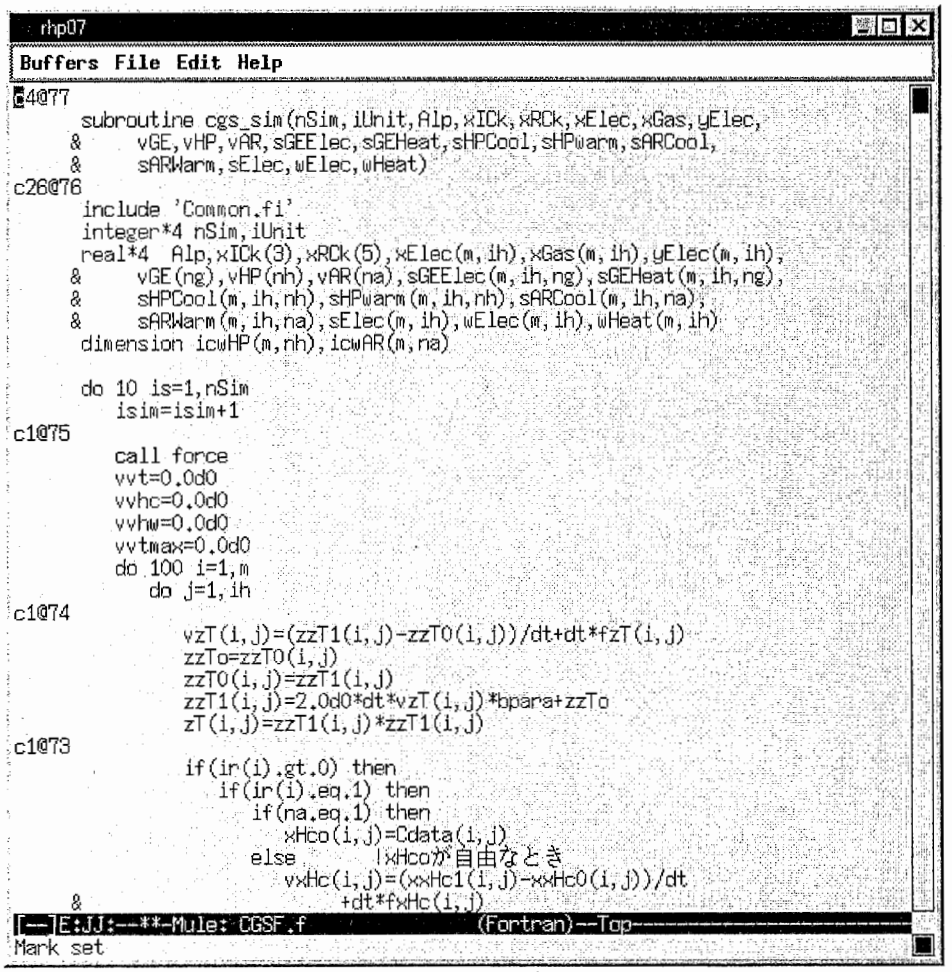
次に、指定する範囲までカーソルを移動し、[F2]キーを押します。


rhp07
Buffers File Edit Help
c4@1
subroutine cgs_sim(nSim,iUnit,Alp,xICk,xRCk,xElec,xGas,yElec,
& vGE,vHP,vAR,sGEElec,sGEHeat,sHPCool,sHPwarm,sARCool,
& sARWarm,sElec,wElec,wHeat)
c
c nSim シミュレーションループ回数
c iUnit 購入単位:kW(=1)/千円(=2)
c Alp 投資回収年数
c xICk(3) CGSのIC<千円>
c (1)=GE, (2)=HP, (3)=AR
c xRCk(5) CGSのRC<千円>
c (1)=電力<基本>, (2)=電力<従量>,
c (3)=ガス<基本>, (4)=ガス<従量>,
c (5)=保守
c xElec(m,ih) CGSの購入電力量
c xGas(m,ih) CGSの購入ガス量
c yElec(m,ih) 比較モデルの購入電力量
c vGE(ng) 各GEサイズ
c vHP(nh) 各HPサイズ
c vAR(na) 各ARサイズ
c sGEElec(m,ih,ng) 各GE発電量
c sGEHeat(m,ih,ng) 各GE排熱量
c sHPCool(m,ih,nh) 各HP冷房供給量
c sHPwarm(m,ih,nh) 各HP暖房供給量
c sARCool(m,ih,na) 各AR冷房供給量
c sARWarm(m,ih,na) 各AR暖房供給量
c sElec(m,ih) 購入電力量
c wElec(m,ih) 廃棄電力量
c wHeat(m,ih) 廃棄排熱量
include 'Common.fi'
integer*4 nSim,iUnit
real*4 Alp,xICk(3),xRCk(5),xElec(m,ih),xGas(m,ih),yElec(m,ih),
& vGE(ng),vHP(nh),vAR(na),sGEElec(m,ih,ng),sGEHeat(m,ih,ng),
& sHPCool(m,ih,nh),sHPwarm(m,ih,nh),sARCool(m,ih,na),
& sARWarm(m,ih,na),sElec(m,ih),wElec(m,ih),wHeat(m,ih)
dimension icwHP(m,nh),icwAR(m,na)
-:E:JJ:--**-Mule: CGSF.f (Fortran)--Top

(F3) ソースコード全体のコメント非表示

ソースコード全体で、連続するコメント文を１行の記号列に変換します。

[F3]キーを押します。



ソースコード全体で、連続したコメント文が、各々まとめられて１行の記号列に変換されます。

cX@Y

X: 変換されたコメント文の行数
Y: コマンドが管理用に付けるユニークな番号

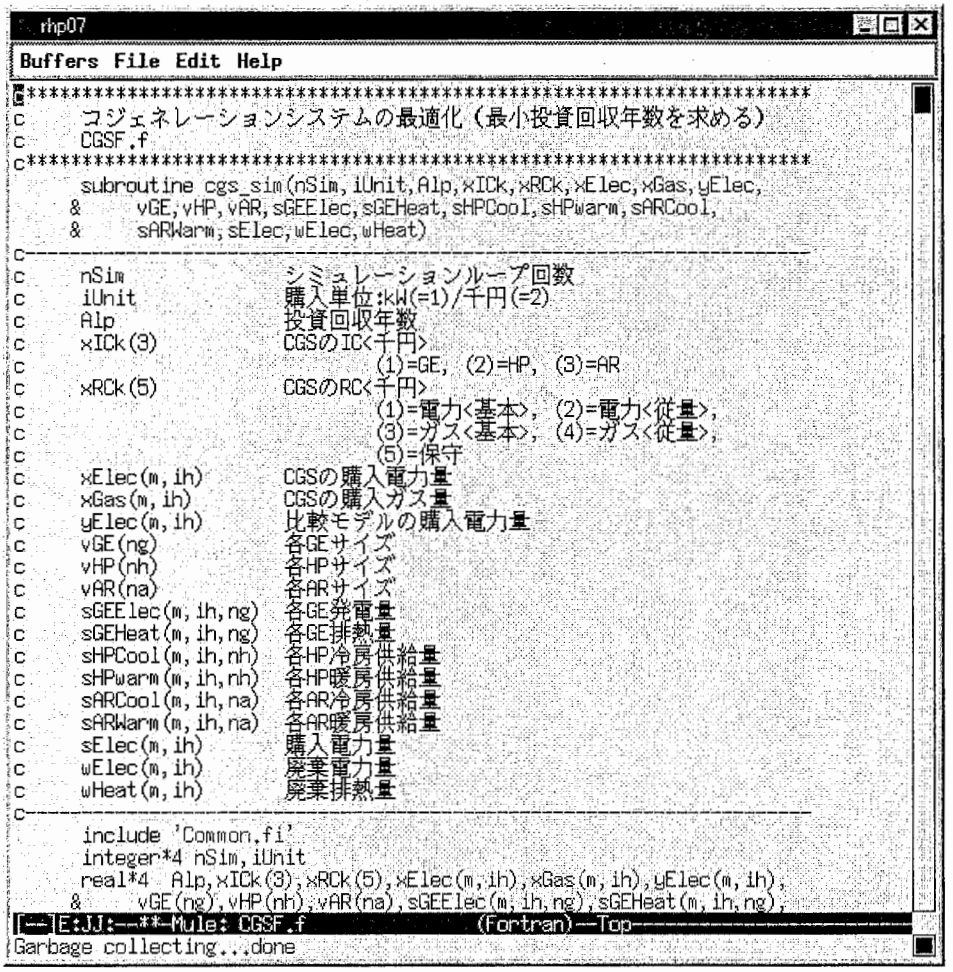
変換されたコメント文は、コメント表示コマンドを使って元に戻すことができます。変換された状態の記号列を削除すると、それに相当するコメント文がまとめて削除されます。なお、記号列の一部を変更した場合、動作は保証されません。

ファイル保存時には、変換されたコメント文は元に戻して保存されます。

(F4) ソースコード全体のコメント表示

ソースコード全体で、記号列を元のコメント文に戻します。

[F4]キーを押します。



以上

A1 プログラムソースリスト

```
;;;
;;; <modified-fortran.el : Fortranコメント文の操作コマンド>
;;; Copyright(C) 1999-2000, ATR Adaptive Communications Research Laboratories
;;; 本プログラムは、GNU一般公有使用許諾、GNUライブラリ一般公有使用許諾に従い、
;;; 無保証であり、再頒布・変更は自由です。
;;;
(load-library "hilit19")
(load-library "fortran")

(defmacro caar (conscell)
  (list 'car (list 'car conscell)))

(defvar after-save-hook nil)

(defvar *fortran-comment-list* nil)
(defvar *fortran-comment-save-list* nil)
(defvar *fortran-comment-seq-no* nil)
(defvar *fortran-current-point* nil)
(defvar *fortran-default-hilit-patterns* nil)

;;;-------------------------------------------------------------------------
;;; hilit19.el で記述されているFORTRANモードのデフォルトの色パターン
;;;-------------------------------------------------------------------------
;;;(setq
;;; *fortran-default-hilit-patterns*
;;; '(
;;;    ("^[*Cc].*$" nil comment)
;;;    ("'[^'\n]*'" nil string)
;;;    ("\\(^[ \t]*[0-9]+\\|[ \t]continue[ \t\n]\\|format\\)" nil define)
;;;    ("[ \t]\\(do\\|do[ \t]*[0-9]+\\|go[ \t]*to[ \t]*[0-9]+\\|end[ \t]*do\\|if\\|else[ \t]*
if\\|then\\|else\\|end[ \t]*if\\)[ \t\n(]" nil define)
;;;    ("[ \t]\\(call\\|program\\|subroutine\\|function\\|stop\\|return\\|end\\|include\\)[ \
t\n]" nil include)
;;;    ("[ \t]\\(parameter[\t\n ]*([^)]*)\\|data\\|save\\|common[ \t\n]*/[^/]*/\\)"
;;;     nil decl)
;;;    ("^     ." nil type)
;;;    ("implicit[ \t]*none" nil decl)
;;;    ("\\([ \t]\\|implicit[ \t]*\\)\\(dimension\\|integer\\|real\\|double[ \t]*precision\\|
character\\|logical\\|complex\\|double[ \t]*complex\\)\\([*][0-9]*\\|[ \t\n]\\)" nil keyword)

;;;    ))

;;;-------------------------------------------------------------------------
;;; fortran.el で定義されている関数の再定義
;;;       書き加えた行には MODIFIED と書いている
;;;-------------------------------------------------------------------------
(defun fortran-mode ()
  "Major mode for editing Fortran code.
\\[fortran-indent-line] indents the current Fortran line correctly.
DO statements must not share a common CONTINUE.

Type ;? or ;\\[help-command] to display a list of built-in\
 abbrevs for Fortran keywords.

Key definitions:
\\{fortran-mode-map}

Variables controlling indentation style and extra features:

 comment-start
    Normally nil in Fortran mode.  If you want to use comments
    starting with '!', set this to the string \"!\".
 fortran-do-indent
    Extra indentation within do blocks.  (default 3)
 fortran-if-indent
    Extra indentation within if blocks.  (default 3)
 fortran-structure-indent
    Extra indentation within structure, union, map and interface blocks.
    (default 3)
 fortran-continuation-indent
    Extra indentation applied to continuation statements.  (default 5)
 fortran-comment-line-extra-indent
    Amount of extra indentation for text within full-line comments. (default 0)
 fortran-comment-indent-style
    nil    means don't change indentation of text in full-line comments,
    fixed  means indent that text at 'fortran-comment-line-extra-indent' beyond
           the value of 'fortran-minimum-statement-indent-fixed' (for fixed
           format continuation style) or 'fortran-minimum-statement-indent-tab'
           (for TAB format continuation style).
    relative  means indent at 'fortran-comment-line-extra-indent' beyond the
              indentation for a line of code.
    (default 'fixed)
```

```
 fortran-comment-indent-char
    Single-character string to be inserted instead of space for
    full-line comment indentation.  (default \" \")
 fortran-minimum-statement-indent-fixed
    Minimum indentation for Fortran statements in fixed format mode. (def.6)
 fortran-minimum-statement-indent-tab
    Minimum indentation for Fortran statements in TAB format mode. (default 9)
 fortran-line-number-indent
    Maximum indentation for line numbers.  A line number will get
    less than this much indentation if necessary to avoid reaching
    column 5.  (default 1)
 fortran-check-all-num-for-matching-do
    Non-nil causes all numbered lines to be treated as possible \"continue\"
    statements.  (default nil)
 fortran-blink-matching-if
    From a Fortran ENDIF statement, blink the matching IF statement.
    Also, from an ENDDO statement, blink on matching DO [WHILE] statement.
    (default nil)
 fortran-continuation-string
    Single-character string to be inserted in column 5 of a continuation
    line.  (default \"$\")
 fortran-comment-region
    String inserted by \\[fortran-comment-region] at start of each line in
    region.  (default \"c$$$\")
 fortran-electric-line-number
    Non-nil causes line number digits to be moved to the correct column
    as typed.  (default t)
 fortran-break-before-delimiters
    Non-nil causes 'fortran-do-auto-fill' breaks lines before delimiters.
    (default t)
 fortran-startup-message
    Set to nil to inhibit message first time Fortran mode is used.

Turning on Fortran mode calls the value of the variable 'fortran-mode-hook'
with no args, if that value is non-nil."
  (interactive)
  (kill-all-local-variables)
  (if fortran-startup-message
      (message "Emacs Fortran mode %s.  Bugs to %s"
               fortran-mode-version bug-fortran-mode))
  (setq fortran-startup-message nil)
  (setq local-abbrev-table fortran-mode-abbrev-table)
  (set-syntax-table fortran-mode-syntax-table)
  (make-local-variable 'fortran-break-before-delimiters)
  (setq fortran-break-before-delimiters t)
  (make-local-variable 'indent-line-function)
  (setq indent-line-function 'fortran-indent-line)
  (make-local-variable 'comment-indent-function)
  (setq comment-indent-function 'fortran-comment-hook)
  (make-local-variable 'comment-line-start-skip)
  (setq comment-line-start-skip
        "^[Cc*]\\(\\([^ \t\n]\\)\\2\\2*\\)?[ \t]*\\|^#.*")
  (make-local-variable 'comment-line-start)
  (setq comment-line-start "c")
  (make-local-variable 'comment-start-skip)
  (setq comment-start-skip "!+ \t]*")
  (make-local-variable 'comment-start)
  (setq comment-start nil)
  (make-local-variable 'require-final-newline)
  (setq require-final-newline t)
  (make-local-variable 'abbrev-all-caps)
  (setq abbrev-all-caps t)
  (make-local-variable 'indent-tabs-mode)
  (setq indent-tabs-mode nil)
;;;(setq abbrev-mode t) ; ?? (abbrev-mode 1) instead??
  (setq fill-column 72) ; Already local?
  (use-local-map fortran-mode-map)
  (setq mode-name "Fortran")
  (setq major-mode 'fortran-mode)
;;;(make-local-variable 'fortran-tab-mode)
  (make-local-variable 'fortran-comment-line-extra-indent)
  (make-local-variable 'fortran-minimum-statement-indent-fixed)
  (make-local-variable 'fortran-minimum-statement-indent-tab)
  (make-local-variable 'fortran-column-ruler-fixed)
  (make-local-variable 'fortran-column-ruler-tab)
  (make-local-variable 'fortran-tab-mode-string)
  (setq fortran-tab-mode-string " TAB-format")
  (setq indent-tabs-mode (fortran-analyze-file-format))
  (make-local-variable '*fortran-comment-list*) ;; MODIFIED
  (setq *fortran-comment-list* nil) ;; MODIFIED
  (make-local-variable '*fortran-comment-save-list*) ;; MODIFIED
  (setq *fortran-comment-save-list* nil) ;; MODIFIED
```

```
  (make-local-variable '*fortran-comment-seq-no*) ;; MODIFIED
  (setq *fortran-comment-seq-no* 0) ;; MODIFIED
  (make-local-variable '*fortran-current-point*) ;; MODIFIED
  (setq *fortran-current-point* 1) ;; MODIFIED
  (make-local-variable 'write-file-hooks) ;; MODIFIED
  (setq write-file-hooks (cons 'fortran-before-save write-file-hooks)) ;; MODIFIED
  (make-local-variable 'after-save-hook) ;; MODIFIED
  (setq after-save-hook (cons 'fortran-after-save after-save-hook)) ;; MODIFIED
  (make-local-variable 'after-change-functions) ;; MODIFIED
  (setq after-change-functions                  ;; MODIFIED
        (append after-change-functions '(fortran-hilit-recenter))) ;; MODIFIED
  (run-hooks 'fortran-mode-hook))

;;;-------------------------------------------------------------------------
;;; 文字列の色指定処理
;;;-------------------------------------------------------------------------
(defun fortran-set-hilit (list)
  "文字列の色を指定する。
   list = ((文字列 色シンボル) .......)
   特殊な場合として、文字列の位置に comment というシンボルがくれば、
   コメントの色を指定する"
  (hilit-set-mode-patterns 'fortran-mode
                           (append (fortran-make-hilit-patterns list)
                                   *fortran-default-hilit-patterns*)
                           nil 'case-insensitive)
  t)

(defun fortran-make-hilit-patterns (list)
  (cond ((null list) nil)
        (t
         (let ((delimiter "\\b") ;;(delimiter "[^a-z0-9A-Z_]")
               (comment-string "^[^ \t\n].*$")
               (string (caar list))
               (color (nth 1 (car list))))
           (setq string
                 (if (eq string 'comment)
                     comment-string
                   (concat delimiter string delimiter)))
           (cons (fortran-make-hilit-pattern string color)
                 (fortran-make-hilit-patterns (cdr list)))))))

(defun fortran-make-hilit-pattern (string color)
  (let ((var (fortran-gensym)))
    (hilit-associate 'hilit-face-translation-table var color)
    (list string nil var)))

(defvar *fortran-gensym-no* 0)
(defun fortran-gensym ()
  (setq *fortran-gensym-no* (1+ *fortran-gensym-no*))
  (intern (format "*fortran-var%d*" *fortran-gensym-no*)))

(defun fortran-hilit-recenter (x y z)
  "テキスト変更があったときの色付けのやり直し"
  (sit-for 0)
  (hilit-repaint-command 'visible))

;;;-------------------------------------------------------------------------
;;; コメントの非表示処理
;;;-------------------------------------------------------------------------
;;;
;;; データ構造
;;;
;;;   *fortran-comment-seq-no*
;;;      コメントに通し番号を付けるために使用
;;;
;;;   *fortran-comment-list*
;;;      コメントの記録
;;;      (通し番号 コメント)
;;;
;;;   *fortran-comment-save-list*
;;;      一時的にコメントマーク (c行数@通し番号) をコメントに戻したときに、
;;;      その記録に使う
;;;      (開始点  終了点  通し番号  行数)
;;;

(define-key fortran-mode-map [f1] 'fortran-erase-comments-region)
(define-key fortran-mode-map [f2] 'fortran-recover-comments-region)
(define-key fortran-mode-map [f3] 'fortran-erase-comments)
(define-key fortran-mode-map [f4] 'fortran-recover-comments)

(defun fortran-message (f &rest args)
  (let ((cur (current-buffer))
```

```
        (string (apply 'format f args))
        (buffer (get-buffer-create "*Fortlan-Mode-Log*")))
    (pop-to-buffer buffer)
    (goto-char (point-max))
    (insert string)
    (pop-to-buffer cur)))

(defun fortran-before-save ()
  "ファイル書き込み前に呼ばれる。
   コメントマークをコメントに戻す"
;;;(fortran-message "fortran-before-save 呼び出し\n")
  (setq *fortran-current-point* (point))
  (cond ((not (fortran-recover-comments))
         (cond ((y-or-n-p "Illegal Comment Mark(s)!  Save to other file ?")
                (write-file (read-file-name "Enter file name"))
                (fortran-recover-from-save)
                (goto-char *fortran-current-point*)
                t)
               (t nil)))
        (t nil)))

(defun fortran-after-save ()
  "ファイル書き込み後に呼ばれる。
   コメントからコメントマークに戻す"
;;;(fortran-message "fortran-after-save 呼び出し\n")
  (fortran-recover-from-save )
  (goto-char *fortran-current-point*)
  (set-buffer-modified-p nil)
  nil)

(defun fortran-erase-comments ()
  "コメントをコメントマークに変える"
  (interactive)
  (fortran-erase-comments-region (point-min) (point-max)))

(defun fortran-erase-comments-region (start end)
  "コメントをコメントマークに変える。リージョン付き"
  (interactive "r")
  (let ((p (point)))
    (goto-char start)
    (beginning-of-line)
    (fortran-erase-regions (fortran-find-comments end nil 0 nil))
;;; (fortran-message "%S\n" *fortran-comment-list*)
    (if (< p (point-max))
        (goto-char p)
      (goto-char (point-max)))))

(defun fortran-find-comments (end cur n result)
  (while (< (point) end)
    (cond ((and (not (member (char-to-string (following-char))
                             '(" " "\t" "\n")))
                (not (fortran-comment-mark-p)))
           (if (not cur)
               (setq cur (point)))
           (setq n (1+ n)))
          (cur
           (setq result (cons (list cur (point) n) result))
           (setq cur nil)
           (setq n 0))
          (t nil))
    (forward-line))
  (if cur
      (setq result (cons (list cur (point) n) result)))
  result)

(defun fortran-comment-mark-p ()
  (let* ((regexp "^c[1-9][0-9]*@[0-9]+[\n]")
         (p (point))
         found
         result)
    (setq found (re-search-forward regexp nil t))
    (if found
        (setq found (re-search-backward regexp nil t)))
    (setq result (equal p found))
    (goto-char p)
    result))

(defun fortran-erase-regions (regs)
  (while regs
    (let* ((s (car (car regs)))
           (e (nth 1 (car regs)))
           (lcnt (nth 2 (car regs)))
```

```
            (mark-str (format "c%d@%d\n" lcnt *fortran-comment-seq-no*)))
      (setq *fortran-comment-list*
            (cons (list *fortran-comment-seq-no*
                        (buffer-substring s e))
                  *fortran-comment-list*))
      (delete-region s e)
      (goto-char s)
      (insert mark-str)
;;;   (fortran-text-intangible s (+ s (length mark-str)))
      (setq *fortran-comment-seq-no* (1+ *fortran-comment-seq-no*))
      (setq regs (cdr regs)))))

(defun fortran-text-intangible (s e)
  "テキストへのカーソル侵入を制限する。
   ただし、プログラムによるカーソル移動等にも不都合あり"
  (put-text-property s e 'intangible t))

(defun fortran-recover-comments ()
  "コメントマークからコメントを復元する"
  (interactive)
  (fortran-recover-comments-exe nil nil))

(defun fortran-recover-comments-region (start end)
  "コメントマークからコメントを復元する。リージョン指定付き"
  (interactive "r")
  (fortran-recover-comments-exe start end))

(defun fortran-recover-comments-exe (start end)
  (setq *fortran-comment-save-list* nil)
  (let ((normalp t)
        (p (point))
        (e-found 0)
        (regexp "^c[1-9][0-9]*@[0-9]+[\n]"))
    (goto-char (if start start (point-min)))
    (while (and e-found (or (null end) (< e-found end)))
      (setq e-found (re-search-forward regexp nil t))
      (if (and e-found (or (null end) (<= e-found end)))
          (progn (let* ((s-found (re-search-backward regexp nil t))
                        (mark (buffer-substring s-found e-found))
                        (lcnt-no (fortran-get-comment-no mark))
                        (lcnt (car lcnt-no))
                        (no (cdr lcnt-no)))
                   (forward-line)
;;;                (fortran-message "FOUND: %S\n" mark)
                   (if (fortran-recover-one-comment s-found e-found no lcnt)
                       nil
                     (setq normalp nil)
                     (fortran-message
                      "コメントマーク %S に該当するコメントはありません\n"
                      (substring mark 0 (1- (length mark))))))))))
    (if (< p (point-max))
        (goto-char p)
      (goto-char (point-max)))
    (if (and (not start) (not end) *fortran-comment-list*)
        (progn
          (setq normalp nil)
          (let ((list *fortran-comment-list*))
            (while list
              (fortran-message
               "コメント(%d) %S を復帰させるコメントマークがありません\n"
               (caar list) (nth 1 (car list)))
              (setq list (cdr list))))
          (setq *fortran-comment-list* nil)))
;;; (fortran-message "%S\n" *fortran-comment-save-list*)
    normalp))

(defun fortran-recover-one-comment (s e no lcnt)
  (let* ((found-and-rest
          (fortran-remove-comment-list *fortran-comment-list* no))
         (found (car found-and-rest))
         (rest (cdr found-and-rest)))
    (cond (found
           (setq *fortran-comment-list* rest)
           (delete-region s e)
           (goto-char s)
           (insert (nth 1 found))
           (setq *fortran-comment-save-list*
                 (cons (list s (point) no lcnt)
                       *fortran-comment-save-list*))
           t)
          (t nil))))
```

```
(defun fortran-remove-comment-list (list no)
  "値は、(x . rest)
       x = no の通し番号が見つかれば no の要素、さもなくば nil
       rest = no の要素を除いた list"
  (cond ((null list) nil)
        ((= (caar list) no)
         (cons (car list) (cdr list)))
        (t
         (let* ((found-and-rest (fortran-remove-comment-list (cdr list) no))
                (found (car found-and-rest))
                (rest (cdr found-and-rest)))
           (cons found
                 (cons (car list) rest))))))

(defun fortran-get-comment-no (mark)
  "コメントマークから行数と通し番号を得る"
  (let ((i 1))
    (while (not (string= (substring mark i (+ i 1)) "@"))
      (setq i (1+ i)))
    (cons (string-to-number (substring mark 1))
          (string-to-number (substring mark (1+ i))))))

(defun fortran-recover-from-save ()
  "一時的に戻したコメントのコメントマークへの復帰"
  (while *fortran-comment-save-list*
    (let* ((rec (car *fortran-comment-save-list*))
           (s (nth 0 rec))
           (e (nth 1 rec))
           (no (nth 2 rec))
           (lcnt (nth 3 rec))
           (mark-str (format "c%d@%d\n" lcnt no)))
      (setq *fortran-comment-list*
            (cons (list no (buffer-substring s e))
                  *fortran-comment-list*))
      (delete-region s e)
      (goto-char s)
      (insert mark-str))
    (setq *fortran-comment-save-list* (cdr *fortran-comment-save-list*))))
```