マイクロソフトエディタのカスタマイズ

行 方 常 幸

目　次

1．はじめに

今日情報化時代の到来で，私達の身の回りにも多種多様の機器が整備され，それを使いこなすための労力も無視できなくなってきている。私に関して言えば，一番身近にあるのは，NECのパーソナルコンピュータ，さらに情報処理センターにある富士通のパーソナルコンピュータ，SUN及びΣの両ワークステーション，等である。例えば，プログラムを作成する際に利用するエディタを考えてみると，それぞれに異なったソフトがあり，その利用法を覚えるだけでも大変である。しかし，この頃は既定のキー操作法を，ある程度自分の好きなように変更できるカスタマイズ機能がついており，それを利用して自分の慣れた操作法に変更すれば，利用法を覚える際の労力をかなり省ける。ここでは，パーソナルコンピュータ用のマイクロソフトエディタをファイナル風にカスタマイズしたのでその概要を述べる。

なぜマイクロソフトエディタを利用する必要があったかと言えば，私が利用可能なMS OS／2のプロテクトモード用のエディタがこれしかなかったからである。

2. MS OS／2とマイクロソフトエディタ

パーソナルコンピュータのオペレーティングシステム（OS）としては，マイクロソフト社のMS-DOSが主流であるが，同社の次期世代のOSまたはMS-DOSのいくつかの限界を克服したOSとしてMS OS／2が注目されている。

MS OS／2を動作させてまず分かることは，シングルユーザー，シングルタスク用のOSであるMS-DOSと違い，MS OS／2はシングルユーザー，マルチタスク用のOSであるということである。すなわち，大型計算機やワークステーションのように複数の人の同時の利用を目的とした（マルチユーザー）OSではないが，1人の人が複数の仕事を同時にするために設計されたOSであると言うことである。もう少し具体的に言うと，CTRL＋ESCまたはGRPH＋ESCを押すことにより，1つのプログラムを実行中に，そのプログラムを終了せずに他のプログラムを実行できる。また，マイクロソフトエディタでは，編集中のファイルをバックグラウンドでコンパイルしながら，他のファイルの編集ができる。MS-DOSでは，編集中のファイルのコンパイル中は何もできず，コンパイルが終わってから次の処理をしなければならない。

MS OS／2には，このように複数の仕事を処理できるプロテクトモード（OS／2モード）があるがこのモードでは今までのMS-DOS用に開発されたソフトは利用できない。そこでこのプロテクトモードの他に，MS-DOSとの互換を実現したリアルモード（DOS互換モード）がある。プロテクトモード用ソフトはまだ余りないが，プログラム言語としては，マイクロソフトCコンパイラ（MS-C ver 5.1），マイクロソフトマクロアセンブラ（MASM ver 5.1）等が発売されており，それに付属しているのがマイクロソフトエディタである。その他にもソースレベルでデバッグできるコードビューも付いている。マイクロソフトエディタはこれからのマイクロソフト言語に標準で装備される標準エディタとして位置づけられている。マイクロソフトエディタにはリアルモード用（M.EXE）とプロテクトモード用（MEP.EXE）の2個があり，キー操作も同じで，リアルモード用はMS-DOS（ver 3. xx）でも利用できるので大変あり

がたいものである。

私が利用したことのあるエディタはファイナル，マイフェス，及びマイクロソフトエディタであるが，それぞれに特徴があり，利用可能な使い慣れたものを利用するのが一番良いと思われるが，私の気が付いた上記のエディタにはないマイクロソフトエディタの特徴を少しあげてみる。単独のエディタでありながら，TURBO PASCAL 等のように，エディタ内からコンパイルできコンパイルエラーがあればソースファイルのエラーの箇所へカーソルが移動する。編集したファイル名や，編集位置，エディタの状況などを保存するため，エディタを編集ファイル名を指定せずに起動すると，一番最近にエディタを終了したときの状態になる。C 言語でプログラムすることにより，新しい機能を追加できる。ただし，マイフェスにもマクロ言語 MIL というのを利用すれば新しい機能を追加できる。

3．マイクロソフトエディタの機能

マイクロソフトエディタには，（表1）にあるように 62 個の基本的な機能があり，それをいろいろな構文に組み合わせることにより，エディタとしての強力な機能を実現している。この 62 個の機能をどのような構文に組み合わせるとどの様な機能が実現できるかに関する詳細はマニュアルを参照していただくとして，いくつか例をあげてみる。

正規表現された文字列を現在位置からファイルの終わりの方へ検索するには，

Arg Arg 検索文字列 Psearch

という構文を使う。デフォルトでは Arg に GRPH＋A，Psearch に F 3 が割り当てられているので，GRPH＋A，GRPH＋A，その後検索文字列を入力するか，カーソルを移動させて検索文字列をハイライトさせ，最後に，F 3 とすれば良い。この例のように，Arg 機能を呼び出すと，キー入力またはカーソル移動によって，次の機能に引数を渡すことができる。

画面上のブロックを削除して，バッファ（クリップボード）にコピーするに

表1　マイクロソフトエディタの機能とデフォルトのキー操作

機　能	デフォルトのキー操作	機　能	デフォルトのキー操作
Arg	GRPH+A	Mark	CTRL+M
Argcompile	F 5	Meta	F 9
Assign	GRPH+=	Mlines	CTRL+W
Backtab	SHIFT+TAB	Mpage	ROLL DOWN または CTRL+R
Begline	HOME	Mpara	CTRL+ROLL DOWN
Cancel	ESC	Msearch	F 4
Cdelete	CTRL+G	Mword	CTRL+←または CTRL+A
Compile	SHIFT+F 3	Newline	
Copy	CTRL+INS	Paste	SHIFT+INS
Curdate		Pbal	CTRL+ [
Curday		Plines	CTRL+Z
Curfile		Ppage	ROLL UP または CTRL+C
Curfileext		Ppara	CTRL+ROLL UP
Curfilenam		Psearch	F 3
Curtime		Pword	CTRL+→または CTRL+F
Curuser		Qreplace	CTRL+¥
Down	CTRL+X または↓	Quote	CTRL+P
Emacscdel	BS	Reflesh	SHIFT+F 7
Emacsnewl	リターン	Replace	CTRL+L
Endline	HELP	Restcur	
Execute	F 7	Right	→または CTRL+D
Exit	F 8	Savecur	
Help	F 1	Sdelete	DEL
Home	CLR	Setfile	F 2
Information	SHIFT+F 1	Setwindow	CTRL+]
Initialize	SHIFT+F 8	Shell	SHIFT+F 9
Insertmode	INS または CTRL+V	Sinsert	CTRL+J
Lasttext	CTRL+O	Tab	TAB
Ldelete	CTRL+Y	Undo	GRPH+BS
Left	CTRL+S または←	Up	↑または CTRL+E
Linsert	CTRL+N	Window	F 6

は,

Arg 範囲指定 Ldelete

Arg 範囲指定 Sdelete

という構文を使う。範囲指定はブロックの部分をハイライトさせることによるが，箱型削除の場合は，前者の Ldelete，文字列型削除の場合は，後者の Sdelete を用いる。デフォルトでは Ldelete には Ctrl＋Y，Sdelete には DEL が割り当てられているので，GRPH＋A，カーソルを移動して範囲指定，Ctrl＋Y（または，DEL）と入力する。

Arg 機能を Ctrl＋B に割り当てたい場合には,

（1） Arg "Arg: Ctrl＋B" Assign

という構文を使う。Assign はデフォルトでは GRPH＋＝に割り当てられているので，GRPH＋A,「Arg: Ctrl＋B」とキー入力，GRPH＋＝とすれば，以後 Ctrl＋B で Arg 機能を呼び出すことができる。1つのキー操作に複数の機能を割り当てた場合には，いちばん最後に割り当てた機能だけが有効となる。

「1行挿入してカーソルをその行の先頭へ移動させる」を Ctrl＋N に割り当てるには,

(a) Arg "x: ＝Linsert Meta Begline" Assign

(b) Arg "x: Ctrl＋N" Assign

という構文を使う。1行目は x というマクロを「1行挿入し（Linsert)，カーソルを行の先頭に移動させる（Meta Begline)」と定義し，2行目でこのマクロに Ctrl＋N というキー操作を割り当てている。ここで注意することは，いくつかの機能をまとめてマクロにするには,

（2） マクロ名 ：＝ 処理する機能

とすれば良いことである。また，キー操作を割り当てるには,

（3） マクロ名または機能名 ： キー操作

とすれば良い。

上記の（a，b）の様にすれば，エディタを利用中に実行可能な機能を自分の好きなキー操作に割り当てることができる。さらに，このことをエディタ起動

時にいっぺんにやってしまうことができる。(a, b) で Assign に渡す引数

x: =Linsert Meta Begline

x: Ctrl+N

すなわち,(2, 3) の形式を TOOLS.INI というファイルに書いておくのである。こうしておくと,エディタ起動時に自動的にこの TOOLS.INI ファイルを読み込んで,エディタ利用時にはこれらマクロ及びキー設定が利用できるようになっている。

以上,エディタの利用法の具体的説明の中で,カスタマイズ機能の1つである,TOOLS.INI ファイルを利用したマクロ及びキー操作の設定方法を述べた。もう1つのカスタマイズ方法は,C 拡張と呼ばれ,マニュアルに指定してある引数及び戻り値の型を持つ関数を C 言語で記述することである。この関数の中では,マニュアルに書かれてあるマイクロソフトエディタの低レベル関数や OS のシステムコールが利用できる。

この C 拡張で定義した関数は,エディタを実際に利用するときには,マクロとして扱われる。また,これら自分で作る関数内で利用できるマイクロソフトエディタの低レベル関数には,マイクロソフトエディタのマクロを実行する fExecute があり,上で述べた (1) の代わりに,プログラム内で,

fExecute ("Arg ¥" Arg: Ctrl+B¥" Assign");

と記述すれば同じ効果が得られる。

4. カスタマイズの方法

ここでは,3. で述べたファイル TOOLS.INI と C 拡張を利用して,ファイナル風にカスタマイズを行なう概要を述べる。ここでファイナル風という意味は (表2) のようにキー操作の割り当てを行なうという意味であり,ファイナルのデフォルトのキー割り当てとまったく同じようにすることではない。また,カスタマイズを行なう際には,マイクロソフトエディタの機能を最大限に利用することを目標とする。

3. で説明したように,TOOLS.INI ファイルの中で,

表2　カスタマイズ後の2ストロークキー操作

キー操作	機　　能	キー操作	機　　能
Ctrl＋JC	ヘルプ画面から戻る	Ctrl＋PO	任意のコマンドの実行
Ctrl＋JH	ヘルプ	Ctrl＋PS	システムの起動
Ctrl＋JP	ファイル名の変更	Ctrl＋PW	DIR/W の実行
Ctrl＋JU	検索時に大文字小文字	Ctrl＋QA	文字列の置換
	を区別する/しない	Ctrl＋QC	ファイルの最後へ
＊Ctrl＋KA	現ファイルの放棄終了	Ctrl＋QD	行の最後へ
Ctrl＋KB	ブロックの指定開始	Ctrl＋QE	画面の上端へ
Ctrl＋KC	クリップボードからカ	Ctrl＋QF	文字列の検索
	ーソル位置へのコピー	Ctrl＋QG	MEGREP の起動
Ctrl＋KD	現ファイルのセーブ終了	Ctrl＋QH	行頭からカーソル
Ctrl＋KM	カーソル行のマーク		位置までの削除
＊Ctrl＋KQ	全ファイルの放棄終了	Ctrl＋QL	指定行へジャンプ
Ctrl＋KR	カーソル位置への	Ctrl＋QM	マーク行へジャンプ
	ファイル読み込み	Ctrl＋QR	ファイルの先頭へ
Ctrl＋KS	現ファイルのセーブ	Ctrl＋QS	行の先頭へ
Ctrl＋KW	クリップボードを他	Ctrl＋QV	直前の位置へ
	のファイルへコピー	Ctrl＋QX	画面の下端へ
＊Ctrl＋KX	全ファイルのセーブ終了	Ctrl＋QY	カーソル位置から
Ctrl＋OM	編集ファイルの選択		行末までの削除
Ctrl＋ON	新ファイルの選択	Ctrl＋UC	ウィンドウクローズ
Ctrl＋OP	対応括弧の検索	Ctrl＋UO	画面横分割
＊Ctrl＋OU	大文字化	Ctrl＋UT	次のウィンドウへ
Ctrl＋PC	コンパイラの起動	Ctrl＋UU	アンドゥ
Ctrl＋PD	DIR の実行	Ctrl＋UY	画面縦分割

（注）＊印の付いている機能はプロテクトモードでは正常に作動しない。
詳しくは，「5．注意事項と問題点」を参照。

機能名またはマクロ名 ： キー操作

と記述すれば，この機能またはマクロに自分の好きなキー操作を割り当てることができると述べた。これを利用すれば，どの機能にも（必要な機能がなければ，C拡張を利用して自分でその機能を実現する関数を作る）自分の好きな1ストロークキーを割り当てることができる（図1参照)。しかし，この方法では2ストロークキーを割り当てることができない。すなわち，

xx: Ctrl+JH

等，としてマクロxxにCtrl+JHを割り当てることができないのである。そこで，これを実現するために，少々工夫が必要となる。余談になるが，マイクロソフトエディタにはワードスター風のカスタマイズを行なっているTOOLS.INIが付いており，さらにCtrl+QS, Ctrl+QDの2ストロークキーを利用するユーザーのためにと，C拡張を利用したEXEファイル（プロテクトモード用にはDLLファイル）が付いている。ソースプログラムはないので，どう実現されているか分からないが，2ストロークキーを実現するには，TOOLS.INIファイルのレベルでは不可能で，C拡張まで利用しなければならないと解釈できそうである。しかし，色々試行錯誤を繰り返してみると，2ストロークキーを実現することは，TOOLS.INIのレベルで可能である。

（表2）のうちCtrl+Jで始まる2ストロークキーの割り当てをどのように実現するかを説明する。他の2ストロークキー割り当ても同様である。Ctrl+Jで始まるのはCtrl+JC, Ctrl+JH, Ctrl+JP, Ctrl+JUの4個であるから，Ctrl+Jが入力された時点で2ストローク目のCtrl+C, Ctrl+H, Ctrl+P, Ctrl+Uに，目的とする処理に必要な機能を割り当て，これら2ストローク目のCtrl+C, Ctrl+H, Ctrl+P, Ctrl+Uのいずれかが入力された時点で，元の処理（これらが1ストローク目に入力されたときに行なう処理）に戻す。具体的には，TOOLS.INIファイルに（図2）のように記述すればよい。

初期設定では，

Ppage: Ctrl+C

Emacscdel: Ctrl+H

図1　TOOLS.INI ファイル（1ストロークキー定義の部分）

```
Mword: Ctrl+A                    ; 1語前へ
Arg: Ctrl+B                      ; Arg 機能の呼び出し
Ppage: Ctrl+C                    ; 次画面へ
Right: Ctrl+D                    ; 右へ
Up: Ctrl+E                       ; 上へ
Pword: Ctrl+F                    ; 1語後へ
Sdelete: Ctrl+G                  ; カーソル位置の文字削除
Emacscdel: Ctrl+H                ; 左の文字削除
ipro: = I_PRO                    ; I_PRO はC拡張で定義
ipro: Ctrl+I                     ; TAB コードの挿入 (*)
Replace: Ctrl+L                  ; 確認なしの文字列置換
Meta: Ctrl+M                     ; メタキー
npro: =Linsert Meta Begline
npro: Ctrl+N                     ; 1行挿入後行頭へ
Mpage: Ctrl+R                    ; 前画面
Left: Ctrl+S                     ; 左へ
tpro: = T_PRO                    ; T_PRO はC拡張で定義
tpro: Ctrl+T                     ; 1語削除 (*)
Insertmode: Ctrl+V               ; 挿入/上書き
Mlines: Ctrl+W                   ; 下へ1行スクロール
Down: Ctrl+X                     ; 下へ
Ldelete: Ctrl+Y                  ; 1行削除
Plines: Ctrl+Z                   ; 上へ1行スクロール
Psearch: F5                      ; ファイル末への検索
Msearch: SHIFT+F5                ; ファイル頭への検索
Quote: Ctrl+]                    ; コントロールコードの入力
escpro: = ESC_PRO                ; ESC_PRO はC拡張で定義
escpro: ESC                      ; 取り消し
```

（注）＊印の付いている機能はプロテクトモードでは正常に作動しない。
詳しくは，「5．注意事項と問題点」を参照。

図2　TOOLS.INIファイル
（Ctrl＋Jで始まる2ストロークキー定義の部分）

```
resetjpro: = Arg "Ppage: Ctrl+C" Assign Arg "Emacscdel: Ctrl+H" Assign Arg
"setppro: Ctrl+P" Assign Arg "setupro: Ctrl+U" Assign
; マクロ resetjpro の定義　resetjpro の内容は
; Ctrl+C, Ctrl+H, Ctrl+P, Ctrl+U の割り当てを元に戻す
; マクロ setppro, setupro は別のところで定義されている。

setjc 1 : = Setfile resetjpro
; マクロ setjc 1 の定義　setjc 1 の内容は元のファイルに戻る

setjh 1 : = Help resetjpro
; マクロ setjh 1 の定義　setjh 1 の内容はヘルプファイルを表示

setjp 2 : = Setfile Arg "Emacsnewl: ENTER" Assign
; マクロ setjp 2 の定義　setjp 2 の内容は
; マクロ setjp 1 の残りの処理とリターンキーの割り当てを元に戻す

setjp 1 : = Arg "msg: 1" Assign MESS_PRO resetjpro Arg "setjp 2 : ENTER" Assign
Arg Arg
; C拡張で定義した MESS_PRO を使ってメッセージ
;「変更後のファイル名を入力してリターンキーを押して下さい」
; を表示後リターンキー（ENTER）に setjp 2 を割り当て Arg 機能を2回呼び出す
; setjp 1, setjp 2 で使われている構文
;「Arg Arg ファイル名 Setfile」はカレントファイルをファイル名でセーブする

setju 1 : = JU_PRO resetjpro
; マクロ setju 1 の定義　JU_PRO はC拡張で定義　setju 1 の内容は
; 検索時に大文字と小文字を区別するか，しないかを交互に指定

setjpro: =Arg "setjc 1 : Ctrl+C" Assign Arg "setjh 1 : Ctrl+H" Assign Arg "setjp 1 :
Ctrl+P" Assign Arg "setju 1 : Ctrl+U" Assign
; マクロ setjpro の定義　setjpro の内容は
```

```
; Ctrl+C, Ctrl+H, Ctrl+P, Ctrl+U に setjc 1, setjh 1, sethp 1, setju 1 をそれぞれ割り
当てる

setjpro: Ctrl+J
; マクロ setjpro を Ctrl+J に割り当てる
```

(注) 紙面の関係で2行にまたがったものもあるが，マクロの定義は1行に書く。

setppro: Ctrl+P

setupro: Ctrl+U

であるが，Ctrl+J が入力された時点で，

setjc 1: Ctrl+C

setjh 1: Ctrl+H

setjp 1: Ctrl+P

setju 1: Ctrl+U

と再割り当てを行なう。これら setjc 1，setjh 1，setjp 1，setju 1 のマクロに必要な処理を定義しておけばよい。この必要な処理には元の割り当てに戻す処理（これもまたマクロ resetjpro で定義されている）が含まれている。マクロ setjc 1 は機能 Setfile でヘルプファイルからもとのファイルに戻し，マクロ setjh 1 は機能 Help でヘルプファイルを呼び出す。マクロ setju 1 は別に C 拡張で定義された機能 JU _ PRO を呼び出し，検索時に大文字と小文字を区別するか，しないかを交互に設定する。最後にファイル名の変更を行なうマクロ setjp 1 であるが，カレントファイルを，変更するファイル名でセーブすることで実現する。これは構文

Arg Arg ファイル名 Setfile

を利用すれば良いのだが，ファイナルに習って Ctrl+JP を入力後，ファイル名，リターンでこれを行なうため，次のように2段階で処理する。

マクロ setjp 2 では「変更後のファイル名を入力してリターンキーを押して下さい」とメッセージを表示し，

setjp 2： ENTER

とリターンキーが入力されたらマクロ setjp 2 を実行するようにしておき，Arg 機能を 2 回呼び出す。ファイル名が入力されリターンキーが押されたときに呼び出されるマクロ setjp 2 に Setfile を入れておけば，この setjp 1 と setjp 2 の 2 個で目的の構文が実行されたことになる。

ここで，注意を要することは，2 ストロークキー操作を以上のように割り当てる際に，マクロの展開が，それが実行されるときである，ということを利用している点である。もし，マイクロソフトエディタのマクロの展開が，マクロの定義時に行なわれるなら，上記の方法では 2 ストロークキーを割り当てることはできない。

マイクロソフトエディタにはない機能を C 拡張で作成したので，そのうちの 1 つ Ctrl＋KX で呼び出される関数 KX _ PRO の概要を説明する。

Ctrl＋KX に割り当てられた機能は「変更のあった全てのファイルをセーブして終了する」である。関数 KX _ PRO に変更のあった全てのファイルをセーブする処理を行なわせ，その後

Arg Meta Exit

でエディタを終了させる。

関数 KX _ PRO の定義は（図 3）に与えられている。エディタの中からマクロとして呼び出す場合は関数の戻り値と引数の型はこの通りとする。修正が行なわれてまだセーブされていないファイルは Information ファイル（図 4 参照）のファイルの行数を表わす数字の前に＊印が付いているので，＊印の付いている行の行頭から始まるファイルをすべてセーブすれば良い。Information ファイルを呼び出す機能 Information はデフォルトでは，SHIFT＋F 1 に割り当てられている。

まず，機能 Information を fExecute で呼び出す。上に述べたように＊印が付いているファイルを見つければ良いのだが，少しだけ手間がかかる。

FileNameToHandle（"" , NULL）

でカレントファイル，今の場合は，Information ファイルのハンドルを得る。

図3　関数 KX_PRO の定義

```
char myext_buf [256] ; /* 1行を読み込むためのバッファ*/

flagType pascal EXTERNAL KX_PRO (argData, pArg, fMeta)
unsigned int argData; /*マイクロソフトエディタ内でマクロとして使う関数は*/
ARG far * pArg;     /*戻り値と引数の型をこのようにしなければならない  */
        flagType fMeta;
{
        COL i;
        LINE ln;
        PFILE pFile, pFile2;

        fExecute("Information"); /*Information ファイルを表示させる（図4参照）*/
        ln = 2;
        while (TRUE) {
                pFile = FileNameToHandle ("", NULL) ; /*カレント(Information)*/
                myext_buf [0] = 0;                    /*ファイルのハンドルを得る*/
                myext_buf [31] = 0;
                GetLine(ln, myext_buf, pFile); /*(ln+1)行目をmyext_bufへ読む*/
                if (myext_buf [0] == '<') {  /*1桁目が '<' なら次の行へ*/
                        ;
                        }
                else if ((myext_buf [0] >= 'a') && (myext_buf [0] <= 'z')) {
                        if (myext_buf [31] == '*') {  /*変更ファイルだけセーブする*/
                                i = 0;
                                MoveCur(i, ln);
                                fExecute("Arg Setfile");
                                fExecute("Arg Arg Setfile");
                                fExecute("Information");
                                }
                        }
                else {  /*それ以外なら終了*/
```

```
                return TRUE;
                }
            ln++;  /*次の行へ*/
            }
        return TRUE;
}
```

図4　機能 information を呼び出した例

```
Version 1.00 - Oct 01 1988

<information-file>              2 lines
a:¥initial¥tools.ini           *211 lines
a:¥source¥myext.c               666 lines
<clipboard>                     0 lines

The clipboard is empty

18 virtual pages/10 local
```

（注）　画面中央行数を表わす数字の前の＊印により，
a:¥initial¥tools.ini は修正されていることが分かる。

以後このハンドルを利用して Information ファイルを処理する。（図4）を見るとファイルが書かれてあるのは3行目からなので，3行目から1行ずつ読み込み該当ファイルかどうか調べる。行番号は0から始まるので，3行目はプログラムでは ln＝2 となっている。

GetLine (ln, myext_buf, pFile)

でバッファ myext_buf へ1行読み込む。

最初の桁が '<' なら Information ファイル，クリップボード，等であるから，次の行へ。ドライブ名から始まるファイル名かどうか調べる。ファイル名でなければ，すべて調べたことになるので終了する。ファイル名なら＊印が付いているかどうか調べ，付いている場合には，

MoveCur

で1桁目にカーソルを戻し,

Arg Setfile

でこのセーブすべきファイルに移り,

Arg Arg Setfile

でカレントファイル，今の場合，セーブすべきファイル，をセーブする。次の行を調べるために,

Information

で Information ファイルへ戻る。これを終了するまで繰り返す。

カレントファイルの内容を得るためにわざわざ FileNameToHandle，GetLine 等を利用しなくても良いように設計してあれば，等と思うのだが，それは素人考えのようである。もし，KX_PRO の内容をエディタ内で，キー入力して行なうと，少なくとも2回は表示画面が変わるはずであるが，Ctrl+KX を入力して実際に KX_PRO の処理を行なうと，画面は変化せずに，ファイルをセーブしたというメッセージがメッセージ行に表示されるだけである。これも素人判断であるが，エディタの処理速度を早めるために，処理に時間のかかる画面表示は省略される場合があるようである。

以上，私が行なったカスタマイズのうち，TOOLS.INI ファイルのレベルで2ストロークキーの割り当てが可能である，エディタを利用しやすいようにC拡張で機能を追加した，の2点の概要を具体例をあげて説明した。

5．注意事項と問題点

カスタマイズを行なったことによる，注意事項及び問題点を述べる。まず，2ストロークキーを入力した場合，処理速度が若干落ちることである。長いマクロを処理するためであろうが，できれば改善したい点である。これに関してリアルモード上では気をつけることがある。2ストローク目が Ctrl+S または Ctrl+C の時，すなわち，Ctrl+JC，Ctrl+KC，Ctrl+KS，Ctrl+PC，Ctrl+PS，Ctrl+QC，Ctrl+QSの時，2ストローク目を早く入力してしまう

と，これらCtrl＋SまたはCtrl＋CがMS-DOSで利用されている意味を持っていると判断されるようで，画面停止を意味するCtrl＋Sでは次のキーを停止解除として取り込み，ブレークを意味するCtrl＋Cではカーソル位置の画面表示が少し乱れる。後者の場合でも，ESCを入力し，Ctrl＋C，Ctrl＋R等で，画面を再表示すれば元に戻る。プロテクトモードでは，キー入力は優先度の高いスレッドで行なうためか，このようなCtrl＋CとCtrl＋Sがこちらが意図した意味と違って取られることはない。

ファイナルのブロック操作では，Ctrl＋KBがブロックの開始位置で，Ctrl＋KKでバッファにコピー，Ctrl＋KYでブロックを削除してバッファに移動させる。すなわち，ブロックの終わりの位置で入力する操作も2ストロークキーである。残念ながらこれには対処できず，1ストロークキーとした。そのかわり「カーソル移動で範囲を指定後，＾B（反転領域（箱型）削除），＾Y（文字列型削除），＾K（コピー）」とメッセージ行に表示を行なった。

このようにファイナルと若干異なり指示がないと分かりにくいところは，メッセージを表示するようにした。

以上のところは新しいキー操作を覚える位ならこれ位の我慢はしようというところであるが，以下の問題点は重大である。

（表2）および（図1）において＊印が付いている機能がプロテクトモードでは正常に動作しない。これらの機能を利用しようとすると，

「SYS 1943：プログラムで保護エラーが発生しました」

「SYS 1811：プロセスは停止しました

ソフトウェア診断コード（トラップ番号）は 013 です」

とエラーメッセージが表示されエディタが終了させられる。複数のプログラム（プロセス）が同時に実行されるMS OS／2のプロテクトモードでは，実行中のあるプロセスのデータが他の関係のないプロセスによって不用意に書き換えられると困る。そのため，特に設定をしない限りプロセスは自分のデータしか処理できない。もし，自分のデータ以外を処理しようとするとシステムに終了させられる。この時に上記のエラーメッセージが表示されるようである。

私の素人判断であるが，正常に動作しないＣ拡張の全部で

FileNameToHandle（"" , NULL）

を利用しており，これを簡単な例で検査しても正常に動作しないみたいである。もちろん，デバッガ等で問題箇所を特定できたわけではないので，あくまでも素人判断の域を出ない。この点に関して，マイクロソフト社に簡単な例を同封して質問中である。

6．おわりに

エディタの処理機能にコントロールキーを利用した２ストロークキーを割り当てることができることは，手をホームポジションから移動させなくても良い，という利点がある。ここでは，私の使い慣れているファイナル風にカスタマイズを行なったが，ソースファイル等を変更すれば，違ったカスタマイズが可能である。５．で述べた問題点を改善して行くのは今後の課題である。なおソースプログラムは冗長なので省略した。

参考文献

[１]「OS／２入門」川井健一，佐貫俊幸，船木義昭著　培風館
[２]「OS／２入門」富士ソフトウェア教育出版部著　富士ソフトウェア
[３]「OS／２標準テクノバイブル［基本編］」中西隆著　技術評論社
[４]「マイクロソフトエディタマニュアル」マイクロソフト社