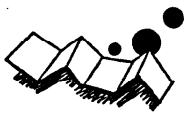
解　説



YALE-TOOLS の画面エディタ Z†

安　川　秀　樹††

1. まえがき

よく言われることだが，エディタはプログラミング・ツールのもっとも基本的かつ重要なものであり，エディタの良し悪しがプログラミングに与える影響は極めて大きいと言える．プログラミング・システムの価値はエディタで決まると言っても過言ではないだろう．どのようなエディタが良いエディタか，あるいは，エディタに求められる機能は何かということは，編集の対象となるテキストの種類やプログラミングの局面によっても異なってくるし，また，エディタがプログラミング・システムにおいてもっとも頻繁に利用されるマンマシン・インタフェースであることを考えると，使い易さ・ヘルプ機能等も重要なファクタである．

本稿で紹介するエディタZは，DEC・SYSTEM-20 の OS, TOPS-20 上のプログラミング環境として，YALE 大学で作られた TOOLS[1] と呼ばれる統合的なユーティリティ・プログラム群の中心を成す画面エディタであり，優れたエディタであるとともに，TOPS-20 の特徴の1つであるプロセス・フォーク機能を活用した優れたプログラミング環境を提供するものである．

Zは，それ以前に YALE で作られたアルファベット1文字の名が付けられた画面エディタの流れをくむもので，基本的には，テキスト指向エディタである．プログラム・エディタとして，テキスト・エディタが良いか構造エディタが良いかという点については，さまざまな議論があるが，Zは，プログラムのもっとも自然な表現はテキストであり，プログラム編集機能を実現するため以外には，テキストに構造を持ち込むべきではないという思想の下に設計されている．その結果，Zは優れたテキスト・エディタであるとともに，LISP, APL, PASCAL, BLISS などの多種のプログラミング言語をサポートし，プログラム・エディタとしても 95% 満足できるものであると，Zの設計者は主張している[2]．

また，Zの設計上のねらいとして，コマンドのレスポンスの良さ，画面へのフィードバックの迅速さという点も挙げられている．

Zと同系統のエディタとして，MIT で作られた EMACS[3] が挙げられ，実際，Zの設計上，かなり影響を与えているように思われる部分も見えるので，適時 EMACS との対比を考えながら説明を行ってゆく．

2. Z の基本構成

ここでは，Zの設計上の基本的要素について述べる．Zの内部でのテキストの表現型式，一般的なコマンドシンタックス，ユーザとのインタフェースの3点について述べることにする．

2.1 ファイル・モデル

Zは，ファイルを，幅・長さ共に無限であるような1/4 平面の上に表現する．すなわち，たて，よこ無限の大きさを持つ配列としてファイルを扱う．ファイルの第1行目の最初の文字が，その平面の原点に位置し，カーソル移動コマンドやページ単位のウインドウ移動コマンド（たとえば〈UP〉,〈DOWN〉,〈LEFT〉,〈RIGHT〉などのカーソル移動コマンドや〈PPAGES〉,〈MPAGES〉などのウインドウ移動コマンド）により，平面の任意の位置に，カーソルやウインドウを移動することができる．編集の途中で，カーソルがウインドウの端に来た場合の動きは，デフォルトでは，ウインドウの右端ではラップし，ウインドウの下端ではスクロールするようにセットされているが，ウインドウの上下左右の端にカーソルが来た場合の動作については，WRAP: と SCROLL: という2つのスイッチによって，どの端でラップするか，あるいはスクロールするのかを指定できる．このように，Zエデ

† The Screen Editor in YALE-TOOLS Programming Environment Z by Hideki YASUKAWA (Institute for New Generation Computer Technology).

†† (財)新世代コンピュータ技術開発機構第二研究室

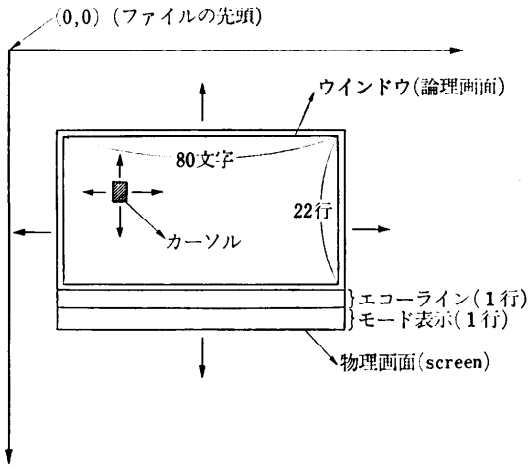


図-1 Z のファイル・モデルと画面・カーソルの移動

ィタでは，その動作のモードを指定するようなパラメータをスイッチと呼んでおり，スイッチの値を自由に設定することにより，エディタの動作の細かい指定ができる．カーソルやウインドウの移動は，EOL (End OF Line) や EOF (End OF File) を越えて行うことができ，EMACSなどのストリーム・エディタ（ファイルを改行やタブなどの入った文字列とみるエディタ）では，EOL や EOFを越えてカーソルやウインドウを移動できないという点と異なる．この制限のため，ほとんどのストリーム・エディタでは，改行やタブなどが，そのコードを挿入するコマンドとなっているがZでは，改行やタブは，その文字の本来の機能，すなわち，次行の先頭にカーソルを移動する，あるいは，次のタブ・ストップの位置までカーソルを移動するという機能を果たす．ただし，改行やタブの機能は，スイッチの設定でソフトウェア的に変更可能である．Zのファイル・モデルと画面・カーソルの移動の様子を**図-1** に示す．図中のエコーラインはコマンドの引数の入力やエディタ側からのプロンプトの表示に用いられ，モード表示ラインは，ファイル名や現在のウインドウの位置やエディタのモードを表示する．

ファイル・モデルに関連して，印刷可能文字をタイプした時の動作について述べておく．ストリーム・エディタでは，基本的にインサート・モード，すなわち印刷可能文字をタイプすると，カーソルの位置に，その文字が挿入されるようになっているのと比べ，Zでは，デフォルトで，オーバライト・モード，すなわち，カーソルの位置から上書きされるようになっている．したがってZでは，ある文字列を他の文字列に変更する場合，たとえば“abc”を“123”に変更しようという場合には，“a”のところへカーソルを移動し，そこから“123”とタイプすればよい．逆に，“134”を“1234”に変更する．すなわち，“1”の次に“2”を挿入する場合は，“3”の位置にカーソルを移動し，そこにブランクを挿入し，そのブランクの位置に“2”と上書きすればよい．このデフォルトのモードの他に，Zにはインサート・モード，ライン・インサート・モードの2種のモードがあり，どのモードを選択するかは，コマンドにより設定できるし，また，Zのイニシャライズ・ファイル中にスイッチの値としてモードを規定しておくことでも可能である(INSERTMODE:, LINE INSERTMODE:)．逆に EMACSには，オーバライト・モードがあり，ユーザは必要に応じて適当なモードを選択できる．

2.2 エディット・コマンドの形式

Zには，テキスト編集やプログラム編集をはじめとして，さまざまな種類のコマンドが準備されている．これらのコマンド自体については，後に説明することにするが，それぞれのコマンドには，基本的には，1〜2 ストロークのキャラクタが割当てられていて，コマンド実行の軽便さが考慮されている．一般によく使用されるコマンドにはコントロール文字が割当てられている．また，32種のコントロール文字の他に，〈SHIFT〉コマンド（通常 ESC キー）を用いて，ソフトウェア的にキーボードを拡張し，オルタネート・キーボード上で各コマンドにキーを割当てることができる．〈SHIFT〉は，次の1文字を読み，オルタネート・キーボード上にマッピングするもので，ASCII 127 字が適用可能である．そのほか，矢印キーやファンクション・キーも使用可能である．どのコマンドにどのようなキー割当てがなされているかは，自由に変更できることもあり，本稿ではあまり触れないが，一例として，代表的なコマンドに対するデフォルトのキー割当てを**表-1** に示す．表中の記号^はコントロールを示す．

表-1 に示したようなキーを入力をすることにより，対応するコマンドのデフォルトの動作が実行されるが，Zでは，各コマンドに対して引数 (argument) を渡したり，あるいは，コマンド・モディファイアを用いて，コマンドの機能自体を多少変更することができ，コマンドの体系を豊かなものへと拡張している．

引数の受渡しは，〈ARG〉コマンドによって指定される．すなわち，〈ARG〉コマンドと引数を取りうる

表-1 Z のデフォルトのキー割当て

NOFUNC	unassigned	SETWINDOW	^]
NOP	SHIFT:^C	SETCURSOR	SHIFT:ESCAPE
SHIFT	ESCAPE	QUOTE	SHIFT:^N
CANCEL	^C	CDELETE	DELETE Key
META	^@	WDELETE	SHIFT:DELETE
HELP	SHIFT:?	SINSERT	^A
LEFT	LEFT Key	SDELETE	unassigned
RIGHT	RIGHT Key	LINSERT	^D
UP	UP Key	LDELETE	^F
DOWN	DOWN Key	PICK	^K
TAB	TAB Key or ^I	PUT	^G
BACKTAB	SHIFT:^I	REPLACE	^O
NEWLINE	RETURN OR ^M	QREPLACE	SHIFT:^O
HOME	HOME Key	BALANCE	^V
BEGLINE	unassigned	INITIALIZE	SHIFT:#
ENDLINE	unassigned	SAVE	unassigned
ARG	^X	UNDELETE	SHIFT:^U
TEXTARG	SHIFT:^X	COMPILE	^U
ASSIGN	^^ and SHIFT:^^	PUSH	SHIFT:^P
SETFILE	^B and SHIFT:^B	EXECUTE	SHIFT:^E
EXIT	^Z	BEXECUTE	unassigned
REFRESH	SHIFT:^]	SM	SHIFT:^Z
INFORMATION	SHIFT:^V	MOVESM	SHIFT:^H
INSERTMODE	^N	SPELL	SHIFT:^R
MARK	^P	FILL	SHIFT:^F
PPAGES	^L	JUSTIFY	SHIFT:^J
MPAGES	unassigned	FLUSHLEFT	SHIFT:<
PLINES	^T	CENTER	SHIFT:^
MLINES	^W	FLUSHRIGHT	SHIFT:>
PSEARCH	^R	LOWERCASE	SHIFT:-
MSEARCH	^E	UPPERCASE	SHIFT:+

コマンドの間に入力されたものが引数として，そのコマンドに渡される．たとえば

〈ARG〉 abc 〈PSEARCH〉

というコマンド列は，〈PSEARCH〉コマンドに “abc” という文字列を引数として渡すことを指定しており，結果として，現在のカーソル位置から前方（ファイルの終わりの方）へ “abc” という文字列をサーチし，もしあれば，その出現位置にカーソルを移動するという機能を果たす．

引数には，null, line, box, stream, text, number の6種のタイプがあり，コマンドにより，どの引数をとり，どのように機能するかが決っている．

null 引数は，文字通り，空の引数を示し，コマンドのデフォルトの機能の次によく用いられる機能を指定する．たとえば〈PPAGES〉は，1スクリーン分だけ先に進むというコマンドだが，これに null 引数を与えた場合，すなわち〈ARG〉〈PPAGES〉は，ファイルの最後の行まで進むという機能を果たす．

text, number 引数は，コマンドに，印刷可能文字からなる文字列あるいは数字を与えるもので，先の〈PSEARCH〉の例は，text 引数の例である．また，〈PLINES〉コマンドは，1回に7行分ずつスクリーンを前方に進める機能を持つが，〈ARG〉3〈PLINES〉と，number 引数を与えると，その数の分，すなわち，3行分だけスクリーンを前方に進める機能を果たす．

残りの stream, box, line 引数は，カーソル引数と呼ばれ，〈ARG〉を実行した時のカーソルの位置（始点）と，引き続いて入力されたカーソル移動コマンドにより決定されるカーソル位置（終点）との間の関係により規定されるファイル内の領域中のテキストを指示する．**図-2** に，カーソル引数の例を示す．図中の “□” は始点を，“○” は終点を表わす．カーソル引数は，ファイル中のある領域のテキストの消去，移動などの機能を簡便かつ素速く行うためのもので，Zの特徴であるとともに，極めて便利なものである．

stream 引数は，始点から終点までの連続した文字列を指示し，文の一部を消したり，移動したり，合わせたりする場合に有用である．

box 引数は，始点と終点を結ぶ線分を対角線とする矩形領域内のテキストを指示し，一かたまりのテキストを，そのまま他の位置へ移動したり，テキストのコ

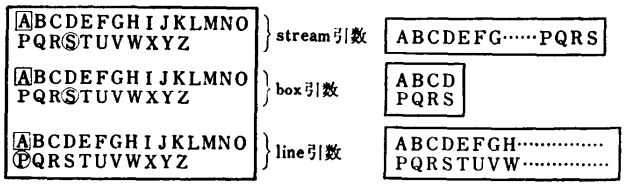


図-2 カーソル引数の例

ラム合わせをしたりする場合に便利である．line 引数は，box 引数の特殊な場合で，始点と終点が同一コラム上にあり，始点を含む行から終点を含む行までのテキストすべて，すなわち右方向に無限の矩形領域内のテキストを指示する．

カーソル引数が，stream 引数なのか，あるいは box，line 引数なのかは，コマンドごとに決っており，たとえば stream の消去・挿入のための〈SDELETE〉，〈SINSERT〉や，box，line の消去・挿入のための〈LDELETE〉，〈LINSERT〉など，対応する2種のエディット・コマンドが準備されている*．

以上に述べたような引数の付与の他に，Zでは，〈META〉コマンドによるコマンドの機能の若干の修正が導入されている．たとえば〈META〉〈UP〉はコラム位置は保持したまま画面の一番上の行までカーソルを移動するという機能を果たす．

これらのコマンドの拡張のための構文は，一見コマンド体系を複雑にしているようだが，同系のコマンド，たとえばサーチ関係のコマンドについては，その引数の果たす役割などは共通した構成となっているため，慣れれば，かえって使いやすい面もある．

2.3 ユーザとのインタフェース

単にユーザとのインタフェースと言っても，実にさまざまな側面がある．特にZにおいては，エディタのバックグラウンド・ジョブとして各種のユーティリティ・プログラムを実行する機能，あるいはエディタをジョブのトップレベルに置き，ログ・ファイルを管理するセッション・マネージャ（SM）機能があり，いろいろな局面で優れたユーザとのインタフェースが提供されうるが，ここでは，特に，編集の局面だけに注目し，カスタマイゼーション・拡張性，コマンド実行の取消し，画面表示およびヘルプ機能について触れることにする．

まず，カスタマイゼーションについてであるが，Zでは，LOGIN：というロジカル名の定義されたディレクトリ中の SWITCH. INI というファイル中に，Zの初期化のためのスイッチのセッティングを書いておき，Zのデフォルトのセット・アップの状態を指定できる．このスイッチとしては，たとえば，ターミナル・タイプの指定，スクロールやラップの動作モードの指定，コマンドへのキーの割当ての指定などが書ける．また，コマンドを組み合わせて，マクロ・コマンドを定義することができ，その定義とキー割当ても，SWITCH. INI ファイル中に書くことができる．Zは基本的にプログラマブル・エディタではないので，EMACS あるいはその下の TECO の場合のように，きめ細かい拡張はできないが，ユーザの必要とする機能はすべてエディト・コマンドとして取り込んでいくという方針の下にバージョン・アップされているようである．エディタの拡張性よりも性能に重きを置いた設計方針で，EMACS と対照的である．

ユーザ・インタフェースとして，編集作業時に必要なこととして，コマンドの実行結果をどのような形でいつユーザに見せるか，誤ったコマンドの実行をどのように中止するか，あるいはその実行により起ったテキストの変化をどのようにして修復するかといったことが挙げられる．

コマンドの実行結果をどのようにして，いつ見せるかということは結局テキストの再表示をどれだけ効率よく行うかということであり，画面エディタにおいてもっとも重要なファクタの1つである．Zは，再表示機能については EMACS の影響をかなり受けており，コマンド実行と画面表示を分離して扱っている．すなわち，コマンド実行により画面が直接変更されるのではなく，コマンド実行後に，ファイルの内容と最近に表示されたものとを比較し，何を表示すべきかを決定する．画面の再表示がその後行われるが，再表示中に画面を変更する可能性の高いコマンドが実行されると再表示は中断され，またその時点で，先と同様に何を表示すべきかが決定される．したがって最適な再表示が可能となる．この再表示のアルゴリズムにより，たとえばページ単位の画面移動コマンドを連続して実行したような場合，1ページずつ移動し表示するといった効率の悪さは省かれ，よりエレガントな表示が可能となっている．大きなファイルの読み込み，サーチ，置き換えなどの非常に時間のかかるコマンドの実行中には，画面の右下すみに，処理の進み具合を

* プレフィックス "S" は stream を "L" は line (box) を示す．Zのコマンド命名上の約束．

示すために，ライン・カウントが表示され，100 行ごとにダイナミックに表示が更新される．またZでは，エコーバックを OS の機能を利用して行っているため，エディタがエコーバックを制御する方式よりも，効率が良い．

コマンドの実行の中止や取消しのために，〈CANCEL〉コマンド，〈UNDELETE〉コマンドが準備されている．〈CANCEL〉は実行中のコマンドを中止するもので，例えばファイル全体に対して，文字列の置き換えを行うなど，時間のかかるコマンドの誤った実行を中止するのに有効である．大抵の場合，ユーザが誤りに気付く前に，コマンドの実行は完了してしまうし，また，気付いて中止した場合も，誤った操作を受けたファイルの内容を，元に戻してやることが必要になる．このために，〈UNDELETE〉コマンドが準備されているが，これは，コマンドの実行による変化を元に戻すというのではなく，ファイルの1つ前のバージョンのものを復活させるもので，したがって，ファイルがセーブされてから後に行われた編集作業の内容についても無効となってしまう．この点で，EMACS には UNDO コマンドがあり，直前に行われたコマンドの効果を元に戻すことができ，より親切で優れていると言える．もう1つ，ユーザ・インタフェースという点で重要なポイントについて触れておく．それは，エディタのヘルプ機能であり，Zのように，コマンド体系が豊富でしかも，各コマンドに，1〜2 ストロークのキーストロークが割当てられているような場合には，絶対に必要な機能である．この用途のために，Z では〈ASSIGN〉と〈HELP〉という2つのコマンドがある．〈ASSIGN〉は，コマンドへのキーストロークの割当てに用いるコマンドだが，〈ARG〉"コマンド名"〈ASSIGN〉により，そのコマンドに対するキー割当てを尋ねることができる．〈HELP〉は，あるキーストロークに対して割当てられたコマンドは何か，逆に，あるコマンドに割当てられたキーストロークは何かを，ヘルプ・インデックス・ファイルとヘルプ・ファイルをうまくエディタの中から利用することによってユーザに表示してくれる．ヘルプ・インデックスは，あるコマンドに関連する各種の情報が記載されたヘルプ・ファイルのメニュを含んでおり，メニュ中の必要なファイルをエディタで見ることにより必要な情報を得る．メニュ中に示されたファイル名を〈SETFILE〉コマンドにカーソル引数として渡すことにより，そのファイルの中を見ることができる．〈SETFILE〉コマンドは，現在のファイル内容のセーブと他のファイルへの移行に用いられるものだが，ヘルプ機能においても重要な役割を果たす．〈SETFILE〉は，現在のファイルの直前に編集していたファイルに移行し，〈ARG〉"FILE名"〈SETFILE〉は，"FILE名"で指定されたファイルをロードし，〈ARG〉〈ARG〉"FILE 名"〈SETFILE〉は現在編集中のファイルの内容を "FILE 名" で指定されたファイルに書き出す．〈ARG〉〈SETFILE〉は現在のカーソル位置から始まるテキストをファイル名として，そのファイルをロードする．

この最後の機能を利用して，計算機内のドキュメントをエディタで見ることにより，必要な情報を得ることができる．すなわち，ヘルプ・メニュには，コマンドに関連する各種トピックについての記述を含むファイル名がリストアップされており，必要な記述を含むファイル名の先頭の位置にカーソルを持ってゆき，〈ARG〉〈SETFILE〉を実行することで，そのファイルの内容をエディタで見て情報を入手できるわけである．**図-3** は，〈HELP〉コマンド実行時に表示されるヘルプ・インデックス・ファイルの一例であり，下線を引いた部分がヘルプ・ファイルのファイル名であ

```
The <setFile> command is assigned to ^B and ESCAPE ^B

                    *** Help Menu ***

To get out of <help>, do <meta><help>, which is assigned to:  ^@ ESCAPE ?
To use the menu, move the cursor to the beginning of the line containing
the topic and do <arg><setFile>, which is assigned to:  ^X ^B

HLP:ZF-SETFILE.HLP <SetFile> moves between files and saves the current file.
HLP:ZS-DEFSPEC.HLP DEFSPEC: gives a default file specification to use when
                   opening a file.

See also file
```

図-3 ヘルプ・インデックス・ファイルの例

る．情報を入手し終えた後に〈META〉〈SETFILE〉コマンドを実行すれば，元のファイルに戻ることができる．個々のトピックごとにドキュメントを作成しておき，エディタをヘルプのトップレベルのインタフェースに利用したもので，使い勝手も良く，うまいやり方だと思う．

Zでは，〈SETFILE〉の場合のように，コマンドを一般的な形で実現し，さまざまな用途に利用している．合理的な方針であり，コマンド体系もシンプルなものになる．(反面，コマンドの使われ方が多少複雑になるが，あまり問題とはならないようだ．)

3. Z の基本的なコマンド

ここでは，Zで用意されている多くのコマンドの中から，代表的なものについて，テキスト編集，プログラム編集という2つの観点から説明を行う．

3.1 テキスト編集用コマンド

テキスト編集用コマンドとしては，カーソルや画面の移動，文字列の挿入・消去・サーチなどの基本的なコマンドや，ワード・プロセッシング用のコマンドが用意されており，機能は豊富である．〈UP〉,〈DOWN〉,〈RIGHT〉,〈LEFT〉といったカーソル移動コマンド，ウインドウを1画面分（22行）だけ前方（後方）に移動する〈PPAGES〉(〈MPAGES〉) コマンド* や数行分ずつ前方（後方）へ移動する〈PLINES〉(〈MLINES〉) コマンドなどのウインドウ移動コマンドの他に，単語単位でカーソルを移動する〈PWORD〉,〈MWORD〉，パラグラフ単位でカーソルを移動する〈PPARAGRAPH〉,〈MPARAGRAPH〉等各種の移動用コマンドが準備されている．〈MARK〉コマンドと〈ARG〉〈PPAGES〉コマンドにより，それぞれファイルの先頭とファイルの最後の行の先頭に移動できる．〈MARK〉コマンドは一般にはファイル中の任意の位置にブックマークをセットしたりあるいは，指定されたブックマークの位置にカーソルを移動させる．ブックマークは，一種の目印であり，〈ARG〉〈ARG〉"文字列"〈MARK〉というコマンドにより，現在のカーソル位置に"文字列"で指定された名前のブックマークをセットできる．ファイル中の任意の位置から，このブックマークの位置までカーソルを移動するためには，〈ARG〉"文字列"〈MARK〉コマンドを用いる．

* プレフィックスの p と w はそれぞれ plus と minus を意味し，forward と backward と同じ意味である．このプレフィックスの付け方は，Zのコマンド名において規則化されている．

ファイル中への文字列の挿入については，先に述べたが，ブランク文字の挿入のための〈SINSERT〉コマンド，空白行や空白の矩形領域の挿入のための〈LINSERT〉コマンドがある．挿入と対応して，消去のためのコマンドも準備されており，文字あるいは文字列を消去するための〈SDELETE〉コマンド，行あるいは矩形領域を消去するための〈LDELETE〉コマンド，単語を消去するための〈WDELETE〉コマンドがある．これらの消去コマンドで消去されたテキストは，バッファにコピーされ，〈PUT〉コマンドを用いることにより，それを現在のカーソル位置に挿入することができる．これらの一連の操作により，テキストを他の任意の位置（他のファイル中も可）に移動させることができる．テキストを他の位置にコピーするために消去コマンドを用いるのは，元のテキストを消去してしまうため，それを〈PUT〉コマンドにより復元してやらねばならず冗長であり，そのため〈PICK〉コマンドが準備されている．〈PICK〉コマンドは，テキストを消去せずにバッファにコピーするためのもので，コピーされたテキストは，〈PUT〉により，任意の位置に挿入できる．通常，消去コマンドや〈PICK〉コマンドの実行により，バッファはリフレッシュされてしまうが，コマンド・モディファイア〈META〉を用いると，バッファの中味はクリアされず，消去あるいはピックされたテキストはバッファの末尾に付加される．これによって，複数のテキストを一まとめにして移動したり，コピーしたりできる．

文字列のサーチについても，現在のカーソル位置から前方（後方）に向けて，指定された文字列をサーチする〈PSEARCH〉(〈MSEARCH〉) コマンドが準備されている．〈ARG〉"文字列"〈PSEARCH〉を実行すると，現在のカーソル位置より前方にある"文字列"で指定された文字列を探し，その先頭にカーソルを移動する．引数で指定された文字列は，テンポラリ・ファイルに記憶されており，単に〈PSEARCH〉を実行すると，最近にサーチされた文字列をもう1度サーチする．したがって，1度引数として文字列を指定してやれば，後は単に〈PSEARCH〉を実行することで同じ文字列の異なる出現を前方に順次サーチしてゆくことができる．〈MSEARCH〉の場合も同様である．また，〈PSEARCH〉と〈MSEARCH〉で記憶している最近にサーチされた文字列は共通であるから，まず〈PSEARCH〉で前方に順次サーチを行い，途中から急に〈MSEARCH〉で後方にサーチを行うといった

ことも可能である．引数として渡せる文字列には，正規表現を含めることも可能で，その場合は，〈ARG〉〈ARG〉"文字列パターン"〈PSEARCH〉というコマンド列でサーチを行う．この正規表現を含んだ文字列パターンのサーチは，非常に有用なことが多い．

また，Zのデフォルトのセッティングでは，サーチの際に文字列の大文字・小文字を区別するようになっているが，もしこれを区別せずにサーチしたいならば，IGNORECASE：というスイッチをセットすればよい．

サーチと関連した重要なコマンドとして，文字列の置き換えコマンドがある．Zでは，〈REPLACE〉と〈QREPLACE〉という2種のコマンドがあり，前者はファイル全体もしくはカーソル引数として与えられた領域（line や box）中のある文字列の出現を指定された文字列で置き換えてしまうのに対し，後者は，置き換えに際し，逐一ユーザの確認（置き換えるか否か，あるいはそれ以降は確認なしで置き換えてしまうか）を求めてくる．たとえば〈REPLACE〉を実行すると画面の下方のエコーラインにプロンプトが出され，どの文字列をどのような文字列に置き換えるかが問われる．〈ARG〉"カーソル引数"〈REPLACE〉というコマンド列により，置き換えを行うファイル上の範囲をカーソル引数（line, box）で指定できる．また〈ARG〉〈ARG〉〈REPLACE〉というコマンド列により，置き換えられる（サーチされる）文字列を正規表現を含めて指定できる．この場合も，カーソル引数により，置き換えを行う範囲を指定できる．

ワード・プロセッシング・コマンドとしては，テキストの整列のための〈FLUSHLEFT〉（左端をカーソル位置にそろえる）と〈FLUSHRIGHT〉（右端を右マージンの位置にそろえる）といったコマンド，テキストのセンタリングのための〈CENTER〉コマンド，各パラグラフを右マージン以内に納まるようにフォーマッティングする〈FILL〉，右マージンの位置でぴったりと各行が終るようにする〈JUSTIFY〉などが準備されており，ドキュメント・エディタとしても有用である．これらの機能に加えて，Zにはウェブスター辞書（約4万語収録）を用いた優れたスペル・チェッカが準備されており，〈SPELL〉コマンドにより呼出される．このスペル・チェッカは，辞書にない語（ミス・スペル語の候補）を発見すると，その語を，修正するのか，そのままにしておくのか，あるいは辞書に登録するのかをユーザに問合わせてくるようになっている．修正アルゴリズムは，かなりよくできたもので大抵の場合すぐに正しいスペルが得られる．また，スペル・チェックのアルゴリズム自体もかなり高速であり，快適にスペル・ミスの検出・修正ができる．

3.2 プログラム編集用コマンド

最初にも触れたが，Zはプログラムをテキストとして見るという基本思想の下で作られており，ある特定の言語の構文に依存していない．Zでは各言語の構文上の特徴だけを分類して持っており，インデンテーションやバランシング等，テキストとしてプログラムを見た場合に最も基本的な支援を行うようになっている．なお，ファイル名のエクステンション部分の記述によって使用される言語を示すことができる．たとえば .LSP (LISP)，.PAS (PASCAL)，.MAC (MACRO)，.RNO (RUNOFF) などの指定が可能である．

まず，Zの自動インデンテーションだが，これはタブとバックタブを用いて実現されており，各言語ごとにどのトークンのときにタブを出力し，どのトークンのときにバックタブを出力するか等の情報がテーブルとして保持されている．たとえば BEGIN～END，WHILE～DO などの構文要素のインデンテーションを自動的に行う．自動的にという意味は〈newline〉コマンド (CR) が入ってくる度にタブやバックタブの数を計算し，自動的に挿入するということである．

LISP のように構文的に極立った要素がない言語の場合は少し複雑で，LISP の場合は，もっとも最近にバランスした（閉じた）S式の左カッコの位置を，現在のカーソルの位置から逆向きにサーチして求めている．このように単純な方式だが，十分に役立つインデンテーション機能が実現されており，プログラムの構造を視覚的にとらえることができる．

バランシングについては，ブロック構造やカッコが既に閉じていれば，そのマッチングを示し，もし開いていればそれを閉じることが必要となる．このために，〈Balance〉コマンドが用意されており，カーソル位置にバランス表現を閉じる文字（例えば ')' ）があればそれに対応する開く文字（例えば '(' ）の位置を示し，もし閉じる文字がなければ，もっとも最近に開かれたバランス表現を閉じるという機能を持っている．このコマンドでは，左右のカッコ，BEGIN～END などのバランス表現を扱いうる．このバランシング機能は，LISPの S-式や PROLOG の項のよう

に，複雑でネストした表現が可能な場合には必須のもので，テキスト・ベースのプログラム編集においても，最少限必要な構文的なサポート機能である．

また，Zには〈COMMENT〉コマンドがあり，言語に応じて，プログラム中にコメント記号を挿入できる．

以上に述べたプログラム編集用コマンドは主に，プログラム言語の構文に依存した支援を与えるものであるが，Zにはプログラムのコンパイルをサポートする機能も準備されており，エディタ内から各言語のコンパイラを呼び，現在編集中のファイルをバックグラウンド・ジョブとしてコンパイルすることができる．エディタとバックグラウンドのコンパイルは非同期通信により結合されており，コンパイラの出力は解析され，もしエラーメッセージであれば，そのエラーに対応するソース・ファイル中のポジション(行と列番号)が記録される．コンパイルが終了するとエディタにシグナルが送られ，ユーザにそれが通知される．その後，先に記録された情報を基に，ソース・ファイル上で，コンパイル・エラーの起った個所をファイルの頭から順番に見ることができる．これは非常に便利な機能で，編集して，コンパイルして，エラー・メッセージをチェックして，再びエディタに入ってエラーを修正するといった不経済な繰返しを取除くことができ，すべてエディタの内で，しかも，ダイナミックにエラーの起った場所がチェックできるわけである．〈ARG〉〈COMPILE〉により，現在編集中のファイルがコンパイラに渡されコンパイルが開始される．コンパイル終了後は，〈COMPILE〉コマンドにより，ソース・ファイル中のコンパイル・エラーの起った個所に頭から順番にカーソルが移動される．どの言語のコンパイラが選ばれるかは，インデンテーション等の場合と同様，ファイル名のエクステンション部分により決まる．

以上で述べたメカニズムは，バックグラウンド・ジョブに引数を渡し，かつバックグラウンド・ジョブの出力を解析し，それを記録しておくというもので，コンパイラに限らず，いろいろな用途で用いられる．バックグラウンド・ジョブの出力情報がエディタで利用できるためには，そのフォーマットが次のようになっていればよい．

“ファイル名”，“行番号”，“列番号”，“メッセージ”

この情報を基に，エディタは，“ファイル名”で示されたファイルの“行番号”と“列番号”で示された位置にカーソルを移動し，バックグラウンド・ジョブのメッセージをファイル上の対応する位置にダイナミックに反映させることができる．この機能を用いた他の例として，正規表現を用いた文字列パターン検索がある．これは，TOOLSのプログラムFGをバックグラウンドで起動するもので，たとえば

〈ARG〉FG “文字列” ＊.TXT〈COMPILE〉

というコマンド列により，.TXTというエクステンションを持つファイルすべてについて，“文字列”(正規表現を含む)で示されるパターンの出現がすべて検索され，検索終了後，〈COMPILE〉コマンドにより，すべてのファイル上の“文字列”の出現位置にカーソルを順次移動してゆくことが可能である．

こういった機能は，ファイルに関する操作の結果等を，そのファイル自体の上で，エディタでチェックできるという環境を与えてくれるわけで，非常に便利で分かりやすいユーザ・インタフェースの実現と言えるだろう．このような，対話性に優れたインタフェース機能の拡張として，次項では，セッション・マネージャ機能について概説する．

4. セッションマネージャ (SM)

従来のプログラミング環境の欠点として，エディタや言語処理系あるいはOSのコマンド処理系などのコンポーネント間の通信手段の不備と，プログラミングの過程の履歴を見る手段の不備ということが挙げられる．このことは，プログラミングに限らず，ドキュメントの作成のような場合にもあてはまる．特に，LISPやPROLOGのようなインタプリタ・ベースの言語の場合には，エディタとインタプリタの間を自由に行ったり来たりできること，インタプリタとの対話の履歴を管理できることは，ぜひとも欲しい機能である．たとえばLISPマシンのプログラミング環境[4]やINTERLISPのprogrammer's assistant[5]はこのようなプログラミング環境の例と言える．YALE-TOOLSでは，これらの要求を，マルチ・プロセスを管理するメカニズムとTTYディスプレイ上へのプリントアウト結果をファイルとして管理するエディタ（Z）とを併用することにより解決している．

マルチプロセス管理はMUF (Multiple User Fork)というプログラムにより行われる．MUFの下には複数のフォーク（子プロセス）が生成可能で，それら子プロセスのそれぞれの識別記号として，アルファベット1文字を割当てる．これらの子プロセスは，一時には1つのプロセスだけが実行され，他のプロセスはサ

スペンドされているか，バックグラウンドで非同期モードで実行されているかという状態にある．どのプロセスを実行するか，すなわちプロセスのスイッチングを，識別記号に対応するコントロール文字(例えばAに対しては^A) をキーボードから入力することで行うことができる．たとえば，Zエディタと PROLOG インタプリタを MUF の下の子プロセスとして生成しておけば（図-4 参照），プログラムを編集して，すぐさま PROLOG インタプリタに移り，それを実行してエラーが見つかれば再度エディタ・プロセスにスイッチしてバグを訂正するというように，エディタとインタプリタが協働する環境を実現することができる．このように，簡便にプロセス・スイッチングができることは，プログラミング環境として重要な機能といえよう．例えば LISP マシンもマウス・メニュを用いて，同様な環境をユーザに提供している．

この MUF を用いたプロセス・スイッチング機能だけでは，各プロセスのディスプレイ出力を管理することができず，たとえばインタプリタとの対話状況の履歴を見ることが不可能となる．そのため，Zには，セッション・マネージャ (SM) 機能が準備されている．

SM の基本構成は，図-4 に示すように，Zの下に MUF を用いて複数プロセスを管理するとともに，各プロセスに対するキーボード入力，あるいは各プロセスのディスプレイ出力のすべてをファイル化し，エディタを用いて管理するものである．

SM においては，Zは edit モードと SM モードの2つのモードを持つ．edit モードは，〈SM〉コマンドにより SM モードに入るということ以外は通常のエディタと同じである．SM モードは，キーボード入力をトランスペアレントに現在実行されている子プロセスに送り，また子プロセスの出力をディスプレイに表示する．これらのターミナル入出力はすべて，SM が管理するログ・ファイルに書き込まれる．SM モード

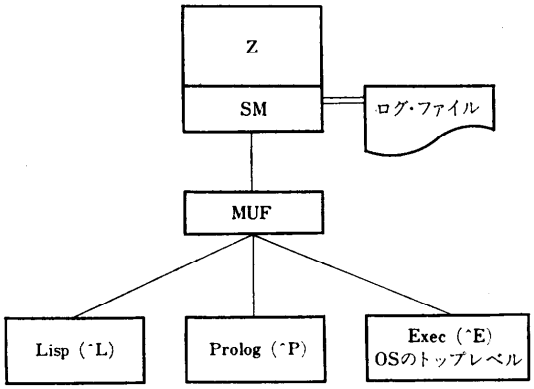
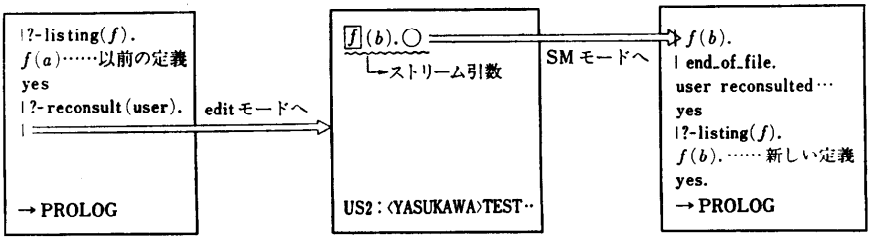


図-4 セッション・マネージャの構成

において〈SMESCAPE〉コマンドが入ると，Zは SM モードから抜け，edit モードに入り，ログ・ファイルを含め，任意のファイルを編集することができる．SM モードと edit モードの間の通信，言いかえれば MUF の下の子プロセスと SM として機能しているZエディタの通信は，単純な方式で実現される．すなわち，edit モードで〈ARG〉"引数"〈SM〉というコマンドを入力することで，"引数" で指定されたテキストをそのまま SM モードの入力，すなわち，現在実行中の子プロセスの入力とすることができる．"引数" としては，text 引数，stream 引数，line 引数，box 引数，null 引数が可能で，ログ・ファイルを含め任意のファイル中の任意の文字列を必要に応じて修正し，子プロセスに引渡すことができる．たとえば，SM モードにある子プロセスの PROLOG インタプリタが述語再定義モード (reconsult) にいる時に，edit モードに入りソース・ファイル中のある述語の定義（プログラム）を修正し，その結果を引数として SM モードに戻れば修正後の述語定義が PROLOG インタプリタに渡され，その述語の再定義が行われる．図-5 は，その一例を図示したもので，$f(a)$ で定義さ



(a) SM モード (再定義前)　(b) edit モード　(c) SM モード (再定義後)

図-5 SM 機能を用いたプログラムの修正例

れていた述語 f を $f(b)$ に修正している．editモードからストリーム引数“$f(b).$”を持ってSMモード(PROLOG)に戻っている．また，デバッグの途中では，同じようなコマンド（プログラム呼出し）を何度も繰返し行うことはよくあることだが，この場合も，ログ・ファイルを編集し，適当なテキストを引数として，子プロセスに渡すことにより簡単に対処できるし，またトレーサやステッパの出力も，たれ流しで出力させておいて，あとからログ・ファイルをエディタで見ることにより，全体を通してチェックすることができる．

TOOLSでは，以上に述べたようにMUFとZをうまく組み合わせることにより，ZをOSのトップレベルや言語処理系のトップレベルのユーザ・インタフェースとして利用することができ，非常に便利で軽快なプログラミング環境と言えるだろう．

図中の（a）は再定義前の状態で，一番下に“→PROLOG”とあるように，現在MUFの子プロセスのPROLOGインタプリタのレベルにある．“listing（f）”は現時点での述語“f”の定義を出力するもので，この時点で“f”の定義は“$f(a).$”となっている．“reconsult (user),”というコマンドにより，述語の再定義をターミナル入力から行うモードにしておき，〈SMESCAPE〉コマンドを実行し，editモードに入る．

(b)は，editモードで，TESTというファイルを編集しているところを示す．述語“f”の定義を“$f(b).$”に変更し，“$f(b).$”をストリーム引数として，〈SM〉コマンドを実行し，SMモードのPROLOGインタプリタに戻る．

(c)は，editモードから渡されたストリーム引数“$f(b).$”が述語再定義モードの入力となっている状況を示す．図中の“end-of-file”は，再定義モードの終了を示すものである．元は“$f(a)$”であった述語“f”の定義が“$f(b)$”に変わっていることが示されている．

5. おわりに

以上，駆け足でZの主な特徴を説明してきたが，十分に説明ができたとは言えないかもしれない．エディタはプログラミング環境におけるもっとも重要なユーザ・インタフェースと考えると，やはりエディタの良し悪しというのは実際に使い込んでみないと分からないと言えるだろう．そのような意味では，ZはYALE大学を中心に，かなりのユーザに使い込まれたエディタであり，MITのEMACSなどと並んで代表的な優れたエディタと言える．

Zにおいて，最も魅力を感じるのは，やはりSMである．最近LISPマシンなど優れた対話型プログラミング環境を持つマシンが注目を集めているが，TOOLSでは，Zをユーザ・インタフェースとして利用し，従来のTSS環境の上で優れた対話的環境を実現しているわけで，今後のエディタの在り方やTSS上のプログラミング環境の在り方に1つの指針を与えていると言ってもよいだろう．

Z (YALE-TOOLS) の利用できる環境は，国内では，ICOT，電総研，武蔵野通研などのDEC-2060システム上と，かなり数が限られており，実際に利用する機会は少ないとは思うが，ぜひ機会を見つけて使ってみられると良いと思う．

参考文献

1) Ellis, J.R., Mishkin, N., Leunen, M. and Wood, S.R.: Tools: An Environment for Timeshared Computing and Programming, Research Report #232, Yale Univ., Department of Computer Science, New Haven, Connecticut (1982).
2) Wood, S.R.: Z—The 95% Program Editor, Proc. of the ACM SIGPLAN SIGOA Symposium on Text Manipulation, Portland, Oregon, pp. 1-7 (1981).
3) Stallman, P.M.: EMACS Manual for TWENEX Users, AI Memo 555, MIT AI Lab., Cambridge, Massachusetts (1981).
4) Weinreb, D. and Moon, D.: LISP Machine Manual 4-th Ed., Symbolics Inc., Cambridge, Massachusetts (1981).
5) Teitelman, W.: A Display Oriented Programmer's Assistant, Technical Report SSL-79-9, Xerox PARC, Palo Alto, California (1977).

(昭和59年5月4日受付)