AN INTRODUCTION TO RELEASE 1 OF EDITOR ALLAN POE

by

C. N. Fischer
G. Johnson
J. Mauney

Computer Sciences Technical Report #451

December 1981

An Introduction to Release 1 of Editor Allan Poe[1]

C. N. Fischer

G. Johnson

J. Mauney

University of Wisconsin-Madison
Madison, Wisconsin 53706
USA

[1]Research supported in part by National Science Foundation Grant MCS78-02570

Introduction

Editor Allan Poe (Pascal Oriented Editor) is a full-screen display editor that knows the structure and rules of Pascal. It is similar in approach to a number of language-based editors (LBEs), including ([ABL81], [AC81], [DHK75], [TRH81]). As a program is entered or modified, Poe automatically structures ("prettyprints") the program and checks it for correctness. Errors are noted and incorrect constructs are marked.

Unlike many LBEs, Poe is designed to be used by both novices and experts. Although a rich set of user commands is available to the expert, the novice needs to learn only a few, easily-remembered commands.

At all times, a "window" into the program is displayed. When development of a new program is begun, the following program prototype is displayed:

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
{LABELS}
{CONSTANTS}
{TYPES}
{VARIABLES}
{PROCEDURES}
BEGIN
    {STMT LIST}
END .
```

Poe displays three kinds of symbols:

(1) Required prompts

These are delimited by "<" and ">" (e.g., <ID>, <FILE ID LIST>). Required prompt symbols are placeholders that must be expanded to obtain a valid Pascal program. The expansion expected by such placeholders is suggested by their names. Thus <ID> should be expanded into an identifier and <FILE ID LIST> should be expanded into a list of file identifiers.

(2) Optional prompts

These are delimited by "{" and "}" (e.g., {VARIABLES}, {STMT LIST}). Optional prompt symbols are placeholders that may be expanded to produce a Pascal construct. If an optional prompt is not expanded, it is "erased", indicating that the suggested construct is not needed in this particular program.

For example, {VARIABLES} marks the place at which program variables can be declared. However, since a Pascal program may use no variables, it is legal to ignore this symbol in developing a program.

(3) Pascal symbols

These are the ordinary symbols found in Pascal (identifiers, numbers, reserved words, etc.). For emphasis,

reserved words are shown in upper case. When a complete Pascal program has been created, only Pascal symbols remain.

To create a Pascal program in Poe, you need only expand optional and required prompts. This is very easy to do. You merely move the cursor to the prompt and type any expansion that agrees with the prompt. Thus if you moved to an <ID> prompt, you could type abc or xxxx or any other valid identifier[2]. Similarly, if you moved to a {VARIABLES} prompt, you could type any legal variable declaration (e.g., VAR i:integer).

Cursor movement is controlled using the usual cursor control keys[3]:

(1) The space bar moves the cursor one symbol right.

(2) The backspace key moves the cursor one symbol left.

(3) The return key moves the cursor to the leftmost symbol of the next line.

---

[2]Poe will allow an undeclared identifier to be entered, but will highlight it until it is properly declared.

[3]"Arrow" keys found on many terminals are not used because they are not standard and not always available. Particular implementations of Poe may allow the use of such keys as an extension.

(4) The "\" key[4] moves the cursor to the leftmost symbol of the previous line.

As a Pascal symbol is entered, a prompt symbol may be replaced by new symbols, representing the detailed structure of a construct. Thus if in the above example, the cursor were to be moved to the {STMT LIST} prompt, and "if " is typed[5] the following structure results:

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
BEGIN
    IF <EXPR>
    THEN {STMT}
    {ELSE CLAUSE} ;
    {MORE STMTs}
END .
```

Note that since a THEN is always created when an IF is recognized, it is impossible to create ill-formed IF statements. In fact, in Poe syntactically incorrect program structures (of any kind) can never be created.

But what if the user were to type something that is illegal at the point at which the cursor is positioned? For example, a THEN (which cannot begin a statement) might be entered at a {STMT} prompt, or BEGIN might be typed at the

---

[4]This choice is arbitrary. Any character that cannot begin a Pascal symbol could have been used. The two-character command !^ is a synonym.

[5]The blank after the "if" is needed so that the editor can distinguish between the symbol "if" and a pause in the entry of, e.g., "iff".

very top of a program. Rather than considering these errors, Poe uses an error-repair algorithm to place all symbols, as they are entered, in their "most reasonable" program context. Thus typing a THEN at a {STMT} prompt will expand the prompt into an IF-THEN construct, with the cursor immediately following the THEN. Similarly, entering a BEGIN at the top of a program will move the cursor beyond the nearest BEGIN.

This approach makes Poe fairly forgiving in the entry of program text. But what if the repair chosen by Poe is not what the user wants? Unwanted constructs can be deleted by using the delete key. Note that some individual symbols (such as IF) may not be deleted (since an illegal syntactic structure might result). Thus the effect of hitting delete is to delete the smallest structure containing the symbol marked by the cursor and to replace it by the appropriate prompt. Hitting delete more than once will delete progressively larger constructs. For example, consider the following program, with the cursor on "writeln":

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
BEGIN
    writeln ( 1 ) ;
END .
```

Delete is hit to obtain:

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
BEGIN
    <VAR ID> ( 1 ) ;
END .
```

Hitting delete again yields:

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
BEGIN
     {STMT} ;
END .
```

Repeatedly hitting delete then produces the following sequence:

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
BEGIN
    ;
END .
```

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
BEGIN
    {STMT LIST}
END .
```

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
BEGIN
END .
```

```
<PROGRAM>
```

At this point no further deletion is possible (or meaningful).

As will be discussed later, Poe has a very general "undo" capability that can be used to undo unwanted commands. However, the novice need not know of this command. In fact,

beyond knowledge of how to move the cursor, and how to enter and delete symbols, only two other points need be known to the novice Poe user: how to initiate an editing session and how to terminate it.

The details of invoking Poe are operating system dependent. Usually only a file name (denoting the file to be edited) needs to be supplied. If the file exists, it is edited. If it does not exist, creation of a new program, to be written into the file, is assumed. In Unix, Poe is invoked as:

```
poe filename
```

To terminate an editing session, the user types "!x". The symbol "!"[6] signals a Poe command. The x represents "exit". As we shall discover below, a wide variety of Poe commands (all beginning with !) are available.

Poe allows an editing session to be terminated, even though a program is incomplete or semantically incorrect. In either case, it prints a warning message prior to termination. Incomplete programs can aways be recognized by the fact that required prompts remain unexpanded. Semantically incorrect programs can be recognized by the fact that portions of the program are highlighted. Moving the cursor to a highlighted symbol will cause a message detailing the

---

[6] Escape is a synonym of !.

error to be printed. If all required prompts are expanded, and no program components are highlighted, the program being edited is guaranteed to be a legal Pascal program.

At this point the reader is invited to experiment with the Poe features discussed so far. While these features are minimal, they are sufficient to make productive use of the editor.

Advanced features of Poe

The minimal set of features required to use Poe productively allows both novices and experts to use the editor with little training. Naturally, however, as a user becomes more experienced, a richer set of commands is desirable. We now discuss some of the advanced commands available in Release 1 of Poe. Note that in future releases these commands may be significantly changed and augmented.

Help facilities

The symbol "?" can always be used to get help or extra information. Normally, it lists a complete menu of available commands. When the cursor is placed on an identifier, ? shows the corresponding definition of the identifier. In future releases, ? will be used to obtain information about error diagnostics, language features and editor commands.

Cursor movement

In addition to the space bar, backspace key, return key and \ key, a number of other cursor control functions are available:

(1) !b

Go back one screen

(2) !f

Go forward one screen

(3) !d

Go down one half screen

(4) !g

Go to top of program

(5) !G

Go to bottom of program

(6) !t

Go to top of screen

(7) !B

Go to bottom of screen

File manipulation and termination

A number of commands that manipulate files and/or terminate editing are available:

(1) !r filename

Read contents of specified file beginning at cursor position.

(2) !w filename

Write current program into specified file.

(3) !q

Quit editing session. If the program has been changed since the last write, the user is queried before editing is terminated.

(4) !x filename

Exit editing session. Equivalent to a "!w filename" followed by a "!q" If no filename is given, the file named on the editor call is used.

Undoing editor commands

It is essential (particularly to novices) that errors not be irredeemable. To this end, Poe makes available a very general "undo" capability. At any time, the most recent Poe command can be undone by typing !u. Entering !u n times causes the most recent n commands to be undone[7]. The undo command can't undo other undo's (else

[7]The number of commands that can be undone is limited by the depth of an internal "history stack". At present, the maximum depth of this stack is 10. The history stack may be displayed by entering !h.

multiple undos wouldn't work as expected). An undo can be undone by entering !U.

For example, assume that at a {STMT LIST} prompt we enter "writeln(1,2 " followed by "backspace delete". We then have the following line:

writeln (1 , <PARAMETER EXPR> {, PARAMETER LIST}) ;

If we now enter a sequence of !u commands, the following lines are produced:

writeln (1 , 2 {MORE EXPR} {, PARAMETER LIST}) ;

writeln (1 , <PARAMETER EXPR> {, PARAMETER LIST}) ;

writeln (1 {, PARAMETER LIST}) ;

writeln (<PARAMETER LIST>) ;

writeln {:= EXPR or other stmt} ;

{STMT LIST}

If we now enter a sequence of !U commands, the sequence of lines is reversed to obtain

writeln {:= EXPR or other stmt} ;

```
writeln ( <PARAMETER LIST> ) ;

        .
        .
        .
```

In general almost any command can be undone, so when in doubt it is safe to try a command and see what happens.

Prompting commands

The use of prompt symbols is central to the design of Poe. At any point, a prompt symbol reminds the programmer what sort of input is expected. But what if a user doesn't know what a given prompt symbol can be expanded to? Poe provides an automatic means of "exploring" the possible expansions of a symbol. If !p is entered while the cursor is on a prompt symbol, a possible expansion of that symbol is shown. If !p is entered again, another expansion is shown, until all possibilities are explored. Thus if the cursor is at a {STMT} prompt and !p is entered, we are shown:

```
{STMT} -->          (nothing)
```

This shows that a statement is optional at this point. Hitting !p again produces:

```
{STMT} --> {LABEL} {UNLABELED STMT}
```

This shows that an optional label followed by an

optional unlabelled statement is another possibility. Hitting !p again returns to the first expansion since there are only two possibilities.

When the programmer sees the expansion he desires, an !e may be entered to expand the prompt symbol according to the expansion currently illustrated. The user may then enter !p again to explore further choices. Thus a lot about the structure of Pascal can be learned by merely exploring via the !p and !e commands.

The handling of optional prompts requires special care. If all optional prompts were displayed at all times, the screen would quickly be cluttered. Therefore Poe suppresses the display of optional prompts once the cursor has moved past them. Note that it is still possible to expand these symbols; they are merely "made invisible" to make the display more succinct. For example when editing of a new program is begun, the cursor is at the PROGRAM symbol, and a number of option prompts are displayed:

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
{LABELS}
{CONSTANTS}
{TYPES}
{VARIABLES}
{PROCEDURES}
BEGIN
    {STMT LIST}
END .
```

If the cursor is moved to the END symbol, most of the

optional prompts are suppressed:[8]

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
BEGIN
    {STMT LIST}
END .
```

Moving the cursor back to the PROGRAM symbol does not redisplay the suppressed prompts[9].

Sometimes it is convenient to force nearby prompts to be redisplayed. This can be done by entering !a. The !a command is especially handy when a number of adjacent optional prompts have been suppressed, and the programmer wishes to move the cursor to one of these. Optional prompts that have been explicitly deleted are not redisplayed by the !a command. However, such prompts can be retrieved by the !A command. The !A command does everything the !a command does as well as redisplaying deleted prompts.

For novices, who are unsure of what symbols are appropriate at a given point, an automatic redisplay of optional prompts can be obtained. This is done by entering !P (entering !P again returns to the original

---

[8]The {STMT LIST} prompt isn't suppressed because the cursor hasn't moved a full line past the prompt.

[9]In earlier versions of Poe this was done. This led to rapid redisplay and suppression of symbols as the cursor was moved. Users found it very annoying.

mode). In the redisplay mode all optional prompts near the cursor are shown. Thus if a programmer is unsure what is expected at a given point, he can move the cursor to that point and see what prompts appear. As noted above, the price of this automatic redisplay is a significant amount of screen repainting as the cursor is moved.

Elision of program structure

One of the great problems in developing large programs at a display terminal is that the screen rapidly becomes filled with minor detail. Poe allows users to elide program structure to enhance readability. Consider the following partially completed program:

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
PROCEDURE p ( i : integer ) ;
    BEGIN
      IF NOT eof
        THEN BEGIN
            writeln ( 'aaaa' ) ;
        END
    END ;
BEGIN
    {STMT LIST}
END .
```

Structure can be elided by moving the cursor to a symbol and entering "!>". Typing !> at the BEGIN within the IF statement would yield:

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
PROCEDURE p ( i : integer ) ;
    BEGIN
      IF NOT eof
        THEN <BEGIN - END...>
    END ;
BEGIN
    {STMT LIST}
END .
```

The symbol <BEGIN - END...> represents an elided BEGIN-END block. Entering !> at the IF will elide the entire IF statement:

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
PROCEDURE p ( i : integer ) ;
    BEGIN
      <IF-THEN-{ELSE}...> ;
    END ;
BEGIN
    {STMT LIST}
END .
```

Similarly, entering !> at the PROCEDURE will elide the entire procedure:

```
PROGRAM <ID> ( <FILE ID LIST> ) ;
<PROC OR FUNC...> ;
BEGIN
    {STMT LIST}
END .
```

Elided structure can be "unelided" (i.e., expanded) by entering !<.

The problem of elision in language based editors is currently receiving much attention [Mik81]. Future releases of Poe will undoubtedly provide enhanced elision capabilities including automatic elision, elision

by comment (in which a comment is "tagged" to a structure) [TRH81] and "first line elision" (in which the first line of a structure elides the entire structure).

Moving structures

Like all editors, Poe provides the ability to move text within a program. However in Poe only complete, syntactically valid structures can be moved. Thus a procedure, or a statement, or even an identifier can be moved, but an IF symbol, or a single "(" cannot. To copy a structure, the Poe user moves the cursor to a symbol in the structure to be copied. The command !c will copy the smallest legal structure containing the symbol marked by the cursor into an unnamed tree. Similarly, "!c name" will do a copy into tree with the given name. If a copy of a larger containing structure is desired, additional !c commands will copy progressively larger structures, up to the entire program. Thus given "a := b + 1", with the cursor on the b, !c would copy just the b, then the right hand side expression, then the entire statement (depending on the length of the !c sequence).

It is also possible to delete a structure into a subtree. This is done by !delete[10] or !delete name. As in

[10] Delete represents the delete key.

the case of !c, additional !delete entries with delete progressively larger structures.

The subtrees created by !c and !delete can be examined and even edited. In fact, at any point the Poe user has access to a forest of Pascal structures. The command !) moves the user to a directory representing all the subtrees currently extant. To examine (or edit) one of these trees, the user moves the cursor to the desired tree. The !(command then takes him into that tree, where it can be examined or edited. To return to the main program, !m may be used. Similarly, !s moves to the most recently visited or created subtree.

To insert the structure represented by a subtree, the cursor is moved to a suitable prototype and the !i command is used. If a name is given (e.g., !i name), the appropriately named subtree is inserted. If no name is given, the most recently created or edited subtree is used. Note that for an insertion to be allowed, the subtree to be inserted must agree in structure with the prompt the cursor is at. Thus at a {STMT} prompt an IF statement structure could be inserted, but an expression or procedure body structure could not.

Miscellaneous commands

Although Poe is primarily an editor, it has the capability to execute a Pascal program. This makes easy the usual edit/test/edit... cycle. The !X command initiates execution of a Pascal program. Execution is only allowed if a program is complete (all required prompts are expanded) and semantically correct (no symbols are highlighted). Program input is accepted from the keyboard, and program output appears at the bottom of the screen. After execution, the program is redisplayed and editing can be performed. If execution terminated abnormally, the cursor is placed on the line deemed in error, and the error message is displayed.

Because Poe is an experimental system, it is inevitable that users will discover bugs and have ideas and comments they wish to contribute. To facilitate this, a "complaint to the management" feature is provided. If !C is entered, you are turned over to the system mail facility to enter a message to be shipped to the Poe implementation group. A copy of the program being edited may also be sent. You are then returned to the editor.

As new features are implemented, a "message of the day" (the beginning of which is displayed at the start of an editing session) notes information of interest. This

message may be viewed at any time by entering !M.

It is often handy to suspend editing and return (temporarily) to the system command interpreter. This can be done by entering !%. After the command interpreter is exited[11] editing can be resumed.

Poe is designed to support a variety of display terminals[12]. To inform Poe of the type of terminal being used, the command "!T termtype" can be used. This command also displays the kind of terminal it currently believes is being used. Terminal type can also be controlled when Poe is called; e.g. typing "poe -Tmime filename" invokes Poe on a file, and immediately sets the terminal type to "mime".

To force the display to be repainted, "control L" may be used[13].

Poe commands may take a repeat count: !nn<cmd> will perform <cmd> nn times. Repeats apply to the cursor movement commands as well: !5\ move the cursor back five

---

[11] In the Unix implementation of Poe this is done by entering "control d".

[12] At present, the HP2621A, Visual 200, H19, and Mime-I terminals are supported.

[13] This command is a vestige of similar usage on earlier, less advanced editors. A synonym of !L will be added for uniformity of command syntax.

lines; it is important that no space come between the count and the command, unless space (move cursor right) is the command you wish to repeat. It is not possible to repeat backspace; instead, backspace is used to correct errors in the repeat count.

Implementation notes and current research

Poe is written in Pascal (to ease transport to new systems). Parsing is done via an extended LL(1) technique that provides direct support for lists of symbols (this is essential to allow insertion and deletion anywhere within a list). Error repair is done via the FMQ [FMQ80] technique. Semantic information is represented as attributes on an abstract syntax tree. Special linkages (e.g, of all uses of an identifier) are employed to ease semantic checking. Algorithms that percolate attribute values within a tree are used to transmit information within a program[14].

Current work on Poe centers on both enhancing user features and improving implementation size and speed. Work on structure elision, semantic error repair and implementation-dependent language features (e.g, separate compilation) are receiving careful study. Future releases of Poe will reflect advances in these areas. Ways of reducing the size

[14] These algorithms are similar to those presented in [DRT81] and [Rep81].

of internal data structures and speeding program analysis (especially attribute analysis) are under study. A non-symbolic external representation of programs (to be stored in files) will speed editing of existing programs. Background processing of attribute information will allow idle time to be effectively employed, while providing excellent response time. Problems in transporting Poe to stand-alone personal computers will be studied. Retargeting Poe to languages other than Pascal will also be investigated.

References

[ABL81] Alberga, C.N., A.L. Brown, G.B. Leeman Jr., M. Mikelsons and M.N. Wegman, A program development tool. 8th POPL Conference, 92-104, 1981

[AC81] Archer, J., and R. Conway, COPE: a cooperative programming environment. Cornell U., TR 81-459, 1981.

[DHK75] Donzeau-Gouge, V., G. Huet, G. Kahn, B. Lang and J. Levy, A structure oriented program editor: a first step towards computer assisted programming. IRIA Laboratories, Technical Report 114, 1975.

[DRT81] Demers, A., T. Reps, and T. Teitelbaum, Incremental evaluation for attribute grammars with application to syntax-directed editors. 8th POPL Conference, 105-116, 1981

[FMQ80] Fischer, C., D. Milton and S. Quiring, Efficient LL(1) Error Correction and Recovery Using Only Insertions. Acta Informatica, 13, 2, 141-154, 1980.

[Mik81] Mikelsons, M, Prettyprinting in an interactive programming environment. Sigplan Notices 16, 6, 108-116, 1981.

[Rep81] Reps, T., Optimal-time incremental semantic analysis for syntax-directed editors. Cornell U., TR 81-453, 1981.

[TRH81] Teitelbaum, T., T. Reps and S. Horwitz, The why and wherefore of the Cornell Program Synthesizer, Sigplan Notices 16, 6, 8-16, 1981.