MINCE: Not Just Another Editor

by Barry A. Dobyns

As computing power gets cheaper and cheaper, we approach the problem of getting useful data into computers in a fashion that is acceptable to the user of the computer. Programmers in particular are going to be under increasingly greater pressure to be more and more productive. Their ability to deal with the problem of getting the solution out of their heads and into the computer will be influenced in great measure by the ease with which they can physically enter data into the machine. Electronic communications, still a budding field in many ways, will have its success measured in part by the ease with which the user can use the system, primarily the ease with which messages can be composed and edited.

With few exceptions, all the programs and data in every computer originated with someone typing to (at) an editor or parser of some sort. This is, I realize, a gross over-simplification, and there are cases which may be considered to be not connected with any editors at all (a friendly IBM 029 card punch, for instance), but the point is well made: in most cases someone uses a text editor of some sort to generate a source program (even DDT can be considered a crude sort of editor for hex), or to enter data, or to generate a program which may, itself, be a restricted type of editor or parser for data entry. A lot of users never consider using any editor other than the one which came with their system for the production of text and programs. Some realize they can improve their efficiency by picking another editor that is better suited to their requirements. Most users have video terminals of one sort of another, yet don't have an editor that takes advantage of the power available to them. This article is about a class of editors whose popularity is gaining, screen editors, and one of these for CP/M in specific, MINCE ($125.00, Scribble $125.00, AMETHYST $350.00, available from Mark of the Unicorn, P. O. Box 423, Arlington, MA 02174).

Screen Editors in General

Simply put, screen editors are a class of editors that are meant to take advantage of the fact that CRT's (Cathode Ray Tube terminals) don't use paper. The major advantage of this is that the image on the screen can be changed, modified, and updated quickly. This is in stark contrast to paper terminals, where any modifications usually require that the entire element to be modified be printed out again rather than just changing the part that was modified; or worse, the editor requires that one remember the changes made, until one specifically requires that the region which has been modified be printed out.

Many people are making the move from paper-terminal-oriented editors to screen-oriented editors which show the text being edited at all times. These editors show the changes in the text dynamically, so the user always knows what effect his changes have made. These usually allow the user to move about in the text with simple single character commands and allow modification of the text at the cursor. (Cursor: 1. the place on a terminal at which the next character will be inserted, usually a block or underscore, sometimes blinking. 2. the place in the text file where most commands will have effect next, sometimes called the Point usually the same as 1. in screen editors.) Many display editor users can dispense with numerous listings of their text, as it is often easier to enter the editor and use a search facility to find something than to peruse many pages of print. Other users may find that the act of composition itself is made much easier, rather than writing the text out on paper first, one can start by typing it into the editor, using the editor to update and revise it. In general, display editors are also easier to use, as the user can see the text he is editing and immediately see the effect of the commands he enters on it. On a printing terminal this usually requires a non-trivial amount of mental effort to remember what has happened, or results in a lot of wasted paper and time, especially if a lot of changes are to be made.

A simple type of screen editor could have merely five commands, and still be very powerful. These would be up, down, right, left, and delete (one character). The screen of the terminal would display the area of text around the cursor, called a "window" because it acts just as a window onto the text might, hopefully redisplaying it if the user moved off the portion of text currently shown on the screen. Usually the commands would be delegated to single characters easily typed from the keyboard (control characters are relatively easy to type on most terminals), and characters which are not commands should be self-inserting, that is they should insert themselves into the text at the place marked by the cursor. Most screen editors do that and a lot more, offering most of the usual search and replace facilities that other editors offer, and often many special features of their own. Some editors also offer an embedded text formatter which can be used for the production of "pretty" printed copy.

What Makes a Screen Editor Good

Unfortunately, even the minimum screen editor is not a simple program to write at all when one has to deal with more text than there is available memory. There is also the problem of getting already prepared text into the editor, and saving it when the user is done. How the editor deals with these problems and what other conveniences or inconveniences it offers to the user, determine in a large part the utility of that editor.

Herewith I examine the features that I believe a good editor should be endowed with (this is by no means the final word on this subject, you can still love ED all you like, I just don't use it myself anymore).

Firstly, it should be at least as fast as I am. I am by no means a speed typist. I flunked every typing course I took (three), and my technique at a keyboard has been described in numerous degrading ways that I will decline to mention. Still, if I am provided with a delete key (the correction key on those very expensive Selectrics does fine), I can make any teletype in the land tremble with fear and awe (not recommended, they tremble enough as it is). Insertion from the keyboard should never be interrupted by disk I/O, calculation, redisplay or Acts of God. I refuse to use any editor that loses characters that I type. However, I may be convinced to use it if it is merely slow in showing me what it is up to, but doesn't lose anything. On the other hand, my spooler causes my system to lose characters all the time. I do not like to see any piece of software choke to death. In particular, the editor should not choke on files over 200K long. Maybe you can always get along with very short files (much less than 64K), but I always seem to have all these files around that exceed 100K (source code for interpreters and

compilers can often get big). Indeed, there should be no limit to the length of the file you can edit other than the amount of storage you have available.

I want to be able to move about faster than one character at a time. Character movement commands in line, screenful, word, sentence, paragraph (or atom, statement, and procedure) increments are a must. There should be modes for different types of text. If I am editing Pascal instead of English I should be able to tell the editor (or insert a keyword at the beginning of the file as a comment that the editor will pick up on) this fact so that my paragraph (English) commands now work on procedures (Pascal) or defuns (LISP) or whatever. There should be commands for deletion of the same units as movement. I should also be able to specify a "region" not normally delimited by the regular units for destruction and other purposes like saving, copying and moving. Any command which kills more than one character at a time should be reversible, should I turn fickle. Having to return to the original file in the event of massive failure is only marginally acceptable (it is expected after catastrophic failure). I should not be forced to save the new text out to the same filename, (nor should backups always have the extension .BAK). I should be able to create more than one output file from a single editing session, without exiting and reentering the editor. I may want to create a number of slightly different files, all with different names from the same initial file. (Form letters the hard way, or ten programs which only differ by one line or so.)

Searching and replacement should be straightforward and related. Insertion from other files should be easy. If it is doing redisplay, and I give it another command which requires redisplay also, it should execute the second command before finishing the redisplay for the first (if I give it sixteen consecutive "next screenful" commands, I want the sixteenth screen, not all fifteen in between). All commands should take numeric arguments where appropriate (one should type a command for "5 line deletes" rather than typing "delete a line" five times). In fact, the single action version of each command may merely be the special case of the command without arguments. In addition, whole command sequences should be delimitable in order to accept repetition counts. No command primitive should require more than two characters (keystrokes). All potentially catastrophic commands (like deleting whole paragraphs, and saving files) should be several keystrokes long.

Command-naming strategies should be as internally consistent as possible. If they are to be mnemonically related, make them all that way. If they are to be positionally related, pick a home key that gives you enough space around it to do the job properly. My personal preference is for mnemonically related command structures, as they tend to be distributed all over the keyboard, and can actually be touch-typed on many terminals, whereas position-oriented schemes usually are impossible to touch-type, and cramp my hand. There is much to be said for an editor, all of whose commands can be touch-typed (that is, typed with the hands remaining in the "home" position on the keyboard). This is often a function of both the editor and the terminal/keyboard design (many keyboards have the "Control" key in an awkward place, and have repositioned many of the "normal" keys, like "DEL" and "BACKSPACE" so that they are hard to get to). In any event I am vehemently opposed to the use of "special function" keys that require the operator to move his hands away from the keyboard. The time lost by locating the function key and relocating the home key is much greater than the time saved by typing one less keystroke. Sufficient documentation should be built in so as to enable a new user who is familiar with editors of that type to be able to learn it quickly.

Special cases should not break the editor (a file made up of only one very long line, for instance, will break many editors; trying to edit object code usually will break an editor, too). Broken programs are not to be tolerated, unless I have authored the program.

The file produced by the editor should be compatible with other editors for the same operating system, and should not be filled with "silly" things. I do not like the idea of an editor doing its bookkeeping in the file being edited. "Soft" characters is one such concept that still gives me the creeps. The problem here seems to be that the editor/formatter can't tell where a paragraph begins or ends, so the user must put in a carraige return to insure that paragraph breaks are not "filled" over. Maybe I'm harsh, but "soft" characters seem to be an attempt to overcome a design oversight, rather than a feature. Why not give the user of the editor a command that "fills" the text out to a specified column, so the user can get the column he wants it filled to and "grind" his text all he wants? Maybe give the user a mode for input that automatically breaks lines off as they are input too, but keep non-ASCII garbage out. I have a need for a utility that does nothing but take "soft characters" and replace them with "hard" ones if PIP didn't do it already.

My, he doesn't want very much, you might say. Indeed, as long as I can pick and choose about what I want, I will pick lots of features designed to make life easier for me. I compose at the keyboard of my terminal, and I like having only one editor for everything (text, papers, documentation, programs, data). I want my editor and my formatter separate though, thank you, as (I can hope) the creation of two programs (an editor and a formatter) will result in a more powerful one of each, as there is more memory available if they don't have to be co-resident. Besides, I may have a number of formatters, one that is fast and clumsy, and one that is slow but fancy, and one that is worthless but I wrote it myself, so I use it anyway. It is difficult to separate the editor and the formatter in many of the do-it-all programs, especially if they do "bookkeeping" in your files. On the other hand, there is the case for several editors and only one formatter, but the need for separate formatters and editors is still valid.

Reality vs. Fantasy

Unfortunately, as you may have suspected, gentle reader, the perfect text editor will take infinite memory, an infinitely fast computer, and will be perfect to only one user. (No two users have ever agreed completely on what constitutes a perfect editor.) I often feel that there is no middle ground in computers at all (be it languages, editors, or hardware), which is fine since many of us depend on just that fact for our livelihoods. As it is, many of the editors now available for CP/M-based 8080/Z80 computers are very good, even by the standards of what is available for many "big mainframes." (Anyone who has ever used the CMS editor on a video terminal and agonized about the fact that there wasn't anything any better understands.)

At some point (usually very early,

before any code ever gets written) the designer of an editor is going to make some very important trade-offs and some decisions that will affect the nature of his editor. One way to protect against making the wrong choice is to put these decisions off as long as possible, i.e., don't start with a command set in mind that you may not want to use later, but build a command dispatch table so that you can link any of the commands (like Control-P) with any of the procedures that actually accomplish the work you want done (like Previous-Line). Many of these sort of choices, if deferred long enough, can be incorporated into a set of configurable options that the user can use to make the designer's version of a "perfect" editor into his own version of perfection.

Most of us can't afford infinite memory, but our editor and operating system can often be expected to gobble up as much as 32K, and that's not very much space left to edit in. The editor needs to make some very smart decisions about what part of the file to read in, and when, if it is to give the user the feeling of having the entire file in memory at once; or it will have to do something like ED does, and make one pass at the file, with no going back to the beginning unless it's still in memory. The editor can swap sections of the file back out to disk on a least-recently-used (LRU) basis, but one can get into trouble if one spends too much time doing disk I/O (one can lose characters from the keyboard), and it usually requires a special file on disk to swap to, which must be as big as any file that is to be edited. Smart swapping on an LRU basis is what many operating systems (and cache buffers) do to act as if available memory (or available fast memory) is much larger than it really is.

MINCE, AMETHYST & EMACS

MINCE (Mince Is Not Complete Emacs) is a recent addition to the proliferation of Screen Editors. MINCE is modeled after EMACS, an editor which grew out of many years of experience with TECO at MIT. EMACS is interactively extensible, a nice feature indeed. MINCE is merely extensible. (The difference being that the High Level Code underlying EMACS (TECO) is interpreted, so that one can edit up a routine, and then execute it immediately, whereas the High Level Code underlying MINCE (C) is compiled.)

The advantages of extensibility are numerous. Short of actual extension, one can customize the editor to meet one's tastes. The simplest sort of customization, the rearrangement of commands, is by itself of inestimable value to those who do not share my penchant for mnemonically related commands. (You can use the "D" key for your home key if you like, and run out of room if you like too, while trying to design a position-oriented command set). Another sort of customization is making supplied commands behave the way you think they should behave, not the way the manual says they do. Movement commands often behave as intended, but rarely as expected. It is a simple matter to change the behavior of existing commands. In fact, the dividing line between extension and complex customization is not a clear one. MINCE can be bought in a standard configuration, or can be bought with the source code for the command set, so that extensions can be made. Users can write commands of their own to perform often-needed functions, or write "modes" in which some command definitions are slightly altered to facilitate easier editing of different text types. For example, one might have an ALGOL mode where the function that looked for paragraphs (an internal one) looked for "BEGIN" and "END" pairs starting in the first column while in English Mode it might merely look for a lone tab in the first column, so that the existing paragraph movement and modification commands will work in a useful manner. This extensible MINCE (which comes with BDS C and Scribble, a text formatter) is called AMETHYST.

MINCE is completely portable. Written in the language C, it is conceivably portable to any machine that will support C. Although developed on a CP/M system, Mark of the Unicorn has brought MINCE up on a DEC 11/70 already and the conversion and installation took a mere two days to perform. This sort of portability means that you can continue to use MINCE on your 16-bit or 32-bit computer which supports **NIX or C. Site license agreements (some clubs can qualify as sites) are also available. It is conceivable that MINCE could be the last editor you will ever need to get, no matter what you upgrade to or grow into.

MINCE tackles the problem of file management with an LRU scheme that works very well, even for files of appreciable size. MINCE does all if its work in in a MINCE.SWP file either on the logged disk, or on the A: drive. MINCE also allows the user to edit up to seven files concurrently. This may seem like a good way to get oneself into trouble at first, but it's an excellent way to compose a paper or program from several other papers or programs. MINCE also will display one or two windows at once. These windows can be into the same file or into different files.

MINCE's ability to display multiple windows and to edit multiple files is an invaluable aid in composing text or programs from many small subunits in different files. One might use it to create a new program which contains routines or functions swiped from a number of others, or to include part of a previous letter in a follow-up letter.

MINCE allows the user to save files to whatever filename (d:filename.ext) he wants, at any time in the edit session. File saving is only done when an explicit command to do so is issued. No .BAK files are created. Furthermore, no files are created automatically, even upon exit from the editor, although an attempt to exit when modifications have been made since the last save will result in a prompt to allow you to reconsider abandoning your work. A disadvantage of this scheme is that the default filename for a C-X C-S (Save File) is the same filename that the file originally had, which could potentially obliterate the original. There is also a C-X C-W (Write File) which prompts for a filename. Users should probably get accustomed to using Write File instead of Save File unless they are certain there is no need for backup. Using Write File with a generation number for the extension (extensions of the form .001, .002 and so on) would probably be the best bet for text creation. An operating system that provided for generation counts as a third extension would be the best.

MINCE produces files which are compatible with ED, and every interpreter and compiler I have tried. MINCE-produced files also can be fed into most of the text formatters I have attempted to use. The only problem will be with "automatic justification" with those formatters that will not take out "hard" carriage returns. Justify your text in MINCE first. MINCE leaves no bookkeeping at all in the files it creates, either during the session or after it is written. (You can reset out of MINCE with impunity, and be assured that nothing is damaged,

except your ego.)

For all my ravings about losing characters, MINCE occasionally loses characters on my system, but that is because my DMA disk controller can't possibly poll my console. MINCE would probably lose even less on a system that had interrupt-driven console input. MINCE has a type-ahead buffer of 80 characters, and does a lot of console input checking. It will only lose characters when a disk operation is in progress, and prints a message on the screen to this effect during every disk operation. Still, MINCE waits until the console has been idle for a while (a time unit that the user sets for himself when he first configures the program for his system) before doing any disk operations. MINCE should never have to go to disk while the user is typing (it really hasn't on me, either), unless you can type continuously for 20K or more without pausing. (If you can, I'm very impressed.)

The largest file size which can be edited is determined by the size of the MINCE.SWP file, another parameter you set when configuring. The authors recommend a 64K .SWP file, which I find quite reasonable for everything but those few monster files I mentioned earlier. Currently, the maximum allowable size for a .SWP file is 248K, and the largest file I have is 208K (but I never edit it anyway, I just print it out), so I'm not in trouble yet. I presume the maximum .SWP file size will be upped in future versions of MINCE (or when the authors get a hard disk).

The command set, as supplied, is a delight. There are several types of commands, determined by command prefix (or lack of prefix). Control-F (Control characters will be represented by the prefix "C-") moves forward one character (F for forward, get it?). C-B moves back one character (guess what the B is for). Meta-F forward one word (Meta characters are actually characters preceded by the ASCII "ESC" character, but for reasons of MINCE's heredity, they are called Meta- characters. (It is possible to generate Meta- characters in hardware, the authors' keyboard does this). If the term Meta- is taken to mean "greater than," then many of the command definitions make lots of sense), and M-B goes back one word. C-A and C-E go to the beginning and end of a line, respectively, while M-A and M-E do the same for sentences. "DEL" and C-D delete forward and back words. C-K deletes lines, and M-K deletes sentences. M-[(square bracket) moves to the beginning of a paragraph, and M-] moves to the end.

File commands are all preceded by C-X (for eXtended commands). C-X C-F finds a file and reads it into a buffer. C-X C-R reads a file into an already existing buffer. C-X C-S saves a file to the last specified filename, destroying anything with that filename. C-X C-W writes to a different filename (prompting you for it). Other C-X commands select Modes (C-X M prompts for and selects a mode, like Page Mode (overwrite)) or tell you just where you happen to be in the text (C-X =). C-X K kills the current buffer (if you are using more than one, and you need to free up space). Termination is accomplished by C-X C-C, which prompts if you have modified anything since the last save, asking if you really want to throw it away.

MINCE uses two lines at the bottom of the screen to give useful information. On the "Model Line" MINCE shows its version number, the currently selected mode (if any, otherwise it shows "normal") the buffer name, the current filename, the distance the point is from the beginning of the buffer expressed as a percentage, whether the buffer has been modified or not and whether there is anything in a special buffer called the "kill buffer." On the last line of the screen, the "Echo Line," MINCE prompts for input, echoes command prefixes if you type too slow, tells you when it is writing to disk, or swapping, and signals completion of certain commands.

There is no help available within MINCE, but I suppose one could make a second window (C-X 2), switch to it (C-X 0) and read the MINCE short command list into it (C-X C-F SCOMM.DOC'CR') and scroll through it C-V forward, and M-V backwards) until one finds what one wants. On the other hand, the tutorial that comes in the manual is as good as any tutorial I have seen in a long time, and it comes in two versions, one for novices, and one for folks who think they have experience. If one has had experience with EMACS one may never need to consult the manual at all (except to confirm that you can't really have minibuffers full of TECO code, and to get the wall chart out of the manual and onto your wall). The wall chart can be cut up and pasted to your terminal (it fits on mine) or posted on the wall by your terminal (or bathroom mirror). Once you successfully make it through the tutorial you can do most of the terribly difficult things, and understand enough that you needn't always look in the manual for assistance. The command set is easy enough to remember so that one doesn't really miss internal documentation. (I learned how to use EMACS without any sort of manual at all, but there really is no excuse for stupidity.)

Extending MINCE with AMETHYST

Purchasers of the AMETHYST package will receive a MINCE, a Scribble (a text formatter to be the subject of another article), a BDS C compiler and the source code in C for the MINCE command set, and support routines for the commands and the terminal in C. It does not include the source for the multiple-window redisplay, or the buffered file I/O, which is most of the body of MINCE. It is the intent of Mark of the Unicorn that you can rewrite the command set, and your own commands all you want, but that you cannot rewrite the editor itself (or gronk it for all eternity) and resell it. For **NIX users, who are usually provided with source code for everything, the source for MINCE is negotiable on a per site basis.

This may sound disheartening at first to all the hard core C hackers out there who are gearing up to perform wizardry on their version of MINCE. However, Mark of the Unicorn has properly abstracted most every conceivable function so that linking to internal functions is a simple task. AMETHYST is provided with an exhaustive program logic manual which details the internal organization and function of all the routines accessible to the programmer, entry point documentation is provided for every primitive available within MINCE, and an excellent discussion of state of the art theory of text editors is included.

MINCE is implemented as two editors, one inside the other. The inner editor consists of the redisplay and file handling functions, and editing primitives. The outer editor parses the terminal input, provides command dispatch, and interprets many of the more complex commands. It is the outer editor for which the source code is provided. In the outer editor, command names (characters typed at the keyboard) are bound to the commands (functions in C which in turn call the primitives contained within

MINCE). Such is the flexibility of this scheme that it is plausible that a programmer could create a MINCE mode to emulate nearly any video editor (although the motivation for such an act is beyond me).

Also provided is a terminal abstraction which could be modified to allow MINCE to support any sort of video device whatsoever. The terminal abstraction could be written to support memory-mapped video boards, terminals which do not support cursor addressing, or terminals which cannot be supported through CONFIG.COM (the MINCE configuration program). The terminal abstraction could also be used to make an inconvenient terminal livable. My terminal has the "DEL" and "←" (underbar) on the same key, but the "DEL" is a shifted character. I could alter the Terminal abstraction to swap the two on input, so I don't have to shift to type "DEL."

Conclusions

MINCE is one of the finest editors I have ever used, on any machine. It is without question, in my opinion, a absolute steal at the base price, and a fantastic bargain as AMETHYST. MINCE is not perfect, and I am ready to admit that. MINCE lacks some features like provision for iteration of more than one command as a group, and a "grind region" in addition to the "grind paragraph" it now has. The exquisite beauty of AMETHYST is that the features I think it lacks, I can write in. And I will. So can you.

I hope there will be some sort of exchange program for AMETHYST functions and modes that users write. Judging from the EMACS history this will be the major way in which the program will grow, through input from its users. Most of the current library functions in EMACS were submitted by users in the community. This sort of exchange and support could ensure that MINCE will stay as far ahead of everyone else as it is now.

Mark of the Unicorn is ambitious about MINCE, AMETHYST, and Scribble. I feel they have every reason to be. The complete system is supportable on nearly every machine available, and is patterned after systems (EMACS and SCRIBE) which have fared very well in demanding user environments. There is no reason why MINCE and Scribble could not make the basis of an office-type word processing system that could rival the best of the "big name" systems. There is no reason that MINCE could not be used as a program development tool of the highest caliber. If you feel like you got locked into an editor that is merely mundane, then maybe you need MINCE.

Bibliography

C. A. Finseth, B. N. Hess, S. Layson, J. T. Linhart, MINCE User Documentation (Mark of the Unicorn, 1980).

C. A. Finseth, B. N. Hess, S. Layson, J. T. Linhart, MINCE Internal Documentaiton (Mark of the Unicorn, 1980).

B. W. Kernighan and D. Ritchie, *The C Programming Language* (Prentice Hall 1978).

B. K. Reid and J. H. Walker, *SCRIBE Introductory Users Manual*, Second Edition (Carnegie-Mellon University, 1979).

R. M. Stallman, *EMACS The Extensible, Customizable, Self-Documenting Display Editor* (Massachusetts Institute of Technology, AI Memo No. 519, 1979).

R. M. Stallman, *EMACS Manual for TWENEX Users* (Massachusetts Institute of Technology, AI Memo No. 555, 1980).

L. Zolman, *BD Software C Compiler V1.4 User's Guide* (BD Software, 1980).

■■

PCNET Protocol

(Continued from page 47)

that is understandable enough, we will follow the example of the FORTH Interest Group, conducting a large workshop in which the owners of many different kinds of computers recode the "model" implementation into their favorite language, for their own machine – but also for all other users of their kind of computer. We will distribute the basic versions of the PCNET protocol without copyright. We will then encourage people to develop other services and products that use the PCNET protocol as the basic means of communication. We have the support of People's Computer Company, which is noted for encouraging the beneficial application of personal computers.

We are grateful for the talented contributions of many individuals who have recognized this effort's importance. We would welcome further contributions in money, materials, or participation. We will soon be ready for the large workshop to recode the model implementation for many kinds of computers, and enthusiastic volunteers from the San Francisco Bay area are particularly welcome. ■■

San Diego Computer Society

The SDCS Board of Directors has authorized the formation of an Educator's Interest Group. In order to better support the use of computers in schools, EDIG will provide teachers, students and administrators (presumably) with information, techniques and other forms of assistance in areas of computers and computer science. They have scheduled monthly meetings for different parts of the San Diego metro area. Questions or comments may be directed to Mel Zeddies, c/o SDCS, P. O. Box 81537, San Diego, CA 92138.

British Computer Club

Hobby computer club members at the Polytechnic of North London will run an inner city computer center in conjunction with staff members and senior students. They will be available to give demonstrations, familiarization courses and advice to visitors.

The computer center aims to help small businesses in the urban area. A £20,000 grant from the Hackney/Islington Inner City Partnership, in association with the Inner London Education Authority, will be used to acquire several computer systems in a broad range of prices. Small businesses will be able to use these for a computer-based business control system, integrated with the rest of their business and with the owner educated in how to exploit the full potential of the computer operations.

The project is hoped to alleviate a major cause of job loss – the high mortality rate of small business in the area. A principal cause of the failure appears to be lack of financial control, poor purchasing and marketing systems and need for better management skills. The Polytechnic has a thriving computer club with an active business computing section. They plan to open the center each evening and all day on Thursdays and Fridays.

FORTH Interest Group

In the San Francisco Bay area, FIG members and interested parties meet the fourth (FOURTH?) Saturday of each month from 10 a.m. to 4 p.m. This longish meeting is in two parts. The mornings are for demonstrations and various presentations, while the afternoon schedule consists of more formal proceedings. You can find it in the Community Hall of the Southland Mall in Hayward.

■■

Mincemeat And Minced Words

Dear Mr. Ouverson,

Sigh. I apologize to all readers who will advise me to get a copy of Harbrace or Turabian and read it. I have copies of them already, and I have read them, but I just didn't take too much time when I wrote the article. I don't usually nest parentheses unless I'm writing LISP code.

The most glaring errors are in the description of the command set for MINCE. I made two big blunders, one in the inclusion of some EMACS commands that are not existent in MINCE, and in another place an entire sentence is missing, and the one that should follow it is simply wrong.

My keyboard does not generate Meta characters. Mark of the Unicorn's does. It is a custom one off design. That's a Mode Line at the bottom of the screen, not a Model Line. C-X K prompts you for a buffer name, and does not kill the current one by default.

There are now plans in the wings for a user's group of some sort for Amethyst. I hear that there is already a C mode floating about someplace. I will advise *DDJ* as soon as something materializes.

I apologize to anyone who feels that I severely maligned his editor or text processor. Send me a copy and I will review it too, and will try to be as objective as I can. It's hard for me to be objective when I love a product as much as I do MINCE.

Yours,
Barry A. Dobyns
1635 Royal Crest #1128
Austin, TX 78741

SCRIBBLE and MINCE Group

Sounds like a club for subway vandals, doesn't it? Or possibly one for cookbook authors. It's really the **Amethyst Users Group,** an organization for those sharing a common interest in the Amethyst editor-formatter package described so enthusiastically in *Dr. Dobb's Journal* #54. Annual dues are $6, which is also the tentative price for diskettes of club programs. The latter will be made available in a number of formats.