Editorial

Editors. This is a word of multiple definitions. One definition is implicit in the job description of myself and my associates at BYTE. Another definition is that applied to a class of utility programs which every BYTE reader's computer system has in some form or another. It is this latter definition which provides the subject for this editorial.

What is often ignored is the fact that editor programs make an excellent form of software project, less complex than an interpreter or a compiler, but of sufficient magnitude to be interesting and educational. The problems and characteristics of editors are analogous to those of compilers and interpreters, especially when viewed as tools of software development. Just as compilers beget better compilers, editors can be used to edit better editors.

By Carl Helmers

On the Virtues of Writing Editors

Software projects are some of the most exciting avocational applications of personal computers, applications which emphasize computing as an art form and means of expression of personal tastes. This editorial on the philosophy of software projects in general and text editing in particular was inspired by recent completion of a text editor begun as a spare time project in March of this year. Five months and many evening and weekend hours later it is now mid August and the editor program is working sufficiently well to serve as the primary monitor program and software tool of my homebrew computer system.

Why Write an Editor Program?

There are alternatives to do-it-yourself. In the usual case of purchasing a commercial system from a computer store, an editor program is built into the systems software of the computer. The present day technology of self-contained desktop computers with read only memory systems software (BASIC interpreters) invariably includes primitive line oriented editors built into the computer.

On computers which have assemblers plus high level language facilities like Pascal, C, FORTRAN or COBOL or brand X, one or more forms of more useful text editing programs may be available in order to prepare source language files. This is the configuration found in the typical $3000+ personal microcomputer or $10,000+ commerical minicomputer system.

Even hardware homebrewers do not necessarily have to start completely from scratch with editor software. Often a microprocessor chosen for a homebrew system has a software package which can be purchased from one of a number of software vendors. And in the newer processor designs like the Motorola 6809, complete dynamic relocatability of code allows systems software to be sold in address space position independent read only memory chip sets which homebrewers can expect to have access to over the next few years.

But to do-it-yourself is one of the principle reasons why people get involved in intricate avocational pursuits. Why does the amateur wine maker inject yeast into juice and wait for the results when Napa Valley and its industries exist? Why does the amateur pilot learn to fly when professional transportation options are available from the largest airline to the smallest air taxi service? Why make your own astronomical telescope when there exists such a perfection as Questar? And similarly in computers, why make your own text editor, compiler or computer when so many options exist in the marketplace? On a practical or short term basis there is no visible advantage to engaging in such pursuits. But on a long-term scale of personal development, actively applying one's mind and energy is always rewarding.

What Was Needed?

With this spirit and a secondary purpose of improved systems software in mind, I proceeded to think about writing an editor program. What is required in a personal computer system to engage in a project of this complexity? The requirements are essentially identical to the requirements for programming any arbitrary application of the personal computer, from electronic music systems to sophisticated information storage and retrieval systems: tools. Software tools are needed in order to create more software, and software tools presuppose a certain hardware basis.

The hardware components needed for a useful system are available right now from a

Continued on page 68

Continued from page 6

number of vendors, at prices in the $2000 to $5000 range. This hardware basis for software tools includes a processor, as much memory above about 16 K bytes as possible, a terminal device, and electronically addressable mass storage on floppy disk or its equivalent. (Two drives are much much preferable to one, so copies can be made; I have only one drive and feel the lack every time I have to transfer to a new disk.) A printer, such as my obsolete Model 33 Teletype, is useful whenever final listings or working maps of the source code are needed. In my case, the processor and peripherals were designed and integrated personally, but numerous commercial hardware products are essentially identical in function and probably less expensive if you value time spent.

Readers interested in doing similar projects with nonhomebrew equipment will undoubtedly have an easier time of it, since chances are that a good operating system and assembler, BASIC interpreter or Pascal compiler would be available for use in creating the programs. I was stuck with my crude earlier version editor, a kluge hand assembled operating system, and a very good macroassembler (but still an assembler with all the defects of assembly language). People using BASIC can probably create assemblers, text editors, or interpreters far easier than I could with my system such as it was: witness the Tiny Pascal project of Herbert Yuen and Kin-Man Chung in this BYTE, several text editors and assemblers I know about, and my friend Dan Fylstra's 6502 assembler in BASIC which is marketed for the PET by Personal Software.

But never mind the lack of the ultimate in tools. Part of the reason for deciding to work on a text editor was that in addition to exploring the software design aspects, I would in effect be using the previous tools to bootstrap a better set of tools. (Ah, you may note, there was a pragmatic and short-term result of the project; like many activities with long-term good effects, there are short-term good effects here, too.)

In my earlier editor, written in 1976, the emphasis was on simplicity and getting an editor written in minimal time subject to the constraints of not having an editor. As a result, I took coding shortcuts which are functionally correct but not particularly fast. Thus all operations were on an incremental character by character basis. For example, insertions in very long files take incredibly long amounts of time per character since the entire file must be shifted.

The new editor program was the last software project which I would implement while having to put up with the older editor's faults.

Designing and Implementing the Program

I have a number of biases in the design of editor programs, biases which reflect the hardware I have and my philosophies of interaction. This is one of the reasons for engaging in a do-it-yourself design.

For example, I am personally devoted to that school of text editing which uses a minimum of keystroking, where single keys for the most part are used to invoke all commands. I am also committed to avoiding my own stupid errors which can be disastrous if the wrong single key is pressed. So, I oriented the editor design to single keystroke inputs which have three general classes of potential disaster if inadvertently invoked.

There are benign functions which can always be executed by mistake without hurting the file. These are commands like *print* or *go to the next line* or *back up the character pointer* or *go to the beginning of the file.* In each case, the command merely manipulates file pointers without many side effects. These commands are executed by single keystrokes.

A second class of functions are what might be called really serious because of the effects they may have on the file. In this class are found things like *insert text at the current character location, replace text at the current character location* or *execute the macrostring.* To protect the file from inadvertent use, these commands require use of the control or shift key on my terminal to generate various ASCII control characters. The control key must be pressed at the same time as some other key, so a 2 key and 2 finger combination is required, typically with the left hand on the control or shift key and the right hand picking the particular command key.

And finally, there is a whole class of potentially disastrous commands like *write the file to disk, read the file from disk, assemble a program, load a program, jump to an arbitrary address* and so on. These commands are typically invoked by a shift or control character so that two keys must be pressed as in the previous class. Then, to provide further protection, a message is displayed on the terminal requesting confirmation with an arbitrary character. An escape character can be input if the operation is to be aborted before the text buffer is actually or potentially zapped.

Another example of a personal bias is that the editor would have to have provisions for search and change operations, execution of macro strings of editor commands, and the ability to move blocks of text around from one place to another in the file. Not all editors (especially not the typical BASIC interpreter's editor) have this sort of capability. Most separately available programs which go under the name of text editor have these capabilities.

There is a myriad of much more detailed choices involved in the design of a program for text editing, for example: how to allocate memory, how to decode commands, internal coding conventions, provision for expansion into a full operating system monitor within the framework of an editor. The educational results of the editor writing process are: a specific personal involvement with complex questions, the necessity of making the choices involved in the design within the framework of these questions, and a broader understanding of computing. So, if you want an answer to the question: "What educational and pragmatic project can I do with my computer?", my answer would be, "write an editor."■

Articles Policy

BYTE is continually seeking quality manuscripts written by individuals who are applying personal computer systems, designing such systems, or who have knowledge which will prove useful to our readers. For a more formal description of procedures and requirements, potential authors should send a large (9 by 12 inch, 30.5 by 22.8 cm), self-addressed envelope, with 28 cents US postage affixed, to BYTE Author's Guide, 70 Main St, Peterborough NH 03458.

Articles which are accepted are purchased with a rate of up to $50 per magazine page, based on technical quality and suitability for BYTE's readership. Each month, the authors of the two leading articles in the reader poll (BYTE's Ongoing Monitor Box or "BOMB") are presented with bonus checks of $100 and $50. Unsolicited materials should be accompanied by full name and address, as well as return postage.■