Reflections of A Computer Language Nut

STEPHEN BLOCH

My introduction to computers came in December 1979. There was a computer room in my high school, and through its windows I had seen some friends at mine typing away feverishly at what I recognized after some thought as computer terminals. It took me about a week to find enough nerve to knock at the door. Interestingly enough, it was not the computer itself that I was afraid of. I had read enough about how computers were a man-made invention completely controlled by humans to be interested, not frightened, by the prospect of controlling them myself. I was afraid of the rites of passage into the "computer clique," which I envisioned as some sort at Pythagorean mystical society. Fingers crossed behind my back, sweat dripping from my brow, I raised a fist to knock at the door and was greeted with the anticlimactic news that I would have to talk to a math teacher down the hall to obtain an ID enabling me to use the computer. Oh Brave New World!

The computer I was first introduced to was a "mini" called the Hewlett-Packard 2000. It could communicate with up to 32 users at a time, most of whom were connected by telephone cable from all over the county. This particular computer "understood" only one of the many artificial languages on the market today, an educational language developed in 1964 called BASIC. BASIC is more readable and hence easier for a beginner to understand than FORTRAN, LisP, or machine language. However, its unstructured style is responsible for most of the sloppy programming in the world today, because so many programmers like myself first learned BASIC. Its greatest selling point is that it is by far the most widely known language for personal and small business computers in the world. Regardless of the language, however, the multi-user system was appropriate for a beginner because there were always plenty of experienced people to talk to via a message-sending program built into the system. The ability to call on more experienced people for help is important in learning to program and use a computer, and every multi-user system I have worked on has had an easily accessible program to perform this message-sending function.

Exploring Computer Languages

During the first year of my computer use I was restricted to BASIC, though I did use two of the dozens of different versions of it. In 11th grade I took the Computer Science II course and was introduced to FORTRAN, one of the oldest computer languages, dating to the late 1950's. I didn't like it, partially because the nearest computer that ran it was a different model that had more complicated commands and no documentation to explain how to use them. Because of this association, and the fact that it contained few features our version of BASIC lacked, I still dislike FORTRAN. However, the idea of the existence of languages other than BASIC, to which I had grown so accustomed that I had almost forgotten it was an invented language to begin with, intrigued me.

I started trying to write my own language, modestly called BLOCH (Branch/Loop Oriented Computer Handier) and found that I could not write it efficiently without a language specifically designed for writing computer languages. Therefore, I started learning SPL, Systems Programming Language. I had little success writing in it, but its modular structure was quite different and far more logical than the linear structure of BASIC and FORTRAN. Then I tried PASCAL, a similar modular instruction language my father told me about. I visited my cousin at MIT and found that his computer science course used one of the oldest structured languages, LisP (List Processing language). I started trying to learn that, which was difficult because I had no LisP instruction manual, and of the two in existence, one was obsolete and the other incomplete. I joined an Explorer post in my area which met every week at the Honeywell building for 2 hours to learn about computers, and found to my delight that their computer ran LisP. A friend in my Computer Science class mentioned a language called FORTH, in which programs consisted entirely of words defined in terms of each other, and this interested me. All of these languages, collectively referred to as "high-level languages," were based upon and defined in a machine language, which is fast, compact, and utterly unreadable. Then I started to wonder what such a low-level language was really like. In short, I went language crazy, and I still am to some extent.

When introduced to computers, most people find some area of special interest in which they, too, "go crazy," though not many are language nuts as I am. For example, one of my friends does only what programming is required for class and spends most of his time improving and testing a homemade microcomputer, which he has now built up to the level of a relatively cheap hand-held programmable calculator, but which has far more capacity for expansion. My Explorer post is divided into three groups, roughly equivalent to patrols in Boy or Girl Scouts, which concern themselves respectively with software, hardware, and simulations (programming a computer to act like a different type of computer or to understand a different language) and although every Explorer does some programming, we work mostly within our groups.

One effect seen in people, especially teenagers, who spend a lot of time with computers is somewhat frightening: In some cases they forget how to communicate with everyday, illogical humans. The closest approach to this I have seen myself is a tendency to semiconsciously speak in jargonese and buzzwords ("trouble utilizing the FOPEN intrinsic in an SPL sysdump program that breaks security by FCOPYing the low-memory end of the working-directory RAM") with people who know little or nothing about computers. This effect can be avoided if the person concerned works on computers less than about four hours a day, but its existence gives one pause.

The Attraction of Computers

We have seen that young people are often naturally attracted to computers for a variety of very basic reasons ("Ooh, look at the pretty colors!" "I take the center square..." "We'll show those Klingons! Ahead Warp Factor Four!" "If I bishop to King's knight five, he'll probably . * * " "Where in the user-input routine do I want the error-trap subroutine to return to?"). Computers are virtually impossible to destroy or even mildly confuse without a long-term, conscious effort, and even with one -- I've tried! The most important requirement to learning about or even with computers is freeing oneself from hangups about robot revolutions and looking foolish, so that one can concentrate on learning from and teaching these idiot-savant machines. Now that you know more or less how to learn about computers, it might be nice to know why as well.

The Need for Computer Literacy

Over the past few years numerous reasons to glorify computers have been found, so I'll skip briefly through the more obvious ones to one I feel is both more valid and more important for gifted children than all the rest combined.

The use of the computer as a teacher's aide depends on optical-scanner forms, from general ones a teacher can use for multiple-choice tests to nationally standardized tests like the SRA/ITED, the PSAT/NMSQT, and the SAT. Without black No. 2 pencils to fill in the entire circle but not lap outside it (see example on your test booklet, back page), our colleges and high schools would come to a screeching halt within a month or two!

Many high schools and most colleges have kept grading, scheduling, and attendance information on computer files for years. But a system in which teachers use computers regularly and students not at all will clearly give young people the impression that computers are "grown-up things" which they are not allowed to use. Admittedly, the feeling of forbiddenness often adds a thrill of danger, but in this case it also prevents students from asking their teachers for help.

One commonly cited reason for teaching kids about computers is the immense practical value such skills will have in a computer-oriented world, both in job-finding and in everyday life. However, a major goal of today's programmers is to make computer programs more and more "friendly" so computer training beyond the point of losing the fear of computers will have little practical everyday use.

A more significant development, however, is the series of programs written in the last 5 years, as program generators. A person with little or no skill at programming can specify in everyday English what he or she wants a program to do, and a program generator such as The Last One will write it quickly. The Last One works best for applications programs, those which serve a specific purpose such as playing a game or predicting the position of a spaceship, but YACC (Yet Another Compiler Constructor) or something similar will quickly write an interpreting program for almost any language you can specify. What all this means, according to many computer authorities, is that in 20 to 30 years there will be little need for human programmers at all.

The purpose of bringing up children with computers may not, therefore, be to prepare them for the abundant jobs of a computer-oriented world; such a world would indeed have plentiful jobs in computer operation and programming. A technological advance cannot be considered as having been truly accepted until it becomes ludicrous to even talk about being "oriented" toward it. One might call our society "television oriented" or even "automobile oriented," but only the most precise anthropologists would call it "wheel oriented." Granted, wheels have been around for thousands of years, but in view of the incredibly accelerating pace of technology in this century, I doubt that computers will take over 50 years to reach the same level of acceptance and ignorability.

As far as I can see, the greatest boon of computer literacy is a fairly intangible one. Teaching the skills, habits, and disciplines of computer programming improves a child's (or an adult's) ability to think, and these skills should be taught for the same reasons the so-called Great Books should. For example, Euclid's Elements contains little factual knowledge that has not been rehashed and then taught in an easier fashion, not to mention its relative inapplicability to everyday life, dealing as it does with a system that does not represent reality, as Euclid knew quite well. The benefit of the Elements is simply that it teaches the habits of clear, rigorous thought in the same way computer programming does.

In computer programming, nothing can be glossed over, nothing can be left to cure itself, and everything must be accounted for. If seemingly insignificant problems are ignored, they often manifest themselves in grandiose ways, commonly known as bugs or zarkas. (A simple example might be a chess program that, whenever you legally take one of its pieces, eliminates instead your corresponding piece.) As a result, programmers, even if they communicate only with "friendly" computers in spoken English, gradually learn to think more clearly and with greater comprehension of the abstract effects of their actions.

Children and Computers

If computer use and familiarity are indeed to be taught widely, as I have suggested, I believe they must be started early in life, preferably in the lower elementary grades. Children have several advantages over adults in the process of learning. As I have already mentioned, they do not have a fear of looking foolish. A child, at least before adolescence, trying to learn a complex skill like computer programming often couldn't care less about his or her reputation and simply wants to learn more about these fascinating machines. An adult not already experienced at computer work might spend hours searching in his own mind for an answer while a bright 10-year-old working on the same problem will ponder for only a minute or two before asking someone more knowledgeable for help.

The danger of kids losing their social skills by immersion in computerese may be averted, not aggravated, by an early introduction. This phenomenon is most common in teenagers who are new to computers and still excited about them as an end in themselves. Children who grow up with computers as a fact of life are less likely to jump into them with the same all-consuming fervor I have occasionally witnessed and fallen victim to, but will be more likely to simply accept computers for what they are.

It is a well-known fact that children are particularly adept at learning languages, and I suspect that this ability applies equally to logical thought patterns of the sort one develops after a few hours of playing chess, twisting a Rubik's cube, or programming a computer. In any case, such computer languages as BASIC, PASCAL, and LisP certainly qualify as languages in the way they are learned, though they may have smaller vocabularies and more logical (and hence more easily learnable) grammars than do human languages. Children, therefore, should have less trouble learning both the techniques and the disciplines of computer use and programming than adults would. In addition, they can more fully internalize the logical thought patterns necessary for good programming, so they benefit more from the training.

Conclusion

The invention of computers has caused irrevocable changes in human society, and within 40 years will bring about changes we cannot conceive of today. Even if it were desirable to eliminate them from our lives, we cannot reject them now without crippling civilization for decades. Obviously, therefore, we must adapt to them: learn why they work, learn how to use them, and take advantage of their growth by learning from them. Computers are not "newfangled" things that will go out of style in 10 years. If we do not teach our kids to adapt to the rapid changes the next 40 years will bring, we are making them maladjusted to their world from the start.