RHUG

GAZETTE

The Official Newsletter of the
Richmond Heath User's Group

* *

Volume I Issue 4
March 1982

* *

Well, Good Day, eh you hosers, and welcome to the next wonderful installment of the GAZETTE. I'm glad to see you're all in attendance while we explore more of the wild wild world of hot electrons and how we can make them do our bidding.

Meeting Notice & Disclaimer

The next meeting of RHUG will be on APRIL 12TH at the reknown Alpha Audio Studios in charming midtown Richmond, VEE-AYE. For those who don't know where it is, try 2049 West Broad St. If you're not familiar with it, it's right down the block from the Fifth Avenue Restaurant (where the meeting ought to be held). It will be at 7:30 as usual. I know...in the last GAZETTE I said it was 7:00 as usual, but after all, who's to say what's usual and what's unusual? Our meetings are QUITE unusual. Or so they say.

Ready to find out how you get in? Catch this one...on the right side of the door there is a touchpad. If you carefully punch in 4302 (the night's secret code) the door will unlock magically. The meeting will be in the conference room on the third floor.

From now on, all meetings will be held at Alpha Audio (compliments of Nick Colleran, the illustrious [or is that notorious?] President of Alpha) unless we need a computer for demonstration, in which case we will decide on another location.

Bobby's article, which was supposed to be a continuation of last month's edition explaining to us all about how to go about implementing the new options to the old Bios of Heath/Zenith's CP/M has been delayed. Due to reasons all his own (HE claims he's been too busy at work) it will have to wait until next month. It'll give you something to look forward to. Carlos will be back with us soon and if anyone else has something, send it along.

Yes, this newsletter is a little erratic in getting out. Since your new Editor started we got two issues out in two months and blew the third. Time is a distinct problem these days...both for the Editorial We and for those writing the articles. If it was suppposed to be a monthly We would have named it so. In the meantime we will try approximate monthly routines on it if possible. If not, well, they say absence makes the Gazette grow fonder.

Language Differences (Part One)

by Jim Scott

INTRODUCTION

This is the third in a series of articles on the principles of programming. This article will discuss some of the advantages and disadvantages of the most widely available programming languages for small computers.

Piteous indeed would be the entrepeneur who offered to the hobbyist market a computer that did not support Basic. That computer's share of the market would be lower than a snake full of buckshot. Or at least it is generally believed that it would. In fact, Basic is second nature to many of the popular boxes, residing in ROM. In others, Basic is the only, or practically the only, language available (for example, the Radio Shack pocket computer).

Why is a small computer without Basic unsellable? Is Basic the best of all languages? Is its popularity a good thing? Do other languages offer any advantages?

If I hadn't asked these questions myself, I would be astounded at their naivete. [Author's Note to Editor: Does your printer

have umlauts?] [Editor's Note to Author: The Epson MX-100 will deal with SEVEN languages other than the one we try to use...but if you think I'm going to play silly games with it because you want an umlaut over "naivete" you're crazy. I'd also have to put the accent over the "e"]. As I was about to say before I was interrupted...let's get right into the topic; I feel the need to pontificate [Ed:Feeling religious, eh Jim?].

LANGUAGE VERSUS IMPLEMENTATION

First, I have to make a very important distinction. When I refer to a "language", I am referring to whatever standard or convention exists for the syntax (grammar) and the semantics (meaning). But a particular "implementation" of a language will be either (in the narrow sense) a specific software package available under a certain name, like Extended Benton Harbor Basic or Microsoft Basic or UCSD Pascal; or (in a broader sense) a way of implementing a language, such as a Basic interpreter or a Basic compiler.

COMPILERS VERSUS INTERPRETERS

Well, I seem to be getting ahead of myself already. Let's define the types of software support for programming languages.

Every language implementation requires software to support it. When you buy a new language for your computer, you're really buying an implementation of a language. (The language itself is free; even if you have no computer, you can learn the language from a book and write programs in it until Irving R. Levine takes off his bowtie. But to be able to run the programs you write, you need a computer and software that supports the language.) Simply stated, the software you buy will be either a compiler or interpreter.

A compiler is a program whose input is a source code (which you key in, using a text editor) and whose output is an executable program (for example, a COM or HEX or ABS file). A distinction is often made between compilers and assemblers, but let's just say that an assembler is a type of compiler; the language it compiles is assembler language.

The compiler translates the source program, which is in reality designed to be readable by humans and not by machines, into the executable program, which is in machine language, and is readable by the machine. Assuming (rashly) that the source program is error-free, the process of compiling takes place only once. After that, the executable program can run on the computer without any further help from the compiler. In fact, even if the family dog chews up the diskette that has your only copy of the compiler, the programs you have already compiled will still run as well as they would before (unless the miserable mutt chewed them up, too).

An interpreter is a program which must be present in the computer's memory whenever you run programs written in the language that the interpreter supports. (If the interpreter is in ROM, then of course it is always there.) Like the compiler, the interpreter translates source program statements into executable machine language. The difference is that, while the compiler does this translation once and gets it over with, the interpreter translates each statement of the source program whenever that statement is to be executed.

For example, consider the following Basic program:

```
1000 DIM X(9,9)
2000 FOR I = 0 TO 9
3000 FOR J = 0 TO 9
4000 X(I,J) = I * 9 + J
5000 NEXT J
6000 NEXT I
```

When this program is run, statement 4000 will be executed 100 times. Each of those times, the interpreter must translate that statement as if it had never seen it before.

This is why language implementations which use an interpreter generally run much slower than those that use compilers; all of the processing required to interpret each statement each time it is executed, plus the fact that this processing is going on whenever you run the program, adds up to a lot of overhead. It is not unusual for a program which provides almost instantaneous response time when compiled to be so slow as to be unusable when interpreted. (Response time is the amount of elapsed time between when the user presses "Return" and when the output becomes available. An example would be the amount of time it takes a chess program to make a move.)

Also, since the interpreter must be present in memory while the program is running, more memory is generally required. If

the interpreter is in ROM, the issue is not quite as clear; but if the ROM occupies part of the address space and reduces the maximum amount of RAM that can be installed, then the computer itself has a disadvantage.

We have just seen two serious disadvantages of interpreters. Now let's look at their main advantage.

INTERACTIVE TESTING

Interpreters provide the ability to do interactive program testing. An "interactive" computer activity is one in which the user participates in the running of a program by replying to requests for input, by making decisions based on what the program has done so far, and/or by interrupting the execution of the program.

Any implementation of any language should provide the capability to write programs which will run interactively. Most programs written for small computers are probably of this type. A non-interactive program would be one which, once you enter the command to start the program running, would not request or accept any keyboard input; it would just run by itself until it was finished. Therefore, games, text editors, spread-sheet calculation programs, and so forth are all interactive.

But an interpreter is interactive in a more general sense. The program you write can be interrupted, changed, and restarted at any point. You can execute pieces of it before the whole thing is written, to make sure that your code will run as you expect it to. You can stop the program, examine the values of variables, and execute one statement at a time, to find out where a bug is occurring. You can do some of these things with a compiler, but only by switching back and forth between the text editor (to change the source program) and the compiler (to recompile the whole program before rerunning it). Interpreters generally include the text editting, or program entry, facility so that you can alternate easily and quickly between adding (or changing) parts of the program and running it. There is no compilation step to go through with an interpreter.

Interactive testing is a great advantage to the beginning programmer, who is just learning a new language and is just learning how to program, period. (As I mentioned in an earlier article in this series, these are two entirely different activities; this series is about the latter, not the former.) Now we can see why Basic was a sine qua non [Ed: Yes, Jim, my printer will handle Latin] for early small computers: Most users would be novice programmers; novice programmers get along better with an interpreter than with a compiler; Basic was the language most generally available with an interpreter implementation; therefore, Basic was the language of choice.

In fact, Basic was created as a language to be used in teaching people who were learning to program. But it was never intended to be a serious language to be used for creating and running real programs. Therefore, its initial selection as the "standard" language for small computers was unfortunate. However, once that point was passed, Basic gained the additional advantage of being ubiquitous, and recognizable by everyone.

So interpreted Basic has the advantages of being widely available and of allowing interactive testing. It has the disadvantage of being very slow.

There is more to be said about Basic, and a lot to be said about other languages; but there are still more concepts to be introduced first. The next concept is the "level" of a languae.

HIGH-LEVEL VERSUS LOW-LEVEL LANGUAGES

The ultimate low-level language is one whose statements have a one-to-one relationship with machine language statements; in other words, any assembler (or machine) language. A high-level language spares the programmer any need to be concerned with what goes on at the machine language level. Some typical high-level languages are Basic, Pascal, Cobol, PL/I and Fortran. These languages provide statements like

```
FOR I = 1 TO 100  and
Y = 4.52 * (X ** 3) + 67.33 * X * A + A ** 2
```

whereas assembler language would require many statements to perform either of these operations.

There are also what we might call very high-level languages. APL is an example. To compute the sum of the elements of a 20 x 20 array A and store the sum in S, APL requires only the following statement (where "←" is the APL left arrow symbol):

```
S ← +/+/A
```

whereas in Pascal we would have to code the following:

```
S := 0;
FOR I := 1 TO 20 do
  FOR J := 1 TO 20 do
    S := S + A [I, J];
```

We might even use the term "ultra high-level language" to describe an application generator such as Pearl (offered in the latest Heath catalogue). Pearl is menu-driven; you answer enough questions to make clear what application you have in mind, and the code is generated for you. In a very real sense, using a system like this is not programming at all, and it is not really a language. Therefore, I will not cover this end of the language spectrum any further.

In brief, the higher the level of a language, the more it spares the programmer from having to be concerned with minor details, and the more it allows the programmer to concentrate on what the program is supposed to accomplish. There are two main disadvantages to higher-level languages. First, the programs written in them tend to take up more memory and to run a little slower. Second, the more a program has to deal with the features of a particular computer, the more likely it is that the program must be written in a low-level language. That is why operating systems like HDOS and CP/M are written in assembler language.

Well, that about fills the space [Ed: Void?] that they'll give me for this month, so I'll continue this topic next month. In next month's issue, I'll discuss the concepts of structured versus unstructured languages, and the ease of writing and reading programs in different languages. Then I'll discuss each language in turn, comparing them in terms of all these concepts.

* * * * * * *

Neophyte News

The Z-90 Hard/Soft Sector Caper.

Okay, folks. If I haven't told you before, I am the proud owner of a Z-90-82 (as they like to call it) which is the Zenith assembled version of the H-89 with 64K memory, the Z-89-37 double density controller board and an internal drive. As you probably know, this is a soft sectored controller board. To ramble a moment, I am new to this. So we're dealing with it from a "dumb" (in terms of computers ONLY!) user's point of view. Hence, "Neophyte News".

No problems so far, except a noisy fan which is a necessity since they've jammed so much electronics inside this thing it makes McDonald's look like an anarchy. THEY (the nefarious ones) said that with this system I'd have more space than a California therapy cult. However, I manage to fill file discs (even at double density) like a shot glass of water and in creating a mailing list file of 1,000 names I ran out of computer memory and had to split it into two files!

EXCEPT (the big "except") that I was soft sectored in what appears to be a world of hard sectors. Like being a cab driver at the bus driver's convention. Both transport people but use different vehicles.

It created the minor inconvenience of not being able to use hard sectored programs. Things were copacetic with both CP/M and Supercalc (which were supplied with the computer) but when Spellbinder arrived it was only available in hard sector. This finally resulted in taking my computer to Carlos' and mating them (I was disappointed we didn't at LEAST get a baby calculator out of it...not to mention an Atari Video Game) to make the transfer.

I decided that I needed both capabilities. People like Software Toolworks and others were turning to soft sector availability (Lexisoft, makers of Spellbinder have recently done the same), but it closed me out of a lot. So I called Benton Harbor and ordered an H-88-1 controller board so I could use both. They said it was not available until March 5th. After all the bitching and griping about how long that would be it got here on March 1st! Not only that, but I got the whole kit, including internal drive mounting hardware! It appears to be the same kit they've been selling, but the delay meant (hopefully) that they did some new and improved (sound like a new cereal?) work on it.

NOTE: Zenith Technical Support says HDOS 3.0 has been delayed until June of July.

Here was the ideal plan:

We would install the hard sectored board so that it was switchable. That way I could remain on soft sector until I

wanted to copy a hard sectored disc, make the switch and change it back by simply unplugging the cable.

Here was the problem:

There has to be a programming plug change on the drive that will be switched, making it a pain.

The solution:

After much consideration and kicking around the pros and cons, I am now operating a system in which the internal drive is hard sectored and the external drive is soft sectored.

Thinking it through we figured that since the most used systems discs (Supercalc and Spellbinder) couldn't fit on one disc anyway (even double density), they could just as easily be on hard sector discs while the file discs (of which I've got many) should have maximum disc space. After all, if for some reason this didn't work out it could be changed back easily enough.

We connected the internal disc drive to the hard sctored board and the external drive to the soft sectored board. The new board fits right in after removing the resistor jumper. The disc drive programming plugs had to be reprogrammed so each one was unit zero. Let's fool these driver boards into thinking they're the "primary" one. By doing this, however, the internal drive becomes Drives A through C while the external drive becomes D through F. CP/M feels better this way. A little adjusting, but no big deal.

Someday in the future it would be interestingly handy (what a phrase...) to make these switchable by installing a DIP switch on the external drive (which has easy top access through the vent holes) and a ribbon cable switch for the internal drive so the internal drive could be either hard or soft sectored. But this type of switch is absolutely no fun at all to put together.

The next step was to make CP/M understand what the hell we were doing. I now probably have more configured versions of CP/M than Digital Research. I had to modify the original soft sectored version to conform to two controller boards and add MOVCPM17 to it. I then transfered it to hard sector and reconfigured that one. In case you haven't checked, MOVCPM17 (as well as other versions) is on Distribution Disk II and must be used for the hard sectored board. Then came the big file copying scheme so everything was as I intended. To make life easier, I bought some adhesive red arrows to put on hard sectored discs to easily spot the difference. I can now use either hard or soft sector, transfer from one to the other or (this takes me back to the early days not so long ago) copy the same format using the flip-flop disc scene (Hey, got a spare half hour?).

After calling Zenith Technical Support (Larry Liddle has been GREAT to work with...excellent man!)they confirmed that not only did we do everything right, but they have a number of computers operating the same way in the office! Unfortunately he forgot to send a gold star along. If you're interested or need more "technical" advice on this maneuver just let me know.

UPDATES!!

Much to my surprise I received a Supercalc update from Heath/Zenith that is quite interesting. And it was free! It's on two discs, a program disc and a utility disc. The program disc came bootable (will wonders never cease) and what they've basically done is given me enough CP/M files to run without buying CP/M! Now I wouldn't guarantee that they're sending new ones out this way, but it's an interesting concept. They include Supercalc, a Supercalc Overlay file, Configur, MOVCPM37 (for my version) and Sysgen on the Program Disc. They have several Supercalc sample files plus Format, DUP, PIP, STAT and SOLO on the Utilitites Disc.

The new touches are that the slightly annoying "flicker screen" problem of Supercalc when scrolling has been replaced by a gentle re-write. You can scroll your little heart out, all the while seeing the number or formula at the cursor point and when you stop it updates the screen instead of driving you crazy. I imagir pipable! See, they still think I am soft sectored, but since I fooled them and am bi-compatable I had to copy it to hard sector for my systems disc. No problem. No patching. No slicing, dicing or Ginsu knives needed.

The utilities are the same as standard CP/M except for SOLO. This may be a public domain utility I'm not familiar with, but it is basically a single drive DUP program that reads, writes and verifies...all on one drive! The disc swapping (is that anything like wife swapping?) is held to a minimum. Very nice.

They also included a series of printer codes so it will automatically set certain parameters such as baud rate, printer ready polarity and printer ready signal.

Spellbinder has a new update on the way. Henry Fale in HSCOOP mentioned it and for $50 and the original disc they are

sending an update along. When I get it I'll pass along the improvements. There ARE some (aren't there always?) that it could really use, but overall it's a great program. At last chat Henry was just getting into it so he didn't know all of the things they changed except that the disc is so crammed it's ridiculous. When I spoke with Lexisoft a while back they told me it was forthcoming and they were taking various dot matrix printers into consideration. I hope they did.

Speaking of printers...The Epson MX-100 is still the dream it was. In fact, when I called them to find out if they had technical papers on using it with Spellbinder (they didn't) they told me how to use double strike printing (which for some reason isn't in the manual) and sent along 12 pages of information and sample programs to help get me started on their bit graphics capabilities! Some people ARE nice! As soon as I find the time, energy, motivation and sense of reality I'll get into the bit graphics and let you know how well it works, how much (or little) fun it is and maybe even show off in a future newsletter.

Enough for now.

* * * *