RHUG

GAZETTE

The Official Newsletter of the
Richmond Heath User's Group

* *

Volume I Issue 2
January 1982

* *

First things first, kids. Introductions, apologies, stories, edification and chit-chat later:

Meeting Notice
****

The January meeting of RHUG will be Monday, January 18th at 7:30 PM (for those is doubt...yes, PM) once again at the humble abode of the illustrious Carlos Chafin. For those who either haven't been there, forgot how they got there before or used their last issue of the GAZETTE (before it was known as such) to start a fire, Carlos lives at 4302 Smithdeal Avenue. If you're heading east on Forest Hill from the Boulevard, turn left on 43rd St., go about three blocks and make a left on Smithdeal. If you're heading any other way figure it out yourself. Or call Carlos at 231-6759.

Someone seems to think that Supercalc may be demonstrated, but that all depends on whether I'm (This is your new Editor speaking) will be there to show it. I will talk more about that later. Maybe more about me later too.

But first, here's the first installment of a series which most of us could use:

* * * * * * * * * * * * * *

HOW TO WRITE A PROGRAM

by Jim Scott

INTRODCTION

This is the first of a series of articles on the principles of programming. Future topics will include:

* What makes a program good.
* Language differences.
* Why nobody uses flowcharts
* Structured programming.
* Modular programming
* Top-down programming.

* Avoiding re-inventing the wheel.
* Debugging.
* Documentation.

Of course, most of these topics have already been discussed in books, classes, magazines and articles like this which will continue to appear until everybody (Ed. Note: Everybody?) can write programs that work and can be understood.

To give you a feel for the whole subject, this first article will outline the whole process of writing a program; from conception to bebugging. The stages are:

- Recognizing a Need.
- Thinking aabout the program.
- Designing the program.
- Choosing a language.
- Coding.
- Keying.
- Debugging.

Recognizing a Need

First, you must have a reason for writing a program. Perhaps there is a game you would like to play against the computer. You might feel that the computer could help with a household (or computer-related) record-keeping problem. Perhaps your employer assigns a program to you, or you have contracted to write a particular type of program for a client. If there is a need, and at least a remote chance that a program can solve the problem in a practical manner, you may proceed to the next step.

Thinking About the Problem

The thinking stage and the design stage are critical in developing a program. Of course, you should be thinking throughout the whole process. But you need to go through a period of nothing but thought. This stage may have a little overlap with later stages, but very little. You may want to take a few brief notes to aid your memory, but don't do any real designing yet. Above all, don't do any coding yet. Even MORE above all, don't do any keying.

So what should you think about? First, develop a more complete idea of what you want the problem to do (or solve). Decide whether it seems practical to write the program, given the time you have available and given your hardware and software resources. If necessary, redefine the scope of the program. Finally, ease into the design stage by considering (but not eliminating) alternative answers to some of the questions that must be dealt with in the design.

Don't try to finish the thinking stage all in one sitting. Do whatever relaxes you and helps you think, like taking a shower, or shaving or playing golf. Sleep on it; an idea may come to you in a dream.

Designing the Program

Program design covers so much territory that it takes volumes to do it justice. See, for example, Donald Knuth's series "The Art of Computer Programming". It is s skill that comes from years of programming experience.

Designing a program involves writing down the logic of the program; choosing (or designing) algorithms for subtasks such as computing a cube root, or permutating a string of characters, or sorting a file; and dertermining the representation of the data, both in memory and on storage files.

It is easy to confuse designing and coding. In fact, they may actually be the same stage, depending on what languge or notation method you use during the design.

Design used to be done by means of flowcharting. A later artcle will explain why flowcharting is little used now. A more modern technique is the HIPO chart, which I will not attempt to cover. Design may also be done in some programming languages, or in an arbitrary language consisting more or less of English. A couple of good languages to use in program design are Pascal and Algol. Assembler language and Basic are poor choices for this purpose. If you design wih the same language you will use to implement the program, then when you finish the design you will have finished, or nearly finished, the coding.

The design process should begin with a simple outline of the overall program logic. This s a statement, in terms of broad generalities, of what the program will do, and in what order. In other words, you start by writing down the design of the main loop, or top level of the program. As you may notice, this is difficult to define, but the important thing is to avoid thinking about the details until later.

Here is an example of a main loop for a word-guessing game program, designed in pseudo-English:

- Display instructions for playing the game.
- Set up a table of words; do other initialization.
- Repeat the following:
    - Select a new word to be guessed, from the table.
    - Repeat the following:
        - Get the next guess from the human until the word is solved or human is out of guesses.
        - Display the word and how the human did until the end of the game (human has had enough,or program is out of words). Be kind: Say"Goodbye".

Here is the main loop, designed in Pascal:

```
BEGIN
     INSTRUCTIONS; (*DISPLAY HOW TO PLAY THE GAME.*)
     INITIALIZE; (*SET UP LIST OF WORDS, ETC.*)
     REPEAT
        NEWORD; (*CHOOSE A NEW WORD.*)
       REPEAT
```

```
            NEXTGUESS (*GET NEXT GUESS FROM HUMAN*)
          UNTIL SOLVED OR OUTOFGUESSES;
          SHOWRESULTS (*DISPLAY WORD AND RESULTS.*)
        UNTIL ENDOFGAME;
        SAYGOODBYE
    END.
```

After the structure of the main loop is clear, you get down to more detail by proceeding to the design of the lower levels. In the Pascal example above, you would design the INSTRUCTIONS routine, the INITIALIZE routine, the NEWWORD routine and so on. Since these routines probably invoke lower-level routines you would continue to design lower and lower routines. This is called "top-down programming".

Design continues until the program is complete; that is, until the lowest level routines do not invoke other routines. Of course, there is no need to repeat the design of routines that you already have finished as part of previous programs. In fact, writing routines in a general way so that they can be easily used in future programs is an important part of" modular" programming.

Choosing a Language

By this time, you may already know what language you will use in coding the program. If not, you must decide at this point. Perhaps Basic is the only language available to you. If you have a choice, you must decide whether to use a high-level language or assembler language, and whether you want the program to be compiled or run under an interpreter. Don't make the decision arbitrarily.

Coding

If you have completed the program design, coding consists of direct translation from the design language to the coding language. If these are the same language, little or no coding needs to be done.

Coding, like design, is done wih paper and pencil. Note that to this point in the programming process, you have not used the computer. It is a waste of computer time, electricity and hardware wear-and-tear to do the keying and the design or coding at the same time It also pressures you to make decisions too quickly.

Furthermore, design and coding typically require quick access to all parts of what you have written. For example, you may flip to other pages to see how one routine interfaces with another, or to see how you defined or spelled a program variable. This is easy to do when working with paper, but very inconvenient when it's done at the terminal.

You do not need to use special forms for coding, unless somebody else will do the keying for you. If you do your own keying, the only requirement for coding is that you can read it and know what you intend. If someone else will do the keying, you need to code on paper marked with lines and columns.

Keying

Now you can turn on the computer. Load your test editor program or the interpreter, and enter the program as you have coded it. The statement made earlier, that you should be thinking throughout the entire programming process, applies least to this stage. Sill, you should watch for errors as you do the keying.

Debugging

The first several attempts to compile your program, or to run it under an interpreter, will probably reveal syntax errors. These are pieces of code that violate the rules of the programming language. Some will be obvious once the location of a problem is pointed out. Others may strain your knowledge of the language. In no case should you expect the error message generated by the compiler or interpreter to tell you exactly what the error is.

After you have a "clean" program, you need to test it with all kinds of data and options, to make sure it functions as you intended. When (not IF) incorrect results are discovered, try to deduce logicaly where the error must be in the program. You may need to add temporary code to the program to help narrow down the location of the problem.

Do not consider a program finished until you have tested it with all combinations of data, user choices and hardware environments that you expect it to handle. And make sure you have allowed enough time for debugging. Unless you are a very experienced programmer, debugging may be the most time-consuming stage of all.

(TO BE CONTINUED)

* * * * * * * * * * * * * * * * *

The Editor's Corner

Well, if you haven't already noticed, there's a new name, new look and new Editor to this Rhug Rag. I hope this meets with the approval of you Rhug Rhats (is it true Rhugger's take bigger bytes?) and if it doesn't, well, let me know.

As a brief introduction, I am THE MAD DOG. My company, Mad Dog Productions, Inc. created and sells Croc O' Shirts (you know, the golf shirt with the embroidered emblem of a dead crocodile on it) and Preptiles (tm) skin transfers (which allow you to put dead or live croc's on your skin). I am also a partner in Mad Dog Music for the Messes along with Carlos, and we are producing an album for SINGLE BULLET THEORY, the next great thing to hit the music scene.

I have a Z-90 (64K) with two soft-sector, double density drives and an Epson MX-100 printer. I am not a "computer nut" but need

it from a user's point of view. I am working with CP/M, Supercalc and Spellbinder (with which this is being written). Thanks to Carlos and Bobby we are getting any modifications that need to be done taken care of. So far so good.

I am looking for all articles, comments, letters, notices, etc. that you may want to put in this newsletter. PLEASE HELP!

As far as my analysis of things so far with "MUFFIN" (my so-called computer) go, here is a brief rundown. No, the condensed type is NOT to show off, but to make this thing shorter than the monumental mess it already is.

The Z-90 is great. There have been some compatability problems, some of which we have ironed out, some still pending. Since I am using soft-sector, we have had some problems, but I am getting a hard-sectored board so I can transfer disks without too much trouble. I am also using CP/M but am getting HDOS so I can also be compatable. I purchased a Tandon double density/double sided drive but so far no one has been able to get it to write. Heath has not been much help so if anyone has any bright ideas, let me know. The Z89-37 Controller board appears to be too new, as is this whole set-up.

SUPERCALC is nothing less than GREAT!! Since I know little or nothing about programming (except what I've learned so far from Jim's article) I feel that this is like an introduction to it. I've been using it for everything from a daily TO DO file (possibly something of a jerk-off factor) to a Cash Receipts Journal, to keeping track of studio expenses and hours and onward. It's not difficult to get into and has more capabilities than any of us can imagine. The documentation manual is a teaching guide that is excellently done. I will probably be demonstrating it at the meeting.

SPELLBINDER is the word processing system I am using for this. We have had to make it somewhat compatable with the Epson MX-100 and are working on other changes yet to come. It combines all word processing including mailing lists, merging mailing lists with letters and about anything you can think of that an expensive word processing system will do. The manual leaves much to be desired but it's not hard to work out. The elementary aspects are simple. The more complex parts, like anything else, takes time but is not hard.

MUNCHKIN is a game program offered by Software Toolworks that was recommended by Henry Fale in HSCOOP (the new name). HE'S RIGHT! Anyone who thought good action/movement games were not for Heath is nuts!! This is a must. It is everything Pacman is...and cheaper in the long run.

Enough of this. Anyone with suggestions, ideas, requests, needs for help, etc. just let us know. This IS your newsletter you know!

And now...the column you've all been holding your breaths for...

* * * * * * * *

BYTES ON A BUDGET!!

by Carlos Chafin

There are three products this month that are worth mentioning, not only as good buys, but also because they show trends as to what types of products we can be expecting for our systems in the future.

The first of these is the amazing 400K disc drives made by Tandon, which is a leading micro-peripheral manufacturer of disk drives in California. These drives are excellent quality items and are now being used as O.E.M. devices for IBM's new small

computer as well as North Star and, of course, HEATH/Zenith. The drives are available from several sources including HEATH. Heath is offering the Tandon only as an option for the H-89 and Z-90 computers in conjunction with the Double Density controller (Z-37). In this configuration it offers a 1.28 mega-byte storage system!! I know what you might be thinking right now if you're an H-8 owner, but DO NOT DESPAIR. The Tandon's are also available from Quickdata Computer Services and the Keyboard Studio. Both offer excellent prices and good back-up in the event you need it. The addresses and phone numbers for both suppliers will be listed at the end of the article.

Second on this month's shopping list is the new Epson MX-100 printer with which this newsletter is now being printed. The new MX-100 offers all the features of the MX-80 plus it will handle 14" paper, comes complete with bit graphics, friction as well as pin feed and costs less than $750 from several mail order supply houses listed in the usual computer mags. The unit which typed this letter (Editor's note: I typed it. The MX-100 only printed it.) belongs to Barry "MAD DOG" Gottlieb and was purchased from Scottsdale Systems in Arizona. They are Zenith dealers and also offer great prices on all Zenith products.

Third in the stack is the SPELLBINDER office system from Lexisoft Software. Barry is still uncovering features of this wonderful product and I think it will be the subject of a forthcoming RHUG meeting soon. (Editor's note: Says who, Carlos?) Spellbinder has functions which put it in a league beyond the other popular word processors if for no other reason than it has resident mail list merge facilities without additional purchases. It is easy to use yet has all the heavy-duty features for the advanced user.

The Heath computer users are in for more and more good software and hardware products it seems. It should also be noted that this quality will not stand without a fair price on the marketplace, so anyone in RHUG(or outside for that matter) who is purchasing software *PLEASE* let RHUG know of your product interest. Together, as a group, we have a good chance of finding out which items are valuable and which ones receive the "CORRUPT MEDIA" award!

Quickdata Computer Systems
2918 S. 7th St.
Sheboygan, WI 53081
(414)452-4172
Ask for Henry Fale

The Keyboard Studio
125 Aspen
Birmingham, MI 48009
(313)645-5365
Ask for Ray Massa

HAPPY 'PUTERING,
CARLOS

* * * * * *

THE SOFTWARE LIBARY

by Bobby "The Librarian" Tulloh

Herewith for your careful consideration is a list, by disk, of the current contents of the RHUG software library. Next month I will publish a new, complete, unabridged and accurate (hopefully) list of the library. In the meantime, refer to last month's newsletter for a detailed explanation of each program.

Disc #1

(CP/M 1.43 ONLY)

- CONNECT4
- DOHTABLE
- HALFAPH
- JUMBLE

Disc #2

(CP/M 1.43 ONLY)

- LETTERSQ
- LIFE
- TTT3D

Disc #3

(CP/M of any kind. ASM disc)

- CONSET
- MENU
- TAPECOPY

Disc #4

(HDOS-MBASIC)

- CHECKBK
- EQUATE

Remember, if you want a copy of some or all of the library, bring blank disks to the meeting with $5.00 per disk requested. If you have any questions call me at 794-6732.

Next month I will also discuss (with some authority, if I do say so myself) the vagaries of CP/M and what Heath is doing to everyone. Something to look forward to.

Well, I think this had best wrap up this month's newsletter. It will give you to either fuel your neurons or fuel your fireplace. All correspondance for RHUG should be to:

Richmond Heath Users Group
c/o Jim Scott
1724 Blakemore Road
Richmond, VA 23225

BYE!!!

(c) 1982 Richmond Heath User's Group
Contents may only be reproduced with proper credit to both the individual author and the Richmond Heath User's Group but are not to be reproduced for commercial use without the express permission of RHug.