```
****   *   *  *   *   ***
*  *  *   *  *   *  *
****   *****  *   *  *  **
*  *   *   *  *   *  *   *
*   *  *   *   ***    ***
```

GAZETTE

The Official Newsletter of the

Richmond Heath Users Group

Volume I Issue 8

JANUARY 1983

Editor: Jim Scott, 1724 Blakemore Road, Richmond, VA 23225

Next editorial deadline: January 31, 1983

MEETING NOTICE

The January meeting will be January 10 at 7:30. Carlos Chafin has offered to demonstrate Compuserve at this meeting. The meeting location is Alpha Audio's third floor conference room, at 2049 West Broad Street. The phone number there is 358-3852. The front door has a touch-pad combination lock, and the combination for the night will be 6378 (six three seven eight).

NEW OFFICERS

At our December 13 meeting, we held our first election. Yes, that's right; we have taken one more step toward legitimacy as an organization. The following slate of officers, which had been nominated at the November meeting, was elected by acclamation for terms of one year:

President:	Carlos Chafin
Vice President:	Jerry Tiller
Secretary/Treasurer:	Parks Watson
Software Librarian:	Bobby Tulloh
Newsletter Editor:	Jim Scott

MESSAGE FROM THE EDITOR

As the new newsletter editor, I would like to define RHUG's editorial policy. We'll print just about anything. All the news that fits, we print. As is usual with small groups, we find ourselves in the position of begging our members to submit articles and other items; but I have never felt it was appropriate to badger people to force themselves to write, just so a newsletter could exist. If nobody has anything to say, a newsletter simply isn't needed, and there's no shame in that. However, I agree with the comments that Carlos Chafin made at our December meeting: we all have things we know about that could help others. Even if you don't think you're an expert at any part of the computer hobby, you're probably an expert at being a novice. For example, if you have just completed constructing a new Heathkit hardware item, or have been learning to use a new software package, your reactions and comments will be valuable to many of our readers, no matter what your level of expertise.

There are a couple of practical considerations for submitting items for the newsletter. First, I would like to make the editorial deadline two weeks before the RHUG meeting, which is usually the second Monday of the month. If you will check your calendar, you will see that the deadline for this issue was December 27, and the next will be January 31. The newsletter needs to be in the mail within a few days after this deadline, so you can see that it just gives me time to get all the pieces tied together. Anything that comes in too late will simply be held for the next issue.

Also, I would like to have each item submitted, except short items (less than one typed page), on disk. For now, I think I will be able to work with almost any format, such as input to TEXT or output from TEXT or just anything created with a text editor such as ED or PIE. I should be able to work with either CP/M or HDOS files. But it will have to be on a hard-sectored 5-inch disk. Just mail me your disk, or bring it to a meeting. Don't worry, I'll return your disk to you. My address is 1724 Blakemore Road, Richmond 23225.

****

LETTERS TO THE EDITOR

I would like to propose a feature for the newsletter, which is not uncommon for publications like ours; namely a "HELP" column. I'm sure many of us are stumped by hardware or software problems from time to time (most of the time in my case) and would like to call on the collective expertise of the group for advice. Requests for help, or even hypothetical questions posed to trigger a lively discussion, would be submitted for inclusion in the newsletter, thus giving any members so inclined, an opportunity for thoughtful consideration of the subject, which could then be discussed at the next meeting. It would in effect, assuming sufficient requests were received, form sort of an agenda for the discussion period of the meeting.

Requests would, of course, have to be submitted to the editor in writing and sufficiently explicit as to be properly understood but also as brief as possible. (Catch 22?)

If there is enough interest in this idea it could fill a real need. If not, it will die of its own accord. We hear a lot about the fine documentation furnished by Heath. In the case of construction manuals, I concur in this. But when it comes to operation manuals, my reaction is that they cover in exquisite detail everything except what I need to know to get off the ground. I don't know what the degree of competency of the newer members is, but I suspect that there are others in the same boat with me. Also, it is reasonable to suppose that our major potential for growth will be among those recently attracted to hobby computing. If the word gets around that RHUG is the place to get the fog lifted, it could be a real drawing card for increasing our membership.

Let me hasten to add that I really enjoy listening to you "old timers" discussing esoteric techniques and applications but most of it goes right over my head. It gives me something to aspire to but I would also like to bring home something I can key in before I go to bed!

Parks Watson

****

NEWS

Jerry Tiller reports that he saw an ad for a used Cray 1 computer. The address was an apartment in San Francisco. If you're interested in buying a real number cruncher, and have free electrical service, see Jerry for more information.

++++++++++

The January issue of Computers & Electronics (formerly Popular Electronics) has a cover article on the new Heathkit robot HERO I, plus an article that reviews its competition.

****

WHY NOBODY USES FLOWCHARTS

by Jim Scott

INTRODUCTION

This is the fifth in a series of articles on the principles of programming. The first four articles were "How To Write a Program", "What Makes a Program Good", and "Language Differences" (parts 1 and 2). For reprints of earlier articles, please contact me.

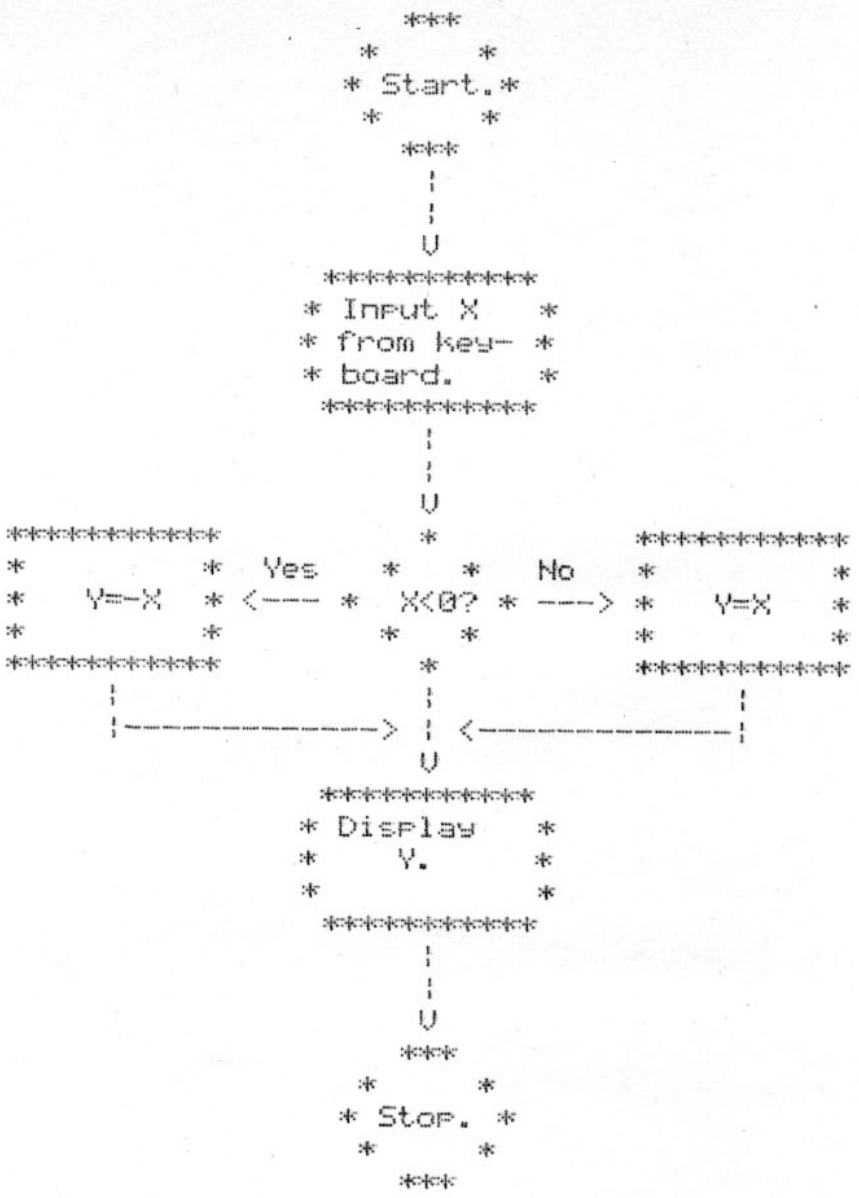
This article will discuss the topic of flowcharting as a program design tool, and why it has fallen into disfavor with many people. It will also make some points about program design along the way.

WHAT IS A FLOWCHART?

A flowchart is a graphic method of describing the logic, or flow, of a computer program, or of any similar well-defined sequence of instructions. According to the usual conventions, ordinary instructions are written into rectangular boxes. Decisions, where a choice must be made among two or more alternatives for what to do next, are written into diamond-shaped boxes. A circle might represent the start or end of a program, and a small circle might indicate a branch to another page of a large flowchart. Special shapes often indicate computer input or output. Usually, for flowcharts intended to be saved (as opposed to those scratch-paper diagrams that are used to help formulate ideas), a plastic template is used to help form the block shapes; also, special paper forms are available to help line up the blocks neatly.

Within the limitations imposed by my printer, the next page shows a very simple example of a flowchart for a program to display the absolute value of an integer X entered at the keyboard.

Structures such as FOR loops may be diagrammed by breaking them down into pieces (initialize index, increment index, test index for exceeding maximum value, branch back to beginning of loop), or by using special graphic structures invented to represent special program structures.



You will have to use your imagination to picture a 20-page flowchart for a large program. However, one important point needs to be made: a modular program will have a modular flowchart, which will be easier to read; an unmodular program will have a flowchart that looks like spaghetti (unless the programmer puts more thought into the flowchart than into the program's design).

Flowcharts are also used to describe systems of programs. For example, a system flowchart for the system I am using in editing this article would have a box for PIE (editing the text), a box for TEXT (displaying the results), a decision box (deciding whether the results are okay, with an arrow running back to the PIE box if they aren't), and another box for TEXT (printing the final draft). This article is not about system flowcharts, which are still widely used.

CONTROVERSY

I had better point out that this article presents my personal point of view, and that the subject is by no means one that everyone agrees upon. In fact, the title, which I chose when I started this series of articles, is very biased. Having recently done an informal

poll of several programmers at the company where I work, I was surprised to find the entire range of opinions, from people who don't use flowcharts at all, to those who feel they can't get along without them.

In the March 1982 issue of Microcomputing, the article "Designer's Delight" by David Carew says, "Flowcharts are excellent for depicting program logic graphically after the logic is done. They simply aren't very productive in the process of developing that logic." In the "Letters to the Editor" column of the June issue, James F. Davis replies to that comment as follows: "What a crock! I have been in this business over 25 years and I think I know a little about developing logic. The bottom line is that flowcharts are essential with few exceptions and I've found very few programmers who can function without them."

I disagree with the first statement by the first writer, as well as with the beliefs of the second writer. Flowcharts are of little practical use for depicting program logic after the logic is done, as well as being relatively unproductive during the design phase of programming. And I know many programmers who function quite well with little or no use of flowcharts.

HOW THINGS USED TO BE

Mr. Davis says he has been in the business for over 25 years. I have been in it for 20 years, and when I started, flowcharts were considered very important; Mr. Davis probably "grew up" with flowcharts, and has not picked up on newer programming methods. The company I worked for then required them for every program. What this meant in practice was that a programmer designing a new program scribbled crude pieces of flowcharts to help him or her visualize difficult areas of program logic. There was no need to do these flowcharts neatly, since the program logic was sure to be changed, either because of debugging or because of new specifications, before the program was finalized. Also, since the drawing of flowcharts was so time-consuming, and afterthoughts usually made a mess of the chart, there was a tendency to skip program design and go directly into the coding phase.

When the program was nearly debugged, or sometimes after it went into production use, a final flowchart was done, on special forms, using a template to draw the individual blocks. This satisfied the company requirement, which was probably aimed more at making the program easier to modify or debug, than to force programmers to use flowcharts as a design tool. However, when it was necessary to modify a program, if the programmer looked at the flowchart at all, he or she generally quickly gave up on trying to use it as an aid in understanding the program logic, and switched to using just the source program.

When changes were made to the program (and most of the work done on programs is maintenance after the programs are put into production), the flowchart often was not brought up to date. When it was, updating the flowchart usually required redoing at least an entire sheet.

At about this time, Applied Data Research was offering a software package called Autoflow. Autoflow would read a user's source program in Cobol (and perhaps other programming languages) and would print out

a flowchart of the program. The fact that this was a fairly successful package implies that many installations were using flowcharts as an after-the-fact form of documentation; either flowcharts were not used as a design tool, or the design flowcharts were not considered suitable as documentation.

Autoflow, and packages like it, solved the problem of keeping flowcharts up to date; whenever the program was modified, one merely ran the new program source through again, to produce a brand new flowchart. They also assured that flowcharts would be neat.

But they did nothing to make flowcharts really usable as documentation. A flowchart of a poorly structured program would still be poorly structured. And they did nothing to make the flowchart a more useful design tool.

NEWER METHODS

Techniques of programming continue to advance. Decision tables and HIPO (Hierarchy-Input-Processing-Output) diagrams are other, newer, somewhat graphical methods of program design, and may also be used as documentation. I will not describe these methods here. Instead, I will emphasize another method, which is sometimes called pseudo-English, or computer Esperanto.

Pseudo-English is any high-level programming language which is easy for humans to read and understand. Whether it is actually implemented as a language that a computer can compile is irrelevant. However, Pascal and Algol are two languages which are implemented, and which qualify as pseudo-English. When using pseudo-English as a program design tool or as program documentation, strict adherence to syntax rules is not important.

As an example, the logic of the flowchart above might be represented in pseudo-English as follows:

```
Input X from keyboard
If X < 0
  then Y = -X
  else Y = X
Display Y
```

In Pascal, it would be:

```
begin
  read(X);
  if X < 0
    then Y := -X
    else Y := X;
  write(Y)
end.
```

If you happen to be programming in Pascal or Algol, or even Cobol, you can design your program and code it at the same time. Use the techniques of structured, modular, top-down programming; indentation and blank lines to emphasize logical structure; and comments where the logic gets tricky (which it should do only when necessary). Then you won't even have to translate from the design language to the compiler language, and the source listing serves as its own documentation. (Actually, it only documents the program

logic; you will also need written documentation on the purpose of the program and how it interacts with other programs. I favor putting this into the form of block comments at the beginning of the source program.)

But pseudo-English is perhaps even more useful when the programming language is an unstructured language, such as Assembler or Basic. In this case you still have the advantage of using a high-level language during the design phase. At coding time, it is fairly straightforward to translate into the final language. For example, an If-Then-Else structure in pseudo-English will include, in Assembler, a compare instruction and one or more conditional or unconditional branches. Be sure to include the pseudo-English version in the source program as block comments, for documentation purposes. Also, for Assembler language, make heavy use of comments on individual statements.

CONCLUSION

Flowcharts, as tools for designing and documenting programs, are still in use; but they have several inherent problems, and it requires quite a bit of labor and conscientiousness to get them to do what they are intended to do. Pseudo-English is a better tool for both design and documentation, and makes the programmer's job easier.