Computers and the Humanities, Vol. 8, pp. 231–237. PERGAMON PRESS, 1974. Printed in the U.S.A.

FLOW: A Teaching Language for Computer Programming in the Humanities

JEF RASKIN

FLOW is an instructional computer programming language which eliminates many of the difficulties beginners usually have when learning to program. The language is very limited in scope, and the pedagogical emphasis is on algorithmic design rather than extensive language features. It is used only for the very first lectures in programming in college and secondary level courses, where it allows immediate hands-on use of the computer. FLOW is always quickly supplanted by whichever "real" computer language is most suited to the needs of the course. The extreme interaction afforded by low cost minicomputers is exploited by the technique of "typing amplification" which completes the typing of keywords once their initial letter is entered. FLOW has proven easy to implement on many computer systems, and has been effective with a wide variety of students over the past four years.

In the Report of the Conference on Computer Technology in the Humanities[1] the committee on courses and curricula concluded that a first course in computer programming in the humanities should "provide extensive instruction and practice in programming. Insofar as possible, programming should be taught algorithmically ... rather than as aspects of a specific programming language. Such algorithms ... provide an abstract structure onto which any given programming language can be mapped. Hence, the logical aspect of programming, rather than the details of a programming language, is given primary emphasis, and thus the inevitable shifts from one programming language to another necessitated by a change in computer or in research goals are facilitated."

An institute[2] in programming in the humanities was proposed and implemented on the basis of the recommendations of this and other committees and during that institute, held in the summer of 1970, the FLOW programming language was developed by the author. It met the requirements outlined above, and it has been in use since then, proving itself to be a powerful teaching aid.[3]

There are a number of impediments that come between the artist or humanities scholar and the art and technique of programming. These problems begin with the often expressed self-doubt of the students who feel that they are about to enter an alien field where they will be at a severe disadvantage. Later they will be forced to memorize arbitrary language conventions, they will work in a hostile and noisy environment and will have to learn many irrelevant facts about computers and operating systems. If they happen to take a course not designed for humanists they will be given problems to solve that are intended to be trivial. But the students will too often find that the problems themselves are as troublesome as the programming part of the exercise.[4] For the beginning student of programming any nonprogramming content serves to confuse the issue.

FLOW solves these as well as a few other problems that arise in the earliest stages of teaching programming. FLOW works especially well when it is embedded in a curriculum designed substantively and psychologically around the needs of the humanities or arts student or professional. Part of FLOW's attractiveness is that it can run on very small (and inexpensive) minicomputers and that it is easy to implement and maintain. The original implementation was written by two of my students and myself in one day.[5] FLOW is designed for interactive terminals, but a version has been used successfully

Jef Raskin is a consultant available to install his FLOW system. As director of the Visual Arts Computing Facility at the Third College of the University of California, San Diego, he taught the courses described here.

in batch mode. However, as many studies have made clear,[6] it is easier to teach programming with an interactive system.

A key problem in learning to program is frustration. There are few things so frustrating as being told that a comma is missing and then having to wait 20 minutes before it can be fixed – or receiving an error message that requires the combined heads of two systems analysts to decode. FLOW solves this problem by having no error messages. In the interactive implementation it is literally impossible to make a syntactic error. This is due to a feature called "typing amplification." A brief example should make it clear: A correct statement in FLOW is

PRINT "YES"

meaning, of course, to print the word "YES" on the terminal when the instruction is executed. The syntax of FLOW is such that when a "P" is the first letter of an instruction, the only legitimate instruction possible is a PRINT. Therefore when the user supplies the "P" the computer automatically supplies the "RINT". You cannot type "PRUNT" or any other error. If the first letter did not signal a FLOW command the mistyped character would appear on the screen briefly and then disappear, leaving the user none the worse for having hit a wrong key. The typing amplification feature is so thorough that if you sit at a keyboard and stroke the keys at random, a syntactically correct program will eventually appear on the screen. Furthermore FLOW is designed so that every syntactically correct program is executable. A side benefit of typing amplification is that students without typing skills are at less of a handicap. The only reason we use computers is to help make some tasks easier for us: to not use its abilities when entering programs is anachronistic.

Logical errors – those where the actions of the program do not meet the intent of the programmer – are another matter. If logical errors were impossible, there would be nothing to learn in programming. To help correct logical errors FLOW provides a powerful trace feature. This feature allows the program to be executed one line at a time with the results of each line's execution immediately seen and the program corrected if necessary. The trace is detailed below.

There are three main phases to programming, and a fourth phase, minor but tedious. The main phases are: defining the problem and specifying an algorithm (or method) that can be used to solve it on the computer; documenting that problem definition and solution; debugging the resultant program. The minor step is coding the method and the documentation[7] into some particular programming language. It is this step that typically forms the major part of the content of most courses in programming. FLOW is designed to minimize this minor and trivial step so that the student can concentrate on the important and intellectually satisfying phases of programming. The syntax of FLOW, when taught with this aim in mind, becomes natural before the student gets to the terminal: this usually is accomplished in the first hour of instruction. The claim here is not that FLOW is English or that it is known or even readable a priori by the student, but that it can become natural very quickly. There are no exceptions to FLOW's few rules and these few can be taught informally.

There is another source of frustration to the beginner: the operating system. Usually the student has to learn to operate the system before trying to enter and run a program. FLOW avoids this problem by being embedded in a transparent (i.e., invisible) system. Without such transparency the instructor is forced to use formulae: "just enter this incantation before using the terminal" or "put five cards ahead of your deck, using this recipe, we'll explain it later." This is not good pedagogy since it reenforces a rote learning pattern that is detrimental to programming.[8]

The teaching method is quite as important as the language features. A sequence of problems each only slightly more difficult than the last is essential. These problems must be very clearly stated, and given with examples of input and expected output: the problem statements must be models of good documentation. In the week or so spent on FLOW before the class moves on to a "real" programming language, between 10 and 15 problems are given. As can be seen from the list of problems[9] they are immediately comprehensible to anyone. Again, the difficulty lies in designing, documenting and debugging, the algorithm, not in struggling with the problem itself.

Every program is a communication between two persons (even if those two are the same human at different times). Some skill at natural language writing is necessary in creating documentation for a program. Documentation defines the problem that the program solves, explains how the program is used, describes how it obtains its results and gives internal information on the particular techniques and mnemonics used by the programmer. Some of

this documentation appears inside the program, although the computer does not read it. This information is for the programmer's own use. Its importance cannot be stressed too much. The other form of documentation is an external essay that is equally important in communicating what the program is intended to do, what its limitations are, and how it is to be used. This is literally as important as the program itself. The external documentation should be repeated in the internal form at the beginning of each program.

When required, the instructor must include some training in exposition as part of the programming course. FLOW makes documentation mechanically easy, the teaching stresses the need. No program, however accurately it may perform its task, should be accepted as complete by the instructor unless it is properly documented. The minimum criteria are: programmer's name and date; a statement of the problem to be solved; examples of typical input and expected output including special and difficult cases; and internal documentation that clarifies the logical flow of the algorithm. The trickier and more advanced the program, the heavier becomes the need for documentation.

Debugging a program is the process of determining if a program is functioning correctly, and fixing it when it is not. It often comes as a surprise to the neophyte that this phase of programming is easily as time consuming and challenging as the other phases. Debugging skills must be taught consciously. They are too often left for the student to acquire in an ad hoc manner. FLOW's trace feature greatly simplifies the task of finding where a program went wrong and exactly what happened there. The trace functions by executing the instructions that comprise the program one at a time. The programmer indicates that an instruction is to be executed by pressing a button on the terminal. FLOW executes the command and immediately displays the result of any input or output, transfer of control or other changes that may have occurred. Debugging includes assuring the correctness of the logical path taken through the program, and this path is very clearly shown in the trace. The command in FLOW that initiates execution with a trace is WALK – as a trace operates quite slowly. The command to execute a program normally is RUN.

While FLOW is a valuable teaching aid, especially as implemented on a system designed for teaching programming (like the one discussed below) it is not necessary. Before FLOW was invented this course was taught quite successfully in whatever language was available.[10] FORTRAN, Algol, BASIC, PL/1, APL, and SNOBOL have been used as follow-up languages in the beginning programming course, and any of them could be used from the outset so long as the problems are chosen carefully. Good teaching can overcome the tremendous handicaps imposed by obtuse operating systems, inconsistent language structures, complicated input and output schemes, unreasonable turnaround or response times, crowded and noisy computer centers, and unintelligible error messages.

After FLOW is mastered in the beginning course a "real" programming language is taught. This enables the student to become operational at almost any computer center with little retraining. One objection to FLOW is that the time spent on it could be better spent on the computer language that the class will have to learn anyway. But most students (and "professional" programmers as well) never master the fundamental skills of constructing, documenting, and debugging algorithms in any systematic fashion. In one experiment a class was divided into two groups: one was taught with FLOW and BASIC and the other was taught BASIC from the first. While this experiment was by no means rigorous the results seem to be typical. The group using only BASIC could use the language with greater facility but had severe problems when the exercises began to have a significant logical complexity. The FLOW group coped with the intricacies of the more complex problems without undue alarm and asked many more questions about details of syntax. This latter kind of question can be referred to a manual, the former cannot.

While FLOW by itself is helpful in teaching programming, it has been found not to work as well when the students must interface with an alien and noisy computer center. It also is rendered less effective when it is part of a science or mathematics curriculum rather than an arts or humanities course. The reason is simple: an incorrect environment excites the students' feelings of inadequacy in what they believe to be something they are not good at. Of course some students take to programming under any conditions. But they are not part of the problem FLOW was designed to solve. Hence the equipment and environment were designed to be reliable and quiet, and as uncomputerlike in appearance as possible. The terminals have what appear to be home TV sets on small tables, the chairs are either pillow cushions or small armchairs. The keyboards are separate from the rest of the terminals so they may be placed comfortably on the lap as well as on

desks. The usual glut of wasted paper common to computer centers[11] is avoided. Since the CRTs are self-illuminating, soft incandescent lights in Japanese lanterns have replaced the usual fluorescent fixtures. Details of the computers and terminals are given in the notes.[12]

Funding, always difficult in the arts (especially in the quantities required for computers), was through the University of California which matched a grant from the National Science Foundation.[13] There were other grants from sources within the University as well. While the original grants totaled $76,000, the same equipment at today's prices would cost less than $20,000. Subsequent expansion has raised the total capital expenditure to about $120,000. Again, the cost for computer equipment has steadily dropped and if one were starting anew more capability could be had for less.

The computers are used exclusively for teaching. These are a pair of identical machines set up so that if one fails, the other can take over its operation, thus assuring continuity of instruction and avoiding another source of student (and teacher) frustration. Any person experienced with computers knows just how clever a computer can be about choosing when to fail. This dual computer set has functioned twenty-four hours a day, seven days a week since its inception, except for downtime due to lack of personnel to oversee the facility. It usually runs unattended. During the period reported on here the average time between failures was about two months. During the same time the University Computer Center had a mean time between failures of less than two hours.

The total operation has been successful. The very smallness and simplicity of the physical facility prevents students from being intimidated by the equipment. The operation is low key and informal. The course, "Beginning Computer Programming for the Arts and Humanities," is one of the most popular at the University according to student evaluations, and it has been very well received at institutes and summer seminars. Many of the students have done fascinating final projects and have gone on to use the computer in their own work.[14]

Most recently FLOW system has been introduced into a Third World curriculum where, as all too often we find in the arts, there is a general antipathy to technology. Here, the students come to realize that there is tremendous potential in the computer, and if they wish to see the problems of the present rectified they will need all the tools they can get. FLOW has been found to be a valuable means for introducing them to the computer.

APPENDIX I
A BRIEF DESCRIPTION OF FLOW

This appendix is neither a manual nor a text, but is intended to give the reader who is somewhat familiar with programming languages a general idea of what FLOW looks like.

The implementation described here is designed for interactive CRT terminals such as the Ann Arbor, VST, Datapoint, or Hazeltine devices or their equivalent. The terminals should be operable in full duplex mode.

Each FLOW instruction requires one line, and each line is numbered. The numbers are always three digits, from 000 to 999. Line numbers are provided automatically, beginning with 010 and incrementing by 10 with each new line. These line numbers are used as statement labels. If the user wishes a line number different from the one provided, the desired number is typed and it replaces the supplied line number.

If, when a digit is expected, some other character is typed, then the incorrect character is not echoed and FLOW waits for a correct character. This action is consistent throughout the system, so that syntactical errors never get into the interpreter.

The system commands are: RUN which executes the program; WALK which executes with the trace feature explained in the body of this report; ERASE which deletes the program both from the screen and from the computer; DISPLAY which exhibits the program on the screen, edited to keep the line numbers in ascending order; NUMBER which renumbers the program. In each of these commands the user types only the first character of the command. Typing amplification supplies the rest. Commands are not preceded by line numbers. Each of these commands may be modified as follows:

ERASE FROM 038 TO 140
RUN FROM FIRST LINE TO 200
DISPLAY FROM FIRST LINE TO END

THE ABOVE APPEARS ON THE SCREEN

The characters actually typed to obtain these commands would have been:

E038140
RF200 *BUT ONLY THIS WAS TYPED*
DFE

Typing amplification provides conciseness of expression without sacrificing readability.

There are seven statement types in FLOW. The TEXT statement, analogous to a DATA statement in BASIC creates a text for the program to work on.

TEXT IS "THE QUALITY OF MERCY IS NOT STRAINED."

Unamplified this would be TTHE QUALITY OF MERCY IS NOT STRAINED." Notice that the open quote is supplied, but the close quote is not. If the programmer reaches the end of the line without a close quote, one is supplied and the line is terminated. This limits the length of a TEXT which is not a bad idea for beginners. There may be more than one TEXT statement in a program. The current text is the content of the one most recently executed.

The GET IT statement accesses the text one letter at a time. Executing a GET IT causes the value of IT to become the next character in the current text. IT is a blank if there is no text or the text has been used up.

Normally the statement with the lowest line number is executed first, followed by the statement with the next highest line number and so on. The statement JUMP TO # (where # represents a three-digit number) causes execution to proceed from the line numbered #. If there is no such line, then execution proceeds from the line with the nearest higher line number. If there are no lines to execute the program stops.[15]

The conditional branch IF IT IS "α" JUMP TO # behaves as a JUMP TO if the value of IT is the single character α, otherwise the statement does nothing. Unamplified this statement is Iα#.

To introduce documentation the COMMENT statement is used. After the amplified word COMMENT the user may introduce one line of commentary.

The STOP statement stops the program, and is not required at the end of a program.

Output is provided for by the PRINT statement which has a few forms: PRINT IT, which prints the current values of IT; PRINT "β" where β is any message desired not containing a double quote; and PRINT ON A NEW LINE which forces the next character to be printed to appear as the first character of a new line.

The following program finds if a word has either a "F" or a "G" in it. It is presented in amplified form.

```
010 COMMENT FIND IF A WORD HAS EITHER AN "F"
    OR A "G" IN IT
020 COMMENT BY LYRA FORET 19 OCTOBER 1971
030 COMMENT
040 COMMENT SOME TEST CASES ARE FOX, GO-
    PHER, RAT, DOG, CAT
050 COMMENT THE RESPECTIVE ANSWERS SHOULD
    BE YES, YES, NO, YES, NO.
060 COMMENT
070 TEXT IS "DOG"
080 COMMENT OBTAIN A LETTER OF THE TEXT
090 GET IT
100 COMMENT CHECK FOR A BLANK WHICH INDI-
    CATES END OF WORD
110 IF IT IS " " JUMP TO 500
120 COMMENT CHECK FOR F'S OR G'S
130 IF IT IS "F" JUMP TO 200
140 IF IT IS "G" JUMP TO 200
150 COMMENT IT WAS SOME OTHER LETTER, SO GO
    ON TO THE NEXT CHAR. IN THE TEXT
160 JUMP TO 080
200 PRINT "THE WORD HAD AN 'F' OR A 'G' IN IT."
210 COMMENT WE ARE DONE
220 STOP
500 PRINT "THE WORD DID NOT HAVE AN 'F' OR A
    'G' IN IT."
```

This program solves problem 12 in the appendix. It is shown with a typical execution.

```
100 COMMENT FIND IF A WORD HAS AN EVEN OR
    ODD NUMBER OF
110 COMMENT LETTERS IN IT. J. ETALOCOHC FEB
    24, 1974
120 COMMENT PRINT THE WORD AS WELL AS THE
    ANSWER
130 COMMENT SOME TEST CASES: ODD, EVEN, X,
    TWO, FIVE
140 COMMENT SHOULD HAVE THESE RESULTS:
    ODD, EVEN, ODD, ODD, EVEN
150 COMMENT
160 TEXT IS "EVEN"
165 PRINT " ' "
170 GET IT
180 PRINT IT
190 IF IT IS " " JUMP TO 280
200 GET IT
210 PRINT IT
220 IF IT IS " " JUMP TO 300
230 JUMP TO 170
280 PRINT " ' HAS AN EVEN NUMBER OF LETTERS."
290 STOP
300 PRINT " ' HAS AN ODD NUMBER OF LETTERS."
    RUN
    'EVEN' HAS AN EVEN NUMBER OF LETTERS.
```

APPENDIX II

A TYPICAL SET OF FLOW PROBLEMS

In class the following definitions are made: A "word" is a sequence of nonblank characters followed by a blank. A "sentence" is a sequence of

words the last of which is followed by a period instead of a blank. These definitions allow the class to work with realistic words and sentences without incurring arguments about just what constitutes a "real" word or sentence.

The problems are in the order that they are presented in class.

1. Does a word contain the letter "E"? The program should print "YES" for words such as EAT, MEET, MORE, and YES; and should print "NO" for SALAMI, BROWN, A, and NO.

2. Find if the letter "E" occurs two times in a word. The program should print "THE WORD HAS TWO E'S" for words such as MEET, EASTERN, and RECEIVE; but should print "THE WORD DOESN'T HAVE TWO E'S." for examples like EAT, FULFILLING, or COMMUNICATE.

The distinction between this problem and the one of finding if a word has exactly two E's is discussed in class.

3. Tell if a word has either a "G" or an "F" in it. These examples meet the criterion: GOOGLE, FORGE, FOX. These examples don't: BROWN, SOPHISTICATED, YOU.

4. Does a word contain both a "G" and an "F"? GOOF, FORGE, and LOFTING do whereas GOOGLE, FOX, and SKI don't.

5. Does a word begin with the letter "B"? BOY does, GIRL doesn't. This problem may seem out of sequence, but it is so simple that it makes those who are trying to learn to create algorithms by rote stop and think.

6. Indicate if a word both begins and ends with the letter "G" such as in the case of GOING or GONG. If the word, like GUSH, FROG, or MY, doesn't begin and end with "G" indicate that as well.

7. Print the words of a sentence in column form. That is change the sentence "FUR FEELS NICE." to the form
FUR
FEELS
NICE

8. Find if a word ends with "ING". SING and SINGING end in "ING", RINGS and FIG don't. This problem seems a bit easier than it is, the test cases INNING and SKIING often trip up attempted solutions. This problem is therefore useful in demonstrating debugging techniques.

9. Print an "X" for every word in a sentence. If the sentence were "THE OFFICE IS NEXT DOOR," the result should be XXXXX, that is, one "X" for each word in the sentence.

10. Print the given word, changing it only if it doesn't follow the rule "'I' before 'E' except after 'C'". The words EXTRA, RECEIVE, and SALIENT should remain unchanged, but CIELING should come out corrected to CEILING, and OREINT to ORIENT, and all similar examples. The program will misspell exceptions such as NEIGHBOR, SEINE, and WEIGH, but that is acceptable for this problem.

11. Given a sentence print the first letter of each word and then print each word that begins with "B" on separate lines. For example EVERY GOOD BOY DOES FINE BY AND BY. becomes
EGBDFBAB
BOY
BY
BY

This problem can be motivated by pointing out that the first half finds an acrostic in the sentence, and the class shows incredible ingenuity in designing meaningful sentences which form clever acronyms.

12. Find if a word has an even or odd number of letters. ODD is odd, and EVEN is even, but TWO is odd, and FIVE is even. Some students are not familiar with the concept of odd and even, but it is easy to explain.

13. Find if a word begins and ends with the same letter. TAPPET has the required property, VALVE does not. This is always given as an optional problem. It requires some insight. The class is warned that impossible problems may be given.

14. Find if a word has a TH two letters before the end. FORSYTHIA does, THEATRE doesn't. THTH does as do THTHTH and THTT. This problem is quite difficult, and taxes even experienced programmers, although a careful algorithmically trained thinker can get it right on the first try.

NOTES

Acknowledgments: I wish to thank the University of California, San Diego, Department of Visual Arts, for granting sabbatical leave so that this and other reports could be prepared. Thanks are also due to the Artificial Intelligence Laboratory of Stanford University for granting me a haven and computing facility for preparing this report.

I would like especially to thank Jon Collins, who made significant contributions to the design and implementation of FLOW and who has suggested many improvements in the teaching techniques.

1. Report of the Conference on Computer Technology in the Humanities, David Ohle, ed. (English Dept., University of Kansas, Lawrence, KS 66044), pp. 41–47. The Committee on Courses and Curricula was chaired by Sally Sedelow. The author was a member of that committee.
2. Summer Training Institute for Humanistic Computation, Floyd Horowitz, director, University of Kansas, Lawrence, KS 66044. The institute ran in two sessions from June 13 to August 18, 1970. The teachers were Sally Sedelow, Gerald Fisher, Jon Collins, and Jef Raskin.
3. A recent study, Donald A. Norman's "Cognitive Organization and Learning" (available from the Center for Human Information Processing, University of California, San Diego, La Jolla, CA 92037), uses FLOW as a medium for studying the learning process itself. It was chosen for the study since "FLOW is unique in several ways. First, it has been designed to simplify the process of entering information into the computer. At any point in the program, only the typewriter keys which lead to legal commands are operative ... In addition, by a system called typing amplification, whenever the user has typed a sufficient number of characters ... the entire command appears on the screen without waiting for the student to finish. Thus, by these two features, the most common problems for the beginner are eliminated: typing errors and difficulty with the keyboard."
4. An excellent article on teaching programming is Niklaus Wirth's "Program Development by Stepwise Refinement," *Communications of the ACM,* 14, 4 (April 1971), 221–227, who cogently asserts that "Programming is usually taught by examples. Experience shows that the success of a programming course critically depends on the choice of these examples. Unfortunately, they are too often selected with the prime intent to demonstrate what a computer can do. Instead, a main criterion ... should be their suitability to exhibit ... techniques."
5. Since this implementation in FORTRAN, FLOW has been implemented in NOVA assembly code, MICRO 800 assembler, BASIC and Algol. Given a specification for FLOW, a good programmer should be able to write an interpreter in a month. FLOW fits into 4k of 16-bit storage. Execution speed is irrelevent since "production runs" never occur. It does, however, run fast enough so that in the assembler versions there is no detectable pause between a command and its execution. The delay is under .1 second in the BASIC version on the NOVAs.
6. "... the course has not been effective. Most significant, it has been batch-oriented while [the computing community at that school] has made routine use of time-sharing ..."
7. Documentation occurs both as an essay which describes a program, and as commentary that becomes part of the program itself. Different computer languages have more and less difficult ways of recording internal documentation: FLOW makes it particularly easy. Placing internal documentation is a part of the coding process.
8. One way to achieve transparency is to use a minicomputer dedicated to FLOW for the first week or so. Once, when forced to use a large system, I convinced the system programmers to make FLOW available automatically whenever someone with my class code signed on. Such techniques are always possible, but may be easy or hard depending on the system and the cooperativeness of the computer center.
9. A typical sequence of problems is given in Appendix II. The sequence changes from term to term to prevent ennui.
10. Jef Raskin, *Beginning Computer Programming for the Arts and Humanities,* Manual G320-2044-0, available from the I.B.M. Corp., 112 East Post Rd., White Plains, N.Y. 10601.
11. "Computer Ecology," *Datamation,* 17, 11 (June 1971).
12. The equipment includes three Data General minicomputers, one teletype, five Video Systems Corporation VST 1200 CRT terminals, a Tektronix 4002 interactive graphics terminal, a Hewlett Packard 7200A plotter and assorted interfaces and acoustic couplers. The computers have 12k 16-bit words of memory. As of September 1973 32k was added to each computer, plus an additional 12 Ann Arbor CRT terminals.
13. National Science Foundation grant for improving undergraduate education in the sciences: in this case the science in question is, of course, Computer Science. The course in which FLOW is taught is considered alternative prerequisite to the Applied Physics department's beginning course.
14. Even a brief listing of the more interesting projects would take up more space than can fit here.
15. While the advocates of GO TO-less programming have some very strong points, with which I agree, most existing languages, especially BASIC and FORTRAN, require the student to be familiar with absolute and conditional jumps. The author's SIMPLE language is similar to FLOW, but is in the spirit of Dijkstra's observations about structured programming.