Journal of Student Financial Aid

Volume 12 | Issue 1 Article 3

2-1-1982

Learning to Program; A Self Help Idea for Financial Aid Officers

John B. Fisher

LEARNING TO PROGRAM; A SELF-HELP IDEA FOR FINANCIAL AID OFFICERS

by John B. Fisher

Financial aid officers need at least some expertise in many disparate fields such as accounting, counseling, employment, loan management, publications, etc. Despite already being involved with so many fields, I would suggest that the financial aid officer might well consider acquiring some proficiency in yet another area: computer programming. Of course, most aid officers at institutions with computers have learned much of the jargon and concepts which managers typically acquire. They know the difference between a printer and a CRT, between batch mode and on-line systems and they often talk about data bases, tape exchanges, system back-up, etc.

But despite this sometimes vast user oriented knowledge, few aid officers have considered learning to program as a way of gaining greater control of data processing and increasing the range of support services available to them from the computer. I would like to suggest that in many situations, learning to program might be both easier and of greater potential benefit than many might think.

I. Why Learn How to Program

Discussions about ways to obtain computer software for the financial aid office usually present a choice between purchasing from a vendor or developing a home-grown system. Packages from a vendor are expensive and often are not very well suited to the unique needs of a particular financial aid office. A home-grown system, one developed by the institution's own data processing staff, suggests the possibility of a system carefully designed around the procedures, priorities and goals of one's own operation. All too often, however, the institution's data processing staff can seem even more remote to the financial aid office than an outside vendor. An outside vendor, although working with the needs of financial aid offices in general, not yours in particular, at least has a staff of programmers more or less familiar with financial aid. The institution's own data processing staff is frequently not familiar with financial aid, and the high turnover which is typical of the field makes it difficult for programmers to develop a deep enough understanding of financial aid to exploit the potential advantages of a home-grown system.

One reason for learning to program might be to take the idea of "home-grown" one step further; that is, financial aid officers might write some or all of their own programs. While this certainly requires an investment of time and energy, the return can be enormous. Not only will the programs provide for the precise needs of one's own operation, but modifications required by changes in programs or procedures can be made without delay.

In the long run, many experts feel that we can expect to see software in many specialized fields come to be written by specialists turned programmers, rather than by programmers who try to learn enough about the field to write useful programs.

Mr. Fisher is the director of financial aid at Bloomfield College in New Jersey.

In the past, programming required very sophisticated skills normally acquired only by engineers. Today, programming languages do so much automatically, that learning to program is not an overwhelming task. In fact, it seems that those who possess the skills and talents that have enabled them to become successful financial aid officers, can probably learn, in a few months of spare time study, to write at least some of the software required by the financial aid office.

Programs differ greatly in the amount of programming skill required to create them. Perhaps the most difficult are the programs that maintain a data base of information on a large number of students. Such programs rely heavily on complex mathematical theories of searching and sorting. On the other hand, a program which performs various types of need analysis and eligibility index calculations is quite easy to write. Because of this, the situation which lends itself most readily to financial aid officer written software is one in which the financial aid office is already part of a system which incorporates financial aid information into the overall administrative data base. Since a primary purpose of incorporating financial aid data is to enable student bills to reflect offsets from financial aid sources, such systems often do not do much to make use of the information for the benefit of the financial aid office. Writing programs to benefit the financial aid office around this already existing data base, is an accomplishable task for a financial aid officer turned amateur programmer.

In some situations, it might not be possible or sensible for the financial aid officer to write any of the programs which are needed. The data processing staff may refuse to provide access to the machine. Or, in a very large school, the financial aid office might have its own full-time programmer/analyst. Even in these situations, however, knowing how to program, even at a rudimentary level, can be very helpful. Knowing the terminology which covers the structure and operations of computer systems is helpful, but does not really allow the aid officer to speak the same language as the data processing staff. Learning how to program will go a long way toward accomplishing this at least.

II. How to Learn to Program

A professional programmer will usually know how to program in many computer languages. A financial aid officer, with limited time available, should probably pick one language, and learn it as well as possible. The institution's computer and existing software system may dictate this choice. However, many computers offer a choice of languages. If it is one of the options available, the best choice is undoubtedly BASIC. Some languages, notably FORTRAN, are designed to perform mathematical calculations, but they do not have a natural ability to handle files containing large amounts of information. Others, such as COBOL, are capable of handling files easily, but are not designed to perform calculations. It is typical of financial aid applications, however, that a single program will need to handle files and perform calculations. A need analysis program, for example, might pull information from files, perform some calculations, then store the results someplace else. BASIC is a compromise of design features. It has a natural ability to perform many kinds of calculations, although it is not very good at more advanced mathematical concepts, such as recursive functions. Fortunately, these are never necessary in financial aid applications. In addition, most implementations of BASIC have a fairly good file handling capability, which should be adequate for situations, short of a system which must handle information for a huge number of students at a large university.

Perhaps the most important advantage of BASIC is the wealth of opportunities to learn it. There is more introductory material on learning BASIC than any other

computer language. (Lien, 1977 and Blackwood, 1981 are two excellent examples.) Of course, to learn to program you need more than written material, you need a computer on which to practice. BASIC offers options other than using the institution's main computer since it is the universal language of personal microcomputers. Access to one of these for several weeks should enable you to learn BASIC well enough to begin writing some truly useful financial aid programs.

Of course, one could also learn by enrolling in a formal course of instruction. I think most aid officers will find this process frustratingly slow, however. The fundamental concepts of programming, particularly in BASIC, can be learned very quickly. Beyond this, developing proficiency as a programmer involves acquiring an increasingly large repertoire of tricks; i.e. clever ways of writing routines which enable the computer to perform increasingly complex tasks with greater speed and efficiency. The fastest way to achieve the ability to write one's own financial aid software is to start writing programs, see what problems and limitations are encountered, and then learn techniques to deal with these specific problems.

III. What Kind of Programs Can you Write?

I. Need Analysis

The easiest financial aid programs are those which perform need analysis calculations. The most difficult aspect of programming is translation of a general idea into a very precise and specific series of steps that the computer can carry out, i.e., an algorithm. A need analysis formula is already such an algorithm. Writing a program to carry it out requires three steps. First a routine must be written to enable the computer to acquire the data required to carry out the calculation. This is easily accomplished in BASIC with a routine which enables the computer to ask the appropriate questions in turn. (Later, however, you may wish to use the more sophisticated technique of printing a "form" on the screen.) Second, the need analysis algorithm must be rewritten using the specific words and phrases available in the BASIC language. This is also easy because the terms available in BASIC lend themselves naturally to this sort of calculation. Finally, a routine is required to report the results of the calculation. This can be anything from a simple series of PRINT statements, to an elaborate report similar to those provided by the College Scholarship Service and the American College Testing Program.

A good program to start with might be the one that performs the calculation to determine a Pell Grant Eligibility Index (or Student Aid Index as it will be called starting with '82-'83). After you've gotten this program debugged and running nicely, you will want to expand it so that it performs all of the calculations used in your operation; uniform methodology, calculation for state program awards, etc. You now have a program which will be immediately useful to help with the task of recalculating eligibility when information on an application needs to be changed. Of course, this same capability is already available with various company-produced calculators. However, a home-grown need analysis program offers the following advantages which can't be obtained in any other way:

a. You can include tables in the program so that the results are not just eligibility indices for entitlements such as for Pell Grants, but actual award amounts for your institution.

b. If anything in an eligibility formula or payment table changes at any point, you can make the changes in your program immediately.

c. The need analysis program can be modified to provide various kinds of estimates and projections. A simple modification would be a program which displays the effect of one variable, say income, on another, say Pell Grant eligibility. BASIC provides a simple way to incorporate any

routine as part of a loop in which the routine is run a number of times, each time changing one or more variable in specific ways. This concept can be applied to do increasingly elaborate projections. Ultimately, one can have the program take a new academic year's methodology and sum up the results of applying it to either the current year's aid population or some hypothetical aid population devised from enrollment projections. This activity can yield, among other things, an estimate of the need for institutional grant funds.

d. The need analysis program can be used to provide an estimate of financial aid eligibility for prospective students. There is often pressure on the financial aid office to provide this kind of service, but the time required to perform a need analysis by hand can severely limit this option. A need analysis program can be used by the financial aid office to make the process speedier, or the program might be made available to the admissions office which could then on its own develop estimates directly.

2. Award Letters

A second type of program within reach of the amateur programmer is one which prints financial aid award letters. (See illustration 1) Assuming that the system is already maintaining a data base on financial aid awards, a program can be written which automatically prints award letters. While such a program itself should not be very difficult to write, it is very important to think through the whole idea in advance and make a number of key decisions, before starting to write the program. Some of the most important questions are:

a. How will those for whom award letters are to be printed be selected for each run? Do you want award letters printed for all those who have had their aid changed since the last run; should only certain changes generate an award letter; or, do you prefer simply providing the computer with a list of those who need award letters on each run?

b. What order should the award letter output be in? Some possibilities are alpha order, ID number order, or in the order that the aid changes were made.

c. How do you want to treat entitlements for which you have estimated an award that is not yet finalized (e.g., a Pell Grant estimate for which the student has not brought in a Student Eligibility Report). Do you want the award listed at all and if so how will it be made clear that it is still an estimate?

d. Do you want to include a figure for cost of attendance and if so, how will the system determine the appropriate figure?

e. Do you want the system to print just the unique data on a pre-printed computer form (which will result in shorter printing times) or use the full text of a letter (which may have a better appearance).

3. Other Aids to Office Management

In addition to need analysis and award letters, there is an infinite variety of potential programs which can help the aid officer. Programs to track students, generate lists, compute totals, etc., can all be important. In addition, the ability to make even slight modifications to existing programs can often make a world of difference. Having learned to program, most aid officers will find the computer a valuable tool rather than an alien presence with its own set of rigid demands.

IV. Technical Tips

This section presents some technical suggestions on writing financial aid programs. To some extent it is an attempt to emphasize general principles of good programming which happen to be particularly important in financial aid. In at least one case, however, it is important to do just the opposite of what programmers are taught.

I will assume some knowledge of BASIC programming in what follows, but it should be possible to understand the ideas before one's study of programming has progressed very far.

1. Write Structured Programs

The concept of a "structured program" is being emphasized generally in programming these days. It is particularly important, however, in writing financial aid programs. The disadvantage of structured programs is that they tend to be a little longer than they need to be. A great advantage, however, is that they are much easier to change or modify for other purposes. Many financial aid programs will need to be changed annually, or even more often than that. This will only be possible if they have initially been written in a structured manner. An excellent guide to the techniques of structured programming in BASIC is Negin and Ledgard (1978).

2. Learn to Write Good Input Routines

Financial aid programs typically require a great deal of data entry, much of it numerical. Financial aid officers who want to write useful programs should take some time early in their study of programming to learn how to write professional input routines. The idea is to avoid having a program "bomb" (i.e., stop running) when an obvious mistake (like entering a letter when a number is required) is made. Rather, the program should point out the error and guide the operator in making an appropriate response.

Some good advice on writing input routines can be obtained from Roche and Nemzow (1981) and Todd (1981). An example of a routine which edits commas and other extraneous characters from a numerical input is shown in Illustration 2.

3. Don't Be Afraid to Disregard Good Programming Principles

A general principle of good programming is to condense a series of operations into as few lines as possible. This has two desirable results: it makes the program shorter, and it reduces the number of distinct variables required. Consider, for example, Illustration 3, which shows a routine to derive the '81-'82 uniform methodology parental contribution for dependent students. A professional programmer looking at this would see many opportunities to condense lines. For example, lines 6430, 6435, 6455, 6460, 6465, 6470 and 6515 could be condensed into one line which reads: C5 = I(5) + (I(6) * .0613 MIN 1588) + (I(7) * .0613 MIN 1588) + (B3*B7) + (I(10)-(B3*.*03 MAX O)*) + (B1 MIN 2400) + C3. While this condensation reduces both the size of the program and the number of variables required, it is not wise in writing a need analysis program to carry the process too far. A key feature of need analysis programs is that intermediate results of calculations need to be used later, either for further calculations or as part of the output which is reported. The contraction above, for example, makes it impossible to retain values of FICA, medical deduction and employment allowance.

Another reason not to condense too much is that when writing a need analysis program one must always keep in mind that it will be necessary to make changes each year. To take another example, lines 6530, 6540, 6545, 6550 and 6555 could be condensed into:

D1 = (I(12) + I(13)-I(14) + I(15)-I(16)-C(((I(1) MAX 40) MIN 65)-39,A5)). If, let's say, the age range in the asset protection allowance should change in the future, it would certainly be easier to change in the original version, where the age figures are part of short, clear lines.

4. Put Tables into Files

BASIC allows one to include numerical information such as the tables required for need analysis directly in a program as lines of DATA. While this may be adequate when getting started, it will be much better in the long run to put this data into files which a program utilizes as necessary. There are two major advantages to this. First, they will make the programs shorter. Since each computer has only a limited amount of work space for each program, and since financial aid programs can become quite long, savings in space can become significant. Secondly, these tables need to be updated often. It is much easier to make the changes to tables stored in files than when they are embedded in programs. In fact you can develop separate programs for the purpose of managing the tables. Such programs can display the information currently in the tables and provide a convenient format for making changes.

V. Conclusion

The purpose of this paper has been to suggest what may be a new idea to financial aid officers. As computer systems play an even larger role in student financial aid, the aid officer is in danger of losing further ground in his or her attempt to remain in control of the process. It seems to me that learning to program can be a key element in the effort to retain the control needed to manage aid programs effectively.