DOCUMENT RESUME

ED 060 627 EM 009 633

AUTHOR	Bork, Alfred M.
TITLE	Learning to Program for the Science Student.
INSTITUTION	California Univ., Irvine. Physics Computer Development Project.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	12 Jun 71
NOTE	11p.
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	Computer Assisted Instruction; Computer Programs; *Computer Science Education; *Programing; Programing Languages; Science Curriculum; *Science Instruction; Teaching Methods; Technical Education

ABSTRACT

The science student has many avenues to learning how to program, including learning directly within the science course, in a special course on programing, or by self-study. Often a formal programing course is neither necessary or advisable. In general a pedagogical approach, aimed at bringing the student quickly to using the language, is better than a stricter, more logical one. Thus teaching program languages is similar to teaching foreign languages. Either a time-sharing or a batch computer may be used. It is best to make the student move quickly to program writing in a particular subject area, so that he may become motivated by solving relevant problems. (RB)

LEARNING TO PROGRAM FOR THE SCIENCE STUDENT

Alfred M. Bork
Physics Computer Development Project
Department of Physics
University of California, Irvine
Irvine, California 92664

June 12, 1971

Abstract

The science student has many avenues to learning how to program; often a formal programming course is not necessary or advisable.

As computers are being increasingly used in the teaching of science, students (and instructors) are more and more faced with the question of learning the programming languages necessary to use the computer. At first glance this task may appear formidable, but rapid and effective ways exist for learning how to use the standard programming languages. Because of his background, the science student has some special advantages. My own experience is with physics, but I believe that the situation is not too different for science students in other areas.

Separate Programming Courses

One can learn how to program in many ways. We can profitably distinguish between courses particularly set up to teach programming, the learning of programming directly within the science course, and the learning through self study. The natural inclination of many teachers is to assume that the student needs a course devoted exclusively or primarily to programming, but this does not turn out to be the case. Learning to program within a modern computer environment, either in or out of a formal course, is a relatively simple task for the science oriented student, and can be accomplished within the science course or through self-study.

That is not to say that it is necessarily bad for the student to take a separate programming course. A beginning programming course could, and should, offer far more to a student than simply learning a language. It should teach the student much about the nature of the algorithmic approach to problem solving, useful far beyond the writing of computer coding, it can give him insight into computer hardware, and it can also provide him with useful numerical and symbolic techniques which he can extend as his needs expand. However, if the course available to the student does nothing but teach how to write programs, it is, I argue, probably an inefficient use of the student's time, because it spends far more time than is necessary to accomplish this task.

Such programming courses work under a number of handicaps. First, they suffer from the fact that very little common background among the students exists, so problem assignments which require students to write other than trivial problems are almost impossible. The same problems appear in book after book--such problems as the solutions of quadratic equations. These tasks are often not the type of problem that one should solve with a computer. Furthermore, the mathematical talents and insights of the students are likely to cover a wide range and this may imply an approach which is too simplistic for the science student. Both of these points are made, in a different context, in a Kingly Amis novel, I Want it Now:

> "I was only nineteen, but I'd had so much sex already then that I thought I knew all about it. I thought I couldn't not know all about it. What I didn't know was what it was for.
>
> I was like someone who knows exactly how a railway engine's put together, and who can put his finger immediately on any part you care to name with his eyes shut, but who it's never occurred to that the point of the bloody thing is that it pulls trains. You do see what I mean, don't you?"

Batch vs. Timesharing

Although the student or teacher must usually accept whatever computer facilities are available, in some situations a choice between batch and timesharing may exist.

The terminal environment provides a very efficient way of learning a language. The student can get large amounts of practice in a relatively small time, if sufficient terminal time is available. Even more important, he can obtain instant error messages, correcting almost immediately the troubles which occur in his particular program. This rapid feedback is a powerful teaching device. The programming language itself, and the implementation existing on a particular machine, takes over some of the role of teaching the material, providing the language is well-implemented. (Later, I will point to a method of learning a language which systemizes this approach.)

On the other hand, the batch computer with jobs originating on cards, usually implies a slower, more laborious approach, particularly if a long delay occurs between the time the student submits the program and the time he retrieves his output. In the batch environment the student can learn more quickly if he can get directly at the computer, perhaps even putting in his own cards and pushing the buttons on the console. This is possible in small computer systems, and perhaps also in larger systems having remote user-operated card readers. Many batch systems offer priority to small student jobs, as compared to research jobs, and this again improves learning efficiency. But some batch facilities tend to discourage student learning programs, particularly if they are not associated with particular courses.

Negative Advice

Whether one is considering a separate programming course, or the learning of programming within a science course, or on ones own, a number of general points can be made. We will start with some "what not to do" comments.

A common, but ineffective, way of teaching programming languages in classes and in books is what might be described as the "grammatical" approach, the logical step by step exposition of the grammar of the computing language. Thus many FORTRAN courses start by defining fixed (integer) and floating (real) variables, assignment statements, etc.. This approach appears to teachers who are logically oriented; they order the grammar in a logical fashion, and use this order as the basis for the course. But a difference exists between a logical approach to a subject area and a pedagogical approach, a way of bringing students quickly to using the language. In the logical approach the student takes a long time to get to the point where he can write programs, and he does not gain much feel for the nature of the programming art because he spends so much of his time on the rules of grammar. This stragegy at one time was widely used in the teaching of foreign languages, but now, I believe correctly, has been discredited. However, many, perhaps even most, approaches to teaching programming languages still proceed this way.

Another fallacy in teaching beginning programming is the belief that a student must learn all about the language facility available, must learn the full language. It is sufficient for the beginning student to absorb only a subset, which may be oriented toward his particular needs and interests. With modern and complex languages, such as PL/1 or APL, the task of learning the whole language, and using all the facilities wisely, may be formidable, and so frightening to the beginning student; but he can learn a subset much quicker. Then as his needs increase, he can learn more of the language.

A final suggestion on the negative slide has already been implied. It is not reasonable to ignore the background and training of the student. For the science student the most profitable approach in learning about computers is through a particular subject matter area. Working within physics, or some other scientific discipline, the student is presented with real subject-motivated problems which demand the use of the computer, and which therefore motivate the learning of programming languages and demonstrate the relevance of computing in the area involved. The material is not motivated by the general idea that one "should" learn to program, but by the needs of the physics being studied. So the tool is not learned abstractly, but rather in the context of subject-matter use. This not only provides a powerful motivation for learning, but from earlier exposure gives the student some understanding as to where the computer can be effectively used.

Positive Advice

One important clue to the learning of programming languages, comes from the experiences with the audiolingual method in teaching foreign languages. This method starts the beginning student with some whole dialogs, conversations which he memorizes the learns to pronounce correctly, without worrying too much about grammatical details or the meaning of individual words. Thus he focuses on the language from the very beginning where it will actually be used, at the whole-sentence, or collection of sentences, level.

The corresponding unit for most computer programming languages is the single program; the "whole program" approach in learning a computer language, introducing the student to the language through showing him initially complete programs, is analogous to the audiolingual method for a foreign language. Just as with French dialogs, one does not start with lengthy programs, but with relatively short (but meaningful) programs, perhaps related to the subject matter with which the student is working.

If a student does find that he must work with a gramatically oriented book, he should try to move a quickly as possible to the writing and running of programs, getting away from pure reading as soon as possible.

It is important even from the beginning that problems be realistic, leading to programs one would actually run on a computer. If a task would be much more quickly accomplished by small hand calculation the student loses any sense of what is appropriate and important for computer calculation. In physics, for example, simple numerical solutions of differential equations, which can be done in very small programs, are a useful way to begin. The harmonic oscillator problem has been shown by many users to be an area of physics where one can successfully combine a high level of physics with the learning of programming language. Material exists here in a number of different languages.

While initial programs may be provided by the instructor, perhaps in printed form, the student should move as rapidly as possible into his own programs. One technique is that of assigning first tasks which involve relatively small modifications of the initialy supplied programs. Thus the student does not have to grapple with all the details of a full and complex program, but can begin to learn by changing existing programs only slightly. These changes would depend on the subject. To expand the suggestion of the harmonic oscillator above, the program could be modified in various ways to treat the motion of other one-dimensional systems. For example, it can be extended into a program for the damped harmonic oscillators; only a few statements in the total program must be changed, and the student can work with these rather than be faced immediately with the task of writing a completely new program, thus not encountering all the difficulties at once.

There is something to be said for the student initially learning not one program language, but two (or more) contrasting programming languages. Programming languages have a problem common to all languages, including natural: When a student is familiar with only one language he does not understand that language as fully as he might, because he has nothing to contrast it with. If a person knows a variety of languages he can see what features are common and what features are different. So by simultaneously studying two or more programming languages a student can get a better felling for the programming situation.

If two languages are taught, it is wise to pick languages which have substantial differences in outlook or structure. For example, one could combine the teaching of an algebraic language, such as APL or

PL/1, with the teaching of a language oriented primarily toward string manipulation, such as SNOBOL. This multilanguage approach offers insight into programming not available if one stays with a single language. Experience indicates that students do not find this a confusing situation. They can distinguish which language they are using and effectively learn to use both languages at the same time.

Ten Finger Exercise

One way of learning a programming language, while not applicable in all situations, has proved to be particularly effective at the University of California, Irvine. This method of teaching a language presumes a terminal environment and a language with "immediate" facilities, such that individual statements in the language can be executed rather than whole programs. APL, JOSS, and some forms of BASIC are so implemented.

We have called the approach the "Ten Finger Exercise" way of learning about a computer language. The student receives little or no instruction in the language before he sits at the terminal for the first time. Furthermore, he does not have any of the usual manuals and texts, at least at first. What he has is a sheet which instructs him as to what to type in, line by line; he is told to type in each line, and then push a carriage return. He usually gets some reply each time from the computer, and this information begins to show him what the computer does. The sentences which he types are arranged in an order to elucidate the facilities of the language being learned. Occasionally, also on a printed page, he will be asked a question, just to make sure that the student has picked up whatever point the

teacher is trying to make at that time. He quickly learns that he can do additional experimentation of the same type he has been using already in answering the questions.

Thus the student sees no didactic material, no grammatical information about the language, hears no lectures, but he learns to use it simply by interacting with the language compiler itself. This technique can be compared to the biologist learning about a strange animal by giving it a series of selective stimuli, and watching the behavior of the animal in response to those stimuli. Its success depends on the teacher formulating a reasonable progression to move a learner through the language. One does not start with the difficult ideas, but rather with the simple ones. Considerable experience is needed to guide the student through the likely difficulties.

The accompanying material shows some beginning material of this kind both in APL and JOSS (for a variant of JOSS called PIL), the two languages used this way at Irvine. Thus, in the APL material when the student types 2 + 2, and pushes a carriage return, the terminal immediately types 4. He quickly gets the idea that the computer is doing the arithmetic he asks it to, and picks up how to instruct the computer to do such calculations. Even the error messages in the system are useful in this mode of teaching; material can be consciously planned so that occasionally a student <u>will</u> get an error message, so he will learn what he can do as well as what he cannot do.

This Ten Finger Exercise approach has been used both with high school and college students with success. For example, in a high school group of mixed 9th to 11th grade students, chosen because of their interest in science, a group of students learned to program in a useable subset of APL in only four hours of time, all spent using the Ten Finger Exercise material at the terminal. In addition to teaching the language rapidly, this stragety gives the student the idea that he can, by experimentation at the terminal, continue to grow in his knowledge of the language.