DOCUMENT RESUME

ED 084 889 EM 011 713

AUTHOR Smith, John B.
TITLE An Advanced Sequence of Computer Courses for Humanities Students: The Penn State Program.
INSTITUTION Pennsylvania State Univ., University Park.
PUB DATE Aug 73
NOTE 14p.; Paper presented at the Conference on Computers in the Undergraduate Curricula (Claremont, California, June 18-20, 1973)

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS *Computer Assisted Instruction; Computer Programs; *Computers; Computer Science; *Computer Science Education; *Humanities; *Humanities Instruction; Problem Solving; Program Descriptions; Programing Languages; Undergraduate Study
IDENTIFIERS Natural Language; Pennsylvania State University; PL1 Programing Language

ABSTRACT

A series of computer science courses at Pennsylvania State University is designed to meet the needs of undergraduate humanities students who wish to use computers. The first of three integrated courses exposes the student to the range of computer applications in the humanities and teaches him to write nontrivial programs in the PL/1 Programing Language. Instruction is arranged around programing problems; students survey the literature of computer applications and do a design project. The second course concentrates upon teaching students how to break complex tasks into their components. Natural language use is stressed and additional technical information is presented, including matters such as job control language and system utilities. In the third course the student solves a complex problem. He develops a thesis, translates it into operational terms and computational procedures, performs an analysis, interprets the results, and maps the results back to the level from which he began. The sequence has been judged successful since it teaches students both the general techniques of problem-solving and, more specifically, creative, substantive ways to use the computer in the humanities. (PB)

An Advanced Sequence of Computer Courses for Humanities Students: The Penn State Program

By John B. Smith
Department of English
Pennsylvania State University
University Park, Pa. 16802
Home: 814-238-2767
Offices: -865-9681
-865-0438

Introduction

During the past five or six years the growth of special courses in computer techniques for humanities students has been dramatic. The September, 1972, Computers and the Humanities indicates that there are at least forty institutions over the country offering introductory courses. Since the survey was conducted by voluntarily submitted forms, this number is probably conservative. This indication of awareness of the necessity for courses sensitive to the needs and intellectual habits of humanists is most encouraging; however, if humanistic studies involving the computer are to move from the level of the mechanical to the creative, if the computer is ever to become an active, easily and naturally applied tool at the undergraduate level, colleges and universities must offer more advanced courses, perhaps concentrating on specific subject areas. Of the ten universities listed in Computers and the Humanities who offer more than one course for humanities students only five offer advanced courses that build on and extend the introductory material. In the remarks that follow, I shall try to show not only the necessity of advanced courses, but the necessity of a tightly integrated sequence of courses each building on the preceding and leading to the course that follows. Since I have proposed and taught at Penn State a program of this sort I shall cast my remarks in the context of this experience.

Rationale for Sequence

Any rationale for a sequence of special courses for humanists must be based on the realization that their mode or habits of thought differ from those of their counterparts in the sciences, business, and agriculture.

Too often, particularly among computer scientists, humanities students are regarded simply as slow if not stupid when it comes to learning computer programming. Much has been written about "the two cultures" or differences in quantitative and verbal reasoning to warrant comment here; a more critical factor has gone relatively unnoticed. Science students and students in the humanities often differ considerably in their tendency to move from the specific to the general and, more importantly, their sense of comfort with situations in which such jumps are impossible. For example, the student of literature is consistently asked to observe in his readings small, subtle patterns among words and ideas and then to extend these observations, quickly and too often imprecisely, to generate abstract theses concerning whole works, canons, and, sometimes, entire literary traditions. If the student works in this manner long enough this mode of thought becomes habitual, if not reflexive. A mathematics undergraduate, on the other hand, may find such jumps quite foreign. For example, a student studying modern algebra begins with a set of axioms and then slowly and deliberately constructs a mathematical system. He must, of course, have some overview of the direction in which his efforts are going; but most of his time is spent focused on specific problems. Few would argue that one mode of thought is inherently "better" than the other, but recognition of their differences leads to different approaches for teaching computing to these two types of individuals. Because of the large volume of unfamiliar detail, the introductory course must be structured in a way that gives the humanities student a sense of continuity and total organization.

For the physical or social science major, the computer fits closely familiar research designs so that once he knows <u>how</u> to use the computer he knows what to use it for. This is not true for the humanist. The work he does in his courses and research projects often involves high levels of abstraction and subjectivity. Once he knows how to program, he may be able to see the usefulness of a concordance, a word index, a reverse spelling dictionary, or a list of relative frequencies among words; but he is unlikely to see right away the more interesting and creative uses to which the computer can be applied. For example, it is unlikely that he will see on his own the application of Fourier analysis for thematic studies because it is quite unlikely that he is familiar with this model if, in fact, he has ever heard of it. Even if presented with distribution graphs of themes, the unfamiliarity of thinking visually or spatially about a piece of literature or music necessitates considerably more exposure to and instruction in these new analytic tools for the humanities student than for others. He must have time in which to assimilate these models and methods in addition to learning rudimentary programming skills. Only then will he be able to make the broad conceptual jump from mechanical skills to humanistic interest that will result in intellectually developmental uses of the computer. In addition to time, this takes an integrated program that can connect the details of programming and control languages with substantive applications in the student's field. The sequence of courses described below represents an attempt to establish such a curriculum.

First Course

The first course has two major objectives: to develop the student's competence to write nontrivial programs in PL/1 and to expose him to a broad range of humanities applications. In teaching programming, one must have some sense of the culture shock that hits many humanities students when faced with the overwhelming detail associated with the computer. The situation is analogous to what someone from Appalachia would experience if suddenly placed in the middle of Manhattan and told to function. Much of what one needs to know to get along is unconscious; consequently if he had an experienced "instructor" at his side it would be impossible to anticipate all of the novitiate's needs and impossible for the novitiate to remember the information if it were given. With encouragement, however, the student will survive if he endures and works hard at learning the intricacies of his new environment.

Translated into an introductory course in computing, the instructor must empathize with the discomfort the humanities student experiences because of his need to grasp the structure of the whole. Although this need cannot be met in the first few weeks, there are ways of organizing the material that make it easier for the humanities student. It is most important to provide some larger context in which the details of a programming language can be placed. That context used in most introductory courses taught by computer scientists is the logical structure of the programming language itself. When they cover I/O they are likely to discuss a number of the various options and control words in the language, comparing and contrasting their various features. Similarly, if discussing looping they often introduce the DO statement with all of its optional features. For the science student who has a sense of where all of this is leading and how it may eventually be applied to a real problem, the approach is fine; but for a great many humanities students, it is disastcrous.

A more reasonable approach *for the humanities student* is to organize the material around a sequence of programming problems with which he can identify. If the problems make sense to him, it is much easier for him to learn the concepts necessary to solve them. Since most humanists share an interest in verbal materials, I use some four or five problems related to text processing. The first assignment is to write a program to read in several lines of text and print them out. Next they read in the text, extract the words, and print them, one word per line. Then they read in the text, extract the words, sort them into alphabetical order, and print them. The fourth exercise is a binary search for particular words in a sorted list. By the time the students have completed these four problems they will have used most of the basic features of the language. It is a source of the greatest comfort to these students to know that it takes about twenty new control words or concepts for the first problem, ten additional for the second, five for the third, and only two or three for the fourth.

As fewer syntactic forms are introduced, more and more emphasis is placed on algorithmic principles. For example, when the concept of a subroutine is introduced with the binary search, it represents a relief from keypunching all those cards again as opposed to any really strange new principle that must be grappled with. These four exercises consume four or five weeks of the term; having survived this, the student is now ready for a relatively quick overview of the language that introduces additional features and places them in context with those they already know. By this time the student has at least a rudimentary knowledge of how to use the computer.

Giving the student a sense of what to use it for is much more subtle and takes much longer (in fact, this is the major focus of the two succeeding courses). A start is made by having the students read a great deal of outside material describing actual applications in the humanities. To assist this process, I make available to them a bibliography of some eight thousand titles I have accumulated along with a retrieval program to access it. During the second half of the course I also lecture more and more on applications showing as closely as possible the actual computational steps involved.

The major programming effort of the second half of the course concerns a term project of the student's design. This experience gives them a chance to integrate the various facets of the course and, indeed, a good deal of their undergraduate curriculum. Typical of the projects produced at this level are one line poetry generators, random number art, and thematic studies of short stories.

The first time this course was taught, we began with twenty-six students; twenty-three finished. Of the twenty-three that finished, eighteen took the second course along with several auditors. This record, we felt, strongly indicated the validity of the course's underlying rationale.

Second Course

At the end of the first course students have a basic knowledge of PL/1 and have been exposed to a variety of applications in the humanities. Students who stop formally studying the computer at this point but who wish to apply it to their interests often find themselves in an awkward position. Having the competence to write an exercise-type program and the intellectual exposure to at least get some glimpse of the potential power of the computer, they often are over-optimistic in the tasks they assume and find themselves embroiled in unrealistic and inefficient programming projects. The frustration that results is sometimes enough to undo the previous semester's work and turn them off from computing completely. The second course attempts to avoid these difficulties by considering the way complex tasks can be broken into individual programs or job steps. Thus, it focuses at a higher level, considering the individual program in the context of the larger resource environment of the system. Because of the backgrounds and need of the humanities students enrolled, this second course also emphasizes natural language techniques although I am careful to emphasize the generality of the principles involved.

Text processing is viewed from a systems analytic perspective. The process of analyzing natural language is broken down into the following sequential steps: encoding, scanning, sorting, file-structuring, and accessing. By realizing that what one gets out at the far end is determined by what one puts in at the near end, the student is able to resolve many of the seemingly arbitrary decisions that plague the novice. For example, if one wishes to consider segmental length distributions, he had better include punctuation marks when he types or punches the text. The student can thus establish principles that guide him without restricting himself to arbitrary conventions that make specific tasks awkward.

The power of this approach is particularly evident in the matter of data structuring. By classifying the kinds of tasks performed in language analysis and the particular data organization they require, I am able to show inferentially the necessity for a random accessible file structure that makes each occurrence of each word available along with its complete context. (See John B. Smith, "RATS: A Middle-Level Text Utility System," CHum 6,5 (May, 1972), 277-283 for a detailed discussion of this text system.) Having thus classified problems and noted the utility and restrictions of various computational approaches, the student is better qualified to analyze subsequent problems and anticipate possible difficulties, preparing him to use the computer in honors-type projects or, possibly, in later graduate work.

The second course also contains much additional technical information. To combine various program modules to perform a sequence of operations demands familiarity with, in this case, the IBM Job Control Language. A thorough introduction to JCL, including readings in the technical manuals, is given. Students are assigned several exercises involving tape and disk data sets.

It is at this level that the student is introduced to some of the system utilities. Because of the extensive use of sorting techniques, consideration and exercises involving both the JCL invoked SORT as well as a PL/1 link-edited sort are given. Other utilities, including IEBGENER, are discussed.

Additional PL/1 concepts are introduced in conjunction with the JCL exercises. Prominent among these are record I/O features, regional data sets, and the PL/1 list-processing features--items omitted or discussed only briefly in the introductory course. Some attention is also given to programming efficiency including timing tests to determine the relative efficiency of various blocks of code that perform the same operation. As before, the student is required to report his outside readings in applications and to present a term project of his own design.

Completion of the second course marks the end of the first phase of the program. By this time the student has progressed far beyond the level of programming competence achieved in a single introductory course. He has learned JCL and is familiar with a fair number of system utilities. He has seen the way programs of his own design and utility programs can be fitted together to produce a rather sophisticated data handling system. By considering problems systematically, he can see how seemingly inconsequential details at one point determine later capabilities. Having completed this sequence of courses, the student has the technical background to apply the computer to problems of his own design and interests.

Third Course

The third course marks a second phase of the program--an opportunity for the student to apply the techniques and information he has gained to an extended project of his own design. The projects done in this seminar are suitable for undergraduate honors projects or, at the graduate level, trial runs for masters and doctoral theses. The student is asked to follow a prescribed method and report on his work at two different stages in progress as well as make a final presentation.

The method prescribed is based upon the assumption that the humanist must use the computer to explore the ideas and problems that interest him and his colleagues, not those problems that may be raised simply because the computer can solve them. Studies that result in frequency counts or various ratios in the absence of a supporting context or rationale have largely been ignored or dismissed in the humanities. Consequently, the first step is to define and justify a thesis within the traditional terms and methods of the student's discipline. At this level, the thesis argument might read like that found in a conventional term paper or dissertation, quite likely omitting any reference to the computer at all. Next the definitions and methods of the thesis must be translated into operative terms and computational procedures. For example, an image might be defined on the first level as any word having sensory or thematic value, but when translated into computational terms, it becomes a partitioning of the vocabulary into those words selected as images and those not. Often the mapping onto the operative level is not one-to-one; for example, I have seen studies that consider the associative relations among textual elements using a number of analytic programs and aids, including frequency counts by text section, concordances, and principal component analysis.

Having defined the thesis in conventional terms and translated it into operative terms and procedures, the student then performs his analysis. When all procedures have been run, the student must interpret his results, but on the operative level. That is, if using factor analysis, he must determine the significant factors, their loadings, and their adequacy. If using Fourier analysis for a thematic study, the student must determine the significant frequencies, produce cumulative plots, and then seek inter-relation among the various thematic distributions. When he has analyzed all of his data in terms of the models and procedures in which they are defined, he is ready to move to the final step.

The last step involves mapping his results back out to the level from which he started. The student must now present an argument supporting and demonstrating his thesis that is once more understandable to his non-computational colleagues. To be sure, computer produced materials and data will be used in support of the argument, but the main presentation must be cast in terms as close to the traditional ones as possible.

Students report on their projects at three different stages. All report on the initial thesis statement and all make final presentations. At some intermediate stage, students report on the mapping process, the operative procedures being used, and anticipated directions. It is my experience that projects executed in this way display a rigor of thought and argument that is uncommon in much student work, particularly in the humanities. Of the students enrolled in my seminar on Computational Stylistics, at least half of their projects should result in publishable work.

This last point raises a question implicit in the design of this entire course--the value of research training at the undergraduate level. The argument usually raised against training undergraduates to do research is that many have no plans to go to graduate school. This argument, in my opinion, ignores the value involved in the training experience itself. In this course and this program students are asked to assimilate and apply a wide variety of skills and information, much of it derived from other parts of their educational experience, to an extended problem that interests them. The problem-solving techniques developed are enough to justify the endeavor, for whether the student ever uses a computer again or not, he will be much better prepared to approach any complex problem, break it into smaller problems, and to assemble a solution. Finally, as no other, this course gives the undergraduate an opportunity to integrate a large number of the different facets of his curriculum. The experience of actually using his training in literature, history, math, physics, and computer techniques on a single project of his design is often an exhilerating experience. What more can we offer any student?

Conclusion

Other colleges and universities implementing computer courses for humanists should give careful consideration to the department or college in which they are to be located. In the past, computer science departments have unquestionably contained the personnel best qualified to teach computing.

As Computer Science becomes more and more a recognized discipline, the amount of attention and prestige associated with "academic" courses as opposed to service courses has changed considerably. Today, the computer scientist who puts the necessary effort into a course for humanists, learning all of the specialized techniques and tantive applications essential to teach it properly, is likely to be penalized within his profession; he simply will not be taken seriously by his colleagues. The major argument in favor of locating such courses in many Computer Science departments seems to be territorial rather than pedagogical: regardless of interests, no department wishes to lose students or credit hours that support graduate programs. Ultimately, however, those of us deeply committed to seeing the computer become in reality the tool it promises to be for the humanities must insist that pedagogical concerns receive highest priority; otherwise the program will be stifled. As more and more humanists complete course sequences of this sort and continue to work in the field, we can look to them for teachers. They are much more likely to have the enthusiasm and dedication needed to teach other humanists than their colleagues in Computer Science. The best of two worlds, however, may be achieved through joint appointments of such individuals and crosslisted courses, thereby insuring pedagogical priorities while distributing credit hours to those most concerned about them.

This sequence of courses has been taught once under an experimental Liberal Arts rubric, and the first two courses will be repeated during the '73 winter and spring terms. We are currently negotiating permanent listings. The program has been closely evaluated and is, we believe, a complete success. The students emerge thoroughly grounded in the programming language as well as the system resources and control language; they are able to approach a complex problem, break it into modules, and anticipate the power and limitations imposed by various decisions. Finally, they have been taken step-by-step through an extended research project. To the best of my knowledge, this program because of its integrated, sequential organization is the most thorough in the country. Obviously it will change, but the rationale behind it and the experience gained in teaching it may be useful to others in establishing their own programs to fit the needs of their students.