INTRODUCING PRACTICAL EXPERIENCE INTO CURRICULUM 68

THROUGH INTEGRATION OF COURSES

Larry D. Menninga
Department of Mathematics and
Computer Science
Western Washington State College
Bellingham, Washington 98225

Abstract

A course which will give students some practical experience with large programming problems is described, The course combines the material from five separate courses from Curriculum 68. The material is presented in conjunction with a major programming project which is the unifying ingredient of the course. The project provides the student with immediate and realistic applications of the ideas and topics presented in the course.

Introduction

During the past year or two there has been an on-going discussion of the relative merits of theory versus applications in computer science education. Various points of view have been presented by Kandel [1], Wegner [2], and others [3,4]. This paper is not intended to be a contribution to that issue, since the proposal here is for a course at the undergraduate level--a level at which theory is not pursued to great depths.

We do not look at a B.A. in computer science as training for a specific job but, rather, we expect it to provide an understanding of computing and computing systems, and we expect the student to gain some facility in using basic programming techniques. It seems to be widely felt that the graduates we are turning out are not suited to the facts of life in the world of industry and commerce.

At the same time, the computing industry is having problems with the production of large programs [5]. These large programming efforts have been not only costly in terms of the time and resources used to produce working systems, but the resulting programs are generally inefficient and full of errors.

Although our graduates may have learned programming techniques which are more efficient and less error-prone than the methods used previously to develop software and although they may have been exposed to structured programming, hierarchical program development, meaningful documentation and other "good" programming practices, they do not live up to our high expectations when they get out on the job.

A major factor in this situation is the lack of experience most B.A. graduates receive as part of their formal education. While it is true that they learn certain techniques--sorting methods and hash coding--they seldom, if ever, have the opportunity to apply these techniques to "real-world" problems. Because their programming experience consists mainly of short isolated routines, written to learn and understand basic algorithms, they do not encounter large-scale programming problems and the difficulties unique to such large systems.

Including Practical Experience in the Curriculum

At Western Washington State College we have used the ACM Curriculum 68 [6] as a guideline for developing an undergraduate computer science program. It is felt that the experience we desire our graduates to have can be provided without completely rejecting our present curriculum. This can be done by choosing a large scale programming project and then combining several courses in which material applicable to the problem is taught. Much of the material from each of the courses would still be presented, but it would be applied directly to the problem at hand. A proposal for such a course will be presented here.

General Description of CPS

The course to be described will be called Computer Programming Systems (CPS). This course will integrate the material from five of the courses proposed in Curriculum 68. They are I1: Data

Structures, I2: Programming Languages, I3: Computer Organization, I4: Compiler Construction, and I5: Systems Programming. CPS would run for a full school year--three quarters at Western. The students taking the course would be in their senior year and would have had at least two quarters of programming in a high level language and two quarters of assembly language programming. Some of the course material would be presented in lectures, other material in a workshop setting, and in addition the students would spend a good deal of time using the computer laboratory.

The unifying factor would be the programming project used in the course. There are several suitable candidates. We propose that the initial project be the development of a paging system to provide virtual memory capability to a minicomputer. (A successor to this could be the development of a time-sharing system.)

Course Organization

The order in which the material is covered will be fairly critical in this course, since the student must have time to write programs applying the concepts he has learned. The ideal arrangement will deal with topics when they are needed for the project. As difficulties are encountered and decisions are to be made during the implementation of a particular aspect of the project, the literature can be consulted and known methods of handling the situation can be presented. This approach will not only provide a motivation for the material, but it will also teach the students to consult the available literature for methods which have been tried before.

CPS will begin with lecturing and workshop sessions on computer organization. Boolean algebra and combinational logic will be presented in lectures. The minicomputer will be used to study I/O facilities, instruction design, storage addressing, and other machine features. Topics such as different architectures will be deferred until late in the course.

In conjunction with the machine organization, some of the basic concepts of data representation can be presented. The properties of peripheral devices such as discs and tapes and their affect on data organization could also be included at this time. These are topics which would otherwise be in the data structures course.

The next area of study would concern the algorithmic languages available for use. This would include a study of their structure and, for those languages on the minicomputer, their run-time representation. Methods used to implement storage management and procedure linkage would be examined. This will involve more of the topics from the data structures course, such as dynamic storage allocation methods, as well as material from the programming languages course.

The first topics from systems programming would be the concepts of relocation, paging, and virtual memory. Then file system organization and management would be covered. In this connection, more of the topics from data structures would be taken up. These would include the use of directories, linked structures, searching, and tree structures.

At this point in CPS enough background material has been presented so that the paging system can be designed. Several hours of workshop will be devoted to specifying the behavior of the system and laying out a general design for it. This should be done with specific reference to earlier material, such as run-time representation of programs and data management in high-level languages, and I/O facilities and organization.

Experience with compiler construction and familiarity with the more formal programming language concepts of syntax specification, precedence relations, etc. can be introduced into the course in either of two ways. The first possibility is the use of a language for systems programming. This could be a subset or modification of an existing language, such as FORTRAN, or a language designed specifically for systems programming, such as BLISS. In either case, a compiler for the language would be written, or modified and compiler techniques and formal specifications would be presented in that context.

The second alternative is to have the students write, or modify, a compiler for an algorithmic language (such as ALGOL) to produce code for the virtual memory system. It may be necessary to implement only a subset of the language in order to complete a production compiler.

Once the major programming effort is under way, less time will be spent lecturing and more time will be devoted to work sessions. Material not directly applicable to the course project, such as sorting techniques, string manipulation languages, and other topics otherwise presented in the five individual courses can be presented in the remaining lecture periods.

Required Facilities

First of all, access to a computer is

absolutely necessary. This should be a minicomputer simple enough in its structure to be understood in a short time. It should be a facility allowing "hands on" experience. An operating system of some kind should be available on the machine. Further, for the project chosen here, the computing system must also have a random access secondary storage device, such as a disc. The class will require dedicated use of the machine for a few hours a week throughout much of the school year.

Class sessions would be held daily for two hours. At the beginning of the course this time would be mainly devoted to lecturing. As the development of the project advances, more of this time will be spent in workshop sessions in which the class discusses the project as a group, or smaller teams of students could coordinate their efforts. The students would spend a good deal of additional time writing programs.

Instructor's Role

The instructor has to serve in three different capacities while teaching CPS. He is first of all a lecturer. He is also a project manager and a programmer, although these two roles may be mainly advisory. (The students must be able to learn from him by example.) The class could be organized in a manner similar to the Chief Programmer Team of Baker [7]. At any rate, the programming system should be developed in a top-down fashion. Besides all of the other advantages, this approach will allow the system to be broken up into subsystems which can be assigned to individual students or small teams of students. By using a top-down approach they should be able to test out their subsystems and integrate them into the larger system without relying heavily upon the work of each other.

The instructor must provide examples of good programming practice in any programming that he may do. He must teach the use of structured programming to those students not familiar with that approach. He must provide guidance, and possibly demand contributions, in the ongoing documentation of the paging system.

Conclusion

A course such as this not only gives the student the opportunity to work with a large-scale programming problem, but as an added bonus this approach provides him with an immediate application of the ideas presented in the class lectures. The proposed course appears to be a viable method of adding practical experience to Curriculum 68. A less ambitious applied systems programming course, offered at the graduate level at the University of Michigan, has been successful [8].

There are some disadvantages to a course such as this. CPS requires a heavy commitment from the students; a ten-hour class lasting for an entire academic year may frighten some students. The instructor has a heavy demand placed upon him also, although if the course were taught by a team, rather than by a single individual, that load would be eased a great deal. The computing resources required for the course are also extensive.

The course, Computer Programming Systems, presented here does not provide a broad range of experience, but students can be given experience in other areas, such as business systems programming, by a similar combination of appropriate courses. Most of the problems in computer science education which were listed by Aiken [9] are addressed by a course such as CPS.

References

1. Kandel, A. Computer science--a vicious circle. Comm. ACM 15, 6 (June 1972), 470-471.

2. Wegner, P. A view of computer science education. American Mathematical Monthly 79, 2 (Feb. 1972), 168-179.

3. DuWors, R.J., and Solian, S.W. The arrogant programmer: Dijkstra and Wegner considered harmful. SIGCSE Bulletin 4, 4 (Dec. 1972, 19-21.

4. Blount, S.E., and Fein, L. The practical aspect of computer science education--discussion. Comm. ACM 16, 1 (Jan. 1973), 45-46.

5. Boehm, B.W. Software and its impact: a quantitative assessment. Datamation 19, 5 (May 1973), 48-59.

6. ACM Curriculum Committee on Computer Science. Curriculum 68: recommendations for academic programs in computer science. Comm. ACM 11, 3 (March 1968), 151-197.

7. Baker, F.T. Chief programmer team management of production programming. IBM Systems Journal 11, 1 (1972), 56-73.

8. Arden, B.W., Flanigan, L.K., and Galler, B.A. An advanced system programming course. Proc. IFIP Congress 71, North-Holland, Amsterdam, 1972, Booklet TA-7, 115-119.

9. Aiken, R.M. Summary of comments following SIGCSE panel discussion on "Computer Science Graduates--An Industry/University Gap." SIGCSE Bulletin 4, 3 (Oct. 1972), 37.