CS39

A UNIVERSITY'S EDUCATIONAL PROGRAM IN COMPUTER SCIENCE

BY

GEORGE E. FORSYTHE

TECHNICAL REPORT NO. CS39
MAY 18, 1966

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

A UNIVERSITY'S EDUCATIONAL PROGRAM IN COMPUTER SCIENCE

by

George E. Forsythe*

Abstract

After a review of the power of contemporary computers, computer science is defined in several ways. The objectives of computer science education are stated, and it is asserted that in a U. S. university these will be achieved only through a computer science department. The program at Stanford University is reviewed as an example. The appendix includes syllabi of Ph. D. qualifying examinations for Stanford's Computer Science Department.

1. The power of computers

With enough money, one could obtain delivery in 1967 of automatic digital computers effectively capable of making decisions in less than 150 nanoseconds and multiplying in less than one microsecond. Such machines can retrieve each of a quarter million words in less than 100 nanoseconds, and each of some 3×10^9 characters in something like 0.5 second. Since a 500-page book holds approximately 1.5×10^6 characters, the above large store holds the equivalent of 2000 monographs--a substantial little library.

Until the first automatic digital computers were available for delivery approximately 15 years ago, we were limited to making decisions, multiplying, and accessing a fast store in something like 10 seconds, while access to each character of a 2000-book library could hardly take

*This is a revision of previous Stanford report CS 26.

The author acknowledges innumerable conversation with Richard Hamming, William Miller, and many other computer scientists.

less than 100 seconds. Thus the past 15 years has seen accelerations of 200 (access to slow store), 10^7 (multiplications), and even 10^8 (access to fast store).

In comparison, the speed-up in travel between walking and going by jet is about 10^2, that in the dissemination of information between manuscript and a large newspaper machine is about 10^6, and that in communication speed between using sound and radio waves is also about 10^6. These changes in rates of transportation and communication have completely remade the world of earlier times. It is evident that the even greater speed-ups in information processing are remaking our world. It is furthermore clear that it will be many years before our capacity to exploit the new computers will catch up with even their present capabilities.

By its speed of processing information and making decisions, the electronic digital computer is an extension of the human mind with capacities incomparably greater than those of any other mental prosthesis heretofore available to man. In describing the computer as a mental prosthesis (a term used by Gerard [4]), I am assuming that it is being used in intimate conjunction with a human mind. While there is a great debate as to how far an unassisted computer can simulate human cognitive processes, it is a priori evident that a computer linked with a human mind is a more powerful tool than an unassisted human mind. One task of computer science is to demonstrate in increasingly broad areas that the computer-human team is vastly more powerful than a human alone.

The exponential growth in numbers of human beings and data about them, and the increasing pace and complexity of our technological civilization create problems of information processing for which the electronic computer holds the only visible means of solution. Examples: air-traffic control over metropolitan areas, air-defense problems, space-vehicle navigation, information retrieval problems, national tax records. In such areas computing can directly simulate applications for which mathematical techniques are as yet unavailable. In other areas, like weather forecasting, the computer is the agent for carrying out mathematical techniques of great complexity.

2. What is comouter science?

I consider computer science in general to be the art and science of representing and processing information and, in particular, processing information with the logical engines called automatic digital computers. Computer science deals with such related problems as designing automatic digital computers and systems, the design and description of suitable languages for representing both processors and algorithms, the design and analysis of methods of representing information by abstract symbols, and of complex processes for manipulating these symbols. Thus the central theme of computer science is like the central theme of engineering--namely, the design of complex systems to optimize the value of resources. Perhaps the main difference is that computer scientists work with a very abstract medium (information), and design systems typically far more complex in detail than most elaborate engineering systems.

Computer science must also concern itself with such theoretical subjects supporting its technology as coding and information theory, the logic of the finitely constructable, numerical mathematical analysis, control theory, switching theory, automata theory, mathematical linguistics, graph theory, and the psychology of problem solving. Naturally these theoretical subjects are shared by computer science with such disciplines as philosophy, mathematics, engineering, operations research, and psychology.

To a modern mathematician, design seems to be a second-rate intellectual activity. But in the most mathematical of the sciences, physics, the role of design is highly appreciated, and Nobel prizes are awarded more or less equally to theoretical and experimental physicists. It is said that Donald Glazer was awarded a Nobel prize solely for the design of a bubble chamber, since this opened up new fields of physics.

The ultimate purpose of physics is the intellectual one of understanding the physical world, and the role of experiment is the secondary one of assisting in the understanding. In computer science, on the other hand, the primary purpose is to design optimal processes and processors of information, and theory serves the secondary role of suggesting techniques, methods of analysis, and proofs of optimality. We conclude that, if experimental work can win half the laurels in physics, then good experimental work in computer science must be rated very high indeed.

Because the representation and processing of information are the core of computer science, many persons refer to our subject as the science of information processing, and indeed the International Federation for Information Processing bears such a name. Until a rather substantial discipline is built up, I prefer to keep explicit our pragmatic goal by retaining the word "computer" in our name. Perhaps the name "computer and information sciences" is a proper compromise at present.

A very concise but general definition of our subject is given in [2], page 175: "The information sciences deal with the body of knowledge that relates to the structure, origination, transmission, and transformation of information--in both naturally existing and artificial systems. This includes the investigation of information representation, as in the genetic code or in codes for efficient message transmission, and the study of information--processing devices and techniques, such as computers and their programming systems."

An interesting comparison of computer science with both pure and applied mathematics has been made by Gorn [5]. While Gorn is thinking only of the scientific applications of computers, the distinctions presented could certainly be translated to cover other areas of computing. To Gorn, the pure mathematician is interested in the syntactical relations among symbols, quite apart from their meaning in the physical world or their computability. Thus the all-important questions of pure mathematics deal with the structures of theories, and not with their meaning or their use. For example, most pure mathematicians are unconcerned with what numbers actually are, and indeed the question is unsettled; they are instead concerned (among other things) with what relations exist among numbers and among objects built up from numbers.

The applied mathematician is primarily concerned with the semantics of symbols--what do mathematical theorems mean as applied to the physical world? For example, how can mathematical analysis help us understand economics or electronics?

Finally, the computer scientist is concerned with the pragmatics of the applications of mathematics to problems. What algorithms can actually be used to calculate the roots of an equation? What does it cost in storage or time or human effort to perform a certain algorithm? What guaranteed or probabilistic or merely plausible error bounds can be constructed for the answers? What languages can the solution be described in? What hardware do they require? Are the languages well suited to the current mode of interaction of the programmer with the computer? How can foolproof algorithms be found? (This is not a trivial question. Hardly any one knows how to solve even a quadratic equation on a computer without unnecessarily risking loss of precision or overflow or underflow!)

One thing computer science is not: it is not merely the union of the applications of a computer to diverse problems. Rather, the core of the field is application-independent and rather abstract, being concerned with languages and techniques that are relevant to a variety of different applications of computing, in much the same way that mathematics is an abstract tool that is relevant in many different applications. However, because of the newness of the field, computer scientists must now be concerned with applications of computer methods in many different areas of technology and learning, in order to gain the insights which will later enter the core of computer science. And so computer scientists engage in a very broad spectrum of activity, ranging, for example, from pure logic to communications engineering, from the psychology of learning to business administration, from computer analysis of the content of documents to medical data processing, and from pure mathematical analysis to methods of plausible inference about the accuracy of experimental algorithms. Thus computer science is in part a young deductive science, in part a young experimental science, and in part a new field of engineering design. Because of this broad sweep of activity, computer science is usually misunderstood by those with an acquaintance, however close, with only one aspect of computing.

3. The objectives of computer science education.

There are at least three different groups of students to whom computer science education should be directed, and the objectives are different for each.

a. Nontechnical students

Almost all citizens of the developed nations will be greatly affected by computers. Hence a general education should include enough background for the citizen to comprehend something about computer science. Since a university educates the future leaders of the community, the students need a background for making decisions in a computerized world. They need to learn what computers are, what they can do, what they cannot do.

Students frequently have a mystical awe of computers as robots with giant brains which will overwhelm the world. Students need to realize that computers are merely obedient slaves with unending capacity to make decisions. Students need to realize the large role of human beings in creating computer systems, and that every bit of automation is achieved by human planning in the large and in detail. Students need also to come to grips with the question, "Can machines think?"

All these things are reasonable objectives of a one-semester lecture-and-laboratory course in which students actually program in an appropriate language, and we have considerable experience with such courses at Stanford.

b. Specialists in other technical fields

The automatic computer is one of the most important tools to have been devised in the history of man. Indeed, Gerard [4] has classed it after the invention of speech and systematic science as the third giant step in man's intellectual history. It is already recognized by many kinds of natural scientists that they must know computing fairly well. Current studies of engineering education emphasize acquiring general purpose tools with long expected lifetimes of utility--like mathematics, English, statistics--at the expense of special information available in handbooks. Automatic computing is indeed becoming recognized as one of these lasting tools, because of the wide domain of its applications.

Zadeh [6], chairman of the Department of Electrical Engineering, University of California, Berkeley, makes a particularly strong claim about the importance of computer science in technical education: he states that electrical engineering will suffer a serious decline in importance as a discipline unless it can absorb a substantial amount of activity in computer science.

It is now clear that students of social science must also acquire a familiarity with computing methods. And the serious student of the humanities will soon find computers indispensable, if he is to carry out research on any substantial volume of data. Moreover, the computer production of music has opened a brand new field of creative activity.

It should be understood that one of the reasons automatic computing is such a valuable tool is the abstractness of the information represented in a computer. It is possible to represent not only numbers, but also letters, punctuation, cars on a freeway, musical notes, particles of a fluid, chess pieces, or practically anything else that is discretizable. Moreover, we are not limited to conventional operations on these symbols, for a computer is able to carry out arbitrarily defined operations. Thus nonlinear mathematical models, three-valued logics, stochastically defined processes, and many other matters difficult for humans to implement can be modeled or simulated with an automatic computer with no intrinsic difficulty. Even when operations are extraordinarily complex, frequently the microsecond speed of a computer will enable a reasonably large process to be completed within tolerable times. Thus the computer not only permits heretofore unmanageable mathematical processes to be carried out for the first time, but it also sometimes permits the experimenter to be freed from any mathematical model whatever! That is, the computer model can sometimes simulate the physical situation directly, without the intermediary step of mathematization of the situation. This enormous potential of computer models is mainly waiting to be exploited.

c. In education of computer science specialists

The most important group to be educated in computer science are the future specialists in the field, for they are the seed who will become the creators and the teachers of the future. They must receive enough background to be able to follow and preferably lead the future development of the subject.

For the graduate student of computer science the faculty must create a curriculum and inculcate standards of performance in it. The student must learn to read and to write the appropriate literature. A background must be built for years to come. Computer science education, like all education, must aim to light a lamp for the student, rather than try to fill the student's bucket of knowledge. This education must create the field's leaders, full of ambition and fire to attack the all-pervasive unsolved problems of computer science. Even the average student must realize his dependence on continuing education, that he will have to learn and relearn almost everything he knows every few years of this revolutionary epoch in technology.

4. How can a U. S. university realize the above objectives?

The first major step by which a U. S. university can realize the objectives stated above is to create a department of computer science, or the same thing under a different name. The reason for forming a department is to enable computer scientists to acquire a faculty of their own choosing, and exercise control over the curriculum of students wishing to specialize in computing. There is abundant experience to show that without such an administrative step computer education will not even keep up with the field. Without a department a university may well acquire a number of computer scientists, but they will be scattered and relatively ineffective in dealing with computer science as a whole. For example, numerical analysis located solely within a mathematics department may be too much concerned with analysis, and too little with computing. Without actually studying the solution of problems on computers no analyst is likely to learn what are the actually pressing problems of scientific computing. And without this knowledge one can hardly serve as a computer scientist. (At the same time, we stress that computer scientists owe a great debt to all mathematicians, past and present, who do research in numerical or constructive mathematics.)

Naturally a computer science department cannot be started until there are two or three computer scientists on the faculty. One may therefore expect that the department-to-be will be in a probationary status for two or three years, as Stanford's was.

~ Computer science departments are known to the author at the following universities, and certainly the list is incomplete: Carnegie Tech, Cornell, Illinois, Michigan, North Carolina, Notre Dame, Pennsylvania, Penn State, Purdue, Stanford, Texas, Toronto, Wisconsin. Stanford's was the first or nearly first of these.

There are strong computer science programs without departments at the following universities (and others): Brown, California, Cal Tech, Chicago, Harvard, M. I. T., Maryland, New York University, Princeton, Virginia. Several of these programs will become departments in the near future.

Probably a department of computer science belongs in the school of letters and sciences, because of its close ties with departments of mathematics, philosophy, and psychology. But its relations with engineering departments concerned with systems analysis and computer hardware technology should be close.

In years to come we may expect a department of computer science to come together with departments of pure mathematics, operations research, statistics, applied mathematics, and so on, inside a school of mathematical sciences. We can hope for some weakening of the autonomy of individual departments, and a concomitant strenghtening of the ability of a university to found and carry out interdisciplinary programs. The Division of Mathematical Sciences at Purdue appears to be a leader in this respect.

A department of computer science has the responsibility of creating curricula for each of the groups mentioned in section 3 above. Thus there will be degree program for its own majors. There will be service courses for majors in other departments in which computing is recognized to be central. There will be "computer appreciation" courses or more substantial courses as part of a general liberal education program. In a university with an adult education program some or all of these will be offered at night for the "retreading" of graduates of past years.

In all these courses the computer science department must strive to extract and teach the essence of computer science, and avoid its transient and accidental aspects. For example, general introductions to computing need no longer spend much time comparing base 2 and base 10 arithmetic, although students of numerical mathematics must be perfectly aware of the arithmetic pecularities of any number system in use. And the detailed study of a machine language seems better relegated to specialized computer science courses than taught in a first course. Whereas at one time performing arithmetic on machine-language commands seemed to be the essence of programming, it is no longer so important. In fact, where several programs use the same systems one cannot permit the letter to be modified at all. Finally, the department should insist that its majors devote substantial time to studies in other departments (mathematics, logic, psychology, electronics, etc.). For having control of the computer science curriculum does not imply actually teaching the entire curriculum!

In addition to a computer science department, a university needs a strong computation center in order to attain its educational goals in computer science. Such a center must be organized with machines, languages, and software capable of receiving a large number of programs from different classes of students. In a batch-processing computation center we expect that each student of introductory programming should have one short pass per calendar day on a computer. More advanced students will come less often, but with longer runs. Thus the center must be well endowed to receive a very large volume of student programs in addition to its traditional and better rewarded goal of serving faculty research work. Unless you have had experience with this job-shop type of computing, you cannot recognize how poorly adapted typical machine systems are to dealing with it. Even with good machine systems it is a large problem in logistics and human relations to run more than 1000 jobs a day for a variety of users.

As we look ahead, it is commonly believed that university computation services will generally take the form of a central processor, time-shared as a university utility from many consoles around the campus. When actually working well, such systems should cure many of the disadvantages of batch-processing systems applied to student work. We may expect a considerable shake-down period before the systems work well, and we may also find some difficulties as yet not dreamt of. The optimization of such time-shared services should provide many research problems for students of computer science and operations research.

5. Stanford University's educational program in computer science.

Since January 1965 Stanford University has had a Computer Science Department within its School of Humanities and Sciences. The new department had been gestating for approximately three years within the Mathematics Department. In September 1966 the Computer Science Department will have positions for approximately twelve faculty members at levels of assistant professor and above. The principal fields of interest of the faculty will be roughly as follows: numerical analysis (3), artificial intelligence (3), programming languages and systems (3), machine organization (1), computer control of physical data (1), logic and linguistics (1).

One measure of the diversity of computer science is the variety of backgrounds of its practitioners. In 1965-66 our faculty have Ph. D.'s as follows: mathematics (4), applied science (1), electrical engineering (2), industrial administration (1), physics (2).

We have no undergraduate degree program in computer science, and no present intention of starting one. At the present time we would regard a B. S. in Computer Science as too likely to be a terminal degree, although it might be excellent preparation for graduate work in fields like business administration or medicine. Our recommendation for Stanford undergraduates interested in computing is that they major in mathematics and take our senior-graduate courses. In our opinion the undergraduate computer science curriculum given in [1] is nearly a master's curriculum.

We have approximately 115 graduate majors, of whom approximately 80 are full time students. We offer a Master of Science in Computer Science, and by June 1966 will have awarded approximately 67 of them since 1961. We offer a Ph. D. in Computer Science. The first two will probably be awarded in 1966, and approximately ten more students have passed the written qualifying examinations for the Ph. D. Two students have taken interdepartmental Ph. D. degrees involving substantial study in computer science. Over the past eight years the author has been the principal thesis advisor of some seven students who have written dissertations in numerical analysis for their Ph. D. in mathematics.

The Bulletin [7] gives our catalog description of the courses to be offered in 1966-67 in our department, together with an indication of the requirements for the M. S. and Ph. D. degrees. (A one-quarter 3-unit course normally consists of 30 lectures.) In the appendix are given syllabi for our qualifying examination, which includes three papers:

(1) numerical analysis and computational mathematics;

(2) computer and programming systems;

(3) advanced nonnumeric applications, artificial intelligence, and the mathematical theory of computation.

Experience has already taught us that many of our graduate students are not interested in all three of the syllabus subjects above. We first saw the need for this flexibility in students who were strongly interested in numerical analysis, but who lacked the inclination to learn either the subjects of our examination (3) or the extensive real and complex analysis required for a Ph. D. in mathematics. As a result, we now permit the substitution of equivalent work from another department for either examination (1) or (3). We regard examination (2) as so central to our subject as to remain compulsory. We have found that several of our students most interested in numerical analysis are happy to write examinations (1) and (2), and also one of the qualifying examinations for the mathematics Ph. D. (real analysis, complex analysis, algebra, or applied analysis). On the other hand, some students most interested in nonnumeric computation are happy to write examinations (2) and (3), and also take an examination in logic or perhaps psychology. We expect other substitutions to be proposed later.

Since computer science is young and in a formative state, and since our examinations cover such a broad sweep, we feel that the flexibility described above is a wise one.

Our main service courses are directed to students who have already had calculus. Our one-quarter introduction to computing (course 136) ranks as one of the most popular elective courses at Stanford. We have two quarters of introductory numerical analysis (courses 137, 138) presupposing course 136, linear algebra, and ordinary differential equations. We have a special form of course 136 (courses 5, 6) directed to younger engineering undergraduates.

We have instituted one course (126) for general students, presupposing only high-school mathematics and directed to students of humanities and the social sciences.

Practically all our courses involve substantial amounts of computer use by the students as individuals. They are effectively laboratory courses.

As time goes on, we anticipate the creation of more sections of course 136, directed to majors in special areas and presupposing different backgrounds and interests.

Several other departments at Stanford teach computing in one form or another. Most of these are applications courses presupposing our course 136.

Our graduate offerings include a two-quarter sequence on programming systems; a one-quarter course on the logical structure of computers; a three-quarter sequence on numerical analysis; a two-quarter sequence on elementary artificial intelligence; a three-quarter sequence covering list-processing, logic of computing, and advanced artificial intelligence; a special numerical analysis course; a one-quarter course on control programming; a programming laboratory course; a one-quarter course next spring on mathematical linguistics; a one-quarter course on graphic data processing; and some courses in mathematical programming.

As we teach computing to specialists of other fields, it is essential that their own faculty be able to follow up with important applications. In the transition period, it was desirable to teach the faculty directly. Earlier in the decade the Computation Center held three special one-week sessions, two for engineering faculty and one for biomedical faculty. These courses take very careful planning, but pay off very well in knowledge and good will.

The Stanford Computation Center is well organized to receive large numbers of student jobs, mainly in Extended Algol for the Burroughs B5500. However, the total cost for handling student jobs in computer science courses has now reached $84,000 per year, and it is essential to receive special funding for this work.

The Computer Science Department is not responsible for the Computation Center's service work, but it does provide leadership for system selection and organization. However, the Computer Science Department will itself soon have some large computer systems acquired in connection with research contracts.

6. The research program of Stanford's department.

A university department has the twin roles of education and research. Though this report has concentrated on education, I should like to summarize the research areas that we happen to be active in. It is, of course, no coincidence that these include the three areas of our qualifying examination listed above. We also have competence in theorem proving by computer, mathematical linguistics, and computer recognition and control of external events. During spring quarter 1966 we have five seminars going on--in machine design, human values in a technological society, mathematical theory of computation, programming languages and systems, and graphic data processing. In other quarters there have been seminars in such subjects as numerical analysis, pattern recognition, artificial intelligence, time-sharing, etc.

The following research projects are or have recently been active in the department:

Numerical analysis: algorithms for computational problems of linear algebra, with error bounds; solving linear partial differential equations by difference methods, and by expansions in particular solutions; solving nonlinear systems; libraries of algorithms; studying iteration functions.

Computer control: control of large linear accelerator; on-line control of experiments; automation of data reduction from experiments; portable computing.

Automaton: Can a general-purpose digital computer (PDP 6) use a TV tube image input to control the action of a mechanical hand to carry out useful work?

Time-sharing: How can one best design a time-sharing system based on cathode-ray-tube displays instead of typewriters?

Artificial intelligence: speech recognition, picture recognition; classification of organic molecules; simulation of human learning.

Programming languages: representation of the semantics of a programming language; design of appropriate successors to ALGOL 60; invention of appropriate languages for persons designing algorithms in a conversational mode at a console; design of operating systems; languages for compilers of computers.

Hardware systems: modular arithmetic.

Computational linguistics : computer procedures for manipulating transformational grammars; the parsing problem for transformational grammars.

Theorem proving: proof procedures and decision procedures in the first order predicate calculus and their implementation on computers.

Psychiatry: simulation of psychiatric treatment of mentally disturbed, with patient at a teletype console typing natural language.

7. Miscellaneous remarks.

It would be desirable to have criteria of Stanford's progress towards our goals in computer science education. So far we are too much involved in getting organized, recruiting, teaching, and so on, to have done much evaluation. We do know that Stanford students appreciate our program very much. Courses 5, 126, and 136 have a campus-wide reputation for being time-consuming, because it takes beginners many hours to think out algorithms, and get them correctly keypunched and run. Nevertheless, the enrollments are very high. With both formal and informal programming courses, we estimate that we reach about 1200 persons a year. If these people average but two years at Stanford, then about 2400 persons know computing in an active way. With some 10,000 students and 1000 faculty and teaching staff, that means that over 20 per cent of the Stanford academic family are well acquainted with computing.

In the winter quarter of 1966, approximately 1400 students in 66 courses used the Computation Center to solve homework problems.

We therefore feel that computing is established at Stanford, and that it is time to turn our attention towards raising the quality of our work. For example, we need not only to teach students to program--we need to teach them to program well. We must give many more of the science and engineering students who use the computer an introduction to good algorithms and a fear of bad algorithms. More important, we need to bring the advantages of symbol manipulation to our students. Even for science, it is probable that the ability of a computer to automate algebraic manipulation will in the end be more important than the ability to automate arithmetic computation. For historical reasons and because of the lack of easily used list-processing languages, the emphasis till now has been mainly on arithmetic computation alone.

We have wondered about the place of numerical analysis in a university with a department of mathematics and another in computer science. Apparently there will be two kinds of numerical analysis Ph. D. degrees. One will be a mathematics degree, following the regular required mathematics courses, with some elective courses in computer science and a thesis in numerical analysis directed by some one in the computer science faculty. This would be appropriate, for example, for a student whose thesis developed asymptotic error bounds for finite difference methods for solving partial differential equations. There might or might not be experimental computations. The other will be a computer science degree, with a number of extra mathematics courses and a thesis in numerical analysis. Such a thesis might emphasize the algorithmic aspects of numerical analysis, and would certainly involve experimental computations on a digital computer. It might, for example, explore the kinds of languages appropriate to solving difficult mathematical problems with computer systems involving intricate man-machine interactions. (Cf. Culler and Fried [3].) Or it might involve creating algorithms which are truly complete and foolproof in regard to error bounds, over- and underflow problems, and so on. In short, such a thesis would face up to the real difficulties of using limited-precision arithmetic to solve mathematical problems. It is often more difficult to decide when an iterative process should be terminated on a computer than it is to prove its convergence, and these theses should help solve such problems.

References

[1] ACM Curriculum Committee on Computer Science, AN UNDERGRADUATE PROGRAM IN COMPUTER SCIENCE--PRELIMINARY RECOMMENDATIONS, Comm. ACM 8 (1965), pp. 543-552.

[2] University of Chicago, GRADUATE PROGRAMS IN THE DIVISIONS, ANNOUNCEMENTS 1966-1967, pp. 175-177, describing Committee on Information Sciences.

[3] G. J. Culler and B. D. Fried, THE TRW TWO-STATION, ON-LINE SCIENTIFIC COMPUTER: GENERAL DESCRIPTION, pp.65-87 of Margo A. Sass and William D. Wilkinson (editors), COMPUTER AUGMENTATION OF HUMAN REASONING, Spartan Books, 1965.

[4] Ralph W. Gerard, COMPUTERS AND THE FUTURE OF EDUCATION, talk to Fall Joint Computer Conference of American Federation of Information-Processing Societies, Las Vegas, Nevada, 2 December 1965.

[5] Saul Gorn, THE COMPUTER AND INFORMATION SCIENCES: A NEW BASIC DISCIPLINE, SIAM Review 5 (1963), pp. 150-155.

[6] L. A. Zadeh, ELECTRICAL ENGINEERING AT THE CROSS-ROADS, paper presented to the Institute for Electrical and Electronic Engineering, March 1965, Proceedings, pp. 47-50.

[7] Stanford University, COURSES AND DEGREES 1966-67, Stanford University Bulletin, May 1966 (obtainable from Registrar, Stanford University, Stanford, California 94305).

Computer Science Department
Stanford University
Stanford, California 94305
9 May 1966

STANFORD UNIVERSITY
COMPUTER SCIENCE DEPARTMENT

Revised
December 15, 1965

Syllabus for Ph.D. Examination
in Numerical Analysis and Computational Mathematics

1. A general familiarity with the following computer problems: solving one or more linear or nonlinear equations; simple problems involving ordinary or partial differential equations; approximation of data by polynomials; approximating integrals and derivatives of functions by linear formulas; locating maxima of functions; computing eigenvalues of matrices. The subjects and the supporting Mathematics or Statistics should be known at a depth like that of courses 137 and 138 and the following books: Hamming, NUMBERICAL METHODS FOR SCIENTISTS AND ENGINEERS; Henrici, ELEMENTS OF NUMERICAL ANALYSIS; Ralston, A FIRST COURSE IN NUMERICAL ANALYSIS; Stiefel, ELEMENTS OF NUMERICAL ANALYSIS. For computational methods of linear algebra, see Forsythe, NOTES ON COMPUTATIONAL METHODS OF LINEAR ALGEBRA (dittoed notes for course 138, 1965, on reserve in Computer Science Library), or Faddeev and Faddeeva, COMPUTATIONAL METHODS OF LINEAR ALGEBRA. For solving parabolic and elliptic partial differential equations, see D. Young's Chap. 11 of John Todd (editor), SURVEY OF NUMERICAL ANALYSIS, or "A survey of numerical methods for parabolic differential equations", by Jim Douglas Jr., in ADVANCES IN COMPUTERS, vol. 2 (1961).

2. A reasonable understanding of the pragmatics of scientific computation with automatic digital computers: the importance of fully automatic procedures for frequently used computations; pitfalls in using standard algorithms of mathematics on computers with (necessarily) limited precision: what constitutes a good and well documented algorithm, and where such algorithms can be found for various problems; the different types of errors in computation, and ways to estimate and (where possible) reduce the errors;

the influence of the logical design of computer hardware and software on the design of algorithms, and the accuracy and cost of computation in time, storage, and human effort.

3. A deeper knowledge of some selected area of numerical analysis and supporting mathematics, with a depth like that of courses 237a, b. Examples: discretization error and stability in solving ordinary differential equations (Henrici, DISCRETE VARIABLE METHODS IN ORDINARY DIFFERENTIAL EQUATIONS), round-off error (Wilkinson, ROUNDING ERRORS IN ALGEBRAIC PROCESSES), solution of partial differential equations (Varga, MATRIX ITERATIVE ANALYSIS, or Forsythe-Wasow, FINITE-DIFFERENCE METHODS FOR PARTIAL DIFFERENTIAL EQUATIONS), approximation (Davis, INTERPOLATION AND APPROXIMATION, or J. R. Rice, THE APPROXIMATION OF FUNCTIONS), computation methods in linear algebra (Wilkinson, THE ALGEBRAIC EIGENVALUE PROBLEM), numerical integration (Krylov, APPROXIMATE CALCULATION OF INTEGRALS); Monte Carlo methods (Hammersley and Handscomb, MONTE CARLO METHODS); combinatorial computation problems (Beckenbach, APPLIED COMBINATORIAL MATHEMATICS); solution of equations (Ostrowski, SOLUTION OF EQUATIONS AND SYSTEMS OF EQUATIONS; Traub, ITERATIVE METHODS FOR THE SOLUTION OF EQUATIONS).

4. Familiarity with the principal reference books for mathematical computation--bibliographies, tables, collections, formulas and algorithms, specialized monographs, etc.

STANFORD UNIVERSITY

COMPUTER SCIENCE DEPARTMENT

March 30, 1965

Syllabus for Ph.D. Examination in Artificial Intelligence Research, Non -numeric Applications, and Mathematical Theory of Computation

The following is a list of topics with which the student should be familiar:

1. List Processing, Symbol-manipulating languages for non-numeric applications.

    1.1 Thorough familiarity with LISP or some other list processing language.

    1.2 Some familiarity with the other languages of this type; similarities, differences, and issues in the construction of list processing languages.

    NOTE: The languages under discussion here are LISP, IPLV, SLIP, COMIT, SNOBOL, plus others you may discover in your reading.

2. Artificial Intelligence

    2.1 Heuristic Programming heavy. emphasis.

    Chess and Checker Playing Programs (various; Samuel)

    SAINT (Slagle)

    Logic Theorist (NSS)

    Geometry Theorist (Gelernter)

    Assembly Line Balancing (Tonge)

    General Problem Solver (NSS)

    Geometric Analogies (Evans)

    Graph Isomorphism Finder

    Finding Group Transformation (Amorel)

    Theorem Proving a la J. A. Robinson, H. Wong, etc.
    (is any of this, or is this not, heuristic programming?)

    STUDENT (Bobrow)

    Advice Taker Concepts (McCarthy)

    Music Composition (Hiller and Issacson)

    MH-1 Hand (Ernst)

2.2 Simulation. of Human Cognition

General Methodological Considerations Regarding Information Processing Models in Psychology

EPAM (Feigenbaum)

Binary Choice Hypothesis Former (Feldman)

"Concept" Formation (Hunt)

Investment Decisions (Clarkson)

Belief Structures, Personality, Neurotic Behavior (Abelson, Loehlin, Colby)

2.3 Learning in Problem Solvers

2.4 Learning in Random and Structured Nets, and The General State of "neural" models.

2.5 Pattern Recognition Theory and Applications

2.6 Other Adaptive Systems

3. Other Non-numeric Applications.

3.1 Question-Answering Programs (BASEBALL, SIR, Black's Program)

3.2 Information Retrieval Research, more generally construed . . . broad outlines of the current state of art.

3.3 Automatic Indexing and Abstracting

3.4 Machine Translation . . , broad outlines of current state of art.

4. Mathematical Theory of Computation.

4.1 Formalisms for Describing Computable Functions - Turing Machines, Post Canonical Systems, General Rec ursive Functions.

4.2 Universality and Undecidability

4.3 Functions Compatable in Terms of Base Functions

4 14 Formal Properties of Conditional Expressions - Proofs of Equivalence of Computations.

4.5 Recursion Induction

4.6 Spaces Representable in Terms of Base Spaces

4.7 Abstract Syntax - Conditions for Correctness of Compilers

5. Mathematical Logic

5.1 Propositional Calculus - Canonical Forms Rules of Inference, Decision Procedures.

5.2 Predicate Calculus - Axiomatization of Theories in Predicate Calculus; Axioms and Rules of Inference, Models, Completeness, Proof Procedures.

5.3 Set Theory - Zermelo-Frankel or Von Neuman Axioms.

References

For Section 1 see the IPLV Manual, the LISP 1.5 Manual, the SLIP Manual (Comm. of ACM, Sept. 1963), COMIT Manual, SNOBOL article (JACM, in 1964). Also, Bobrow's and Raphael's survey article (Comm. of ACM, in 1964).

For Section 2.1 and 2.3 the prime reference is Computers and Thought. This book will lead, you to the original sources (e.g., Slagle's SAINT was originally a Ph. D. thesis, as were many others), and also to many andvarious references as you read through the articles and scan through the Index to the Bibliography. Minsky's survey article is not to be missed! The host of Newell - Shaw - Simon papers, in RAND reports, in published literature, in the library, should be read. See Feigenbaum's short survey article in the IEEE Information Theory Transactions, plus other review articles in these Transactions. See the volumes Self -Organizing Systems (1960 and 1962). Also, the numerious JCC Proceedings and ACM Conference Proceedings, Bobrow's papers is Project Mac TR-1. Hiller and Isaacson wrote a book, Experimental Music.

For Section 2.2 prime reference again is Computers and Thought. See also Newell and Simon in the Handbook of Mathematical Psychology, chapter on Computers in Psychology. EPAM papers will' be put in the library. Hunt has a book, Concept Formation: An Information Processing Problem. See also Tompkins and Messick, Computer Simulation of Personality.

For Section 2.4 literature abounds. Minsky's "Steps" article is a good start.

For Section 2.5, ditto. See, also, Sebestyen,Decision Making Processes in Pattern Recognition. Look at Selfridge's Pandemonium and Uhr's work (cf. Computers and Thought).

For Section 2.6 - purposely vague. For example, see Ashby, Design for A Brain(second edition); Automata Studies (Shannon and McCarthy).

For Section 3.1 see Computers and Thought. Raphael's SIR is Project Mac TR-2. Black's report is in' Fran Thomson's office.

For Sections 3.2 and 3.3, scan recent issues of the NSF publication "Current Research and Development in Scientific Documentation" as a means of ascertaining state of art and diving into literature.

For Section 3.4, ditto, perhaps, for this. Look at survey article by Dick See of NSF in the magazine, "Science", sometime in Spring of 1964. Use as springboard to literature. See also, Bar-Hillel's famous article, highly critical of MT work (Advances in Computers, Vol. 1)

On pages 521-523 of Computers and Thought there is a list of collections and special proceedings and symposia that are relatively dense in articles with which you should be somewhat familiar. Try to get a look at as many as you can.

With respect to Sections 4 and 5, the following references are recommended:

Davis - Computability and Unsolvability

Suppes - Introduction to Mathematical Logic; Axiomatic Set Theory

McCarthy - A Basis for a Mathematical Theory of Computation in Computers and Formal Systems

- Towards a Mathematical Science of Computation, ICIP, 1962.

- A Formal Description of a Subset of ALGOL, AI Memo.

- Problems in the Theory of Computation , ICIP 1965, AI Memo.

STANFORD UNIVERSITY

COMPUTER SCIENCE DEPARTMENT

March 10, 1965

Syllabus for the Ph.D Examination in Computer Systems,

Programming Systems and Programming Languages

The student taking the examination should have a background in the following areas:

1. Computer Components and Memories

1.1 A general familiarity with the elementary physical properties of the following devices which make them useful in digital circuitry: tubes, transistors, diodes, tunnel diodes, integrated circuits, cryotrons, #in films, and magnetic cores.

1.2 An understanding of the principles of operation of mechanically scanned memories (tape, drum, disc, card) and electrically scanned memories (core planes).

2. - Logical Design and the Structure of Digital Computers

2.1 Some familiarity with Boolean Algebra and the synthesis of Boolean Expressions (this assumes some knowledge of elementary minimization procedures; Karnaugh or Veitch diagrams).

2.2 A knowledge of the properties of elementary sequential circuits (flip flops, shift registers and counters).

2.3 An understanding of the common number systems used in present machines (sign magnitude, l's complement, 2's complement and binary coded decimal). The student should have some appreciation and familiarity with the fact that many algorithms exist for performing arithmetic in various number systems.

2.4 Computer Organization

An understanding of the basic organizational features of a Von Neuman Sequential Processor (the 7090 as an example) a Stack Machine (the B-5000 as an example) and an understanding of how data flows through a machine; and how I-O can be organized.

3. Programming Languages

3.1 A thorough familiarity with ALGOL 60 and B5500 ALGOL.

3.2 A general familiarity with some other programming languages such as FORTRAN, LISP, etc. and at least one machine code.

3.3 A knowledge of some of the more useful features of Iverson's notation.

4. Programming Systems

4.1 A general familiarity with the principles and organization of assemblers, interpreters and compilers and the knowledge of the problems connected with the implementation of various computer languages, in particular ALGOL (the mechanisms for compiling expressions, procedures etc., and the problems of storage allocation).

4.2 An understanding of the general principles and tasks of supervisory systems and the problems involved with time-sharing a computer and with parallel processing.

4.3 A thorough familiarity with at least 2 computers, and a general familiarity with the capabilities and organization of a machine of the "new generation".

5. Phrase Structure Languages

A knowledge of production grammars, the specification of a programming language by a syntax (BNF) parsing of sentences of phrase structure languages, and syntax directed compiling.

REFERENCES

A partial list of useful references is included below. Other references of a similar nature can be found in the library.

Components, Memories, Logical Design and Computer Structure

Braun, Digital Computer Design

Phister, Logical Design of Digital Computers

Buchholz, Planning a Computer System

Ledley, Digital Computer and Control Engineering

Beckman,.. . . "Development in the Logical Organization of Computer Arithmetic and Control Units", Proceedings of the IRE, January 1961, pp 53.

Rajchman, "Computer Memories : A Survey of the State of the Art", Proceedings of the IRE, January 1961, pp 104.

Mac Sorely, "High Speed Arithmetic in Binary Computers", Proceedings of the IRE, January 1961, pp 67.

Burks, A.W., Goldstine, H.H., and von Neumann, J. "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument", Collected Works of Von Neuman.

(References, cont'd)

Critchlow, A.J., "Generalized Multiprocessing and Multiprogramming Systems", AFIPS Conference Proceedings (FJCC - 1963), pp 107-126.

Amdahl, . . . "Architecture of the IBM System 1360", IBM Journal of Research and Development, Vol. 8, No. 2, April 1964, pp 87 - 101.

"The Structure of System 1360", IBM Systems Journal, Vol. 3, No's 2,3,1964.

Barton, "A New Approach to the Functional Design of a Digital Computer", Proc. WJCC 19, (1961) pp 393 - 396.

Programming Languages and Systems

Barton, R.S., "A Critical Review of the State of the Programming Art", AFIPS, Conference Proceedings (SJCC - 1963) pp 169 - 177.

Bobrow, Daniel and Raphael, Bertram. "A Comparison of List Processing Computer Languages", Vol 7, No, 4, April, 1964, Comm ACM pp 231 - 240.

Floyd, 'R.W., "The Syntax of Programming Languages - A Survey" IEEE Transactions on Electronic Computers, Vol, EC-13, no. 4, August 1964.

Iverson, A Programming Language, Wiley, 1962

Rosen, Saul, "Programming Systems and Languages, A Historical Survey", AFIPS Conference Proceedings (SJCC - 1964), pp 1-15.

Survey of Programming Languages and Processors", Comm ACM 6,3 (March 1963), PP 93-99.

Survey Issue on Programming Languages: IEEE Transactions on Electronics Computers, Vol, EC-13, No. 4, August 1964.

B. Randell and B. Russell; "ALGOL 60" Implementation; AP 1964