

COMPUTING CURRICULA GUIDELINES
FOR
ASSOCIATE DEGREE PROGRAMS IN

COMPUTING SCIENCES

TWO-YEAR COLLEGE
COMPUTING CURRICULA TASK FORCE

Computing Sciences Committee

Robert Campbell (Chair)	Manatee Community College
H. Paul Haiduk	Amarillo College
Carol Janik	Tompkins Cortland Comm. College
Karl Klee	Jamestown Community College
Paul Morneau	Adirondack Community College
Lawrence Page	Onondaga Community College

THE ASSOCIATION FOR COMPUTING MACHINERY
1993

ACM Two-Year College Education Committee
and
Task Force Steering Committee

Karl Klee (Chair)	Jamestown Community College
Richard Austing	University of Maryland
Helene Chlopan	University of Kentucky
John Impagliazzo	Hofstra University
Joyce Currie Little	Towson State University

© Copyright 1993 by the Association for Computing Machinery, Inc.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

Association for Computing Machinery, Inc.
1515 Broadway, 17th Floor
New York, New York 10036
(212) 869-7440

GUIDELINES FOR
COMPUTING SCIENCES

TABLE OF CONTENTS

PART I
CURRICULA STRUCTURE

1.0 Introduction and Charter 1

2.0 Goals and Overview of this Report..... 2

3.0 Curricula Goals and Profiles of Graduates 3

 3.1 Background of Students 3

 3.2 Curricular Goals 4

 3.3 Profiles of Graduates 4

4.0 Components of Curricula Design..... 5

 4.1 Subject Areas and Knowledge Units 5

 4.2 Emphasis, Depth and Underlying Themes..... 6

5.0 Building the Curricula..... 6

 5.1 Subject Areas 7

 5.2 Curriculum Options 8

 5.3 Curriculum Electives 9

 5.4 Social and Professional Context..... 9

6.0 Requirements from Other Disciplines..... 11

7.0 Resources..... 11

 7.1 Laboratories 12

 7.2 Faculty and Staff 12

 7.3 Institutional Support 13

8.0 Program Issues..... 13

 8.1 Accreditation 14

 8.2 Articulation 14

 8.3 Assessment and Evaluation 14

 8.4 Methodology of Instruction 15

9.0 Service Courses 15

 9.1 Service to Other Disciplines..... 15

 9.2 Needs of Industry and the Community 15

10.0 Conclusion 16

PART II
CURRICULA KNOWLEDGE UNIT DETAILS

1.0 Overview of Knowledge Units 17

2.0 Subject Areas and Related Knowledge Units..... 18

3.0 Details of Knowledge Units..... 20

PART III
CURRICULA IMPLEMENTATION SAMPLES

1.0 Curriculum Overview..... 45

 1.1 Course Structure..... 45

 1.2 Emphasis of Knowledge 48

 1.3 Depth of Knowledge 48

 1.4 Curriculum Structure..... 49

 1.5 Curriculum Courses 50

2.0 Underlying Themes 52

3.0 Sample Curricula 55

 3.1 Knowledge Unit Prerequisite Structure..... 55

 3.2 Knowledge Unit Lecture Hours..... 57

 3.3 A Sample Curriculum for Career Students 58

 3.4 A Sample Curriculum for Transfer Students 62

4.0	Descriptions of Sample Courses	64
4.1	A Prerequisite Course in Programming	65
4.2	Courses Required for Career and Transfer Students	66
4.3	Elective Courses for Career Students	73

**PART IV
SUPPORTIVE INFORMATION**

Endnotes.....	83
Bibliography	85
Career Data	87
Index of Knowledge Units	89
Index of Courses.....	91

LIST OF FIGURES

Figure II-1 Model of Knowledge Unit	18
Figure III-1 Course Format Example	47
Figure III-2 Knowledge Unit Prerequisite Structure	56
Figure III-3 Sequence of Required Courses - Transfer and Career	61

LIST OF TABLES

Table III-1 Subject Area Content by Lecture Hours..... 57

Table III-2 Course Scheduling Sequence - Career Program 60

Table III-3 Background-Dependent Sequencing Options - Transfer and Career.. 61

Table III-4 Course Scheduling Sequence - Transfer Program 64

Table III-5 Summary of Sample Courses by Lecture Hours 71

Table III-6 Percentages of Subject Areas in Sample Courses and Curricula 71

Table III-7 Expected Exit Competencies 72

PART I

CURRICULA STRUCTURE

1.0 INTRODUCTION AND CHARTER

In 1990 the Association for Computing Machinery (ACM) awarded the ACM Two-Year College Computing Curricula Task Force preliminary funding to develop curricular guidelines for two-year computing programs. The Task Force identified four broad but distinct curricular areas and the area of computing for other disciplines. Five separate committees of the Task Force were formed to investigate the following areas:

- Computer Support Services (CSS)
- Computing and Engineering Technology (CET)
- Computing for Information Processing (CIP)
- Computing Sciences (CS)
- Computing for Other Disciplines (COD)

Each committee solicited participation by a large number of individuals and produced a separate report. The Executive Report of the Task Force integrates the five reports and identifies commonalities between them. The Executive Report of the Task Force acknowledges all those who contributed to the work of the Task Force, participated in working sessions, or provided critical reviews.

This report addresses the computing sciences. These guidelines will serve the needs of those academic institutions (two-year colleges, four-year colleges and universities) offering two-year programs leading to an associate degree. The format of these guidelines is based in part upon *Computing Curricula 1991* [1], a publication sponsored jointly by ACM and the Computer Society of the IEEE, which presents curricular guidelines for four-year programs leading to a bachelor's degree in computer science or computer engineering. The *Computing Curricula 1991* report, which presents a comprehensive discussion of the curriculum together with suggestions, recommendations, and sample implementations, was based in part on the final report of the Task Force on the Core of Computer Science called *Computing as a Discipline* [2], but commonly referred to as the *Denning Report*.

The ACM is a leader in formulating curricula guidelines for the computing field. In 1968 the ACM Curriculum Committee on Computer Science published a report called *Curriculum 68* [3], which contained recommendations for academic programs in computer science. Curriculum development work continued to be undertaken by the ACM Curriculum Committee on Computer Science. In March, 1979, ACM published an updated refinement to *Curriculum 68* called *Curriculum 78* [4].

The report detailed a model curriculum for the study of computer science which included eight core computer science courses, five core mathematics courses, ten elective computer science courses, and two elective mathematics courses. This model generated a great deal of interest and curriculum implementation by two-year colleges, four-year colleges, and universities. Updates to CS 1 and CS 2, the first two core computer science courses in the *Curriculum 78* report, were published in 1984 [5] and in 1985 [6]. The most recent set of curriculum guidelines that influenced the direction of computing at two-year colleges was the 1981 working paper of the Subcommittee on Community and Junior College Curriculum of the ACM Committee on Curriculum in Computer Science [7]. The ACM has recently established a Task Force of the Pre-College Committee of the ACM Education Board which has produced a 1991 draft report titled *ACM Model High School Computer Science Curriculum* [8].

2.0 GOALS AND OVERVIEW OF THIS REPORT

The Computing Sciences Committee of the ACM Two-Year College Computing Curricula Task Force was charged with developing program guidelines which specifically address the two-year college setting. The Committee considered the differing needs of students preparing for an entry-level position in computer science after two years of study and of students preparing to transfer to four-year degree programs in computer science. In addition, the Committee addressed the minimal faculty and equipment resources needed to support these programs.

The goal of the Committee was to provide a framework for the development of programs and curricula in the computing sciences discipline. The recommendations in this report are divided into the following three parts:

- I. A set of curricular and pedagogical considerations that govern the mapping of the requirements into two-year programs of study. These include the roles of laboratories, requirements from other disciplines, social and professional context, and other educational experiences that combine to make up an entire two-year program of study.
- II. A collection of subject matter modules called *knowledge units*, that comprise the common requirements for two-year curricula in the computing sciences.
- III. A curricular mapping of the knowledge units into sample courses and curricula. The knowledge units within the courses have a measure of the emphasis and depth of coverage for each unit.

The curricula recommendations are intentionally flexible. Two-year programs must respond to local conditions, and the curricula offered must consider these needs. Consequently this report does not contain a single prescription of courses for all associate-level degree programs in the computing sciences. Part III includes sample courses and curricula. Individual institutions should style their courses and programs to fit local institutional and community constraints as well as local

3.0 CURRICULA GOALS AND PROFILES OF GRADUATES

The Computing Sciences Committee developed the curriculum recommendations by considering the differing needs of the two-year college computing sciences students. Since few two-year colleges have the resources and students to implement different curricula in the computing sciences, the Committee designed programs having a common core of knowledge units for students in the career track and students intending to transfer.

3.1 Background of Students

Experience with programming is an important prerequisite. The preparation of an entering student should include a working knowledge of a contemporary, block-structured procedural language and the tools for writing, compiling, and executing programs written in the language. The student may have received this preparation in a high-school course such as the Advanced Placement Computer Science course, or a course based upon the recommendations of the Task Force of the ACM Pre-College Committee. Colleges should be prepared to assess skills (for example, work-related experiences) of other students to determine if those individuals have sufficient preparation to enter the program.

Entering students should also have achieved mathematics preparation equivalent to four years of college preparatory mathematics, including preparation for calculus. In some states this preparation will include courses in algebra, geometry, and trigonometry; in others, it may be a sequence of integrated mathematics including sets, logic, probability, statistics, and number theory, in addition to algebra and trigonometry.

Colleges implementing a curriculum in the computing sciences based on this report should be prepared to offer prerequisite courses in computing and mathematics. Students interested in the computing sciences, but without sufficient background in mathematics or in programming, may need more than two years to prepare for and complete an associate-level degree program. For some students, an introductory programming course will provide adequate preparation. Such a course might be offered in a summer session, or as part of a tech-prep program, or as an element of one of the college's computing programs. A semester or more of study consisting of mathematics, an introductory programming course, an applications software course, and general education courses should be planned for students in need of more preparation.

3.2 Curricular Goals

The career-track students, including existing workers in the field, can best be served by a core curriculum in the computing sciences, with appropriate career-enhancement courses to provide

greater depth in application areas of computing. The nature of such elective courses will depend on the needs of local industries, the interests and expertise of the faculty, and the availability of specialized equipment. Students should develop a reasonable level of understanding in the various subject areas that define the discipline, possess an appreciation for the interrelationships among them, and study the ethical and societal issues in the computing field. Students should be prepared to apply their knowledge to specific real-world problems and produce appropriate solutions.

Students intending to transfer with junior-level status to a four-year college or university in a baccalaureate degree program in the computing sciences need a program which articulates well with the four-year colleges, but is consistent with the philosophy of two-year colleges. Therefore, the curriculum should provide transfer courses, the proper mix of academic and technical courses, and an environment in which students study the ethical and societal issues in the computing field.

Another group of students are those with an interest in the computing sciences. These students might be majoring in mathematics, other sciences, engineering, or business, or might be individuals in need of retraining. They can best be served by some of the core or elective computing courses of the two-year curriculum.

3.3 Profiles of Graduates

A student who has completed the career track should be ready, upon graduation, for employment in the computing field. Example career titles include Applications Programmer, Systems Programmer, Maintenance Programmer, Programmer/Analyst, and Numerical Analyst. Descriptions of selected career titles may be found in Part IV of this report. Graduates may wish to continue taking courses on a part-time basis at a two-year or four-year college to enhance their expertise, or begin working towards the completion of a baccalaureate degree.

A student who has completed the transfer track should be able to enter a baccalaureate degree program in computer science or computer engineering. This student should be admitted into the program with full junior status and be able to complete the program in two additional years.

4.0 COMPONENTS OF CURRICULA DESIGN

The computing sciences discipline is divided into ten major subject areas. Each subject area is then divided into knowledge units (KU). Courses within the curricula are composed of collections of knowledge units. Since a given knowledge unit may occur in more than one course, the expected emphasis and depth of the coverage of the knowledge unit is listed within the given course.

4.1 Subject Areas and Knowledge Units

The ten subject areas of the computing sciences discipline are:

1. Algorithms and Data Structures (AL)
2. Architecture (AR)
3. Artificial Intelligence and Robotics (AI)
4. Database and Information Retrieval (DB)
5. Numerical and Symbolic Computation (NU)
6. Operating Systems (OS)
7. Programming Languages (PL)
8. Introduction to Programming (PR)
9. Social, Ethical, and Professional Issues (SP)
10. Software Methodology and Engineering (SE)

Each of the ten subject areas is divided into a set of knowledge units. Each of the knowledge units within a subject area has a code tag, for example *AL6*. The two letters identify the subject area name. A number is used for a numerical ordering of the different knowledge units under the given subject name. Code tags are then identified by a knowledge unit name. Knowledge unit details are included in Part II of this report.

Another subject area, *Human-Computer Communication*, was identified by the ACM Task Force on the Core of Computer Science [9], and also included in *Computing Curricula 1991*. This subject area is not listed above and there are no separate knowledge units or lecture hours devoted to this subject area in the two-year computing sciences curricula. However, principles of good interface design, use of graphical interfaces, and related programming methods and issues are to be included in closed laboratory exercises as well as assigned projects. The shift from lecture to laboratory is not intended to diminish the importance of human-computer communications, but to emphasize the very practical aspects of this subject area.

The knowledge units in the subject area *Introduction to Programming* (PR) are prerequisite knowledge units. They should be completed prior to a course of study involving the other nine subject areas.

4.2 Emphasis, Depth and Underlying Themes

Within courses, each knowledge unit contains a listing of its emphasis. The emphasis is indicated by some combination of the three letters T, A, or D, which stand for the three paradigms *Theory*, *Analysis*, or *Design*, respectively. The complete explanation and usage are included in Part III of this report.

Within courses, an integer in the range 1 through 5 is used to indicate the depth of knowledge or understanding of each knowledge unit. The interpretation and usage of these integers are also included in Part III of the report.

Certain fundamental concepts recur throughout the discipline and play an important role in the

design of individual courses and whole curricula. These fundamental concepts are referred to as *underlying themes*. The underlying themes represent *glue* that can be used to provide cohesiveness for the discipline. The discipline can convey a wholeness by directly acknowledging and discussing these underlying themes as they appear at different times during the students' educational experiences. Done properly, students will avoid the problem of seeing the discipline as a fragmented collection of unrelated topics. Furthermore, learning will be facilitated by the presence of generalizations and analogies. Given a reference to an already understood concept, students can assimilate new ideas more easily. The details of the underlying themes for the computing sciences are included in Part III of this report.

5.0 BUILDING THE CURRICULA

The discipline of computing, as defined by the ACM Task Force on the Core of Computer Science, is the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. Their report also stated that the fundamental question underlying all of computing is, *What can be (efficiently) automated?* [10].

This definition provided the direction for developing the knowledge units and the sample curricula based upon these knowledge units. It is important to note in Parts II and III of this report that programming languages represent less than 25% of the time devoted to the various knowledge units. The Computing Sciences Committee realizes that this may represent a significant departure from current offerings at some two-year colleges in which a large proportion of time is spent studying various programming languages such as COBOL, FORTRAN, BASIC and Pascal.

5.1 Subject Areas

The subject areas for the Computing Sciences are identified and briefly summarized in this section. The subject titles also indicate the two-letter tag code of the knowledge unit format discussed in Section 4.1. Details of the knowledge units for each subject area may be found in Part II of this report. The subject areas for the computing sciences are the following:

Algorithms and Data Structures (AL)

This subject area emphasizes basic data structures, abstract data types, objects, problem-solving strategies, recursive algorithms, sorting and searching, and introductory complexity analysis.

Architecture (AR)

This subject area emphasizes machine level representation of data, organization and operation of major system components, instructions, memories, and interfacing and communication. Some alternative architectures are surveyed.

Artificial Intelligence and Robotics (AI)

This subject area emphasizes applied artificial intelligence, problem-solving strategies, expert systems, and robotics.

Database and Information Retrieval (DB)

This subject area emphasizes database and information retrieval concepts, including external and logical storage structures and devices, relational models, and integrity concerns.

Numerical and Symbolic Computation (NU)

This subject area emphasizes iterative approximation methods and errors that can occur in computer arithmetic.

Operating Systems (OS)

This subject area emphasizes history of operating systems, tasks, processes, concurrency, physical and virtual memory organization, management of devices and files, data communications and security concerns.

Programming Languages (PL)

This subject area emphasizes the history and overview of programming languages, specification of data types, sequence control, data access, programming paradigms, concurrency, and an introduction to language translation systems.

Introduction to Programming (PR)

This subject area introduces the syntactic and structural characteristics of a modern, block-structured programming language and its use in solving simple algorithmic problems. This subject area is prerequisite to the other nine subject areas.

Social, Ethical, and Professional Issues (SP)

This subject area emphasizes the historical and social context of computing, responsibilities of the computing professional, risks and liabilities, and intellectual property.

Software Methodology and Engineering (SE)

This subject area emphasizes problem solving concepts, the software development process, software specifications, software design and implementation, verification, and validation.

5.2 Curriculum Options

The knowledge units in one subject area do not have to be grouped into one course. They may be distributed among several courses. Furthermore, some knowledge units may be covered either in a standard class setting or in a laboratory setting. Thus, the knowledge units can be combined in

various ways to form courses.

In Part III of this report, the knowledge units are arranged into sample curricula for the two-year college transfer and career computer science programs. While the two sample curricula are different, they share a common core of four computing courses. The sample curricula describe the content of each course making up the core and the number of suggested hours for each knowledge unit. Underlying themes such as efficiency, consistency, levels of abstraction and correctness are basic concerns throughout the discipline. The emphasis on theory, analysis, and design present in each knowledge unit within courses is also described. Each of the four core courses is designed to be a four semester-hour course meeting in lecture three hours per week, and supported by weekly closed laboratory periods of two to three hours. Suggestions for laboratory activities are also noted for each of the core courses.

5.3 Curriculum Electives

The knowledge units within the subject area *Introduction to Programming* are prerequisite to the computing sciences curriculum. The knowledge units within the other nine subject areas form the core of the computing sciences program.

Students who plan to seek employment after the completion of the associate degree must complete additional computing courses. Many of these courses may already be available in other programs such as computing for information processing, computing and engineering technology, or computer support services. Two-year colleges may wish to develop additional elective courses, depending, in part, upon local industrial and business needs. These courses may include, but are not limited to the following:

- Database management systems
- Data communications
- Artificial intelligence and robotics
- Operating systems
- Computer graphics
- Digital electronics
- Numerical computing
- Computer networks
- Computer-aided design
- Programming in another language
- Microcomputer applications
- Object-oriented systems

Section III of this report contains the descriptions and objectives of sample courses in these areas.

For students planning to transfer to a four-year degree program in one of the computing sciences,

additional courses, suited to the requirements of the transfer institution, should be chosen in consultation with an academic advisor.

5.4 Social and Professional Context

To prepare the graduate for the modern workplace, the program should introduce the student to elements of professional responsibility and ethical behavior, to problem solving and decision making, and to the fundamentals of intellectual property rights. Professional responsibility and ethical behavior may be presented as case studies which demonstrate the potential conflicts in loyalties to the firm or to the customer. Attention should be paid to the issues of dealing with confidential information of the firm or the customer. Most professional societies have adopted a standard of ethics, such as the ACM Code of Ethics.

Intellectual property rights are the bases for most contracts in computer systems, because the value resides more often in the software than in the hardware. Responsible computer systems personnel need to understand what copyright means and what typical software license contracts require of each of the parties.

Students need to understand the basic cultural, social, legal and ethical issues inherent in the discipline of computing. They should be aware of the history, current issues, and trends of the discipline. They should also examine their individual roles in this process, as well as appreciate the philosophical questions, technical problems, and aesthetic values that have contributed to the development of the discipline. Students also need to question the social impact of computing and to evaluate proposed solutions. Future practitioners must be able to anticipate a product's effect on a given environment. Will that product enhance or degrade the quality of life? How will it impact individuals, groups and institutions? What will be the trade-offs between improved information technology and personal privacy?

Students also need to be aware of the basic legal rights of software and hardware vendors and users. They need to appreciate the ethical values which determine those rights, the responsibility that they will bear, and the consequences of failure to maintain appropriate professional standards.

Perhaps the most important but least considered ethical aspect in computing is the robust design of software. A program is robust if it can recover from unexpected or erroneous input, and therefore, must be able to recognize rarely-occurring input as possible cases. Considering all possible real-life cases and responding effectively to each requires good design and careful program verification. The risks of poor design may include losing lives in a medical, military, or outer-space application, or losing property and profits in a business or technical application.

These social and professional issues are included in the subject area Social, Ethical and Professional Issues. The knowledge units within this subject area are incorporated into the four required computing courses for the computing sciences curricula.

6.0 REQUIREMENTS FROM OTHER DISCIPLINES

The computing sciences have become disciplines in their own rights, with a theoretical basis, a framework of analysis, and a design aspect. Within these contexts, they draw their methods and practices very heavily from mathematics and the natural sciences. Therefore a strong foundation in mathematics and in the natural sciences is essential for the computing sciences students.

For both career and transfer-level computer science, the suggested minimum mathematical competency needed for graduation will include two semesters of calculus and a semester of discrete mathematics.

Laboratory science courses, particularly physics, are an important part of the computing sciences programs. These courses should emphasize the understanding, measurement, and quantitative expression of physical processes. Laboratory work, including experimentation, observation, measurement, and report writing, should be required.

Technically trained individuals should be able to communicate, both orally and in writing, their observations, research, and viewpoints. Therefore, associate-level degree programs should include the following courses in English: written and oral presentation, literature, and technical writing. Writing must be integrated into the computing courses. Student reports should be neat, grammatically correct, and coherent. Individual and group oral presentations should be encouraged. Furthermore, since the ability to read and assimilate technical literature is an important and necessary skill, students should be encouraged to read technical publications and to use manufacturer's literature and data books in support of their laboratory assignments.

Because of the international nature of technical markets, students should understand other peoples, their cultures and languages. Students must also sense the responsibility they have to protect both the local and the global society and the environment in which they live. General education courses that support global understanding should be encouraged.

7.0 RESOURCES

This section summarizes the requirements for laboratories, faculty and staff, and institutional support critical for the implementation and maintenance of a computing sciences program.

7.1 Laboratories

Institutions must provide instruction in the computing sciences in a realistic and up-to-date environment. Adequate laboratory facilities are imperative. Laboratory equipment and computers should be similar to that encountered in industry and practice. Furthermore, sufficient equipment must be available to provide each student with the opportunity to become thoroughly familiar with

the use and operation of equipment and computers common to their major field of study. In addition, manuals, equipment catalogs, professional magazines, and journals should supplement the usual library resources.

Students in the computing sciences must have access to both open and closed laboratories. Open laboratories, staffed by trained assistants, must be available for homework and project assignments. Laboratory experiences must go beyond program-design assignments developed by instructors. Students need to learn an experimental approach to problem solving in much the same way as in other sciences such as chemistry, physics, and biology. As students form conjectures and discover results, they develop a greater sense of accomplishment. This experience becomes a valuable portion of their training for future study and eventual employment. For that reason, the core courses for the computing sciences program require what are called closed laboratories. A closed laboratory experience is structured, held at a scheduled time, has an instructor present, and is summarized in a laboratory report. It may require specialized software tools, hardware or instrumentation. Some experiments should encourage teamwork among students; others should include oral presentations by students.

Provision for updating equipment in response to changing practices and processes is very important. A plan for laboratory improvement should be implemented. This plan should include the purchase, or lease, of equipment and software, licenses and contracts, and regular maintenance. This plan should also provide adequate technical support for hardware and software rather than expecting faculty to provide this support. The plan should also be reviewed and updated periodically in light of changing student enrollment patterns and new software and hardware developments.

7.2 Faculty and Staff

The faculty and staff determine the strength of a good program. Full-time faculty are encouraged to have current training in the changing discipline of the computing sciences. New faculty members should have at least a master's degree or the equivalent in computer science. Since the two-year college serves the community, it should rely, in part, upon adjunct faculty working in the profession, as well as faculty in related disciplines, to teach some of the courses. Their expertise should provide additional support for the program. The number of full-time faculty must be great enough to provide a breadth of perspective, program continuity, and proper frequency of course offerings. Full-time faculty must also constitute the nucleus for planning and coordinating the computing sciences program.

Due to the rapidly changing field of technology, faculty members must maintain current knowledge of their field and understanding of the tasks industry expects their graduates to perform. Faculty members can remain current by active participation in professional societies, continuing education, reading the literature, and periodic returns to graduate school or to industry. The institution should

have a planned, adequately funded program for professional development of its faculty. As these curricular recommendations are being implemented, local, regional, or national workshops may be needed to prepare existing faculty to effectively teach the core computing sciences topics. Grants from government and industry should be sought to help finance these workshops.

7.3 Institutional Support

The institution must provide adequate support for the instructional program including secretarial and technical support, office space, travel funds and released time. Satisfactory procedures and qualified support personnel are required to keep laboratory and instructional equipment in good repair and working order. Faculty office space should provide adequate privacy for student counseling and advisement. Institutional and department budgets should provide travel funding and released time for faculty to attend conferences, workshops, and professional meetings.

Adequate library holdings are important to any program. Current publications, including a variety of technical journals, are critical to the viability of a computing sciences program. In such a rapidly changing field, a special effort must be made to keep the library holdings current. Faculty should assign work which is dependent upon the library publications and should emphasize the on-going need to remain current in the ever-changing computing sciences field.

8.0 PROGRAM ISSUES

This section summarizes the recommendations on accreditation and the articulation with other institutions and with industry.

8.1 Accreditation

Currently there is no formal accreditation mechanism for programs in the computing sciences at two-year colleges. The Committee has strongly considered the recommendations of the Computer Science Accreditation Board (CSAB) [11], and the recent ACM/IEEE-CS publication *Computing Curricula 1991*. It is vitally important that institutions implement programs of study which reflect standards necessary for the integrity of the institutions, the assurance of transferability, and the preparation for employment in today's world. The curriculum must contain strong foundations in mathematics, science, and other supporting disciplines. The courses in the computing sciences must be rigorous, thorough, and current. The faculty must meet or exceed certain minimum standards, and must participate in professional development activities, such as conference and workshops, to remain current in the field. The laboratory facilities must provide the necessary support critical to maintain a comprehensive curriculum. Other support areas, such as libraries, office equipment, and classroom facilities, must be sufficient as well.

8.2 Articulation

Articulation issues are critical to the vitality of a two-year college curriculum. Articulation is an activity which must be pursued energetically and maintained vigilantly by two-year colleges with local colleges, universities, businesses, and industries.

For career programs, articulation helps to assure that students are adequately prepared to compete for entry-level positions in targeted job titles. This process is greatly enhanced by establishing local advisory councils which are actively involved in the program development, review, and revision processes.

For transfer programs, articulation means that written agreements have been made concerning the awarding of credit at particular colleges or universities for courses completed at the two-year college. The recommendations contained in this report for the transfer computer science curriculum are consistent with *Computing Curricula 1991*. This was done, in part, to stimulate discussions between two-year college faculty and their colleagues at the transfer institutions, and to enhance the likelihood that courses would transfer to four-year degree programs in the computing sciences.

8.3 Assessment and Evaluation

Recommendations for the design and content of assessment programs are beyond the scope of this report. However, this is an important area for any institution that wishes to maintain quality, up-to-date programs in the computing sciences.

8.4 Methodology of Instruction

The delivery method needs to evolve from the traditional lecture method to methods which allow the student to become more involved and more active in the learning processes. Examples of other methodologies include:

- Using interactive learning devices
- Using multimedia in the classroom
- Researching a subject using networked databases
- Solving actual campus problems in course projects
- Requiring students to learn and present new material to the class
- Designing team projects for certain closed and open laboratory assignments
- Exploring the subject through field trips

9.0 SERVICE COURSES

This section discusses the role of the computing sciences faculty in serving the needs of other

disciplines within the college, local industry, and community groups.

9.1 Service to Other Disciplines

The computing sciences faculty are valuable resources to other departments within the college. They are frequently called upon to provide the development and the instruction for computer-related courses within the curriculum of another department. Examples include the teaching of scientific programming, using FORTRAN or C, to engineering or engineering technology students, and the teaching of microcomputer applications, using word processing, database management, and spreadsheets, to students in a wide variety of majors. Being able to offer this service strengthens the position of the department and enhances its role within the structure of the college.

9.2 Needs of Industry and the Community

Since a role of the two-year college is to help serve the needs of local industry and the community, the computing sciences faculty must be especially sensitive to these needs and receptive to meeting them. Some of these needs may be met by regularly-offered credit-bearing courses such as artificial intelligence and robotics, and database management, which might serve a local need or be a catalyst for attracting industry to the community.

A very specific need of a local industry or community group might be best served by a credit-free, continuing-education course. An example is the hands-on study of the Unix operating system. Other credit-free courses might include hands-on microcomputer applications training in packages such as Lotus 1-2-3, dBase IV, Pagemaker, or Hypercard.

10.0 CONCLUSION

This report presents curricular recommendations for associate degree programs in the discipline of computing sciences. The recommendations prescribe knowledge units along with sample programs of study which show how these knowledge units may be aligned into courses. The use of knowledge units in courses indicates not only topics, but also an emphasis and a depth of intensity. Course development is not merely the expansion of a subject area. Rather it is the creative selection from various knowledge units of different subject areas. The manner of selection of knowledge units is a function of each institution, and should result in a modern presentation of courses offered by these colleges.

This report was prepared by the Computing Sciences Committee of the Two-Year College Computing Curricula Task Force of the Association for Computing Machinery. The Steering Committee of the Task Force is appreciative of the many individuals and organizations that supported this curriculum project. A listing of their names appears in the Executive Report.

PART II

CURRICULA KNOWLEDGE UNIT DETAILS

1.0 OVERVIEW OF KNOWLEDGE UNITS

The fundamental elements upon which courses in a curriculum are formulated are called knowledge units (KU). Knowledge units may be considered as content units within a curriculum. These units are made up of the following elements: subject name, knowledge unit tag and name, description, topics, prerequisites, and requisite for. The *Subject Name* identifies the topic category. Each of the knowledge units within a subject name has a *Knowledge Unit Tag and Name* with the following format:

Letter-1 Letter-2 Integer: Knowledge Unit Name

The two letters identify the subject name. The integer is used for a numerical ordering of the different knowledge units under a given subject name. For example:

AL2: STRUCTURED DATA TYPES

indicates that the knowledge unit entitled *Structured Data Types* is in the subject area *Algorithms and Data Structures*, and it is the second in the set of knowledge units for that subject area. The *Description* section contains a synopsis of the particular knowledge unit. The *Topics* section identifies the particular concepts to be addressed when the knowledge unit is covered. The indicated *Minimum number of hours:* is lecture hours; there may be additional time devoted to the knowledge unit in the laboratory. Knowledge units are related to other knowledge units within the curriculum as shown in the *Prerequisite* and *Requisite For* sections.

Each of the ten subject areas of the computing sciences discipline is divided into a set of knowledge units. Each knowledge unit is presented in the format shown in Figure II-1.

SUBJECT NAME **ALGORITHMS AND DATA STRUCTURES**

KNOWLEDGE UNIT **AL2: STRUCTURED DATA TYPES**
TAG AND NAME

DESCRIPTION Definitions and use of structured data types provided in most procedural languages. Emphasis on specification, set of values, set of operations, and representation.

TOPICS Minimum number of hours: 8
 1. One-dimensional arrays
 2. Strings
 3. Records
 4. Multidimensional arrays

PREREQUISITES AL1

REQUISITE FOR AL3, AL4, PL2, SE1

Figure II-1 Model of Knowledge Unit

2.0 SUBJECT AREAS AND RELATED KNOWLEDGE UNITS

This section contains a listing of the knowledge units within the ten subject areas.

Algorithms and Data Structures (AL)

- AL1: Introduction to Problem-Solving Strategies
- AL2: Structured Data Types
- AL3: Introduction to Sorting and Searching
- AL4: Data Structures
- AL5: Recursive Algorithms
- AL6: Sorting and Searching
- AL7: Objects
- AL8: Complexity
- AL9: Problem-Solving Strategies

Architecture (AR)

- AR1: Machine Representation of Data

- AR2: Machine Organization
- AR3: Machine and Assembly Language Programming
- AR4: Memory System and Architecture
- AR5: Interfacing and Communication
- AR6: Alternative Architectures

Artificial Intelligence and Robotics (AI)

- AI1: Overview of Applied Artificial Intelligence
- AI2: Search Strategies

Database and Information Retrieval (DB)

- DB1: File and Physical Database Organization

Numerical and Symbolic Computation (NU)

- NU1: Numerical Computation

Operating Systems (OS)

- OS1: Tasking and Processes

Programming Languages (PL)

- PL1: History of Programming Languages
- PL2: Specification of Data Types
- PL3: Sequence Control
- PL4: Object-Oriented Programming
- PL5: Data Access
- PL6: Language Translation Systems
- PL7: Programming Paradigms
- PL8: Concurrency

Introduction to Programming (PR)

- PR1: Elements of a Programming Language
- PR2: Data Types
- PR3: Algorithms and Program Design
- PR4: Documentation, Testing, and Verification

Social, Ethical, and Professional Issues (SP)

- SP1: Evolution of Computing and the Professional
- SP2: Intellectual Property

- SP3: Software Protection and Security
- SP4: System Security
- SP5: Social Responsibility of Professionals
- SP6: Data Collection and Privacy
- SP7: Artificial Intelligence and Society
- SP8: Risks in Large Systems

Software Methodology and Engineering (SE)

- SE1: Fundamental Problem-Solving Concepts
- SE2: Software Development Models
- SE3: Software Requirements and Specifications
- SE4: Software Design and Development
- SE5: Verification and Validation
- SE6: Software Design and Implementation

3.0 DETAILS OF KNOWLEDGE UNITS

This section contains the details of the knowledge units of the ten subject areas within the computing sciences discipline. The details follow the format listed in Figure II-1. Figure III-1 in Part III of this report contains a graph of the prerequisite-requisite structure of the knowledge units.

The knowledge units in the subject area *Social, Ethical, and Professional Issues* (SP) are designed to be nearly independent from one another. This was done to encourage the integration of these knowledge units throughout the curriculum. The knowledge units in the subject area *Introduction to Programming* (PR) are prerequisite knowledge units. They should be completed prior to a course of study involving the other nine subject areas.

Knowledge units within the same subject area are separated by a single bar; knowledge units in different subject areas are separated by double bars.

SUBJECT NAME	ALGORITHMS AND DATA STRUCTURES
KNOWLEDGE UNIT TAG AND NAME	AL1: INTRODUCTION TO PROBLEM-SOLVING STRATEGIES
DESCRIPTION	Examples of problem-solving strategies.
TOPICS	Minimum number of hours: 2 1. Problem subtasks 2. Algorithm development 3. Algorithm refinement
PREREQUISITES	None
REQUISITE FOR	AL2, PL3

SUBJECT NAME	ALGORITHMS AND DATA STRUCTURES
KNOWLEDGE UNIT TAG AND NAME	AL2: STRUCTURED DATA TYPES
DESCRIPTION	Definitions and use of structured data types provided in most procedural languages. Emphasis on specification, set of values, set of operations, and representation.
TOPICS	Minimum number of hours: 8 1. One-dimensional arrays 2. Strings 3. Records 4. Multidimensional arrays
PREREQUISITES	AL1
REQUISITE FOR	AL3, AL4, PL2, SE1

SUBJECT NAME	ALGORITHMS AND DATA STRUCTURES
KNOWLEDGE UNIT TAG AND NAME	AL3: INTRODUCTION TO SORTING AND SEARCHING
DESCRIPTION	Simple sorting and searching algorithms.
TOPICS	Minimum number of hours: 3 1. Linear search of an array 2. Sorting an array (e.g., selection)
PREREQUISITES	AL2
REQUISITE FOR	AL6, PL4

SUBJECT NAME	ALGORITHMS AND DATA STRUCTURES
KNOWLEDGE UNIT TAG AND NAME	AL4: DATA STRUCTURES
DESCRIPTION	Specification and representation of fundamental data structures; definition and concepts of abstract data types.
TOPICS	Minimum number of hours: 7 1. Stacks and queues 2. Linked lists 3. Trees and graphs 4. Information hiding
PREREQUISITES	AL2
REQUISITE FOR	AL5, AL7, DB1, SE3

SUBJECT NAME	ALGORITHMS AND DATA STRUCTURES
KNOWLEDGE UNIT TAG AND NAME	AL5: RECURSIVE ALGORITHMS
DESCRIPTION	Recursive algorithms and their relation to mathematical induction. Comparison of iterative and recursive algorithms.
TOPICS	<p>Minimum number of hours: 3</p> <ol style="list-style-type: none"> 1. Introduction to recursive algorithms 2. Connections with mathematical induction 3. Comparisons of iterative and recursive algorithms 4. Using stacks to implement recursion
PREREQUISITES	AL4, Mathematical Induction
REQUISITE FOR	AL6, PL7

SUBJECT NAME	ALGORITHMS AND DATA STRUCTURES
KNOWLEDGE UNIT TAG AND NAME	AL6: SORTING AND SEARCHING
DESCRIPTION	Examination of fundamental sorting and searching algorithms; analysis and comparison to support selection of appropriate algorithms in various situations.
TOPICS	<p>Minimum number of hours: 6</p> <ol style="list-style-type: none"> 1. $O(n*n)$ and $O(n*\log n)$ sorting algorithms 2. Linear and binary search algorithms 3. Analysis of best, average, and worst cases 4. Comparison of algorithms
PREREQUISITES	AL3, AL5
REQUISITE FOR	AL8, AL9, AI1

SUBJECT NAME	ALGORITHMS AND DATA STRUCTURES
KNOWLEDGE UNIT TAG AND NAME	AL7: OBJECTS
DESCRIPTION	Object-oriented data types and paradigm.
TOPICS	Minimum number of hours: 3 1. Definition of object-oriented paradigm 2. Examples of objects in an object-oriented language 3. Advantages and disadvantages of using objects
PREREQUISITES	AL4, PL4
REQUISITE FOR	PL7, SE3

SUBJECT NAME	ALGORITHMS AND DATA STRUCTURES
KNOWLEDGE UNIT TAG AND NAME	AL8: COMPLEXITY
DESCRIPTION	An introduction to asymptotic analysis of algorithms and complexity classes.
TOPICS	Minimum number of hours: 5 1. Asymptotic analysis and notation 2. Time-space tradeoffs in algorithms 3. Complexity classes P, NP, and P-Space
PREREQUISITES	AL6
REQUISITE FOR	AI1

SUBJECT NAME	ALGORITHMS AND DATA STRUCTURES
KNOWLEDGE UNIT TAG AND NAME	AL9: PROBLEM-SOLVING STRATEGIES
DESCRIPTION	The correctness and complexity of algorithms.
TOPICS	Minimum number of hours: 4 1. Greedy algorithm 2. Divide and conquer algorithm 3. Backtracking algorithm
PREREQUISITES	AL6
REQUISITE FOR	None

SUBJECT NAME	ARCHITECTURE
KNOWLEDGE UNIT TAG AND NAME	AR1: MACHINE REPRESENTATION OF DATA
DESCRIPTION	Methods of representing numeric and non-numeric data in a machine.
TOPICS	Minimum number of hours: 3 1. Binary, octal, and hexadecimal number systems 2. Unsigned, signed, one's and two's complement integers 3. Floating point numbers 4. Decimal and BCD numbers 5. Alphanumeric data
PREREQUISITES	None
REQUISITE FOR	AR2, NU1

SUBJECT NAME	ARCHITECTURE
KNOWLEDGE UNIT TAG AND NAME	AR2: MACHINE ORGANIZATION
DESCRIPTION	Introduction to major system components and their functions.
TOPICS	Minimum number of hours: 3 1. Basic system organization and function (processor, storage, I/O units) 2. Instruction representation 3. Control unit, instruction fetch and execution, operand fetch
PREREQUISITES	AR1
REQUISITE FOR	AR3, AR4, AR6

SUBJECT NAME	ARCHITECTURE
KNOWLEDGE UNIT PROGRAMMING TAG AND NAME	AR3: MACHINE AND ASSEMBLY LANGUAGE
DESCRIPTION	Programming in assembly and machine language.
TOPICS	Minimum number of hours: 7 1. Introduction to assembly language programming 2. Machine representation and execution of instructions 3. Implementation of high-level language structures
PREREQUISITES	AR2, PL3
REQUISITE FOR	AR5, PL6

SUBJECT NAME	ARCHITECTURE
KNOWLEDGE UNIT TAG AND NAME	AR4: MEMORY SYSTEM AND ARCHITECTURE
DESCRIPTION	Introduction to main memory organization and addressing.
TOPICS indexed)	Minimum number of hours: 4 1. Main memory organization 2. Memory addressing (direct, memory indirect, register indirect, 3. Bus operation, control, direct memory access (DMA) 4. Cache memory, read and write 5. Virtual memory
PREREQUISITES	AR2
REQUISITE FOR	OS1

SUBJECT NAME	ARCHITECTURE
KNOWLEDGE UNIT TAG AND NAME	AR5: INTERFACING AND COMMUNICATION
DESCRIPTION	Input and output operations and control.
TOPICS	Minimum number of hours: 4 1. I/O operations, control methods 2. Interrupts and handlers 3. Synchronization
PREREQUISITES	AR3
REQUISITE FOR	OS1

SUBJECT NAME	ARCHITECTURE
KNOWLEDGE UNIT TAG AND NAME	AR6: ALTERNATIVE ARCHITECTURES
DESCRIPTION	Survey of various architectures in use and in development.
TOPICS	Minimum number of hours: 2 1. Stack, vector, and array processors 2. RISC and CISC architectures
PREREQUISITES	AR2
REQUISITE FOR	None

SUBJECT NAME	ARTIFICIAL INTELLIGENCE AND ROBOTICS
KNOWLEDGE UNIT TAG AND NAME	AI1: OVERVIEW OF APPLIED ARTIFICIAL INTELLIGENCE
DESCRIPTION	An overview of applied artificial intelligence, contrasting intelligent computing with conventional computing. The advantages and limitations of artificial intelligence are emphasized.
TOPICS	Minimum number of hours: 3 1. General problem solving 2. Expert systems 3. Natural language 4. Computer vision 5. Robotics
PREREQUISITES	AL6, AL8
REQUISITE FOR	AI2

SUBJECT NAME	ARTIFICIAL INTELLIGENCE AND ROBOTICS
KNOWLEDGE UNIT TAG AND NAME	AI2: SEARCH STRATEGIES
DESCRIPTION	Introduction to search strategies used in artificial intelligence.
TOPICS	Minimum number of hours: 3 1. Depth-first and breadth-first searches 2. Heuristic search algorithms such as generate-and-test and hill climbing
PREREQUISITES	AI1
REQUISITE FOR	SP7

SUBJECT NAME	DATABASE AND INFORMATION RETRIEVAL
KNOWLEDGE UNIT TAG AND NAME	DB1: FILE AND PHYSICAL DATABASE ORGANIZATION
DESCRIPTION	Fundamental file organization and access methods.
TOPICS	Minimum number of hours: 3 1. Sequential and non-sequential file organization 2. Sequential and non-sequential access methods 3. Blocking and buffering
PREREQUISITES	AL4
REQUISITE FOR	None

SUBJECT NAME	NUMERICAL AND SYMBOLIC COMPUTING
KNOWLEDGE UNIT TAG AND NAME	NU1: NUMERICAL COMPUTATION
DESCRIPTION	Standard methods for computing iterative approximations, and sources and propagation of errors in computer arithmetic.
TOPICS	Minimum number of hours: 4 1. Iterative techniques and convergence 2. Overview of numerical applications in science and engineering 3. Errors in representation and arithmetic 4. Effects of errors on portability of numerical software
PREREQUISITES	AR1, PL3, Mathematics to support applications
REQUISITE FOR	None

SUBJECT NAME	OPERATING SYSTEMS
KNOWLEDGE UNIT TAG AND NAME	OS1: TASKING AND PROCESSES
DESCRIPTION	An introduction to multi-tasking.
TOPICS	Minimum number of hours: 5 1. Tasks and processes 2. Dispatching, scheduling, context switching 3. Time sharing
PREREQUISITES	AR4, AR5
REQUISITE FOR	None

SUBJECT NAME	PROGRAMMING LANGUAGES
KNOWLEDGE UNIT TAG AND NAME	PL1: HISTORY OF PROGRAMMING LANGUAGES
DESCRIPTION	The history of early programming languages and the evolution of procedural languages.
TOPICS	Minimum number of hours: 1 1. Early languages (FORTRAN, ALGOL) 2. Development of procedural languages (e.g., PL/1, C, Pascal, Ada)
PREREQUISITES	None
REQUISITE FOR	None

SUBJECT NAME	PROGRAMMING LANGUAGES
KNOWLEDGE UNIT TAG AND NAME	PL2: SPECIFICATION OF DATA TYPES
DESCRIPTION	Methods of specifying commonly available simple and composite data types, and creating new composite types.
TOPICS	Minimum number of hours: 2 1. Specifying types provided by a language 2. Specifying and representing programmer-defined composite types
PREREQUISITES	AL2
REQUISITE FOR	PL4

SUBJECT NAME	PROGRAMMING LANGUAGES
KNOWLEDGE UNIT TAG AND NAME	PL3: SEQUENCE CONTROL
DESCRIPTION	Sequence of operations in evaluating expressions and implementing statements, including exception handling.
TOPICS	Minimum number of hours: 4 1. Expression evaluation 2. Simple and compound statements 3. User-defined subprograms as expression and statement extensions 4. Defensive programming and error recovery
PREREQUISITES	AL1
REQUISITE FOR	AR3, NU1, PL5

SUBJECT NAME	PROGRAMMING LANGUAGES
KNOWLEDGE UNIT TAG AND NAME	PL4: OBJECT-ORIENTED PROGRAMMING
DESCRIPTION	An introduction to the objected-oriented paradigm.
TOPICS	Minimum number of hours: 4 1. Encapsulation 2. Methods 3. Messages 4. Languages
PREREQUISITES	AL3, PL2
REQUISITE FOR	AL7, PL7

SUBJECT NAME	PROGRAMMING LANGUAGES
KNOWLEDGE UNIT TAG AND NAME	PL5: DATA ACCESS
DESCRIPTION	The mechanisms for providing access to data, and control of those mechanisms.
TOPICS	Minimum number of hours: 4 <ol style="list-style-type: none"> 1. Static and dynamic scope 2. Parameter passing mechanisms 3. Type checking, including static, dynamic, untyped, explicit, implicit 4. Run-time storage management
PREREQUISITES	PL3
REQUISITE FOR	None

SUBJECT NAME	PROGRAMMING LANGUAGES
KNOWLEDGE UNIT TAG AND NAME	PL6: LANGUAGE TRANSLATION SYSTEMS
DESCRIPTION	Overview of programming language translation processes.
TOPICS	Minimum number of hours: 4 <ol style="list-style-type: none"> 1. Lexical analysis, symbol tables 2. Interpreters 3. Assemblers and linkers 4. Compilers, code generation and optimization
PREREQUISITES	AR3
REQUISITE FOR	None

SUBJECT NAME	PROGRAMMING LANGUAGES
KNOWLEDGE UNIT TAG AND NAME	PL7: PROGRAMMING PARADIGMS
DESCRIPTION	Non-procedural paradigms and languages, including applications of each and comparison with procedural programming.
TOPICS	Minimum number of hours: 10 1. Functional languages 2. Logic languages 3. Object-oriented languages
PREREQUISITES	AL5, AL7, PL4
REQUISITE FOR	PL8

SUBJECT NAME	PROGRAMMING LANGUAGES
KNOWLEDGE UNIT TAG AND NAME	PL8: CONCURRENCY
DESCRIPTION	Parallel processing, tasks and processes, scheduling algorithms, algorithm design considerations.
TOPICS	Minimum number of hours: 4 1. Semaphores 2. Signals 3. Coroutines 4. Deadlocks 5. Process schedulers
PREREQUISITES	PL7
REQUISITE FOR	None

SUBJECT NAME	INTRODUCTION TO PROGRAMMING
KNOWLEDGE UNIT TAG AND NAME	PR1: ELEMENTS OF A PROGRAMMING LANGUAGE
DESCRIPTION	An introduction to the syntax, logic and concepts of a contemporary block-structured programming language in its system environment.
TOPICS	<p>Minimum number of hours: 12</p> <ol style="list-style-type: none"> 1. Overall program structure 2. Type declarations 3. Arithmetic operators and assignment statements 4. Selection and iteration 5. Subprograms with parameters 6. Arrays
PREREQUISITES	None
REQUISITE FOR	Study in the computing sciences

SUBJECT NAME	INTRODUCTION TO PROGRAMMING
KNOWLEDGE UNIT TAG AND NAME	PR2: DATA TYPES
DESCRIPTION	Atomic verses structured data types. An introduction to user-defined data types and applications.
TOPICS	<p>Minimum number of hours: 3</p> <ol style="list-style-type: none"> 1. Choice and representation of data types 2. User-defined data types with applications
PREREQUISITES	PR1
REQUISITE FOR	Study in the computing sciences

SUBJECT NAME	INTRODUCTION TO PROGRAMMING
KNOWLEDGE UNIT TAG AND NAME	PR3: ALGORITHMS AND PROGRAM DESIGN
DESCRIPTION	Design and implementation of computer solutions to problems.
TOPICS	Minimum number of hours: 7 1. Definition of an algorithm 2. Use of algorithms in problem solving 3. Program implementation using top-down design
PREREQUISITES	PR1
REQUISITE FOR	Study in the computing sciences

SUBJECT NAME	INTRODUCTION TO PROGRAMMING
KNOWLEDGE UNIT TAG AND NAME	PR4: DOCUMENTATION, TESTING, AND VERIFICATION
DESCRIPTION	General requirements for program documentation. Elements of software engineering as related to program testing and verification.
TOPICS	Minimum number of hours: 3 1. Style and documentation 2. Tracing, testing, and verification of program correctness
PREREQUISITES	PR1
REQUISITE FOR	Study in the computing sciences

SUBJECT NAME	SOCIAL, ETHICAL, AND PROFESSIONAL ISSUES
KNOWLEDGE UNIT TAG AND NAME	SP1: EVOLUTION OF COMPUTING AND THE PROFESSIONAL
DESCRIPTION	History of computing, culminating with the definition of the discipline and the role of today's computing professional.
TOPICS	Minimum number of hours: 1 1. Early roots of computing science 2. Evolution of computing science as a discipline in its own right 3. The role of the professional
PREREQUISITES	None
REQUISITE FOR	None

SUBJECT NAME	SOCIAL, ETHICAL, AND PROFESSIONAL ISSUES
KNOWLEDGE UNIT TAG AND NAME	SP2: INTELLECTUAL PROPERTY
DESCRIPTION	Introduction to the concept of software and algorithms as intellectual property, and means of legally protecting that property.
TOPICS	Minimum number of hours: 1 1. Definition of intellectual property 2. Copyright and patent protection 3. Protection afforded to trade secrets
PREREQUISITES	None
REQUISITE FOR	None

SUBJECT NAME	SOCIAL, ETHICAL, AND PROFESSIONAL ISSUES
KNOWLEDGE UNIT TAG AND NAME	SP3: SOFTWARE PROTECTION AND SECURITY
DESCRIPTION	Survey of design methodologies to reduce the incidence of software theft and misuse.
TOPICS	Minimum number of hours: 1 1. Software ownership and license 2. Software copy protection and encryption 3. Hardware keys
PREREQUISITES	None
REQUISITE FOR	None

SUBJECT NAME	SOCIAL, ETHICAL, AND PROFESSIONAL ISSUES
KNOWLEDGE UNIT TAG AND NAME	SP4: SYSTEM SECURITY
DESCRIPTION	Survey of measures to protect computer systems and databases from unauthorized use and tampering.
TOPICS	Minimum number of hours: 1 1. Access security, passwords 2. Physical security 3. Virus detection and prevention
PREREQUISITES	None
REQUISITE FOR	None

SUBJECT NAME	SOCIAL, ETHICAL, AND PROFESSIONAL ISSUES
KNOWLEDGE UNIT TAG AND NAME	SP5: SOCIAL RESPONSIBILITY OF PROFESSIONALS
DESCRIPTION	The computing specialist as a professional with responsibilities to society.
TOPICS	Minimum number of hours: 1 1. Legal and social consequences of errors in software 2. Ethics 3. Licensing of computing professionals
PREREQUISITES	None
REQUISITE FOR	None

SUBJECT NAME	SOCIAL, ETHICAL, AND PROFESSIONAL ISSUES
KNOWLEDGE UNIT TAG AND NAME	SP6: DATA COLLECTION AND PRIVACY
DESCRIPTION	An introduction to the problems surrounding fair use of data, the right of individuals to privacy, and reporting requirements.
TOPICS	Minimum number of hours: 1 1. Requirements for protecting the privacy of individuals 2. Data reporting under right-to-know laws 3. Record matching and its implications
PREREQUISITES	None
REQUISITE FOR	None

SUBJECT NAME	SOCIAL, ETHICAL, AND PROFESSIONAL ISSUES
KNOWLEDGE UNIT TAG AND NAME	SP7: ARTIFICIAL INTELLIGENCE AND SOCIETY
DESCRIPTION	The philosophical and practical implications of artificial intelligence.
TOPICS	Minimum number of hours: 1 <ol style="list-style-type: none"> 1. Human mistrust of machines 2. Human reliance on machines 3. Worker displacement by intelligent machines
PREREQUISITES	AI2
REQUISITE FOR	None

SUBJECT NAME	SOCIAL, ETHICAL, AND PROFESSIONAL ISSUES
KNOWLEDGE UNIT TAG AND NAME	SP8: RISKS IN LARGE SYSTEMS
DESCRIPTION	Survey of issues that accompany the development of large software systems.
TOPICS	Minimum number of hours: 1 <ol style="list-style-type: none"> 1. Design and implementation teams 2. Software validation and testing 3. Risk assessment
PREREQUISITES	SE6
REQUISITE FOR	None

SUBJECT NAME	SOFTWARE METHODOLOGY AND ENGINEERING
KNOWLEDGE UNIT TAG AND NAME	SE1: FUNDAMENTAL PROBLEM-SOLVING CONCEPTS
DESCRIPTION	Introduction to methods of problem analysis and decomposition, leading to development of algorithms.
TOPICS	<p>Minimum number of hours: 12</p> <ol style="list-style-type: none"> 1. Top-down design, stepwise refinement 2. Use of sequence, selection, and repetition control structures 3. Standard data types and their uses 4. Procedural abstraction, parameters
PREREQUISITES	AL2
REQUISITE FOR	SE2

SUBJECT NAME	SOFTWARE METHODOLOGY AND ENGINEERING
KNOWLEDGE UNIT TAG AND NAME	SE2: SOFTWARE DEVELOPMENT MODELS
DESCRIPTION	Introduction to models used in developing software, and related issues.
TOPICS	<p>Minimum number of hours: 3</p> <ol style="list-style-type: none"> 1. Software life-cycle models 2. Software design objectives 3. Documentation and support
PREREQUISITES	SE1
REQUISITE FOR	SE4

SUBJECT NAME	SOFTWARE METHODOLOGY AND ENGINEERING
KNOWLEDGE UNIT TAG AND NAME	SE3: SOFTWARE REQUIREMENTS AND SPECIFICATIONS
DESCRIPTION	An introduction to writing specifications that define software systems.
TOPICS	<p>Minimum number of hours: 2</p> <ol style="list-style-type: none"> 1. Requirements analysis (e.g. systems performance requirements) 2. Formal and informal specifications 3. Formal specifications of abstract data types (ADT)
PREREQUISITES	AL4, AL7
REQUISITE FOR	SE4

SUBJECT NAME	SOFTWARE METHODOLOGY AND ENGINEERING
KNOWLEDGE UNIT TAG AND NAME	SE4: SOFTWARE DESIGN AND DEVELOPMENT
DESCRIPTION	Methodologies and tools used in designing and implementing software systems.
TOPICS	<p>Minimum number of hours: 3</p> <ol style="list-style-type: none"> 1. Software design objectives and strategies for meeting them 2. Software reliability 3. Tools for design, prototype and system development 4. Documentation
PREREQUISITES	SE2, SE3
REQUISITE FOR	SE5

SUBJECT NAME	SOFTWARE METHODOLOGY AND ENGINEERING
KNOWLEDGE UNIT TAG AND NAME	SE5: VERIFICATION AND VALIDATION
DESCRIPTION	Methodologies to assure production of quality software.
TOPICS	Minimum number of hours: 2 <ol style="list-style-type: none"> 1. Pre-conditions and post-conditions 2. Invariants 3. Testing strategies and analysis of results
PREREQUISITES	SE4
REQUISITE FOR	SE6

SUBJECT NAME	SOFTWARE METHODOLOGY AND ENGINEERING
KNOWLEDGE UNIT TAG AND NAME	SE6: SOFTWARE DESIGN AND IMPLEMENTATION
DESCRIPTION	Designing and implementing large software systems.
TOPICS	Minimum number of hours: 7 <ol style="list-style-type: none"> 1. Design strategies for large systems 2. Top-down and bottom-up implementation strategies 3. Managing programming teams 4. Testing, debugging, and antidebugging strategies 5. Performance measurement, profiling
PREREQUISITES	SE5
REQUISITE FOR	SP8

PART III

CURRICULA IMPLEMENTATION SAMPLES

1.0 CURRICULUM OVERVIEW

There are two sample curricula for the computing sciences presented in this report. One is designed to prepare students for entry-level positions in the computing sciences after two years of study. The other is designed to prepare students to transfer to a baccalaureate degree program in the computing sciences. Sample courses to support the transfer curricula and the career curricula are also included. Part II presented a detailed listing of the knowledge units which individual colleges can use to develop courses and curricula in the computing sciences. Recognizing that such courses and curricula should reflect the character and mission of each institution, the implementations in this report are to be viewed as examples of how to assemble the knowledge units into courses and curricula.

Section 1 includes a description of the structure and components of courses and curricula along with a listing of the courses that support the computing sciences curricula. Section 2 discusses underlying themes that pervade the computing sciences discipline. Sample associate-level curricula for the computing sciences are detailed in Section 3. Section 4 contains descriptions of sample courses to support each curriculum.

1.1 Course Structure

The subject area and knowledge units, detailed in Part II, specify the scope of topics in the computing sciences that all students should study. The lecture hours associated with each knowledge unit give an approximate indication of the minimum amount of time needed to cover the topics in the knowledge unit. This does not include additional time in the closed laboratory setting that can be devoted to a given topic.

The knowledge units in one subject area do not have to be grouped into one course or a series of courses. Each knowledge unit may be covered in one course or distributed among several courses with proportional time spans. Some knowledge units may be covered either in a standard class setting or in a laboratory setting. However, all *required* (non-PR) knowledge units of the computing sciences discipline must be included in the program of study. Thus the knowledge units

can be combined in various ways to form courses. For this activity, the following guidelines are advised:

- Knowledge units should be combined so that the composite subject matter of a course forms a coherent body of topics.
- The combined set of courses that comprise the discipline should have a prerequisite structure that is consistent with the prerequisite structure among the constituent knowledge units.
- The combined set of courses must cover all of the *required* knowledge units that make up the curriculum.

The following components are included in course descriptions: (See Figure III-1 for an example.)

Course Title

A clear but brief name of the course with course code and number is provided.

Number of Semester Hours

The total number of semester credits and hours are listed in the following format:

Lecture(number of credits : number of hours)

Laboratory (number of credits : number of hours)

Prerequisites

Courses are listed which must be completed prior to taking the course being described.

Goal or Purpose of the Course

This part contains a short paragraph which expresses what the course should accomplish.

Behavioral Objectives for Students

Competencies expected from students who successfully complete the course are detailed.

Subject Matter

A table is constructed with the following four columns:

- (a) Knowledge Unit Tag (listed alphabetically)
- (b) Ratio of the hours of the knowledge unit covered in this course to the total minimum hours for the knowledge unit.
- (c) Depth of knowledge expected to be achieved for the knowledge unit within this course.
- (d) Emphasis of knowledge for the given knowledge unit, which is expressed in terms of the three paradigms of theory, analysis, and design.

CS 102 Computing Fundamentals II Lecture (3:3)

Laboratory (1:3)

Prerequisite:

CS 101 Computing Fundamentals I

Goal:

This course develops in depth the discipline of computing and the roles of professionals. It is a course in software methodology, analysis of algorithms and data structures. Appropriate use of methodologies and choice of data structures are treated.

Objectives:

Upon successful completion of the course, the student will be able to:

- Apply appropriate software design methodologies for larger programs.
- Use appropriate data structures and data abstraction.
- Use algorithmic analysis for searching, sorting, and data structure operations.
- Differentiate between different file organizations and access methods.
- Use an object-oriented language.
- Understand the social responsibilities of the computing professional and be aware of the impact of computing on society.

Subject Matter:

<u>KU Tag</u>	<u>Portion of Hours</u>	<u>Depth</u>	<u>Emphasis</u>
AL4	7/7	4	T,A,D
AL5	3/3	4	T,A,D
AL6	6/6	4	T,A,D
AL7	3/3	4	T,A,D
DB1	3/3	3	A,D
PL5	4/4	4	A,D
SP*	2/6	3	T
SE2	3/3	3	D
SE3	2/2	3	T
SE4	3/3	3	D
SE5	2/2	3	T,A,D

Note: SP* means 2 of SP1 through SP6

Laboratory Experiences:

Suggested laboratory experiences include: Comparison of a recursive and non-recursive implementation of a search algorithm, changing a program from one data structure implementation to another, comparison of ADT's, interaction with a database management system, refinement of user interface, command languages, language syntax and semantics, storage management, exception handling, object-oriented programming, developing introductory programs in a non-procedural

language, program modification originating from specification revision, and writing a simulation program using an appropriate data structure.

Figure III-1 Course Format Example

Laboratory/Exercise Component

Suggestions for open or closed laboratory experiences or exercises for the subject matter are given. The laboratory component is an essential experience and should be included in all required computing courses within a curriculum.

1.2 Emphasis of Knowledge

Within courses, each knowledge unit contains a listing of its emphasis. The emphasis is indicated by some combination of the three letters T, A, or D, which stand for the three paradigms of *Theory*, *Analysis* or *Design*, respectively. *Theory*, which is deeply rooted in mathematics, corresponds to the fundamental knowledge base of the particular unit that is generally supported by fact or accepted understanding. To develop a coherent, valid theory, one uses the mathematical concepts of definition, theorem, and proof. *Analysis*, or modeling, corresponds to the scientific method used in verifying or testing a theoretical phenomenon, or in discovering a certain principle in the subject.

With this paradigm, one forms a hypothesis, constructs a model and makes predictions, designs an experiment and collects data, and finally analyzes the results. *Design*, which is rooted in engineering, corresponds to building an entity that demonstrates the general theme of the knowledge unit. With this paradigm, one states the requirements, states the specifications, designs and implements the system, and then tests the results.

For example, consider the knowledge unit:

AL6: SORTING AND SEARCHING

For this knowledge unit, depending on the scope of the given course, the student might be expected to formally derive the *big-O* efficiency of the quicksort algorithm (theory emphasis), determine the efficiencies of different sorting algorithms in a laboratory setting (analysis emphasis), and/or implement a binary search algorithm by writing a program (design emphasis).

1.3 Depth of Knowledge

Within courses, the knowledge units contain a listing of the expected depth of knowledge or understanding of the knowledge unit. This *depth indicator* is identified by an integer in the range 1 through 5. The interpretation of these integers is:

- 1 Awareness as expressed by general knowledge, definition, or recognition.

- 2 Description as evidence of conceptual understanding.
- 3 Differentiation as expressed by the ability to compare/contrast or to make connections with related topics.
- 4 Application as evidence of using predefined principles, methods, or tools in a structured environment.
- 5 Judgement as evidence of innovative decision making based upon analysis, synthesis or evaluation.

The depth of knowledge or understanding, as described above, is considered minimal for each knowledge unit within its course setting. A depth of 5 for a knowledge unit will be rare in the first two years of study. The depth concept is based in part on the taxonomy developed by the Software Engineering Institute [11].

1.4 Curriculum Structure

The following components are included in the description of each curricula:

Curriculum Title and Degree

A specification is provided of the degree or certificate which will be granted when all requirements are fulfilled. An area of specialization for the curriculum is provided which depicts the intended purpose of the program.

Purpose

A restatement of the purpose for each curriculum is provided, followed by the purpose of each option (if any) within the curriculum.

Curriculum Course Content

This section includes the list of courses that are required in the curriculum, including the different options (when applicable), possible electives, courses from other disciplines, and general education requirements.

Subject Matter Content

This section contains the subject area knowledge units required in each curriculum and for each option in the curriculum, if any.

Course Sequence

This section contains a diagram which shows the anticipated sequence of courses which are

Exit Competencies

Students completing a curriculum in the computing sciences are expected to achieve a certain level of competency in the knowledge units that make up that curriculum. The index used to indicate the depth of achievement within a curriculum is the depth of knowledge indicator discussed in Section 1.3. That is, the knowledge units within a curriculum have not only a measure of the depth of understanding within a course of a curriculum, but also an overall listing of the depth of understanding of the knowledge unit upon the completion of the curriculum. The exit competencies upon completion of a curriculum can differ from the expected achievement in an individual course depending on whether or not the knowledge unit is reinforced in later studies.

1.5 Curriculum Courses

Four computing courses are required for the career and the transfer computer science curricula. They are listed below as CS 101, CS 102, CS 201, and CS 202. The prerequisite to the first course (CS 101) is a working knowledge of the material in the subject area *Introduction to Programming* (PR). For completeness, a sample description is included of the prerequisite course, called CS 100 - Introduction to Programming, which is constructed from the knowledge units within this subject area. A brief description of these five courses is included in this section. Complete course descriptions are contained in Section 4.

CS 100 Introduction to Programming

This course is an introduction to programming and problem solving using a block-structured, procedural programming language.

CS 101 Computing Fundamentals I

This course is an introduction to the discipline, integrating algorithmic design, abstraction, and aspects of programming.

CS 102 Computing Fundamentals II

This is a course in data structures, software methodologies, analysis of algorithms and an introduction to the use of an object-oriented language.

CS 201 Machine Realization of Algorithms

This course emphasizes the organization and operation of computer systems at the assembly language level. Numerical computational methods are also examined.

CS 202 Algorithms and Paradigms in Computing

The course applies software engineering principles to the development of a large system and examines the different programming paradigms, parallel processing, and an introduction to artificial

The career computer science curriculum requires two additional courses in computing. The local institution should decide which courses are most appropriate, depending on the needs of the local industries, the expertise of the faculty, and the availability of specialized equipment and software. Some of these courses may be offered in other areas of the college, such as computing and engineering technology or computing for information processing. A brief description of sample career-oriented courses, which are currently being offered at some two-year colleges, is included in this section. More complete descriptions are included in Section 4.

CS 203 Database Management Systems

This course examines the physical and logical storage of data in database systems, and the use of a relational database system.

CS 204 Data Communications

This course studies the various hardware and software systems used to implement and maintain data communications systems.

CS 205 Artificial Intelligence and Robotics

This course is an introduction to the concepts and techniques of artificial intelligence, with application areas of learning, expert systems development, robotics, and vision.

CS 206 Operating Systems

This course covers the principles of a software interface between a computer and its users. The theory, design, and utilization of operating systems are detailed.

CS 207 Computer Graphics

This course presents the principles and methodologies of computer graphics, including the representation, manipulation, and display of two and three dimensional objects.

CS 208 Digital Electronics

This course involves a detailed introduction to digital systems, including the design of circuits using adders, flip-flops, registers and counters, and TTL devices.

CS 209 Numerical Computing

This course examines the standard techniques of numerical computing for approximating solutions to problems in mathematics, science and engineering.

CS 210 Computer Networks

This course deals with long-haul and local-area networks, including communication system components, communications software, network control, and carrier issues.

CS 211 Computer-Aided Design

This course uses a commercially-available software package for studying two and three dimensional graphics techniques and applications.

CS 212 Programming in Another Language

This course is an intensified coverage, with applications, of a programming language which is not a part of the required sequence. Examples include C, C++, and Prolog.

CS 213 Microcomputer Applications

This is a course in problem solving and programming with general-purpose application software tools such as spreadsheets, databases, word processors, and graphics software.

CS 214 Object-Oriented Systems

This course covers the object-oriented programming paradigm, including objects, messages, encapsulation, classes, inheritance, and implementation issues.

2.0 UNDERLYING THEMES

Underlying themes refer to basic themes that span the curriculum and tie individual knowledge units together. Since underlying themes span all subject areas, they serve to unify the discipline. Furthermore, given a reference to an understood concept, students can assimilate new ideas more easily. The ability to generalize and transfer concepts is important not only when moving into a new subject area in a course, but also when embarking on new areas of professional activity.

The underlying themes for the computing sciences, grouped by their emphasis of knowledge, are:

Underlying Themes -- All Paradigms:

Completeness: Completeness is characterized by containing all essential elements; it includes the related concept of robustness.

Examples: Sufficiency of an abstract data type to represent all values of the type; of an algorithm to anticipate all potential circumstances.

Consistency: Consistency is expressed by agreement between elements; it includes the related concepts of repeatability and predictability.

Examples: Agreement of axioms in a theory; agreement between observed and predicted behavior of an algorithm; harmony within the definition of an abstract data type.

Correctness: Correctness deals with producing desired results; it includes the related concept of reliability.

Examples: Correctness of an axiomatic system as shown by a rigorous proof; ability of a sorting algorithm to arrange elements in order regardless of original order; convergence of an iterative technique such as Newton's method.

Underlying Themes -- Theory:

Complexity of Large Problems: This concept considers the effects of nonlinear relationships between complexity and problem size.

Examples: Nonlinear increase in time to sort as size of list increases.

Formal Models: The establishment of formal models to involve ways of representing or thinking about ideas and problems based on established mathematical and scientific principles.

Examples: Formal models in logic; formal models of programming languages.

Underlying Themes -- Analysis:

Binding: Binding is defined as associating properties with an abstraction.

Examples: Assigning attributes to symbols during program translation; assigning memory addresses to objects at load-time.

Conceptual Models: Building conceptual models involves ways of representing or thinking about ideas and problems, based on insights and accepted practices.

Examples: Abstract data types; semantic models of programming languages.

Levels of Abstraction: Levels of abstraction refers to thinking about or representing ideas and problems with layers of description, each with a different degree of specification and detail.

Examples: Problem decomposition; memory hierarchies (storage devices, main memory, caches, registers).

Spatial Properties: Spatial properties are relationships between entities based on their physical or logical locations.

Examples: Storage structures; organization of language statements; scope of identifiers.

Temporal Properties: Properties that consider time in the ordering of events are temporal properties.

Examples: Order of steps in an algorithm and the sequential execution of instructions; time as a measure of efficiency.

Underlying Themes -- Design:

Analysis of Alternatives: Analysis of alternatives considers the consequences of selecting one alternative over others; it includes technical, economic, societal, and other concerns that influence or constrain the selection.

Examples: Time-space requirements of algorithms; conflicts in design objectives (cost vs. reliability, portability vs. performance, ease of use vs. security).

Efficiency: Efficiency is the cost to produce a desired effect measured in terms of space, time, effort, and other resources.

Examples: Time to search or sort; hit rate of hashing functions; effective use of memory in implementing data structures.

Reuse: This concept involves the use of previously developed methods, concepts, and components in new situations and contexts.

Examples: Portability of software libraries; inheritance of objects.

Underlying Themes -- Other:

Historical Perspective: A historical overview of computing evaluates the impact of technological developments on the computing environment; it is more than a simple recital of significant events in computing's history.

Examples: Development of early operating systems to improve system utilization; software cost and reliability issues that lead to adoption of structured techniques and now encourage object-oriented technologies.

Social Implications: The social implications of computing consider the effects of widespread use of computing on society, and how society's concerns have shaped development of computing science.

Examples: Employment in computing; effects of computing on employment in other fields; legal issues; privacy concerns; reliability of major systems.

3.0 SAMPLE CURRICULA

This section contains a sample curriculum for the career computer science students, and a sample curriculum for the transfer computer science students. The curricula are presented in the format discussed in Section 1.4.

3.1 Knowledge Unit Prerequisite Structure

Before designing a curriculum for the computing sciences, it is important to examine the prerequisite structure of the knowledge units which make up the curriculum. Figure III-2 displays a graph of the prerequisite structure for the knowledge units. Knowledge units PR1 through PR4 and SP1 through SP6 are not included in the graph. PL4 is a prerequisite for both AL7 and PL7; this is indicated with arrows to avoid overlapping lines. A knowledge unit in Figure III-2 is a prerequisite of another if there is a direct downward path from the former to the latter.

Figure III-2 Knowledge Unit Prerequisite Structure

3.2 Knowledge Unit Lecture Hours

Before designing a curriculum for the computing sciences, it is also important to consider the total minimum number of lecture hours to be devoted to each subject area making up the discipline. As Table III-1 shows, the subject area of *Introduction to Programming* contains a minimum of 25 lecture hours. Hence this subject area may be covered in a 2 credit-hour course without a laboratory component, or a 3 credit-hour course, if a laboratory component is included. The remaining nine subject areas require a total of 152 lecture hours. These knowledge units may be integrated into four 4 credit-hour laboratory-based courses.

Subject Areas	Lecture Hours
---------------	---------------

Introduction to Programming (PR)	25
----------------------------------	----

Algorithms & Data Structures (AL)	41
Architecture (AR)	23
Artificial Intelligence & Robotics (AI)	6
Database & Information Retrieval (DB)	3
Numerical & Symbolic Computation (NU)	4
Operating Systems (OS)	5
Programming Languages (PL)	33
Social, Ethical, & Professional Issues (SP)	8
Software Methodology & Engineering (SE)	29

Total Lecture Hours for Subject Areas (Except PR)	152
---	-----

Table III-1 Subject Area Content by Lecture Hours

Sample curricula for the computing sciences are detailed in the following two sections. Descriptions of courses that support these curricula are listed in Section 4. The structure of a curriculum, and the courses that support the curriculum, should reflect the mission of the program at each institution and the needs of the constituency. This is an important feature in designing a curriculum and is necessary to enable each institution and program to preserve its own identity.

3.3 A Sample Curriculum for Career Students

Curriculum Title and Degree

Degree: Associate

Area of Specialization: Computer Science -- Career Emphasis

Purpose

As stated in Part I of this report, the purpose of this program is to prepare students for a career in the computing sciences upon completion of the associate-level degree program. Career-track students, including people currently employed in the computing field, are best served by a core curriculum in computer science, with appropriate career-enhancement courses to provide greater depth in one or more application areas of computing. The nature of such elective courses depends, in great measure, on the needs of local industries. Students will develop a reasonable level of understanding in the various subject areas that define the discipline, as well as develop an appreciation for the interrelationships among them. With the additional electives, students should be prepared to apply their knowledge to specific problems and produce appropriate solutions.

Curriculum Course Content

Required Computer Science Courses:

- CS 101 Computing Fundamentals I
- CS 102 Computing Fundamentals II
- CS 201 Machine Realization of Algorithms
- CS 202 Algorithms and Paradigms in Computing

Required Computer Science Electives:

Each student must complete a minimum of two additional computing courses. Course offerings should be contingent upon the needs of local business and industry. Elective courses include, but are not limited to, the following:

- CS 203 Database Management Systems
- CS 204 Data Communications
- CS 205 Artificial Intelligence and Robotics
- CS 206 Operating Systems
- CS 207 Computer Graphics
- CS 208 Digital Electronics
- CS 209 Numerical Computing
- CS 210 Computer Networks
- CS 211 Computer-Aided Design
- CS 212 Programming in Another Language
- CS 213 Microcomputer Applications

CS 214 Object-Oriented Systems

Required Mathematics Courses:

Calculus I

Calculus II

Discrete Mathematics

Required Science Courses:

A one-year sequence of majors-level science courses (including laboratories), such as calculus-based physics I and II or technical physics I and II.

General Education Courses:

English I and II

Humanities and Social Science, as required by the local institution

(Technical Writing is suggested)

Table III-2 shows the general structure of a sample career computer science curriculum. The elective courses could consist of general education courses or additional technical electives.

Subject Matter Content

The subject matter content is summarized in Table III-1, which lists the minimum number of lecture hours for each of the subject areas making up the curriculum.

Course Sequence

Figure III-3 shows the sequencing of the required courses for the career and the transfer computer science curricula. The two additional required computing courses for the career computer science program are not included in the figure. Table III-3 lists some alternate semester sequences depending on whether or not CS 100 (Introduction to Programming), or its equivalent, is completed prior to commencing either computer science curriculum.

Exit Competencies

A listing of the exit competencies for each of the *required* knowledge units of the computing sciences curricula is included in Table III-7.

Semester 1	Cr redits		Semester 2	Cre dits
CS 101	4		CS 102	4
English 1	3		English 2	3
Mathematics 1	3-4		Mathematics 2	3-4
Social Science 1	3		Social Science 2	3
Elective 1	3		Elective 2	3
Total Credits	16- 17		Total Credits	16- 17
Semester 3	Cr redits		Semester 4	Cred its
CS 201	4		CS 202	4
Science 1	4		Science 2	4
Mathematics 3	3-4		Humanities	3
CS Elective 1	4		CS Elective 2	4
Total Credits	15- 16		Total Credits	15

Table III-2 Course Scheduling Sequence - Career Program

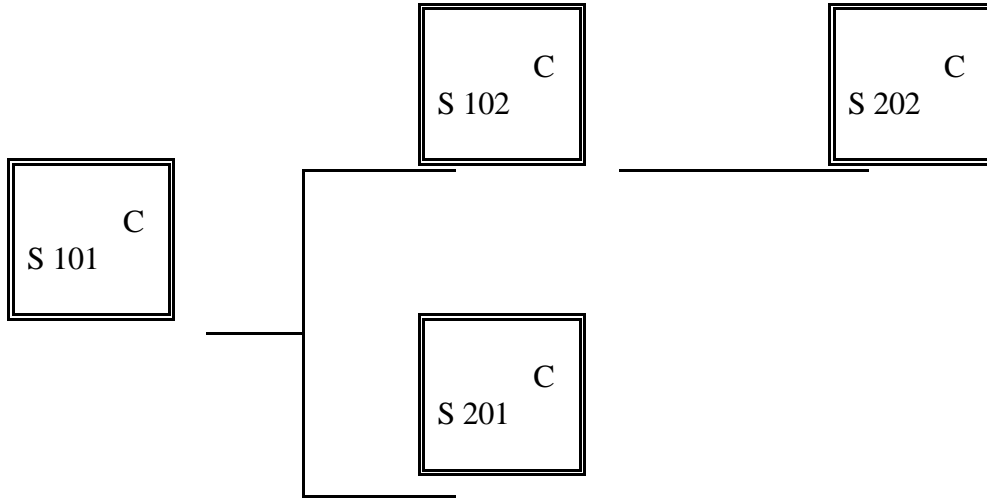


Figure III-3 Sequence of Required Courses - Transfer and Career

Background	Fall-1	Spring-1	Fall-2	Spring-2	Fall-3
CS 100	CS 101	CS 102	CS 201	CS 202	-
<u>No</u> CS 100	CS 100	CS 101	CS 102	CS 202	CS 201
<u>No</u> CS 100	CS 100	CS 101	CS 102	CS 202	-
					CS 201

Table III-3 Background-Dependent Sequencing Options - Transfer and Career

3.4 A Sample Curriculum for Transfer Students

Curriculum Title and Degree

Degree: Associate

Area of Specialization: Computer Science -- Transfer Emphasis

Purpose

As stated in Part I of this report, the purpose of this program is to prepare students for transfer with full junior-level status to a baccalaureate degree program in computer science.

Curriculum Content

Required Computer Science Courses:

CS 101 Computing Fundamentals I
CS 102 Computing Fundamentals II
CS 201 Machine Realization of Algorithms
CS 202 Algorithms and Paradigms in Computing

Required Mathematics Courses:

Calculus I
Calculus II
Discrete Mathematics

Required Science Courses:

A one-year sequence of majors-level science courses (including laboratories), such as calculus-based physics I and II.

Additional Recommended Courses:

Additional courses should be chosen contingent upon the requirements of the transferring institution. These recommended courses are intended to strengthen the student's academic preparation in mathematics and science. Possible courses include, but are not limited to, the following:

- Calculus III
- Linear Algebra
- Probability and Statistics
- Digital Electronics

General Education Courses:

English I and II
Humanities

Social Science

others courses as required by the local institution

Table III-4 shows the general structure of a sample transfer computer science curriculum. The elective courses could consist of general education courses or additional technical or supportive electives, depending in part upon the requirements of the transfer institution.

Subject Matter Content

The subject matter content is summarized in Table III-1, which lists the minimum number of lecture hours for each of the subject areas making up the curriculum.

Course Sequence

Figure III-3 shows the sequencing of the required courses for the career and the transfer computer science curricula. Table III-3 lists some alternate semester sequences depending on whether or not CS 100 (Introduction to Programming), or its equivalent, is completed prior to commencing either computer science curriculum.

Exit Competencies

A listing of the exit competencies for each of the *required* knowledge units of the computing sciences curricula is included in Table III-7.

Semester 1	Cr edits		Semester 2	Cre dits
CS 101	4		CS 102	4
English 1	3		English 2	3
Mathematics 1	3-4		Mathematics 2	3-4
Social Science 1	3		Social Science 2	3
Elective 1	3		Elective 2	3
Total Credits	16- 17		Total Credits	16- 17
Semester 3	Cr edits		Semester 4	Cred its
CS 201	4		CS 202	4
Science 1	4		Science 2	4
Mathematics 3	3-4		Humanities	3
Elective 3	3-4		Elective 4	3-4
Total Credits	14- 16		Total Credits	14- 15

Table III-4 Course Scheduling Sequence - Transfer Program

4.0 DESCRIPTIONS OF SAMPLE COURSES

This section contains descriptions of the courses which support the career and the transfer computing sciences curricula. The descriptions of the prerequisite course, CS 100, and the required courses CS 101, CS 102, CS 201, and CS 202 follow the format listed in Section 1.1. The descriptions of the elective computing courses CS 203 through CS 214 do not include knowledge units or suggestions for laboratory activities.

4.2 Courses Required for Career and Transfer Students

This section contains sample descriptions of the four required computer science courses, CS 101, CS 102, CS 201, and CS 202. The courses were mapped from the *required* knowledge units which were described in Part II of this report.

Following the format given in Section 1, each course description includes the prerequisite knowledge required of the student, a description of the course, course objectives, subject matter in terms of knowledge units, and suggested laboratory activities. The laboratory setting would be an excellent setting to discuss and experiment with the characteristics of the language used in the course. For example, the laboratory activities in the first half of CS 101 could be used to reinforce and expand the student's proficiency in a block-structured, contemporary programming language. Such topics would include fundamental types, I/O routines, control structures, and subprograms. The laboratory activities in the second half of CS 101 could be more focused on analysis and design and less on language syntax, with an emphasis on problem solving techniques.

Table III-5, which follows the descriptions of the four required courses, contains a listing of how the lecture hours of the *required* knowledge units are distributed within the four sample courses. Table III-6 contains the percentages of the knowledge units of the *required* subject areas distributed within the four courses and within the curriculum as a whole. It is important to note in Table III-6 that the total percent of lecture material over the four courses which is devoted to the subject area of *Programming Languages* (PL) is only 22%, while the total percent of the subject area *Algorithms and Data Structures* (AL) is 27%.

CS 101 Computing Fundamentals I Lecture (3:3)
 Laboratory (1:3)

Prerequisites:

CS 100 Introduction to Programming (or equivalent experience)
 Precalculus mathematics

Goal:

This course provides the essential foundation for a program of study in computer science. It introduces the discipline of computing and the roles of professionals. It integrates an introduction to algorithm design, an understanding of abstraction applied to data types and structures, and an appreciation of a procedural programming language as means of describing algorithms and data structures. The course introduces searching and sorting algorithms, and object-oriented programming.

Objectives:

Upon successful completion of this course, the student will be able to:

- Solve problems and develop algorithms using the control structure abstractions of sequence, selection, and repetition, following a disciplined approach.
- Use procedural abstraction, and top-down algorithm design and step-wise refinement methods.
- Use the basic data structures available in a procedural language, including specifications, operations, set of values, and representation.
- Use elementary algorithms for sorting and searching.
- Understand the social responsibilities of the computing professional and be aware of the impact of computing on society.
- Use an operating system, including its hierarchical system and utilities.
- Specify simple abstract data types and object types.

Subject Matter:

<u>KU Tag</u>	<u>Portion of Hours</u>	<u>Depth</u>	<u>Breadth</u>
AL1	2/2	2	T
AL2	8/8	4	A,D
AL3	3/3	4	A,D
PL1	1/1	1	T
PL2	2/2	4	D
PL3	4/4	4	A,D
PL4	4/4	3	A,D
SP1	1/1	3	T
SP2	1/1	3	T
SE1	12/12	4	A,D

Laboratory Experiences:

Syntax of a programming language, problem solving techniques, toolkits, design of user interfaces, operating systems purpose and use, hierarchical file systems, files and naming conventions, system configuration and management of executables, hierarchy of virtual computers as implementations of programming languages, searching and sorting algorithms, and tools to support software development, testing, and debugging.

CS 102 Computing Fundamentals II Lecture (3:3)

Laboratory (1:3)

Prerequisite:

CS 101 Computing Fundamentals I

Goal:

This course develops in depth the discipline of computing and the roles of professionals. It is a course in software methodology, analysis of algorithms and data structures. Appropriate use of methodologies and choice of data structures are treated.

Objectives:

Upon successful completion of the course, the student will be able to:

- Apply appropriate software design methodologies for larger programs.
- Use appropriate data structures and data abstraction.
- Use algorithmic analysis for searching, sorting, and data structure operations.
- Differentiate between different file organizations and access methods.
- Use an object-oriented language.
- Understand the social responsibilities of the computing professional and be aware of the impact of computing on society.

Subject Matter:

<u>KU Tag</u>	<u>Portion of Hours</u>	<u>Depth</u>	<u>Breadth</u>
AL4	7/7	4	T,A,D
AL5	3/3	4	T,A,D
AL6	6/6	4	T,A,D
AL7	3/3	4	T,A,D
DB1	3/3	3	A,D
PL5	4/4	4	A,D
SP3	1/1	3	T
SP4	1/1	3	T
SE2	3/3	3	D
SE3	2/2	3	T
SE4	3/3	3	D
SE5	2/2	3	T,A,D

Laboratory Experiences:

Suggested laboratory experiences include: Comparison of a recursive and non-recursive implementation of a search algorithm, changing a program from one data structure implementation to another, comparison of ADT's, interaction with a database management system, refinement of user interface, command languages, language syntax and semantics, storage management, exception handling, object-oriented programming, developing introductory programs in a non-procedural language, program modification originating from specification revision, and writing a simulation program using an appropriate data structure.

CS 201 Machine Realization of Algorithms Lecture (3:3)
 Laboratory (1:3)

Prerequisites:

CS 101 Computing Fundamentals I
 Calculus I

Goal:

This course emphasizes the organization and operation of real computer systems at the assembly-language level. The mapping of statements and constructs in a high-level language onto sequences of machine instructions is studied, as well as the internal representation of simple data types and structures. Numerical computation is examined, noting the various data representation errors and potential procedural errors.

Objectives:

Upon successful completion of the course, the student will be able to:

- Analyze computer system organization at the assembly language level.
- Implement high-level procedural language elements at the assembly language level.
- Describe the programming language translation process.
- Compare/contrast the limits of data representations and their impact on numerical computation.
- Describe the basic principles of operating systems.
- Use iterative approximation methods when solving mathematical problems.
- Understand the social responsibilities of the computing professional and be aware of the impact of computing on society.

Subject Matter:

<u>KU Tag</u>	<u>Portion of Hours</u>	<u>Depth</u>	<u>Breadth</u>
AR1	3/3	4	A,D
AR2	3/3	3	A,D
AR3	7/7	4	A,D
AR4	4/4	3	T,D
AR5	4/4	4	A,D
AR6	2/2	2	T
NU1	4/4	4	A,D
OS1	5/5	2	T
PL6	4/4	3	A,D
SP5	1/1	3	T
SP6	1/1	3	T

Laboratory Experiences:

Experience with assembly language programming, tools that support programming at assembly and machine levels, numerical computation at the level of Calculus I, selected components of a compiler or interpreter, screen formatting, and low-level I/O operations.

CS 202 Algorithms and Paradigms in Computing Lecture (3:3)
 Laboratory (1:3)

Prerequisite:

CS 102 Computing Fundamentals II

Goal:

This course continues the study of the design and analysis of algorithms, including concurrency and parallel processing. The major programming paradigms, logic, functional, and object-oriented, along with representative languages, are covered and contrasted with procedural programming. Search strategies and other artificial intelligence concepts are introduced from a problem-solving point of view. The course includes the design and implementation of a multi-faceted software system.

Objectives:

Upon successful completion of this course, the student will be able to:

- Apply software engineering principles, including specific methodologies currently advocated and used for software design, development, testing, and verification.
- Apply analysis and design criteria in the selection of algorithms and data structures to solve problems.
- Utilize the basic control and search strategies of artificial intelligence.
- Use alternate programming paradigms and languages along with an understanding of their advantages and disadvantages.
- Implement concurrent algorithms.
- Understand the social responsibilities of the computing professional and be aware of the impact of computing on society.

Subject Matter:

<u>KU Tag</u>	<u>Portion of Hours</u>	<u>Depth</u>	<u>Breadth</u>
AL8	5/5	3	T
AL9	4/4	3	T
AI1	3/3	3	D
AI2	3/3	4	A,D
PL7	10/10	4	T,A,D
PL8	4/4	4	A,D
SP7	1/1	3	T
SP8	1/1	3	T
SE6	7/7	5	A,D

Laboratory Experiences:

Timing experiments to verify asymptotic analysis, heuristic search algorithms, syntax & semantics of object-oriented programming language, syntax & semantics of logic programming language, syntax & semantics of functional programming language, use of CASE tools, and graphical interfaces.

Subject Areas	101	CS 102	CS 201	CS 202	CS Total			
Algorithms & Data Structures	13	19	0	9	41			
Architecture		0	0	23	0	23		
Artificial Intelligence & Robotics		0	0	0	6	6		
Database & Information Retrieval		0	3	0	0	3		
Numerical & Symbolic Computation	0	0	4	0	4			
Operating Systems		0	0	5	0	5		
Programming Languages		11	4	4	14	33		
Social, Ethical, & Professional Issues	2	2	2	2	8			
Software Methodology & Engineering		12	10	0	7	29		
-----							Total	by
Course	38	38	38	38	152			

Table III-5 Summary of Sample Courses by Lecture Hours

Subject Areas	101	CS 102	CS 201	CS 202	CS Total	%		
Algorithms & Data Structures	34%	50%	0%	24%	27%			
Architecture		0%	0%	60%	0%	15%		
Artificial Intelligence & Robotics		0%	0%	0%	16%	4%		
Database & Information Retrieval		0%	8%	0%	0%	2%		
Numerical & Symbolic Computation	0%	0%	11%	0%	3%			
Operating Systems		0%	0%	13%	0%	3%		
Programming Languages		29%	11%	11%	37%	22%		
Social, Ethical, & Professional Issues	5%	5%	5%	5%	5%			
Software Methodology & Engineering		32%	26%	0%	18%	19%		
-----							Total	by
Course	100%	100%	100%	100%	100%			

Table III-6 Percentages of Subject Areas in Sample Courses and Curricula

Tag	Name	Depth
AL1:	Introduction to Problem-Solving Strategies	4
AL2:	Structured Data Types	4
AL3:	Introduction to Sorting and Searching	4
AL4:	Data Structures	4
AL5:	Recursive Algorithms	3
AL6:	Sorting and Searching	4
AL7:	Objects	3
AL8:	Complexity	2
AL9:	Problem-Solving Strategies	4
AR1:	Machine Representation of Data	4
AR2:	Machine Organization	3
AR3:	Machine and Assembly Language Programming	4
AR4:	Memory System and Architecture	3
AR5:	Interfacing and Communication	3
AR6:	Alternative Architectures	2
AI1:	Overview of Applied Artificial Intelligence	2
AI2:	Search Strategies	3
DB1:	File and Physical Database Organization	3
NU1:	Numerical Computation	3
OS1:	Tasking and Processes	2
PL1:	History of Programming Languages	2
PL2:	Specification of Data Types	4
PL3:	Sequence Control	4
PL4:	Object-Oriented Programming	3
PL5:	Data Access	3
PL6:	Language Translation Systems	2
PL7:	Programming Paradigms	3
PL8:	Concurrency	3
SP1:	Evolution of Computing and the Professional	3
SP2:	Intellectual Property	3
SP3:	Software Protection and Security	3
SP4:	System Security	3
SP5:	Social Responsibility of Professionals	3
SP6:	Data Collection and Privacy	3
SP7:	Artificial Intelligence and Society	3
SP8:	Risks in Large Systems	3
SE1:	Fundamental Problem-Solving Concepts	4
SE2:	Software Development Models	3
SE3:	Software Requirements and Specifications	3
SE4:	Software Design and Development	3
SE5:	Verification and Validation	3
SE6:	Software Design and Implementation	4

Table III-7 Expected Exit Competencies

4.3 Elective Courses for Career Students

This section contains the descriptions of the elective courses CS 203 through CS 214 for the career computer science program. These courses could also be made available to transfer-track computer science students as well as suitably-qualified individuals in need of upgrading their computing skills. The course descriptions are samples of what is currently being taught for career-oriented students at some two-year colleges.

The course descriptions do not contain subject matter knowledge units since these courses are not part of the core material for the two-year computing science curricula. The descriptions also do not contain suggestions for laboratory activities.

All courses in this section are labeled with a CS prefix and have a 200-level number. However, some of these courses may not necessarily be taught at the sophomore level, and some courses are being taught by other departments within the college. For example, CS 211 (Computer-Aided Design) and CS 208 (Digital Electronics) are generally taught in an engineering technology department, CS 213 (Microcomputer Applications) is sometimes a freshman-level course taught in an information systems department, and CS 207 (Computer Graphics) may be taught cooperatively with an art department.

CS 203 Database Management Systems Lecture (3:3)

Laboratory (1:3)

Prerequisite:

CS 201 Machine Realization of Algorithms

Goal:

This course examines the physical and logical structure of data stored on physical devices, and the data structures and algorithms that support modern database systems. Case studies and applications of a relational database system are included. The design of the user interface supplements the lecture topics.

Objectives:

Upon successful completion of this course, the student will be able to:

- Identify and use the principle components of a database system.
 - Differentiate between the various database models.
 - Analyze the suitability of a storage device for a particular application.
 - Select among file organizations and data structures in implementing a database system.
 - Design and implement programs that search, merge, sort, and update files.
 - Use a relational database system, including the design of records and use of a query language.
-

CS 204 Data Communications Lecture (3:3)

Laboratory (1:3)

Prerequisite:

CS 201 Machine Realization of Algorithms

Goal:

This course introduces the student to the different means for the transmission of data to and from computers and components of computer systems.

Objectives:

Upon successful completion of the course, the student will be able to:

- Describe the various media used for data communications.
- Demonstrate an understanding of the telephone system and its characteristics as they apply to data communication.
- Describe and explain the use of analog signals for data transmission.
- Demonstrate an understanding of the various types of protocols and methods of error checking used in data communications.
- Demonstrate an understanding of the generation and use of digital signals for data communications.
- Describe and explain the use and application of optical methods for data communications.
- Demonstrate an understanding of the equipment, software and operation of local area networks.

CS 205 Artificial Intelligence and Robotics Lecture (3:3)
Laboratory (1:3)

Prerequisite:

CS 102 Computing Fundamentals II

Goal:

This course is a study of how computers may be programmed to behave in ways normally attributed to intelligence when observed in humans. Topics include the nature of intelligence, basic problem solving and search strategies, knowledge representation, expert system development tools, robotics and vision. Application areas include learning, expert systems development, robotics, and vision.

Objectives:

Upon successful completion of the course, the student will be able to:

- Understand the architecture for intelligent systems and how these systems represent their knowledge of the world.
 - Appreciate the uses and limitations of intelligent systems.
 - Use expert system development concepts with shells and logic programming.
 - Appreciate the uses and limitations of robotics.
 - Apply robotic concepts to an industrial environment.
 - Use robotic programming with vision systems.
-

CS 206 Operating Systems Lecture (3:3)
Laboratory (1:3)

Prerequisite:

CS 201 Machine Realization of Algorithms

Goal:

This course is a study of the concepts, designs, and operations of modern real-time, general-purpose operating systems.

Objectives:

Upon successful completion of the course, the student will be able to:

- Describe operating systems development.
- Differentiate among monolithic, layered and object-oriented structuring methods.
- Understand and use multi-tasking, pre-emptive scheduling and time sharing.
- Describe process coordination and synchronization.
- Apply process scheduling and dispatching.
- Describe physical and virtual memory organization.
- Use device management concepts.
- Use file systems and their naming conventions.
- Describe system security and protection.
- Work with distributed and real-time systems.
- Understand communications and networking concepts.

CS 207 Computer Graphics Lecture (3:3)

Laboratory (1:3)

Prerequisite:

CS 102 Computing Fundamentals II

Goal:

This course presents the hardware and software principles and methodologies of computer graphics, including raster graphics, three-dimensional graphics, graphics packages, and graphics systems. Topics include conceptual models, graphics standards, interactive graphics techniques, modeling, rendering, and the representation, manipulation, and display of two and three dimensional objects.

Objectives:

Upon successful completion of the course, the student will be able to:

- Apply the fundamental concepts of computer graphics.
 - Visualize two and three-dimensional data.
 - Use raster, vector, and color graphing techniques.
 - Understand and use key graphics standards.
 - Evaluate graphics terminals, plotters, and workstations.
 - Implement computer graphics into a variety of applications.
-

CS 208 Digital Electronics Lecture (3:3)

Laboratory (1:3)

Prerequisite:

CS 101 Computing Fundamentals I

Goal:

This course involves a detailed introduction to digital systems, including the design of circuits using adders, flip-flops, registers and counters, and TTL devices.

Objectives:

Upon successful completion of the course, the student will be able to:

- Understand and use the binary, octal, and hexadecimal numbering systems.
- Analyze combinational logic circuits in terms of function and timing.
- Express circuit function in terms of Boolean expressions.
- Design, implement and test combinational circuits.
- Analyze, implement, and test flip-flop and latch circuits.
- Analyze and troubleshoot digital circuits.

CS 209 Numerical Computing Lecture (3:3)

Laboratory (1:3)

Prerequisite:

CS 201 Machine Realization of Algorithms
Calculus II

Goal:

This course examines the standard techniques of numerical computing for approximating solutions to problems in mathematics, science and engineering. Topics include error analysis, approximating roots of equations, solving systems of equations, interpolating and approximating polynomials, numerical integration, and approximating solutions of elementary first-order differential equations.

Objectives:

Upon successful completion of the course, the student will be able to:

- Understand the standard algorithms of numerical computing and how to apply them.
 - Implement, in a scientific programming language, standard numerical analysis techniques for solving problems in mathematics, science, and engineering.
 - Understand the theoretical foundation behind the standard numerical analysis techniques.
 - Understand the need for error analysis, and analyze the error, when implementing standard numerical analysis techniques.
-

CS 210 Computer Networks Lecture (3:3)

Laboratory (1:3)

Prerequisite:

CS 201 Machine Realization of Algorithms

Goal:

This course deals with long-haul and local-area networks (LAN), including communication system components, communications software, network control, and carrier issues.

Objectives:

Upon successful completion of the course, the student will be able to:

- Describe principal LAN networking standards.
- Describe LAN topologies, protocols, transmission media, and access methods.
- Understand the use of routers, bridges, and gateways.
- Install and run network applications on a PC LAN.
- Describe PC LAN components.
- Describe the media, signals, functions, and steps necessary to achieve reliable transfer of data across point to point data links.
- Explain alternative data encoding schemes and clock synchronization schemes.

CS 211 Computer-Aided Design Lecture (3:3)

Laboratory (1:3)

Prerequisite:

CS 101 Computing Fundamentals I

Goal:

This course uses an available computer-aided design software package for studying two and three-dimensional graphics techniques and applications.

Objectives:

Upon successful completion of the course, the student will be able to:

- Demonstrate the use of a commercially-available computer-aided design package.
 - Appreciate the wide range of CAD applications available with today's packages.
 - Understand surface models and their applications.
 - Use 3-dimensional design techniques to solve mechanical design/drafting problems.
 - Describe the uses of figure files, mass property calculations, plotting, and sheet-metal unfolding.
 - Understand the mathematical elements for 3-dimensional computer graphics.
 - Describe the management of an industrial CAD system center.
-

CS 212 Programming in Another Language Lecture (3:3)

Laboratory (1:3)

Prerequisite:

CS 101 Computing Fundamentals I

Goal:

This course is an intensified coverage, with applications, of a programming language which is not a part of the required sequence. Examples include C, C++, and Prolog.

Objectives:

Upon successful completion of the course, the student will be able to:

- Understand the uses, syntax, and limitations of the covered language.
- Develop well-structured programs with the covered language to solve problems appropriate for the language.
- Properly document and test their programs so users and other programmers can understand the uses and the processing.
- Explain the similarities and differences in usage between the covered language and other languages studied by the student.

CS 213 Microcomputer Applications Lecture (3:3)

Laboratory (1:3)

Prerequisite:

Computer literacy (for example, COD 1 Computer Communications, which is described in the Computing for Other Disciplines Committee report)

Goal:

This is a course in problem solving and programming with general-purpose application software tools such as spreadsheets, databases, word processors, and graphics software.

Objectives:

Upon successful completion of the course, the student will be able to:

- Demonstrate an in-depth knowledge of integrated software in order to create, edit, revise, and print word processing, database and spreadsheet documents.
 - Identify, operate, and develop an understanding of the peripherals of computers.
 - Demonstrate the use of advanced features of software such as macros, creating forms, creating charts and tables, outlining, and other functions.
 - Format, revise, and integrate graphics in any of the documents.
 - Demonstrate decision making skills as they relate to computer business applications.
-

CS 214 Object-Oriented Systems Lecture (3:3)

Laboratory (1:3)

Prerequisite:

CS 102 Computing Fundamentals II

Goal:

This course covers the object-oriented programming paradigm, including objects, messages, encapsulation, classes, inheritance, and implementation issues. Implementations will be written in a current object-oriented language such as C++.

Objectives:

Upon successful completion of the course, the student will be able to:

- Understand the advantages of object-oriented design techniques, including encapsulation, abstraction, inheritance, and reusability.
- Analyze the system requirements and define the important objects.
- Classify the system objects into a hierarchy.
- Select the attributes of the objects at each level of the hierarchy.
- Determine the method designs for each object at each level.
- Implement the design in a given object-oriented language.