Curriculum and Course Syllabi for a High-School Program in Computer Science*

Judith Gal-Ezer† David Harel‡

Abstract

The authors served on a committee that designed a high-school curriculum in computer science and has been supervising the preparation of a comprehensive study program based on it. The new program is intended for the Israeli high-school system, has been formally approved by the Ministry of Education, and is expected to fully replace the old one in the near future. The program emphasizes the foundations of algorithmics, and teaches programming as a way to get the computer to carry out an algorithm. The purpose of this paper is to describe the program's curriculum and syllabi in detail.

Keywords: algorithmics, computer science, curriculum, education, high-school.

1 Introduction

An Israeli committee, on which we served as members, put together a new high-school program in computer science. The program's emphasis is on the basics of algorithmics, and although it teaches programming it does so only as a means for getting a computer to carry out an algorithm. In this paper we give a detailed description of the program's curriculum, module by module. To complement the present rather technical paper, it might be worth reading [GBHY], in which we discuss the background and motivation for the entire program, and provide some information about its implementation.

Since the publication of [GBHY], the program has been formally approved by the Israel Ministry of Education. In a sense, this brings one of the main tasks of our committee to an end. The

*This paper has two authors, but in a way this is only a formality. We would like to point the reader to the Acknowledements section for information regarding its many other actual authors. The added contribution of the two of us, beyond the extremely large amount of joint work, was essentially in taking upon ourselves the task of writing the material up for publication.

†Dept. of Mathematics and Computer Science, The Open University of Israel.
Email: galezer@cs.openu.ac.il,

‡Dept. of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel. Email: harel@wisdom.weizmann.ac.il

implementation of the program in Israeli high-schools has already started, and is now in the hands of the Ministry; the committee plays a supervisory role. The program is expected to fully replace existing programs in computer science in the near future.

Our 10-member committee, formed in 1990, included computer scientists who are also involved in various kinds of educational activities, experienced high-school teachers of computer science, and computer science education professionals from the Ministry of Education. In our work we were greatly aided by the members of the many course development teams. While the main role of these teams was to prepare the study material based on the curriculum, their work very often improved the relevant parts of the curriculum itself. In some cases, members of the development team participated in designing the curriculum itself too. We are extremely grateful to all these people, whose names are listed in the appropriate subsections.

Section 2 contains a brief description of the structure of the curriculum and three main programs of study based on it. The heart of the paper is the lengthy Section 3, which contains the detailed descriptions of the program modules themselves.

2 General Structure of the Program

The material of the program is divided into five modules, some of which have more than one alternative, as follows:[1]

1. **Fundamentals 1** and **2** (double-length; 180 hours): This module provides the foundation for the entire program. It is the one that introduces the central concepts of algorithmics, and in parallel teaches how algorithms are implemented in a procedural programming language.

2. **Software Design** (90 hours): This module is a continuation of Fundamentals. It concentrates on data structures, introducing abstract data-types in the process. It also goes beyond stand-alone algorithms by discussing the design of complete systems.

3. **Second Paradigm** (90 hours): This module introduces the student to a second programming paradigm, which is to be conceptually quite different from the procedural approach adopted in the previous modules. Currently approved alternatives are logic programming and assembly language programming; other future possibilities include object-oriented programming, functional programming, and concurrent programming.

4. **Applications** (90 hours): This module concentrates on some general kind of application area, and teaches both its principles and its practice. Currently approved alternatives are computer graphics and information systems.

5. **Theory** (90 hours): This module is intended to expose the student to selected topics in theoretical computer science. Currently approved alternatives are a full 90-hour unit on models of computation, and a two-part unit consisting of 45 hours on models of computation and 45 hours on numerical analysis.

There are three possible study programs that can be constructed from these modules: A low-level 90-hour one, an intermediate 270-hour one, and a full 450-hour one intended for scientifically-oriented students interested in continuing their studies in computer science or related fields.

[1] The information in this section constitutes an updated version of the program's structure, as described in [GBHY], reflecting the slight changes introduced since that publication.

The first program (called the "one-unit program") consists of the 90-hour Fundamentals 1 unit only. It provides but a brief "taste" of the material.

The second program (the "three-unit program") consists of Fundamentals 1 and 2, and an additional 90-hour unit, which can be taken from the Second Paradigm or Applications modules.

The third program (the "five-unit program") consists of Fundamentals 1 and 2, one 90-hour unit taken from the Second Paradigm or the Applications modules, Software Design and Theory.

We expect more of the advanced students taking the five-unit program (or their teachers) to prefer the Second Paradigm module over the Applications module. Moreover, the five-unit program entails a deeper treatment of the Fundamentals and Second Paradigm modules, including some topics that are not taught in the one- and three-unit programs of study.

The table below shows how the final exam for the entire program (called the "bagrut" in Israel) is broken up according to the combination of modules in the various study programs:

<table>
<tr><th>study units</th><th>written exam weight</th><th>lab exam weight</th><th>total weight in 3-unit program</th><th>total weight in 5-unit program</th></tr>
<tr><td>Fundamentals 1</td><td>75%</td><td>25%</td><td>30%</td><td rowspan="3">50%</td></tr>
<tr><td>Fundamentals 2</td><td>50%</td><td></td><td rowspan="2">70%</td></tr>
<tr><td>Second Paradigm or Applications</td><td>25%</td><td>25%</td></tr>
<tr><td>Software Design</td><td>40%</td><td>20%</td><td rowspan="2"></td><td rowspan="2">50%</td></tr>
<tr><td>Theory</td><td>40%</td><td></td></tr>
</table>

In [GBHY] we discussed some of the problematics of teaching the program and training the teachers. Partly as a result of our experience in this, we later wrote [GH], which describes a course intended for prospective computer science educators. In addition, the Israeli Ministry of Education supports an extensive in-service training program for teachers, now taking place under the auspices of Israeli universities. This training is often conducted by members of the course development teams.

We now turn to a detailed description of the modules, one by one.

3 Fundamentals 1

This unit is the basis of the entire program, and is taught in the 10th grade. It is also intended to be usable as a stand alone mini-course for students who will not study any more computer science, as explained earlier. It covers the basic concepts of an algorithmic problem and its solution — the algorithm. It also briefly discusses functions as a refinement mechanism, and introduces the notions of algorithmic correctness and efficiency.

This unit reflects a major pedagogical principle we have followed in the design of the entire program, whereby conceptual and experimental issues are interweaved.[2] In [GBHY], where this is explained in more detail, we dubbed it the "zipper approach": A little of this, followed by a little of that, and so on, combining to form a unified whole. Progress along the two tracks is made in parallel. Each new concept is first discussed in the classroom, and then, if needed, relevant parts of the programming language are introduced and the student gets to practice them in the computer laboratory.

We shall not discuss here the controversial issue of the first programming language — the "mother tongue". Many good arguments have been made for adopting any one of the many possible styles of programming in the first course. We became convinced that for high-school it is probably best to remain in the conservative mainstream, and we thus decided to adopt a "vanilla" procedural approach[3]. The implementation parts of the detailed program below are based on Pascal. It is important to realize, however, that the program itself does not impose a specific language, and the future might find teams developing courseware that uses other languages.

Background bibliography: [ClCo, Kof1, Sedg, WiJe]

Development team:

Work done at the Department of Science Teaching of the Weizmann Institute of Science, by David Ginat, Orna Lichtenstein, Ela Zur, Mordechai Ben-Ari, Esther Hershkovitz, Yifat Kolikant, Hannah Mahleb, and Nurit Reich.

Fundamentals 1

	conceptual	experimental
Ch. 1 – Introduction	5	–
Ch. 2 – A basic computational model	9	6
Ch. 3 – Introduction to algorithm development	3	–
Ch. 4 – Conditional execution	5	4
Ch. 5 – Correctness of algorithms	3	1
Ch. 6 – Iterative execution	10	5
Ch. 7 – Efficiency of algorithms	3	–
Ch. 8 – Subtasks: Functions	3	5
Ch. 9 – One-dimensional arrays	7	5
Ch. 10 – Control structures revisited	12	4
total hours	60	30

[2]By 'experimental' here we actually mean the implementational aspects of the material, and not only things having to do with experiments in the narrow sense of the word.

[3]Recall, however, that we have a Second Paradigm module, in which a new – totally different – programming paradigm, is studied.

Chapter 1: Introduction

Goals: Presenting the basic concepts of input/processing/output, program, compilation and execution. Providing an initial exposure to the unit's major concepts, the algorithm and the algorithmic problem. Demonstrating algorithms and "non-algorithms".

Contents: Algorithm, algorithmic problem, the process of execution (or run) induced by an algorithm and an input. Algorithms and non-algorithms demonstrated at the pseudocode level or informal language, including conditions and iterations. Also, very briefly: computers, hardware, software, operating system, programming language, compilation.

Chapter 2: A basic computational model

Goals: Becoming familiar with the computational model of input and output, variables and value assignment, and the notion of a simple sequential algorithm.

Contents: Data, variables, input, output; a simple algorithm as a sequence of instructions that cause changes in values of variables, the process of execution (or run), states during the execution; development of "flat" algorithms (identifying inputs and outputs, identifying and documenting variables, choosing types for variables). Emphasis is put on the choice of variable types, which is part of the development and documentation of the program.

Becoming acquainted with the programming environment, and working within it on the following: Variable and type definition, assignment statements, arithmetic expressions, input and output statements, structure of basic programs.

Examples include both informal algorithms (verbal or pseudo-code) and actual program segments. Program execution is demonstrated by trace tables. The concept of correctness is mentioned here, but is studied in detail only later, in Chapter 5.

Chapter 3: Introduction to algorithm development

Goals: Demonstrating the steps in the development of an algorithm (idea, problem decomposition, variables, pseudo-code, code). Justifying the need for control structures and more complex algorithms.

Contents: Decomposition of a problem into subproblems, mechanisms for constructing complex algorithms out of simpler parts, and understanding the connection between the subproblems and algorithm's parts. The fact that a simple (sequential) algorithm is not always sufficient is illustrated by examples.

Chapter 4: Conditional execution

Goals: Mastering the basic control structure for conditional execution, and designing algorithms that use it.

Contents: Condition and conditional execution; *and*'s and *or*'s, truth tables for *and* and *or*. Boolean conditions with order relations; the *if* statement (in both the *if-then* and *if-then-else* versions).

Emphasis is placed on the fact that conditioning and branching can be used to prevent some parts of the algorithm from being executed in a given run. Examples and exercises include conditions containing *and* and *or*, but not combinations thereof. The *not* operator is not taught (not that *not* is not important...), but students do practice the use of relations and their explicitly negated versions, such as < and =>. Nesting of operators is not taught either.

Chapter 5: Correctness of algorithms

Goals: Introducing the concept of correctness, which accompanies the material throughout the entire program. Understanding the difference between logical errors and errors resulting from an incorrect implementation of the algorithm in the programming language. Getting acquainted with the advantages and disadvantages of debugging by testing. Realizing the importance of good test-case selection, correct algorithmic development, and proper documentation.

Contents: The domain of legal inputs, correctness with respect to an algorithmic problem and its inputs, correct and incorrect algorithms (syntax errors, run-time errors and incorrect outputs). Debugging an algorithm by running it on test inputs and the limitations of this technique, documentation as an aid in achieving and checking correctness. Running and debugging given programs. Illustrating the importance of thoroughly constructed test-cases.

Chapter 6: Iterative execution

Goals: Mastering some control structures for iterative execution, and designing algorithms that use them. Using iteration for algorithms that require counting or value accumulation.

Contents: Iteration: Common cases (a single repeating instruction, an instruction that repeats on different data, iteration with accumulating activity), counters, accumulators, termination conditions, infinite iteration. The *while* statement, the Boolean condition as a sentinel (or guard).

Emphasis is placed on the fact that with iteration the algorithm's text remains unchanged but it can induce runs of different lengths (including length 0) depending on the input. This is linked to the issue of non-terminating iteration (i.e., runs of infinite length). The special significance of correctness in the presence of iteration is also emphasized (e.g., fulfilling the conditions of the iteration, and ensuring termination).

Note: The experimental part of this chapter focuses on the *while* statement. Examples are also given in pseudocode that does not directly correspond to a programming language construct, such as: "For each element in the set do ...", in which case practicing can be carried out without using a computer. Also, a simple iteration of the form "Repeat the following n times" (implemented by a simple form of *for*) can be presented, although an in-depth study of the *for* statement is not carried out here.

Chapter 7: Efficiency of algorithms

Goals: Introducing the concept of algorithmic efficiency, which will recur throughout the entire curriculum. Emphasizing the importance of counting the number of loop iterations.

Contents: Identification and enumeration of the dominant statements of the algorithm; execution time as a function of input size, comparisons of execution time, worst case estimates. The influence on execution time of the programming language, the compiler and the computer. Measurement and comparison of execution times of given programs and modifications thereof.

The teacher illustrates what the "important" or dominant instructions are for different types of problems. The big-O notation is not taught here.

A possible example to be discussed in the laboratory sessions is checking whether n is a prime number by iterating through the numbers $2, \ldots, n$ and then checking $2, \ldots \sqrt{n}$. (It is also possible to demonstrate executions of a probabilistic algorithm for this problem and the improvement it affords, but without getting into detailed explanations). Emphasis is placed on the difference between a program with a short text and an efficient program.

Chapter 8: Subtasks: Functions

Goals: Becoming acquainted with the concept of the function as a tool for solving problems by means of division into subproblems. Using functions in the programming language.

Contents: The use of functions for dividing a problem into subproblems, an algorithmic representation of the function as a solution to the subproblem, calling the function as an instruction in the calling algorithm. Functions as implemented in the programming language: Function definition, the body of the function, call-by-value parameters, local variables.

The material is linked to the concepts studied in Chapter 3. Functions written by students are combined with library functions. Special emphasis is placed on the need for the called function to be correct with respect to its input assumptions, and on documentation to aid in viewing the function as a 'black box'; this is done using entrance and exit assertions.

Chapter 9: One-dimensional arrays

Goals: Introducing the need for data structures. Presenting the array data structure as a linear collection of variables of the same type.

Contents: Data structures and why they are needed. Arrays, types of arrays, indices, range and type of indices. Characters and arrays of characters. The *for* sentence as a tool for 'going through' an entire array.

Emphasis is placed on the the similarity of variables and array elements, and on the fact that arrays are used to collect a group of variables used for a common purpose in order to facilitate unified algorithmic treatment. The strong link between iteration as a mechanism for constructing algorithms and the array as a mechanism for organizing data is demonstrated. The link between the *for* sentence and one-dimensional arrays and the dual role of the indices in this context is also emphasized.

Chapter 10: Control structures revisited

Goals: Enhancing the students' knowledge of the material studied in the previous chapters. Studying the search problem.

Contents: The construction of an algorithm for a complex problem by combining control structures. Pseudocode is used to strengthen the student's ability to analyze and decompose a problem before implementation in the programming language.

The type Boolean, the *not* connector; nesting and combining constructs (nested *while*'s, combined *if* and *while*, the use of control mechanisms inside functions, etc.).

Demonstrating and practicing the studied material by solving more advanced problems. For instance, problems on sorted or almost sorted arrays (but not the sorting itself), sequential search, finding pairs that are in the wrong order, etc. For more advanced students, binary search may be added. More material on characters may be given, using examples from word processing. Algorithmic efficiency and correctness are dealt with in greater depth, as are different kinds of iteration and combinations thereof.

4 Fundamentals 2

This second unit first revisits and expands upon some of the topics covered in Fundamentals 1, in order to deepen the student's understanding of that material. A number of new facets of algorithmic analysis and design are emphasized, such as stepwise refinement, and top-down techniques. In addition, the following new topics are introduced: Recursion (only for the "5-unit-ers"), procedures, two-dimensional arrays, type declarations and records. This unit treats correctness and time efficiency in more detail. A special section is devoted to particular algorithms for specific problems (such as sorting), which serve to better illustrate the studied issues.

Background bibliography: [Kof1, Sedg, WiJe]

Development team:

Work done at the Department of Science Teaching of the Weizmann Institute of Science, by Mordechai Ben-Ari, David Ginat, Orna Lichtenstein, Ela Zur, Esther Hershkovitz, Hannah Mahleb, Yifat Kolikant and Nurit Reich.

Fundamentals 2

	conceptual	experimental
Ch. 1 – Developing algorithms	8	–
Ch. 2 – Subtasks: Procedures	8	10
Ch. 3 – Types	2	2
Ch. 4 – Recursion	6	4
Ch. 5 – Characters and strings	2	4
Ch. 6 – Advanced algorithmic problems	18	6
Ch. 7 – Correctness and efficiency revisited	10	–
Ch. 8 – Introduction to data structures	6	4
total hours	60	30

(The allocation of hours applies to students of 5-units.)

Chapter 1: Developing algorithms

Goals: Practicing and expanding the student's ability to analyze and solve algorithmic problems.

Contents: Analysis of more complex problems, and the development of solutions using stepwise refinement, in a top-down fashion; modular programming.

Chapter 2: Subtasks: Procedures

Goals: Introducing procedures as an additional tool for solving a problem by decomposition into subproblems. Expanding the student's knowledge of procedures and functions, and the differences between them.

Contents: The use of procedures as solutions for subproblems, the different kinds of parameters. Procedures as implemented in the programming language: Procedure declaration, calling a procedure, review of call-by-value parameters, call-by-reference parameters, the scope of variables, documentation of procedures.

Connection with the material of Chapter 8 of Fundamentals 1 is made, and the differences between procedures and functions are elaborated upon.

Chapter 3: Types

Goals: Introducing the concept of data type.

Contents: The concept of type; type declaration, enumerated type, the array type.

Type declaration is needed for dealing with parameters of user-defined types. However, this chapter does not get into an in-depth study of the concept of a data type.

Chapter 4: Recursion

Goals: Presenting the concept of recursion as a tool for problem solving. Introducing the advantages and disadvantages of recursion.

Contents: Recursive calls, the base of the recursion, termination condition. Tracing recursive algorithms, writing recursive algorithms. The relationship between recursive definitions and their corresponding recursive program (examples to be used: The factorial function and the Fibonacci sequence).

Advantages and disadvantages of recursive programming should be discussed, with respect to the process of programming, and the program's time and memory efficiency.

Chapter 5: Characters and strings

Goals: Providing tools for solving textual problems.

Contents: Strings, lexicographic order. Representation of strings as arrays of characters, and the construction of string functions. The compression of characters by packing.

This chapter's material may be integrated into other chapters rather than being taught separately.

Chapter 6: Advanced algorithmic problems

Goals: Introducing searching, sorting and merging, and studying various algorithms for solving them. Providing additional practice and expansion of the previous topics studied in the unit.

Contents: Searching and sorting, merging sorted arrays.

Two search methods are taught: Searching with a sentinel and binary search. 5-unit students are taught at least two sorting methods, one of which is recursive, such as mergesort.

(The first 12 hours of this chapter are taught here, and the remaining 6 hours are better taught after Chapter 7.)

Chapter 7: Correctness and efficiency revisited

Goals: Expanding the discussion on the correctness and efficiency of algorithms. Providing tools and methods for debugging.

Contents: Methods for counting executions of the main operations according to the type of problem (for example, in sorting — the number of comparisons, and in primality testing — the number of multiplications and divisions); the efficiency of recursive algorithms, debugging program segments before combining them into a whole program. Debugging parts of programs by using "scaffolds". Using computerized tools for debugging.

When constructing an algorithm for solving a problem, the problem's decomposition into subproblems is related to checking the correctness of the algorithm and its parts.

Chapter 8: Introduction to data structures

Goals: Introducing the record data structure. Constructing more complex data structures.

Contents: Compound data structures: Arrays, records and their integration; fields of a record.

The definition of a two-dimensional array and its indices; defining records, accessing a field of a record, input to a record.

5-unit students also study files of records, focusing on the following: Sequential access, unbounded length, files as a robust memory, access time and its influence on a program's execution time.

5 Software Design

This unit's main goal is to take the student beyond stand-alone algorithms, and to teach the basic principles of system design. It also develops abstract thinking skills, especially by the definition, selection and usage of abstract data types. Particular data structures are taught, such as stacks and binary trees, and are then utilized in solving problems. The student's ability to analyze efficiency is deepened. Dynamic memory allocation is touched upon. One of the most important subjects of this unit is the full integration of the conceptual material and the actual hands-on experience in designing and constructing a real system.

Background bibliography: [Har, Aho, HoSh, Kof1, ReHa]

Development team:

Work done at the Science Teaching of the Hebrew University of Jerusalem, by Ofra Brandes, Tamar Vilner, Daniel Goldschmidt, Gadi Shamir, Ran Cohen, Yishai Mor, Yoram Cohen, Rachel Katzman and Naomi Lindenstrauss.

Software Design

	conceptual	experimental
Ch. 1 – Introduction	1	–
Ch. 2 – Library unit	2	4
Ch. 3 – Data types	5	7
Ch. 4 – Stack	3	3
Ch. 5 – Efficiency	7	4
Ch. 6 – List	12	9
Ch. 7 – Binary tree	11	6
Ch. 8 – Case study	6	4
total hours	45	45

Chapter 1: Introduction

Goals: Presenting the benefits of dividing a system into subtasks. Providing a more in-depth treatment of the top-down design method. Preparing the students for the case study by exposing them to the basic concepts required for dealing with it.

Contents: Topics from software engineering: System specification, modules and their interfaces, implementation, information hiding, code reuse, user interfaces.

Properties of a software system design, such as correctness, robustness, efficiency, and documentation. System maintenance and upgrade.

Chapter 2: Library unit

Goals: Providing in-depth acquaintance with the library unit of the work environment. Reinforcing the idea of a software system's modularity by designing and using library units.

Contents: The construction of a library unit in the work environment. Linking the library unit to the program.

Chapter 3: Data types

Goals: Introducing the concept of an abstract data type. Learning and practicing the definition, representation, implementation and usage of an abstract data type. Emphasizing the importance of dealing with exceptions and boundary cases.

Contents: Predefined data types, such as integers. Definition of an abstract data type, including values, operations and scopes. Various representations of data types and their limitations. Implementing an abstract data type.

Chapter 4: Stack

Goals: Exercising the definition and implementation of abstract data types. Introducing the abstract data type stack and its various uses.

Contents: Definition of the abstract data type stack, with its values and operations. Implementation of the stack using an array.

Chapter 5: Efficiency

Goals: Deepening the understanding of the concept of efficiency via time complexity, and recognizing its importance. Discussing basic concepts related to efficiency. Introducing order of magnitude estimates. Developing the ability to analyze an algorithm's run-time complexity.

Contents: Efficiency of an algorithm in terms of time and memory space. Analysis of the run-time complexity of algorithms; input length; improving an algorithm's efficiency by a constant factor; order of magnitude complexity (the big-O notation); special cases thereof (e.g., logarithmic, linear, quadratic and exponential). Comparison of different orders of magnitude for different algorithms; best, worst and average case complexity; improving an algorithm's efficiency by an order of magnitude.

Efficiency analysis of sequential search and binary search, bubble-sort and mergesort.

(Throughout the remaining chapters of the unit the efficiency of the various algorithms is always discussed.)

Chapter 6: List

Goals: Introducing the data type list. Mastering the principle of information hiding by studying different implementations of a common operation. Introducing the mechanism of dynamic memory allocation. Implementing an abstract data type by using existing ones.

Contents: Design of an interface for the data type list; representation of a list by an array and by a linked list; comparison of the efficiency of different algorithms according to these different representations. The stack and queue as special cases of a list; insertion sort; dynamic memory allocation.

Chapter 7: Binary tree

Goals: Introducing the data type binary tree and some of its uses. Practicing the use of recursive routines and analyzing their efficiency. Introducing the binary search tree and its use in sorting.

Contents: The data type binary tree; tree concepts, such as parent, ancestor, sibling, offspring (left and right), descendent, height, path, leaf, full tree, node, edge, depth, root, sub-tree (left and right).

Design of an interface for the data type binary tree. Post-order traversal, in-order traversal, pre-order traversal, traversal by levels, search tree, sorting with a search tree.

Chapter 8: Case study

Goals and contents: Gaining hands-on experience in constructing a (small) software system. This starts with an analysis of the customer's requirements, moving on to the system's specification and design, and finally constructing a structured computerized software system that satisfies the requirements with possibilities for expansion and improvement. Practicing the practical implementation of the data types discussed in the study unit, while using and applying the general design concepts presented herein, such as going from the general case to the special case, modularity, information hiding, library units, system specification, interfaces, data types, efficiency of the implementation, documentation and readability. Gaining some experience in teamwork.

6 Logic Programming

The main goal of this unit is to introduce the student to the logic programming paradigm, and to its applications in the representation and manipulation of knowledge. It also introduces some basic concepts from logic and discusses knowledge representation by facts and clausal rules. Programming is carried out in (a Hebrew version of) Prolog, with recursion, lists and trees taking a prominent place in the material.

Background bibliography: [BHBS, Brat, ClMe, Habe, Kowa, StSh]

Development team:

Work done at the Department of Science Teaching of the Weizmann Institute of Science, by Zahava Scherz, Ehud Shapiro, Bruria Haberman, Noa Ragonis, Ronit Ben-Basat and Moshe Pontch.

Logic Programming

	conceptual	experimental
Ch. 1 – Propositional and predicate calculus	6	–
Ch. 2 – Logic programming at the propositional level	2	2
Ch. 3 – Logic programming at the predicate level	2	2
Ch. 4 – Rules and inferences	3	2
Ch. 5 – Representing and formalizing knowledge	8	4
Ch. 6 – Input/output and numerical computations	2	2
Ch. 7 – Negation	4	2
Ch. 8 – Recursion	8	4
Ch. 9 – Compound data structures	3	2
Ch. 10 – Abstract data types for representing knowledge	4	–
Ch. 11 – Lists	8	4
Ch. 12 – Trees	3	1
Ch. 13 – Graphs	3	1
Ch. 14 – Project	4	4
total hours	60	30

Chapter 1: Propositional and predicate calculus

Goals: Becoming familiar with the basics of the propositional and predicate calculus, logical connectives and rules of inference.

Contents: Logical inference, the propositional calculus; the *or*, *and*, and *not* connectives; truth-preserving deductions; valid proofs and rules of inference; the predicate (relational) calculus.

Note: Much of the subject is presented informally. The propositional calculus can be presented in part via the Prolog work environment.

Chapter 2: Logic programming at the propositional level

Goals: Becoming familiar with the structure of logic programming at the propositional level, and its implementation in Prolog. Representing facts by means of logic programming and Prolog.

Contents: Logical statements and their representation as facts in Prolog; the *or* and *and* connectives in Prolog; the inference mechanism. Becoming acquainted with the Prolog environment, and working within it on the development of simple programs.

Chapter 3: Logic programming at the predicate level

Goals: Becoming familiar with logic programming at the predicate calculus level, and its implementation in Prolog.

Contents: Predicates and arguments; using variables in queries; quantified variables. Writing simple programs in Prolog.

Note: The treatment of variables emphasizes the difference between variables in Prolog and variables in a procedural programming language such as Pascal.

Chapter 4: Rules and inferences

Goals: Learning how to form rules in logic programming using variables. Understanding Prolog's inference mechanism.

Contents: Anonymous variables, *is_a* rules; the workings of the Prolog inference mechanism; locating errors.

Note: Emphasis is placed on the difference between the Horn-clause rules of logic programming and the more common conditional *if_then* rules.

Chapter 5: Representing and formalizing knowledge

Goals: Learning how to solve problems and represent knowledge in logic programming, with implementation in Prolog. Becoming familiar with various methods of knowledge representation.

Contents: Knowledge representation using data structures such as trees, graphs and tables; a variety of examples; development of appropriate programs in Prolog.

Notes: This chapter also discusses some methodological principles of Prolog programming, such as the various stages in writing a program and top-down, bottom-up and stepwise refinement methods.

Chapter 6: Input/output and numerical computations

Goals: Becoming familiar with system predicates and extra-logical predicates in Prolog. Learning to carry out numerical calculations in Prolog. Learning to use input/output predicates for interactive programs.

Contents: Numerical predicates, extra-logical predicates, and input/output predicates. Writing programs that combine input/output and numerical operations; System predicates, such as *listing*, *consult* and *systemp*.

Notes: Attention should be drawn to the fact that, unlike many other elements in Prolog, input/output relations are not declarative but are commands for execution, and overuse of them might damage the declarative character of a Prolog program.

Chapter 7: Negation

Goals: Introducing the negation predicate in logic programming, and its use in forming rules and queries.

Contents: The *not* predicate in Prolog; negation by failure of proof; the use of negation in queries and in defining predicates; the effect of negation on different kinds of terms; the effect of *not* in the Prolog proof mechanism.

Chapter 8: Recursion

Goals: Becoming familiar with simple recursion (non list recursion) in logic programming and its implementation in Prolog.

Contents: The declarative aspect of recursive rules in logic programming; writing simple recursive programs in Prolog, such as ones involving ancestoral relationships and arithmetic predicates.

Notes: Recursion in Prolog is compared with that of a procedural language such as Pascal. Graphical descriptions of proof trees can be used in order to help follow the recursion mechanism in Prolog. List recursions is not to be introduced here.

Chapter 9: Compound data structures

Goals: Becoming familiar with different types of compound data structures and their use in Prolog.

Contents: Knowledge representation using compound data structures; the functor data structure; representation of general trees using compound data structures.

Chapter 10: Abstract data types for representing knowledge

Goals: Becoming familiar with abstract data types and their use in knowledge representation. Studying particular abstract data types.

Contents: Specification of the following abstract data types: list, set, multiset, tree and graph. Determination of the appropriate abstract data types for various problems.

Notes: Emphasis is placed on the fact that the specification is made on an abstract level and does not depend on the implementation environment.

Chapter 11: Lists

Goals: Deepening familiarity with the list abstract data type, its use in knowledge representation, and its implementation in Prolog. Learning the use of a predefined (built-in) list predicate in Prolog programs.

Contents: The use, in writing programs, of predefined predicates that represent a list; the solution of problems using lists; use of the list data type in Prolog for implementing abstract data types, such as list, set and multiset. Knowledge representation using lists; lists of functors, lists of lists.

Notes: The difference between the list abstract data type and the list data structure in Prolog is discussed. Advanced students can be taught the [head—tail] syntax of Prolog.

Chapter 12: Trees

Goals: Deepening familiarity with the tree abstract data type; its use in knowledge representation, and its implementation in Prolog.

Contents: Use in writing programs of predefined predicates that represent a tree; solution of problems using trees; implementation of the various concepts constituting a tree, such as root, leaf, node level and tree height.

Chapter 13: Graphs

Goals: Deepening familiarity with the graph abstract data type; its use in knowledge representation, and its implementation in Prolog.

Contents: Use in writing programs of predefined predicates that represent a graph; solution of problems using graphs;

Chapter 14: Project

Goals: Enabling students to acquire a broad view of the entire unit by implementing the material in a project of their choice, resulting in a working program in Prolog.

Contents: Choice of a subject and the gathering of relevant data; stepwise development of the program in Prolog, including definition of objectives, knowledge representation means, choice of predicates, programming and testing.

Notes: It is recommended that the Prolog program involve at least two of the following: negation, recursion, structures, lists.

7 Computer Organization and Assembly Language

This unit, which one of the possible alternatives for the Second Paradigm unit, presents the conceptual structure of a computer system, and provides an introduction to programming in assembly language. The work environment used is *Easy CPU*, a simple visual framework for teaching these topics. An alternative to this is the *Turbo-Assembler* tool.

Background bibliography: [Hoff, RaHo, Some, Tann, Thor, ZaWo]

Development team:

Work done at the Department of Mathematics and Computer Science of the Open University of Israel, by Hanita Zilberman, Dina Kraus, David Lupo and Cecile Yehezkel.

Computer Organization and Assembly Language

	conceptual	experimental
Ch. 1 – Number systems	5	2
Ch. 2 – Computer organization	4	1
Ch. 3 – Organization and execution of programs	8	–
Ch. 4 – Basic concepts of assembly language	7	25
Ch. 5 – Assembling, linking and loading	8	–
Ch. 6 – The stack and subprograms	6	8
Ch. 7 – Interrupts	5	5
Ch. 8 – From high-level languages to assembly language	2	4
total hours	45	45

Chapter 1: Number systems

Goals: Becoming familiar with number bases in general, and in particular the binary, octal and hexadecimal bases. Learning the 2-complement method of representing binary numbers.

Contents: Number bases; binary, octal and hexadecimal bases and their inter-relationships; positive binary numbers of fixed length; negative binary numbers; the 2-complement method; floating point numbers, ASCII code.

Chapter 2: Computer organization

Goals: Getting acquainted with the CPU and the memory, and the relationships between them.

Contents: Schematic view of the computer system; the general structure of memory; accessing the memory; coding instructions; allocating memory to instructions and to data; other components of the CPU.

Chapter 3: Organization and execution of programs

Goals: Understanding instructions of the machine language, their components and execution. Understanding the process of executing a program.

Contents: Machine language; the construction of a collection of basic instructions in machine language; writing a program in machine language; the fetch cycle; executing a program in machine language.

Chapter 4: Basic concepts of assembly language

Goals: Running an assembly language program. Writing simple programs in assembly language, using variables, indirect addressing and labels. Tracing the execution of assembly language programs and debugging them.

Contents: The general structure of an instruction; basic instructions; the flag register; the general structure of a program in assembly language; memory access using indirect addressing; segments; jump instructions; logical instructions and move and rotate instructions.

Chapter 5: Assembling, linking and loading

Goals: Becoming familiar with the process leading to the full execution of an assembly language program; this includes translation of the various instructions and operands into machine language. Becoming familiar with two-pass assembly.

Contents: Instruction translation; two-pass assembly; using labels that are not yet defined; locating and dealing with errors; linking and loading.

Chapter 6: The stack and subprograms

Goals: Becoming familiar with the stack data structure and the actions executed on it. Writing a subprogram in assembly language. Becoming familiar with the call and return mechanism for subprograms, and with parameter passing mechanisms.

Contents: The stack and its representation in memory; storing data in the stack; subprograms, calling a subprogram, returning from a subprogram; passing parameters to a subprogram via registers and via a stack.

Chapter 7: Interrupts

Goals: Introducing interrupts, their types and implementation. Becoming familiar with the masking technique for interrupts and the order of priorities for executing them.

Contents: The nature of interrupts; masking interrupts, internal and external interrupts, order of priorities in executing interrupts; the interrupt execution process; the role of the stack in interrupt processing; the interrupt vector table; assembly instructions used by interrupts; using interrupts in a program.

Chapter 8: From high-level languages to assembly language

Goals: Connecting the material of this unit with that of Fundamentals 1 and 2, from both conceptual and practical points of view. Providing an overview of the uses of assembly language.

Contents: The connection between a high-level language and assembly language. Using assembly language procedures in Pascal programs; using Pascal for replacing existing interrupts with new interrupts; the minicontroller.

8 Introduction to Computer Graphics

This unit teaches the basics of representing and manipulating graphic objects, and implementing systems that use them. The main topics include the internal structure and operational principles of graphic software systems, algorithms for handling geometric data, and the use of three-dimensional graphic software for creating computerized three-dimensional models. The unit combines conceptual and practical aspects of the subject matter.

Background bibliography: [FDFH, BuGi, Zeid]

Development team: Work done at the School of Technology of the Open University of Israel, by Yaakov Sheinbaum and Amos Sorgen.

Introduction to Computer Graphics

	conceptual	experimental
Ch. 1 – Introduction	3	17
Ch. 2 – Geometric models	8	4
Ch. 3 – The geometric data base	6	3
Ch. 4 – Representing curves and surfaces	6	4
Ch. 5 – From the data base to the screen	5	2
Ch. 6 – Transforms	7	2
Ch. 7 – Algorithms for image production	6	3
Ch. 8 – Information transfer between systems	4	2
Ch. 9 – Project	–	8
total hours	45	45

Chapter 1: Introduction

Goals: Getting acquainted with three-dimensional (3D) computer graphics and its various uses, and the difference between it and the 2D case. Becoming familiar with the general logical structure of a graphic system, with its main operations, and with various input and output devices. Teaching the basics of the graphic work environment.

Contents: A review of some areas in which 3D systems are used, such as architecture, computer games, and topographical mapping. Means for user interaction with the graphic system; the system's main hardware components and their interconnections; the main operations performed in the system; display management via the display file and frame buffer; parallel management of the graphics pipeline; input and analysis of the user's instructions through the user interface; data base management. Various input/output devices, such as mouse, glove, and virtual reality helmet with head mounted display. Hands on experience in executing the main operations of the graphic work environment.

Chapter 2: Geometric models

Goals: Becoming familiar with the concept of geometric models. Studying a 2D model of simple graphic shapes, and various 3D models for simple objects.

Contents 2D geometric models: 2D representation of points, line segments, regular polygons and circles. Representations that duplicate information vs. those that use pointers to existing information. 3D geometric models: 3D representation of simple objects using the wire frame method, the boundary representation and constructive solid geometry (CSG). Typical applications of each of these methods. Geometric features that can be extracted from the model, such as length, slope, curvature, area and volume. Non-geometric features that can be extracted from the model.

Chapter 3: The geometric data base

Goals: Becoming acquainted with the notion of a geometric data base, including the representation of objects using the methods of Chapter 2. Learning how to implement groups and layers.

Contents: Geometric data base, record and field. Basic operations on records: Creating, deleting and updating. Record identifiers and directory; 3D object representation using the methods of Chapter 2. The need for groups, and their implementation in the data base. The need for layers and their implementation in the data base. Implementation in the graphic system of various operation, such as the creation of an object represented by several records, check of whether two faces of an object are adjacent, and location of all the faces adjacent to a given face.

Chapter 4: Representing curves and surfaces

Goals: Becoming acquainted with methods for representing general 3D curves and surfaces.

Contents: General three dimensional curves; representing a curve as a polygon; B-spline curves and Bézier curves. Surface representation; representing polygonal surfaces, ruled surfaces, surfaces of revolution, swept surfaces, B-spline surfaces and Bézier surfaces.

Chapter 5: From the data base to the screen

Goals: Becoming acquainted with the route of information from the data base to the display screen. Learning about the display file and the frame buffer, and studying methods for saving information in them.

Contents: Graphic information as organized in three parts: A model saved in the data base, a display file and a frame buffer. The display file: Its contents, arrangement by segments and windows in the display file. The frame buffer: Its contents, the direct method for saving data, the color table method, producing information in the frame buffer from the display file. The viewport; the role of the display driver in creating the frame buffer. Identifying an object indicated by the cursor.

Chapter 6: Transforms

Goals: Becoming familiar with the concept of transform. Understanding the need for transforms in a graphic system, and learning how to compute basic transforms.

Contents: The need for transforms in the model, the display file and the frame buffer. 2D transforms: Translation, rotation around the origin, rotation around any point, scaling relative to the origin, scaling relative to any point. 3D transforms: Translation, rotation around an axis parallel to the z axis, scaling.

Chapter 7: Algorithms for image production

Goals: Understanding the need for image production algorithms. Learning several particular algorithms.

Contents: Algorithms for drawing lines, smoothing lines, clipping, filling in areas, removing hidden lines, removing hidden surfaces and shading. The Cohen-Sutherland algorithm for line clipping; the DDA algorithm for drawing a line; an algorithm for removing hidden lines; the z-buffer algorithm for removing hidden surfaces.

Chapter 8: Information transfer between systems

Goals: Understanding the need to standardize the transfer of graphic information from one graphic system to another. Becoming familiar with a number of standards for transferring graphic information.

Contents: The compatibility problem in transferring graphic information between systems, and the role of standardization in solving it. Import and export software for transforming information between the standard format and the graphic system's internal format. Principles of representation in several standards, such as IGES, STEP (formerly CAD-I), OPENGL, DXF, GGM, RLC, PIF and GIF.

Chapter 9: Project

Goals: To build a working 3D computerized model of a complex object.

Contents: The student will be required to execute his/her own project, in which a 3D computerized model for a complex object will be built. The model will include at least three general surfaces (such as those studied in Chapter 4). Examples of suitable objects are a telephone, a watch, a simple aircraft, an electric kettle or a computer screen.

9 Introduction to Information Systems

This unit teaches the basics of management information systems. It discusses logical file and data organization, a system's life cycle, and basic system modeling and analysis. The unit combines conceptual and practical aspects of the subject matter. The work environment chosen for the current implementation of the unit is *Microsoft Access*.

Background bibliography: [ElNa, KoSi, Oneil]

Development team:
Work done at the School of Technology of the Open University of Israel, by Zvi First, Arie Ben-David, Eyal Shifroni, Israel Zilberstein, Moradian Nissan and Shula Shazman.

Introduction to Information Systems

	conceptual	experimental
Ch. 1 – Introduction	4	2
Ch. 2 – Relations	10	8
Ch. 3 – The relational data base	10	10
Ch. 4 – Data base management	3	–
Ch. 5 – Information systems	4	–
Ch. 6 – Data base design	6	5
Ch. 7 – Data flow diagrams	4	4
Ch. 8 – User interfaces	4	4
Ch. 9 – Project	–	12
total hours	45	45

Chapter 1: Introduction

Goals: Becoming exposed to the concept of an information system (IS), its characteristics and usage. Becoming familiar with the main types of activity in information systems, and experiencing interaction with an information system.

Contents: Definition of an information system; data and information; the use of data for action implementation, control, decision-making, forecast, etc. Actions in an information system: Interactive vs. batch processing, queries and reports, updates and transactions. Components of an information system: Data bases software for their management, communications channels, I/O interfaces, operators and users. Information system representation: Static, dynamic, functional. Initial experience in interacting with a simple information system.

Chapter 2: Relations

Goals: Understanding the importance of modeling data as sets of entities with common characteristics. Introducing relations (i.e., tables) as the basic units of data organization, and their use in representing a set of entities. Experiencing actions on relations.

Contents: Entities constituting an information system and their partition into collections with common structure and characteristics. Records and fields, keys. Queries: Selection by conditions and key values, projections; formalization in SQL, aggregation and group-by operations. Updates: Addition and deletion of records, changing values.

Chapter 3: The relational data base

Goals: Understanding a data model as consisting of entities and relationships, and representing these as a relational data base. Experiencing the querying and updating of a relational data base.

Contents: Entities and relationships; representation of a set of relationships by a table/relation; the use of disjoint keys. Functional dependencies and their representation in a common relation.

Relational schemes. Querying and transactions on multiple relations; formalization in SQL; report generation. Updates: The need for consistency, input checking.

Chapter 4: Data base management

Goals: Surveying the functions and services of a data base management system.

Contents: Definition and update of data base structure; modification of the structure definition language; data storage, data generation, data updates, query execution; access control, data consistency and reliability maintenance; support for multiple users; forms and visual interfaces.

Chapter 5: Information systems

Goals: Introducing the components of an information system and the links between them. Gaining experience in defining the goals of an information system and its environment.

Contents: Information system hardware, data bases, software for data management and communication; operators and users; software for human interaction, system procedures. Identification of the objects and goals of the system, determining the environment and its interaction with the system, identification of sources and consumers of data; user interfaces and interaction scenarios between the system and its users. Isolation of the system functions, identification of its main components and the links between them; data and information processing,

Chapter 6: Data base design

Goals: Gaining deeper understanding of the notion of a data model, and comparing the entity-relationship (ER) model, used for specification and analysis, with the relational model, used for implementation. Acquiring the ability to translate from an ER model to a relational model. Gaining experience in designing a simple data base.

Contents: The data model as a tool for the uniform description of data bases, and for description and abstraction. The relational model as an implementation model; abstraction in the relational model, hiding the internal organization of the data. The entity-relationship model as a tool for data specification and analysis; entities, relationships and their cardinalities, attributes, sets of entities and sets of relationships. Entity-relationship diagrams (ERDs). Transforming an ERD into a relational schema. The quality of a data base schema, normalization as a method for improving the schema. Designing a data base for a simple application.

Chapter 7: Data flow diagrams

Goals: Introducing data flow diagrams (DFDs) and their use in the description of an information system.

Contents: Data flow diagrams and their components. Constructing a DFD for a simple information system.

Chapter 8: User interfaces

Goals: Understanding the need for designing user interfaces, and becoming familiar with design principles. Understanding the connection between the user interface and the system's behavior.

Contents: The need for designing user interfaces. General principles in man-machine interface and in designing interfaces for information systems; components of the interface. The need for state-based behavioral specification of an information system, and designing the interfaces accordingly. Designing an interface for a simple information system.

Chapter 9: Project

Goals: Experiencing the design and setting up of a simple information system.

Contents: The project involves choosing a system, gathering relevant data, producing system requirements, designing the system, building a logical model and constructing a working prototype.

10 Models of Computation

This unit exposes the student to some basic models of computation, which are considered fundamental from both theoretical and practical points of view. It includes finite automata and regular languages, pushdown automata and context-free languages, and some basic material on Turing machines and the Church–Turing thesis. The notion of expressive (or computational) power is discussed, as are closure properties.

While some of the material could have been illustrated and practiced via a computer, this unit remains theoretical in spirit (with proofs and mathematical rigor taking the place of programming and experimentation). It thus has a conceptual track only.

The unit comes in two versions: A full 90-hour version that constitutes the entire Theory module, and an abridged 45-hour version that forms one half of the Theory module alongside the 45-hour Numerical Analysis unit.

Background bibliography: [AuFL, BaEt, DaSW, Har, HoUl]

Development team:

Work done at the Department of Mathematics and Computer Science of the Open University of Israel, by Michal Armoni and Yossi Kaufman.

10.1 The 90-hour version

Models of Computation: 90-hours

	conceptual
Ch. 1 – Finite systems	6
Ch. 2 – Deterministic finite automata	15
Ch. 3 – Regular languages	15
Ch. 4 – Variants of finite automata	15
Ch. 5 – Pushdown automata	12
Ch. 6 – The power of pushdown automata	12
Ch. 7 – Turing machines	15
total hours	90

Chapter 1: Finite systems

Goals: Introducing the basic concepts of a finite system and its description via states and transitions.

Contents: Graphic description of finite systems; states, inputs, transitions, initial states. Solving riddles with the help of graphic descriptions.

Chapter 2: Deterministic finite automata

Goals: Introducing the deterministic finite automaton model (DFA). Practicing the construction of DFAs.

Contents: Deterministic finite automata; accepting and non accepting computations; graphic, tabular and functional description of automata; counting automata and search automata.

Chapter 3: Regular languages

Goals: Introducing the basics of formal languages. Studying the expressive power of DFAs and the regular languages they accept.

Contents: Basic concepts of formal languages, including symbols, alphabets, words, word length, the empty word and languages. Operations on words and languages, such as concatenation, power, and inverse. Regular and non-regular languages; closure of the family of regular languages under complementation, intersection and union.

Chapter 4: Variants of finite automata

Goals: Showing how a new model of computation can be obtained from a given one by adding or modifying features. Introducing the concept of nondeterminism. Comparing the power of computational models.

Contents: Incomplete automata, nondeterministic finite automata (NFAs); the equivalence of the deterministic and nondeterministic finite automata; further closure properties of regular languages, including concatenation and inverse.

Chapter 5: Pushdown automata

Goals: Introducing the nondeterministic pushdown automaton model (PDAs). Practicing the construction of PDAs.

Contents: The use of a pushdown stack as an aid in automata computation; the nondeterministic pushdown automaton.

Chapter 6: The power of pushdown automata

Goals: Understanding the power and limitations of PDAs. Comparing pushdown automata and finite automata.

Contents: Deterministic pushdown automata (DPDAs); comparing PDAs and DPDAs; context-free languages, languages that are not context-free; discussion of the closure status of context-free languages under complementation, intersection, union, concatenation and inverse.

Chapter 7: Turing machines

Goals: Presenting Turing machines as a model for general computer programs. Introducing the Church–Turing thesis.

Contents: Turing machines: Definition and examples; non-termination of Turing machines, Turing machines computing functions; Turing machines modeling computer programs; the Church–Turing thesis; the halting problem.

10.2 The 45-hour version

Models of Computation: 45-hours

	conceptual
Ch. 1 – Finite systems	4
Ch. 2 – Deterministic finite automata	10
Ch. 3 – Regular languages	10
Ch. 4 – Variants of finite automata	9
Ch. 5 – Turing machines	12
total hours	45

As can be seen from the table, this version omits the material on pushdown automata and context-free languages. In addition, the remaining chapters are given somewhat less hours. The contents of these chapters is generally the same as in the 90-hour version, except that Chapter 2 leaves out counting and search automata, and Chapter 3 leaves out incomplete automata. Moreover, in comparison with the full version of the unit, the treatment of the topics here inevitably sacrifices some depth and detail.

11 Introduction to Numerical Analysis

This unit discusses the algorithmic solution of numerical problems, and introduces the notion of approximate solutions. It also motivates the need for evaluating the accuracy of the results. It provides hands-on experience in developing various algorithms, and in checking and comparing their results. The two main problems it deals with are solving systems of linear equations and extracting the roots of functions. Issues treated include round-off errors, absolute and relative errors, approximate solutions with error control, and ill-conditioned problems. The algorithms are studied together with the needed underlying mathematics, including proofs on a high-school level.

Background bibliography: [Atki, BrZw, CoBo, Enge]

Development team:

Work done at the Department of Science Teaching in the School of Education of Tel-Aviv University, by Gideon Zwas and Ronit Hoffman.

Introduction to Numerical Analysis

	conceptual	experimental
Ch. 1 – Basic concepts	6	2
Ch. 2 – An algorithmic approach to linear systems	11	6
Ch. 3 – Iterative solutions of non-linear equations	13	7
total hours	30	15

Chapter 1: Basic concepts

Goals: Introducing the types of errors that can occur during the execution of computerized calculations. Becoming familiar with various number representation schemes.

Contents: Approximate solutions, the need for evaluating the quality of the approximation, error bounds; round-off errors, absolute errors and relative errors. The floating point representation method, binary representation.

Chapter 2: An algorithmic approach to linear systems

Goals: Introducing numerical methods for solving systems of linear equations, and analyzing their computational complexity.

Contents: Gauss elimination for linear systems; triangulation and backward substitution and their complexity; the general algorithm for the naive case. The need for pivoting, the need for scaling, ill-conditioned systems.

Chapter 3: Iterative solutions of non-linear equations

Goals: Introducing an iterative process for extracting the roots of a non-linear function. Understanding the notion of order of convergence. Constructing an iterative formula adhering to requirements on the convergence of the iteration.

Contents: Simplifying root extraction by reducing the domain and range of the function; the bisection method, second order iterations, quadratic vs. cubic convergence for extracting square roots, the importance of initialization and its quality, generalizations leading to the Newton-Raphson method.

Acknowledgements

This paper reflects the efforts of many people. In fact, as mentioned in the footnote to the title, the material in it can actually be said to have been multiply-authored by a large group. First and foremost in this group are the members of the committee that designed the entire program, chaired and led throughout by Amiram Yehudai from the Computer Science Department of Tel-Aviv University. Besides the Chairman and ourselves, the committee included one additional computer scientist, Catriel Beeri from the Computer Science Institute of the Hebrew University. The other members of the committee were associated in one way or another with the Ministry of Education, and included the head of the computers and computer science section in the Ministry (for most of the time this was Ben-Zion Barta, and more recently Bruria Haberman), as well as Meir Komar, Roni Dayan, Ephraim Engel, David Levant, and David Sela.

Obviously, an enormous debt is owed to the members of the various material development teams, who are all listed in the appropriate sections of the paper. The dedication and contributions of these people were invaluable throughout the entire effort. Their work went far beyond merely writing course material based on a curriculum "handed down" by the committee; they contributed to the very construction of the curriculum itself and to its "debugging" and modification. Without them the project could not have been carried out.

References

[Aho] Aho, A.V., *Data Structures and Algorithms*, Addison-Wesley, 1983.

[Atki] Atkinson, K., *Elementary Numerical Analysis*, Addison-Wesley, 1993.

[BaEt] Barwise, J., and J. Etchemendy, *Turing's World: An Introduction to Computability Theory*, CSLI Publication, Stanford, CA, 1993.

[BHBS] Scherz, Z., B. Haberman, E. Ben-Zaken and N. Globerman, *Computer Science: Logic Programming (Prolog)*, Ramot, 1996 (in Hebrew).

[Brat] Bratko, I., *Prolog Programming for Artificial Intelligence*, Addison-Wesley, 1990.

[BrZw] Breur, S., and G. Zwas, *Numerical Mathematics — A Laboratory Approach*, Cambridge University Press, 1993.

[BuGi] Burger, P., and D. Gillies, *Interactive Computer Graphics: Functional, Procedural and Device-Level Methods*, Addison-Wesley, 1989.

[ClCo] Clancy, M., and D. Cooper, *Oh! Pascal!* (3rd edition), W. W. Norton and Company, 1993.

[ClMe] Clocksin, W.F., and C.S. Mellich, *Programming in Prolog*, 2nd Edition, Prentice-Hall, 1984.

[CoBo] Conte, S.D., and C. de Boor, *Elementary Numerical Analysis: An Algorithmic Approach*, 3rd Edition, McGraw-Hill, 1981.

[DaSW] Davis, M.D., R. Sigal, and E.J. Weyuker, *Computability, Complexity and Languages: Fundamentals of Theoretical Computer Science*, Academic Press, 1994.

[ElNa] Elmasri, R., and S. Navathe, *Fundamentals of Database Systems* (2nd edn.), Benjamin/Cummings, 1997.

[Enge] Engel, A., *Exploring Mathematics with your Computer*, The Mathematical Association of America, 1993.

[FDFH] Foley, J.D., A. van Dam, S.K. Feiner, J.F. Hughes, and R.L. Phillips, *Introduction to Computer Graphics*, Addison-Wesley, 1994.

[AuFL] Francez, N., and S. Zaks, *Automata and Formal Languages*, The Open University of Israel, Vols. 1 and 2, 1991 (in Hebrew).

[GBHY] Gal-Ezer J., C. Beeri, D. Harel and A. Yehudai, "A High-School Program in Computer Science", *Computer* **28**:10 (1995), 73–80.

[GH] Gal-Ezer, J., and D. Harel, "What (else) should CS educators know?", *Comm. Assoc. Comput. Mach.* **41**:9 (1998), 77–84.

[Habe] Haberman, B., *Introduction to Artificial Intelligence*, Weizmann Institute of Science, Rehovot, 1998 (in Hebrew).

[Har] Harel, D., *Algorithmics: The Spirit of Computing*, Addison-Wesley, 1987 (2nd Edition, 1992).

[Hoff] Hoffman, A., *PC Assembly Language Step by Step*, Abacus, 1994.

[HoUl] Hopcroft, J.E., and J.D. Ullman, *Introduction to Automata Theory, Languages and Computations*, Addison-Wesley, 1979.

[HoSh] Horowitz, E., and S. Sahni, *Fundamentals of Data Structures in Pascal*, 3rd Edition, Computer Science Press, 1990.

[KoSi] Korth, H. F., and A. Silberschatz, *Database System Concepts* (3rd edn.), McGraw-Hill, 1997.

[Kof1] Koffman, E.B., *Pascal*, 3rd Edition, Addison-Wesley, 1989.

[Kowa] Kowalski, R., *Logic for Problem Solving*, North Holland, 1979.

[Oneil] O'Neil, P., *Databases: Principles, Programming, Performance*, Morgan-Kaufman, 1994.

[RaHo] Rasth and Hoffman, *PC Underground*, Abacus, 1996.

[ReHa] Reingold, E.M., and W.J. Hansen, *Data Structures in Pascal*, Little, Brown and Co., 1986.

[Sedg] Sedgwick, R., *Algorithms*, 2nd Edition, Addison-Wesley, 1988.

[Some] Somerson, P., *PC DOS Power Tools*, Bantam Books, 1990.

[StSh] Sterling, L., and E. Shapiro, *The Art of Prolog*, 2nd Edition, MIT Press, 1994.

[Tann] Tannenbaum, A.S., *Structured Computer Organization*, Prentice Hall, 1990.

[Thor] Thorne, M., *Computer Organization and Assembly Language*, The Benjamin/Cummings Publishing Company, Inc., 1991.

[WiJe] Wirth, N., and K. Jensen, *Pascal: User Manual and Report*, 3rd Edition, Springer-Verlag, 1988.

[ZaWo] Zaks, R., and A. Wolfe, *From Chips to Systems*, Sybex, 1987.

[Zeid] Zeid, I., *CAD/CAM, Theory and Practice*, McGraw-Hill International, 1991.