Working Paper Series
ISSN 1170-487X

Computer Concepts without Computers: A First Course in Computer Science

by Geoffrey Holmes, Tony C. Smith and William J. Rogers

Working Paper 97/7
February 1997

COMPUTER CONCEPTS WITHOUT COMPUTERS: A FIRST COURSE IN COMPUTER SCIENCE

Geoffrey Holmes, Tony C. Smith and William J. Rogers
Department of Computer Science, University of Waikato,
Hamilton, New Zealand.
geoff,tcs,w.rogers@cs.waikato.ac.nz

Abstract

While some institutions seek to make CS1 curricula more enjoyable by incorporating specialised educational software [1] or by setting more enjoyable programming assignments [2], we have joined the growing number of Computer Science departments that seek to improve the quality of the CS1 experience by focusing student attention away from the computer monitor [3, 4]. Sophisticated computing concepts usually reserved for senior level courses are presented in a *popular science* manner, and given equal time alongside the essential introductory programming material. By exposing students to a broad range of specific computational problems we endeavour to make the introductory course more interesting and enjoyable, and instil in students a sense of vision for areas they might specialise in as computing majors.

Introduction

CS1 curricula are often developed from a core requirement that computing majors must be skilled programmers before they can undertake the more interesting and compelling topics of Computer Science in subsequent courses. This emphasis on *the machine* as the object of study serves only to implant the mistaken notion that "Computer Science is programming", and that taking up the subject as a major will inevitably result in years of staring bleary eyed at a computer screen hacking format directives over and over until columns of output are properly aligned.

The challenge for educators in any discipline is to present topics of interest in a manner that will capture the imagination of the learners and release the creative energies that will carry them through their programme of study, fostering a sense of enjoyment for what may develop into a lifelong career.

Introductory courses in Computer Science have a great many conflicting agendas to address. How to introduce programming to students who have no prior experience; how to correct for prior experience; how to motivate students who can program and who have acquired "good habits"; how to introduce abstraction as a key concept; how to introduce the subject of Computer Science, its scope, beauty, genius, and fun. Add to this list the issue of making the curriculum more inclusive for females, the religious and sometimes fanatical wars over programming language, and the need to encourage good written and oral communication skills and we have a problem as complex as any encountered in the subject itself.

This situation has not deterred attempts by educators to find a solution, many of which are adapted to local conditions [5, 6]. In this paper we present our local solution which we believe strikes a nice balance among these conflicting agendas by dividing all lectures, laboratories and tutorials into two parallel streams: one dedicated to teaching introductory programming, and the other to giving students exposure to a broad spectrum of computing concepts and computational problems. Sophisticated topics usually reserved for more senior level courses are presented in a manner that can be understood by those with little computing experience. Our hope is that this sneak preview of the subject gives students a better sense of what Computer Science is about, and puts them in a better position from which to make decisions about undertaking a major programme of study.

Background

At Waikato University, Computer Science majors must pass two 12 week semester courses in order to qualify for entry into second year. In this paper we concentrate on describing the first of these 12 week courses. Students enter the course with a variety of computing experiences, with the majority having had no prior computer programming experience. The objectives of the course are to introduce students to programming in C++, and to basic ideas and concepts relating to a wide variety of topics in Computer Science.

The course has per week: three lectures, two tutorials and between two and four hours of hands-on programming practice. The first lecture of the week is on programming, the second is on a Computer Science concept and the third is a review lecture in which the lecturer presents five or six small problems related to the previous two lectures. The students attempt the problems in the lecture, and then the lecturer, resisting the temptation to reveal the answers too soon, carefully goes through the solutions. In this way students can assess their progress with the material. The tutorials are split between programming and concepts.

The course is assessed by a combination of internal coursework and final examination (50% for each). The internal coursework is based on "laboratory live" practical tests, a small programming project, programming practicals and homework from the concept tutorials. In order to stress the importance of the work done in the concept tutorials and at home, we allocate almost half of the internal assessment to this activity.

The programming side of the course is to all intents and purposes fairly standard, and so we will only mention it further where necessary. There is an open question, however, as to whether it is important to bind the work on programming with the concepts. We will return to this question after we have described what we mean by a concept tutorial.

Concept Tutorials

The course contains ten concept tutorials on a range of Computer Science topics. The tutorials are dovetailed into the lecture program, and represent the practical consequence of the material presented in lectures. Students attend the tutorial where the practical material is introduced. They start to tackle the problems here where they can gain assistance from their tutor. Remaining work is taken home to complete, and their solutions are handed in at the next concept tutorial.

The tutorials cover (in order) the representation of numbers, data security and encryption, text compression, image coding, error-correcting codes, searching, sorting, graph problems, resource scheduling and systems analysis.

The programming material is presented to the students in small pieces and is drawn together in the final two weeks of the course alongside a programming project and material on systems analysis. The idea is to bind the notion of programming in the large with systems development. This is why systems analysis is the topic for the final concept tutorial.

The first five topics cover issues pertaining to representation, which we consider to be fundamental to an understanding of what Computer Science is all about and how programming is achieved on a computer. This material helps to cement programming ideas that are being presented in parallel on data types, variables, a basic data model of a computer and the principles of abstraction.

The second five topics are more concerned with algorithms and representing procedures. This material is presented in conjunction with programming notions such as control structures, functions, modularisation, arrays, and systems analysis to try to achieve the same effect.

Inspiration

The material that we have collected together in our concept tutorials is drawn and adapted from two excellent sources: A. K. Dewdney's *Turing Omnibus* [7] and an, as yet, unpublished book by Tim Bell, Ian Witten and Mike Fellows called *Computer Science Unplugged ... off-line activities and games for all ages* [8]. This latter work is aimed at presenting important topics in Computer Science, without using computers, to five to twelve year-olds. It is essentially a manual for teachers, presenting lessons on each topic to be conducted in the classroom.

We have adapted material so that the level of presentation in Bell et al's book is raised and the level in Dewdney's book is lowered (some topics in the "Omnibus" are at the appropriate level, but not all). In keeping with the "Unplugged" book, none of our concept tutorials involve computers. They are pencil and paper exercises which attempt to get across the ideas without the "threat" of programming a solution.

Examples

Example 1: Representation - Data Security

This tutorial begins by explaining the need to preserve confidential information when it is transmitted down a communications channel. It addresses the problem of transmitting text-messages and deals with encryption/decryption using ciphers and Huffman coding. The emphasis is on representation and is a good topic to address at the same time as programming material on character data types and the binary representation of characters.

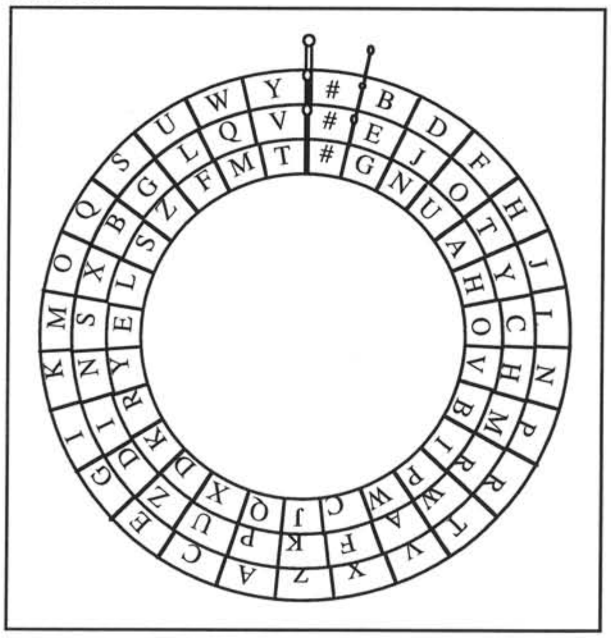

Figure 1: Two-Rotor Enigma Machine

Students are first shown an example of the k-th substitution[1] method using 27 letter English (A-Z plus a space character). A sequence is encoded, using $k = 4$, and then decoded. A discussion follows on how this type of encryption is vulnerable to statistical attacks, after which students attempt to decode a message using just such a method.

Avoiding statistical attacks is addressed by introducing a multi-rotor Enigma[2] machine.

[1] k-th substitution, sometimes called the Caesar cipher, works by encrypting each letter of the message with the k-th letter after it (modulo the size of the symbol set so that references beyond the last symbol wrap to the beginning again).

[2] To encrypt a letter using the Enigma machine in Figure 1, find the letter on the inside rotor and note the letter adjacent to it on the outside wheel (the backplate), then find that

An exercise is set in which students decode and then encode a message that we provide using the two-rotor Enigma machine shown in Figure 1. They do this by cutting out the rotors from an appendix in their manual (which has been printed on thick card) and pinning the three rotors together using a drawing pin (bent over at the back). Clearly, this is an opportunity in lectures to discuss the work done during the war by Alan Turing and his colleagues.

Once the general principles have been established we can talk about their realisation on a computer. For example, it is possible to discuss the relative ease with which the k-th substitution method can be programmed using a fixed offset of the ASCII values of the characters (this is possible, of course, because the characters are represented by binary numbers—the topic of the previous weeks concept tutorial). In order to make the connection more concrete, we follow the Enigma machine with an explanation of Huffman coding.

The advantage of Huffman coding is that it deals with the bit-level representation of a code in a file. It also presents yet another representation of characters—a representation which does not use a fixed number of bits per character. This is an example of a piece of lateral thinking in computing which leads to an extremely clever solution to the encryption problem (the story of how Huffman found this solution[3] is again excellent lecture material). This is precisely the kind of idea that makes Computer Science an interesting, fun and challenging subject. Further, the introduction of Huffman coding serves to introduce the next representation topic: data compression.

Example 2: Algorithms/Procedures - Graph Problems

The objective of this tutorial is to show students the connection between representation, algorithms/procedures that use the underlying representation, and real-world problem solving.

letter on the second rotor (i.e. the middle wheel) and output the one adjacent to it on the backplate. After a letter is encrypted, turn the inside rotor clockwise one step. Whenever the inside rotor returns to its original orientation, the second rotor turns in lock-step, just like the odometer in a car. Such a machine makes unauthorised decryption difficult by creating a new permutation of character-substitutions after each letter is encrypted.

[3] Legend has it that, as a student, Huffman's professor invited students to solve a related problem and receive an automatic A grade for the course or attend lectures and take the final examination. Huffman chose the former and on the eve of the exam (after many failed attempts) solved the problem.

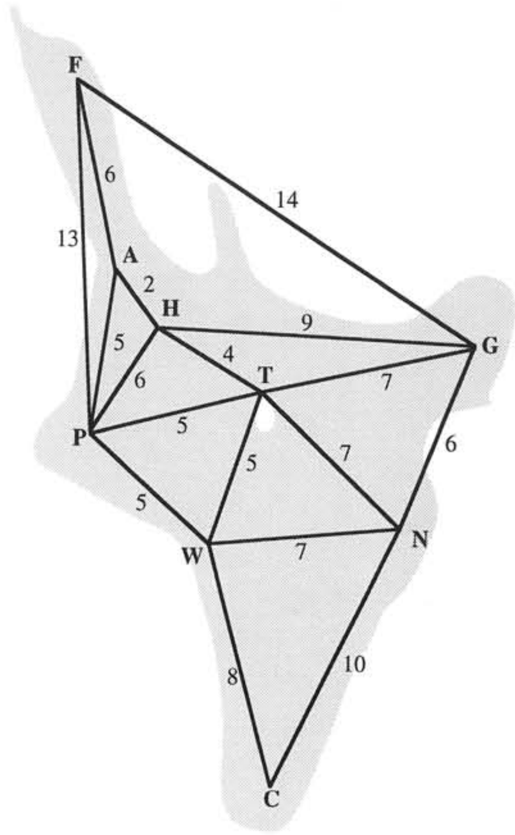

Figure 2: Fictitious Airline Network

We begin by explaining the concept of an acyclic subnetwork and a minimal spanning tree (MST[4]), and work through an abstract example showing how to construct a MST. The construction process is laid out as an algorithm, and students are encouraged to follow the steps and to answer questions such as whether there exists more than one MST for a given network.

From this abstract example we present a fictitious network depicting domestic air routes between major cities of New Zealand's North Island, shown in Figure 2. The students are then asked to construct a variety of MSTs for the network, and to decide which would be preferred if an airline wanted to minimise stop-overs.

The remainder of the tutorial is dedicated to describing Dijkstra's shortest path algorithm for the traveling salesman problem. Again we motivate the algorithm through the previous real world example and ask the question: What if a traveller wants to travel on the cheapest route between two cities?

Before tackling this question on the network of New Zealand cities, we present a careful step by step solution to a smaller, more abstract network, using Dijkstra's algorithm. Students are then asked to answer the traveller's question using the network of New Zealand

[4] A minimum spanning tree for a graph is a subset of its edges where 1) a unique path is maintained for all vertex-pairs, and 2) the total weight of these edges is the minimum possible for all subsets satisfying 1).

Tutorial	Activities
Representation of Numbers	Base conversion, 2's complement, binary addition/subtraction, develop an algorithm for base conversion
Data Security	Decode message using frequency/context analysis, build Enigma machine, encode/decode messages using Huffman coding
Text Compression	Perform Ziv-Lempel coding on stanza from Dr. Seuss, decompress "file" by hand
Image Coding	Draw picture on graph paper and run-length encode it, draw quadtree for a given picture (Dewdney's cat and dish), compare techniques by encoding original picture as a quadtree.
Error Codes	Describe errors in EFTPOS transmission, discuss problems with single-bit parity checking, find Hamming code for 5-bit numbers.
Searching	Calculate time taken to search telephone book, play a guess a number and determine how long it takes to find the number.
Sorting	Sort packs of cards using quicksort, radix sort, bubble sort and selection sort. Compare times and estimate times for large numbers of cards.
Graphs	Construct a minimal spanning tree for fictitious network of air routes around New Zealand, use Dijkstra's algorithm to find shortest path through network.
Resource Scheduling	Calculate cost of packing crates using first-fit, next-fit and best-fit algorithms.
System Analysis	Perform systems analysis of wholesale paper warehouse identifying key components and processes

Table 1: Summary of Concept Tutorials

cities and their new found knowledge of Dijkstra's algorithm.

Discussion

Table 1 provides a summary of topics and the activities that students have to do to complete each of the concept tutorials.

We use the concept tutorials to present a view of Computer Science which demonstrates that ideas and creative solutions rather than programming are the main focus of the subject.

If we bind these concepts to the programming portion of the course then we might compromise this distinction, and therefore lose some of the benefits of working away from the computer.

We prefer to delay the implementation of many of these concepts until the practice of programming has become more established—specifically at the third year level in a course on the design and analysis of algorithms. Even so, the exercises and tutorials from the programming side of the course are designed to incorporate some specific component of the concept being studied at the same time. In fact, for their final programming project students implement a specification they constructed during an earlier system analysis tutorial.

We realise that many of the students with prior experience might "breeze through" the programming material. It is highly unlikely, however, that they would have met all of the concepts. This group of students, more than any other, will need to be persuaded that Computer Science is not equivalent to computer programming.

In order to capture the imagination of the students who "breeze through" the concept material, and to extend them a little, we conclude each concept tutorial with some open questions (similar to the way Knuth uses starred exercises in his books [9]).

Conclusion

The material presented in this paper is being trialed for the first time in 1996. By the time of the conference in July we will have 300 student opinions to report. Ahead of this data, we believe that this approach to teaching Computer Science has great merit.

Firstly, it is exciting material to present to the students. Some of the material may well be the reason why the lecturer for the course studied Computer Science, and therefore, their enthusiasm for the subject should come across.

Secondly, elegant and neat solutions together with the ugly and brute-force can co-exist and be discussed without the burden of discussing the relative merits of ease of implementation, and without concern for the wide variation in programming ability, which must play some role in discouraging less experienced students from continuing in Computer Science.

Finally, the material is very flexible and adaptable to local conditions. We have a tutorial on text compression because state-of-the-art algorithms to perform this task have been the outcome of research at Waikato University [10]. This shows students that research in the subject is ongoing and achievable in a university thousands of miles from the main centres of Computer Science in North America and Europe.

References

[1] Connelly, Christopher and Biermann, Alan W. (1996) "Home-Study Software: Flexible, Interactive, and Distributed Software for Independent Study". In *The Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education*, pp 63–67, Philadelphia.

[2] Feldman, Todd J. and Zelenski, Julie D. (1996) "The Quest for Excellence in Designing Cs1/Cs2 Assignments". In *The Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education*, pp 319-323, Philadelphia.

[3] Reek, Margaret M. (1995) "A Top-Down Approach to Teaching Programming". *SIGSCE Bulletin, 27*(1) pp 6–9.

[4] Scragg, Greg and Baldwin, Doug and Koomen, Hans (1994) "Computer Science Needs an Insight-Based Curriculum". *SIGSCE Bulletin, 26*(1) pp 150–154.

[5] Paxton, John T. and Ross, Rockford J. and Starkey, J. Denbigh (1994) "A Methodology for Teaching an Integrated Computer Science Curriculum". *SIGSCE Bulletin, 26*(1) pp 1–5.

[6] Barrett, Martin L. (1996) "Emphasising Design in CS1". In *The Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education*, pp 315–318, Philadelphia.

[7] Dewdney, A. K. (1993) The (new) Turing Omnibus. Computer Science Press. New York.

[8] Bell, T. and Witten, I. H. and Fellows, M. (draft) "Computer Science Unplugged ... off-line activities and games for all ages".

[9] Knuth, D. E. (1973) The art of computer programming, 2nd edition. Addison-Wesley.

[10] Cleary, John G., Teahan, W. J. and Witten, Ian H. (1995) Unbounded context lengths for PPM. In Storer, J. A. and M. Cohn (eds). *Data Compression conference proceedings.* pp 52–61, Utah.