A High School Curriculum on Computer Systems Organization

João Pedro Franco Martiniano

Departamento de Informática, Universidade do Minho
4710 - 057 Braga, Portugal
mi7103@di.uminho.pt

Abstract. As the role of computers and related technologies continues to grow in our societies, the mere study of computer applications in high school is perceived as being insufficient. Therefore, this communication presents a curriculum for a one year course on computer systems organization, designed for use in the 11th or 12th grades. While not profound in depth, it has a relative broad scope, aiming to provide students with a scientific-oriented view of the field, while also trying to identify future developments.

1 Introduction

Current societies of developed countries are characterized by rapid rates of change, and accelerated life styles. These societies require a well prepared workforce, possessing not only solid knowledge on specific areas, but also, knowledge and skills on other areas. To be able to face modern challenges, these citizens are required to be able to adapt to change and to engage in life-long learning. Because of the central role that computers and related technologies play in modern societies, educational institutions are continuously challenged to educate a better kind of citizen: one which is capable of, not only understanding and using information technology to it's best, but also to actively update that knowledge. To achieve these goals it is crucial that curriculums do not focus narrowly on specific applications, vendors, or implementations, under the penalty of leaving students vulnerable to the problem of obsolescence [1]. Further, it is no longer sufficient for students to take courses on Information Technology, which are fundamentally devoted to study of computer applications.

Currently in the Portuguese educational system, high school students willing to study computers, can take a two-year course called *Introdução às Tecnologias da Informação*. This course is strongly focused on learning to use day-to-day computer applications (e.g. word processor, spreadsheet, etc.). Elementary students from 6th to 9th grades take a three year course that also covers some of those contents.

In the near future there will be a curricular change: students of the 9th and 10th grades will be required to take a course entitled *Tecnologias da Informação e Comunicação*. The 9th grade course provides an introduction to information technology, to graphical operating systems, the internet and word processing. The 10th grade course focuses on spreadsheets, databases and webpage development. This course also requires that students develop a project.

Neither of these approaches promote the study of computer science, which is fast becoming an essential requisite to comprehend our technological society.

This paper presents a curriculum for an optional one year-course on computer systems organization, accessible to all 11th or 12th graders, that have successfully concluded the

Tecnologias da Informação e Comunicação course, and wish to undergo further studies in the field.

2 Course Curriculum

It is very hard to design a course that both satisfies all of the students interests and expectations: typically, students that will take this course, will have different motivations and expectations on the learning outcomes. Some will be skilled users of computer applications (like word processors, spreadsheets, etc.), while others may be proficient programmers on one or more programming languages.

The proposed curriculum is based on the work developed by the Task Force of the Pre-College Committee of the Education Board of the ACM [2], and by the Joint Task Force on Computing Curricula of the IEEE Computer Society and the Association for Computing Machinery.

The course comprises the following knowledge areas:

- Algorithms
- Introduction to a Programming Language
- Computer Systems Organization

These knowledge areas are subdivided into topics, containing one or more learning objectives. Also, for each topic there is an indication of a minimum number of hours. The course uses a top-down approach, starting with algorithms, and ending with computer architecture, moving from a rather abstract view towards a more concrete view.

2.1 Algorithms

Algorithmic reasoning is fundamental to the understanding of computational processing. Therefore, this knowledge area introduces the concept of the algorithm, emphasizing the importance of analysis as an integral part of problem solving. The classes should be, as much as possible, based on problem solving exercises: students should be challenged with various problems and be required to develop algorithms to solve them.

Topics:

- Introduction to the concept of algorithm
- Examples of algorithms in the "real world"
- Introduction to the concept of programming language
- Pseudocode:
    - variables
    - assignment
    - expressions
    - operator precedence
    - input and output
    - conditional statement (if-then-else)
    - loops (repeat-until, while-do)

- Flow diagrams
- Debugging

Learning objectives:

- Understand what is an algorithm.
- Identify real-life examples of algorithms (for example, culinary recipes, following directions to a certain place, etc.).
- Identify simple mathematical algorithms.
- Develop algorithms to solve real-life problems.
- Understand what are programming languages and computer programs.
- Understand the concept of pseudo-code.
- Understand the concept of variable.
- Assign values to variables.
- Write simple expressions, using arithmetic operators.
- Input data (i.e., from the keyboard).
- Output data (i.e., to the screen).
- Perform conditional execution of instructions using the if-then-else statement.
- Understand the concept of loop and perform repetition of instructions using the repeat-until and while-do statements.
- Understand what flow diagrams are, and how they can help in developing algorithms.
- Understand the concept of debugging and debugging strategies.

Activities/Lab work

Students should develop two algorithms, to be used in real-life situations.

Students should write a minimum of six algorithms of growing complexity, using both pseudo-code and flow diagrams:

- assign values to two variables, perform a sum, store the result in a third variable and ouput the third variable
- given two numbers, find the greater
- calculate the factorial of a number
- etc.

2.2 Introduction to Programming

This knowledge area teaches students to program using a high-level language (Basic, Pascal, C, etc.). Students will be introduced to the fundamental concepts of programming, using the contents and skills that were covered in the preceding knowledge area without delving deeply into details: students aren't required to become proficient on a programming language but should understand the concepts and acquire the skills, which will facilitate learning other languages.

Topics:

- Introduction
    - history of programming languages

    - overview of programming paradigms (procedural languages, object-oriented languages, functional languages, declarative languages, scripting languages)
    - brief overview of existing computer languages (machine-code, assembly, C, Basic, Pascal, Java, etc.)
- Variables and scalar data types
- Operators
    - assignment operator
    - arithmetic operators
    - operator precedence
- Input/Output
    - screen output
    - keyboard input
- Comments
- Conditional structures
    - relational operators
    - equality operators
    - logical operators
    - if-then-else
    - select-case/switch
- Loops
    - repeat-until/do-while
    - while-do
    - for
- Arrays
- Functions

Learning objectives:

- Basic understanding of the evolution of programming languages, since the first computers to modern times.
- Identify different programming paradigms, and what distinguishes them.
- Be able to evaluate the characteristics for a given situation, which requires the use of a given language.
- Understand the concept of scalar data type.
- Identify the possible scalar data types that a given programming language supports.
- Declare variables.
- Understand the importance of choosing appropriate names for variables.
- Assign values to variables.
- Write expressions using arithmetic operators.
- Understand the rules of operator precedence.
- Perform simple output to the screen.
- Perform simple input from the keyboard.
- Understand the concept of comments and it's importance in maintaining code readability.
- Identify the relational operators of a given language: greater than, less than, greater or equal than, less or equal than.

- Identify the equality operators of a given language: equal to, different from.
- Identify the logical operators AND, OR, NOT.
- Write truth tables for each logical operator.
- Write logical expressions using the relational operators, equality operators and logical operators.
- Perform conditional execution of instructions using the if-then-else and select-case/switch statements.
- Understand the concept of loop and perform repetition of instructions using the repeat-until/do-while, while-do and for statements.
- Understand the concept of array.
- Manipulate arrays.
- Understand the concept of function.
- Understand the benefits of code modularization.
- Understand the concepts of function argument and return value.
- Understand the concept of function libraries.
- Use simple, built-in, functions (date/time, square root, etc.).
- Write functions.

Activities/Lab work
Desirably students should write some programs (from three to six) for each topic that is covered using, whenever possible, algorithms that were developed earlier. For example:

- when learning about input/output, write a program that reads a value from the keyboard into a variable and output the variable to the screen
- when learning loops, write a program that calculates the factorial of a given number
- when learning about arrays, write a program that reads some numbers and calculates the sum, the average, the greatest and the smallest numbers
- when learning about functions, create subroutines to modularize programs that have already been written

2.3 Computer Systems Organization

Topics:

- History of computer architecture
- Basic organization of the von Neumann machine
- Bits and bytes
- Representation of information
- CPU
    - Interpreters and compilers
    - Language translation
    - Virtual machines
    - Organization of the CPU
    - The role of the CPU
- Memory
    - The concept of memory

    - Memory hierarchy
- Bus
- The role of operating systems
    - Processes
- Network Systems
    - The concept of computer network and its benefits
    - History of computer networks
    - Network types: LAN, MAN, WAN
    - Client/server and peer-to-peer computing
    - Network topology
    - Analogic and digital signals
    - Modulation/demodulation
    - Transmission rates
    - Transmission media
    - …

Activities/Lab work
Resume an article on any of the topics covered in this knowledge area.

References

[1] IEEE Computer Society, Association for Computing Machinery: Computing Curricula 2001: Computer Science, http://www.computer.org/education/cc2001/final/cc2001.pdf, (2001)

[2] Association for Computing Machinery: ACM Model High School Computer Science Curriculum, http://www.acm.org/education/hscur, (1997)