Computer Science

A series of lectures for students of Environment and Resource Management

Prof. Dr. Winfried KURTH
Chair for Practical Computer Science /
Graphics Systems, BTU Cottbus

Office: Ewald-Haase-Str. 12/13, room 116
Phone (0355) 69-3816
e-mail wk@informatik.tu-cottbus.de

http://www-gs.informatik.tu-cottbus.de/~wwwgs/

Winter term 2004/2005

Lectures:

Tuesday, 7:30–9:00am, room SR3

Script to be found in the WWW under the address

http://www-gs.informatik.tu-cottbus.de/~wwwgs/cs1_lect.htm

Exercises:

Friday, 7:30–9:00am, room EH/213 (Ewald-Haase-Str.);
alternatively: Computer pool in EHS
Content of exercise units:

- Basic introduction to computer usage (first 2 weeks)
- Discussion of home work
- Repetition of key parts of the lectures
- Written test examination in December

Grading:

Written examination at end of semester term

2nd try: Beginning of April

next written examination after this: not earlier than February 2006

⇒

Recommendation:

Take continuously part in the exercises and prepare yourself for the examination *already during the semester term* – then it will be easy to have success already in February, and if not, you have the second chance already in April!

Schedule:

Lectures "Computer Science"

Prof. Dr. W. Kurth

Winter term 2004/2005, Tuesday, 7:30-9:00am, room SR3

19. 10.	Introduction; information, codes, number systems
26. 10.	Representation of information; computer architecture
2. 11.	Hardware; operating systems, programming paradigms
12. 11.	Programming languages, operator notations; propositional logic and Boolean algebra
16. 11.	From problem to programme; simple Java programmes (1)
23. 11.	Simple Java programmes (2)
30. 11.	Time complexity of algorithms
7. 12.	Basic algorithmic strategies and examples (1)
14. 12.	Basic algorithmic strategies and examples (2); image processing
4. 1. 2005	Working with databases (1)
11. 1.	Working with databases (2); Geographical Information Systems; the World-Wide Web
18. 1.	Automata and languages, grammars
25. 1.	Cellular automata and L-systems
1. 2.	Foundations of computer graphics, Virtual Reality, VRML
8. 2.	Software engineering; ethical aspects and Open Source
	Written examination

1. Introduction

Fundamental notions;
historical and systematical overview

What is "Computer Science" / "Informatics" ?

"Computer Science" – science about a tool?
better names would be: "*science of computing*"
or "*data processing science*"
(focuses on activity instead of tool)

"Informatics": continental-European for "computer science"
- French: "Informatique" (since 1960s)
- German: "Informatik"

Definition: "Science of the systematical processing of information, especially the automatic processing by use of digital computers".

Latin "informare":
to give structure to something; to educate; to picture

Information:
- independent fundamental entity of the world besides matter and energy
- depends on previous knowledge of the receiver of the information
- various approaches to quantify it
- we can consider information simply as "interpreted data".

Data: represented information
(e.g. text in a book, magnetic patterns on a harddisk, ...)

But:
Hermeneutics – "the art of interpretation" – is *not* part of informatics, despite its name. Social and cultural aspects of information are largely ignored.

"Computer": comes from "to compute" = "to calculate".

"*Algorithm*":
The word comes from the Persian textbook writer *Abu Ja'far Mohammed ibn Mûsâ* al-Khowârizmî (= "father of Ja'far Mohammed, son of Moses, coming from Khowârizm" – a town in Usbekistan, today called *Khiva*.)

Al-Khowârizmî lived in Bhagdad, in the "House of Wisdom"
wrote book about calculation:
"Kitab al jabr w'*al-muqabala*" (= "rules of reconstitution and reduction")
– here the word "algebra" comes from!

Modern meaning of "algorithm":
Finite set of rules which specify a sequence of operations in order to solve a certain problem, with the following properties:

1. **Termination**: An algorithm must come to an end after a finite number of steps.
2. **Definitness**: Each step must be defined precisely.
3. **Input**: An algorithm *can* need input values (e.g. numbers).
4. **Output**: An algorithm *must* give one or more output values.
5. **Feasibility**: An algorithm must be feasible; e.g., no intermediate step must depend on the solution of some still unsolved mathematical problem.

(after Knuth 1973)

"*Programme*" (in American English: "program"):

Version of an algorithm which can be read, interpreted and carried out by a computer.

Programming languages were designed to write precise programmes (more precise than possible in our natural language!) suitable for computers.

History of computer science:

Place-valued decimal number system:
Helps to perform complicated arithmetical operations by simple steps. Example: Multiplication of large numbers.
Decimal number system came from *India* via the *Arab* world to Europe:

Leonardo of Pisa, called Fibonacci (1170–1250)
- was educated in North Africa where his father held a diplomatic position
- recognised the enormous advantages of the decimal system
- Book *Liber abaci* (1202): introduced Indian-Arabic number system to Italy

Germany:
Adam Ries(e), 1492–1559
- popularised the Indian-Arabic numbers in Germany
- Book "Rechnung auff der linihen vnd federn" (1522)

Since the 17th century in Europe: *Machines* which should help with computations.

Wilhelm Schickard (1592-1635, Germany): Professor of astronomy, mathematics and Hebrew in Tübingen. Had built, around 1623, a mechanical machine for doing automatically the operations of addition and subtraction, and partially automatically multiplication and division. Died during plagues of the Thirty Years War. Work remained forgotten for a long time.

Blaise Pascal (1623-1662, France): Mathematician and philosopher. Built a mechanical computing machine around 1643. **Addition and subtraction.**

Gottfried Wilhelm Leibniz (1646-1716, Germany): Philosopher, mathematician and historian. Had, around 1673, a machine built which could fully automatically multiply and divide, in addition to add and subtract. **Propositional logic. Binary number system.** Tried to develop a symbolic logic as a way to **mimic thought by way of computation**.

Charles Babbage (1791-1871, England): Many ideas, technically not viable at the time.

"Difference engine". Goal: A machine for computing tables for polynomials up to degree six. To be driven by a steam engine.

"Analytical engine": first general purpose computer concept.

Ideas from the **Jacquard loom** (French invention, automation in weaving). Basis: Series of cards with punched holes representing the weaving pattern.

Hermann Hollerith (1860-1929, USA): Census of 1890 supported by punched-card encoding of data.

Alan Turing (1912-1954, UK): Most influential mathematical base concept ("Turing machine", not a physical machine, but a mathematical idealization of a completely general computing machine)

During WW II, Turing worked on breaking the military code of the German ENIGMA cipher machines.

WW II: Electromechanical computers (relays, current-driven magnet switches)

Specialized computers for:

calculations in development of atomic bomb (USA)

breaking the military ENIGMA codes (UK)

calculations for measuring wings of war planes (Germany)

Konrad Zuse (1910-1995, Germany): First electro-mechanical computer operational in 1941 ("Z3"), using punched cards for input. 2600 relays. Multiplication time: 3s.

John von Neumann (1902-1957, Hungary, later USA): Mathematician. Worked on the atomic bomb from 1943 onward. Developed concept of stored-program architecture of computer, which is in general use today, today called Von-Neumann-architecture.

Howard Aiken (1900-1973, USA): 1944 the MARK I is ready, also electro-mechanical. Multiplication time: 6s.

J. P. Eckert and J. W. Mauchley (USA): 1946 the ENIAC is ready, using "electronics", i.e. not electro-mechanical parts, but 18,000 vacuum tubes. Multiplication time: 3ms. The speed difference with respect to electro-mechanical devices meant an end to them.

From 1950s: development speeds up.

Period 1: 1953-1958: **Vacuum tubes**

Period 2: 1958-1966: **Transistors;** core memory; starting high-level languages and compilers; batch use, with punched card decks and printouts as user interface

Period 3: 1966-1974: **Integrated circuits (ICs);** cheaper hardware, thus: software became a cost factor; time-sharing starts (CRT-Terminals); few experts

Period 4: 1974-1982: **Highly integrated circuits (LSI)**: CPUs on a chip (130000 transistors on a single chip); Home Computers; LANs start; Modems for connecting computers over phone lines

Period 5: 1982-1990: **Very highly integrated circuits (VLSI)** (1 Mio. transistors on a chip); used in industry; GUIs

Period 6: Since 1990: **10 Mio. transistors on a chip;** more Personal Computers; more embedded systems (cheap enough for toasters); standard-software-components; WWW becomes general

(Rust 2003)

Another history: that of *programming*

Early phases of computer history: *Hardware* (= the machines) was in focus (reason for the name "computer science")
Later: *Software* (= programmes) increasingly important, increasingly expensive in comparison to hardware.

First "programmer": Was a woman (**Lady Ada Lovelace**, daughter of the poet Lord Byron): Developed programs for Babbage's (non-functional) "analytical engine"

An early concept for a programming notation was the "Plankalkül" (Zuse 1944), but it was not used in practice.

Programming these machines: Started with today so-called **"machine languages"** and **"assembler languages"** (both machine-specific).

Later: so-called "high-level languages"
- more abstraction
- better readability for humans
- trying to integrate traditional mathematical notations
- *platform-independent* (not specific to certain machine)

FORTRAN (1954), COBOL (1958), LISP (1960), Pascal (1971), C (1971), C++ (extension of C, 1992), Java (1995) ...

(see Chapter 4, later)

Subject areas of computer science

1960s/1970s: Development of specialized university curricula

Basis: Mathematics, electrical engineering; no interest in social or cultural conditions and consequences, or more specifically: in consequences for life at working place and leisure

Classical branches (from first recommendations for curricula in the 1960s): (a) **theoretical** informatics, (b) **technical** informatics, (c) **practical** informatics, (d) **applied** informatics

Theoretical informatics: mathematical basis: not general “theory” (which would include disciplines from the humanities and social sciences relevant to informatics), but specialized “mathematical base”. Example questions:

Which problems **can in principle be solved** by a machine?

How can **syntax and semantics** of programming languages be described?

Which kinds of **logic** can be used for automatic problem solving?

How do we measure **how complicated problems are**, for example with respect to time or memory requirements?

Which kinds of problems can be solved with which abstract models of computation?

How can be the **correctness** of a program be **proved** with mathematical exactness?

Technical informatics: focused on **hardware**. Example questions:

How can computational objects and operations be represented with **physical means**?

Which are the **basic parts** from which a computer should be built?

Which are the appropriate **architectural decisions** for a computer?

How can a processor be organized in order to execute a special kind of program especially quickly?

How is information **stored** for quick access with small cost?

Which are the technical conditions for building **networks** from separate computers?

How do we build computers which **survive some defects**?

Practical informatics: **non application specific programming**. Example questions:

Which are the standard problems occurring **in many application areas**, and how can they be solved?

Which **data structures** allow efficient solving of problems, and which algorithms are best used on these data structures?

What types of **programming languages** are best suited to different types of problems?

How must **service programs** be organized which provide the user with an easier to use view of the machine than the bare hardware would do?

How are high-level programs **translated** into a form which can be executed by the underlying hardware?

How does one design **user interfaces** for end users?

How does one organize the **development process** of large software systems? ("Software engineering")

Applied informatics: programming for **specific application fields**. Example questions:

How are **graphical objects** represented in the computer, and how can the be visualized?

Which **numerical methods** exist to model states and processes happening in natural environments?

How should **data base systems** be structured to support the work processes in a company?

Which techniques exist to simulate the working of the **human mind** with computers?

What consequences has the use of computers for the **quality of life**, both in general and at the working place in particular?

Informatics in the social context:

What **ethical questions** arise from the use of computers, and how can they be answered?
(data security, privacy questions, computer viruses, hackers, violence-promoting games, software piracy, ownership of software and ideas, use of information technology for warfare, for crime, for sexual exploitation, for terrorism...)

How does the use of computers influence our **way of thinking** (about the world, about humans, about the mind, about personal relationships of people...)?

How can computers and the WWW be used to improve education / autonomy of people / human rights / political participation... ?

2. Codes and number systems

Normally information is represented **digitally** in computers, and for almost everything, **bits** are used to model the alternatives.

Bit: Binary digit. Expresses which of two alternatives is the case. Alternatives are often written 0 and 1, or as 0 and L.

Binary codes: two-element codes

Technically: many alternatives. Typical: **relays** (open/closed), **vacuum tubes**, **transistors** (each can be 'on' or 'off'), **condensators** (charged/not charged), **ferro-magnetic material** (magnetized/not magnetized).

n bits: represent one of 2^n alternatives

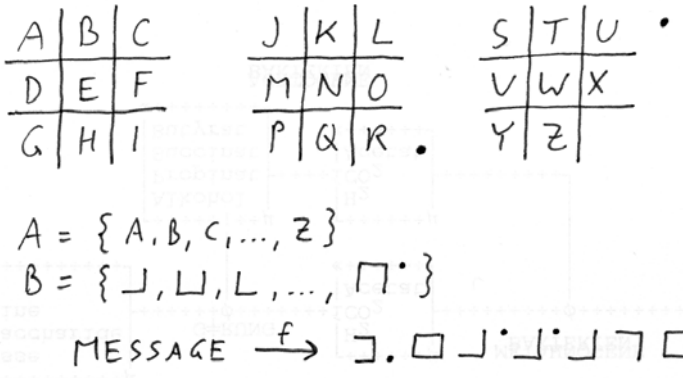
$\Rightarrow$ To represent information in a computer, we must *encode* all with the two symbols 0 and 1 !

What is a *code* ?

Code (1): A mapping f: $A \rightarrow B$ from a set A of elements to be stored or transferred to a set B used for storage or transfer.

Code (2): The set B from definition (1).

Example:



digital (discrete) and **analogue** (continuous) codes

Analogue computers (representation of quantities with continuously changing quantities): have vanished

Example: Vinyl records (analogue) vs. compact disks (discrete)

Benefit of discrete data representations: avoiding **noise**

For digital computers, we need *binary* codes:
B is a set of combinations of 0 and 1.

Examples:

For the primary **compass direction**: two bits necessary, and some convention which bit-pair represents which direction. Example code:

$$\{N, E, S, W\} \rightarrow \{0,1\}^2, N \mapsto 00, E \mapsto 01, S \mapsto 10, W \mapsto 11$$

For **Boolean** values 'True' and 'False':

$$\{T, F\} \rightarrow \{0,1\}, T \mapsto 1, F \mapsto 0$$

For **numbers** 0 to 9: Binary Coded Decimal (BCD, non-total code, i.e. some combinations are unused)

$$\{0,1,\ldots,9\} \rightarrow \{0,1\}^4$$
$$0 \mapsto 0000, 1 \mapsto 0001, 2 \mapsto 0010, 3 \mapsto 0011,$$
$$4 \mapsto 0100, 5 \mapsto 0101, 6 \mapsto 0110, 7 \mapsto 0111,$$
$$8 \mapsto 1000, 9 \mapsto 1001$$

Multiples of bits

Bits seldom occur as singles. Certain multiples of bits are used as *units for information (storage) capacity*.

1 Byte: 8 bits (can represent 1 of $2^8 = 256$ alternatives).
Example: one of the integer numbers between –128 and +127.
1 Halfbyte: 4 bits.

Typically, memory stores are built for *multiples of bytes*.

Prefixes: kilo, mega, giga, tera, peta, exa
- used in physics for the factors 10^3, 10^6, 10^9, 10^{12}, 10^{15}, 10^{18}
- in computer science often used for the factors 2^{10}, 2^{20}, 2^{30}, 2^{40}, 2^{50}, 2^{60}, which are slightly larger

abbre-viation	meaning	factor
KB	Kilobytes	$2^{10} = 1024$
MB	Megabytes	$2^{20} = 1{,}048{,}576$
GB	Gigabytes	$2^{30} = 1{,}073{,}741{,}824$
TB	Terabytes	$2^{40} = 1{,}099{,}511{,}627{,}776$
PB	Petabytes	$2^{50} = 1{,}125{,}899{,}906{,}842{,}624$
EB	Exabytes	$2^{60} = 1{,}152{,}921{,}504{,}606{,}846{,}976$

Number systems

Decimal number system: base 10; each digit represents a multiple of an exponent of 10. Digits 0..9.

Example: $123.456_{10} = 1 * 10^2 + 2 * 10^1 + 3 * 10^0 + 4 * 10^{-1} + 5 * 10^{-2} + 6 * 10^{-3}$.

Binary number system: base 2. Only two digits: 0 and 1.

Example: $1101.01_2 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} = 13.25_{10}$.

Hexadecimal system (better but unhistorical name: sedecimal number system): Base 16, digits 0..9,A..F. One digit for four bits. Examples: $A2.8_{16} = 162.5_{10}, FF_{16} = 255_{10}$.

The additional digits in the hexadecimal system:
A = 10, *B* = 11, *C* = 12, *D* = 13, *E* = 14, *F* = 15.

Transformation from one number system to the other:

- Special case (easy): from binary to hexadecimal

Each halfbyte corresponds directly to a hexadecimal digit

Example: 0000 0010 1100 0110
→ 0 2 C 6

- from arbitrary system to decimal: *Horner scheme*

Input: $z_{n-1}\ z_{n-2}\ \ldots\ z_0$ to base b
start with $h_{n-1} = z_{n-1}$
calculate for $k = n-1, n-2, \ldots, 0$: $h_{k-1} = h_k * b + z_{k-1}$
Output: $z = h_0$

- from decimal to arbitrary: *Inverse Horner scheme*

start with $h_0 = z$ (= input)
calculate for $k = 0, 1, 2, ...$:

$$z_{k-1} = h_{k-1} \bmod b,$$
$$h_k = h_{k-1} \operatorname{div} b$$

(mod: rest when dividing by b, div: integral part from dividing by b)

Output: $z_{n-1}\ z_{n-2}\ ...\ z_0$ to base b