Computer Science for Engineers

Lecture 2

Basics

Prof. Dr. Dr.-Ing. Jivka Ovtcharova
Dipl. Wi.-Ing. Dan Gutu
6th of November 2009
1. Introduction

2. Basics
   2.1. Information representation and processing
      2.1.1. Alphabet - Data - Signals - Information
      2.1.2. Coding – number systems
      2.1.3. Boolean algebra and propositional logic
      2.1.4. Computer architecture
      2.1.5. Introduction to data structures
      2.1.6. Introduction to algorithms
   2.2. Programming paradigms
      2.2.1. Term programming
      2.2.2. Procedural programming
      2.2.3. Object-oriented programming

3. Object Orientation
In order to use a computer to process information, the data must be in a **usable form**.

A **data representation** is created through defining an **alphabet** such as:

**Definition 1.2:** An alphabet is a collection of valid symbols in a language.

**Example**

Alphabet $\alpha = \{a, b, c, d, e, f, g, h, i, m, n, p, r, s, t, u\}$

Valid words in the language using alphabet $\alpha$ would be:

„computer“, „science“, „not“, „is“, „fun“. 
Definition 1.3: A signal is the representation of a message by the temporal modification of a physical value

- **Signal parameter**: modifiable property of a signal
  - Example: Frequency or amplitude of a wave
- **Signal transmission**: passing a message
- **Persistent inscription**: durable representation of a message
- **Storage medium**: physical carrier of an inscription

Computer Science is based on signals, inscriptions and their processing
Definition 1.4: Data is a sequence of signals, linked by meaningful context

- We speak about data, when abstraction can be made of its usage
- A signal is the smallest accessible data element when a program is executed [Tami04].
- In a hierarchy, signals are beneath data
Definition 1.5: Informations are interpreted data, when seen in the context of a problem and used to reach a goal

- Data becomes information, when it is seen in a certain context
- Information is strongly linked to its usage
Definition 1.6: Knowledge is the information how to interpret data

- Knowledge by facts: Knowledge of the information given by data
- Procedural knowledge: Knowledge on interpretation rules for the creation of information

Examples:
- The multiplication table is knowledge by facts
- Computation of the multiplication table is procedural knowledge

Engineers must deal with both kinds of knowledge
• Precondition for data and information processing:
  - Fragment the information into signs or signal chains, ex.
    ▪ Fragment a text into words and letters
    ▪ Fragment a picture into picture elements

• Representation and transmission of information units in different ways:
  - Speech: acoustic signals
  - Storage in a book: graphic symbols (letters)
  - Processing by a computer: electromagnetic signals
The translation of a data representation into another form can be performed through the use of a code.

**Definition 1.4:** A code is a defined method of transforming the characters of one alphabet into the characters of another.

**Example**

- A → Coding definition → • –
- B → „Morsecode“ → – • •
Coding Example: Voice-over-IP (VoIP) Telephony

- Different data representations must be able to be transformed between one another, so that exchange of data between different systems is possible.

Data transfer takes place most using **TCP** (Transmission Control Protocol) / **IP** (Internetwork Protocol) – Industry standard network technology.
Elementary Data Representation

- The elementary data entity of a digital computer is the **Bit**.
- **A bit can take one of 2 states**, defined by 0 and 1. Thus the alphabet of a digital computer is:
  \[ \alpha = \{0, 1\} \]

**Example**

<table>
<thead>
<tr>
<th>Light switch</th>
<th>Punch card</th>
<th>Digital Logic</th>
<th>Morsecode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: current &gt; 0</td>
<td>hole</td>
<td>voltage &gt; 3V</td>
<td>• („short“)</td>
</tr>
<tr>
<td>0: current = 0</td>
<td>no hole</td>
<td>voltage &lt; 2V</td>
<td>– („long“)</td>
</tr>
</tbody>
</table>

- The 0 and 1 represent the basics of digital computer language (dual system/ binary number system)
The general equation for the number system is:

\[ a = \sum_{i=0}^{n-1} z_i B^i \quad (n \geq 0, n \text{ integer}) \]

with \( B \) as the base and \( z_i \) (\( 0 \leq z_i < B \)) as respective numbers.

The most important number systems are:

<table>
<thead>
<tr>
<th>Number system</th>
<th>Base</th>
<th>Possible numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual system</td>
<td>2</td>
<td>0, 1</td>
</tr>
<tr>
<td>Octal system</td>
<td>8</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Decimal system</td>
<td>10</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>Hexadecimal system</td>
<td>16</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F</td>
</tr>
</tbody>
</table>

(The letters A-F stand for the values 10-15)

Source: Bronstein, Taschenbuch der Mathematik
Convert „Binary representation → Decimal representation“

- **Calculation:**

\[
(b_{n-1} \ldots b_1 b_0)_2 = \left( \sum_{i=0}^{n-1} b_i 2^i \right)_{10}
\]

- **Example:**

\[
(101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 4 + 1 = (5)_{10}
\]

\[
(10110)_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 16 + 4 + 2 = (22)_{10}
\]
Converting to other number systems (2)

• Convert „Decimal representation → Binary representation“:
  - Divide the decimal number by 2 and round up until you reach 0. At each step note the remainder (0,1).
  The binary representation is produced from the noted remainders (in reverse order)

  - Example: \((22)_{10}\)

  \[
  \begin{align*}
  22 : 2 & = 11 \quad \text{remainder 0} \\
  11 : 2 & = 5 \quad \text{remainder 1} \\
  5 : 2 & = 2 \quad \text{remainder 1} \\
  2 : 2 & = 1 \quad \text{remainder 0} \\
  1 : 2 & = 0 \quad \text{remainder 1}
  \end{align*}
  \]
  \[
  \Rightarrow (22)_{10} = (10110)_{2}
  \]
• Digital circuits are created „top-down“
• Several mathematic procedures are necessary for chip development:
  - Logic description
  - Minimization
• The basis for the **binary and logic description** of circuits gives a close coherence between boolean functions and propositional logic formulas
• With these description methods, circuits can be developed, minimized and analyzed
Boolean Algebra

• A computer built on the binary numbering system is able to perform logic operations with the help of a logic processing unit.
• Mathematics describes this using Boolean algebra.

**Definition 1.5:** Boolean Algebra consists of a carrier $A$, for which the following operations are defined:

$$\land : A \times A \rightarrow A$$

„and“

$$\lor : A \times A \rightarrow A$$

„or“

$$\neg : A \rightarrow A$$

Negation

• All other complex operations performed by a computer are built upon these basic operations.
## Rules of Boolean Algebra

### Elementary Rules

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Rule Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Involution rule</strong></td>
<td>( \neg (\neg f) = f )</td>
</tr>
<tr>
<td><strong>Commutative rule</strong></td>
<td>( f \land g = g \land f ) \quad \quad \quad f \lor g = g \lor f )</td>
</tr>
<tr>
<td><strong>Associative rule</strong></td>
<td>( (f \land g) \land h = h \land (g \land f) ) \quad \quad \quad (f \lor g) \lor h = h \lor (g \lor f) )</td>
</tr>
<tr>
<td><strong>Idempotency rule</strong></td>
<td>( f \land f = f ) \quad \quad \quad f \lor f = f )</td>
</tr>
<tr>
<td><strong>Absorption rule</strong></td>
<td>( f \land (f \lor g) = f ) \quad \quad \quad f \lor (f \land g) = f )</td>
</tr>
<tr>
<td><strong>Distribution rule</strong></td>
<td>( f \land (g \lor h) = (f \land g) \lor (f \land h) ) \quad \quad \quad f \lor (g \land h) = (f \lor g) \land (f \lor h) )</td>
</tr>
<tr>
<td><strong>De Morgan’s law</strong></td>
<td>( \neg (f \land g) = \neg f \lor \neg g ) \quad \quad \quad \neg (f \lor g) = \neg f \land \neg g )</td>
</tr>
<tr>
<td><strong>Neutrality rule</strong></td>
<td>( f \land (g \lor \neg g) = f ) \quad \quad \quad f \lor (g \land \neg g) = f )</td>
</tr>
</tbody>
</table>

### Created Rules
Propositional Logic

- The **propositional logic** works with **elementary terms** that can have the values "true" (T=1) or "false" (F=0)

- Formulas of propositional logic:
  - Elementary terms are formulas
  - If $M$ and $N$ are formulas, then $(M \land N)$ and $(M \lor N)$ are formulas, too.
  - If $M$ is a formula, then $\neg M$ is a formula, too.
Propositional logic operations

- **And-operation:** \( a \land b \)
  - The term is true, if \( a \) and \( b \) are true

- **Or-operation:** \( a \lor b \)
  - The term is true, if at least \( a \) or \( b \) are true

- **XOR-operation** (exclusiv or):
  - The term is true, if either \( a \) or \( b \) are true.

<table>
<thead>
<tr>
<th>( a )</th>
<th>( b )</th>
<th>( a \land b )</th>
<th>( a \lor b )</th>
<th>( a \oplus b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Correlation: Boolean Algebra and Propositional Logic

- The correlation is realized by the **equalization** of the binary character set \{0,1\} with the set \{F,T\}
- Each Boolean function can be transformed into an equivalent logic expression
  - **Circuit synthesis**: building a circuit from a boolean function
- The contrariwise transformation is possible, too
  - **Circuit analysis**: an existing circuit shall be examined

Example: Mechatronics

[LeRe03]
Computer architecture means:

- The **internal structure** of the computer, its components
- The **organisation of the activities** in the computer

Diagram:

```
Computer Architecture

von Neumann-Architecture  Parallel Architectures

  Vector computers  Multiprocessor Systems
```
von-Neumann-Architektur

- Von Neumann (1903-1957) created the first electronic computer, with the running program stored in its memory
- Today, all computers are based on the von Neumann principle
- Components
  - **Processor** (Arithmetic-Logic Unit (ALU), control unit, that „executes programs“)
  - **Main Memory** (stores instructions and data)
  - **Peripheral** (Input / Output devices (ex. printer, mouse, monitor, harddisk)
  - **System bus** for interconecting the components
Functional units of a von-Neumann-Computer

Processor (Central Processing Unit): executes machine-coded programs
- The program is a sequence of machine instructions
- Machine instructions define operations

The main memory stores machine instructions and the required data

System bus (bidirectional): interconnects existing components, transmits data, instructions, control signals; bus allocation depending on the devices priorities
Von-Neumann-Architecture: working principle

1. Instruction are fetched from the main memory to the processor (read access)
2. Decode instruction, possibly read operand from the main memory or the computers peripherals
3. Execute command, possibly write to the main memory or the peripherals (write access)
4. Restart with step 1

- Computer executes commands permanently
- Computer is a reactive system
Von-Neumann bottle neck

- Processor executes the instructions 10 to 100 times faster than memory access is

von-Neumann – bottle neck:

- Work-around:
  - Registers: Memory on the processor
  - Buffer storage: (engl. cache) fast storage
  - separate data and bus instructions
Organisation of the main memory (today)

- Organized as a sequence of storage cells of size of 8 bits (1 byte), counting starting with 0

**Address:**
- Number of the storage cell

**Direct memory access:**
- Peripherals access the memory directly (no detour over the processor)

- Several peripherals (printer, network card) have their own processor

**Observation:** 98% of the processor sold today are employed in embedded systems (especially in cars, TVs, telecommunication devices, …)

**But:** Intel delivers 2% of the rest and makes 90% of the profit
• A **data structure** is a special way to administer and interlink data, in order to be able to access and manipulate this data properly. Data structures are always interconnected with operations.

• The **definition** of data structures is done by a **concrete specification** of the data management and of the required operations. This concrete specification determines the general behavior of the operations and prescinds it from the concrete implementation of the data structure [Wik104].

• Data structures permit to model and implement the relations between data in a way understandable by humans

• Data structures are of fundamental relevance for the practical usage of programming languages by humans. [Webe98].
An example is the simplified representation of a product structure in PDM-system. In this example, a product is made of several parts, that are described by the attributes geometry, weight, costs and version number.

**Scheme**

- **Product**
  - contains
  - **Part**
    - Geometry
    - Weight
    - Costs
    - Version number

**Scheme specification**

- **Window fay**
  - contains
  - **cover**
    - coverGEO
      - 0,4 kg
      - 3,67 EUR
      - VerNr. 0,91

- **Rotor**
  - Rotor GEO
    - 1,8 kg
    - 37,16 EUR
    - VerNr. 1.40

**Legend**

- Entity Typ
- Relation
- Attribute
Example of a data structure (2)

2. Basics
2.1 Information Representation and processing / 2.1.5. Introduction to data structures

L1 WINDOWFAY NEW AS
L2 AE_ACTUATORUNIT
L3 AE_BOTTOM PART
L3 AE_MECHANIC A
L3 AE_ACTUATOR
L3 AE_BOX P
L2 PE_ROTOR ASSEMB
L3 PE_ROTOR_BOT
• In several domains, a solution must be found to a technical problem
• For creating this solution, an amount of **activities** is required, meaning that a process is running
• To run this process, a **processor** (human, automat, machine, computer) is required.
• A **connection** must exist to this processor. **Communication devices** (speech, button press, move lever, machine language) are used for this purpose.
• A problem is solved, if the solution path is given in a format understandable by the processor.
**Definition 1.9**: An algorithm is a **precise** (meaning written in a determined language), **finite** description of a general procedure with the usage of executable **elementary** steps.

**Characteristic properties of algorithms**

- **precise description** → **unambiguous communication**
- **elementary step** → **sequential or parallel execution**
- Described with a finite number of steps → **Algorithm is finite**
- The result is always unambiguous → **Algorithm is determinate**
Views on algorithms

**Functional view:** what: the problem to be solved by the algorithm

**Operational view:** how: the modality, how the task is solved
- Elementary step
- Description of the processing
- Data and parameters, that are processed by the algorithm

For an algorithmic solvable problem (functional view) there exist different solution paths (operational view)

**Therefore:** differentiate the specification of a task from its algorithmic solution
Elements that describe an algorithm

Elements:

- Use a **(primitive) function**, or
- **Sequence**: a consecution of steps

Usage:

1. **Loop**: repeated execution of a function or a sequence
   2. **Condition**: conditionated execution of a function, sequence or loop
1. Introduction

2. Basics
   2.1. Information representation and processing
      2.1.1. Alphabet - Data – Signals- Information
      2.1.2. Coding – number systems
      2.1.3. Boolean algebra and propositional logic
      2.1.4. Computer architecture
      2.1.5. Introduction to data structures
      2.1.6. Introduction to algorithms
   2.2. Programming paradigms
      2.2.1. Term programming
      2.2.2. Procedural programming
      2.2.3. Object-oriented programming

3. Object Orientation
What is programming?

• **Programming** is an activity in which a specific problem is tried to be solved through **systematic utilization of a given programming language**

• A program is an **algorithm verbalized into a programming language**

• 2 types of **programming languages**:
  - **Machine language**, esp. assembly language: optimized towards a specific processor
  - **Higher level programming languages**: allow a problem-oriented formulation of a designed algorithm

• Programmers use **higher level programming languages** when developing applications. In the programs (interpreted or compiled), the **source code is automatically translated from computer instruction to machine instructions**, so that ultimately the source code can be worked upon by the **hardware** (the processor)
Basics about programming

- Using a certain computer serves predominately to complete a **purpose** (i.e. construction, navigation of airplanes, text processing, money transfers, or simply for computer games)

- One of the deciding prerequisites of use is the computers’ **error-free performance** and it's **intuitive** operation mode
  
  - The computer must be designed in such manner that the user doesn’t have to program the computer. The user only must use the computer and not program it
  
  - The programmer writes the order of instructions with the help of high level programming languages and software development tools. These instructions build the application program
  
  - The system programmer sees the computer from a different point of view. He prepares the software tools, compiler, interpreter, and help menus that serve the programmer
  
  - The computer hardware changes the order of machine commands into micro-order for the control of single hardware components
Different computer perspectives

- **perspective**
  - user
  - programmer
  - system programmer
  - hardware engineer

- **program**
  - User program
  - Compiler, editors, and software tools
  - Machine commands
  - Micro program
  - Hardware

- **example**
  - if Liststart ≠ nil then begin (the list is not empty)
    - indicator := Liststart;
    - while indicator^.next ≠ nil . . .
  - Address command
  - loop: CMP W I0, ! R0
  - JEQ list_ende
  - MOVE W ! R0, R0

- **example tools**
  - CATIA
  - AutoCAD
  - ProEngineer
  - CATIA
  - SmarTeam
  - SAP
  - TeamCenter

---

Prof. Dr. Dr.-Ing. Jivka Ovtcharova – CSE-Lecture – Ch. 2 - WS 09/10 - Slide 37
Historical development of important programming languages

1958
1959
1964
1965
1967
1968
1969
1970
1971
1972
1983
1985
1986
1987
1988
1994
2002

FORTRAN
LISP
ALGOL
BASIC
LOGO
SIMULA
BCPL
PASCAL
SMALLTALK
C
Java
C++
C#

Imperative
Functional
Logic based
Object oriented
procedural

[Fara04]
Programming Paradigms

- **Paradigma** = pattern of thought, superior principle
- **Programming paradigm** = basic appendages, how you can manage a computer through programming

<table>
<thead>
<tr>
<th>Paradigma</th>
<th>In the foreground are...</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>imperative</td>
<td>Variables, that correspond to one or more storage cells of a computer, and instructions as abstractions of the processor functions</td>
<td>Assembly languages, FORTRAN, ALGOL</td>
</tr>
<tr>
<td>procedural</td>
<td>Procedures (under programs), to which arguments (parameter) are assigned. The procedures calculate the worth according to an algorithm and return this value</td>
<td>Pascal, C</td>
</tr>
<tr>
<td>functional</td>
<td>(mathematical) functions, that assign a value to a vector of parameters (also functions)</td>
<td>LISP, LOGO</td>
</tr>
<tr>
<td>Logic based</td>
<td>Logical expressions, that in general contain free variables, and that will be verified through an appropriate binding of these variables</td>
<td>PROLOG</td>
</tr>
<tr>
<td>Object oriented</td>
<td>autonome, interaged objects, that communicate through messages and are grouped with other similar objects in classes</td>
<td>Smalltalk, C++, Java, C#</td>
</tr>
</tbody>
</table>
Definition 2.0: A procedure is a sequence of instructions, that represent an autonomous part of the program, that can be called from another autonomous part of the program.

- A procedure has an interface, that specifies its name, its allowable input objects (also called arguments) and its result (the resulting data after running the procedure).
- The procedural programming paradigm extends the imperative programming by abstraction mechanisms for the creation of procedures (subprograms), as well as means of expression for the iteration and selection.
In the program example, a global variable \( v \) of type "Integer" and 2 global procedures (subprograms, subroutines) are declared.

The 2 procedures in this example perform a well-known algebraic function: incrementation, resp. decrementation of a number. Both procedures receive a parameter "a" of type integer. This parameter is incremented, resp. decremented with 1.

In the main program (between the keywords "begin" and "end."):  
1. the variable "\( v \)" is initialized with the value 0  
2. the procedure increment (\( v \)) is called with "\( v \)" as parameter. \( v \) is incremented and the result is stored in \( v \). After calling increment(\( v \)) we have \( v=1 \)  
3. The write and writeln prints the result on the screen  
4. The procedure decrement is called analogously the procedure increment  
5. The write and writeln prints the result on the screen  
6. end. – the program ends
Advantages of Procedural Programming (1)

• Widely spread for system and application programming
  - UNIX and WINDOWS operating systems are mostly written in C

**What is an operating system?** It is a software that facilitates the functioning of a computer. It administrates the main memory, the input-output devices and manages the execution of programs.
Advantages of Procedural Programming (2)

- **Widely distributed** by system and user programming
  
  - The programming language C is an essential part of the UNIX operating system: the C-compiler is installed together with the operating system. Many programs in the UNIX world are delivered as source code and must be locally compiled

**What is a compiler?** A compiler (also called translator) is a computer program, that transforms the source code of a program, written in a specific programming language, into a semantically equivalent program, written in the target language. Usually, the source code that was written by a programmer in an established programming language is translated into assembler code, byte code or code written in machine language. This translation of the source program (the source code) into the target program is also called **compilation**.
Advantages of Procedural Programming (3)

- **Transparency** of the language
- Support of the **structured programming** through determined selection of program and control mechanisms
- Possibility to exactly describe the data that is given through a multitude of data types
- Simple design of the language, as well as efficient implementation
Disadvantages of Procedural Programming

- Changes in the data structures prove to be very elaborate and error-prone
- High costs for the maintenance and service of large complex programs
- Careless supply of procedures or false interpretation can lead to the system run time errors
- Searching for errors is difficult
- Limited reusability
- Hard to read due to chained complex procedures