System-Level Programming

1 Introduction

Peter Wägemann

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)

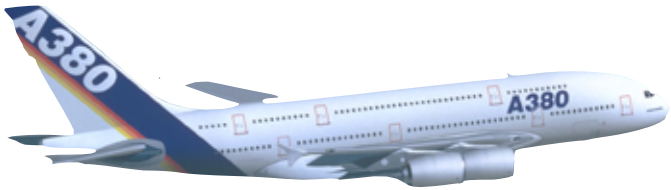
Summer Term 2025

http://sys.cs.fau.de/lehre/ss25

Introduction

- **Deepen** knowledge of concepts and techniques of computer science and software development
  - Starting point: Algorithms, Programming, and Data Representation
  - Main focus: System-Level Programming (SLP) in C
- **Development** of software in C for a µController (µC) and an operating-system platform (Linux)
  - SPiCboard learning development platform with an ATmega-µC
  - Practical experience in hardware and system-level software development
- **Understanding** of language and hardware basics for the development of system-level software
  - Being able to understand and assess the language C and
  - Dealing with concurrency and hardware orientation
  - Dealing with the abstractions of an operating system (files, processes, . . .)

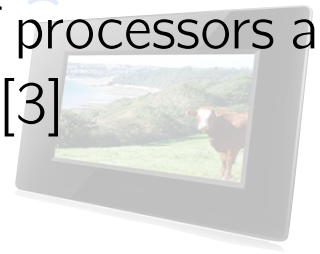
Motivation: Embedded Systems



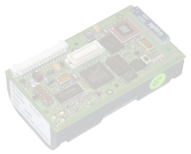
Motivation: Embedded Systems

- **Omnipresent:** **98–99 percent** of processors are being used in **embedded systems** [3]



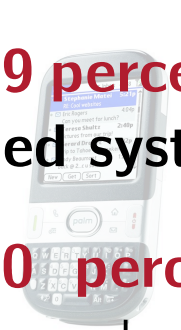
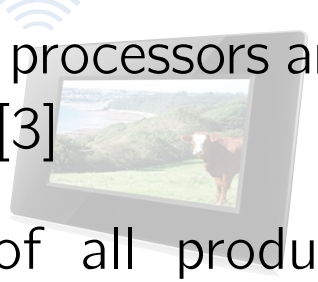






Motivation: Embedded Systems

- **Omnipresent:** **98–99 percent** of processors are being used in **embedded systems** [3]
- **Cost-sensitive:** **70–80 percent** of all produced processors are DSPs and µController, based on **8-bit architecture or smaller** [3, 4]

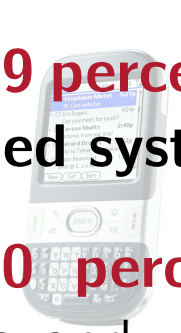




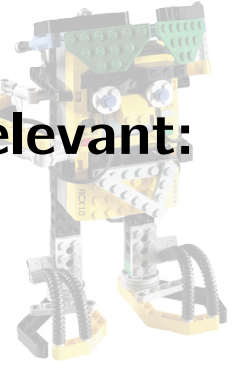





Motivation: Embedded Systems

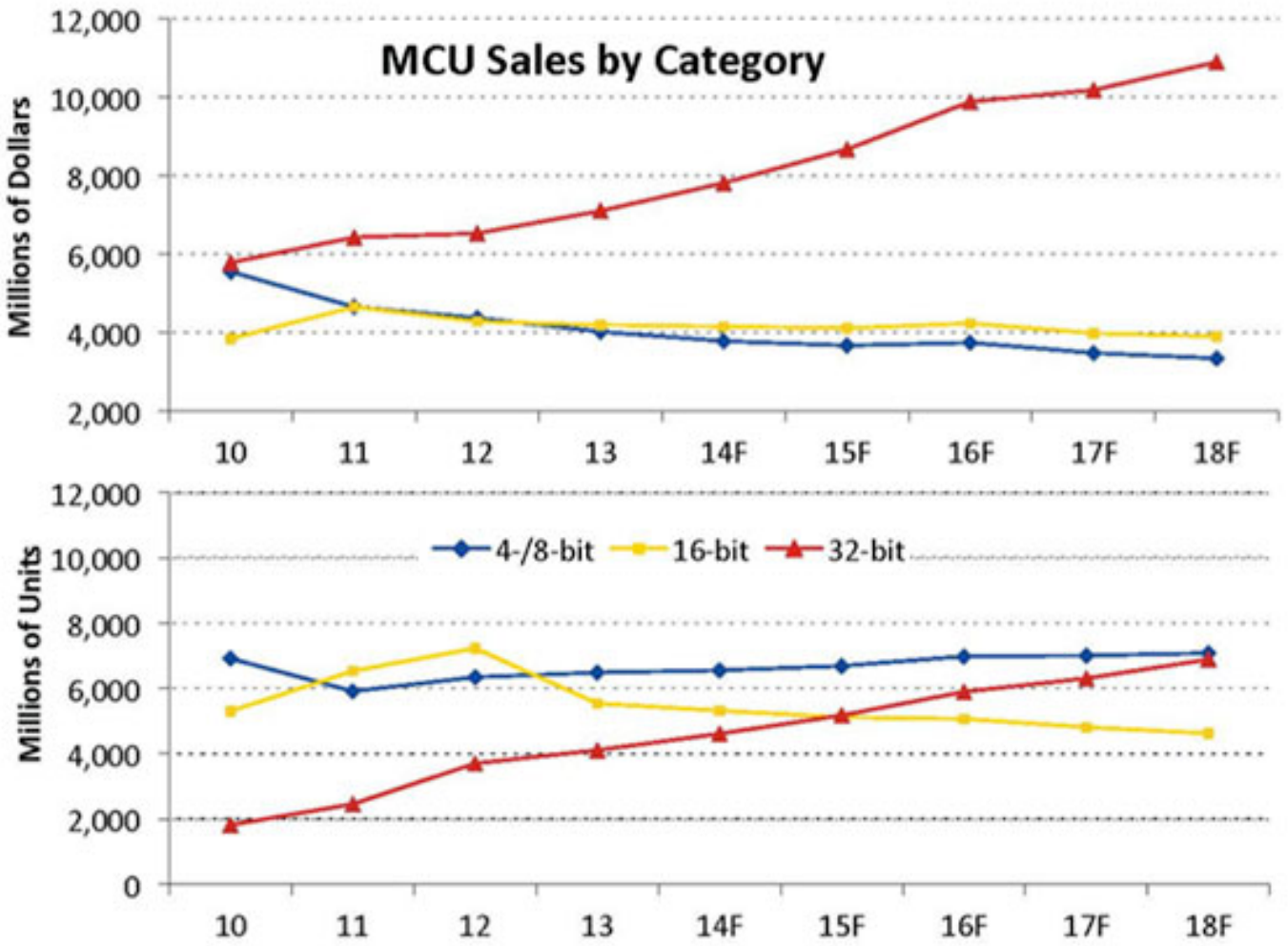
- **Omnipresent:** **98–99 percent** of processors are being used in **embedded systems** [3]
- **Cost-sensitive:** **70–80 percent** of all produced processors are DSPs and µController, based on **8-bit architecture or smaller** [3, 4]
- **Relevant:** **25 percent** of job offers for EE engineers do contain the terms *embedded* or *automotive* (`http://stepstone.com`)




Motivation: Embedded Systems



Source: IC Insights 2014 *McClean Report*

Motivation: The ATmega-µC Family (8-bit)

Type	Flash	SRAM	IO	Timer 8/16	UART	SPI	ADC	PWM	EUR
ATTINY13	1 KiB	64 B	6	1/-	-	-	1*4	-	**2,20**
ATTINY2313	2 KiB	128 B	18	1/1	-	1	-	-	**2,99**
ATMEGA48	4 KiB	512 B	23	2/1	1	1	8*10	6	**2,40**
ATMEGA16	16 KiB	1024 B	32	2/1	1	1	8*10	4	**6,40**
ATMEGA32	32 KiB	2048 B	32	2/1	1	1	8*10	4	**5,40**
ATMEGA64	64 KiB	4096 B	53	2/2	2	1	8*10	8	**–**
ATMEGA128	128 KiB	4096 B	53	2/2	2	1	8*10	8	**19,80**
ATMEGA256	256 KiB	8192 B	86	2/2	4	1	16*10	16	**15,50**

ATmega variants (selection) and market prices (Reichelt Elektronik, April 2023)

Motivation: The ATmega-µC Family (8-bit)

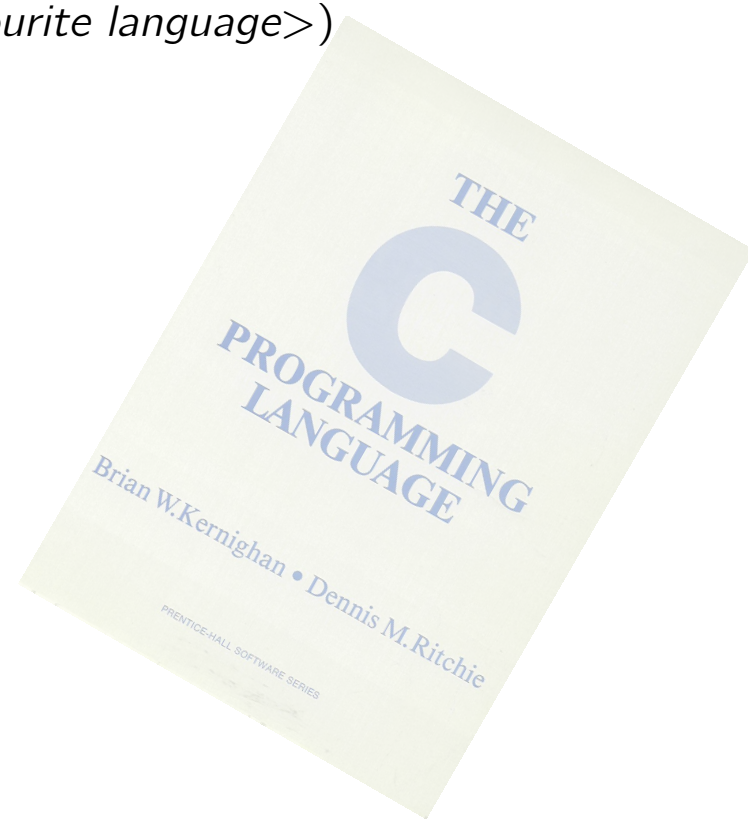
Type	Flash	SRAM	IO	Timer 8/16	UART	SPI	ADC	PWM	EUR
ATTINY13	1 KiB	64 B	6	1/-	-	-	1*4	-	**2,20**
ATTINY2313	2 KiB	128 B	18	1/1	-	1	-	-	**2,99**
ATMEGA48	4 KiB	512 B	23	2/1	1	1	8*10	6	**2,40**
ATMEGA16	16 KiB	1024 B	32	2/1	1	1	8*10	4	**6,40**
ATMEGA32	32 KiB	2048 B	32	2/1	1	1	8*10	4	**5,40**
ATMEGA64	64 KiB	4096 B	53	2/2	2	1	8*10	8	**–**
ATMEGA128	128 KiB	4096 B	53	2/2	2	1	8*10	8	**19,80**
ATMEGA256	256 KiB	8192 B	86	2/2	4	1	16*10	16	**15,50**

ATmega variants (selection) and market prices (Reichelt Elektronik, April 2023)

- Becomes visible: **resource scarcity**
  - **Flash** (memory for program code and constant data) is **scarce**
  - **RAM** (memory for runtime variables) is **extremely scarce**
  - few bytes "wasted" $\rightsquigarrow$ significantly higher cost per piece

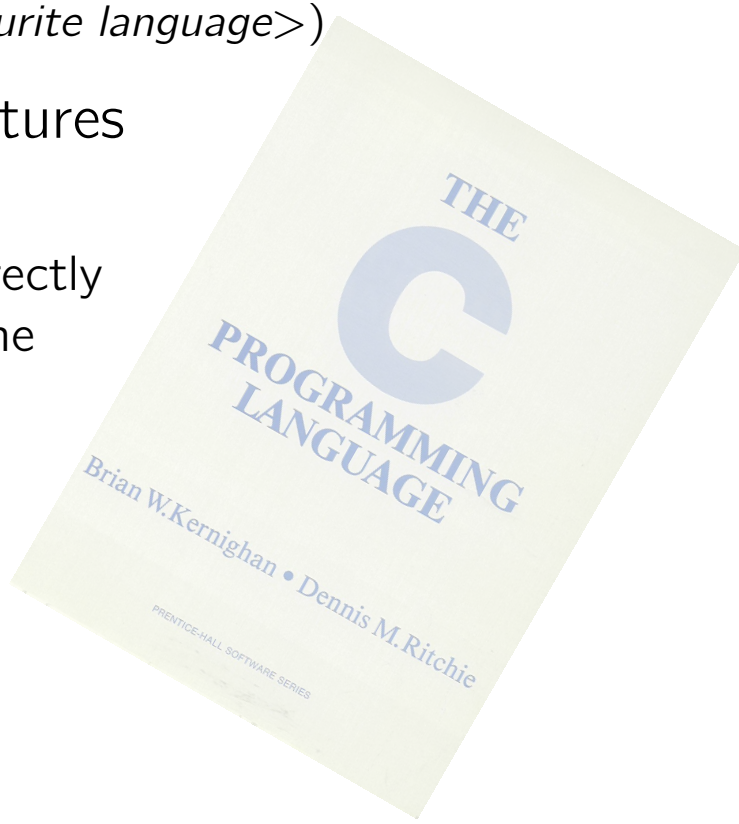
Motivation: C as a Language

- System-level software development mostly uses **C**.
  - **Why C?** (and not Python/Java/Scala/<*favourite language*>)



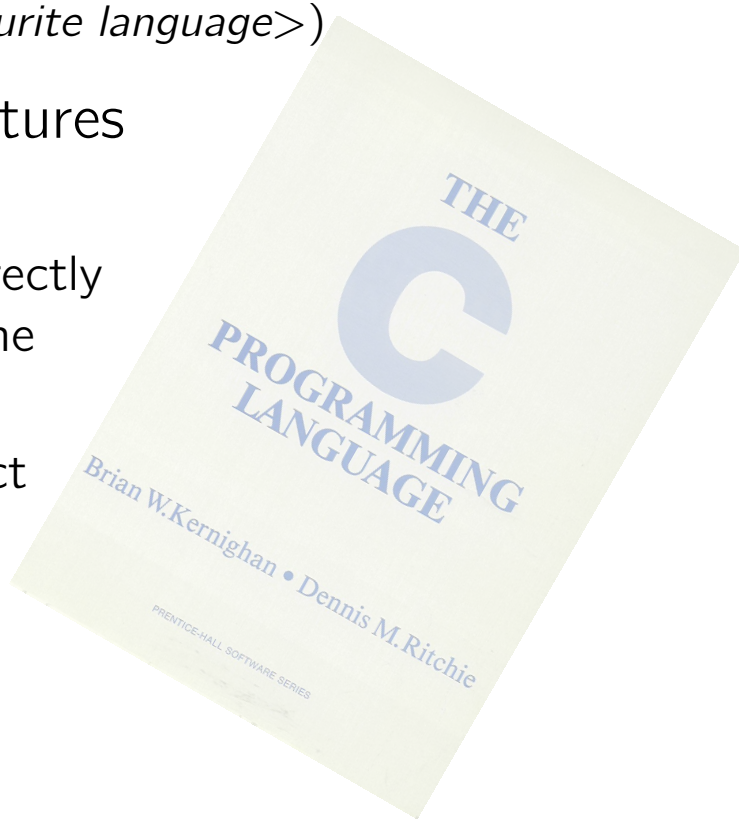
Motivation: C as a Language

- System-level software development mostly uses **C**.
  - **Why C?** (and not Python/Java/Scala/<*favourite language*>)
- C stands for a multitude of important features
  - **Runtime efficiency** (CPU)
    - Translated C code runs on the processor directly
    - No checks for programming errors at runtime



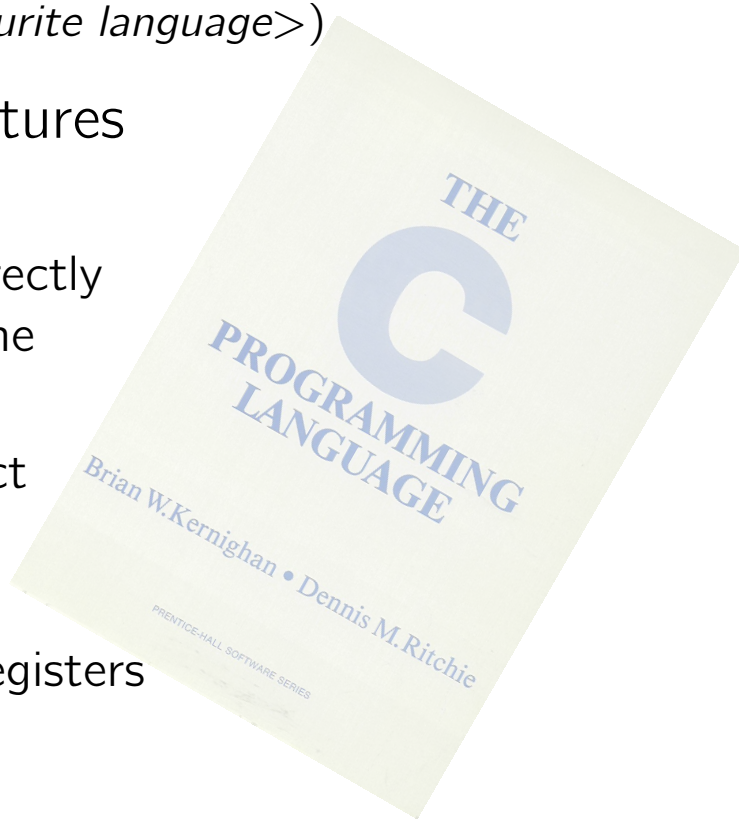
Motivation: C as a Language

- System-level software development mostly uses **C**.
  - **Why C?** (and not Python/Java/Scala/<*favourite language*>)
- C stands for a multitude of important features
  - **Runtime efficiency** (CPU)
    - Translated C code runs on the processor directly
    - No checks for programming errors at runtime
  - **Space efficiency** (memory)
    - Code and data can be stored rather compact
    - No checks for data access at runtime



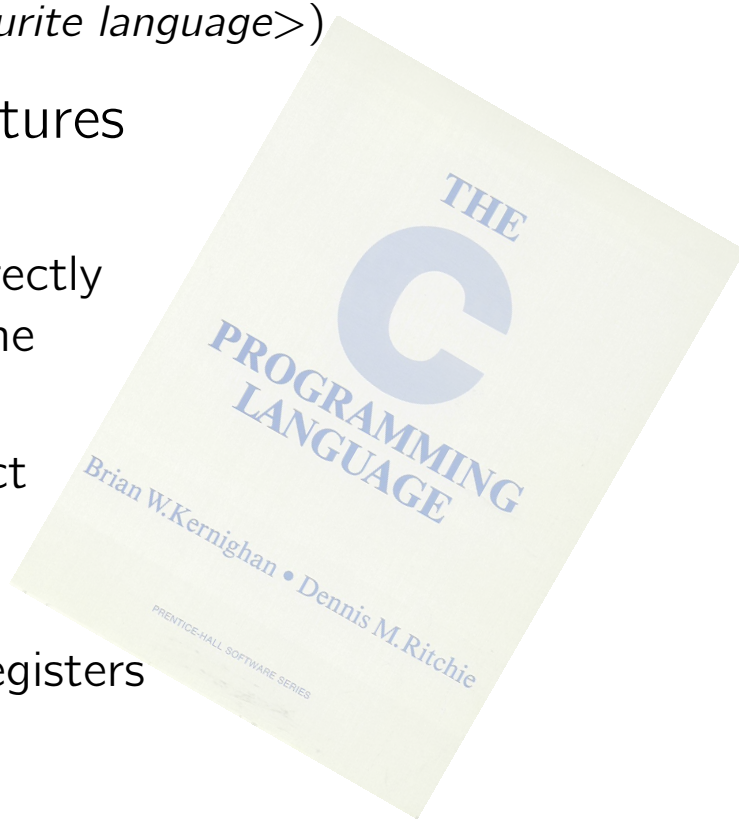
Motivation: C as a Language

- System-level software development mostly uses **C**.
  - **Why C?** (and not Python/Java/Scala/<*favourite language*>)
- C stands for a multitude of important features
  - **Runtime efficiency** (CPU)
    - Translated C code runs on the processor directly
    - No checks for programming errors at runtime
  - **Space efficiency** (memory)
    - Code and data can be stored rather compact
    - No checks for data access at runtime
  - **Directness** (machine orientation)
    - C allows for direct access to memory and registers



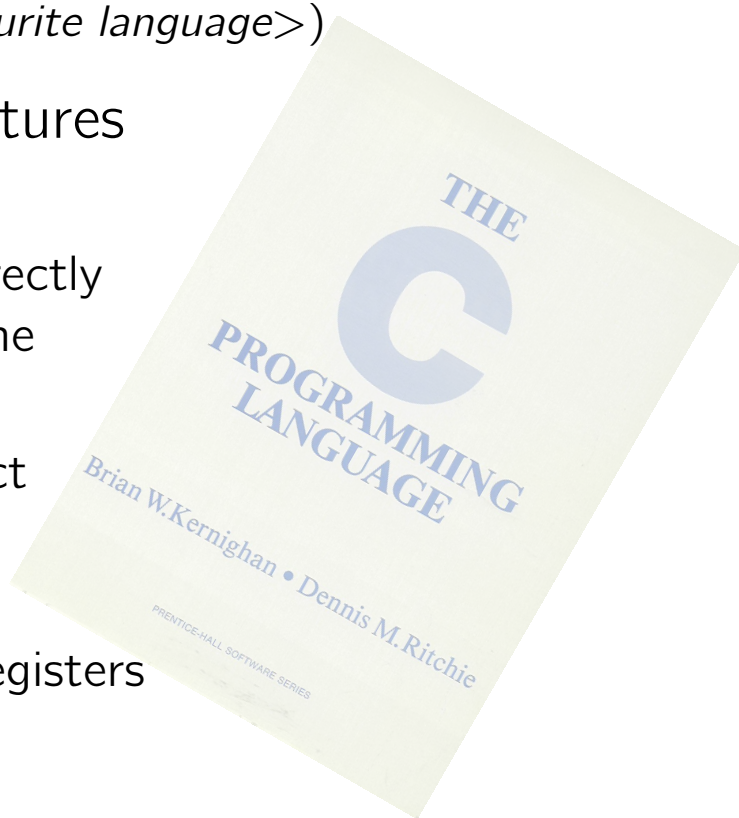
Motivation: C as a Language

- System-level software development mostly uses **C**.
  - **Why C?** (and not Python/Java/Scala/<*favourite language*>)
- C stands for a multitude of important features
  - **Runtime efficiency** (CPU)
    - Translated C code runs on the processor directly
    - No checks for programming errors at runtime
  - **Space efficiency** (memory)
    - Code and data can be stored rather compact
    - No checks for data access at runtime
  - **Directness** (machine orientation)
    - C allows for direct access to memory and registers
  - **Portability**
    - There is a C compiler for **every platform**
    - C was "invented" (1973), to implement the OS UNIX portable [1, 0]



Motivation: C as a Language

- System-level software development mostly uses **C**.
  - **Why C?** (and not Python/Java/Scala/<*favourite language*>)
- C stands for a multitude of important features
  - **Runtime efficiency** (CPU)
    - Translated C code runs on the processor directly
    - No checks for programming errors at runtime
  - **Space efficiency** (memory)
    - Code and data can be stored rather compact
    - No checks for data access at runtime
  - **Directness** (machine orientation)
    - C allows for direct access to memory and registers
  - **Portability**
    - There is a C compiler for **every platform**
    - C was "invented" (1973), to implement the OS UNIX portable [1, 0]



↝ **C** is the **lingua franca** of system-level programming!

Motivation: SLP – Syllabus and Concept

- **Teaching objective:** system-level programming in C
  - This is a really broad field: hardware programming, operating systems, middleware, data bases, distributed systems, compiler construction, ...
  - Additionally, we have the goal of learning the language C itself
- **Approach**
  - Concentration on two domains
    - µC programming
    - Software development for Linux system interface
  - Experience contrast µC environment $\leftrightarrow$ operating system (OS)
  - Concepts and techniques taught and experienced with the help of various examples
  - **High relevance** for the target audience (electrical & mechanical engineering, ...)

Motivation: SLP

At the end of the lecture, everyone should be able to assess,

- what a µC can (not) do,
- how labor-intensive & beneficial µC programming is,
- what an OS does (not) provide,
- how labor-intensive & beneficial it is to use a µC.

Everyone should be able to work with a computer scientist, if necessary...

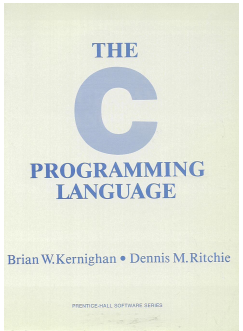
Lecture Notes

- This handout of the lecture notes will be provided online.
  - Chapters are available as individual files
  - The handout contains (some) additional information
- **However, the handout cannot be used as a substitute for making your own notes!**

Literature Recommendations

[2] standard book (more suitable as a reference):

Brian W. Kernighan und Dennis MacAlistair Ritchie. *The C Programming Language (2nd Edition)*. Englewood Cliffs, NJ, USA: Prentice Hall PTR, 1988. ISBN: 978-8120305960

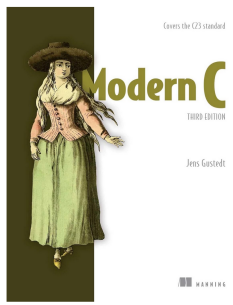


[0] open-access book (guide for audience with basic programming knowledge):

Brian "Beej Jorgensen" Hall. *Beej's Guide to C Programming*. 2025. URL: https://beej.us/guide/bgc/

[0] open-access book (covers modern C standards):

Jens Gustedt. *Modern C*. Manning, 2024. URL: https://inria.hal.science/hal-02383654



References

[1] Brian W. Kernighan und Dennis MacAlistair Ritchie. *The C Programming Language*. Englewood Cliffs, NJ, USA: Prentice Hall PTR, 1978.

[2] Brian W. Kernighan und Dennis MacAlistair Ritchie. *The C Programming Language (2nd Edition)*. Englewood Cliffs, NJ, USA: Prentice Hall PTR, 1988. ISBN: 978-8120305960.

[3] David Tennenhouse. "Proactive Computing". In: *Communications of the ACM* (Mai 2000), S. 43–45.

[4] Jim Turley. "The Two Percent Solution". In: *embedded.com* (Dez. 2002). `http://www.embedded.com/story/OEG20021217S0039`, visited 2011-04-08.