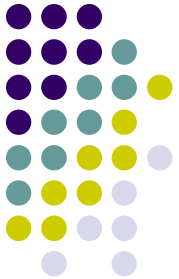
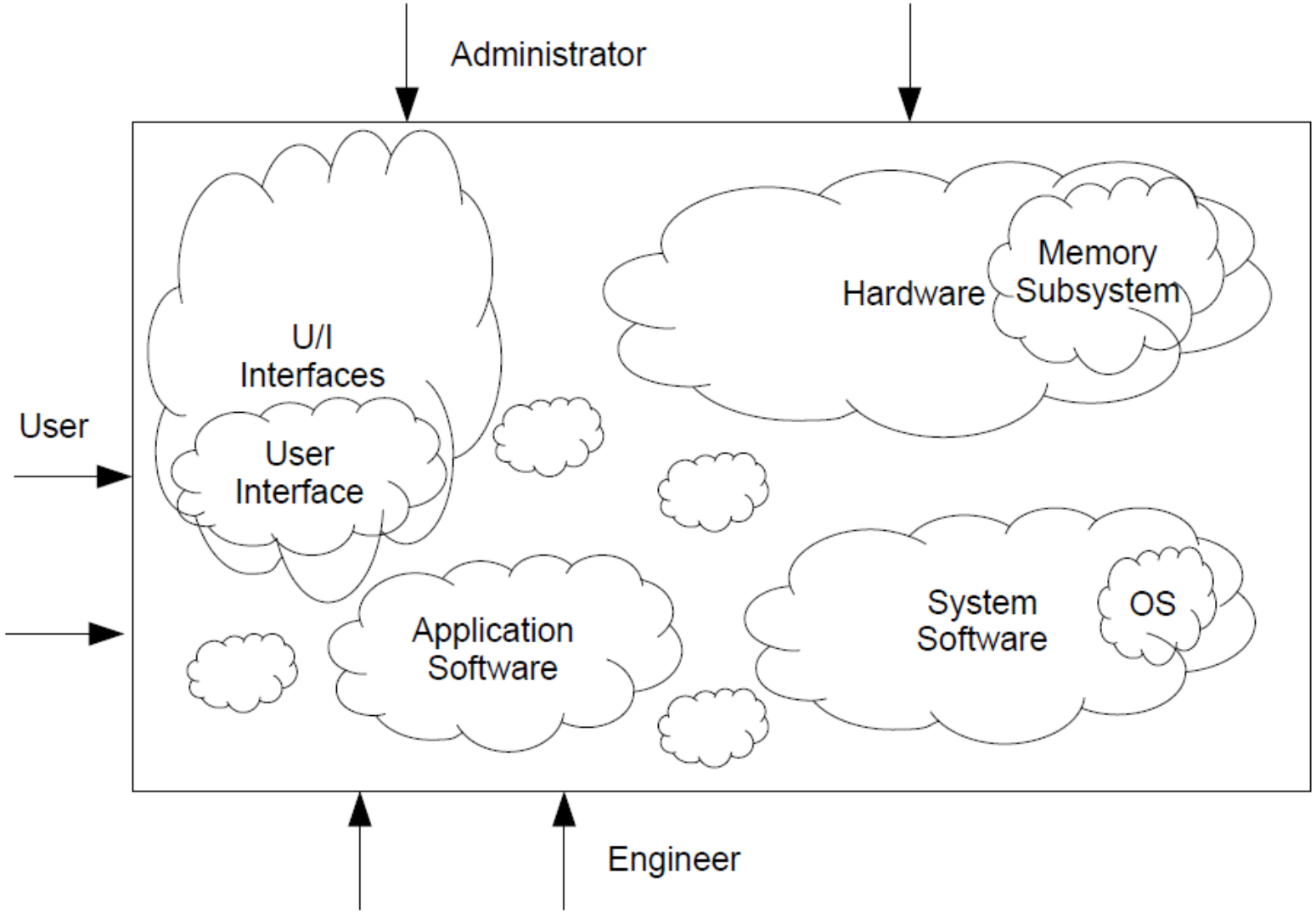
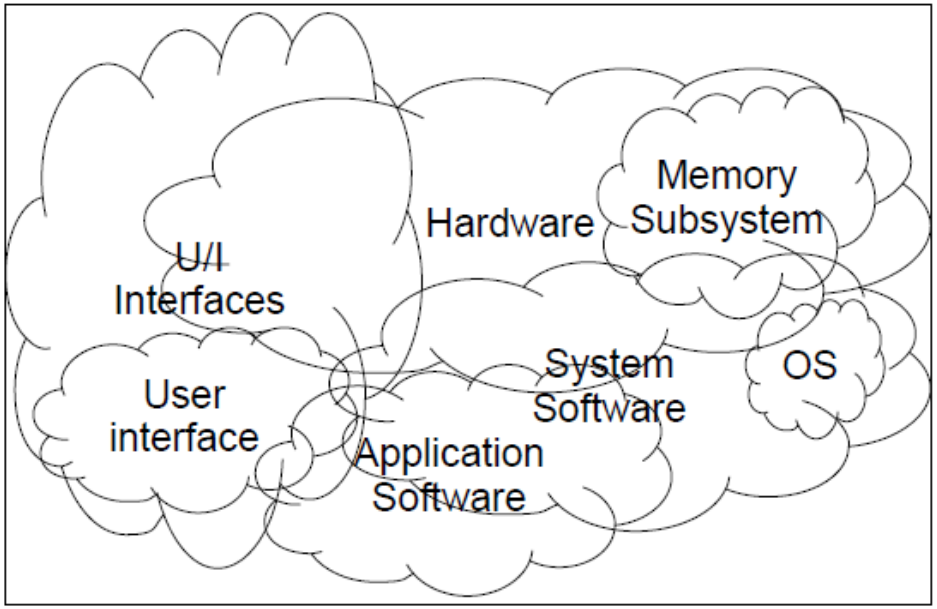


Introduction

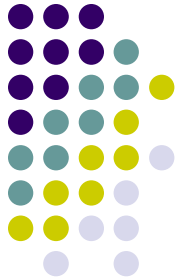
Computer system



Computer system

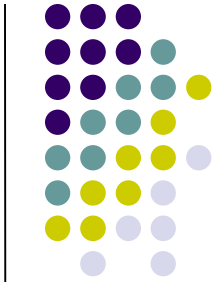


Computer system



- Hardware - software
- Usually approached separately, but in computer engineering they are tightly connected, and when working on one you have to thing about the other.
- Since mostly development of hardware starts before software development, in the common case development of software depends more on hardware then the other way.

Computer system



- As computer size increases, price increases.
- As computer size increases, power consumption increases.

- Computer system has to work **as good as possible**, its price to be as low as possible, and to consume power as low as possible.

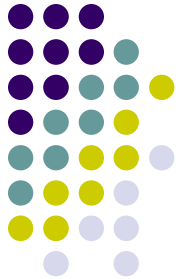
Computer system

- As computer size increases, price increases.
- As computer size increases, power consumption increases.

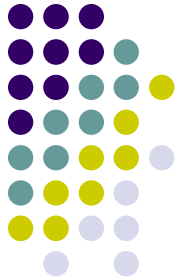
- Computer system has to work **good enough**, its price to be as low as possible, and to consume power as low as possible.

- **What** to do good enough?
  - To run expected programs fast enough.

Computer system



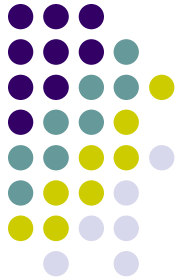
- Real time system
- **Embedded system**
- System with limited resources -> Processor with limited resources
- Special purpose systems -> Special purpose processor



Real time systems

- Systems that work in real time.
- Key thing: tasks have a dead-line.
- When programming such systems it is almost impossible to make clear separating between “what” and “how”.

Example: Computer 1 starts text processing program in 3 minutes, computer 2 starts is in 15 second. Which computer works in real time?



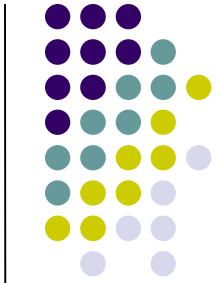
Real time systems

- Systems that work in real time.
- Key thing: tasks have a dead-line.
- When programming such systems it is almost impossible to make clear separating between “what” and “how”.

Example: Computer 1 starts text processing program in 3 minutes, computer 2 starts is in 15 second. Which computer works in real time?

Trick question – there is no predefined dead-line.

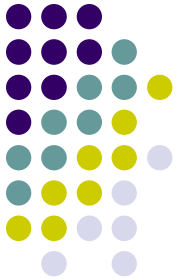
Embedded system



- No human-machine interface (or it is heavily reduced)
  - There is no: keyboard, screen etc.

- 99% of processors are in embedded systems
- System lifecycle: development, production, maintenance.
- For embedded systems development cost is usually under 5% of the whole cycle cost.

System with limited resources

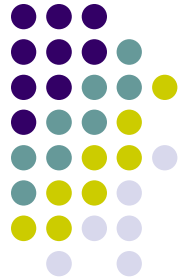


- Resources:
  - Memory, registers
  - **Speed (MIPS)**
  - Energy
  - ...

Example: The main computer is the most reliable spacecraft – Soyuz, is TsVM-101, with 6MHz and around 3MB of memory.

- Being limited is a relative thing
- but resources in these kind of systems are usually much smaller than in common PCs

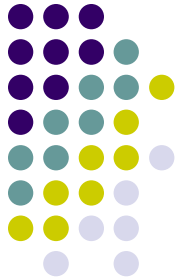
Special purpose systems



- Systems that target a specific use-case

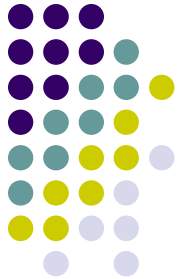
- Special purpose processors:
  - Does every processor have an instruction for multiplication? How about for adding numbers... ?

Processor classes



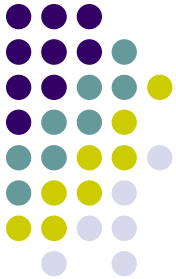
- General purpose processors
- Special purpose processors
  - DSPs
  - Microcontrollers
  - GPUs...

General purpose processors

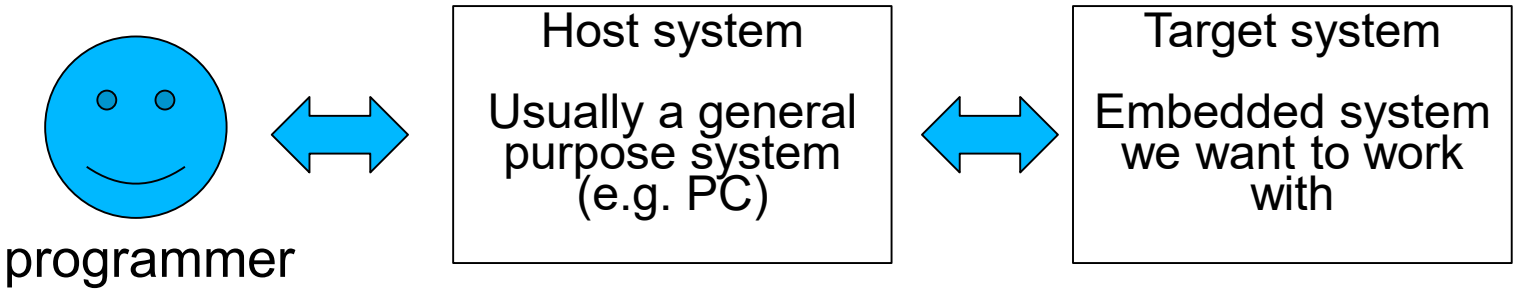


- The best performance in running all possible programs.
- Equivalent performance in running all possible programs.
- Some classes of programs run better, some worse, but in general the difference in the performance is less then one order of magnitude (10x).

With what do we program embedded systems

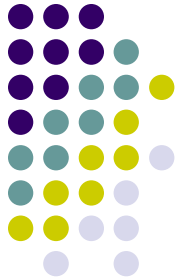


- Work with ES is usually not direct.



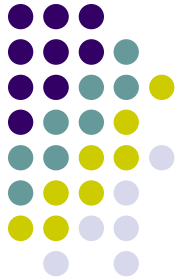
- Part of system software runs on the host system.
- Host is usually a system with faster and with more resources, and therefore it is easier to work on it.
- Many elements of system software could not even run on the target system (e.g. compiler, IDE, etc.).

With what do we program embedded systems



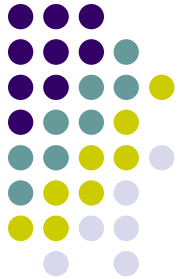
- Tools (part of the system software):
  - Assembler
  - Compiler (for some high level language)
  - Debugger
  - Simulator (enables better execution control and insight)
  - Integrated development environment
- Operating system and libraries

When do we program embedded systems



- Before the hardware is finished
  - To accelerate development of the system
  - Pert of the code is already available
- When the hardware architecture is locked, but it is still not produced
  - Development boards with prototype
  - Simulators
- When the hardware is finished, but system software is not (tools)
  - Development of system software usually starts before the hardware is finished
- When the hardware is ready and mature
  - For some systems this already end of their life cycle
  - For some other systems – it is a new beginning

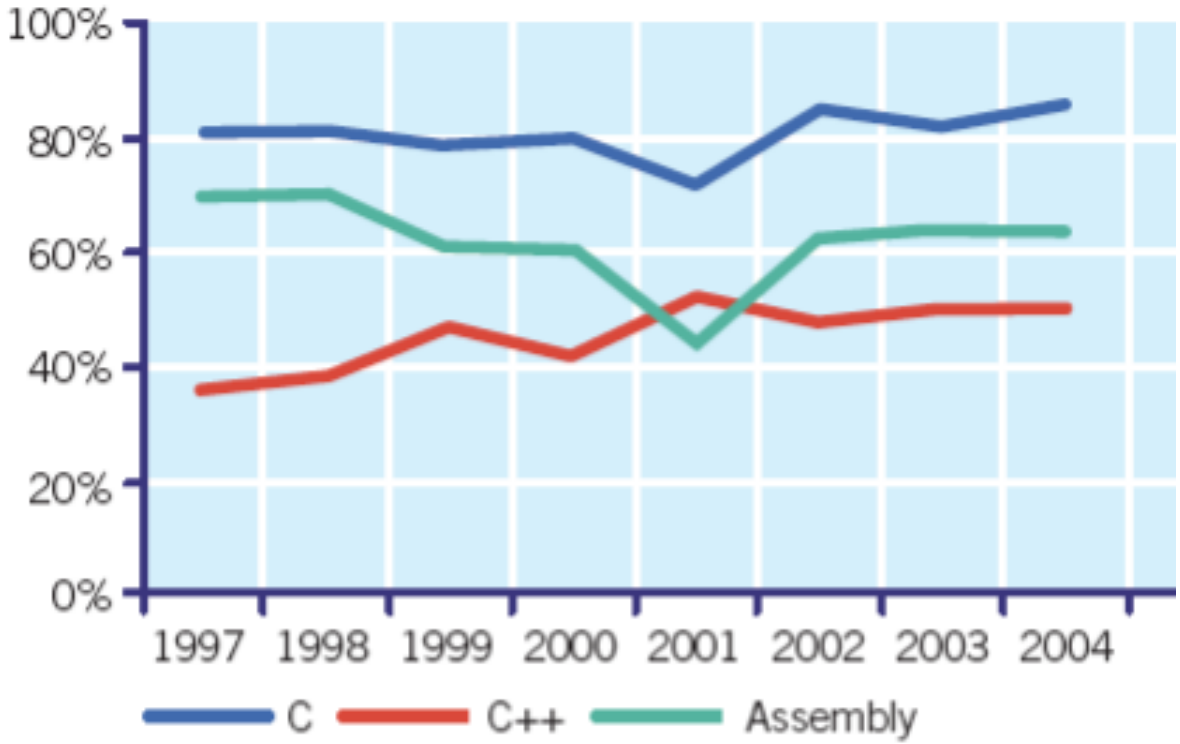
How do we program embedded systems



- Depends on may factors:
  - Development phase – readiness of the hardware and tools
  - Invested development effort – in the system software, mostly tools
  - Size and complexity of the particular software – small programs can be programed even in binary code
  - Requirements of the particular software – more performance can be improved on lower abstraction level
- Assembler
  - Closer to hardware
  - Translator easier to make
  - More complicated for programming
- Higher level language
  - Forder away from hardware
  - Translator harder to make
  - Easier for programming

C programming language

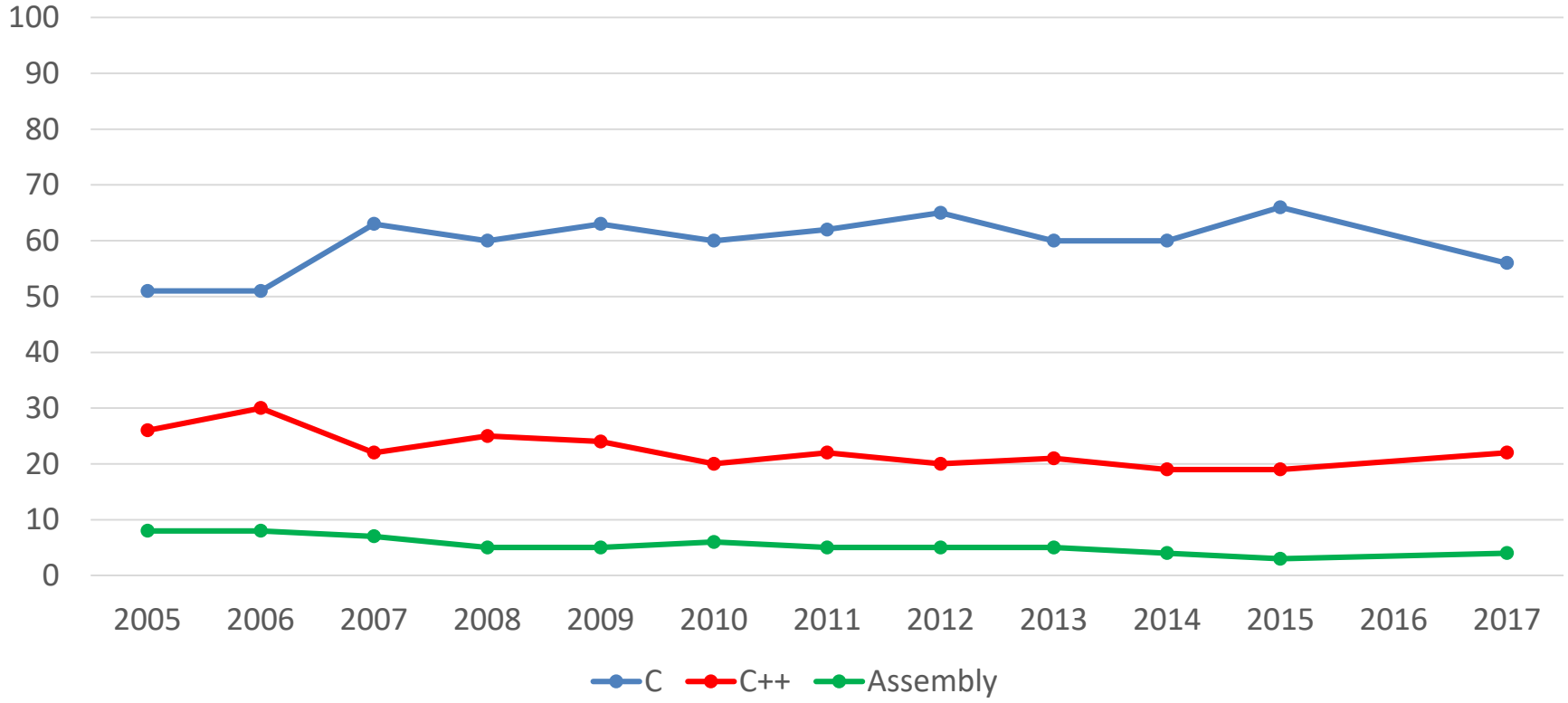
- High level programming language that is mostly used for programming embedded systems.



Percentage of projects in which a given language is used at least somewhere.

C programming language

- High level programming language that is mostly used for programming embedded systems.



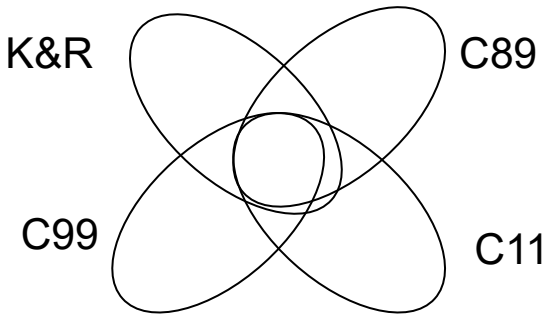
Percentage of projects in which a given language is predominantly used.

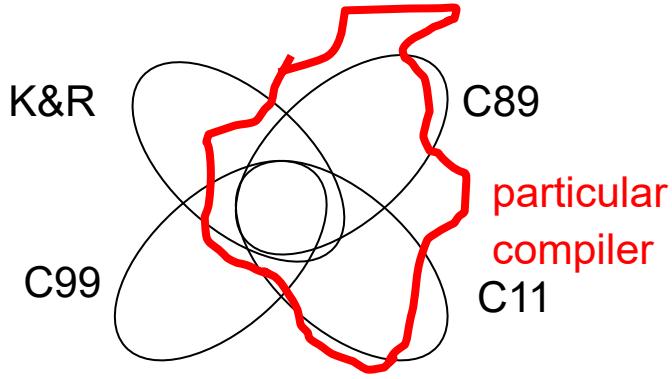
C programming language

- Why C?
  - Not too high
    - Enables better performance optimization and system control
    - Relatively good mapping of its operations on hardware operations
  - Historical reasons
    - A lot of existing code
    - Large number of C programmers and experts
    - Compiler construction relatively easy (e.g. a lot of existing front-ends, etc.)

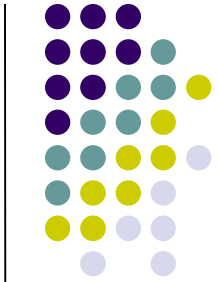
C programming language

- Why **NOT** C?
  - Not too high
    - Higher level of abstraction would make programming easier
    - Relatively **bad** mapping of its operations on hardware operations
  - Historical reasons
    - Something better could have been invented by now?
    - Large number of C “programmers” and “experts”
    - Which standard this particular system supports?





The goal of this course



Train students to be capable of:

- Writing new C code for embedded systems
  - Learn parts of the language which are not in focus when programming for general purpose systems and ad hoc programming, but are important when programming embedded systems
  - Learn good practice for writing nice, clear and useful code
  - Detect gray areas of the language so they would not use it unnecessarily.

- Understanding existing C code for embedded systems
  - Detect gray areas of the language so they would be able to understand them