ELEC-H410
REAL-TIME SYSTEMS

Chapter 6

Programming languages

Programming languages

CONTENTS

- **introduction**
- selection criteria
- MMI
- real-time
- conclusions

Choosing a language

a jungle ...

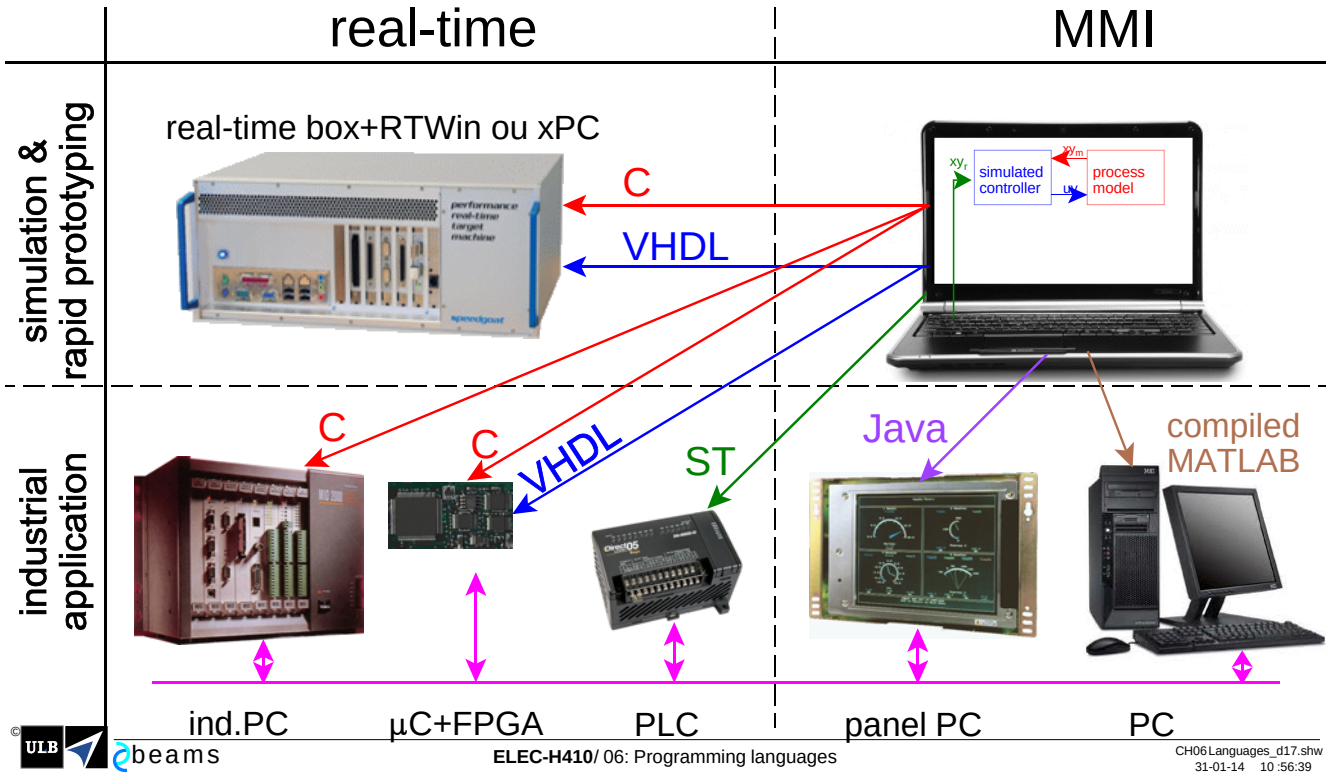
A#.NET, A# (Axiom), A-0 System, A+, A++, ABAP, ABC, ABC ALGOL, Abel, ABLE, ABSET, ABSYS, Abundance, ACC, Accent, Ace DASL, ACT-III, Actionl, ActionScript, **Ada**, Adenine, Agda, Agora, AIMMS, Alef, ALF, ALGOL 58, ALGOL 60, ALGOL 68, Alice, Alma-0, AmbientTalk, Amiga E, AMOS, AMPL, APL, AppleScript, Arc, ARexx, Argus, AspectJ, **Assembly language,** ATS, Ateji PX, AutoHotkey, Autocoder, AutoIt, AutoLISP / Visual LISP, Averest, AWK, Axum, B, Babbage, Bash, BASIC, bc, BCPL, BeanShell, Batch (Windows/Dos), Bertrand, BETA, Bigwig, Bistro, BitC, BLISS, Blue, Bon, Boo, Boomerang, Bourne shell (including bash and ksh), BREW, BPEL, BUGSYS, BuildProfessional, **C**, C--, **C++** - ISO/IEC 14882, **C#** - ISO/IEC 23270, C/AL, Caché ObjectScript, C Shell, Caml, Candle, Cayenne, CDuce, Cecil, Cel, Cesil, CFML, Cg, Chapel, CHAIN, Charity, Charm, Chef, CHILL, CHIP-8, chomski, Oxygene (formerly Chrome), ChucK, CICS, Clik, CL (IBM), Claire, Clarion, Clean, Clipper, CLIST, Clojure, CLU, CMS-2, COBOL - ISO/IEC 1989, CobolScript, Cobra, CODE, CoffeeScript, Cola, ColdC, ColdFusion, Cool, COMAL, Combined Programming Language (CPL), Common Intermediate Language (CIL), Common Lisp (also known as CL), COMPASS, Component Pascal, COMIT, Constraint Handling Rules (CHR), Converge, Coral 66, Corn, CorVision, Coq, COWSEL, CPL, csh, CSP, Csound, Curl, Curry, Cyclone, Cython, D, DASL (Datapoint's Advanced Systems Language), DASL (Distributed Application Specification Language), Dart, DataFlex, Datalog, DATATRIEVE, dBase, dc, DCL, Deesel (formerly G), **Delphi**, DinkC, DIBOL, DL/I, Draco, Dylan, DYNAMO, E, Ease, EASY, Easy PL/I, EASYTRIEVE PLUS, ECMAScript, Edinburgh IMP, EGL, Eiffel, ELAN, Emacs Lisp, Emerald, Epigram, Erlang, Escapade, Escher, ESPOL, Esterel, Etoys, Euclid, Euler, Euphoria, EusLisp Robot Programming Language, CMS EXEC, EXEC 2, F, F#, Factor, Falcon, Fancy, Fantom, Felix, Ferite, FFP, Fjölnir, FL, Flavors, Flex, FLOW-MATIC, FOCAL, FOCUS, FOIL, FORMAC, @Formula, Forth, Fortran -ISO/IEC 1539, Fortress, FoxBase, FoxPro, FP, FPr, Franz Lisp, Frink, F-Script, Fuxi, G, Game Maker Language, GameMonkey Script, GAMS, GAP, G-code, Genie, GDL, Gibiane, GJ, GLSL, GNU E, GM, Go, Go!, GOAL, Gödel, Godiva, GOM (Good Old Mad), Goo, GOTRAN, GPSS, GraphTalk, GRASS, Green, Groovy, HAL/S, Hamilton C shell, Harbour, IBM HAScript, Haskell, HaXe, High Level Assembly, HLSL, Hop, Hope, Hugo, Hume, HyperTalk, IBM Basic assembly language, IBM Informix-4GL, IBM RPG, ICI, Icon, Id, IDL, IMP, Inform, Io, Ioke, IPL, IPTSCRAE, ISPF, ISWIM, HTML, J, J#, J++, JADE, Jako, JAL, Janus, JASS, **Java,** JavaScript, JCL, JEAN, Join Java, JOSS, Joule, JOVIAL, Joy, JScript, Jython, JavaFX Script, K, Kaleidoscope, Karel, Karel++, Kaya, KEE, KIF, KRC, KRL, KRL (KUKA Robot Language), KRYPTON, ksh, L, L# .NET, **LabVIEW,** Ladder, Lagoona, LANSA, Lasso, LaTeX, Lava, LC-3, Leadwerks Script, Leda, Legoscript, LilyPond, Limbo, Limnor, LINC, Lingo, Linoleum, LIS, LISA, Lisaac, Lisp - ISO/IEC 13816, Lite-C Lite-c, Lithe, Little b, Logo, Logtalk, LPC, LSE, LSL, Lua, Lucid, Lustre, LYaPAS, Lynx, M, M2001, M4, Machine code, MAD (Michigan Algorithm Decoder), MAD/I, Magik, Magma, make, Maple, MAPPER (Unisys/Sperry) now part of BIS, MARK-IV (Sterling/Informatics) now VISION:BUILDER of CA, Mary, MASM Microsoft Assembly x86, Mathematica, **MATLAB,** Maxima (see also Macsyma), MaxScript internal language 3D Studio Max, Maya (MEL), MDL, Mercury, Mesa, Metacard, Metafont, MetaL, Microcode, MicroScript, MIIS, MillScript, MIMIC, Mirah, Miranda, MIVA Script, ML, Moby, Model 204, Modelica, Modula, Modula-2, Modula-3, Mohol, MOO, Mortran, Mouse, MPD, MSIL -deprecated name for CIL, MSL, MUMPS, Napier88, NASM, NATURAL, NEAT chipset, Neko, Nemerle, NESL, Net.Data, NetLogo, NewLISP, NEWP, Newspeak, NewtonScript, NGL, Nial, Nice, Nickle, NPL, Not eXactly C (NXC), Not Quite C (NQC), Nu, NSIS, o:XML, Oak, Oberon, Object Lisp, ObjectLOGO, Object REXX, Object Pascal, Objective-C, Objective Caml, Objective-J, Obliq, Obol, occam, occam-p, Octave, OmniMark, Onyx, Opa, Opal, OpenEdge ABL, OPL, OPS5, OptimJ, Orc, ORCA/Modula-2, Orwell, Oxygene, Oz, P#, PARI/GP, Pascal -ISO 7185, Pawn, PCASTL, PCF, PEARL, PeopleCode, Perl, PDL, PHP, Phrogram, Pico, Pict, Pike, PIKT, PILOT, Pizza, PL-11, PL/0, PL/B, PL/C, PL/I - ISO 6160, PL/M, PL/P, PL/SQL, PL360, PLANC, Plankalkül, PLEX, PLEXIL, Plus, POP-11, PostScript, PortableE, Powerhouse, PowerBuilder - 4GL GUI appl. generator from Sybase, PPL, Processing, Prograph, PROIV, Prolog, Visual Prolog, Promela, PROTEL, ProvideX, Pro*C, Pure, **Python**, Q (equational programming language), Q (programming language from Kx Systems), Qi, QtScript, QuakeC, QPL, R, R++, Racket, RAPID, Rapira, Ratfiv, Ratfor, rc, REBOL, Redcode, REFAL, Reia, Revolution, rex, REXX, Rlab, ROOP, RPG, RPL, RSL, RTL/2, Ruby, Rust, S, S2, S3, S-Lang, S-PLUS, SA-C, SabreTalk, SAIL, SALSA, SAM76, SAS, SASL, Sather, Sawzall, SBL, Scala, Scheme, Scilab, Scratch, Script.NET, Sed, Self, SenseTalk, SETL, Shift Script, SIMPLE, SIMPOL, SIMSCRIPT, Simula, **Simulink**, SISAL, SLIP, SMALL, Smalltalk, Small Basic, SML, SNOBOL(SPITBOL), Snowball, SOAP, SOL, Span, SPARK, SPIN, SP/k, SPS, Squeak, Squirrel, SR, S/SL, Strand, STATA, Stateflow, Subtext, Suneido, SuperCollider, SuperTalk, SYMPL, SyncCharts, SystemVerilog, T, TACL, TACPOL, TADS, TAL, **Tcl**, Tea, TECO, TELCOMP, TeX, TEX, TIE, Timber, Tom, TOM, Topspeed, TPU, Trac, T-SQL, TTCN, Turing, TUTOR, TXL, Ubercode, Unicon, Uniface, UNITY, Unix shell, UnrealScript, Vala, VBA, VBScript, Verilog, VHDL, **Visual Basic,** Visual Basic .NET, Visual C++, Visual C++ .Net, Visual C#, Visual DataFlex, Visual DialogScript, Visual FoxPro, Visual J++, Visual J#, Visual Objects, VSXu, Vvvv, WATFIV, WATFOR, WebQL, Winbatch, X++, X10, XBL, XC (exploits XMOS architecture), xHarbour, XL, XOTcl, XPL, XPL0, XQuery, XSB, XSLT - See XPath, Yorick, YQL, Yoix, Z notation, Zeno, ZOPL, ZPL, ZZT-oop,

source Wikipedia

The embedded market only resorts to a small subset of programming languages.

Choosing a language

if you follow the workflow



If we follow the workflow of chapter 2, with MATLAB/Simulink as an example, this slide illustrates the whole process and which languages are involved.

First of all, the workflow has been simplified here in only two phases:
- simulation/rapid prototyping
- application, i.e. the final product running in the industry

Secondly, the problems related to real time and to the man-machine interface (MMI) occur at different time-constants and have not the same requirements.
The fictive process represented at the bottom of the slide includes all types of hardware that we have already presented interconnected by a network.

For the rapid prototyping phase, the simulator code will be compiled for the "real-time box" that emulates the controller and/or the process. We shall obtain
- **C** code that can run on a special proprietary RTOS from MATLAB (Real-Time Windows or xPC)
- **VHDL** code if hard and fast real-time tasks require an FPGA

The MMI is on the development PC.

When we want to port the code to the final hardware, MATLAB sells an "Embedded Coder" which can produce C/C++ code,
- either in a generic form that can serve as a base for the source code of the target
- or specifically for a microprocessor belonging to a rather long list

If the project includes an FPGA, the VHDL code is also available for several targets.
Structured text (ST) code can also be downloaded to PLCs (see PLC in real-time hardware chapter).

For the MMI, panel PCs or PCs can run a Graphical User Interfaces written in MATLAB and compiled for a special run-time (so that MATLAB must not be installed on the MMI machine). The MATLAB code can also be translated in Java.

In the rest of this chapter, we shall present several programming languages suitable for real-time and MMI.

Programming languages

CONTENTS

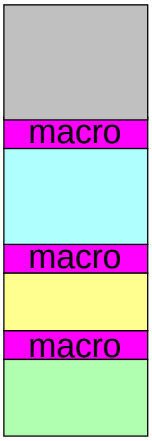
- introduction
- **selection criteria**
- MMI
- real-time
- conclusions

Selection criteria

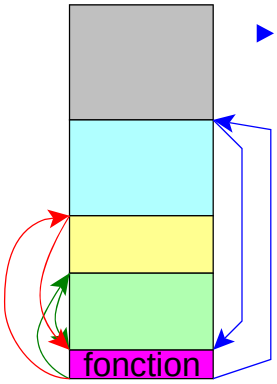
- **speed of execution**
  - measured by a benchmark
    - task or group of tasks, sometimes "normalized"
    - /!\ a benchmark not specific to the type of application is useless
- **portability of the source code**
  - 1/effort necessary to pass a code running on µP1(OS1) to µP2(OS2)
  - /!\ availability of the language on another µP2

Selection criteria

- **compactness of the machine code**
  - measured for a given ask on a given µP
  - sometimes contradictory with speed (ex macro/function)



- macro
  - ⊕ speed ↗
  - ⊖ code ↗



- function
  - ⊕ compact code
  - ⊖ slower execution
    - passing parameters
    - return to calling point

Compactness of the code used to be one of the prominent criteria in the past. The spectacular decrease of the price of all kind of memories has changed the point of view. Nevertheless, for large series of production, resorting to a smaller and cheaper hardware remains an argument in favour of a compact code.

Selection criteria

- **facility of debugging**
  - user-friendlyness
  - quality of the debugging tools
  - clarity of error messages
  - quality of the error detection at run-time
- **facility of the maintenance**
  - easy to update the code ?
    - during 10 or 20 years
    - by several programmers
  - dependent on legibility
  - 1/effort to document

Selection criteria

- **existing libraries**
  - object code or source code (more expensive)
  - already coded, already debugged
  - saving of time
  - often cheaper to buy code than to write it
  - payment
    - one-time license
    - initial fee + royalties per equipment sold

Programming languages

CONTENTS

- introduction
- selection criteria
- **MMI**
- real-time
- conclusions

MMI

MMI screen is very important

- end-users are very demanding
  - they use beautiful screens every day
  - they must be productive and make no error
  - ergonomics: paramount importance
    - response time to
      - key or click <0.1s
      - display value, open/close window <1s
      - display complex graph <10s
    - avoid wrong operation (ex toggle 2 times)
    - save time
- important factor of acceptation or rejection
- commercial and psychological argument
- **Graphical User Interface**

MMI

How to implement

- 1st choice: off-the-shelf products
  - data acquisition / control / rapid prototyping
    - sold by instrument makers
    - MATLAB data acquisition toolbox
    - LabVIEW
  - data/signal processing / graphs
    - idem
    - spreadsheets (Excel)
  - can be compiled and executed on a run-time on any PC

MMI

How to implement

- 2d choice: OOP languages
  - allow reusing lots of libraries
  - languages (alphabetic order)
    - C++
    - C#
    - Delphi
    - Java
    - Python
    - Tcl/Tk
    - Visual BASIC
  - the best is the one you know the best

Programming languages

CONTENTS

- introduction
- selection criteria
- MMI
- **real-time**
  - **description and comparison**
  - ranking
- conclusions

Real-time programming

Assembly language

- speed: ++
  - potentially the quickest of all
  - total management by the programmer
    - choice of each instruction
    - optimal usage of all resources (registers, ...)
  - ASM is "as good as the programmer"

- compactness: ++
  - programmer can optimize the size
  - sometimes tradeoff speed vs compactness
    - ex macros instead of functions (cf.supra)

Real-time programming

Assembly language

- debugging : - -
  - everything is permitted, included fatal errors
    - bad manipulation of the stack pointer
    - change the content of a crucial register
  - "side-effect" of a modification
  - absolutely requires powerful debugging tools
  - tiresome numerical calculations
    - multiple precision
    - floating-point
- maintenance : - -
  - bad legibility
  - looooooooooooong listings
  - 1 comment/line

Real-time programming

Assembly language

- portability : - - or □/+
  - null if you go for another processor family
    - other instruction set, other addressing modes
    - other set of registers
  - average-to-good within a family (upwards compatibility)
    - the new processor is probably faster
    - code has to be written if you want to take advantage of new instructions (e.g. hardware multiplier)
- libraries : -
  - not always in source code
  - difficult to interface
  - chiefly for µC et DSP (telecom, motor control, signal processing)
  - sometimes free (forum of developers, manufacturer of the µC)
  - sometimes sold with the hardware to control

Real-time programming

C

- speed : +/++
  - High-Level-Language (HLL) with low-level instructions (<<)
  - "register" variables
  - coding style can influence the performances but excellent optimized compilers
  - run-time not heavy
  - macros or inlining of ASM can accelerate the execution
- compactness : +/++
  - very good optimisers
- debug : □/+
  - + modular language
  - □ allows "low-level" manipulations => fatal errors are possibles
    - direct manipulation of pointers
    - *typecasting* sometimes dangerous (16bits=>8bits keeps LSByte)

Real-time programming

C

- maintenance: +/++
  - /!\ can be very cryptic

```
#include <stdio.h>
main(t,_,a)
char *a;
{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86, 0, a+1 )+a)):1,t<_?main(t+1.,_, a ):3,main ( -94, -27+t, a )
&&t == 2 ?_<13?main ( 2, _+1, "%s %d %d\n" ):9:16:t<0?t<-72?.main(_, t,
"@n'+.#'/*{}w+/v#cdnr/+,{}r/*de}+./*{*+./v{*/*+,/v#q#n+,/#{1,+,/n{n+,/+#n+,/#;\
#q#n+,/+k#;*+,/'r :1d*13,Xw+Kw'K:'+}e#l;dq#11 q#l+d'K#!/+k#;\
q#lr}eKK#}wlr}eKK{nlll/#;#q#n,){}#}w'){}{nlll/+#nl;d}rw1i;# Knll!/n{n#l; \
r{#vlr nc{nll'/#{1,+lK{rvl iK{;[{nll'/v#q#\
nlwk nw' iwk{KK{nll!/v{%11##w#'i; :{nlll/*{q#lld;r'}{nlwb!/*de}lc;;\
{nll-{}rvlll+,}##'*}#nc,',#nvll/+kdl+e}+;\
#lrdq#v! nrl/ 1) }+}{rl#l{n,l)# }'+}##(!!/")
:t<-50?-==*a ?putchar(a[311):inain(-65,_,a+l):main((
*a == 1/1) + t, --- a +1 M<t?main ( 2, 2 , "%s"):*a==,/,ilmain(O,main(-61,<
*a, "!ek;dc iCbK'(q)-[w]*'/.n+r3#1,{}:\nuwloca-D;mvpbks,fxntdCeghiry"),a+l);}
```

see http://www.es.ioccc.org/main.html

Real-time programming

C

- portability : ++
  - normalized by ANSI
  - few extensions are required
  - extensions easily identified
  - ! µC : SFR => reduces portability
- libraries : +
  - communication, signal processing, control
  - OS : lots of RTOS for µC
  - numeric computation (mainly for DSP)
  - I/O sold with data acquisition hardware

The portability of C is good in computers, but the problem is somewhat different in micro-controllers, because they contain a lot of peripherals having their own structure and SFRs (Special Function Registers).

Let us take the example of 2 µC (event from the same manufacturer). We want to use an analog input and the analog-to-digital converter (ADC). Many differences can appear
- the input pins are not the same
- the analog multiplexer works not in the same way
- the ADC has not the same precision, uses different registers for configuration and for data
- the timer controlling the sampling frequency is not the same
- the interrupt request is not the same

All the hardware-dependent code has to be rewritten.

Real-time programming

C++ used as a better C

- syntax is improved
  - easier to read
  - more rigorous
  - avoid errors
- current "C" compiler
  - generally accept several C++ improvements
  - be carefull and read documentation to
    - avoid compilation errors
    - avoid unexpected results

Real-time programming

C++ in OOP: advantages

- needs a preliminary analysis => long time benefit
- clean modular structure
- reusability of the code
- good framework for libraries
- maintenance
- a data structure and its methods can be changed without changing the rest of the program
- protection of data by the encapsulation
  - data can only be accessed by ad-hoc functions
  - those functions can include validation

Real-time programming

C++ in OOP: drawbacks

- needs more time before beginning to write the code
  - generally turns into an advantage at a long term
- code is not as efficient as in procedural programming
  - size ↗ and speed ↘
  - why:
    - more initialisations, more space is required for dynamic memory
    - run-time overhead
    - more function calls
  - also depends on compiler optimizer
- availability
  - OK for all µP
  - not for every embedded processor and microcontroller (ex: PIC)
- process control libraries
  - procedural programming is well-adapted to control
  - historically, lots of libraries of "functions" exist
  - reuse them in OOP requires transposition in classes

Real-time programming

Java: lot of advantages

- well-known language
  - half of the young programmer begin with Java
- security (designed for network environment)
  - no pointers, no direct access to memory
  - runs on a virtual machine "managed execution" => lots of verifications at run-time (indexes, typecasting, stack overflow)
- modularity
- portability
  - on any platform provided the virtual machine exists
  - testing the code is possible even if hardware is not ready
- *bytecode* extremely compact
- *applets* can be downloaded when required
- ∃ "light" versions: Personal Java, Embedded Java

Real-time programming

Java: drawbacks for real-time embedded applications

- slow speed
  - more critical on small embedded processors
  - due to manage execution in standard
    - the program that actually runs is the virtual machine
    - the bytecode is interpreted (i.e. translated in machine code) "on-the-fly" each time it is executed
    - the virtual machine can decide to compile frequently used parts of the bytecode in a cache
- execution is not deterministic (even stops !)
  - compilation of a function when it is called
  - start of the garbage collector (clean the heap)
  - loading an applet
  - management of exceptions

Real-time programming

Java: future in real-time ?

- ∃ µP that can run native bytecode
  - ARM family with Jazelle extension
  - faster hardware virtual machine
- pre-compilation
  - at run-time slows down the start of the application
  - before run-time
  - no more interpretation afterwards
- ∃ RTSJ Real-Time Specification for Java
  - not for small machines currently
    - Dual core or dual CPU system with 512 MB
    - Real-Time OS:
      - Solaris 10 (Update 6, Update 7)
      - SUSE Linux Enterprise Real Time 10 Service Pack 2
      - Red Hat Enterprise MRG

Real-time programming

C#.NET

- same philosophy as JAVA
  - managed execution (virtual machine) common with VB
  - sporadic garbage collection
- efficiency
  - C# is 5 times slower than C (toggling I/O pin)
  - overhead due
    - to extra function calls
    - not to the virtual machine
  - fast access to I/O is possible by recreating pointer and making direct access without function call
- works well on WinCE
- limited to the Microsoft world

Programming languages

CONTENTS

- introduction
- selection criteria
- MMI
- **real-time**
  - description and comparison
  - **ranking**
- conclusions

Real-time ranking

1st choice: C

- advantages
  - available for every µP and µC
  - sometimes the only available HLL
  - well-optimized fast and compact code
  - procedural programming is well-adapted to control
- if not fast enough
  - macros
  - critical parts in ASM
  - change for another µP
  - some task executed on an FPGA
  - multi-processor architecture
- C not always allowed (MIL, space)
  - not rigorous enough
  - direct manipulation of pointers

Real-time ranking

first accessits

- C++
  - slightly less efficient than C
  - available for many processors
  - C or C++ is chiefly a question of experience
- ADA
  - U.S. Army standard
  - strict types, very safe, but heavy
  - MIL, space, avionics, railways
  - adapted to real-time multi-tasking control

Real-time ranking

only if you can't avoid it

- ASM
  - quickest and most compact code (potentially)
  - hardest
  - longest
  - most expensive

  } to develop
  - use only for parts of the code
    - critical functions
    - library functions

Programming languages

CONTENTS

- introduction
- selection criteria
- MMI
- real-time
  - description and comparison
  - ranking
- **conclusions**

Conclusions

- industrial programming is not a sport
  - performance/price ratio of µP is continuously rising
  - ∃ excellent compilers
  - => C on a faster µP should be preferred to ASM on a cheap µP
  - work with a maximum of code already debugged
- devote your efforts and time to
  - the application itself
  - documentation and facilitation of the maintenance
  - meet standards and quality labels (ISO 900X)
- break those rules only for
  - special needs (speed, size, security, safety, norms, cost, confidentiality)
  - large series of production