EECE 276
Embedded Systems

Real-time programming languages
Basics
Real-time Programming Languages

- Programming languages: the nexus of design and structure.
- Tools:
  - Compiler
  - Editor/Debugger/Analyzer/Generators/…
- Language categories:
  - Procedural: C, Fortran, Pascal, Ada 83
  - Object-oriented: C++, Java, Smalltalk, C#, Ada 95
Questions about languages

- **Economy of execution:**
  - How fast does the program run?

- **Economy of compilation:**
  - How long does it take to go from sources to executable?

- **Economy of small-scale development:**
  - How hard must an individual programmer work?

- **Economy of large-scale development:**
  - How hard must a team of programmers work?

- **Economy of language features:**
  - How hard is it to learn or use a programming language?
Language issues: Assembly

- Assembly language
  » Difficult to learn, error prone
  » Highly efficient code
    – Compilers are getting better!
  » Practice: 99% is in high-level language.
  » In summary:
    – Excellent economy of execution and compilation
    – Poor economy of development
Language issues: Procedural

- Basic abstraction: *procedure* (hierarchical!)
- Parameter passing techniques
  - Call-by-value vs. call-by-reference:
    - Copy data onto stack/pass a reference to data via stack
  - Performance implications
- Global variables
  - Sharing
  - Data access consistency
    - COMMON in Fortran
    - Pointers in C
Language issues: Procedural

- **Recursion**
  - Elegant, mathematical
  - Unbounded? Run-time costs? Stack overflow?

- **Dynamic memory allocation**
  - Dynamic data structures: lists, trees, queues, etc.
  - Allocation and de-allocation
  - Costs in real-time systems
  - Hard RT system practice: NO! (only at initialization time)
Language issues: Procedural

- Typing: enforcing type compatibility rules
- Example (C, with automatic type conversion):
  ```c
  int x,y; float k,l,m;
  ...
  y = x * k + m; /* NO CHECK FOR OVERFLOW */
  ```
- Ariane-5:
Language issues: Procedural

- Exception handling:
  - Mechanism to change control flow upon errors
    - Non-local transfer of control, possibly across procedure levels
  - C: Call handler when signal is “raised”
    - Signal handler: void * signal (int S, void(*func)(int));
    - Signal trigger: int raise (int S);
  - Ada: Exception handling block
    ```ada
    begin
      -- calculate
      exception
        when SINGULAR: NUMERIC_ERROR => put("Matrix singular");
        when others => put("Fatal error");
        raise ERROR;
    end
    ```
Language issues: Procedural

- **Modularity: support for large-scale development**
  - C: not much
  - Ada “packages”: information hiding
    - Specification and declaration

- **Metrics in procedural languages**
  - Execution: good - optimizing compilers
  - Compilation: good
  - Development:
    - Type checking/systems
    - Orthogonality (composable features)
    - Modularization

- **Note:** Common procedural languages do not provide support for real-time programming on the language level.