

J.6 Programmierung eingebetteter Systeme

■ Vorteile von C in eingebetteten Systemen:

- leichter Zugriff auf die Hardware
- gute Kontrolle über die verwendeten Ressourcen (Speicher, CPU)
- effizienter Code
- geringer Speicherbedarf

■ Probleme mit C:

- stark eingeschränkte Laufzeitüberprüfungen
- ISO C-Standard oft nicht eindeutig
Verhalten ist implementationsabhängig oder undefiniert
- in C kann man viele Fehler machen (v. a. Anfänger)

1 MISRA C

■ 1998 von der MISRA entwickelt

(MISRA = Motor Industry Software Reliability Association)

■ Richtlinien für den Gebrauch von C in sicherheitskritischen Systemen (1998, 2004 überarbeitet)

■ Regeln, die den Umgang mit bestimmten Konstrukturen in C präzisieren z. B. bei Zeigeroperationen

- 141 Regeln (121 verbindlich, 20 empfohlen)

◆ Beispiele

- Anweisungen nach if, while, ... müssen immer in {} geklammert sein
- spezielle Typen (z.B. uint32_t) statt der Standardtypen

■ ursprünglich Automobilbereich, seit 2004 neue verallgemeinerte Version

2 Java für eingebettete Systeme?

■ mögliche Vorteile:

- objektorientiert
- typsicher

3 Objektorientierte Programmierung

■ Zusammenfassung von Funktionen und ihren Daten

- Strukturierung

■ Vorteile / Nachteile

- Zusammengehörendes kommt zusammen
- kaum noch globale Variablen
- höhere Abstraktion möglich
- weniger Kontrolle über Ressourcenverbrauch
- Werkzeuge müssen für eingebettete Systeme optimiert sein

4 Typsichere Sprachen

■ Beispiele: Java, .NET-Sprachen (z. B. C#)

■ Grundprinzip:

- allen Werten ist immer ein Typ zugeordnet
- Umwandlung in einen anderen Typ ist nicht ohne Weiteres möglich

■ Umwandlung muss möglicherweise zur Laufzeit geprüft werden

- keine Zeigeroperationen

■ Automatische Speicherbereinigung möglich

■ Typsystem zur Laufzeit nötig

- meist interpretiert

4 Typsichere Sprachen (2)

- Vorteile / Nachteile:
 - mehr Ressourcenbedarf durch Laufzeitüberprüfungen
 - schlechtes Laufzeitverhalten bei Interpretation
 - + Interpretation ermöglicht leichte Erweiterbarkeit zur Laufzeit (dynamisches Nachladen von Klassen)
 - + Typsicherheit ermöglicht die Kapselung der Anwendung ohne Hardwareunterstützung (vertrauenswürdiger Compiler!)

- Für tief eingebettete Systeme ist Ressourcenbedarf wichtig
 - starke Einschränkung der dynamischen Erweiterbarkeit
 - compilieren zu statischem Programm

6 Modellbasiertes Design

- MDA: Model Driven Architecture
- Beschreibung der Anwendung in Form eines Modells auf einer abstrakten Ebene
- Konkretes Programm wird von einem Werkzeug erstellt
- Vorteile/Nachteile
 - + architekturunabhängig
 - + Konzentration auf die Funktionalität der Anwendung
 - +/- Detailwissen zur Implementierung in den Werkzeugen
 - Werkzeug für die Zielplattform wird benötigt
- Beispiel: Labview, Simulink

5 Java für eingebettete Systeme

- Java Micro-Edition (J2ME):
 - schlanke JVM und angepasste Klassenbibliothek
 - als J2ME/CLDC-MIDP auf den meisten Handys
 - als J2ME/CDC auf einigen PDAs

- JavaCard: stark abgespeckte JVM
 - für Chipkarten

7 Familienbasierter Ansatz

- Programme haben meist eine Vielzahl von Funktionen
- Nicht immer wird alles benötigt
 - nur das auswählen was für die aktuelle Aufgabe benötigt wird

- Vorteile / Nachteile
 - + Ressourceneinsparung, da nur benötigte Funktionalität vorhanden
 - + durch Werkzeugunterstützung ist trotzdem eine komfortable Codeverwaltung möglich
 - Abhängigkeiten zwischen den Funktionen müssen erfasst und berücksichtigt werden