6.1 Grundbegriffe

Vorgehensweisen bei der Erstellung der Programme

Arten von Programmiersprachen

Vorgehensweisen bei der Erstellung der Programme

Speicherprogrammierbare Steuerungen

- ❒ textuelle Programmiersprachen
- ❒ graphische Programmiersprachen

Mikrocontroller

- ❒ Assembler
- ❒ niedere maschinenunabhängige Programmiersprachen

PC und IPC

- ❒ Softwarepakete
- ❒ universelle Echtzeitprogrammiersprache

Prozessleitsysteme

- ❒ Funktionsbausteintechnologie

6.1 Grundbegriffe

Vorgehensweisen bei der Erstellung der Programme

Arten von Programmiersprachen

Klassifikation nach Art der Notation

- textuelle Programmiersprachen *Ada, C, SPS-Anweisungsliste*
- graphische Programmiersprache *SPS-Kontaktplan*

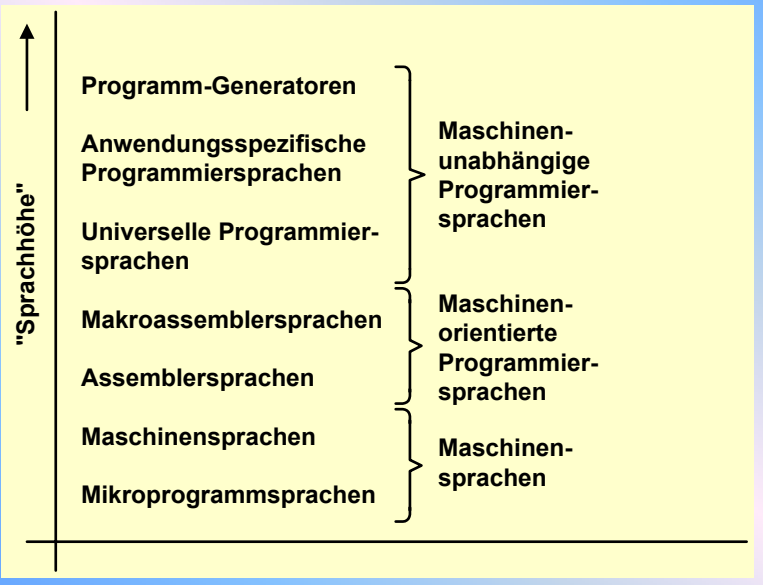
Klassifikation nach dem Programmiersprachenparadigma

- prozedurale Programmiersprachen C, Ada 83
- funktionale Programmiersprachen LISP
- logische Programmiersprachen PROLOG
- objektorientierte Programmiersprachen C++, Smalltalk, Ada 95

Klassifikation nach der Sprachhöhe

- **hoch:** an der Verstehbarkeit durch den Menschen orientiert
- **nieder:** an den Hardware-Eigenschaften eines Computers orientiert

Klassifizierung von Programmiersprachen nach der "Sprachhöhe"



Mikroprogrammsprachen

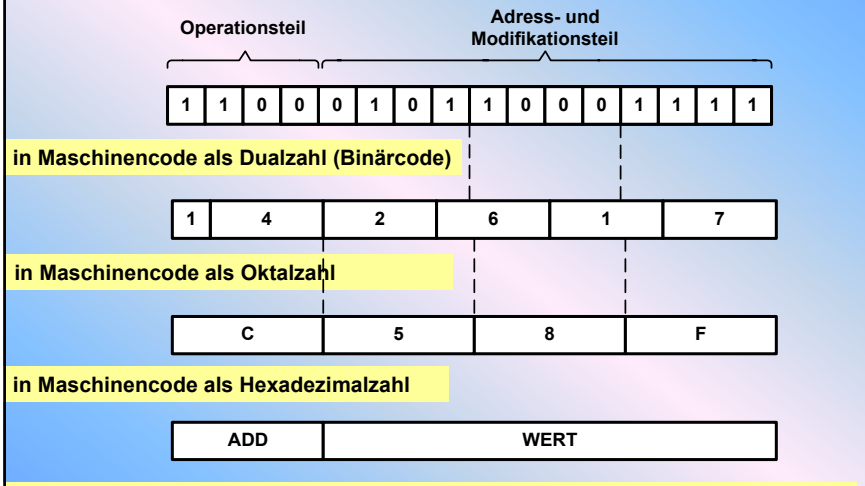
- Realisierung der Ablaufsteuerungen zur Ausführung der Maschinenbefehle
  - fest verdrahtete Verknüpfungsglieder
  - Mikroprogramme
- Mikroprogramme (Firmware)
  - Speicherung in schnellen Schreib-/Lesespeicher (RAM's) oder in Festwertspeicher (ROM's)
  - nicht zugänglich für Anwenderprogrammierung
- Maschinensprachen
  - Sprachelemente: **Befehle und Daten in Form von Bitmustern**
  - Zusammenfassung als Oktal-bzw. Hexadezimalzahl
  - mühsame Handhabung
  - nicht für Anwendungsprogrammierung

Assemblersprachen

Vermeidung der mühsamen Handhabung der Maschinensprachen unter Beibehaltung der Eigenschaften der Maschinenbefehle

- Ersetzung der oktalen/hexadezimalen Schreibweise des Operationsteils der Befehle durch symbolische, mnemotechnisch günstige Buchstabenabkürzungen
- Einführung eines symbolischen Namens anstelle der zahlenmäßigen Darstellung des Adressteils
- Eindeutige Zuordnung zwischen den Befehlen der Assemblersprache und den Befehlen der Maschinensprache
- Abhängigkeit von gerätetechnischen Eigenschaften der jeweiligen Rechenanlage

Beispiel für die Darstellung eines Maschinenbefehls



Makroassemblersprachen (Makrosprachen)

- weiteres Hilfsmittel zur einfacheren Handhabung: Makros

Makro: **Abkürzung für eine bestimmte Befehlsfolge**

- Unterscheidung
  - Makrodefinition
  - Makroaufruf
  - Makroexpansion
- Aufbau einer Makrodefinition

  MAKRO Makroname (P_1, P_2, ..., P_N)
  Makrokörper
  Endzeichen

  Makroaufruf
  Makroname (A_1, A_2, ..., A_N)
- Eindeutige Zuordnung von Makroassemblerbefehlen zu Befehlen der Maschinensprache
- Einem Makrobefehl entsprechen mehrere Maschinenbefehle

Beispiel Makro Dreiadressadd

Makrodefinition:
```
MAKRO Dreiadressadd (P1, P2, P3)
LADE P1
ADDIERE P2
SPEICHERE P3
ENDE
```

Makroaufruf: Dreiadressadd (A,B, SUMME)

Makroexpansion:
```
...
LADE A
ADDIERE B
SPEICHERE SUMME
...
```

Unterschied zwischen Unterprogramm-Aufrufen und Makros

- Unterprogramm wird nur einmal gespeichert, kann mehrfach aufgerufen und verwendet werden
- Makro wird an jeder Stelle, an der es aufgerufen wird, expandiert

Klassifizierung von Makros

- Standardmakros: fest vorgegeben
- Anwendermakros: vom Anwender selbst definierbare Makros für häufig vorkommende Befehlsfolgen

Universelle Programmiersprachen

universell ≙ nicht auf ein Anwendungsgebiet ausgerichtet

universelle niedere Programmiersprachen

Systemprogrammier-sprachen

- Zweck: Erstellung von Systemprogrammen
  - Compiler
  - Betriebssysteme
  - Editoren
  - Treiberprogramme
- Ziel: 1. Ausnutzung der Hardwareeigenschaften
  2. Portabilität
- Beispiel: C

universelle höhere Programmiersprachen

- Zweck: Erstellung von allgemeinen Programmen
- Ziel: 1. einfache Formulierbarkeit
  2. umfangreiche Compilerprüfungen
  3. Portabilität
- Beispiel: Ada, Java, Smalltalk

Anwendungspezifische Sprachen

Deskripitive Sprachen, nicht-prozedurale höhere Sprachen, very high level languages

Unterscheidung zu prozeduralen Sprachen

- ❒ keine Beschreibung des Lösungsverfahrens, sondern Beschreibung der Problemstellung selbst
- ❒ Einschränkung auf bestimmtes Anwendungsgebiet

Bsp.:	FUP	Funktionsplan
	KOP	Kontaktplan
	AWL	Anweisungsliste
	EXAPT	für Werkzeugmaschinensteuerungen
	ATLAS	für automatische Prüfsysteme

Vorteile/Nachteile:

\+ **bequeme, anwendungsspezifische Ausdrucksweise**

\- **Inflexibilität**

Programm-Generatoren (Fill-in-the-blanks-Sprachen)

- ❒ Formulierungsverfahren für Programme
- ❒ Beantwortung von Fragen in Form von Menues am Bildschirm durch den Anwender (Konfigurierung)
- ❒ Umsetzung der Antworten durch den Programm-Generator in ein ausführbares Programm
- ❒ Vorteil: keine Programmierkenntnisse notwendig
- ❒ Nachteil:
  - ❋ **Einengung auf bestimmtes Anwendungsgebiet**
  - ❋ **Abhängigkeit von einem bestimmten Hersteller**

Anwendungsgebiete von Programm-Generatoren im Bereich Prozessautomatisierung

- Leittechniksysteme in der Energie-und Verfahrenstechnik
  - TELEPERM-M (Siemens)
  - PROCONTROL-B (ABB)
  - CONTRONIC-P (Hartmann & Braun)
- Speicherprogrammierbare Steuerungen in Form von Anweisungsliste oder Kontaktplan
  - SIMATIC (Siemens)

6.2 Höhere Programmiersprachen für die Echtzeitprogrammierung

Problematik der Echtzeitprogrammierung

Vor- und Nachteile der Assemblerprogrammierung

Entwicklungsrichtungen zur Anwendung maschinenunabhängiger, universeller Echtzeit-Programmiersprachen

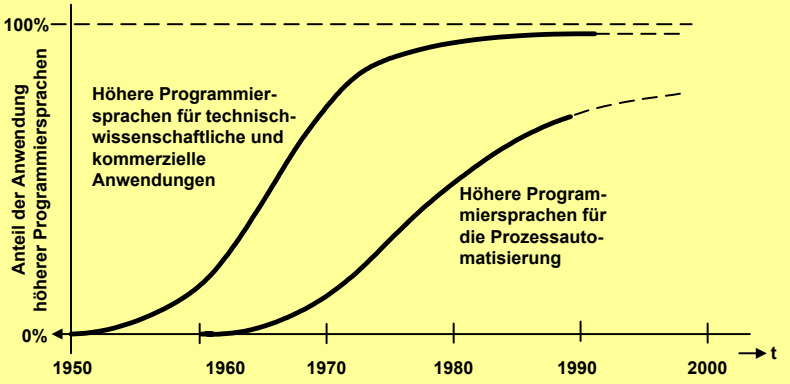
Problematik der Echtzeit-Programmierung

- Höhere Programmiersprachen wie BASIC, FORTRAN, COBOL, PASCAL sind für Echtzeit-Programmierung ungeeignet
  - **keine Echtzeitsprachmittel**
  - **keine Einzelbitoperationen**
  - **keine Befehle für Prozess-Ein-/Ausgabe**
- Einsatz von höheren Programmiersprachen im Vergleich zu Assemblersprachen bringt Erhöhung von Speicherbedarf und Rechenzeit mit sich
  - **Produktautomatisierung:** **Speicherplatz und Rechenzeit kritisch**
  - **Anlagenautomatisierung:** **Rechenzeit unter Umständen kritisch**

Verfügbarkeit

- **Compiler für Zielrechner**
- **Echtzeit-Betriebssystem**

Vergleich der Verwendung höherer Programmiersprachen



6.2 Höhere Programmiersprachen für die Echtzeitprogrammierung

Problematik der Echtzeitprogrammierung

Vor- und Nachteile der Assemblerprogrammierung

Entwicklungsrichtungen zur Anwendung maschinenunabhängiger, universeller Echtzeit-Programmiersprachen

Vor- und Nachteile der Assemblerprogrammierung

+	Speicher- und Rechenzeit-Effizienz
-	höhere Programm-Entwicklungskosten
-	geringe Wartbarkeit
-	Probleme mit der Zuverlässigkeit
-	schlechte Lesbarkeit, geringer Dokumentationswert
-	fehlende Portabilität

Einsatz von Assemblersprachen

ja: Automatisierung von Geräten oder Maschinen, wo es um Serien- oder Massenanwendungen geht, kleine Programme

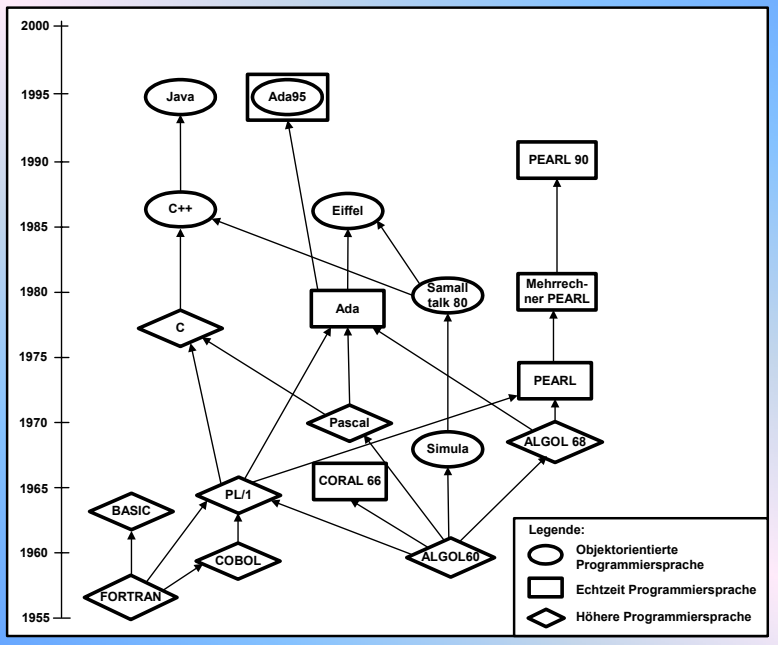
nein: langlebige, große zu automatisierende Prozesse

6.2 Höhere Programmiersprachen für die Echtzeitprogrammierung

Problematik der Echtzeitprogrammierung

Vor- und Nachteile der Assemblerprogrammierung

Entwicklungsrichtungen zur Anwendung maschinenunabhängiger, universeller Echtzeit-Programmiersprachen



6.3 Programmierung von Speicherprogrammierbaren Steuerungen (SPS)

Ausgangssituation

Zunahme der Funktionalität und Leistungsfähigkeit von speicherprogrammierbaren Steuerungen

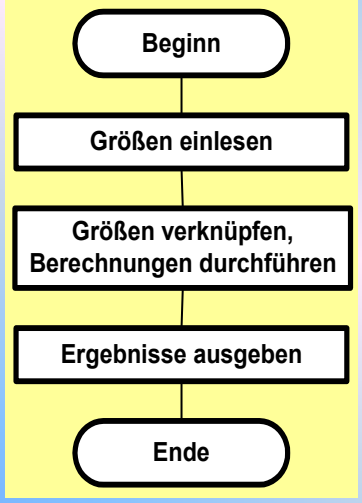
1980	:	2	kByte
1990	:	20	kByte
2000	:	2000	kByte

Standardisierung der Entwicklung von SPS-Systemen
IEC 1131
DIN EN 611131

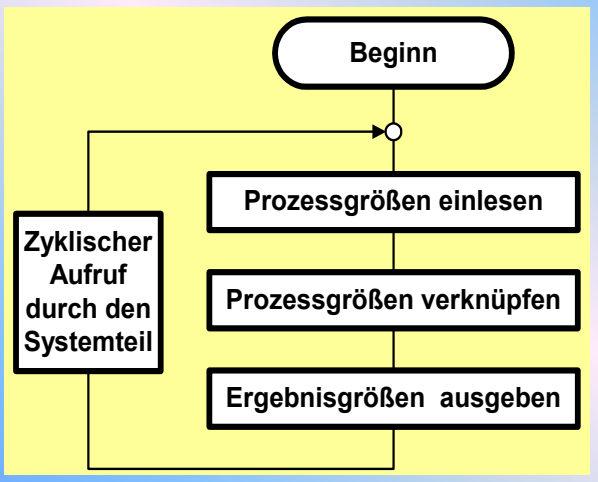
6.3 Programmierung von Speicherprogrammierbaren Steuerungen (SPS)

Programmablauf in SPS-Systemen

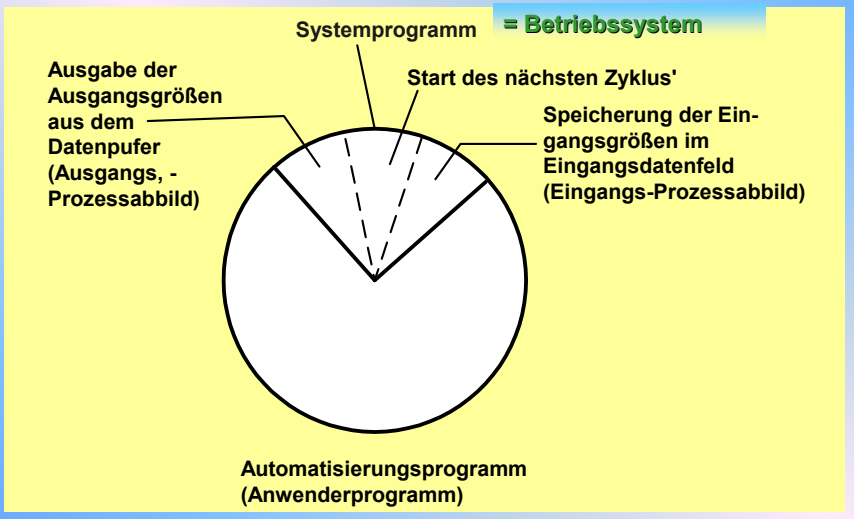
Programmablauf bei einem üblichen Computer



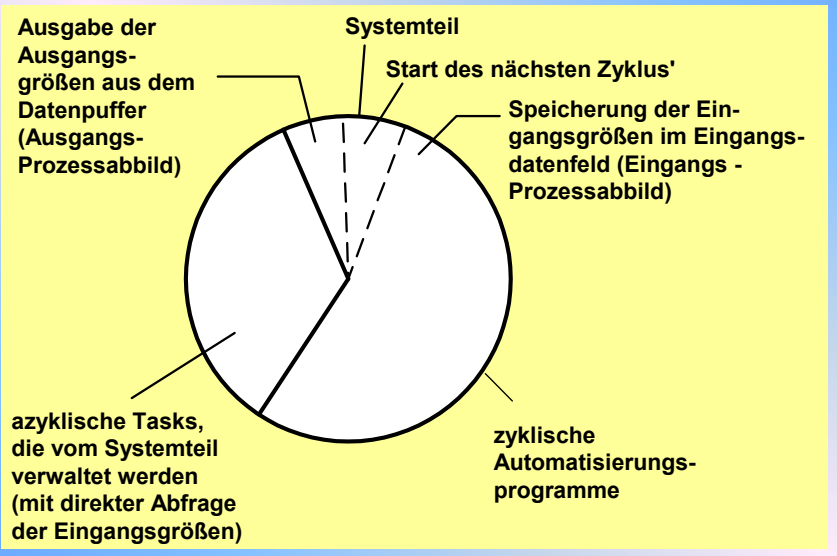
Zyklischer Programmablauf bei einer SPS



Verarbeitung ausschließlich zyklischer Automatisierungsprogramme



Verarbeitung von zyklischen und azyklischen Automatisierungsprogrammen



6.3 Programmierung von Speicherprogrammierbaren Steuerungen (SPS)

- Ausgangssituation
- Programmablauf in SPS-Systemen
- Programmiersprachen für SPS-Systeme
- Programmorganisation

Programmiersprachen für SPS-Systeme

IEC 1131:
keine Programmiersprache vorgeschrieben

4 Grundsprachen mit genormter Syntax

❒	**AWL**	Anweisungsliste	*textuell*
❒	**ST**	Strukturierter Text	*textuell*
❒	**KOP**	Kontaktplan	*graphisch*
❒	**FBS**	Funktionsbausteinsprache	*graphisch*

Beispiel:
Ausgang 1(Aus1) und Ausgang 2 (Aus2) sind nur dann wahr, wenn entweder Eingang 3 (Ein3) wahr ist oder wenn die beiden Eingänge 1 (Ein1) und 2 (Ein2) gleichzeitig wahr sind

Anweisungsliste (AWL) (IL Instruction List)

Aufbau einer Anweisungszeile

```
Marke (optional)   Operator    Operand      Kommentar (optional)
START:             LD          Ein1         (* Beispielanweisung*)
```

Beispiel in AWL

```
LD    Ein1
AND   Ein2
OR    Ein3
ST    Aus1
ST    Aus2
```

Bitbefehle:	LD, ST, AND, OR, XOR
Arithmetikbefehle:	ADD, SUB, MUL, DIV
Ablaufsteuerung:	IMP, RET, CAL

Strukturierter Text (ST)

Beispiel in ST

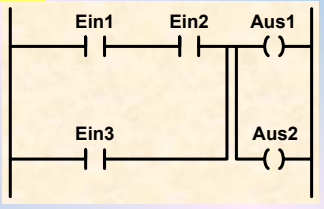
```
Aus1 :=      Ein1 OR ( Ein2 AND Ein3 );
             Aus 2 := Aus1;
```

Steuer-anweisungen in ST

Anweisung	Bemerkung
FB_Name (IN := Schalter1);	Aufruf von Funktionsbausteinen
RETURN;	Verlassen einer Funktion/Funktionsbausteins
IF ... END_IF;	If-Verzweigung
CASE ... END_FOR;	Case-Verzweigung
FOR ... END_WHILE;	For-Schleife
WHILE ... END_WHILE;	While-Schleife
REPEAT ... END_ REPEAT;	Repeat-Schleife, mindestens ein Durchlauf
EXIT;	verlassen der Schleife

Kontaktplan (KOP)

Beispiel in KOP



von links nach rechts

von oben nach unten

Prinzip:

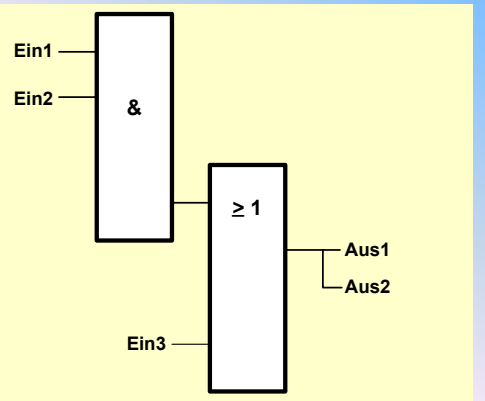
Nachahmung von verdrahteten Logikschaltungen

Eigenschaften

- ❒ einfache Darstellung
- ❒ Negation nicht darstellbar
- ❒ komplexe mathematische Funktionen schlecht darstellbar

Funktionsbausteinsprache (FBS)

Beispiel in FBS



Eigenschaften

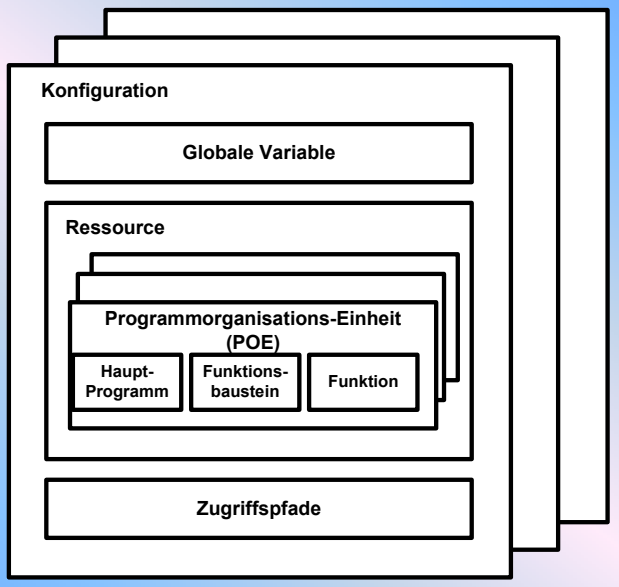
- ❒ viele Gemeinsamkeiten mit KOP
- ❒ übersichtliche Darstellung

6.3 Programmierung von Speicherprogrammierbaren Steuerungen (SPS)

- Ausgangssituation
- Programmablauf in SPS-Systemen
- Programmiersprachen für SPS-Systeme
- Programmorganisation

Programmorganisation

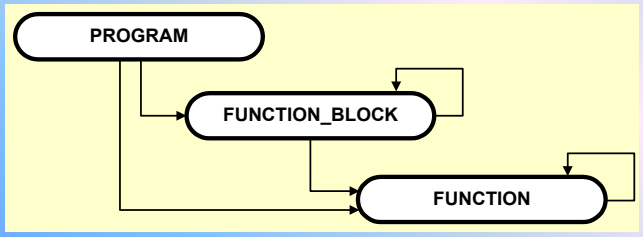
SPS-Programm-system



Beispiel

```
FUNCTION-BLOCK Zaehler
        externe Variablen
        lokale Variablen
        Bausteinrumpf
                Befehle in der jeweiligen
                Programmiersprache
END_FUNCTION_BLOCK
```

Zulässige Bausteinaufrufe



Bausteinvariablen

Variablenarten

Variablenart	Erklärung	verwendet in
VAR	lokale Variable	PROGRAM; FUNCTION_BLOCK; FUNCTION
VAR_INPUT	Eingangsvariable	PROGRAM; FUNCTION_BLOCK
VAR_OUTPUT	Ausgangsvariable	PROGRAM; FUNCTION_BLOCK
VAR_IN_OUT	Ein- oder Ausgangsvariable	PROGRAM; FUNCTION_BLOCK
VAR_EXTERNAL	externe Variable	PROGRAM; FUNCTION_BLOCK
VAR_GLOBAL	globale Variable, für alle Bausteine gültig	PROGRAM
VAR_ACCESS	Zugriffspfad auf Konfiguration	PROGRAM

Variablen-vereinbarung

```
VAR_IN                  (* Eingabevariblen *)
ventil3 :  BOOL;
messwert5 : INT :=146  (* Initialisierungswert 146 *)
END_VAR
```

Zulässige Datentypen

BOOL	Boolsche Zahl	INT	Ganzzahl
REAL	Gleitpunktzahl	TIME	Zeitdauer
DATE	Datum	TOD	Uhrzeit
DT	Datum und Uhrzeit	STRING	Zeichenfolge
BYTE	8-bit-Folge	WORD	16-bit-Folge

6.4 Echtzeitprogrammiersprache Ada 95

- **Entstehungsgeschichte von Ada**
- **Sprachkonstrukte für die algorithmische Programmierung**
- **Sprachkonstrukte für die Echtzeitprogrammierung**
- **Spracherweiterungen in Ada 95**

Entstehungsgeschichte von Ada

Ada
Name der 1. Programmiererin
Zu Ehren der Mathematikerin Augusta Ada Byron, Countess of Lovelace, Tochter von Lord Byron benannt. Ada Lovelace (1815- 1851) arbeitete mit Charles Babbage an seiner “Difference-and Analytic-Engine”.
Auf Ada Lovelace geht die Idee zurück, diese Maschine mit Lochkarten zu programmieren.

Entwicklung von Ada

1975	Gründung von HOLWG zur Entwicklung einer Programmiersprache für das DOD (USA) für “embedded systems”
1980	Festschreibung der Sprachdefinition, Ada 80
1983	ANSI Ada Standard, Ada 83
1987	ISO-Norm für Ada
1988	DIN-Norm (DIN 66268)
1995	Festschreibung von Ada9X zu Ada95, Spracherweiterungen zur Unterstützung der objektorientierten Programmierung

Eigenschaften von Ada

- Eignung für sehr umfangreiche Softwaresysteme bis über 10^7 Zeilen
- Modularisierungskonzept, das eine koordinierte, parallele Entwicklung ermöglicht
- “State of the art”-Sprachkonzepte zur Unterstützung von Überprüfungen zur Übersetzungszeit
- Wiederverwendbarkeit von Softwarekomponenten
- Sichere, moderne und effiziente Echtzeit-Konzepte
- Durchgängiges Fehlerbehandlungskonzept mit Laufzeitüberprüfungen
- Vereinbarkeit der Portierbarkeit mit Maschinennähe
- Internationale Normung von Syntax und Semantik
- Prüfung jedes Ada-Compilers in einer umfangreichen Validierungsprozedur auf die Einhaltung der Norm (mehrere tausend Testprogramme)

6.4 Echtzeitprogrammiersprache Ada 95

Entstehungsgeschichte von Ada

Sprachkonstrukte für die algorithmische Programmierung

Sprachkonstrukte für die Echtzeitprogrammierung

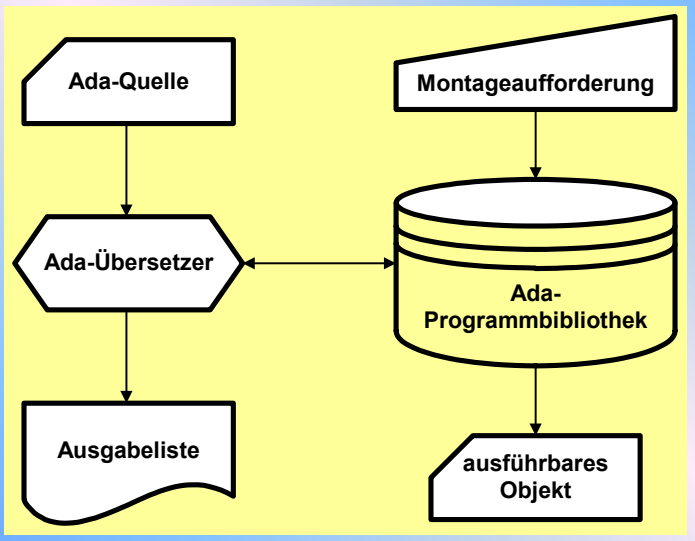
Spracherweiterungen in Ada 95

Sprachkonstrukte für die algorithmische Programmierung

Programmstruktur

- Übersetzbare Einheiten
  - Unterprogramme
    - Funktionen
    - Prozeduren
  - Pakete (Module)
- Aufbau von Übersetzungseinheiten
  - Vereinbarungen
  - Anweisungen
  - Pragmas zur Steuerung des Übersetzungsvorgangs

Bibliotheksbezogene Programmgenerierung



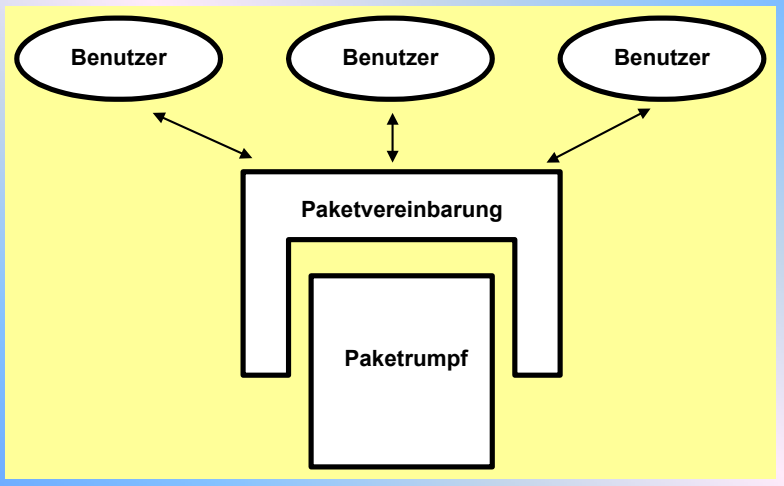
Programmeinheiten

- Pakete (Module)
- Tasks (Rechenprozesse)
- protected units (Monitore)
- generische Einheiten
  (parametrisierte Module und Unterprogramme)

Aufbau von Programmeinheiten

- **Spezifikation:** Schnittstelle nach außen
- **Rumpf:** Realisierung der Programmeinheit

Paketvereinbarung, Paketrumpf und ihre Benutzer



Aufbau eines Pakets

sichtbarer Teil

nach außen nicht sichtbar

Paketvereinbarung (package declaration)

```
PACKAGE paket_name IS

Vereinbarungen
(Typen, Objekte,
Unterprogrammm-
Vereinbarung
...)

PRIVATE

privater Teil
(Vereinbarungen)

END paket_name;
```

Paketrumpf (package body)

```
PACKAGE BODY paket_name IS

Weitere Vereinbarungen

Unterprogramm-Rümpfe

BEGIN

Anweisungen
(werden bei der
Abarbeitung des
Paketrumpfes
ausgeführt)

END paket_name;
```

Verein-barungen

Anweisungen

Beispiel eines Ada-Programms

```
PACKAGE Stack IS
      PROCEDURE    Push (X: REAL);
      FUNCTION     POP RETURN REAL;
END;
PACKAGE BODY stack IS
      Max:   CONSTANT:=100;
      S:     ARRAY (1..Max) OF REAL;
      Ptr:   INTEGER RANGE 0..Max;
      PROCEDURE    Push (X:REAL)IS
            BEGIN
                  Ptr:= Ptr + 1
                  S (Ptr):= X;
            END Push;
      FUNCTION Pop RETURN REAL IS
            BEGIN
                  Ptr:= Ptr - 1
                  RETURN S (Ptr + 1);
            END pop;
      BEGIN
                  Ptr:= o
      END Stack;
```

```
Aufruf  Stack.Push (Y):
        ...
        A:=     Stack.Pop ();
```


Tasks

- Tasks sind keine Übersetzungseinheiten
- Ablauf autonom und nur lose an den Ablauf anderer Ablaufeinheiten gebunden
- Vereinbarung erfolgt im Vereinbarungsteil eines Blocks, eines Unterprogramms, eines Paketrumpfs oder einer anderen Task

Beispiel:

```
PROCEDURE Arrive_at_Airport is
        TASK Claim_Baggage;
        TASK BODY Claim_Baggage is
        ...
        END;
        TASK Rent_a_Car;
        TASK BODY Rent_a_Car is
        ...
        END;
BEGIN
        Book_Hotel;
END   Arrive_at_Airport
```


Algorithmik

Vereinbarungen

PI: CONSTANT:=3.141_159

Grunddatentypen

INTEGER	10
LONG_INTEGER	
REAL	1.23
LONG_REAL	
CHARACTER	‘A’
STRING	“ABC”
BOOLEAN	FALSE

Zuweisungsschutz

- streng typengebundene Sprache
- alle Bezeichner explizit vereinbart
- Operationen werden festgelegt
- explizite Typkonvertierung

  Meine_Birnen < Birnen (Meine_Aepfel)

Typdefinitionen

- Aufzählungstypen (enumeration type)

```
TYPE    Wochentag IS
        (Montag, Dienstag, Mittwoch, Donnerstag, Freitag,
        Samstag, Sonntag)
```

- ARRAY-Typen (array-type)

```
TYPE Table is ARRAY (INTEGER RANGE 1...6) OF INTEGER
```

- RECORD-Typen (record type)

```
TYPE    Datum IS RECORD
        Jahr:   INTEGER;
        Monat:  Monatsname;
        Tag:    INTEGER RANGE 1...31;
```


- ACCESS-Typen (access type)

```
TYPE Zeiger_auf_einen-Puffer IS ACCESS Puffer;
```

- Abgeleitete Typen (derived type)

```
TYPE Birnen IS NEW Aepfel
```

- Numerische Typen (numeric type)

```
TYPE Float IS DIGITS 6
```

Anweisungen und Kontrollstrukturen

- **Leere Anweisungen**

  NULL

- **Zuweisung**

  Akt_Druck:=0.0;

- **Prozeduraufruf**

  Lies (Gleitkommazahl)

  Schreibe (Objekt ↑ Ganzzahl)

- **Blockanweisung**

  BEGIN Tausch

  ...

  END Tausch

- **Selektion**

  - IF-Anweisung

```
IF Note <= 1.5 THEN
        Schreibe („sehr gut“)
ELSIF Note <= 2.5 THEN
        Schreibe („gut“)
ELSE Schreibe („befriedigend“)
END IF;
```

  - CASE-Anweisung

```
CASE  Punkte IS
        WHEN 85...100     =>   Schreibe („sehr gut“);
        WHEN 75. ..85     =>   Schreibe („gut“);
                ...
        WHEN OTHERS       =>   Schreibe
                               („ungenügend“)
```

- **Schleifen**
  - Endlosschleife
    ```
    LOOP
    ...
    END LOOP
    ```
  - bedingte Schleife
    ```
    WHILE I <= 10 LOOP
    ...
    END LOOP
    ```
  - Laufschleife
    ```
    FOR I IN 1...3 LOOP
    ...
    END LOOP
    ```
- **Sprunganweisung**
  - kontrollierter Schleifenausstieg
    EXIT
  - unkontrollierter Sprung
    GOTO

Ein-/ Ausgabe

- nicht explizit definiert
- Paket mit Standard E/ A-Funktionen wird bereitgestellt
- Bsp.: Get, Put_line, Page, New_Line

Ausnahmesituationen

- vordefinierte Ausnahmesituationen
  - Constraint-Error
    Bsp.: Division durch Null
  - RAISE Constraint-Error

- Behandlung von Ausnahmesituationen
  ```
  EXCEPTION
      When Constraint-Error  => Ueberlauf:= TRUE
  ```

6.4 Echtzeitprogrammiersprache Ada 95

Entstehungsgeschichte von Ada

Sprachkonstrukte für die algorithmische Programmierung

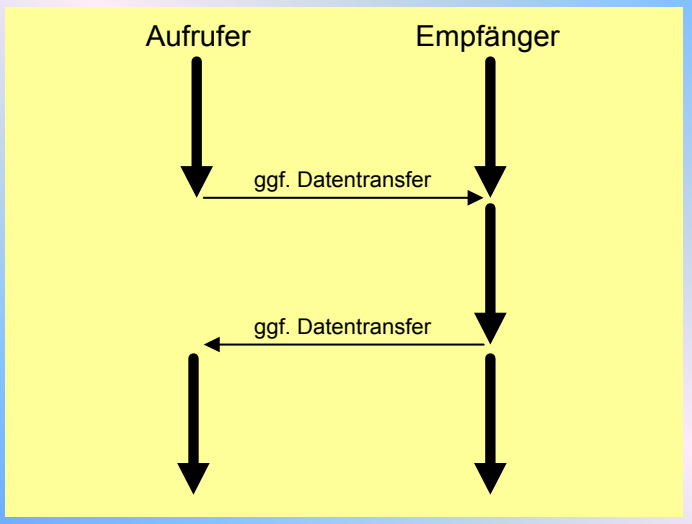
Sprachkonstrukte für die Echtzeitprogrammierung

Spracherweiterungen in Ada 95

Sprachkonstrukte für die Echtzeitprogrammierung

- Tasks
  Aktivierung durch Eintritt in die Umgebung
- Verzögerungsanweisung
  DELAY 3.0;
- Taskabbruch
  ABORT
- Synchronisierung
  - Rendezvous-Konzept
    ENTRY
    ACCEPT auf Empfängerseite
  - selektive Synchronisierung
    SELECT
    Eingangsaufruf
    OR
    Eingangsaufruf
  - zeitbedingtes Warten
    SELECT
    ...
    OR DELAY

Rendezvous-Konzept (1)



Rendezvous-Konzept (2)

Handshake-Verfahren

- Beide Partner müssen warten
- Aus der Sicht des Aufrufers hat das Rendezvous die Gestalt eines Prozeduraufrufs

Aufrufvereinbarung

ENTRY

Aufrufpunkt

ACCEPT

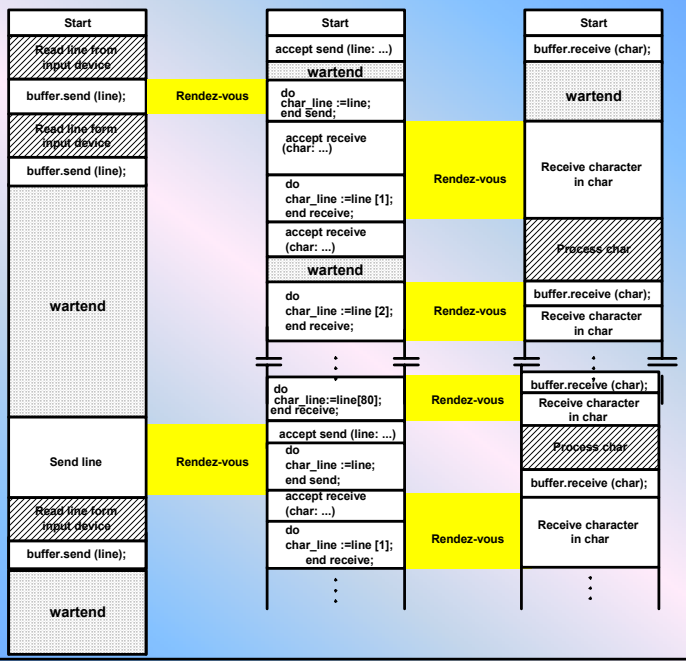
Erzeuger-Verbraucher-Problem (1)

```
task buffer is
      type line_bf is array (1..80) of character;
      entry send (line: in line-bf);
      entry receice (char: out character);
end buffer;
  task body buffer is
      char_line: line_bf;
      position : integer;
begin
      loop
            accept send (line :in line_bf) do
              char_line := line;
            end send;
            for position in 1..80 loop
              accept receive (char: out character) do
                 char := char_line [position];
              end receive;
            end loop;
      end loop;
end buffer;
```


Erzeuger-Verbraucher-Problem (2)

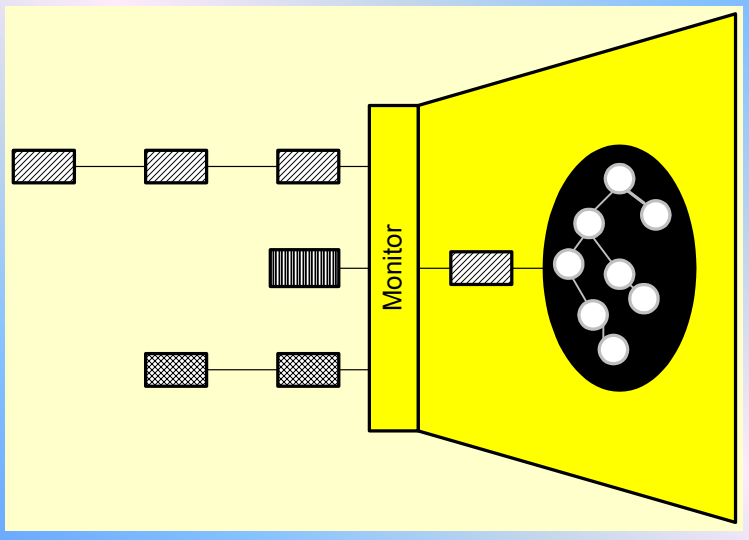
```
task producer;
task body producer is
      use buffer;
      input_line : line_bf>;
begin
      loop
        ... (* Eine Zeile vom Eingabegerät in input_line lesen *)
         buffer.send(inout_line);
      end loop;
end producer;

task consumer;
task body consumer is
      use buffer;
      input_char : character;
begin
      loop
        buffer.receive(input_char);
        ... (* Das eingelesene Zeichen bearbeiten *)
      end loop;
end consumer;
```

Möglicher Ablauf eines Mehrtask-programms



Geschützte Typen

Monitor zur Sequentialisierung von Zugriffen



Beispiel

```
PACKAGE Fifo_Puffer_Verwaltung IS
        PROTECTED TYPE Fifo_Puffer IS
                ENTRY       Deponiere     (N:IN Info_Type);
                ENTRY       Gib_Frei      (N:OUT Info_Type);
        END Fifo_Puffer;
End Fifo_Puffer_Verwaltung
```


6.4 Echtzeitprogrammiersprache Ada 95

Entstehungsgeschichte von Ada

Sprachkonstrukte für die algorithmische Programmierung

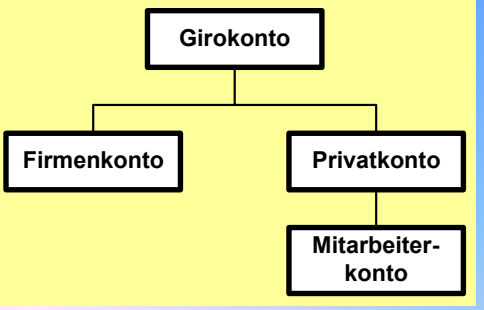
Sprachkonstrukte für die Echtzeitprogrammierung

Spracherweiterungen in Ada 95

Spracherweiterungen in Ada 95

- **Objektorientierte Programmierung**
  - Deklaration neuer Typen, die ähnliche Eigenschaften haben wie der Elterntyp, Klassen, Objekte
  - Vererbung, Modifikation, Hinzufügen von Komponenten der Datenstruktur bzw. der darauf definierten Operationen


```
type Geldtyp is ...;
type Girokonto is tagged
        record
            Inhaber: String( 1...40);
            Nummer. Positive;
            Stand: Geldtyp;
        end record;
```




```
type Privatkonto is new
Girokonto
        with record Dispolimit:
        Geldtyp; end record;
```

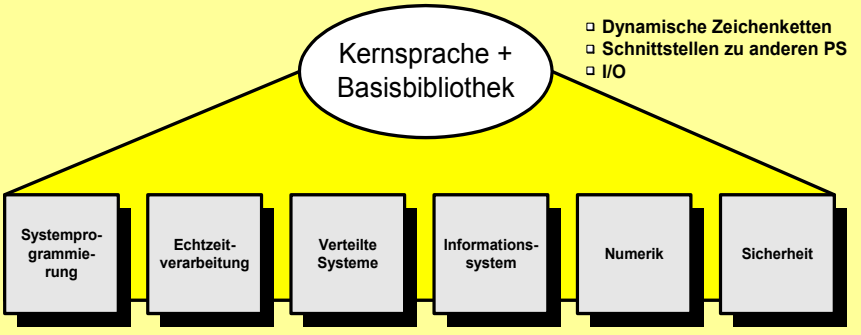
```
type Firmenkonto is new
Girokonto with record
        Branche: String(1..10);
        Eigenkapital: Geldtyp;
        end record;
```

```
type Mitarbeiterkonto is new
Privatkonto with record
        Abteilung: String (1...5);
        Position: String (1... 10);
        end record;
```

Erweiterungen für die Echtzeitverarbeitung

- ❒ Dynamische Prioritäten
- ❒ Beeinflussung der Bearbeitung von Warteschlangen
- ❒ Beeinflussung des Abbruchverhaltens von Tasks
- ❒ Veränderungen des Taskzustandsmodells
- ❒ Festlegung des Zeitbegriffs und Beeinflussung der Zeitgebung

Spracherweiterungen in Ada 95



6.5 Echtzeitprogrammiersprache PEARL

Entstehungsgeschichte von PEARL

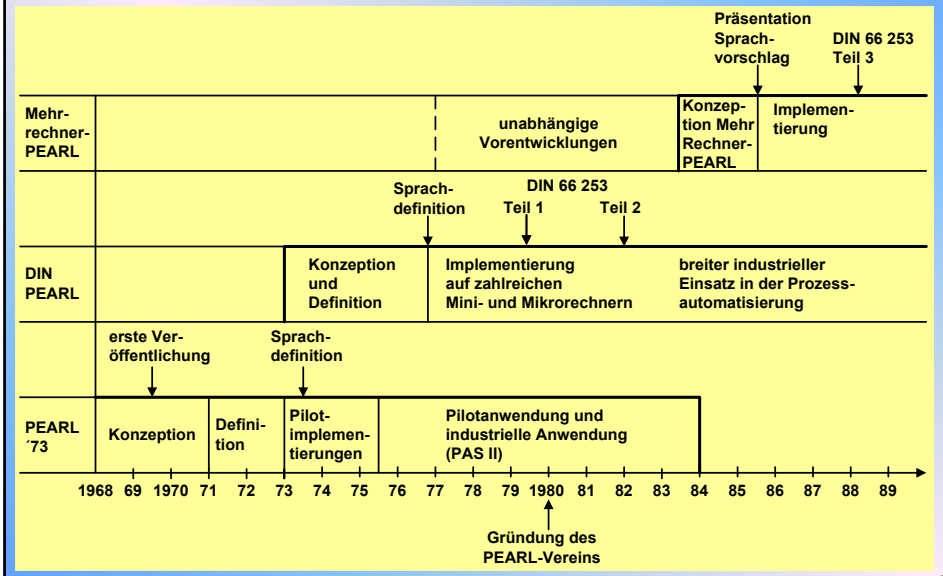
Übersicht über die wichtigsten Spracheigenschaften

Entstehungsgeschichte von PEARL

Process and Experiment Automation Realtime Language

- Echtzeitprogrammiersprache für die in der Anlagen-automatisierung eingesetzten Prozessrechner

Entwicklung der Echtzeit-Programmiersprache PEARL



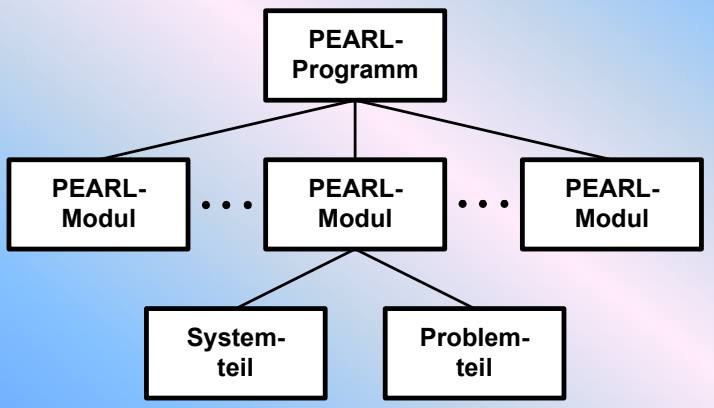
6.5 Echtzeitprogrammiersprache PEARL

Entstehungsgeschichte von PEARL

Übersicht über die wichtigsten Spracheigenschaften

Sprachkonzepte von PEARL

❒ Strukturierung von PEARL-Programmen



❒ Formulierung algorithmischer Zusammenhänge

vergleichbar mit Pascal

❒ Ein- und Ausgabe von Prozess-Signalen einheitliches Konzept für Prozess- und Standardperipherie

Beispiel

TAKE DRUCK FROM DRUCKGEBER;	Eingabe der Prozessdaten DRUCK vom Anschluss DRUCKGEBER
SEND AUF TO SCHIEBER;	Ausgabe der Prozessdaten AUF an den Anschluss SCHIEBER

Echtzeit-Programmierung

Umfassende und leicht verständliche Spracheigenschaften

- zur Vereinbarung von Rechenprozessen (Task)
- zur Steuerung der Übergänge zwischen Taskzuständen
- zur einmaligen oder zyklischen Einplanung von Tasks in Abhängigkeit von Zeitbedingungen oder Unterbrechungs-signalen
- zur Synchronisierung von Tasks

Beispiele

```
AFTER 5 SEC ALL 7 SEC
DURING 106 MIN ACTIVATE
RELAIS PRIORITY 5;
```

Die Task RELAIS wird nach 5 Sekunden während 106 Minuten im 7 Sekunden-Zyklus mit Priorität 5 gestartet (in den Zustand „bereit“ überführt).

```
AT 12:00:00 ALL 1 MIN
UNTIL 12:59:00 ACTIVATE
MESSUNG;
```

Die Task MESSUNG wird von 12 Uhr an jede Minute bis 12:59 Uhr gestartet (in den Zustand „bereit“ überführt)

6.6 Die Programmiersprachen C und C++

Entstehungsgeschichte von C und C++

Sprachkonzepte von C

Sprachkonzepte von C++

Eignung von C und C++ für die Echtzeitprogrammierung

Entstehungsgeschichte von C und C++

1978	Entwicklung des Betriebssystems UNIX durch Dennis Richie in der Programmiersprache C
1986	Erweiterung von C für die objektorientierte Programmierung zur Programmiersprache C++

Entwicklungsziele von C und C++

C:

- Maschinennahe effiziente Systemprogrammiersprache
- flexibel wie Assembler
- Kontrollflussmöglichkeiten höherer Programmiersprachen
- universelle Verwendbarkeit
- begrenzter Sprachumfang

C++:

- Erweiterung von C um Objektorientierung
- Beibehaltung der Effizienz von C
- Verbesserung der Produktivität und Qualität

Hybride Programmiersprache

Concurrent C:

- Erweiterung von C um Konzepte zur Echtzeitverarbeitung

6.6 Die Programmiersprachen C und C++

Sprachkonzepte

C:

- Vier Datentypen: char, int, float, double
- Zusammenfassung von Daten: Vektoren, Strukturen
- Kontrollstrukturen: if, switch, while, do-while, for, continue, break, exit, goto
- Ein-/Ausgabe: Bibliotheksfunktionen
- große Vielfalt an Bit-Manipulationsmöglichkeiten
- schwaches Typkonzept
- getrennte Compilierbarkeit von Sourcefiles
- schwaches Exception-Handling
- keine expliziten Möglichkeiten für Parallelverarbeitung

Programmaufbau

Einfügen bestimmter Bibliotheken bzw. Dateien
Festlegen der Namen für Konstanten und Macros

Deklaration von globalen Variablen;

```
main ( )
{
        Variablen, die innerhalb der Funktion main bekannt sind,
        müssen deklariert werden;
        Anweisungen;
}
```

```
Funktionstype Funktionsname (Liste der Parameter)
{
        Variablen, die innerhalb der Funktion bekannt sind, müssen hier
        deklariert werden;
        verschiedene Anweisungen;
        return (Wert);
}
```

Beispiel

```
#include <stdio.h>                    Standard-Ein-/Ausgabebibliothek

main ( )                              Kennzeichnung des Programm-
{                                     anfangs

        printf („Mein erstes Programm/n“)        Ausgabeanweisung

}
```


6.6 Die Programmiersprachen C und C++

- Entstehungsgeschichte von C und C++
- Sprachkonzepte von C
- Sprachkonzepte von C++
- Eignung von C und C++ für die Echtzeitprogrammierung

Sprachkonzepte

C++:

- Klasse (class)
  - Datenstruktur mit Daten und Methoden (memberfunction)
- Konstruktor (constructor)
  - Anlegen einer Instanz einer Klasse (Objekt)
- Destruktor (destructor)
  - Freigabe von Klassenobjekten
- Überladen von Funktionen (Overloading)
- Datenkapselung (encapsulation)
- Vererbung (inheritance)
- Polymorphismus
  - Auslösung unterschiedlicher Verarbeitungs-schritte durch Botschaften

Struktur eines C++-Programms

Klasse

```
class      W_maschine
{
```

Member

Daten

```
int        Drezahl1, Drehzahl2;
float      Leistung;
```

Methoden

```
void       rechne(Drehzahl1, Drehzahl2);
void       schreibe(float);
};
```

Objekt

```
W_maschine      Trommel;
```

6.6 Die Programmiersprachen C und C++

Entstehungsgeschichte von C und C++

Sprachkonzepte von C

Sprachkonzepte von C++

Eignung von C und C++ für die Echtzeitprogrammierung

Eignung von C und C++ für die Echtzeitsystemen (1)

⇨ C und C++ enthalten keine Echtzeit-Sprachmittel

⇨ Einsatz von Echtzeit-Betriebssystemen zur Realisierung von Echtzeitsystemen

Aufruf von Betriebssystemfunktionen im C-Programm

⇨ Bereitstellung von Bibliotheken

Eignung von C und C++ für die Echtzeitsystemen (2)

Programmiersprachen C und C++

↑ Am häufigsten verwendete Programmiersprache für Echtzeit-Anwendungen

- große Anzahl und Vielfalt an Unterstützungs-werkzeugen
- gut ausgebaute Programmierumgebungen
- für die meisten Mikroprozessoren sind Compiler verfügbar
- Anschluss an Echtzeitbetriebssysteme wie QNX, OS9, RTS, VxWorks

⇨ **Vorsicht bei objektorientierten Sprachmitteln**
- **nicht-deterministisches Laufzeitverhalten**
- **schlechte Speicherplatzausnutzung**

6.7 Die Programmierumgebung Java

Entstehungsgeschichte von Java

Sprachkonzepte von Java

Sprachkonstrukte von Java

Eignung von Java für die Entwicklung von Echtzeitsystemen

Entstehungsgeschichte von Java

1990 Konzept der Programmiersprache Java durch die Fa. Sun (James Gosling, Bill Joy)

Ziel: *Programmiersprache für die Unterhaltungs-elektronik (interaktives Fernsehen)*

Namensgebung nach der Kaffeesorte Java

1995 Neuorientierung der Entwicklungsrichtung zu einer Sprache, um Programme im World Wide Web zu übertragen und auszuführen

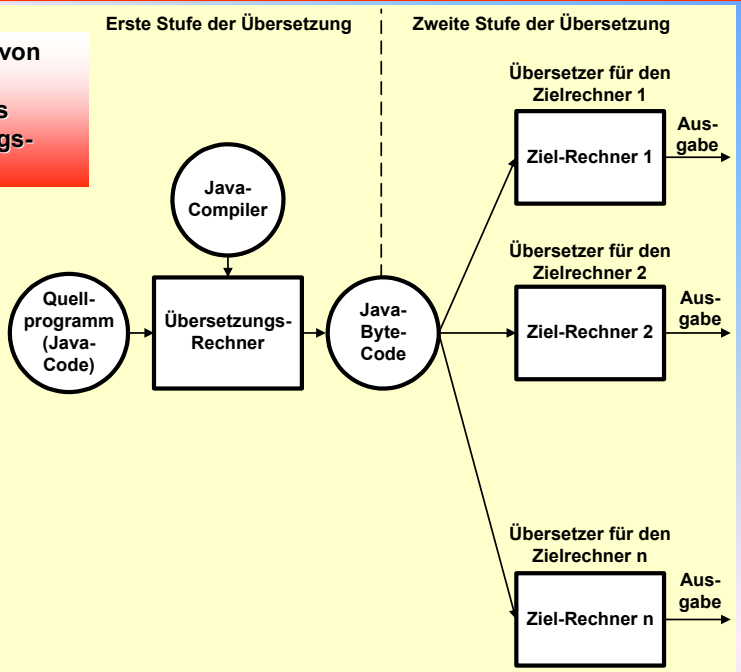
Frei verfügbar für nicht-kommerzielle Zwecke

6.7 Die Programmierumgebung Java

Sprachkonzepte von Java

- Objektorientierte Konzepte
- Interpretierung des Codes
  - schneller Entwicklungszyklus
  - schlechtes Laufzeitverhalten und hoher Speicherplatzbedarf
  - höhere Portabilität
- Bereitstellung eines Speichermanagers
- Verzicht auf herkömmliche Zeigertechnik
- Strikte Typprüfung zur Kompilier- und Laufzeit
- Leichtgewichtsprozesse
- GUI-Klassenbibliothek

Portabilität von Java durch zweistufiges Übersetzungs-verfahren



6.7 Die Programmierumgebung Java

Entstehungsgeschichte von Java

Sprachkonzepte von Java

Sprachkonstrukte von Java

Eignung von Java für die Entwicklung von Echtzeitsystemen

Sprachkonstrukte von Java

- Numerische Datentypen
  - byte : 8 Bit
  - short : 16 Bit
  - int : 32 Bit
  - long : 64 Bit
  - float : 32 Bit
  - double : 64 Bit
- Zeichendatentypen
  - char : 16 Bit
- Logischer Datentyp
  - boolean

Unterschiede zu C++

- Keine Preprozessoranweisungen wie #define oder #include
- keine typedef-Klauseln
- Structures und Unions in Form von Klassen
- keine Funktionen
- keine Mehrfachvererbung
- kein goto
- kein Überladen von Operatoren
- umfangreiche Klassenbibliothek
  - Basisklassen (Object, Float, Integer)
  - GUI-Klassen
  - Klassen für Ein-und Ausgabe
  - Klassen für Netzwerkunterstützung

6.7 Die Programmierumgebung Java

Entstehungsgeschichte von Java

Sprachkonzepte von Java

Sprachkonstrukte von Java

Eignung von Java für die Entwicklung von Echtzeitsystemen

Eignung von Java für die Entwicklung von Echtzeitsystemen

Anwendungsfelder

- ⇨ rapid prototyping im Client/Server-Bereich
- ⇨ multimediale Darstellungen (Video, Sound, Animation)
- ⇨ Intranetapplikationen
- ⇨ Echtzeitanwendungen

- ⇨ **Speicherverwaltung (garbage collection)**
- ⇨ **hoher Speicherbedarf**
- ⇨ **schlechtes Laufzeitverhalten**

Echtzeitsprachmittel in Java

- ❒ Eingabe und Ausgabe von Prozesswerten
  - ❋ vergleichbar mit C/ C++
- ❒ Parallelität
  - ❋ keine Prozessunterstützung
  - ❋ Leichtgewichtsprozesse
  - ❋ Zeitscheibenverfahren
- ❒ Synchronisierung
  - ❋ Monitore
  - ❋ Semaphorvariablen
- ❒ Interprozesskommunikation
  - ❋ nur für Leichtgewichtsprozesse über gemeinsame Daten
- ❒ Bitoperationen
  - ❋ vergleichbar mit C/ C++

Java-Beispiele

Java Applikation

```
class HelloWorldApplication
{
        public static void main(String argv[ ])
        {
                System.out.println(„Hello World!“);
        }
}
```

Java Applet

```
import java.applet.*;
import java.awt.Graphics;
public class HelloWorldApplet extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString(„Hello World!“),5,25);
        }
}
```