Programmentwicklung

Ziele:

1. effizient (wenig Rechenzeit und Kernspeicher)
2. lesbar (Kommentare, sinnvolle Namen)
3. transportabel (Standard–FORTRAN, rechnerunabhängig)
4. allgemein (modularer Aufbau, Unterprogramme universell einsetzbar)

FORTRAN77 (F77)

Das Programm besteht aus Segmenten (Modulen).
Der Compiler behandelt jedes Segment getrennt.
Der Loader verbindet die Segmente (und evtl. benutzte Bibliotheksroutinen) zu einem lauffähigen Programm.

Segmente:

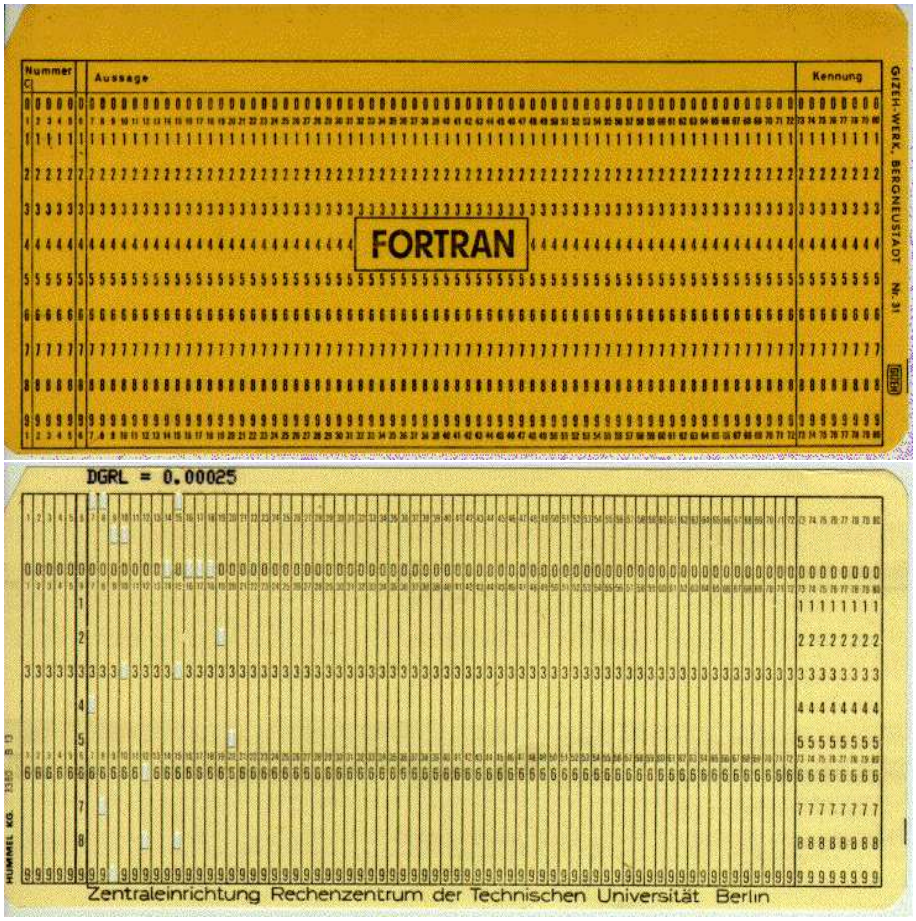
- Hauptprogramm (PROGRAM)
- Unterprogramme (SUBROUTINE)
- Funktionen (FUNCTION)
- COMMON–Blöcke (COMMON / /)
- [BLOCK DATA – Bereiche](BLOCK DATA)

Jedes Segment besteht aus:

- Deklarationsteil (nichtausführbare *statements*)
- Rumpf (ausführbare *statements*)
- END (außer bei COMMON)

FORTRAN–statements

orientiert an Lochkarte:



- Spalte 1–5: Label
  `C` oder `*` in Spalte 1 bedeutet Kommentarzeile
- Spalte 6: Fortsetzungskarte (max. 19)
- Spalte 7–72: eigentliche statements
- Spalte 73–80: Kennung (wird nicht vom Compiler bearbeitet)

FORTRAN – Zeichenvorrat

- **Groß**buchstaben: `A-Z`
- Ziffern: 0 . . . 9
- ␣ (Leerstelle, *blank*)
- Sonderzeichen: `+ - * / ( ) , . $ ' : =`

Andere Zeichen dürfen in Standard–FORTRAN–statements nicht vorkommen (mit der Ausnahme von *strings*, z.B. für Filenamen)!

Namenskonvention für Segmentnamen

- maximal 6 Zeichen
- keine Sonderzeichen
- erstes Zeichen = Buchstabe

Die meisten Compiler sind nicht mehr auf diese restriktiven Normen fixiert. ⇒ Im Zweifelsfall kann man daher der Übersichtlichkeit den Vorzug geben.

Variablen und Konstanten

- Variable bzw. Konstanten werden durch symbolische Namen angesprochen
  Richtig: `XYZ`, `ALPHA1`, `ZEHN`
  Falsch: `0815`, `A$`, `HUNDERT`,
- Variablennamen sind symbolische Namen für Speicherplätze; sie werden vom Compiler freigehalten und erst vom Loader oder bei Programmausführung belegt.
- Speicherplätze für Konstanten werden bereits beim Compilieren belegt (und sind nicht änderbar!).
- **Typen von Variablen bzw. Konstanten**
  - ganze Zahlen: `INTEGER` (2)
  - reelle Zahlen: `REAL` (2.0, 2., 3.E+17)
    (Punkt beachten!)
    `DOUBLE PRECISION` (2.D0) (manchmal auch: `REAL*4` oder `REAL*8`)
  - komplexe Zahlen: `COMPLEX` ((1.0,2.0) = 1+2i)
  - Text (*strings*): `CHARACTER` ('guten Tag')
    (`CHARACTER*40` bedeutet ein String mit 40 Zeichen)
  - logische Variable: `LOGICAL` (`.TRUE.`,`.FALSE.`)

- Implizite Typ–Vereinbarung: alle Variablen bzw. Konstanten, deren Name mit `I,J,K,L,M,N` beginnt, sind vom Typ INTEGER, alle anderen vom Typ REAL.
- Die implizite Typ–Vereinbarung sollte stets ausgeschaltet werden mit `IMPLICIT NONE` (nicht Standard!)
- Variablen bzw. Konstanten sollten explizit deklariert werden:

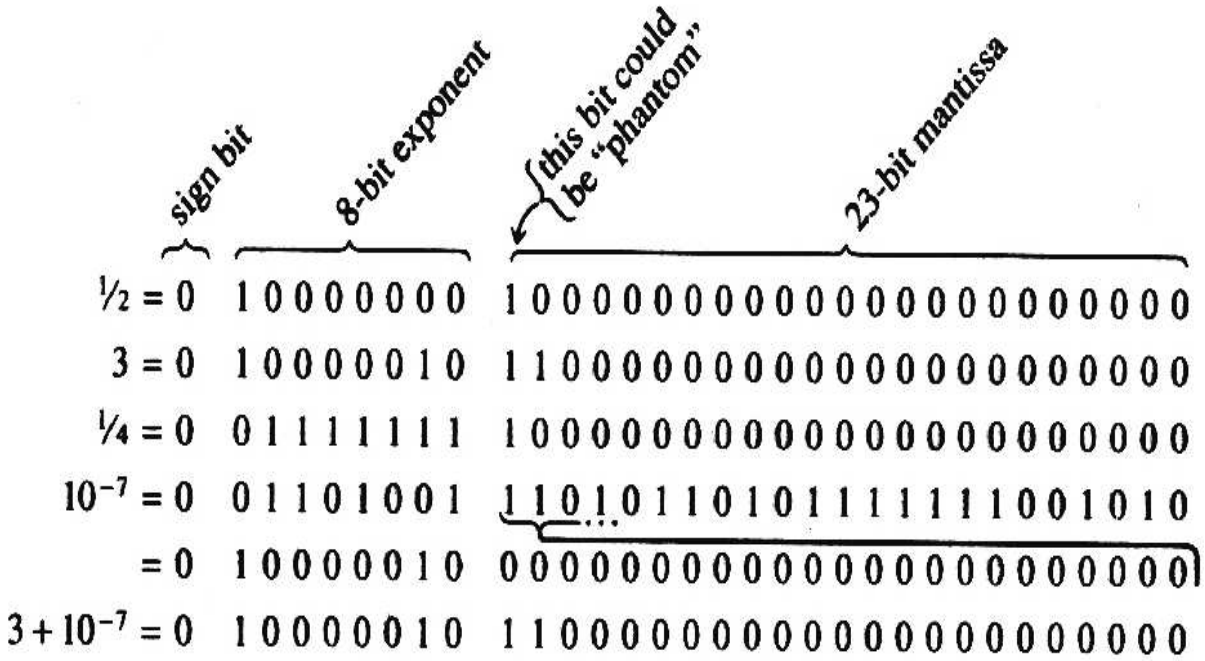
```
␣␣␣␣␣␣PROGRAM␣PROG1
␣␣␣␣␣␣INTEGER␣C
␣␣␣␣␣␣REAL␣IX,X,Y,Z
␣␣␣␣␣␣CHARACTER*10␣NAME,VORNAM*5
␣␣␣␣␣␣LOGICAL␣A
␣␣␣␣␣␣DOUBLE␣PRECISION␣B

              ⋮


␣␣␣␣␣␣END
```

Zuweisung von Werten an Variable bzw. Konstanten

1. Speicherinhalt einer Variablen wird während der Ausführung des Programms belegt bzw. verändert:
   - explizite Zuweisung: `X = 0.`
   - I/O–statement: `READ(3,*) X`
   - (Einlesen von Tastatur (*stdin*): `READ *,X`)
   - Übergabeargument in Unterprogrammaufruf
   - *internal file*
   - Änderung des Speicherplatzinhalts über COMMON–Block–Zugriff oder EQUIVALENCE–Anweisung
   - Änderung des Speicherplatzinhalts durch freien Zugriff auf (beliebige) Array–Elemente
2. Loader besetzt über die nicht–ausführbare Anweisung `DATA` die Speicherplätze einer Variablen vor:
   ```
   DATA X,Y,Z /3*0./
   ```
   (im Deklarationsteil)
3. Compiler legt Wert einer Konstanten fest:
   ```
   PARAMETER (PI=3.1415D0)
   ```
   (im Deklarationsteil)

Zahlendarstellung

1. Ganze Zahlen (INTEGER): exakte Arithmetik (sofern im darstellbaren Zahlenbereich)
2. Gleitkommazahlen (REAL, DOUBLE PRECISION): endliche Darstellungsgenauigkeit $\Rightarrow$ numerischer Fehler !!



- Zahlendarstellung: $s \times M \times B^{e-E}$ (s: Vorzeichen, M: Mantisse, B: Basis (typ. 2 oder 16), e: Exponent, E: *bias* (maschinenabhängig))
- *normalized number*: „linkes" Bit in Mantisse $= 1$

Maschinengenauigkeit und Fehler

- Maschinengenauigkeit ϵ_m: Die kleinste Gleitkommazahl, die zu 1 addiert eine – in Maschinendarstellung – von 1 abweichende Zahl ergibt (REAL (4 Byte): typ. $3 \cdot 10^{-8}$).
- Rundungsfehler durch endlich genaue Zahlendarstellung
- Rundungsfehler nach N Operationen: $\sim \sqrt{N}\epsilon_m$ (falls die einzelnen Fehler zufällig verteilt sind!)
- kann erheblich größer werden, etwa bei Subtraktion ähnlich großer Zahlen; durchaus häufiges Auftreten solcher Probleme:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

- weiterer Fehler: Abbruchfehler (hervorgerufen durch diskrete Näherung vieler numerischer Größen, etwa endliche Summation etc.)
- Abbruchfehler können – im Prinzip – vom Programmierer umgangen bzw. minimiert werden, Rundungsfehler sind dagegen nicht vermeidbar und lediglich durch geeignete Methoden bzw. Typdeklarationen zu reduzieren.

Ein- und Ausgabe

- Lesen: `READ(K,L)`
- Schreiben: `WRITE(K,L)`

wobei
K = Kanalnummer
L = Label

Kanalnummer:

- $0 \leq K \leq 99$
- vordefinierte Kanalnummern (nicht einheitlich):

  5 : Standardeingabe (Tastatur)

  6 : Standardausgabe (Bildschirm)

  7 : Stanzausgabe
- Kanalnummern ≥ 10 sind "sicher".
- Ausgabefile: z.B. fort.10
- Kanal"nummer" $*$ steht für Standardein-/ausgabe

Label:

- L=*: formatfreies Lesen bzw. Schreiben
  Beispiel:

  ```
  ␣␣␣␣␣␣WRITE(6,*)␣'guten␣Tag'
  ```

- L=Nummer eines FORMAT –Statements
  Beispiel:

  ```
  ␣␣␣␣␣␣WRITE(6,100)␣'guten␣Tag'
  100␣␣␣FORMAT(A)
  ```

- Labels befinden sich in den ersten 5 Spalten einer Zeile.

- L=Formatangabe
  Beispiel:

  ```
  ␣␣␣␣␣␣WRITE(6,'(A)')␣'guten␣Tag'
  ```

Das FORMAT–Statement

`FORMAT` ist eine nicht–ausführbare Anweisung, die an beliebiger Stelle eines Moduls auftreten darf. Sie muß ein Label haben.

```
<Labelnummer> FORMAT(Spezifikatorenliste)
```

Spezifikatoren

- `X`: Leerstelle
- `In`: Integer der Länge n
- `Fn.m`: Festkommazahl mit n Stellen, wobei m Nachkommastellen vorhanden sind
- `En.m`: Exponentialdarstellung
- `nP`: Skalenfaktor (multipliziert Mantisse mit 10^n, während der Exponent um n verringert wird. Vorsicht bei F-FORMAT)!
- `Gn.m`: F bzw. E
- `An`: Text der Länge n
- `A`: beliebiger Text
- `/` erzeugt neue Zeile.
- Wiederholungsfaktoren möglich, z.B. `5I7` oder `5(2X,I7)`

- Text darf in FORMAT–Statenment eingebunden werden, z.B.
  `FORMAT(1X,' x = ',F10.3)`

Beispiele:

1. Nicht–formatiertes Schreiben in file `fort.10`

   ```
   ␣␣␣␣␣␣WRITE(10,*)␣X,Y,Z
   ```

2. Formatiertes Schreiben auf Standard–Output:

   ```
   ␣␣␣␣␣␣WRITE(6,111)␣X,Y,Z
   111␣␣␣FORMAT(1X,'x=␣',F10.2,E10.2,/,F10.2)
   ```

3. Lesen von fort.11

   ```
   ␣␣␣␣␣␣READ(11,*)␣X,Y
   ```

Arithmetische Ausdrücke

- geschrieben wie mathematische Formel:
  `X=(Y+Z)/3.`
- Zuweisung erfolgt von rechts nach links:
  `I=I+1`
- weitere Operationen:
  - Potenzieren: `**`, z.B. `A**2.5`
  - Wurzel: `SQRT(X)`
  - Exponentialfunktion: `EXP(X)`
- nur Variablen vom selben Typ verwenden

Logische (boolesche) Ausdrücke

- Operatoren: `.AND.` `.OR.` `.NOT.`
- Zuweisung wie bei arithmetischem Ausdruck:

```
LOGICAL A,B,C
   ⋮
A = B .AND. C
```

- Vergleichsoperatoren: `.LT.` `.LE.` `.EQ.` `.NE.` `.GT.` `.GE.`

```
REAL B,C
LOGICAL A
   ⋮
A = B .LT. C
```

- funktioniert auch mit CHARACTER–Variablen (Vergleichskriterium: Alphabet):

```
CHARACTER*10 NAME
LOGICAL A
   ⋮
B = NAME .LT. 'D'
```

  Alternative für CHARACTER-Variablen: `LLT, LLE, LGT, LGE`

```
B = LLT (NAME,'D')
```