Modula-3
Programmierkonventionen

Die wichtigsten Bestandteile der Programmierkonventionen für Modula-3 werden hier erläutert. Dies sind:

- Schreibweise von Bezeichnern
- Verwendung von Leerzeichen
- Einrückkonventionen
- Kommentare

Einleitung

- **Spielraum** bei der Gestaltung der **Details**
- **Richtlinien** sollen dazu beitragen, daß die **Programme stets leicht zu verstehen** sind.
- **Lesbarkeit** und **Wartbarkeit** sind wichtig.
- Es gibt zwar keinen "korrekten" Programmierstil, aber die folgenden Konventionen haben sich erfolgreich durchgesetzt.

Bezeichner - 1

- **Bezeichner** müssen (bis auf Laufvariablen in Schleifen, etc...) **aussagekräftig** sein**.**
  - Negativbeispiele:

```
MODULE A ...
PROCEDURE B ...
```

```
MODULE Gaius ...
PROCEDURE Julius ...
```


Bezeichner - 2

- Bei **Bezeichnern**, die **aus mehreren Worten** zusammengesetzt werden, muß der erste Buchstabe eines Wortes jeweils groß geschrieben oder durch "_" getrennt werden.
  - Beispiele:
    roemischeZahl, gebeTextEin, gebe_Text_ein
  - Negativbeispiel:
    diesisteinschlechterbezeichnerfuereinezahl

Bezeichner - 3

- **Bezeichner für folgende Komponenten müssen mit einem Großbuchstaben beginnen:**
  - Modulnamen
  - Prozeduren und Funktionen
  - Typen
  - Konstanten
  - Elemente von Aufzählungstypen
- **Bezeichner für alle anderen Komponenten müssen mit einem Kleinbuchstaben beginnen. Dies sind:**
  - Variablen
  - Formale Parameter
  - Record-Komponenten
  - Alias-Namen in WITH-Anweisung

Bezeichner - 4

- Bezeichner, die aus mehr als einem Buchstaben bestehen, sollten nicht nur aus Großbuchstaben bestehen. Dieses kann zu **Konflikten mit von Modula-3 als Bezeichner belegten Wörtern** führen.
- Derartige Wörter können jedoch benutzt werden, wenn Groß-/Kleinschreibung verändert wird.
  - Beispiel: begin, real dürfen als Variablennamen benutzt werden.
  - Negativbeispiel: BEGIN, REAL dürfen nicht als Variablennamen benutzt werden.

Verwendung von Leerzeichen

- **Leerzeichen stets vor und nach einem Operator**.
  - Beispiel: x := (a + b) * c + F(x);
  - Negativbeispiel: x:=(a+b)*c+F(x);

- **Ein Leerzeichen steht immer nach:**
  - geschlossener Klammer)
  - Doppelpunkt :
  - Semikolon **;**
    - Beispiel:

```
PROCEDURE P(x, y: REAL; int: INTEGER) =
```

    - Negativbeispiel:

```
PROCEDURE P(x,y:REAL;int:INTEGER)=
```


Einrückung - allgemein

- Jede **Ebene** wird **um 2 Leerzeichen eingerückt** (kein Tabulator!!!).

- Die Festlegung der Ebenen ist durch die **Einrückkonventionen** gegeben.

- Durch die Einhaltung der Einrückkonventionen wird die **Struktur** der Programme leichter erfaßbar.

Einrücken bei Modulen und Prozeduren

```
MODULE Test EXPORTS Test;
<Importe>
<Deklarationen>
BEGIN
 <Anweisungen>
END Test.
```

Die Anweisungen zwischen den Bezeichnern BEGIN und END sind eine Ebene tiefer als diese.

```
PROCEDURE P(x, y: REAL; int: INTEGER) =
 <Deklarationen>
 BEGIN
  <Anweisungen>
 END P;
```

Die Definition der Prozedur ist eine Ebene tiefer.

Nach den Deklarationen beginnt keine neue Ebene.

Einrücken bei Interface und Variablendeklarationen

```
INTERFACE Test;
<Importe>
<Deklarationen>
END Test.
```

Hier findet kein Ebenenwechsel statt.

```
VAR
 x, y, z: Typ := <Ausdruck>
 a: [1..10];
```

Die Deklaration der Variablen ist eine Ebene tiefer als der Bezeichner VAR.

Einrücken bei Schleifen

```
FOR i := a TO b DO
  <Anweisungen>
END;

REPEAT
  <Anweisungen>
UNTIL <Ausdruck>;

WHILE <Ausdruck> DO
  <Anweisungen>
END;
```

Die Anweisungen der Schleife sind eine Ebene tiefer als Schleifenkopf und -ende anzuordnen.

Einrücken bei if-Ausdrücken

```
IF <Ausdruck> THEN
  <Anweisungen>
ELSEIF <Ausdruck> THEN
  <Anweisungen>
ELSE
  <Anweisungen>
END;
```

Die Anweisungen sind eine Ebene tiefer als Bedingungsteile der if-Anweisung.

Kommentare - 1

- **(* steht zu Beginn eines Kommentars.**
- ***) steht am Ende eines Kommentars.**
- **Ein Leerzeichen steht nach (* und vor *).**
  - Beispiel: (* ein kurzer Kommentar in einer Zeile *)
- Beim **Auskommentieren von Quelltext** (d.h. der betreffende Quelltext soll nicht beachtet werden) sollte folgende Formatierung verwendet werden:

```
(* Die folgende(n) Codezeile(n) werden
auskommentiert.
          IF (x < a) THEN y := b;
*)
```


Kommentare - 2

- Für jeden Modul soll zu Beginn kurz die **Funktionalität** beschrieben werden.
- Zusätzlich soll der Kopfkommentar die Informationen über den **Autor**, die **Entwicklungsumgebung**, sowie das **Datum** der **Erstellung** und der **letzten Änderung** enhalten.
- Daher ist für jeden Modul ein Kopfkommentar der folgenden Form zu erstellen:

Kommentare - 3

```
MODULE Test EXPORTS Main;
(* Dieses Programm zeigt einen Willkommensgruss
    Autor : Horst Lichter, RWTH Aachen
    Umgebung : SRC-Modula-3 rel. 3.6, Windows NT 4.0
    Erstellt : 16.08.98
    Letzte Aenderung: 20.08.98
*)

IMPORT <Importe>
[...]
BEGIN
 <Anweisungen>
END Test.
```


Kommentare - 4

- Nur **sinnvolle Kommentare** sollten eingefügt werden.
- Kommentare sollten **keine redundante Information** enthalten.
  - Negativbeispiel:

```
(* weise c die Summe von a und b zu *)
c := a + b
```

- Kommentar **vor dem Quelltext**, der kommentiert wird.
- Geht aus den Bezeichnern nicht die Aufgabe der bezeichneten Variablen bzw. Parameter hervor, so sollte dieser im Prozedurkopf erläutert werden

Kommentare - 5

- Kommentare dienen dem Verständnis (ebenso wie der übrige Teil der Programmierrichtlinien) des Programmes. Entsprechend **verständlich** sollten die Kommentare verfaßt sein.
  - Beispiel:

```
(* wenn der Rechner nicht benutzt werden kann,
dann soll <Anweisung> ausgeführt werden
*)
IF (Stromausfall OR Stecker_draussen OR
Rechner_kaputt OR ... ) THEN
  <Anweisung>
```


Zusammenfassung - 1

- **Programme müssen gut verständlich sein**
  - "Wir programmieren nicht für uns, sondern für andere"
  - Lesbarkeit und Wartbarkeit
  - Programmierkonventionen müssen daher beachtet werden
  - dies gilt auch für die Vorlesung "Programmierung"
- **Bezeichnerwahl ist sehr wichtig**
  - Programme leichter verständlich
  - geringere Fehlerhäufigkeit
- **Leerzeichen**
  - helfen bei der Erfassung der Einheiten eines Ausdrurck

Zusammenfassung - 2

- **Einrückungen**
  - helfen, die Übersicht zu behalten
  - die Struktur des Programmes schneller zu erfassen
- **Aussagekräftige Kommentare**
  - helfen, schwierige Stellen zu verstehen
  - sind sehr wichtig für die Wartung