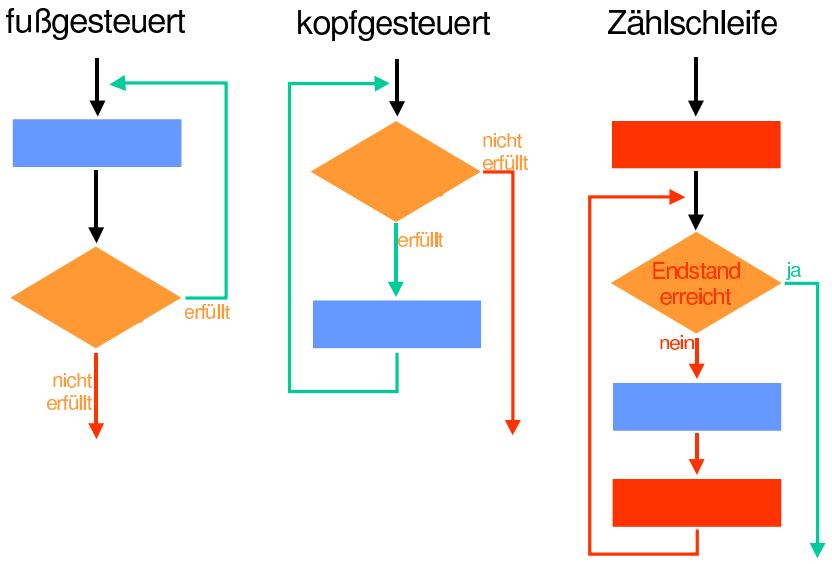
Wiederholungsanweisungen



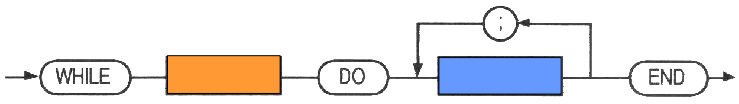
Wiederholungsanweisungen

➢ WHILE-Schleife

- Struktur:

```
WHILE Bedingung DO
  (*Anweisungsfolge*)
END
```

- Syntax:

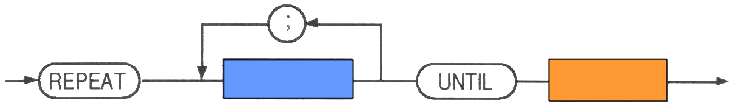


Wiederholungsanweisungen

- REPEAT-Schleife
  - Struktur:

```
REPEAT
⋮ (*Anweisungsfolge*)
UNTIL Bedingung
```

  - Syntax:

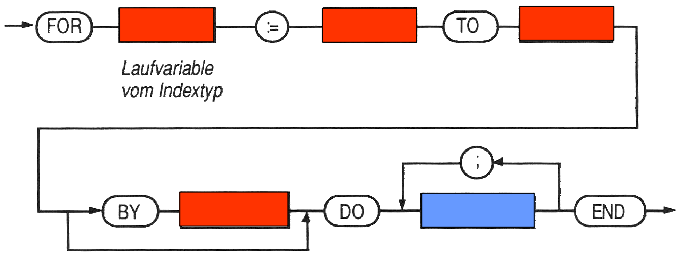


Wiederholungsanweisungen

- FOR-Schleife
  - Struktur:

```
FOR index:=ausdruck1 TO ausdruck2 BY KonstAusdruck DO
⋮ (*Anweisungsfolge*)
END
```

  - Syntax:



Wiederholungsanweisungen

- FOR-Schleife (cont'd)
  - Hinweise:
    - Die Anweisungsfolge wird mit fest vorgegebener Wertefolge durchlaufen
    - index = Zählvariable (Laufvariable); Ausdruck1 = Startwert von index
    - Ausdruck2 = Endwert der Iteration
    - KonstAusdruck = Schrittweite der Laufvariable (= 1 falls nichts angegeben)
    - Ausdruck1, Ausdruck2, KonstAusdruck dürfen in der Anweisungsfolge **nicht** verändert werden (ggf. WHILE bzw. REPEAT verwenden)
    - index ist nach Abarbeitung der FOR-Anweisung undefiniert!

Beispiel „FOR“: Normalverteilungswerte

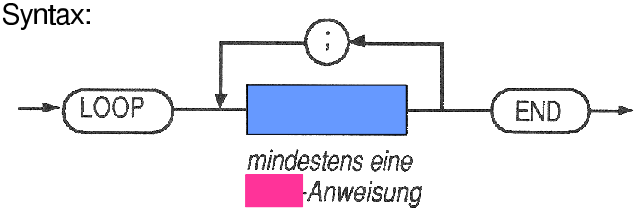
```
MODULE Normalverteilung;
  FROM RealInOut      IMPORT ReadReal, WriteFloat;
  FROM InOut          IMPORT WriteString, WriteLn, WriteInt;
  FROM MathLib        IMPORT exp, sqrt;

CONST
  pi  = 3.14159265358979323846264338 3;
  Max = 10;

VAR
  i                                : INTEGER;
  std scale, faktor, GaussWert     : REAL;

BEGIN
  WriteString("Berechnung der Normalverteilung ...);
  WriteLn;
  WriteString("Eingabe der Standardabweichung: ");
  ReadReal(std); WriteLn;

  scale := 1.0 / (sqrt(2.0 * pi) * std);

  FOR i := -max TO max DO
    faktor := -(FLOAT(i) * FLOAT(i)) / (2.0 * std);
    GaussWert  := scale * exp(faktor);

    WriteInt(i, 5); WriteString(" : ");
    WriteFloat(GaussWert, 4, 8); WriteLn
  END
END Normalverteilung.
```

Wiederholungsanweisungen

- Loop-Anweisung
  - Struktur:

```
LOOP
: (*Anweisungsfolge mit EXIT(s)*)
END (*LOOP*)
```

  - Syntax:



  - Verwendung:
    „gewollte Endlosschleife“ mit Ausstieg(en) an beliebiger(/n) Stelle(n) innerhalb der Schleife

Beispiel „LOOP“: Werbeseiten (aus Puchau et al.)

Der Computergroßhandel Peek & Poke möchte auf einer Messe zu Werbezwecken einen Computer laufen lassen. Dieser soll im 30-Sekunden-Takt zwei Werbeseiten anzeigen und jederzeit auf Tastendruck das Werbeprogramm verlassen können, damit er für andere Zwecke zur Verfügung steht.

Lösungsidee :

```
Schleife                                                    LOOP
    zeige erste Seite
    falls Eingabe innerhalb von 30 Sekunden,
        dann verlasse Schleife                              
    zeige zweite Seite
    falls Eingabe innerhalb von 30 Sekunden,
        dann verlasse Schleife                              
Schleifenende                                               END
```

Beispiel „LOOP“: Werbeseiten (aus Puchau et al.)

```
MODULE Werbung;

FROM STextIO              IMPORT Write, WriteString, WriteLn;
FROM NochzuSchreiben      IMPORT eingabein30s,(*geeignete Warteroutine*)
                                 CLS;         (*löscht den Bildschirm *)
BEGIN
  LOOP
    (* Ausgabe 1. Bildschirmseite mit Werbung *)
    CLS();  (* Bildschirm löschen *)
    WriteString("Schauen Sie sich diesen Rechner an.");
    WriteLn;
    WriteString("Es gibt nichts, was der nicht kann.");
    WriteLn;
    IF eingabein30s() THEN
      EXIT
    END(*IF*);

    (* Ausgabe z. Bildschirmseite mit Werbung *)
    CLS () ;
    WriteString("Drum sollten Sie ihn rasch erwerben,");
    WriteLn;
    WriteString("Dann freuen sich auch Ihre Erben!!");
    IF eingabein30s() THEN
      EXIT
    END(* IF *);
  END(* LOOP *)
END Werbung.
```

Konvertierung: REPEAT ↔ WHILE

- ... im Prinzip durch Negierung der Schleifenbedingung
  - beachte bei logisch (mit und / oder) zusammengesetzten Bedingungen die de Morgan‘schen Regeln!
    - Beispiel:
      Bedingung: q OR (i = n)
      negierte Bedingung: (NOT q) AND (NOT(i=n))
      also: (NOT q) AND (i<>n)
- Beispiel: WHILE → REPEAT (nicht zu empfehlen)

```
WHILE bedingung DO anweisung END
```

konvertiert in:

```
IF bedingung THEN
       REPEAT anweisung UNTIL NOT bedingung
END
```

Konvertierung: REPEAT ↔ WHILE

➢ Beispiel: REPEAT → WHILE

- mit Code-Verdopplung

```
REPEAT anweisung UNTIL abbruchbedingung
```

konvertiert in:

```
anweisung;
WHILE NOT abbruchbedingung DO anweisung END
```

- mit Schaltervariable **schalter**

```
REPEAT anweisung UNTIL abbruchbedingung
```

konvertiert in:

```
schalter := TRUE;
WHILE schalter DO
anweisung;
schalter = NOT abbruchbedingung
END
```

Programmbeispiele

➢ größter gemeinsamer Teiler (ggT)

- Der ggT(a,b) zweier ganzer Zahlen a und b ist die größte ganze Zahl, durch die sowohl a als auch b teilbar ist.
- Beispiel: ggT(16,36) = 4
- Anwendung: optimales Kürzen
- Es gilt:
  ggT(a,a) = a falls b = a
  ggT(a,b) = ggT(a-b,b) falls b < a
  ggT(a,b) = ggT(a,b-a) falls b > a
  → iteratives Programm zur Berechnung des ggT
- effektiver: Euklid'scher Algorithmus für $a \geq b$:
  ggT(a,b) = b falls a MOD b = 0
  ggT(a,b) = ggT(b, a MOD b) sonst

ggT (iterative Berechnung)

```
MODULE ggT;
(* Berechnung des groessten gemeinsamen Teilers
 * zweier positiver ganzer Zahlen *)

  FROM InOut IMPORT WriteString, WriteLn, WriteCard, ReadCard

  VAR  x, y : CARDINAL;

BEGIN
  WriteString("Bestimmung des GGT"); WriteLn;
  WriteString("==================="); WriteLn;
  WriteString("Erste positive Zahl eingeben: ");
  ReadCard(x); WriteLn;
  WriteString("Zweite positive Zahl eingeben: ");
  ReadCard(y); WriteLn;

  IF (x > 0) AND (y > 0) THEN
    WHILE (x <> y) DO
      IF (x > y ) THEN x := x - y
      ELSE (* y < x *) y := y - x
      END (* IF *)
    END; (* WHILE *)
    WriteString("GGT = ");
    WriteCard(x,0)
  ELSE
    WriteString("fehlerhafte Eingabe!")
  END; (* IF *)
  END ggT.
```

ggT

Wordcount

```
MODULE wcount;
(* counts chars, words and lines from stdin *)

 FROM InOut IMPORT Read, Done, WriteString, WriteLn, WriteCard;

 CONST
  newline  = 12C;
  blank    = 40C;
  tab      = 11C;

 VAR
  chars, words, lines   : CARDINAL;
  ch                    : CHAR;
  inword                : BOOLEAN;

BEGIN
    chars  := 0;
    words  := 0;
    lines  := 0;
    inword := FALSE;
  Read(ch);
  WHILE Done DO
    chars := chars + 1;

    IF inword THEN
      CASE ch OF
      | blank, tab, newline : inword := FALSE;
                              words  := words + 1
      ELSE (* nichts *)
      END (* CASE *)
    ELSE
      CASE ch OF
      | blank, tab, newline : (* nichts *)
      ELSE                    inword := TRUE
      END (* CASE *)
    END; (* IF *)

    IF ch = newline THEN
      lines := lines + 1
    END; (* IF *)

    Read(ch)

  END; (* WHILE *)

  WriteCard(chars,14); WriteString(" character(s)"); WriteLn;
  WriteCard(words,14); WriteString(" word(s)"); WriteLn;
  WriteCard(lines,14); WriteString(" line(s)"); WriteLn
END wcount.
```

Steuerzeichen im Text

Initialisierung der Zähler chars, words und lines, sowie des „Zustands“ „im Wort“

Einlesen des 1. Zeichens

Schleife „arbeitet“ solange Zeichen eingelesen werden.
Analyse aller eingelesenen Zeichen:
Zeichen im Wort?
Wenn nein: neue Zeile, Tab, oder Blank

Einlesen des „nächsten“ Zeichens

Ausgabe des Ergebnisses

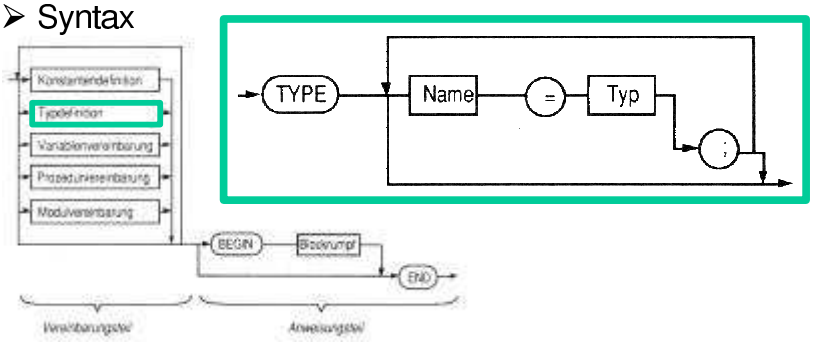
Zeilennummerierung

```
MODULE numlines;
(* text files with line numbers *)

FROM InOut IMPORT Read, Done, Write, WriteString, WriteLn, WriteCard;

CONST  newline = 12C;

VAR
   number : CARDINAL;
   ch     : CHAR;
   first  : BOOLEAN;

BEGIN
  number := 0;
  first  := TRUE;

  Read(ch);
  WHILE Done DO
    IF first THEN
      number := number + 1;
      WriteCard(number,4); WriteString(": ");
      first := FALSE
    END; (* IF *)

    IF ch <> newline THEN
      Write(ch)
    ELSE WriteLn; first := TRUE
    END; (* IF *)

    Read(ch)
  END (* WHILE *)
END numlines.
```

Einfache selbstdefinierte Typen

Typ-Definition

- Einordnung
  - bisher: „nur“ vordefinierte Datentypen (Standardrepertoire der Sprache)
  - jetzt: Datentypen selbst definieren
    - Aufzählungs- und Unterbereichstypen → Mittel zur Strukturierung
    - kompliziertere Strukturen → Mittel zur Abstraktion (später)
- Syntax



Aufzählungs- und Unterbereichstypen

- Allgemein: Ordinal-Datentypen (geordnet)
  - bisher: vordefinierte Datentypen mit interner Ordnung
  - jetzt: explizite Definition von Wertemengen durch
    - Aufzählung (Reihenfolge = Ordnung)
    - Selektion von Teilmengen (Unterbereiche)
- Aufzählungstyp ("enumeration")
  - eine geordnete Menge von Werten wird spezifiziert durch Aufzählung (konstanter Identifikatoren)
  - Beispiele:
    - mit Variablendeklaration

```
VAR
 Tag : (Montag, Dienstag, Mittwoch, Donnerstag,
        Freitag, Samstag, Sonntag);

 Farbe : (Weiss, Rot, Blau, Gelb, Gruen, Schwarz);
```

Aufzählungs- und Unterbereichstypen

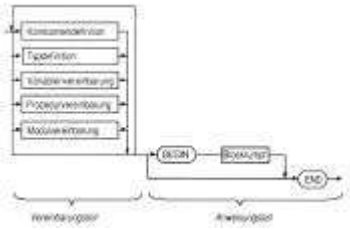
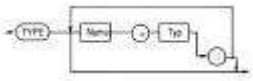
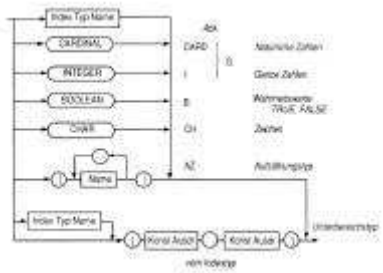
- Aufzählungstyp ("enumeration") (cont'd)
  - Beispiele:
    - mit Typdefinition

```
TYPE
  Wochentag = (Montag, Dienstag, Mittwoch, Donnerstag,
               Freitag, Samstag, Sonntag);
  FarbPalette = (Weiss, Rot, Blau, Gelb, Gruen, Schwarz);

VAR
  Tag        : Wochentag;
  Farbe      : FarbPalette;
```

    - wäre BOOLEAN nicht schon Standard

```
TYPE
  BOOLEAN = (FALSE, TRUE)
```

Aufzählungs- und Unterbereichstypen

➢ Aufzählungstyp (“enumeration”) (cont'd)

- Bemerkungen:
  - Elemente müssen eindeutig definiert sein, dürfen also nicht in 2 Typen vorkommen
  - Bezeichner dürfen nur in einer festgelegten Bedeutung vorkommen
    Beispiele:
    ```
    Karte : (7, 8, 9, Bube, Dame, König, 10, As)
    Tag   : (MO, DI, MI, DO, FR, SA, SO)
    ```
    aber erlaubt:
    ```
    Tag   : (Mo, Di, Mi, Do, Fr, Sa, So)
    ```
  - die Elemente einer Aufzählung erhalten analog ihrer Position in der Aufzählung eine Ordnungszahl zugeordnet
- Operationen und Aktionen
  - Wertzuweisung wie gewohnt: `Tag := Dienstag; Farbe := Blau;`
  - durch die Def. der Ordnung sind Vergleiche möglich {=,<>,<,<=,>,>=}
  - Funktionen
    - Minimum, Maximum: MIN(T) = a_0 , MAX(T) = a_n ← Typ T = (a_0,a_1,...,a_n)
    - Ordnungszahl- und Wertfunktion: ORD(...), VAL(...)
    - Nachfolger und Vorgänger: INC(...), DEC(...)