Kontroll-strukturen

- **Ablaufkontrolle**
- **Fallunterscheidungen**
  - IF
  - CASE
- **Wiederholungsanweisungen (Schleifen)**
  - WHILE, FOR
  - REPEAT-UNTIL, LOOP-EXIT

Ablaufkontrolle

Kontrollfluß

- **Ablaufsteuerung in der von Neumann-Maschine:**
  - Befehle stehen ***hintereinander*** im Speicher, werden vom Steuerwerk in den zentralen Prozessor geholt, dort decodiert und verarbeitet.
  - Durch ***Sprungbefehle*** kann von der Reihenfolge der gespeicherten Befehle abgewichen werden.
- **Abstraktion:**
  - Die Ausführungsreihenfolge der Aktionen eines Algorithmus entspricht zunächst der textuellen Anordnung (***Sequenz***). Davon kann aber abgewichen werden. Dazu gibt es eigene Anweisungen.
  - Ablaufsteuerung:
    - Fallunterscheidung,
    - Wiederholungen.
- **Ablaufsteuerung in imperativen Programmiersprachen:**
    - Anweisungsfolgen,
    - Fallunterscheidungen **`IF`**- und **`CASE`**-Anweisungen,
    - Schleifenformen **`WHILE`**-, **`UNTIL`**-, **`LOOP`**-Anweisungen.

Ablaufkontrolle

Kontrollstrukturen

- **Berechnungen in imperativen Programmen erfolgen durch die *Auswertung von Ausdrücken* und die *Zuweisung* von Werten zu Variablen.**
- **Zur Erhöhung der Flexibilität von Programmen sind zwei weitere Sprachmechanismen notwendig:**
  - die Steuerung des Kontrollflusses zur ***Auswahl*** zwischen unterschiedlichen Anweisungen, (Fallunterscheidungen)
  - die ***wiederholte*** Ausführung einer Folge von Anweisungen.
- **Sprachmechanismen, die dies leisten,**
  - heißen ***Kontrollstrukturen***.
- **Kontrollstrukturen**
  - zusammen mit den Anweisungsfolgen, die von ihnen kontrolliert werden, stellen den Anweisungsteil eines Programms dar.

Ablaufkontrolle

Beispiel GOTO (Sprunganweisung)

- **FORTRAN-Programm zur Bewertung von Meßdaten**

```
Beispiel GOTO in einem FORTRAN-Programm
100   format()
200   format(56h anzahl messwerte gut brauchbar schlecht unbrauchbar)
      ischl = 0
      ibr = 0
      igut = 0
      do 1 i = 1,n
      read (5,100) x
      abso = abs(x - s);
      if (abso .le. 0.01) goto 10
      if (abso .le. 0.05) goto 11
      if (abso .le. 0.2)  goto 12
      iunb = iunb + 1
      goto 1
10    igut = igut + 1
      goto 1
11    ibr = ibr + 1
      goto 1
12    ischl = ischl + 1
1     continue
      ...
```

Ablaufkontrolle

Diskussion: Kontrollstrukturen

- **Von Mitte der 60er bis Mitte der 70er wurde in der Informatik viel über *Kontrollstrukturen* diskutiert.**
- **Ein Ergebnis war:**
  - Obwohl eine einzige ***Anweisungsform*** (bedingter oder unbedingter Sprung) ausreicht, sollte genau diese Anweisung durch eine ***kleine Zahl*** von Kontrollanweisungen ***ersetzt*** werden.
- **Boehm und Jacopini haben 1966 nachgewiesen, daß alle Flußdiagramm-Programme durch zwei Kontrollstrukturen implementierbar sind:**
  - ***Auswahl*** zwischen alternativen Kontrollflüssen,
  - logisch kontrollierte ***Wiederholung***.
- **Kontrollstrukturen sollen *einen Einstieg* und *einen Ausstieg* (single entry, single exit) besitzen.**

Ablaufkontrolle

Diskussion Goto

- **Obwohl die *unbedingte Verzweigung* (Goto) ausreicht,**
  - alle anderen Kontrollstrukturen nachzubilden, führt ihre uneingeschränkte Verwendung zu unlesbaren und unzuverlässigen Programmen.
- **Hauptgrund:**
  - Durch Goto kann im Ablauf ***jede beliebige Reihenfolge*** von Anweisungen unabhängig von ihrer textlichen Anordnung erreicht werden.
  - In seinem Artikel ("Goto statement considered harmful", CACM, 1968, Vol.11, No.3, pp.147-149) schreibt E.W. Dijkstra: "The goto statement as it stands is just too primitive; it is too much an invitation to make a mess of one's program."
- **Dies hat die Goto-Debatte entzündet,**
  - die zwar zur softwaretechnischen Ablehnung des ***uneingeschränkten Goto*** geführt hat,
  - aber nur ***wenige*** Programmiersprachen haben völlig auf dieses Konstrukt verzichtet (Modula-2, Modula-3, Java).

Ablaufkontrolle

Konzept: Sequenz

- **Sequenz von Aktionen:**
  - Eine Aktion wird ***nach der anderen*** abgearbeitet.
  - Dazu muß nur klar sein, wie zwei Aktionsbeschreibungen voneinander ***getrennt*** sind.
  - Ein Aktion ist auch "tue nichts".

- **Beispiel:**
  - Algorithmus Telefonieren:
    - `hebe den Hörer ab;`
    - `wähle die Telefonnummer;`
    - `führe das Gespräch;`
    - `lege den Hörer auf`

Trennzeichen

Fallunter-scheidung

Konzept: Bedingte Anweisung (if)

- **Fallunterscheidungen kommen vor als:**
  - Einwegauswahl (one-way selection)
  - Zweiwegauswahl (two-way-selection)
  - allgemeine bedingte Anweisung

```
IF b THEN
  stmts;
ELSE
  stmts;
END;
```

Zweiwegauswahl, zweiseitig bedingte Anweisung

```
IF b THEN
  stmts;
END;
```

Einwegauswahl, einseitig bedingte Anweisung

...

allgemeine bedingte Anweisung

- **Anwendbar**
  - Typ des Ausdrucks, der die Auswahl bestimmt, ist BOOLEAN

Fallunter-scheidung

Auswahlanweisung (case)

- **Die *Auswahlanweisung* ermöglicht die Auswahl aus einer *beliebigen* Anzahl explizit angegebener Alternativen.**
- **Designentscheidungen sind:**
  - Welche Form und welcher Typ von Ausdruck ***kontrolliert*** die Mehrfachselektion?
  - Welche Form von Anweisung kann ***ausgewählt*** werden?
  - Wie ist der ***Kontrollfluß*** innerhalb der Mehrfachselektion?
  - Wie werden Selektionswerte behandelt, für die es ***keine*** passende Anweisungsalternative gibt?
- **Programmiersprachen realisieren die Auswahl durch die**
  - CASE-Anweisung

Fallunter-scheidung

CASE-Anweisung - 1

- **Hängen die Fälle einer Fallunterscheidung nur von unterschiedlichen Werten *eines Ausdrucks* ab,**
  - können wir die in modernen imperativen Sprachen vorhandene *CASE-Anweisung* verwenden.
- **Der ELSE-Teil ist im Beispiel leer, um einen *Fehlerfall* zu vermeiden.**
- **Dies ist eine häufige, aber *schlechte* Programmiertechnik.**
- **Besser:**
  - möglichen Fehlerfall explizit behandeln.

```
CASE zweierpotenz OF

  0 => ergebnis := 1;
| 1 => ergebnis := 2;
| 2 => ergebnis := 4;
| 3 => ergebnis := 8;
| 4 => ergebnis := 16;
ELSE
END;
```

Fallunter-scheidung

CASE-Anweisung - 2

```
MODULE CaseDemo EXPORTS Main;
IMPORT SIO;
VAR zweierpotenz, ergebnis: INTEGER;

BEGIN
  zweierpotenz := SIO.GetInt();
  ergebnis := 0;

  CASE zweierpotenz OF
     0 => ergebnis := 1;
   | 1 => ergebnis := 2;
   | 2 => ergebnis := 4;
   | 3 => ergebnis := 8;
   | 4 => ergebnis := 16;
  ELSE
    SIO.PutText ("Wert nicht definiert!");
  END;

  SIO.Nl(); SIO.PutInt(ergebnis);
END CaseDemo.
```

Bemerkung:

Werden nicht alle Werte als Alternativen aufgeführt und fehlt des ELSE-Zweig, dann kann das zu *Laufzeitfehlern* führen!!

Fehlerbehandlung im ELSE-Zweig ???

Fallunter-scheidung

CASE-Anweisung - 3

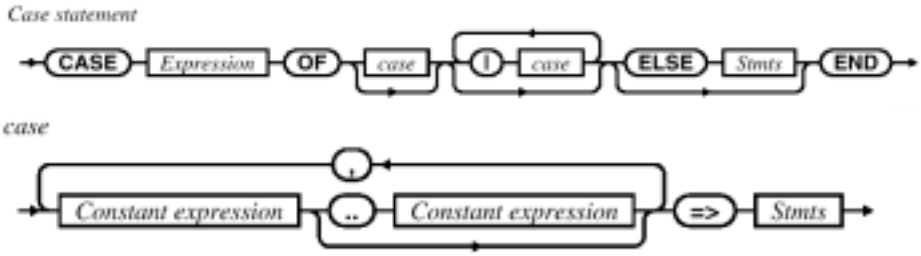
```
MODULE CaseDemo EXPORTS Main;
IMPORT SIO;
VAR zweierpotenz, ergebnis: INTEGER;

BEGIN
  zweierpotenz := SIO.GetInt();
  IF (zweierpotenz >= 0 AND zweierpotenz <= 4) THEN
    CASE zweierpotenz OF
      0 => ergebnis := 1;
    | 1 => ergebnis := 2;
    | 2 => ergebnis := 4;
    | 3 => ergebnis := 8;
    | 4 => ergebnis := 16;
    END;
    SIO.PutInt(ergebnis);
  ELSE
    SIO.PutText ("Wert nicht definiert!");
  END;
END CaseDemo.
```

Fehlersituation wird explizit vor Ausführung der CASE-Anweisung behandelt!

Fallunter-scheidung

Syntax CASE-Anweisung



Auswahlisten: Bequemlichkeit!

- **Zusatzbedingungen für die Case-Anweisung:**
  - Ergebnis von *Expression* und Ergebnis der *Constant expressions* müssen ***denselben*** Typ haben.
  - Zugelassen sind nur ***Ordinaltypen*** (z.B. BOOLEAN oder CHAR)
  - Die Werte dürfen jeweils ***nur einmal*** auftreten.

Schleifen

Konzept: Schleifen

- **Schleifen (Wiederholungsanweisungen, Iterationen) werden benötigt,**
  - um ***iterative Algorithmen*** zu formulieren.
- **Die historisch ersten Sprachkonstrukte zur Wiederholung waren**
  - für die ***Array-Bearbeitung*** gedacht,
  - da anfangs vorrangig numerische Probleme gelöst werden mußten.
- **Designentscheidungen sind:**
  - Wie wird die Wiederholung kontrolliert?
    - durch einen ***logischen Ausdruck***,
    - durch ***Abzählen***
  - Wo steht der Kontrollmechanismus im Programmtext?
    - am ***Anfang***/***Ende***,
    - ***automatisch***/***benutzerdefiniert***

Schleifen

Zählschleife (FOR-Anweisung)

- **Die FOR-Anweisung ist eine spezielle Schleifenform**
  - Anzahl der Wiederholungen ist ***im voraus*** bekannt
  - Wiederholungen werden durch Abzählen kontrolliert
- **Syntax:**

- **Anmerkungen:**
  - Die Steuerung und der Abbruch geschieht mit Hilfe der sog. ***Laufvariablen***, deren Schrittweite und Laufrichtung festgelegt werden kann.
  - In Modula-3: Die Laufvariable muß ***nicht deklariert*** werden.
  - Bei jedem Durchlauf wird die Laufvariable "***automatisch***" erhöht oder erniedrigt.
  - FOR-Schleifen werden oft bei der Bearbeitung von ***Arrays*** verwendet.

Schleifen

Beispiel: FOR-Anweisung

- **Ein Beispiel für die FOR-Schleife:**

```
(* Berechne Summe I = 1 bis 100 ueber I *)
r := 0;
FOR i := 1 TO 100 DO
  r := r + i
END;
```

- **FOR-Schleife mit Schrittweite:**

```
(* Berechne Summe I = 1 bis 100 ueber I *)
r := 0;
FOR i := 100 TO 1 BY -1 DO
  r := r + i
END;
```

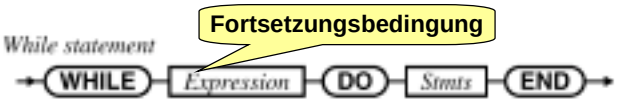
Innerhalb der FOR-Schleife muß die Laufvariable wie eine ***Konstante*** behandelt werden, darf also nicht auf der linken Seite einer Wertzuweisung oder als Referenzparameter stehen und damit ***manipulierbar*** sein.

Schleifen

Bedingte Schleife mit vorheriger Prüfung

- **WHILE-Schleife**
  - "***ablehnende Schleife***", da Prüfung vor Schleifendurchlauf gemacht wird
- **Syntax:**

```
WHILE E DO
  S;
END;
```



- **Bemerkung**
  - Ergebnis der "Expression" muß vom ***Typ BOOLEAN*** sein
  - Der Programmierer muß dafür sorgen, daß das Ergebnis von "Expression" ***irgendwann FALSE*** ist; ansonsten Endlosschleife; Programm terminiert nicht
- **Semantik (rekursiv definiert):**

```
IF E THEN
  S;
  WHILE E DO S; END;
END;
```


Schleifen

Beispiel: WHILE-Anweisung

- **Aufgabe:**
  - Man berechne den Quotienten und den Rest der ganzzahligen Division zweier positiver ganzer Zahlen a und b.

```
MODULE GanzzahlDivision EXPORTS Main;
IMPORT SIO;
VAR a, b, c: INTEGER;

BEGIN
  a := SIO.GetInt();
  b := SIO.GetInt();

  c := 0;  (*enthalte den Quotienten *)
  WHILE a >= b DO
    a := a - b;
    c := c + 1;
  END;

  SIO.PutText(" Quo :");  SIO.PutInt(c);
  SIO.PutText(" Rest :"); SIO.PutInt(a);

END GanzzahlDivision.
```

Wiederhol-bedingung

Schleifen

Bedingte Schleife mit nachfolgender Prüfung

- **REPEAT-UNTIL Anweisung (UNTIL-Schleifen)**
  - "***annehmende***" Schleife, da eine erste Ausführung stattfindet, ohne daß die Bedingung (in diesem Fall eine ***Abbruch-Bedingung***) geprüft wird
- **Syntax:**

- **Bemerkung**
  - REPEAT-UNTIL erfordert besondere Vorsicht
  - REPEAT-UNTIL ist immer dann sinnvoll, wenn der Bedingungswert erst durch die ***Anweisungen der Schleife*** entsteht
  - z.B. Eingabe mit Fehlerbehandlung

Schleifen

Beispiel: UNTIL

```
MODULE Einleseschleife EXPORTS Main;
IMPORT SIO;
VAR a: INTEGER;

BEGIN

  REPEAT
    SIO.PutText("Geben Sie eine Zahl zwischen 1 und 5 ein! ");
    SIO.Nl();
    a := SIO.GetInt();
  UNTIL (a >=1 AND a <=5);

  SIO.PutText("Ihre Eingabe war korrekt! ");

END Einleseschleife.
```

Abbruch-bedingung

Schleifen

Vergleich WHILE - UNTIL

```
PROCEDURE Einlesen () : TEXT =
  CONST Punkt = '.';
  VAR c : CHAR; t : TEXT := "";
BEGIN
  c := SIO.GetChar();
  WHILE c # Punkt DO
    t := t & Text.FromChar(c);
    c := SIO.GetChar();
  END;
  RETURN t;
END Einlesen;
```

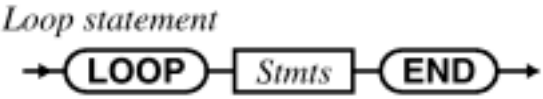
```
PROCEDURE Einlesen () : TEXT =
  CONST Punkt = '.';
  VAR c : CHAR; t : TEXT := "";
BEGIN
  c := SIO.GetChar();
  IF c # Punkt THEN
    REPEAT
      t := t & Text.FromChar(c);
      c := SIO.GetChar();
    UNTIL c = Punkt;
  END;
  RETURN t;
END Einlesen;
```

```
PROCEDURE Einlesen () : TEXT =
  CONST Punkt = '.';
  VAR c : CHAR; t, cText : TEXT := "";
BEGIN
  REPEAT
    t := t & cText;
    c := SIO.GetChar();
    cText := Text.FromChar(c);
  UNTIL c = Punkt
  RETURN t;
END Einlesen;
```


Schleifen

Endlosschleife

- **LOOP-Anweisung**
  - die Schleife wird solange durchlaufen, bis eine darin enthaltene ***EXIT-Anweisung*** ausgeführt wird
- **Syntax:**



- **Bemerkungen:**
  - in einer LOOP-Anweisung können ***mehrere*** EXIT-Anweisungen stehen
  - single-entry - single-exit wird dadurch verletzt
  - ist keine EXIT-Anweisung enthalten, so realisiert die LOOP-Anweisung eine ***nichtterminierende Schleife***

Schleifen

Beispiel: LOOP-Anweisung - 1

Algorithmus GGT

❶ Falls n < m ist, so vertausche man n und m

❷ Falls m = 0 ist, dann ist n der ggt(n,m) und man beende den Algorithmus

❸ Falls m≠ 0 ist, so bilde man den Rest r, der bei der Division von n durch m bleibt, dann ersetze man n durch m und m durch r und beginne von vorn.

```
MODULE GGT EXPORTS Main;
VAR n, m, hilf, r : INTEGER;
BEGIN
  n := SIO.GetInt();
  m := SIO.GetInt();
  LOOP
    IF n < m THEN  (* 1 *)
      hilf := m;
      m := n;
      n := m;
    END;
    IF m = 0 THEN  (* 2 *)
      EXIT;
    ELSE           (* 3 *)
      r := n MOD m;
      n := m;
      m := r;
    END;
  END (* LOOP *);
  SIO.PutText("GGT = ");SIO.PutInt(n);
END GGT.
```


Schleifen

Beispiel: LOOP-Anweisung - 2

Bemerkungen

- LOOP-Anweisungen terminieren nicht!
- Die Ausführung der EXIT-Anweisung terminiert die LOOP-Anweisung. Die Kontrolle geht zur Anweisung ***nach dem END***.
- Bei geschachtelten LOOP-Strukturen bewirkt die EXIT-Anweisung nur das Verlassen der ***zugehörigen*** LOOP-Struktur.

Empfehlung

- Aus Gründen der besseren Lesbarkeit sollte man überall dort WHILE- und REPEAT-Schleifen zu verwenden, wo nur eine Abfrage am ***Anfang*** oder ***Ende*** der Schleife erforderlich ist!

```
MODULE GGT1 EXPORTS Main;
IMPORT SIO;
VAR n, m, r : INTEGER;

BEGIN
  n := SIO.GetInt();
  m := SIO.GetInt();
  r := n MOD m;
  LOOP
    IF r = 0 THEN
      EXIT;
    ELSE
      n := m;
      m := r;
      r := n MOD m;
    END;
  END;
  SIO.PutText ("GGT = ");
  SIO.PutInt(m);
END GGT1.
```

Schleifen

LOOP versus WHILE

```
n := SIO.GetInt();
m := SIO.GetInt();

r := n MOD m;
LOOP
  IF r = 0 THEN
    EXIT;
  ELSE
    n := m;
    m := r;
    r := n MOD m;
  END;
END;

SIO.PutText ("GGT = ");
SIO.PutInt(m);
```

```
n := SIO.GetInt();
m := SIO.GetInt();

r := n MOD m;
WHILE r # 0 DO
  n := m;
  m := r;
  r := n MOD m;
END;

SIO.PutText ("GGT = ");
SIO.PutInt(m);
```

- **Hier bietet sich die WHILE-Anweisung an,**
  - da am Beginn der Schleife die Bedingung geprüft werden soll.

Schleifen

Wahl des Schleifenkonstrukts

- **REPEAT...UNTIL**
  - ist mit Vorsicht zu verwenden, denn die Anweisung in der Schleife wird ***mindestens einmal*** durchlaufen.
  - Sie ist nur sinnvoll, wenn der Bedingungswert ***erst in der Schleife entsteht***
  - kann durch Schleife ohne Prüfung realisiert werden (sollte man aber nicht!)

```
REPEAT
  SIO.PutText("Zahl zwischen 1 und 5! ");
  SIO.Nl();
  a := SIO.GetInt();
UNTIL (a >=1 AND a <=5);

LOOP
 SIO.PutText("Zahl zwischen 1 und 5! ");
 SIO.Nl();
 a := SIO.GetInt();
 IF (a >=1 AND a <=5) THEN EXIT END;
END;
```

Schleifen

Was haben wir gelernt!

- **Fallunterscheidungen**
  - einseitig bedingte — IF-THEN
  - zweiseitig bedingte — IF-THEN-ELSE
  - allgemeine bedingte — mit ELSIF
  - Auswahlanweisung — CASE
- **Wiederholungsanweisungen (Schleifen)**
  - WHILE
  - FOR
  - REPEAT-UNTIL
  - LOOP-EXIT
- **Einsatz der Wiederholungsanweisungen**
  - Die Wahl der geeigneten Ausdrucksmittel hängt vom jeweiligen Konstruktionsproblem ab.
  - Implementationsgesichtspunkte sind ggf. zu berücksichtigen.
  - Softwaretechnische Überlegungen wie Verständlichkeit und Sicherheit spielen bei der Verwendung eine zentrale Rolle.

Glossar

- **Ablaufkontrolle, Kontrollfluß, zugehörige Kontrollstrukturen für zusammengesetzte Anweisungen, Kontrollstrukturen als Konstruktoren für die Ablaufkontrolle**
- **Fallunterscheidung und Zusammenführung im Kontrollfluß**
- **Kontrollstrukturen für Fallunterscheidungen: bedingte Anweisung (einseitig, zweiseitig, mehrseitig), Auswahlanweisung (Mehrfachselektion), Steuerung des Kontrollflusses durch Boolesche Ausdrücke bzw. Aufzählen der Fälle durch Auswahllisten**
- **GOTO-Kontroverse, wohlstrukturierte Programme (Sequenz, Fallunterscheidung, Iteration), wohlstrukturierte Programme mit Escape (Exit), Umwandlung beliebiger Programme in wohlstrukturierte**
- **Schleifenformen: Zählschleife, WHILE-Schleife, UNTIL-Schleife, Endlosschleife**
- **Anwendungsfälle für verschiedene Schleifenformen**
- **Termination bei verschiedenen Schleifenformen**