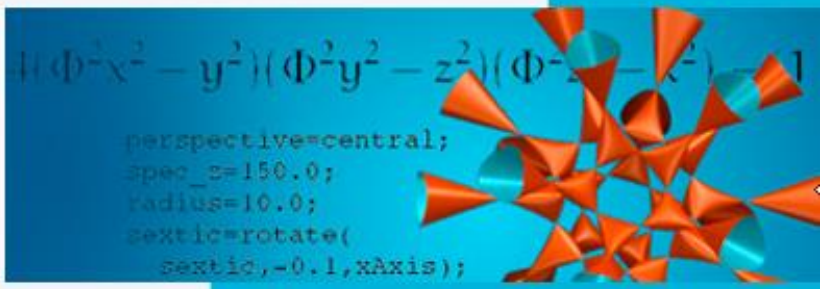
Philipps Universität Marburg

Vorkurs Informatik

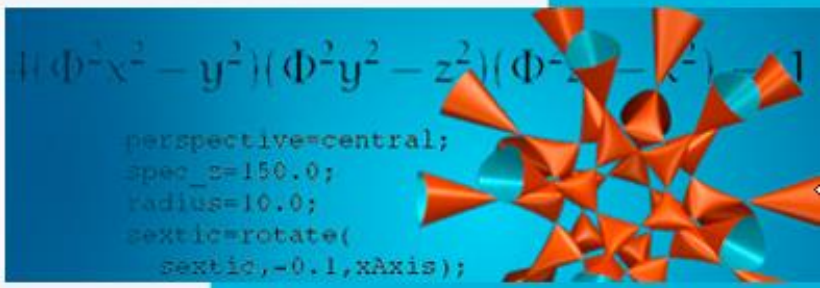
Wintersemester 2024/25

Prof. Christoph Bockisch (Programmiersprachen und –werkzeuge)
Teresa Dreyer



Philipps Universität Marburg

KONTROLLSTRUKTUREN IN PASCAL



Der Datentyp Boolean

- Um Kontrollstrukturen zu verwenden, müssen wir Bedingungen formulieren und überprüfen, ob diese Bedingungen erfüllt sind. Dazu verwenden wir den Datentyp **boolean**.

- Eine Variable vom Datentyp boolean hat entweder den Wert *true* oder den Wert *false*.

- Ausdrücke über Integervariablen oder Stringvariablen können auch Werte des Typs boolean liefern:

- 10<12 liefert ...
- 12<10 liefert ...
- 10=12 liefert ...

Der Datentyp Boolean

- Um Kontrollstrukturen zu verwenden, müssen wir Bedingungen formulieren und überprüfen, ob diese Bedingungen erfüllt sind. Dazu verwenden wir den Datentyp **boolean**.

- Eine Variable vom Datentyp boolean hat entweder den Wert *true* oder den Wert *false*.

- Ausdrücke über Integervariablen oder Stringvariablen können auch Werte des Typs boolean liefern:

- 10<12 liefert ...
- 12<10 liefert ...
- 10=12 liefert ...
- '23+19' = '42' liefert ...
- 'Informatik' < 'Mathematik' liefert ...
- 'ABC' >= 'ABC' liefert …

Operationen mit Booleans

- Auch mit booleschen Werten kann gerechnet werden. Dazu verwendet man logische Operatoren, deren Anwendung auf ein oder zwei boolesche Werte wiederum einen booleschen Wert liefert.

Für boolesche Werte *a* und *b* sind die wichtigsten Operatoren wie folgt definiert:

- **not** *a* hat Wert *true* genau dann, wenn *a* den Wert *false* hat
- *a* **or** *b* hat Wert *true* genau dann, wenn mindestens eins von *a* und *b* den Wert *true* hat
- a **and** b hat Wert *true* genau dann, wenn *a* und *b* beide den Wert *true* haben

Auswertungsreihenfolge

- In welcher Reihenfolge werden Ausdrücke ausgewertet?

- Kennen Sie eine Regel?

```
program Hello;
var
  ergebnis1 : Integer;
  ergebnis2 : Boolean;
begin
  ergebnis1 := 1 + 2 * 3;
  ergebnis2 := not true and false or true;
  writeln(ergebnis1);
  writeln(ergebnis2);
end.
```

Auswertungsreihenfolge

- In welcher Reihenfolge werden Ausdrücke ausgewertet?

- Kennen Sie eine Regel?

```
program Hello;
var
  ergebnis1 : Integer;
  ergebnis2 : Boolean;
begin
  ergebnis1 := 1 + 2 * 3;
  ergebnis2 := not true and false or true;
  writeln(ergebnis1);
  writeln(ergebnis2);
end.
```

Ergibt: 7

Ergbit: TRUE

Auswertungsreihenfolge

Operatoren haben eine Auswertungsreihenfolge:
Bekannt: Punkt vor Strich
Für andere Operatoren existieren ähnliche Regeln, z.B.:
not vor and beziehungsweise not vor or
and vor or
Arithmetische Operatoren (+, *, …) vor Vergleichsoperatoren (<, >, …)
Klammern legen die Auswertungsreihenfolge fest.
Im Zweifel immer vollständig klammern:

```
ergebnis1 := 1 + (2 * 3);
ergebnis2 := (((not true) and false) or true);
```

Kontrollstrukturen

- **Ziel:** Das Programm soll auf verschiedene Eingaben unterschiedlich reagieren.

- Dazu verwendet man **Kontrollstrukturen**. Diese steuern den Ablauf des Programms in Abhängigkeit von durch Programmierer definierte **Bedingungen**.

- **Zwei Haupttypen**:
- Verzweigungen
  - Zweifache Verzweigung: if …, if .. then … else
  - Mehrfache Verzweifung: case (…) of … end
- Schleifen
  - Vorgegebene Anzahl Durchläufe: for … do
  - Keine feste Anzahl: repeat … until, while (…) do …, repeat … until

Dokumentation

Zum Beispiel:

Verzweigungen:
https://www.tutorialspoint.com/pascal/pascal_decision_making.htm
Schleifen:
https://www.tutorialspoint.com/pascal/pascal_loops.htm

Verzweigung mittels if-Statement

- Abhängig vom Wert eines booleschen Ausdrucks sollen verschiedene Programmabläufe stattfinden.
- **Syntax:** if <boolean-Wert> then <Anweisung> else <Anweisung>;

```
begin
  a:= 3;
  b:= 4;
  c:= 5;
  if a*a+b*b=c*c then
            Writeln('Pythagoras hatte recht.')
  else
            Writeln('Pythagoras hatte unrecht oder' +
            'mit den Zahlen stimmt etwas nicht.');
end.
```

Achtung: kein Semikolon vor dem else. Die if-Anweisung ist noch nicht abgeschlossen.

Eine einzelne Anweisung. Was, wenn mehrere Anweisungen ausgeführt werden sollen?

Achtung: hier kommt das Semikolon, denn die if-Anweisung ist jetzt fertig.

Verzweigung mittels if-Statement

- Abhängig vom Wert eines booleschen Ausdrucks sollen verschiedene Programmabläufe stattfinden.
- **Syntax:** if <boolean-Ausdruck> then <Anweisung> else <Anweisung>;

```
begin
  a:= 3;
  b:= 4;
  c:= 5;
  if a*a+b*b=c*c then
    begin
          Writeln('Pythagoras hatte recht.');
    end
  else
    begin
          Writeln('Pythagoras hatte unrecht oder');
          Writeln( 'mit den Zahlen stimmt etwas nicht.');
    end;
end.
```

Mehere Anweisungen können in einen begin…end Block gruppiert werden.

Innerhalb von begin .. end werden Anweisungen wieder mit ; abgeschlossen.

Beispiel: Test einer geratenen Zahl

- **Ziel:** Wir wollen ein Spiel programmieren, bei dem der Benutzer soll eine Zahl zwischen 1 und 100 erraten.
- Der Test, ob er richtig geraten hat, soll durch ein if-Statement erfolgen. Der Benutzer soll entsprechendes Feedback erhalten.

  → Gemeinsame Erarbeitung des Quellcodes.

Die Repeat-Schleife

- **Ziel:** Der Nutzer soll solange raten dürfen, bis er richtig liegt.
- Dazu eignet sich eine repeat-Schleife, bei der ein Anweisungsblock wiederholt wird bis eine vorgegebene Bedingung erfüllt ist.
- **Syntax: repeat** <Anweisungen> **until** <boolean-Ausdruck>;

```
begin
  i:= 0;
  j:= 0;
  repeat
              i:=i+1;
              j:=j+i;
  until i=100;
  Writeln(j);
end.
```

Kein begin…end nötig für mehrere Anweisungen, da repeat…until schon einen Block markiert.

For-Schleifen

- Im vorigen Beispiel: Die Anzahl der Schleifendurchläufe hängen nur von dem Integer-Wert *i* ab, der sich sehr regelmäßig ändert.
- In solchen Fällen verwendet man lieber for-Schleifen.
- **Syntax:**

for <Variablenname>:= <Anfangswert> **to** <Endwert> **do** <Anweisung>**;**

```
begin
  j:=0
  for       i:= 0 to 100 do
            j:= j+i;
  Writeln(j);
end.
```

Anfangswert und Endwert können natürlich auch durch einen Ausdruck angegeben werden.

Arrays

- Wichtige Anwendung von For-Schleifen: Iterieren über **Arrays**.
- Ein Array ist eine Zusammenfassung von Elementen des gleichen Typs.
- Im Array liegen die Elemente nebeneinander, der Zugriff erfolgt über einen Indexwert.
- Ein Array hat eine feste Länge (= Anzahl der Einträge).
  - Haben wir so etwas schon einmal gesehen?

Arrays

- Wichtige Anwendung von For-Schleifen: Iterieren über **Arrays**.
- Ein Array ist eine Zusammenfassung von Elementen des gleichen Typs.
- Im Array liegen die Elemente nebeneinander, der Zugriff erfolgt über einen Indexwert.
- Ein Array hat eine feste Länge (= Anzahl der Einträge).

- Sei *A* ein Array mit 100 Werten und Startindex 0. Man erhält mit
- *A*[9]
- *A*[5000]
- *A*[0]
- *A*[100]

Anders als bei Strings fangen wir hier bei ‘0’ an zu zählen!

- **Hinweis:** In Pascal kann zwar das Indexintervall flexibel festgelegt werden, in anderen Sprachen beginnt der Index aber immer bei 0!

Arrays

- Wichtige Anwendung von For-Schleifen: Iterieren über **Arrays**.
- Ein Array ist eine Zusammenfassung von Elementen des gleichen Typs.
- Im Array liegen die Elemente nebeneinander, der Zugriff erfolgt über einen Indexwert.
- Ein Array hat eine feste Länge (= Anzahl der Einträge).

- Sei *A* ein Array mit 100 Werten und Startindex 0. Man erhält mit
- *A*[9] — Den 10. Eintrag.
- *A*[5000] — Einen Fehler.
- *A*[0] — Den 1. Eintrag.
- *A*[100] — Einen Fehler.
- **Hinweis:** In Pascal kann zwar das Indexintervall flexibel festgelegt werden, in anderen Sprachen beginnt der Index aber immer bei 0!

Arrays: Beispiel

- **Syntax zur Deklarierung von Arrays:**
- a : array [<Indizes>] of <Elementtyp>;

```
var
  A : array [0 .. 9] of Integer;

begin
  for i := 0 to 9 do
    A[i] := i * 10;
  for i := 0 to 9 do
    Writeln(A[i]);
end.
```

- **Ziel:** Speichere im Ratespiel alle geratenen Zahlen und gib diese
- am Ende aus.
- ⟶ Gemeinsame Erarbeitung am Quellcode.

Stringoperationen

- Einige nützliche Operationen auf Strings werden von der Bibliothek *StrUtils* bereitgestellt. Diese kann man etwa verwenden um aus einer Datei strukturierte Informationen auszulesen.

- *LastDelimiter*(*c,s*); gibt das letzte Vorkommen des von *c* in *s* aus.
- *RightStr*(*s,p*) gibt die letzten *p* Positionen von *s* aus.
- *LeftStr*(*s,p*) gibt die ersten *p* Positionen von *s* aus.

```
uses SysUtils,StrUtils;

begin
 Writeln(LastDelimiter('b','abba'));
 Writeln(RightStr('abba',2));
 Writeln(LeftStr('abba',length('abba')-1);
end.
```