Informationstechnik

Programmieren mit PERL eine Einführung

Prof. Dr. Wolf-Fritz Riekert
Hochschule für Bibliotheks- und Informationswesen (HBI) Stuttgart

mailto:riekert@hbi-stuttgart.de
http://v.hbi-stuttgart.de/~riekert

Definitionen

Algorithmus: Arbeitsanleitung zum Lösen eines Problems oder einer Aufgabe, die so präzise formuliert ist, dass sie im Prinzip auch von einem Computer ausgeführt werden kann.

Programmablaufpläne (Flussdiagramme) und **Struktogramme** (Nassi-Shneidermann-Diagramme) dienen zur graphischen Darstellung von Algorithmen.

Programmiersprachen dienen zur Formulierung von Algorithmen.

Ein in einer Programmiersprache formulierter Algorithmus heißt **Programm**.

In Form von Programmen können Algorithmen durch einen **Computer** ausgeführt werden.

Aufgabe: Der „Quadratzahltest“

Aufgabe:

Es ist ein Programm zu schreiben, das testet, ob eine vom Benutzer eingegebene Zahl eine Quadratzahl ist.

Beispiel:

Benutzer:	**9**
System:	**Quadratzahl**
Benutzer:	**3**
System:	**keine Quadratzahl**

Ein Algorithmus zur Lösung der Aufgabe

Algorithmus	**Beispiel 1**	**Beispiel 2**
Zahl *n* einlesen.	n = 9	n = 3
Der Reihe nach alle Quadratzahlen q bilden: *q* = 0, 1, 4, usw.	q = 0*0 = 0 q = 1*1 = 1 q = 2*2 = 4 q = 3*3 = 9	q = 0*0 = 0 q = 1*1 = 1 q = 2*2 = 4
Weitermachen, solange *q* < *n* (kleiner als *n*) ist. Aufhören, wenn das nicht mehr der Fall ist.	Jetzt gilt q < n nicht mehr	Jetzt gilt q < n nicht mehr
Wenn nun *n* = *q* ist, war *n* eine Quadratzahl, sonst nicht.	n = q ? Ja! ⇒ n ist Quadratzahl	n = q ? Nein! ⇒ n ist keine Q'zahl.

Umsetzung des Algorithmus in ein Struktogramm

Algorithmus

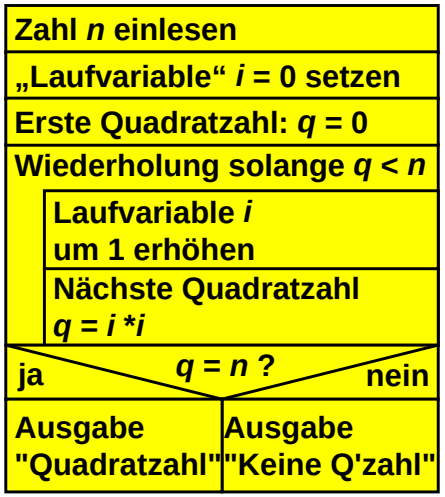
Zahl *n* einlesen.

Der Reihe nach alle Quadratzahlen *q* bilden: *q* = 0, 1, 4, usw.

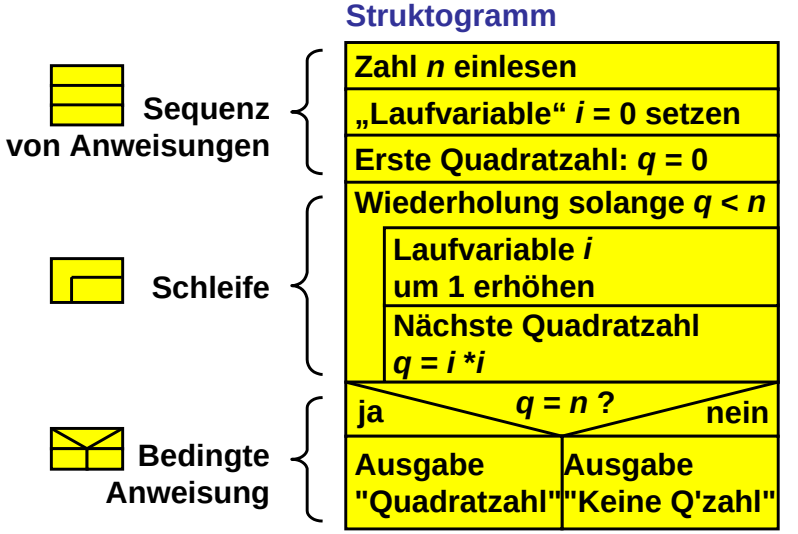
Weitermachen, solange *q* < *n* (kleiner als *n*) ist. Aufhören, wenn das nicht mehr der Fall ist.

Wenn nun *q* = *n* ist, war *n* eine Quadratzahl, sonst nicht.

Struktogramm



Aufbau von Struktogrammen

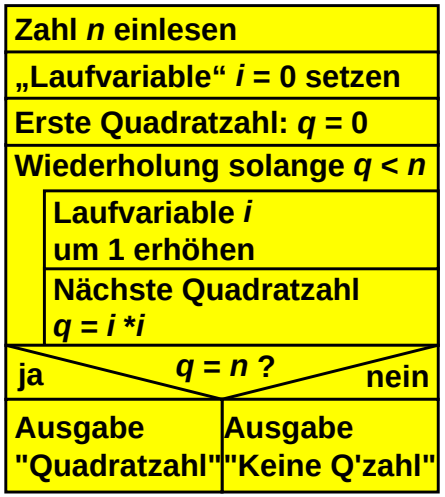


Umsetzung des Struktogramms in ein PERL-Programm

PERL-Programm

```
$n = <STDIN>;
$i = 0;
$q = 0;
while ($q < $n) {
  $i = $i + 1;
  $q = $i * $i
  }
if ($q == $n) {
  print "Quadratzahl"
  }
else {
  print "Keine Q'zahl"
  }
```

Struktogramm

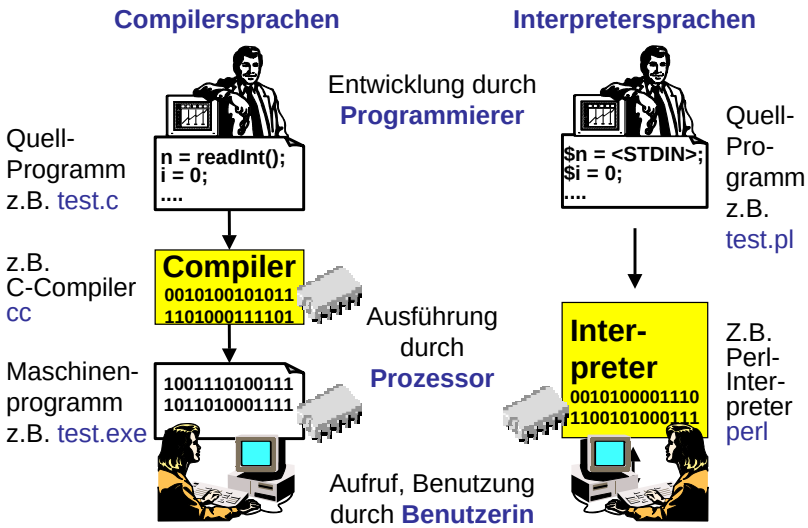


Art der Ausführung eines Programms durch den Computer

- Ein Prozessor (z.B. der Pentium-Prozessor) kann nur sogenannte **Maschinenprogramme** ausführen, diese bestehen aus einer Folge von Zahlencodes.
- Menschen schreiben Maschinenprogramme in einer sog. **Assemblersprache**, dabei sind die Zahlencodes durch Namen von Maschinenbefehlen ersetzt.
- Meist schreibt man Programme aber in **höheren Programmiersprachen** (z.B. C, Java, Perl)
  - ⇨ Sie werden dann entweder mit einem **Compiler** in Maschinenprogramme übersetzt und anschließend durch den Prozessor ausgeführt
  - ⇨ oder durch einen sog. **Interpreter** ausgeführt.

Compiler und Interpreter sind selbst Programme.

Compilersprachen und Interpretersprachen



Eigenschaften von Compilersprachen

- Der Compiler wird nur bei der Programmentwicklung gebraucht, im Betrieb läuft das übersetzte Programm ab.
- Deshalb hat der Compiler Zeit für aufwendige Programmüberprüfungen und Optimierungen.
- Compiler überprüfen Programme hinsichtlich „Vokabular“ und „Grammatik“, so dass viele Programmierfehler bereits bei der Compilierung entdeckt werden können.
- Durch Compiler übersetzte Programme sind Maschinenprogramme, die in der Regel sehr schnell und effizient ablaufen.
- Die Programmentwicklung mit Compilersprachen ist etwas mühevoll, da ein Programm nach jeder Änderung neu compiliert werden muss.
- Wichtigstes Beispiel für Compilersprachen: C/C++

Eigenschaften von Interpretersprachen

- Interpreterprogramme funktionieren nicht für sich alleine, sie benötigen zur Ausführung einen Interpreter.
- Da der Interpreter zur Laufzeit des Programms aktiv ist, hat er wenig Zeit für aufwendige Prüfungen. Fehlerhafte Programme „stürzen“ oft mit einer kurzen Meldung „ab“.
- Es gibt jedoch Programmentwicklungsumgebungen mit Editoren, die Syntaxüberprüfungen vornehmen.
- Interpretierte Programme sind deutlich langsamer als compilierte, was mit den heutigen schnellen Computern allerdings kein großes Problem mehr darstellt.
- Die Programmentwicklung ist erleichtert, da Programme nach Änderungen sofort wieder gestartet werden können.
- Beispiele für Interpretersprachen: Visual Basic for Applications (VBA), Javascript, **PERL**

Anweisungen in PERL

PERL-Programme werden aus Anweisungen gebildet, die mit „`;`“ verkettet werden.

Elementare Anweisungen sind:

- `$i=$i+1` **Zuweisungen**, z.B.: `$i = $i + 1`
- `print $i` **Befehle**, z.B.: `print $i`
- `&up` **Unterprogrammaufrufe**, z.B.: `&TitelAusgeben`

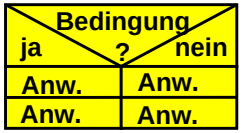
Zusammengesetzte Anweisungen sind:

- **Sequenzen** von Anweisungen, verkettet mit „`;`“, geklammert durch „`{}`“
  z.B.: `{$i = $i+1; $s = $s+i}`
- **Bedingte Anweisungen**, z.B.: `if`-Anweisung
  `if ($x>0) {$y=$x} else {$y=-$x}`
- **Schleifen**, z.B.: `while`-Anweisung
  `while($i<10){$erg = $erg+$i; $i=$i+1}`

Bedingte Anweisungen und Programmschleifen in PERL

Bedingte Anweisung:

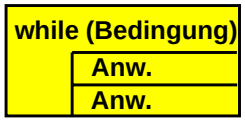
```
if (Bedingung)
      {Anweisung; Anweisung; ...}
else
      {Anweisung; Anweisung; ...}
```



Der Else-Zweig kann auch entfallen, sofern er nicht nötig ist

Schleife:

```
while (Bedingung)
      {Anweisung; Anweisung; ...}
```



Variablen: Skalare und ihre Werte

In Variablen können Werte gespeichert werden.
Die wichtigste Art von Variablen sind sog. Skalare.
Die Namen von Skalaren beginnen mit $ (Dollarzeichen)
Skalare können genau einen Wert speichern.

Die Werte können Zahlen sein oder Texte (Strings).
Texte werden mit Hilfe von einfachen (') oder doppelten (") Anführungszeichen angegeben.
Doppelte Anführungszeichen ermöglichen „Interpolation“.

Interpolation

```
$hausnummer = 32;
$strasse = 'Wolframstraße';
$satz = "Komm in die $strasse $hausnummer";
print $satz;
Ausgabe: Komm in die Wolframstraße 32
```


Stringliterale

Sonderzeichen in Strings lassen sich mit Hilfe des Gegenschrägstriches (Backslash) angeben. Die Strings müssen dazu in Doppelanführungszeichen (“) stehen:

\"	doppeltes Anführungszeichen
\'	einfaches Anführungszeichen
\\	Gegenschrägstrich (Backslash)
\n	Zeilenvorschub (Newline)
\033	Sonderzeichen mit Oktalcode 33 (ESC)
\x7F	Sonderreichen mit Hexcode 7F (DEL)
\cC	Strg-C (Control-C)

```
print "Der \"zwanghafte\nProgrammierer\"";
```

⇨ Der "zwanghafte
Programmierer"

Ausdrücke

Zahlenwerte lassen sich mit Hilfe von Rechenoperatoren (Grundrechenarten: `+ - * /`, Divisionsrestbildung: `%`, Potenzbildung: `**`) zu Ausdrücken kombinieren:

```
$umfang = 2 * ($laenge + $breite);
$neunerrest = $wert % 9;
$xquadrat = $x ** 2;
```

Stringoperatoren dienen zur Bildung von Texten (Stringverkettung: `.` , String-„Multiplikation“: `x`):

```
print "Gesamtpreis = " . 2*3 . " Euro";
```

⇨ Gesamtpreis = 6 Euro

```
print "LA" x 5
```

⇨ LALALALALA

Vergleich von Werten

Für Zahlen und Texte gibt es unterschiedliche Vergleichsoperatoren

```
if ($hausnummer == 34)
   {print "Nebengebäude"}
```

Achtung! Nicht verwechseln:
== Zahlenvergleich
eq Textvergleich
= Zuweisung, kein Vergleich

```
if ($strasse eq 'Wolframstraße')
   {$hausmeister = "Rüber"}
```

Zahlen	Texte*	Bedeutung
==	eq	gleich
!=	ne	nicht gleich
<	lt	kleiner als
<=	le	kleiner oder gleich
>	gt	größer als
>=	ge	größer oder gleich

***** Der Vergleich von Texten erfolgt alphabetisch (wie im Lexikon), es gilt: **'Anna' lt 'Otto'** und sogar: **100 lt 20 (!)**

Variablen: Arrays (Listen)

Arrays (Listen) sind besondere Variablen, die mehrere Werte speichern können.

- Die Namen von Arrays beginnen stets mit dem @-Zeichen.
- Für den Zugriff auf einzelne Elemente verwendet man das **$**-Zeichen sowie einen numerischen Index in **[**eckigen Klammern**]** , der ab 0 zählt.

```
@woche = ('mo', 'di', 'mi', 'do', 'fr', 'sa');
print "$woche[1]\n";
```

⇨ di

```
$woche[6] = 'so';
print "Alle Tage: @woche\n";
```

⇨ Alle Tage: mo di mi do fr sa so

Variablen: Hashs (Assoziationslisten)

Hashs sind besondere Variablen, die Assoziationslisten speichern können.

- Die Namen von Hashs beginnen stets mit dem **%**-Zeichen.
- Für den Zugriff auf einzelne Elemente verwendet man das **$**-Zeichen sowie einen Zugriffsschlüssel in **{**geschweiften Klammern**}**.

```
%translate = ('gut' => 'good', 'schlecht' => 'bad');
print "I feel $translate{'schlecht'}\n";
```

⇨ I feel bad

```
$translate{'sehr'} = 'very';
print "PERL is $translate{'sehr'} $translate{'gut'}\n";
```

⇨ PERL is very good

Variablen: Handles

Handles sind eine besondere Art von Variablen, die für Ein-/Ausgabeströme stehen.

Die Standard-Handles **STDOUT** (für Ausgabe), **STDIN** (für Eingabe) und **STDERR** (für Fehlerausgabe) sind defaultmäßig mit Tastatur und Bildschirm verbunden.

Mit dem Befehl **open** können weitere Handles mit Dateien verbunden werden.

```
print "Diese Ausgabe geht per Default nach STDOUT";
$dateiname = <STDIN>;
if (open (MYFILE, ">$dateiname")) {
      print MYFILE "Das steht gleich in der Datei";
      close (MYFILE) }
else {print STDERR "Fehler beim Oeffnen"}
```

Wie weiter?

- Versuchen Sie, vorhandene Skripte nach Ihren Bedürfnissen zu modifizieren!
- SELFHTML (http://www.teamone.de/selfhtml/)
- Selbststudium nach Büchern
  - ⇨ Thorsten Roßner: Das Einsteigerseminar Perl. bhv Verlag, 1998. (19,80 DM).
    Enthält auch CGI-Skript-Programmierung
  - ⇨ L. Wall, R. Schwartz, S. Potter: Programmieren mit Perl. O'Reilly.
  - ⇨ R. Schwartz, T. Christiansen: Einführung in Perl. O'Reilly
- Online Documentation von Active Perl (http://www.ActiveState.com/ActivePerl/download.htm)