|  |  |
| --- | --- |
| www.ovm-kassel.de  |  Infoblatt  | 21.04.1966 |  |
| Infoblatt "Pseudocode“ |  |
| **Code AE-MS-I-3** |  |
| **Datum** |  |
| **Links** |  |
| **Verwandte Lernjobs** |  |

**Übersicht über die wichtigsten Pseudocode**

**Anweisungen**

Der Pseudocode stellt einen Algorithmus in einer Schreibweise in der Art einer Programmiersprache dar, die aber in der Regel näher an der natürlichen Sprache ist als am formalen Programmcode. Es gibt keine Standardisierung bzw. Vorschrift, wie ein Pseudocode auszusehen hat; die Ablaufstrukturen werden grob durch Texte und fest vorgeschriebene Schlüsselwörter beschrieben. Guter Pseudocode zeichnet sich durch eine kurze präzise Beschreibung des Algorithmus aus.

a)**Einrücken**

Durch Einrücken wird die Blockstruktur gekennzeichnet. Auf diese Weise wird der

Pseudocode besser leserlich. Die zu den folgenden Punkten gehörenden Beispiele

werden dies verdeutlichen.

b)**// Kommentar**

Durch “//” wird gekennzeichnet, dass es sich beim Rest der Zeile um einen Kom-

mentar handelt. Kommentare werden nicht ausgeführt!

Beispiel: i = 0 // Ab hier beginnt der Kommentar

// Diese Zeile besteht nur aus Kommentar

c)**print**

**print** gibt einen Text aus. Dabei werden Zeichenketten in doppelte Anführungszei-

chen gesetzt. Bei Variablen (außerhalb von doppelten Anführungszeichen) wird der

Wert der Variablen ausgegeben.

Beispiel: Das erste**print** gibt Hello World aus. Das zweite**print** gibt i = 1 aus, da i den

Wert 1 hat.

i=1

**print** “Hello World”

**print** “i = ” i

d)**while**

Eine while Schleife für einen Schleifenkörper so lange aus, wie eine Bedingung WAHR

ist. Der Schleifenkörper besteht aus einer oder mehreren Anweisungen, die durch

Einrücken gekennzeichnet sind.

Beispiel: Die folgende Schleife gibt den Wert von i mit**print** aus und erhöht dann i

um 1. Wenn i den Wert 5 erreicht, nimmt die Bedingung “i < 5” den Wert FALSCH an,

und die Schleife wird nicht mehr ausgeführt.

// “i < 5” stellt bei dieser while-Schleife die Bedingung dar

**while** i < 5

// Die auf den Kommentar folgenden beiden eingerückten Zeilen bilden

// bei dieser while-Schleife den Schleifenkörper

**print** “i hat den Wert ” i

i=i+1

e)**for** <var> = a**to** b

Ein**for**-Schleife zählt die Variable <var> von a ausgehend hoch und führt in jedem

Schritt die Anweisungen in der Schleife aus. Wenn der Wert von <var> b überschrei-

tet, wird die Codeausführung nach der Scheife fortgesetzt. Wie bei**while** bilden auch

hier die eingerückten Zeilen den Schleifenkörper.

Beispiel: Die folgende Schleife summiert in der Variablen i alle Werte von 1 bis 6 auf.

Danach werden mit**print** die Werte von i und j ausgegeben.

i = 0**for** j = 1**to** 6

i=i+j

// Die Zeile mit dem**print** unter diesem Kommentarblock ist nicht

// eingerückt - sie beﬁndet sich also außerhalb des Schleifenkörpers

// Die Ausgabe lautet i hat jetzt den Wert 7 und j ist 21, denn:

// i hat erst mit dem Wert 7 die Bedingung der**for**-Schleife verletzt

// (6 überschritten) und j ist die Summe der Zahlen von 1 bis 6

**print** “i hat jetzt den Wert ” i “ und j ist ” j

Die**for**-Schleife kann auch an Stelle von**to** das Schlüsselwort**downto** beinhalten -

dann zählt die Schleife nach unten statt nach oben.

Beispiel: Die folgende Schleife summiert in der Variablen i alle Werte von 6 bis 1 auf.

i=0

**for** j = 6**downto** 1

i=i+j

Weiterhin kann durch Angabe von**by** eine andere Schrittweite als 1 festgelegt werden.

Beispiel: Die folgende Schleife summiert in der Variablen i alle ungeraden Werte von

1 bis 5 auf, weil auf jeden vorhergehenden Wert 2 addiert wird.

i=0

**for** j = 1**to** 5**by** 2

i=i+j

f)**return**

Die**return** Anweisung übergibt die Kontrolle an die aufrufende Prozedur zurück und

liefert dabei ein oder mehrere Ergebnisse.

Beispiel: Im Folgenden wird an die aufrufende Prozedur der Wert von i als Ergebnis

zurückgegeben.

// i wurde zuvor der Wert einer Berechnung zugewiesen.

**return** i

g)**if**

Mit Hilfe der**if** Anweisung lassen sich Verzweigungen im Code erstellen. Durch das

**if**-statement wird eine Bedingung überprüft - ergibt diese WAHR, so wird die Anwei-

sung im**if**-Block ausgeführt. Falls die Bedingung FALSCH ergibt wird, sofern vorhan-

den, die nächste von beliebig vielen**elseif**-Anweisungen geprüft und so weiter. Falls

keiner der (else)**if** Zweige WAHR ergibt, wird der**else** Zweig ausgeführt, sofern es

einen gibt. Die jeweiligen Blöcke (auch Zweige genannt) werden analog zu den schon

bekannten Schleifenkörpern durch Einrücken gekennzeichnet. Es wird immer der ers-

te Block, bei dem die zugehörige Bedingung WAHR ergibt, ausgeführt - alle darauf

folgenden werden dann ignoriert.

Beispiel: Falls der Wert von i 0 ist, wird eine 0 zurückgegeben. Ist i kleiner als 3, so

wird eine 1 ausgegeben. Allerdings wird dieser Zweig nur erreicht, falls i nicht 0 ist -

ansonsten wurde der**if**-Block darüber bereits gewählt und alle**elseif** Zweige und der

**else** Zweig werden ignoriert. Ist i weder gleich 0 noch kleiner 3, so wird geprüft, ob i

kleiner 5 ist. Ist dies der Fall, so wird eine 2 ausgegeben. Wenn keine der bisher über-

prüften Bedingungen WAHR ergeben hat, so wird im**else** Zweig eine 3 ausgegeben.

Der**else** Zweig hat nie eine Bedingung - er wird immer ausgewählt, wenn keine der

vorhergehenden Bedingungen WAHR ergeben hat. Im Anschluss wird die Zeichen-

kette AuD ausgegeben. Die Ausgabe besteht also in jedem Fall aus**einer** Zahl (0, 1, 2

oder 3) und der Zeichenkette AuD.

**if** i == 0

**print** 0

**elseif** i < 3

**print** 1

**elseif** i < 5

**print** 2

**else**

**print** 3

**print** “AuD”

Man beachte das “==”, das bei Gleichheit der beiden Werte rechts und links davon

WAHR zurückgibt und FALSCH sonst. Dies unterscheidet sich von “=”, bei dem im-

mer eine Zuweisung und kein Vergleich stattﬁndet!

Beispiel:

// Weist i den Wert 3 zu

i=3

// Überprüft, ob i den Wert 3 besitzt

**if** i == 3

h)**i = j = e**

In einer Anweisung kann ein Wert mehreren Variablen zugeordnet werden.

Beispiel: Hier werden i und j gleichzeitig mit dem Wert 5 belegt.

i=j=5

i)**Felder**

Felder beinhalten Werte, auf die mit Hilfe eines Index zugegriffen werden kann. Mit

A[i] wird dabei das i-te Element im Feld bezeichnet, wobei A das Feld und i den Index

repräsentiert. A[1..j] bezeichnet die Elemente A[1], A[2], ..., A[j].

Beispiel: Das Feld P beinhaltet vier Primzahlen. In der Schleife darunter wird auf diese

zugegriffen, so dass im Anschluss damit gearbeitet werden könnte.

// Hier werden dem Feld P Werte zugewiesen.

P[1] = 2

P[2] = 3

P[3] = 5

P[4] = 7

// In dieser Schleife wird nacheinander auf die Elemente des Felds zugegriffen.

**for** i = 1**to** 4

z = P[i]

...

j)**Objekte und Attribute**

Mit <obj>**.**<attr> kann auf ein Attribut <attr> des Objekts <obj> zugegriffen werden.

Man kann also umgangssprachlich Eigenschaften von Objekten auf diese Weise ab-

fragen.

Beispiel: Das Objekt A ist ein Feld mit 5 Elementen. Es besitzt ein Attribut “länge”,

das dementsprechend den Wert 5 hat. Hier wird dieser Wert der Variablen l zugewie-

sen. Das Datenobjekt “schrank” besitzt ein Attribut “farbe”, welches der Variablen f

zugeordnet wird.

l = A.länge

f = schrank.farbe

k)**und** und**oder**

Bei**und** und**oder** handelt es sich um Boolesche Operatoren, die zwei Wahrheitswerte

(WAHR oder FALSCH) miteinander verknüpfen und einen Wahrheitswert zurückge-

ben (siehe Auﬂistung unten). Dabei wird das zweite Argument nur dann ausgewertet,

falls es zur Entscheidung des Rückgabewerts notwendig ist. Dies ist bei**und** nur dann

der Fall, wenn das erste Argument den Wert WAHR einnimmt. Bei**oder** muss analog

der erste Wert FALSCH sein, um das zweite Argument noch auszuwerten.**und** und

**oder** werden auf Grund dieser Eigenschaft**träge** Operatoren genannt.

**und**

WAHR**und** WAHR = WAHR

WAHR**und** FALSCH = FALSCH

FALSCH**und** WAHR = FALSCH

FALSCH**und** FALSCH = FALSCH

**oder**

WAHR**oder** WAHR = WAHR

WAHR**oder** FALSCH = WAHR

FALSCH**oder** WAHR = WAHR

FALSCH**oder** FALSCH = FALSCH

Beispiel: // Der erste Ausdruck bei diesem “und” ist FALSCH,

// daher muss der zweite nicht mehr ausgewertet werden

// b nimmt hier den Wert FALSCH an

b = (1 < 0)**und** (7 == 7)

// Der erste Ausdruck bei diesem “oder” ist FALSCH,

// daher muss der zweite auch ausgewertet werden

// b nimmt hier den Wert WAHR an,

// weil das zweite Argument von**oder** WAHR ist

b = (1 < 0)**oder** (7 == 7)

l)**error**

**error** übergibt im Fehlerfall in einer aufgerufenen Prozedur eine Fehlermeldung an

die aufrufende Prozedur.

Beispiel: Diese Prozedur gibt einen Fehler zurück, falls die Variable i einen negativen

Wert besitzt

**if** i < 0

**error** “Keine negativen Werte erlaubt!”

**Wichtig!**

**Wenn Wertezuweisung einer Variablen mit “=”, dann Vergleichsoperator “==”.**

**Wenn Wertezuweisung mit “:=”, dann kann Vergleichsoperator auch “=” sein.**

**Beispiel:**

**program** **Name** und Kurzbeschreibung

LiesDatenStruktur()

LiesDatenInhalt()

...

**if** DatenUnvollständig **then**

FehlerMelden

exit

**end** **if**

HauptstatistikBerechnen

ZusammenstellungBerechnen

Resultate **in** HTML-Datei schreiben

**end** **program** **Name**

# **Grundelemente des Pseudocode, von Struktogrammen und Programmablaufplänen**

|  |  |  |  |
| --- | --- | --- | --- |
| **Strukturelement** | **Pseudocode** | **Struktogramm** | **Programmablaufplan** |
| **Verarbeitung** (Elementarblock) | Anweisung | [Strukto anw.png](http://www.info-wsf.de/index.php/Datei:Strukto_anw.png) | [Pap anw.png](http://www.info-wsf.de/index.php/Datei:Pap_anw.png) |
| **Reihenfolge** (Sequenz) | Anweisung1 Anweisung2 Anweisung3 | [Strukto reihe.png](http://www.info-wsf.de/index.php/Datei:Strukto_reihe.png) | [Pap reihe.png](http://www.info-wsf.de/index.php/Datei:Pap_reihe.png) |
| **Verzweigung** (Alternative) | wenn Bedingung   dann Anweisung | [Strukto alt1.png](http://www.info-wsf.de/index.php/Datei:Strukto_alt1.png) | [Pap alt1.png](http://www.info-wsf.de/index.php/Datei:Pap_alt1.png) |
| wenn Bedingung   dann Anweisung1   sonst Anweisung2 | [Strukto alt2.png](http://www.info-wsf.de/index.php/Datei:Strukto_alt2.png) | [Pap alt2.png](http://www.info-wsf.de/index.php/Datei:Pap_alt2.png) |
| **Fallunterscheidung** | für Variable   Wert 1 : Anweisung1   Wert 2 : Anweisung2   ...   [sonst Anweisung] | [Strukto fall.png](http://www.info-wsf.de/index.php/Datei:Strukto_fall.png) | [Pap fall.png](http://www.info-wsf.de/index.php/Datei:Pap_fall.png) |
| **Zählschleife** | von Zähler:=Startwert bis Endwert   Anweisung | [Strukto zaehlschl.png](http://www.info-wsf.de/index.php/Datei:Strukto_zaehlschl.png) | [Pap zaehlschl.png](http://www.info-wsf.de/index.php/Datei:Pap_zaehlschl.png) |
| **Strukturelement** | **Pseudocode** | **Struktogramm** | **Programmablaufplan** |
| **anfangsgeprüfte Schleife** | solange Bedingung   Anweisung | [Strukto anfschl.png](http://www.info-wsf.de/index.php/Datei:Strukto_anfschl.png) | [Pap anfschl.png](http://www.info-wsf.de/index.php/Datei:Pap_anfschl.png) |
| **endgeprüfte Schleife** | wiederhole   Anweisung bis Bedingung | [Strukto endschl.png](http://www.info-wsf.de/index.php/Datei:Strukto_endschl.png) | [Pap endschl.png](http://www.info-wsf.de/index.php/Datei:Pap_endschl.png) |