



3 Variablen und Vereinbarungen

3.1 Variablen und Namen

Neben der Verwendung von Konstanten, also der direkten Wertangabe in Form von Zahlen, Zeichen, etc., können in KRL auch Variablen und andere Formen von Daten im Programm benutzt werden.

Bei der Programmierung von Industrierobotern sind Variablen beispielsweise für die Sensorverarbeitung notwendig. Sie erlauben es, den vom Sensor eingelesenen Wert zu speichern und an verschiedenen Stellen im Programm auszuwerten. Außerdem können arithmetische Operationen durchgeführt werden, um etwa eine neue Position zu berechnen.

Eine Variable wird im Programm durch einen Namen dargestellt, wobei die Bezeichnung des Namens in gewissen Grenzen frei wählbar ist.

Namen

Namen in KRL

Sdürfen maximal 12 Zeichen lang sein,

Sdürfen Buchstaben (A-Z), Ziffern (0-9) sowie die Zeichen '_' und '\$' enthalten,

Sdürfen nicht mit Ziffern beginnen,

Sdürfen keine Schlüsselwörter sein.



Da alle Systemvariablen (s. Abschnitt 3.4) mit dem '\$'-Zeichen beginnen, sollten Sie dieses Zeichen nicht als erstes Zeichen in selbstdefinierten Namen verwenden.

Gültige KRL-Namen sind z.B.

SENSOR_1

KLEBEDUESE13

P1_BIS_P12

Eine Variable ist als fester Speicherbereich anzusehen, dessen Inhalt über den Variablennamen ansprechbar ist. Die Variable ist daher zur Laufzeit des Programms durch einen Speicherplatz (Ort) und einen Speicherinhalt (Wert) realisiert.

Wertzuweisung

Mit dem Gleichheitszeichen (=) werden den Variablen Werte zugewiesen. Die Anweisung

ANZAHL = 5

bedeutet also, daß im Speicherbereich mit der Adresse von **ANZAHL** der Wert 5 eingetragen wird. Wie die Adresse genau aussieht, ist für den Programmierer uninteressant, sie wird daher selbständig vom Compiler vergeben. Wichtig ist nur, daß der Speicherinhalt mit Hilfe seines Namens jederzeit im Programm ansprechbar ist.

Da unterschiedliche Datenobjekte (s. Abschnitt 3.2) auch unterschiedlichen Speicherbedarf haben, muß der Datentyp einer Variablen vor der Verwendung vereinbart (deklariert) werden (s. Abschnitt 3.2.1).



Lebensdauer Die Lebensdauer einer Variablen ist die Zeit, während der der Variablen Speicherplatz zugewiesen ist. Sie ist abhängig davon, ob die Variable in einer SRC-Datei oder einer Datenliste deklariert ist:

G Variable in einer SRC-Datei deklariert

Die Lebensdauer beschränkt sich auf die Laufzeit des Programms. Nach Beendigung der Abarbeitung wird der Speicherbereich wieder freigegeben. Der Wert der Variablen geht somit verloren.

G Variable in einer Datenliste (s. Abschnitt 11) deklariert

Die Lebensdauer ist unabhängig von der Laufzeit des Programms. Die Variable existiert so lange, wie die Datenliste existiert. Solche Variablen sind also permanent. Wurde die Variable in der Datenliste auch initialisiert, so wird ihr aktueller Wert ebenfalls permanent erhalten.



NOTIZEN:



3.2 Datenobjekte

Unter Datenobjekten sind benennbare Speichereinheiten eines bestimmten Datentyps zu verstehen. Die Speichereinheiten können dabei aus unterschiedlich vielen Speicherzellen (Bytes, Worte, etc.) bestehen. Wird ein solches Datenobjekt vom Programmierer unter einem Namen vereinbart, erhält man eine Variable. Die Variable belegt nun eine oder mehrere Speicherzellen, in denen Daten durch das Programm geschrieben und gelesen werden können. Durch die symbolische Benennung der Speicherzellen mit frei wählbaren Namen wird die Programmierung einfacher und übersichtlicher, das Programm besser lesbar.

Zur Klärung des Begriffes Datentyp diene folgendes Beispiel: In einer Speicherzelle mit 8 Bit befinde sich die Bitkombination

00110101

Wie ist diese Bitkombination zu interpretieren? Handelt es sich um die binäre Darstellung der Zahl 53 oder um das ASCII-Zeichen "5", was mit dem gleichen Bitmuster dargestellt wird?

Datentyp

Zur eindeutigen Beantwortung dieser Frage fehlt noch eine wichtige Information, nämlich die Angabe des Datentyps eines Datenobjekts. Im obigen Fall könnte das beispielsweise der Typ "ganze Zahl" (INTEGER) oder "Zeichen" (CHARACTER) sein.

Neben diesem computertechnischen Grund für die Einführung von Datentypen, ist die Verwendung von Datentypen auch benutzerfreundlicher, da man mit genau den Typen arbeiten kann, die für die spezielle Anwendung besonders gut geeignet sind.

3.2.1 Vereinbarung und Initialisierung von Datenobjekten

DECL

Die Zuordnung eines Variablennamens zu einem Datentyp und die Reservierung des Speicherplatzes geschieht in KRL mit Hilfe der DECL-Vereinbarung. Mit

DECL INT ANZAHL, NUMMER

vereinbaren Sie z.B. zwei Variablen **ANZAHL** und **NUMMER** vom Datentyp "ganze Zahl" (Integer).

Damit kennt der Compiler diese beiden Variablen sowie den zugehörigen Datentyp und kann bei Benutzung der Variablen überprüfen, ob dieser Datentyp die beabsichtigte Operation überhaupt erlaubt.

Die Deklaration beginnt, wie im Beispiel gezeigt, mit dem Schlüsselwort **DECL**, gefolgt vom Datentyp und der Liste von Variablen, die diesen Datentyp erhalten sollen.



Bei der Deklaration von Variablen und Feldern eines vordefinierten Datentyps kann das Schlüsselwort DECL entfallen. Neben den einfachen Datentypen INT, REAL, CHAR und BOOL (s. Abschnitt 3.2.2) sind unter anderem die Strukturdatentypen POS, E6POS, FRAME, AXIS und E6AXIS (s. Abschnitt 3.2.5) vordefiniert. Für Variablen (keine Felder !) des Datentyps POS kann die Deklaration komplett entfallen. Der Datentyp POS gilt als Standarddatentyp für Variablen. Unverzichtbar ist das Schlüsselwort DECL bei der Vereinbarung frei definierbarer Struktur- oder Aufzählungstypen (s. Abschnitt 3.2.5 und 3.2.6).

Initialisierung

Nach der Vereinbarung einer Variablen ist deren Wert zunächst auf ungültig gesetzt, da er sonst von der zufälligen Speicherbelegung abhängen würde. Um mit der Variablen arbeiten zu können, muß sie daher mit einem bestimmten Wert vorbelegt werden. Diese 1. Wertzuweisung an eine Variable nennt man Initialisierung.



Bei der Erstellung neuer Dateien über den Softkey "Neu" auf der KUKA-Bedienoberfläche wird automatisch auch eine INI-Sequenz erzeugt. Die Vereinbarung von Variablen muß stets vor dieser Sequenz erfolgen.

Eine Wertzuweisung an eine Variable ist eine Anweisung, und darf daher grundsätzlich nicht im Vereinbarungsteil stehen. Die Initialisierung kann aber jederzeit im Anweisungsteil erfolgen. Alle vereinbarten Variablen sollten jedoch zweckmäßigerweise in einem Initialisierungsabschnitt direkt nach dem Deklarationsteil initialisiert werden (s. Abb. 11).

Nur in Datenlisten ist es zulässig, Variablen direkt in der Deklarationszeile zu initialisieren.

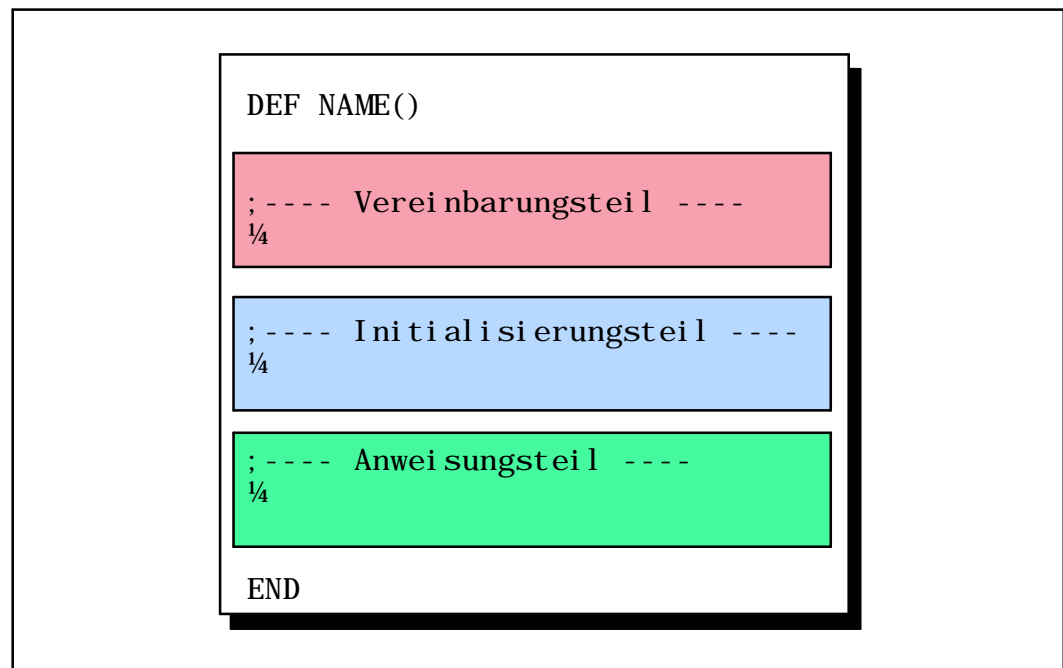
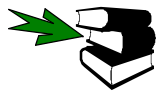
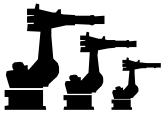


Abb. 11 Grundaufbau eines Roboterprogramms



Abschnitt 11 Datenlisten



3.2.2 Einfache Datentypen

Unter den einfachen Datentypen versteht man einige grundsätzliche Datentypen, die in den meisten Programmiersprachen vorhanden sind. Einfache Datentypen enthalten im Gegensatz zu den strukturierten Datentypen (s. Abschnitt 3.2.3–3.2.6) nur einen einzigen Wert. Die in KRL bekannten Datentypen sind zusammen mit ihrem jeweiligen Wertebereich in Tab. 3 aufgelistet.

Datentyp	Integer	Real	Boolean	Character
Schlüsselwort	INT	REAL	BOOL	CHAR
Bedeutung	ganze Zahl	Gleitkomma-zahl	logischer Zustand	1 Zeichen
Wertebereich	$-2^{31} \frac{1}{4}$ $2^{31} - 1$	$\pm 1.1\text{E}-38 \frac{1}{4}$ $\pm 3.4\text{E}+38$	TRUE, FALSE	ASCII-Zeichen

Tab. 3 Einfache Datentype

INT

Der Datentyp Integer ist eine Teilmenge aus der Menge der ganzen Zahlen. Eine Teilmenge kann es deshalb nur sein, weil kein Rechner die theoretisch unendliche Menge der ganzen Zahlen darstellen kann. Die in der KR C1 für Integer-Typen vorgesehenen 32 Bit ergeben daher 2^{31} ganze Zahlen plus Vorzeichen. Die Zahl 0 wird dabei zu den positiven Zahlen gezählt.

Mit

NUMMER = -23456

wird der Variablen **NUMMER** der Wert **-23456** zugewiesen.

Weisen Sie einer INTEGER-Variablen einen REAL-Wert zu, so wird der Wert nach den allgemeinen Regeln gerundet (x.0 bis x.49 abrunden, x.5 bis x.99 aufrunden). Durch die Anweisung

NUMMER = 45.78

erhält die INTEGER-Variable **NUMMER** den Wert 46.



Ausnahme: Bei Integer-Division wird die Nachkommastelle abgeschnitten, z.B.: $7/4 = 1$

Binärsystem
Hexadezimalsystem

Während der Mensch im Dezimalsystem rechnet und denkt, kennt ein Computer nur Nullen und Einsen, welche die beiden Zustände Aus und Ein repräsentieren. Ein Zustand (Aus oder Ein) wird also mit einem Bit dargestellt. Aus Geschwindigkeitsgründen greift der Rechner im allgemeinen auf ein ganzes Paket solcher Nullen und Einsen zu. Typische Paketgrößen sind 8 Bit (=1 Byte), 16 Bit oder 32 Bit. Bei maschinennahen Operationen ist daher oftmals die Darstellung im Binärsystem (2er-System: Ziffern 0 und 1) oder im Hexadezimalsystem (16er-System: Ziffern 0–9 und A–F) hilfreich (s. Tab. 3.1). Binäre oder hexadezimale Integerwerte können Sie in KRL mit Hilfe des Hochkommas (') und der Angabe von B für Binärdarstellung oder H für Hexadezimaldarstellung angeben.

D	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

Tab. 3.1 Die ersten 17 Zahlen im Dezimal- und Hexadezimalsystem

Die Zahl 90 können Sie also in KRL auf drei verschiedene Arten einer Integervariablen zuweisen:

INTZAHL = 90 ; Dezimal system

INTZAHL = ' B1011010' ; Bi naersystem

INTZAHL = ' H5A' ; Hexadezi mal system



Bin ® Dez Die Umrechnung von Binärzahlen in das Dezimalsystem gestaltet sich wie folgt:

1	0	1	1	0	1	0	$= 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 90$
2^6	2^5	2^4	2^3	2^2	2^1	2^0	

Hex ® Dez Zum Transfer von Zahlen aus dem Hexadezimalsystem in das Dezimalsystem gehen Sie folgendermaßen vor:

5	A	$= 5 \cdot 16^1 + 10 \cdot 16^0 = 90$
16^1	16^0	

REAL

Bei dem Begriff der Gleitkommadarstellung handelt es sich um die Aufteilung einer Zahl in Mantisse und Exponent und deren Darstellung in normierter Form. So wird z.B.

5.3 als 0.5300000 E+01
-100 als -0.1000000 E+03
0.0513 als 0.5130000 E-01

dargestellt.

Beim Rechnen mit Realwerten muß wegen der begrenzten Anzahl der Gleitkommastellen und der damit einhergehenden Ungenauigkeit beachtet werden, daß die gewohnten algebraischen Gesetze nicht mehr in allen Fällen gültig sind. So gilt beispielsweise in der Algebra:

$$\frac{1}{3} \cdot 3 = 1$$

Läßt man dies einen Computer nachrechnen, so kann sich ergeben, daß das Ergebnis nur 0.9999999 E+00 ist. Ein logischer Vergleich dieser Zahl mit der Zahl 1 ergäbe den Wert **FALSE**. Für praktische Anwendungen im Roboterbereich reicht diese Genauigkeit jedoch im allgemeinen aus, wenn man beachtet, daß der logische Test auf Gleichheit bei Realgrößen nur innerhalb eines kleinen Toleranzbereichs sinnvoll durchgeführt werden kann.

Zulässige Zuweisungen an Real-Variablen sind z.B.:

REALZAHL1 = -13.653

REALZAHL2 = 10

REALZAHL3 = 34.56 E-12



Wird einer REAL-Variablen ein INTEGER-Wert zugewiesen, so wird eine automatische Typumwandlung nach REAL vorgenommen. Die Variable REALZAHL2 hat also nach der obigen Zuweisung den Wert 10.0!

BOOL

Die boolschen Variablen dienen zur Beschreibung von logischen Zuständen (z.B. Ein-/Ausgängen). Sie können nur die Werte **TRUE** (wahr) und **FALSE** (falsch) annehmen:

ZUSTAND1 = TRUE

ZUSTAND2 = FALSE

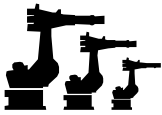
CHAR

Character-Variablen können genau 1 Zeichen aus dem ASCII-Zeichensatz darstellen. Bei der Zuweisung eines ASCII-Zeichens zu einer CHAR-Variablen muß das zugewiesene Zeichen in Anführungszeichen (") eingeschlossen sein:

ZEICHEN1 = "G"

ZEICHEN2 = "?"

Zur Speicherung ganzer Zeichenfolgen siehe Abschnitt 3.2.4.



3.2.3 Felder

Unter Feldern versteht man die Zusammenfassung von Objekten gleichen Datentyps zu einem Datenobjekt, wobei die einzelnen Komponenten eines Feldes über Indizes angesprochen werden können. Durch die Vereinbarung

DECL INT OTTO[7]

Feldindex

können Sie z.B. 7 verschiedene Integer-Zahlen in dem Feld **OTTO[]** ablegen. Auf jede einzelne Komponente des Feldes können Sie durch Angabe des zugehörigen Index zugreifen (erster Index ist immer die 1):

OTTO[1] = 5 ; dem 1. Element wird die Zahl 5 zugewiesen
OTTO[2] = 10 ; dem 2. Element wird die Zahl 10 zugewiesen
OTTO[3] = 15 ; dem 3. Element wird die Zahl 15 zugewiesen
OTTO[4] = 20 ; dem 4. Element wird die Zahl 20 zugewiesen
OTTO[5] = 25 ; dem 5. Element wird die Zahl 25 zugewiesen
OTTO[6] = 30 ; dem 6. Element wird die Zahl 30 zugewiesen
OTTO[7] = 35 ; dem 7. Element wird die Zahl 35 zugewiesen

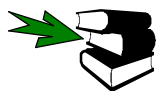
Anschaulich kann man sich das Feld mit dem Namen **OTTO[]** als Regal mit 7 Fächern vorstellen. Die Fächerbelegung würde nach den vorangegangenen Zuweisungen nun so aussehen:

5	10	15	20	25	30	35
---	----	----	----	----	----	----

OTTO [1] [2] [3] [4] [5] [6] [7]

Sollen nun alle Elemente eines Feldes mit der gleichen Zahl, z.B. 0, initialisiert werden, so müssen Sie nicht jede Zuweisung explizit programmieren, sondern Sie können die Vorbesetzung mit Hilfe einer Schleife und einer Zählvariablen "automatisieren":

FOR I = 1 TO 7
OTTO[I] = 0
ENDFOR



Abschnitt 6.2 Schleifen

Die Zählvariable ist in diesem Fall die Integervariable **I**. Sie muß vor der Verwendung als Integer deklariert worden sein.

Zähler

Da solche Zähler beim Programmieren sehr häufig benötigt werden, stellt die KRC1 das vordefinierte Feld **I[]** mit den 10 Zählern **I[1] ¼ I[10]** zur Verfügung.

Der Wert dieser Zähler bleibt nach Programm-Reset, Programm-Abwahl sowie Aus- und Einschalten erhalten.

Mit der Verwendung eines solchen Zählers können Sie das Feld **OTTO[]** also auch folgendermaßen initialisieren:

FOR I[5] = 1 TO 7
OTTO[I[5]] = 0
ENDFOR



- S Der Datentyp eines Feldes ist beliebig. Somit können die einzelnen Elemente wiederum aus zusammengesetzten Datentypen bestehen (z.B.: Feld aus Feldern)**
- S Für die Index sind nur Integer-Datentypen zulässig**
- S Neben Konstanten und Variablen sind auch arithmetische Ausdrücke für den Index zulässig (s. Abschnitt 3.3.1)**
- S Der Index zählt immer ab 1**



2-dimens.

Neben den bisher besprochenen eindimensionalen Feldern, d.h. Feldern mit nur einem Index, können Sie in KRL auch zwei- oder dreidimensionale Felder verwenden. Mit

DECL REAL MATRIX[7, 3]

vereinbaren Sie ein zweidimensionales 7×3 Feld mit $7 \cdot 3 = 21$ REAL-Elementen. Anschaulich läßt sich dieses Feld als Matrix mit 7 Spalten und 3 Reihen darstellen. Mit der Programmsequenz

```
I[3] = 0
FOR SPALTE = 1 TO 7
  FOR REIHE = 1 TO 3
    I[3] = I[3] + 1
    MATRIX[SPALTE, REIHE] = I[3]
  ENDFOR
ENDFOR
```

werden die Elemente der Matrix mit einem Wert entsprechend der Reihenfolge ihrer Belegung besetzt. Man erhält daher folgende Matrixbelegung:

Reihe\Spalte	1	2	3	4	5	6	7
1	1.0	4.0	7.0	10.0	13.0	16.0	19.0
2	2.0	5.0	8.0	11.0	14.0	17.0	20.0
3	3.0	6.0	9.0	12.0	15.0	18.0	21.0

3-dimens.

Dreidimensionale Felder kann man sich schließlich so vorstellen, daß mehrere zweidimensionale Matrizen hintereinander liegen. Die dritte Dimension gibt also sozusagen die Ebene an, in der die Matrix liegt (s. Abb. 12). Vereinbart wird ein dreidimensionales Feld analog zu den ein- oder zweidimensionalen Feldern, z.B.:

DECL BOOL FELD_3D[5, 3, 4]

Die Initialisierungssequenz könnte so aussehen:

```
FOR SPALTE = 1 TO 5
  FOR REIHE = 1 TO 3
    FOR EBENE = 1 TO 4
      FELD_3D[SPALTE, REIHE, EBENE] = FALSE
    ENDFOR
  ENDFOR
ENDFOR
```

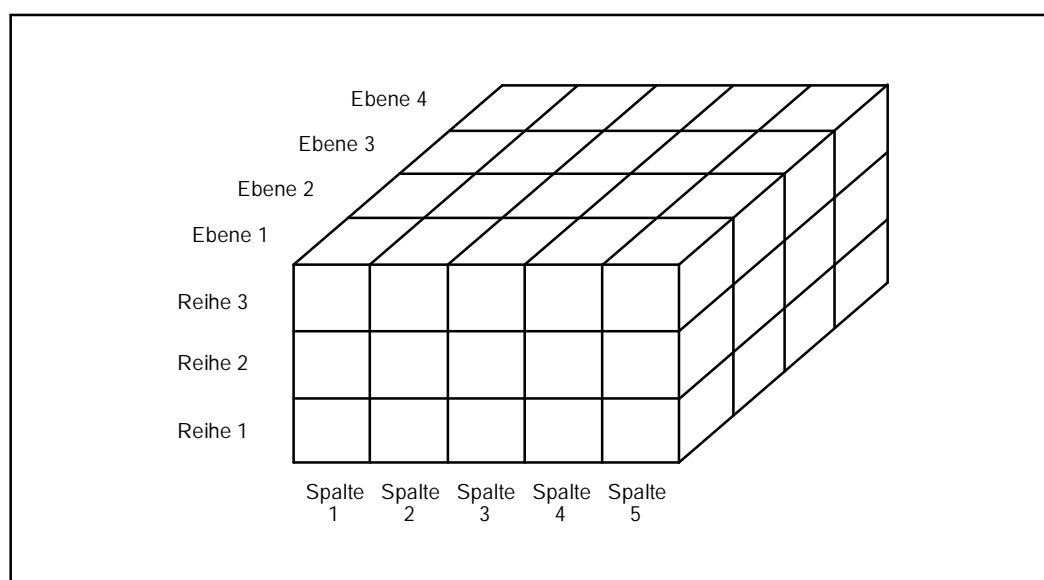


Abb. 12 Darstellung eines dreidimensionalen Feldes



3.2.4 Zeichenketten

Mit dem Datentyp **CHAR** können Sie, wie beschrieben, nur einzelne Zeichen abspeichern. Zur Verwendung ganzer Zeichenketten, also z.B. von Wörtern, definiert man einfach ein ein-dimensionales Feld vom Typ **CHAR**:

DECL CHAR NAME[8]

Sie können jetzt wie bisher üblich jedes einzelne Element des Feldes **NAME[]** ansprechen, z.B.:

NAME[3] = "G"

		G					
--	--	---	--	--	--	--	--

Sie können aber auch gleich ganze Zeichenketten eingeben:

NAME[] = "ABCDEFGG"

belegt die ersten sieben Elemente des Feldes **NAME[]** mit den Buchstaben **A, B, C, D, E, F und G**:

A	B	C	D	E	F	G	
---	---	---	---	---	---	---	--



3.2.5 Strukturen

STRUC

Sollen verschiedene Datentypen zusammengefaßt werden, dann ist das Feld ungeeignet und man muß auf die allgemeinere Form des Verbundes zurückgreifen. Mit der Vereinbarungsanweisung **STRUC** können unterschiedliche Datentypen, die zuvor definiert wurden bzw. vordefinierte Datentypen sind, zu einem neuen Verbunddatentyp zusammengefaßt werden. Insbesondere können auch andere Verbunde und Felder Bestandteil eines Verbundes sein.

Typisches Beispiel für die Verwendung von Verbunden ist der Standarddatentyp **POS**. Er besteht aus 6 **REAL**-Werten und 2 **INT**-Werten und wurde in der Datei **\$OPERATE.SRC** folgendermaßen deklariert:

STRUC POS REAL X, Y, Z, A, B, C, INT S, T

Punkt-Separator

Verwenden Sie jetzt z.B. eine Variable **POSITION** vom Strukturdatentyp **POS** (Variable braucht nicht deklariert zu werden, da **POS** der Standarddatentyp ist), so können Sie die Elemente entweder einzeln mit Hilfe des Punkt-Separators, z.B.:

POSITION.X = 34.4
POSITION.Y = -23.2
POSITION.Z = 100.0
POSITION.A = 90
POSITION.B = 29.5
POSITION.C = 3.5
POSITION.S = 2
POSITION.T = 6

Aggregat

oder gemeinsam mittels eines sogenannten Aggregates

POSITION={X 34.4, Y -23.2, Z 100.0, A 90, B 29.5, C 3.5, S 2, T 6}
besetzen.

Für Aggregate gelten folgende Bestimmungen:



- S** Die Werte eines Aggregats können einfache Konstanten oder selbst Aggregate sein.
- S** In einem Aggregat müssen nicht sämtliche Komponenten der Struktur angegeben werden.
- S** Die Komponenten brauchen nicht in der Reihenfolge angegeben zu werden, in der diese definiert wurden.
- S** Jede Komponente darf in einem Aggregat nur einmal enthalten sein.
- S** Bei Feldern aus Strukturen beschreibt ein Aggregat den Wert eines einzelnen Feldelementes.
- S** Am Anfang eines Aggregates kann - durch Doppelpunkt abgetrennt - der Name des Strukturtyps angegeben sein.

Folgende Zuweisungen sind für **POS**-Variablen also beispielsweise auch zulässig:

POSITION={B 100.0, X 29.5, T 6}
POSITION={A 54.6, B -125.64, C 245.6}
POSITION={POS: X 230, Y 0.0, Z 342.5}

Bei **POS**, **E6POS**, **AXIS**, **E6AXIS** und **FRAME**-Strukturen werden fehlende Komponenten nicht verändert. Bei allen sonstigen Aggregaten werden nicht vorhandene Komponenten auf ungültig gesetzt.

Das Vorgehen beim Erstellen eigener Strukturvariablen sei an folgendem Beispiel erläutert:

In ein Unterprogramm zum Lichtbogenschweißen soll in einer Variablen **S PARA** folgende Information übergeben werden:

REAL	V_DRAHT	Drahtgeschwindigkeit
INT	KENNL	Kennlinie 0¼ 100%
BOOL	LIBO	mit/ohne Lichtbogen (für Simulation)



Die Variable **S_PARA** muß aus 3 Elementen unterschiedlichen Datentyps bestehen. Zunächst muß ein neuer Datentyp definiert werden, der diese Forderungen erfüllt:

```
STRUC SCHWEISSTYP REAL V_DRAHT, INT KENNL, BOOL LIBO
```

Hiermit ist ein neuer Datentyp mit der Bezeichnung **SCHWEISSTYP** entstanden (**SCHWEISSTYP** ist keine Variable!). **SCHWEISSTYP** besteht aus den 3 Komponenten **V_DRAHT**, **KENNL** und **LIBO**. Nun können Sie eine beliebige Variable des neuen Datentyps deklarieren, z.B.:

```
DECL SCHWEISSTYP S_PARA
```

Damit haben Sie eine Variable **S_PARA** des Datentyps **SCHWEISSTYP** geschaffen. Die einzelnen Elemente lassen sich – wie schon beschrieben – mit Hilfe des Punkt-Separators oder des Aggregats ansprechen:

```
S_PARA.V_DRAHT = 10.2
```

```
S_PARA.KENNL = 66
```

```
S_PARA.LIBO = TRUE
```

oder

```
S_PARA = {V_DRAHT 10.2, KENNL 50, LIBO TRUE}
```



Um selbstdefinierte Datentypen von Variablen besser unterscheiden zu können, sollten die Namen der neuen Datentypen mit **¼ TYP** enden.

vordefinierte
Strukturen

In der Datei **\$OPERATE.SRC** sind folgende Strukturen vordefiniert:

```
STRUC AXIS REAL A1, A2, A3, A4, A5, A6
```

```
STRUC E6AXIS REAL A1, A2, A3, A4, A5, A6, E1, E2, E3, E4, E5, E6
```

```
STRUC FRAME REAL X, Y, Z, A, B, C
```

```
STRUC POS REAL X, Y, Z, A, B, C, INT S, T
```

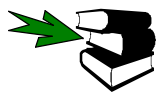
```
STRUC E6POS REAL X, Y, Z, A, B, C, E1, E2, E3, E4, E5, E6, INT S, T
```

Die Komponenten **A1¼ A6** der Struktur **AXIS** sind Winkelwerte (rotatorische Achsen) oder Translationswerte (translatorische Achsen) zum achsspezifischen Verfahren der Roboterachsen 1¼ 6.

Mit den zusätzlichen Komponenten **E1¼ E6** in der Struktur **E6AXIS** können Zusatzachsen angesprochen werden.

In der Struktur **FRAME** können Sie 3 Positionswerte im Raum (**X, Y** und **Z**) sowie 3 Orientierungen im Raum (**A, B** und **C**) festlegen. Ein Punkt im Raum ist damit nach Lage und Orientierung eindeutig definiert.

Da es Roboter gibt, die ein und den selben Punkt im Raum mit mehreren Achsstellungen anfahren können, dienen die Integervariablen **S** und **T** in der Struktur **POS** zum Festlegen einer eindeutigen Achsstellung.

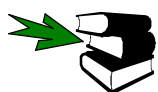


Abschnitt 4.2.2 Status (S) und Turn (T)

geometrische
Datentypen

Mit den Komponenten **E1¼ E6** in der Struktur **E6POS** können wieder Zusatzachsen angesprochen werden.

Die Typen **AXIS**, **E6AXIS**, **POS**, **E6POS** und **FRAME** nennt man auch geometrische Datentypen, da der Programmierer mit ihnen auf einfache Weise geometrische Beziehungen beschreiben kann.



Abschnitt 4.1 Koordinatensysteme



3.2.6 Aufzählungstypen

Ein Aufzählungsdatentyp ist ein Datentyp, der sich aus einer begrenzten Menge von Konstanten zusammensetzt. Die Konstanten sind frei definierbare Namen und können vom Benutzer festgelegt werden. Eine Variable dieses Datentyps (Aufzählungsvariable) kann nur den Wert einer dieser Konstanten annehmen.

Zur Erläuterung diene die Systemvariable **\$MODE_OP**. In ihr wird abgespeichert welche Betriebsarten gerade angewählt ist. Zur Auswahl stehen die Betriebsarten **T1**, **T2**, **AUT** und **EX**.

Man könnte nun **\$MODE_OP** als Integervariable deklarieren, jeder Betriebsart eine Zahl zuordnen und diese dann in **\$MODE_OP** abspeichern. Dies wäre jedoch sehr unübersichtlich.

ENUM

Eine weitaus elegantere Lösung bietet der Aufzählungstyp. In der Datei **\$OPE-RATE.SRC** wurde dazu ein Aufzählungsdatentyp mit dem Namen **MODE_OP** generiert:

```
ENUM MODE_OP T1, T2, AUT, EX, INVALID
```

Der Befehl zur Vereinbarung von Aufzählungstypen lautet also **ENUM** Variablen des Aufzählungstyps **MODE_OP** können nur die Werte **T1**, **T2**, **AUT**, **EX** oder **INVALID** annehmen. Die Variablenvereinbarung erfolgt wieder mit dem Schlüsselwort **DECL**:

```
DECL MODE_OP $MODE_OP
```

Die Aufzählungsvariable **\$MODE_OP** können sie nun durch normale Zuweisung mit einem der vier Werte des Datentyps **MODE_OP** belegen. Zur Unterscheidung von einfachen Konstanten wird den selbstdefinierten Aufzählungskonstanten bei Initialisierungen oder Abfragen ein **"#"**-Zeichen vorangestellt, z.B.:

- Zeichen

```
$MODE_OP = #T1
```

Mit **ENUM** können Sie sich nun beliebig viele selbstdefinierte Aufzählungsdatentypen erzeugen.



NOTIZEN:



3.3 Datenmanipulation

Zur Manipulation der verschiedenen Datenobjekte gibt es eine Fülle von Operatoren und Funktionen, mit deren Hilfe Formeln aufgebaut werden können. Die Mächtigkeit einer Roboterprogrammiersprache hängt gleichermaßen von den zugelassenen Datenobjekten und deren Manipulationsmöglichkeiten ab.

3.3.1 Operatoren

Operand

Unter Operatoren sind die üblichen mathematischen Operatoren zu verstehen im Gegensatz zu Funktionen wie beispielsweise **SIN(30)**, welche den Sinus des Winkels 30° liefert. In der Operation $5+7$ bezeichnet man demnach 5 und 7 als Operanden und + als Operator.

Bei jeder Operation prüft der Compiler die Zulässigkeit der Operanden. So ist beispielsweise $7-3$ als Subtraktion zweier Integerzahlen eine zulässige Operation, $5 + "A"$ als Addition eines Integerwertes zu einem Zeichen eine ungültige Operation.

Bei manchen Operationen, wie $5 + 7.1$, also der Addition von Integer- mit Realwerten wird eine Typanpassung vorgenommen und der Integerwert 5 in den Realwert 5.0 umgewandelt. Auf diese Problematik wird bei der Besprechung der einzelnen Operatoren noch näher eingegangen.

3.3.1.1 Arithmetische Operatoren

Arithmetische Operatoren betreffen die Datentypen INTEGER und REAL. Alle 4 Grundrechenarten sind in KRL zulässig (s. Tab. 4).

Operator	Beschreibung
+	Addition oder positives Vorzeichen
-	Subtraktion oder negatives Vorzeichen
*	Multiplikation
/	Division

Tab. 4 Arithmetische Operatoren

Das Ergebnis einer arithmetischen Operation ist nur dann INT, wenn beide Operanden vom Typ INTEGER sind. Ist das Ergebnis einer Integerdivision nicht ganzzahlig, so wird die Nachkommastelle abgeschnitten. Wenn mindestens einer der beiden Operanden REAL ist, dann ist auch das Ergebnis vom Typ REAL (s. Tab. 5).

Operanden	INT	REAL
INT	INT	REAL
REAL	REAL	REAL

Tab. 5 Ergebnis einer arithmetischen Operation



Zur Verdeutlichung dient folgendes Programmbeispiel:



DEF ARITH()

;----- Deklarationsteil -----

INT A, B, C

REAL K, L, M

;----- Initialisierung -----

; vor der initialisierung sind alle variablen ungueltig!

A = 2 ; A=2

B = 9.8 ; B=10

C = 7/4 ; C=1

K = 3.5 ; K=3.5

L = 0.1 E01 ; L=1.0

M = 3 ; M=3.0

;----- Hauptteil -----

A = A * C ; A=2

B = B - 'HB' ; B=-1

C = C + K ; C=5

K = K * 10 ; K=35.0

L = 10 / 4 ; L=2

L = 10 / 4.0 ; L=2.5

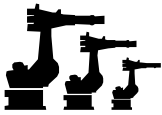
L = 10 / 4. ; L=2.5

L = 10. / 4 ; L=2.5

C = 10. / 4. ; C=3

M = (10/3) * M ; M=9.0

END



3.3.1.2 Geometrischer Operator

Der geometrische Operator wird in KRL durch einen Doppelpunkt ":" symbolisiert. Er führt zwischen den Datentypen **FRAME** und **POS** eine Frameverknüpfung durch.

Die Verknüpfung zweier Frames ist die übliche Transformation von Koordinatensystemen. Daher wirkt sich die Verknüpfung einer **FRAME**- mit einer **POS**-Struktur nur auf das Frame innerhalb der **POS**-Struktur aus. Die Komponenten **S** und **T** bleiben von der Transformation unberührt und müssen daher auch nicht mit einem Wert besetzt sein. Die Werte **X**, **Y**, **Z**, **A**, **B** und **C** müssen jedoch sowohl bei **POS**-Operanden als auch bei **FRAME**-Operanden immer mit einem Wert besetzt sein.

Frame-
verknüpfung

Eine Frameverknüpfung wird von links nach rechts ausgewertet. Das Ergebnis hat immer den Datentyp des am weitesten rechts stehenden Operanden (s. Tab. 6).

linker Operand (Bezugs-KS)	Operator	rechter Operand (Ziel-KS)	Ergebnis
POS	:	POS	POS
POS	:	FRAME	FRAME
FRAME	:	POS	POS
FRAME	:	FRAME	FRAME

Tab. 6 Datentyp-Kombinationen beim geometrischen Operator



Wenn der linke Operand den Datentyp **POS** hat, dann findet eine Typanpassung statt. Die durch die **POS**-Struktur angegebene Position wird in ein Frame umgewandelt. Das heißt, das System ermittelt das Werkzeug-Frame zu dieser Position.

Die Wirkungsweise des geometrischen Operators sei an einem einfachen Beispiel erläutert (s. Abb. 13):

In einem Raum steht ein Tisch. Das **RAUM**-Koordinatensystem sei als festes Koordinatensystem in der linken vorderen Ecke des Raumes definiert.

Der Tisch steht parallel zu den Wänden des Raumes. Die linke vordere Ecke des Tisches liegt genau 600 mm von der vorderen Wand und 450 mm von der linken Wand des Raumes entfernt. Der Tisch ist 800 mm hoch.

Auf dem Tisch steht ein quaderförmiges Werkstück. Das **WERKSTUECK**-Koordinatensystem legen Sie wie in Abb. 13 gezeigt in eine Ecke des Werkstücks. Um das Teil später zweckmäßig handhaben zu können, zeigt die Z-Achse des **WERKSTUECK**-Koordinatensystems nach unten. Das Werkstück ist bezüglich der Z-Achse des **TI SCH**-Koordinatensystems um 40° gedreht. Die Position des **WERKSTUECK**-Koordinatensystems bezogen auf das **TI SCH**-Koordinatensystem ist X=80 mm, Y = 110 mm und Z = 55 mm.

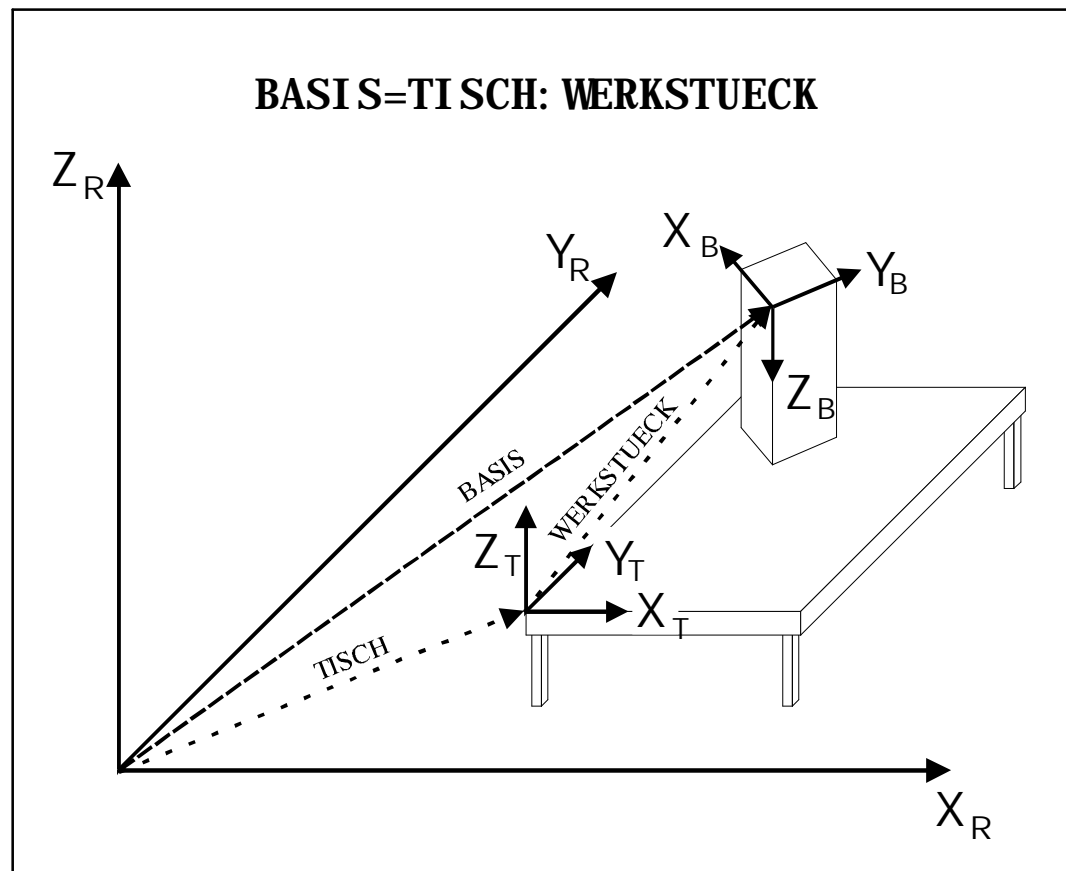


Abb. 13 Wirkungsweise des geometrischen Operators

Die Aufgabenstellung ist nun, das **WERKSTUECK**-Koordinatensystem bezüglich des **RAUM**-Koordinatensystems zu beschreiben. Dazu vereinbaren Sie zunächst folgende Framevariablen:

FRAME TISCH, WERKSTUECK, BASIS

Das **RAUM**-Koordinatensystem sei bereits systemspezifisch festgelegt. Die Koordinatensysteme **TISCH** und **WERKSTUECK** werden nun entsprechend den Randbedingungen initialisiert:

TISCH = {X 450, Y 600, Z 800, A 0, B 0, C 0}

WERKSTUECK = {X 80, Y 110, Z 55, A -40, B 180, C 0}

Das **WERKSTUECK**-Koordinatensystem bezüglich des **RAUM**-Koordinatensystems ergibt sich nun mit Hilfe des geometrischen Operators zu

BASIS = TISCH: WERKSTUECK

In unserem Fall ist **BASIS** nun folgendermaßen besetzt:

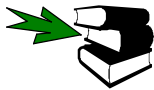
BASIS = {X 530, Y 710, Z 855, A -40, B 180, C 0}



Nur in diesem speziellen Fall ergeben sich die Komponenten von BASIS als Addition der Komponenten von TISCH und WERKSTUECK. Dies liegt daran, daß das TISCH-Koordinatensystem nicht bezüglich des RAUM-Koordinatensystems verdreht ist.

Im allgemeinen ist jedoch eine einfache Addition der Komponenten nicht möglich!

Eine Frameverknüpfung ist auch nicht kommutativ, das heißt, durch Vertauschen von Bezugsframe und Zielframe wird sich normalerweise auch das Ergebnis ändern!



Abschnitt 4.1 Koordinatensysteme

Zur Anwendung des geometrischen Operators ein weiteres Beispiel: Verschiedene Koordinatensysteme und Verknüpfungen von Koordinatensystemen werden darin angefahren. Zur Verdeutlichung von Orientierungsänderungen verfährt die Werkzeugspitze in jedem Koordinatensystem zunächst ein Stück in X-Richtung, dann ein Stück in Y-Richtung und schließlich ein Stück in Z-Richtung.



```
DEF  GEOM_OP ( );----- Deklarationsteil -----
EXT  BAS (BAS_COMMAND :IN, REAL :IN )
DECL AXIS HOME           ;Variable HOME vom Typ AXIS
DECL FRAME MYBASE[2]     ;Feld vom Typ FRAME;----- Initia-
lisierung -----
BAS (#INITMOV, 0 ) ;Initialisierung von Geschwindigkeiten,
                   ;Beschleunigungen, $BASE, $TOOL, etc.
HOME={AXIS: A1 0, A2 -90, A3 90, A4 0, A5 30, A6 0}; Basiskoordinaten-
system setzen
$BASE={X 1000, Y 0, Z 1000, A 0, B 0, C 0} REF_POS_X={X 100, Y 0, Z 0, A
0, B 0, C 0} ;Referenzpos.
REF_POS_Y={X 100, Y 100, Z 0, A 0, B 0, C 0}
REF_POS_Z={X 100, Y 100, Z 100, A 0, B 0, C 0}; eigene Koordinatensy-
steme definieren
MYBASE[1]={X 200, Y 100, Z 0, A 0, B 0, C 180}
MYBASE[2]={X 0, Y 200, Z 250, A 0, B 90, C 0};----- Hauptteil
-----
PTP HOME ; SAK-Fahrt; Bewegung bezueglich des $BASE-Koordina-
tensystems
PTP $NULLFRAME ;Ursprung des $BASE-KS direkt anfahren
WAIT SEC 2 ;2 Sekunden warten
PTP REF_POS_X ;100mm in x-Richtung fahren
PTP REF_POS_Y ;100mm in y-Richtung fahren
PTP REF_POS_Z ;100mm in z-Richtung fahren; Bewegung be-
zueglich des um MYBASE[1] verschobenen $BASE-KS
PTP MYBASE[1]
WAIT SEC 2
PTP MYBASE[1]:REF_POS_X
PTP MYBASE[1]:REF_POS_Y
PTP MYBASE[1]:REF_POS_Z; Bewegung bezueglich des um MYBASE[2]
verschobenen $BASE-KS
PTP MYBASE[2]
WAIT SEC 2
PTP MYBASE[2]:REF_POS_X
PTP MYBASE[2]:REF_POS_Y
PTP MYBASE[2]:REF_POS_Z; Bewegung bez. des um MYBASE[1]:MY-
BASE[2] versch. $BASE-KS
PTP MYBASE[1]:MYBASE[2]
WAIT SEC 2
PTP MYBASE[1]:MYBASE[2]:REF_POS_X
PTP MYBASE[1]:MYBASE[2]:REF_POS_Y
PTP MYBASE[1]:MYBASE[2]:REF_POS_Z; Bewegung bez. des um MY-
BASE[2]:MYBASE[1] versch. $BASE-KS
PTP MYBASE[2]:MYBASE[1]
WAIT SEC 2
PTP MYBASE[2]:MYBASE[1]:REF_POS_X
PTP MYBASE[2]:MYBASE[1]:REF_POS_Y
PTP MYBASE[2]:MYBASE[1]:REF_POS_ZPTP HOME
END
```



3.3.1.3 Vergleichsoperatoren

Mit den in Tab. 7 aufgeführten Vergleichsoperatoren können logische Ausdrücke gebildet werden. Das Ergebnis eines Vergleichs ist daher immer vom Datentyp **BOOL**, da ein Vergleich immer nur wahr (**TRUE**) oder falsch (**FALSE**) sein kann.

Operator	Beschreibung	zulässige Datentypen
==	gleich	INT, REAL, CHAR, ENUM, BOOL
<>	ungleich	INT, REAL, CHAR, ENUM, BOOL
>	größer	INT, REAL, CHAR, ENUM
<	kleiner	INT, REAL, CHAR, ENUM
>=	größer gleich	INT, REAL, CHAR, ENUM
<=	kleiner gleich	INT, REAL, CHAR, ENUM

Tab. 7 Vergleichsoperatoren

Vergleiche können in Programmablaufanweisungen (s. Abschnitt 6) verwendet werden, das Ergebnis eines Vergleichs kann einer booleschen Variablen zugewiesen werden.

Der Test auf Gleichheit oder Ungleichheit ist bei Realwerten nur bedingt sinnvoll, da durch Rundungsfehler bei Berechnung der zu vergleichenden Werte algebraisch identische Formeln ungleiche Werte liefern können (s. 3.2.2).



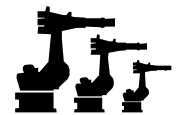
S Kombinationen von Operanden aus INT, REAL, und CHAR sind möglich.
S Ein ENUM-Typ darf nur mit dem selben ENUM-Typ verglichen werden.
S Ein BOOL-Typ darf nur mit einem BOOL-Typ verglichen werden.

Der Vergleich von Zahlenwerten (INT, REAL) und Zeichenwerten (CHAR) ist deshalb möglich, weil jedem ASCII-Zeichen ein ASCII-Code zugeordnet ist. Dieser Code ist eine Zahl, die die Reihenfolge der Zeichen im Zeichensatz bestimmt.

Die einzelnen Konstanten eines Aufzählungstyps werden bei ihrer Deklaration in der Reihenfolge ihres Auftretens durchnummeriert. Die Vergleichsoperatoren beziehen sich auf diese Nummern.

Neben einfachen sind auch mehrfache Vergleiche zulässig. Dazu einige Beispiele:

```
¼
BOOL A, B
¼
B = 10 < 3 ; B=FALSE
A = 10/3 == 3 ; A=TRUE
B = ((B == A) <> (10.00001 >= 10)) == TRUE ; B=TRUE
A = "F" < "Z" ; A=TRUE
¼
```



3.3.1.4 Logische Operatoren

Diese dienen zur logischen Verknüpfung von booleschen Variablen, Konstanten und einfachen logischen Ausdrücken, wie sie mit Hilfe der Vergleichsoperatoren gebildet werden. So hat z.B. der Ausdruck

(A > 5) AND (A < 12)

nur dann den Wert TRUE, wenn A im Bereich zwischen 5 und 12 liegt. Solche Ausdrücke werden häufig in Anweisungen zur Ablaufkontrolle (s. Abschnitt 6) verwendet. Die logischen Operatoren sind in Tab. 8 aufgelistet.

Operator	Operandenzahl	Beschreibung
NOT	1	Invertierung
AND	2	logisches UND
OR	2	logisches ODER
EXOR	2	exklusives ODER

Tab. 8 Logische Operatoren

Die Operanden einer logischen Verknüpfung müssen vom Typ **BOOL** sein, das Ergebnis ist ebenfalls immer vom Typ **BOOL**. In Tab. 9 sind die möglichen Ergebnisse der jeweiligen Verknüpfungen in Abhängigkeit vom Wert der Operanden dargestellt.

Operation		NOT A	A AND B	A OR B	A EXOR B
A = TRUE	B = TRUE	FALSE	TRUE	TRUE	FALSE
A = TRUE	B = FALSE	FALSE	FALSE	TRUE	TRUE
A = FALSE	B = TRUE	TRUE	FALSE	TRUE	TRUE
A = FALSE	B = FALSE	TRUE	FALSE	FALSE	FALSE

Tab. 9 Wahrheitstabelle für logische Verknüpfungen

Einige Beispiele zu logischen Verknüpfungen:

¼

BOOL A, B, C

¼

A = TRUE

B = NOT A

C = (A AND B) OR NOT (B EXOR NOT A)

A = NOT NOT C

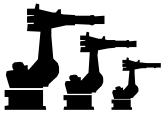
¼

; A=TRUE

; B=FALSE

; C=TRUE

; A=TRUE



3.3.1.5 Bit-Operatoren

Mit Hilfe der Bit-Operatoren (s. Tab. 10) lassen sich ganze Zahlen miteinander verknüpfen, indem man die einzelnen Bits der Zahlen logisch miteinander verknüpft. Die Bit-Operatoren verknüpfen einzelne Bits genauso wie die logischen Operatoren zwei boolsche Werte, wenn man den Bit-Wert 1 als **TRUE** und den Wert 0 als **FALSE** ansieht.

Eine bitweise **UND**-Verknüpfung der Zahlen 5 und 12 ergibt somit z.B. die Zahl 4, eine bitweise **ODER**-Verknüpfung die Zahl 13 und eine bitweise exklusiv **ODER**-Verknüpfung die Zahl 9:

	0	1	0	1	= 5
	1	1	0	0	= 12
AND	0	1	0	0	= 4
OR	1	1	0	1	= 13
EXOR	1	0	0	1	= 9

Bei der bitweisen Invertierung werden nicht alle Bits einfach umgedreht. Stattdessen wird bei Verwendung von **B_NOT** zum Operanden 1 dazu addiert und das Vorzeichen gekippt, z.B:

B_NOT 10 = -11
B_NOT -10 = 9

Bit-Operatoren werden beispielsweise eingesetzt, um digitale Ein-/Ausgangssignale miteinander zu verknüpfen (s. 7.3).

Operator	Operandenzahl	Beschreibung
B_NOT	1	bitweise Invertierung
B_AND	2	bitweise UND-Verknüpfung
B_OR	2	bitweise ODER-Verknüpfung
B_EXOR	2	bitweise exklusive ODER-Verknüpfung

Tab. 10 Logische Bit-Operatoren



Da ASCII-Zeichen auch über den ganzzahligen ASCII-CODE ansprechbar sind, kann der Datentyp der Operanden neben INT auch CHAR sein. Das Ergebnis ist immer vom Typ INT.



Beispiele zur Verwendung von Bit-Operatoren:

¼

INT A

¼

A = 10 B_AND 9	; A=8
A = 10 B_OR 9	; A=11
A = 10 B_EXOR 9	; A=3
A = B_NOT 197	; A=- 198
A = B_NOT 'HC5'	; A=- 198
A = B_NOT 'B11000101'	; A=- 198
A = B_NOT "E"	; A=- 70

¼

Angenommen, Sie haben einen 8-Bit breiten digitalen Ausgang definiert. Sie können den Ausgang über die INTEGER-Variable **DIG** ansprechen. Zum Setzen der Bits 0, 2, 3 und 7 können Sie nun einfach

Einblenden
von Bits

DIG = 'B10001101' B_OR DIG

programmieren. Alle anderen Bits bleiben unbeeinflusst, egal welchen Wert sie haben.

Wollen Sie z.B. die Bits 1, 2 und 6 ausblenden, so programmieren Sie

Ausblenden
von Bits

DIG = 'B10111001' B_AND DIG

Alle anderen Bits werden dadurch nicht verändert.

Genauso leicht können Sie mit den Bit-Operatoren überprüfen, ob einzelne Bits des Ausgangs gesetzt sind. Der Ausdruck

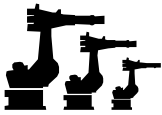
Herausfiltern
von Bits

('B10000001' B_AND DIG) == 'B10000001'

wird **TRUE**, wenn die Bits 0 und 7 gesetzt sind, ansonsten wird er **FALSE**.

Wollen Sie nur Testen, ob wenigstens eines der beiden Bits 0 oder 7 gesetzt ist, so muß die bitweise UND-Verknüpfung lediglich größer Null sein:

('B10000001' B_AND DIG) > 0



3.3.1.6 Prioritäten von Operatoren

Priorität

Verwenden Sie komplexere Ausdrücke mit mehreren Operatoren, so müssen Sie die unterschiedlichen Prioritäten der einzelnen Operatoren beachten (s. Tab. 11), da die einzelnen Ausdrücke in der Reihenfolge ihrer Prioritäten ausgeführt werden.

Priorität	Operator
1	NOT B_NOT
2	* /
3	+ -
4	AND B_AND
5	EXOR B_EXOR
6	OR B_OR
7	== <> < > >= <=

Tab. 11 Prioritäten von Operatoren

Grundsätzlich gilt:

- G Geklammerte Ausdrücke werden zuerst bearbeitet.
- G Bei ungeklammerten Ausdrücken wird in der Reihenfolge der Priorität ausgewertet.
- G Verknüpfungen mit Operatoren gleicher Priorität werden von links nach rechts ausgeführt.

Beispiele:

¼

INT A, B

BOOL E, F

¼

A = 4

B = 7

E = TRUE

F = FALSE

¼

E = NOT E OR F AND NOT (-3 + A * 2 > B) ; E=FALSE

A = 4 + 5 * 3 - B_NOT B / 2 ; A=23

B = 7 B_EXOR 3 B_OR 4 B_EXOR 3 B_AND 5 ; B=5

F = TRUE == (5 >= B) AND NOT F ; F=TRUE

¼



3.3.2 Standardfunktionen

Zur Berechnung gewisser mathematischer Probleme sind in KRL einige Standardfunktionen vordefiniert (s. Tab. 12). Sie können ohne weitere Vereinbarung direkt benutzt werden.

Beschreibung	Funktion	Datentyp Argument	Wertebereich Argument	Datentyp Funktion	Wertebereich Ergebnis
Betrag	ABS(X)	REAL	$-8\frac{1}{4} + 8$	REAL	$0\frac{1}{4} + 8$
Wurzel	SQRT(X)	REAL	$0\frac{1}{4} + 8$	REAL	$0\frac{1}{4} + 8$
Sinus	SIN(X)	REAL	$-8\frac{1}{4} + 8$	REAL	$-1\frac{1}{4} + 1$
Cosinus	COS(X)	REAL	$-8\frac{1}{4} + 8$	REAL	$-1\frac{1}{4} + 1$
Tangens	TAN(X)	REAL	$-8\frac{1}{4} + 8^*$	REAL	$-8\frac{1}{4} + 8$
Arcuscos.	ACOS(x)	REAL	$-1\frac{1}{4} + 1$	REAL	$0\frac{1}{4} 180\frac{1}{4}$
Arcustang.	ATAN2(Y, X)	REAL	$-8\frac{1}{4} + 8$	REAL	$-90\frac{1}{4} + 90\frac{1}{4}$
* keine ungeradzahlig Vielfache von $90\frac{1}{4}$ d.h. $X \neq (2k-1) * 90\frac{1}{4} \text{ k} \in \mathbb{Z}$					

Tab. 12 Mathematische Standardfunktionen

Betrag	Die Funktion ABS(X) berechnet den Betrag des Wertes X, z.B.: B = -3.4 A = 5 * ABS(B) ; A=17.0
Wurzel	SQRT(X) errechnet die Quadratwurzel aus der Zahl X, z.B.: A = SQRT(16.0801) ; A=4.01
Sinus Cosinus Tangens	Die trigonometrischen Funktionen SIN(X), COS(X) und TAN(X) berechnen den Sinus, Cosinus oder Tangens des Winkels X, z.B.: A = SIN(30) ; A=0.5 B = 2 * COS(45) ; B=1.41421356 C = TAN(45) ; C=1.0 Der Tangens von $\pm 90\frac{1}{4}$ und ungeradzahlig Vielfachen von $\pm 90\frac{1}{4}$ ($\pm 270\frac{1}{4}$, $\pm 450\frac{1}{4}$, $\pm 630\frac{1}{4}$), ist unendlich. Deshalb führt der Versuch der Berechnung eines dieser Werte zu einer Fehlermeldung.
Arcuscosinus	ACOS(X) ist die Umkehrfunktion zu COS(X): A = COS(60) ; A=0.5 B = ACOS(A) ; B=60
Arcussinus	Für den Arcussinus, die Umkehrfunktion zu SIN(X), ist keine Standardfunktion vordefiniert. Aufgrund der Beziehung SIN(X) = COS(90-X) können sie aber auch diesen sehr leicht berechnen: A = SIN(60) ; A=0.8660254 B = 90 - ACOS(A) ; B=60



Arcustangens

Der Tangens eines Winkels ist definiert als Gegenkathete (**Y**) dividiert durch Ankathete (**X**) im rechtwinkligen Dreieck. Hat man die Länge der beiden Katheten, kann man also den Winkel zwischen Ankathete und Hypotenuse mit dem Arcustangens berechnen.

Betrachtet man jetzt einen Vollkreis, so ist es entscheidend, welches Vorzeichen die Komponenten **X** und **Y** haben. Würde man nur den Quotienten berücksichtigen, so könnten mit dem Arcustangens nur Winkel zwischen 0__und 180__berechnet werden. Dies ist auch bei allen üblichen Taschenrechnern der Fall: Der Arcustangens von positiven Werten ergibt einen Winkel zwischen 0__und 90__, der Arcustangens von negativen Werten einen Winkel zwischen 90__und 180__.

Durch die explizite Angabe von **X** und **Y** ist durch deren Vorzeichen eindeutig der Quadrant festgelegt, in dem der Winkel liegt (s. Abb. 14). Sie können daher auch Winkel in den Quadranten III und IV berechnen. Deshalb sind zur Berechnung des Arcustangens in der Funktion **ATAN2(Y, X)** auch diese beiden Angaben notwendig, z.B.:

A = ATAN2(0.5, 0.5)	; A=45
B = ATAN2(0.5, -0.5)	; B=135
C = ATAN2(-0.5, -0.5)	; C=225
D = ATAN2(-0.5, 0.5)	; D=315

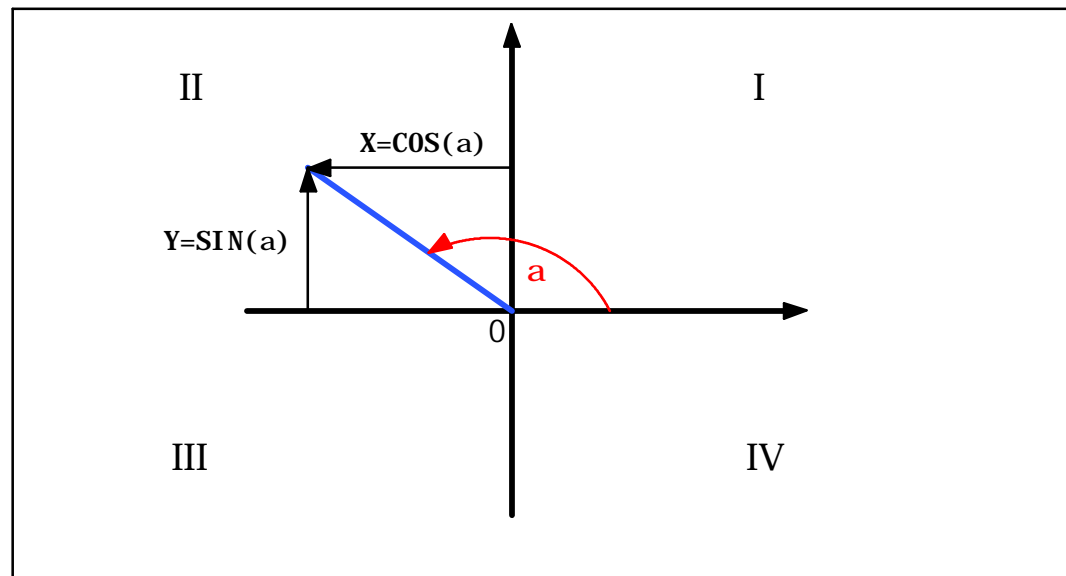
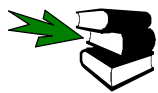


Abb. 14 Verwendung von **X** und **Y** in der Funktion **ATAN2(Y, X)**



Abschnitt 8 zum Definieren von eigenen Funktionen



NOTIZEN:



3.4 Systemvariablen und Systemdateien

Eine wichtige Voraussetzung für die Bearbeitung von komplexen Anwendungen in der Robotertechnik ist eine frei und komfortabel programmierbare Steuerung.

Dazu muß in einfacher Weise die Funktionalität der Robotersteuerung in der Robotersprache programmierbar sein. Erst wenn die Integration der Steuerungsparameter in ein Roboterprogramm vollständig und doch einfach möglich ist, können Sie die volle Funktionalität einer Robotersteuerung ausnützen. Bei der KR C1 ist dies durch das Konzept der vordefinierten Systemvariablen und -dateien hervorragend gelöst.

Beispiele für vordefinierte Variablen sind **\$POS_ACT** (aktuelle Roboterposition), **\$BASE** (Basiskoordinatensystem) oder **\$VEL_CP** (Bahngeschwindigkeit). Eine genaue Beschreibung sowie eine Auflistung aller vordefinierter Variablen finden Sie im Schulungshandbuch.

Systemvariablen sind vollständig in das Variablenkonzept von KRL integriert. Sie besitzen einen entsprechenden Datentyp und können von Ihnen wie jede andere Variable im Programm gelesen und geschrieben werden, wenn es nicht Einschränkungen wegen der Art der Daten gibt. Die aktuelle Roboterposition kann z.B. nur gelesen, nicht geschrieben werden. Solche Einschränkungen werden von der Steuerung geprüft.

Soweit es sicherheitstechnisch möglich ist, verfügen Sie sogar über den schreibenden Zugriff auf Systemdaten. Damit eröffnen sich viele Möglichkeiten für die Diagnose, da vom KCP und vom Programmiersystem eine Vielzahl von Systemdaten geladen oder beeinflußt werden können.

Nützliche Systemvariablen mit Schreibzugriff sind zum Beispiel **\$TIMER[]** und **\$FLAG[]**.

Timer

Die 16 Timervariablen **\$TIMER[1]** ¼ **\$TIMER[16]** dienen zum Messen von zeitlichen Abläufen und können somit als "Stoppuhr" eingesetzt werden. Das Starten und Stoppen des Meßvorganges erfolgt mit den Systemvariablen **\$TI - MER_STOP[1]** ¼ **\$TIMER_STOP[16]**:

\$TIMER_STOP[4] = FALSE

startet beispielsweise den Timer 4,

\$TIMER_STOP[4] = TRUE

stoppt den Timer 4 wieder. Über eine normale Wertzuweisung kann die betreffende Timervariable jederzeit zurückgesetzt werden, z.B.:

\$TIMER[4] = 0

Wechselt der Wert einer Timervariablen von Minus nach Plus, so wird ein zugehöriges Flag auf **TRUE** gesetzt (Timer-Out-Bedingung), z.B.:

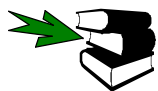
\$TIMER_FLAG[4] = TRUE

Beim Steuerungshochlauf werden alle Timervariablen mit 0, die Flags **\$TI - MER_FLAG[1]** ¼ **\$TIMER_FLAG[16]** mit **FALSE** und die Variablen **\$TI - MER_STOP[1]** ¼ **\$TIMER_STOP[16]** mit **TRUE** vorbesetzt.

Die Einheit der Timervariablen ist Millisekunden (ms). Die Aktualisierung von **\$TI - MER[1]** ¼ **\$TIMER[16]** sowie **\$TIMER_FLAG[1]** ¼ **\$TIMER_FLAG[16]** erfolgt im 12ms-Takt.



Flags	Die 1024 Flags \$FLAG[1] ¼ \$FLAG[1024] werden als globale Merker eingesetzt. Diese boolschen Variablen werden mit FALSE vorbesetzt. Sie können sich den aktuellen Wert der Flags jederzeit auf der Bedienoberfläche unter dem Menüpunkt "Anzeige" ansehen.
Zyklische Flags	<p>Des weiteren stehen in der KR C1 32 zyklische Flags \$CYCFLAG[1] ¼ \$CYCFLAG[32] zur Verfügung. Nach dem Steuerungshochlauf sind sie alle mit FALSE vorbelegt.</p> <p>Die Flags sind nur auf der Roboterebene zyklisch aktiv. In einer Submit-Datei sind die zyklischen Flags zwar zulässig, es erfolgt aber keine zyklische Auswertung.</p> <p>Zyklische Flags können auch in Unterprogrammen, Funktionen und Interrupt-Unterprogrammen definiert und aktiviert werden.</p> <p>\$CYCFLAG[1] ¼ \$CYCFLAG[32] haben den Datentyp BOOL. Bei einer Zuweisung an ein zyklisches Flag kann ein beliebiger boolscher Ausdruck verwendet werden.</p> <p>Zulässig sind</p> <ul style="list-style-type: none">G boolsche Systemvariablen undG boolsche Variablen, welche in einer Datenliste deklariert und initialisiert wurden. <p>Nicht zulässig sind</p> <ul style="list-style-type: none">G Funktionen, die einen boolschen Wert zurückliefern. <p>Die Anweisung</p> <p>\$CYCFLAG[10] = \$IN[2] AND \$IN[13]</p> <p>bewirkt beispielsweise, daß der boolsche Ausdruck "\$IN[2] AND \$IN[13]" zyklisch ausgewertet wird. Das heißt, sobald sich Eingang 2 oder Eingang 13 ändert, ändert sich auch \$CYCFLAG[10], egal an welcher Stelle sich der Programmlaufzeiger nach Abarbeitung des obigen Ausdrucks befindet.</p> <p>Alle definierten zyklischen Flags bleiben solange gültig, bis ein Modul abgewählt oder mit Reset eine Satzanwahl durchgeführt wird. Wird das Programmende erreicht, bleiben alle zyklischen Flags weiterhin aktiv.</p>



Abschnitt 9.5 zur Verwendung zyklischer Flags

\$ - Zeichen

Allgemein sind die Namen der vordefinierten Variablen so gewählt, daß man sie sich leicht einprägen kann. Sie beginnen alle mit einem \$-Zeichen und bestehen dann aus einer sinnvollen englischen Abkürzung. Da sie wie gewöhnliche Variablen behandelt werden, müssen Sie sich keine außergewöhnlichen Befehle oder ausgefallene Optionen merken.

Um Verwechslungen zu vermeiden, sollten Sie selbst keine Variablen vereinbaren, die mit einem \$-Zeichen beginnen.

Ein Teil der vordefinierten Variablen bezieht sich auf die gesamte Steuerung KR C1 (z.B. **\$ALARM_STOP** für die Definition des Ausgangs für das Not-Aus-Signal zur SPS). Andere sind dagegen nur für den Roboter von Bedeutung (z.B. **\$BASE** für das Basiskoordinatensystem).



Die steuerungsrelevanten Daten sind auf der Festplatte im Verzeichnis "Steu" abgelegt, die roboterrelevanten Daten im Verzeichnis "R1". Auf der KUKA-Bof erscheint das Steuerungssystem als "/" und das Robotersystem als Unterverzeichnis des Steuerungssystems mit dem Namen "R1" (s. Abb. 15).

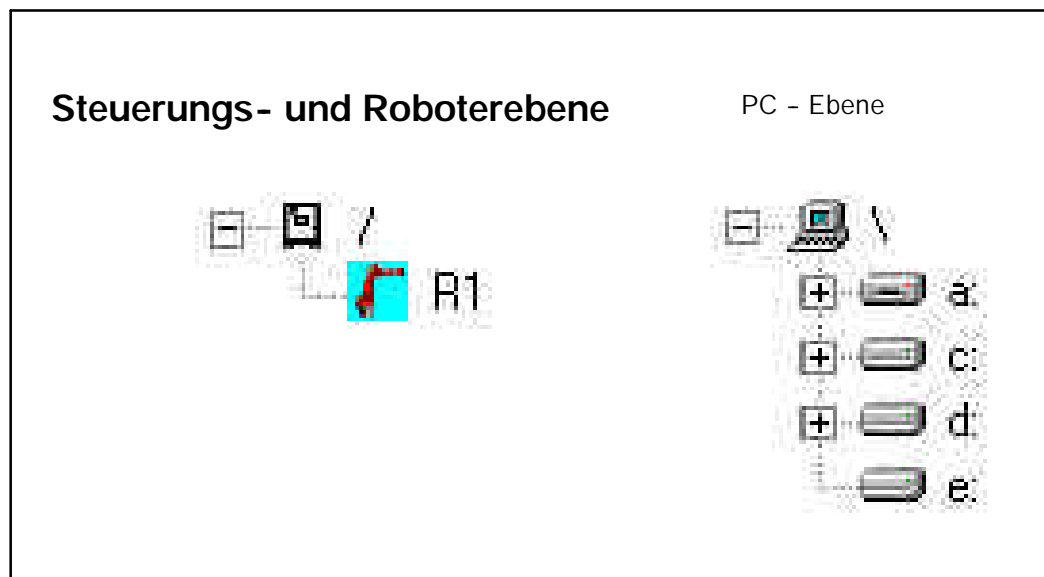
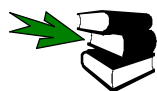


Abb. 15 Verschiedene Ebenen auf der KUKA-Bedienoberfläche

Bei der Programmierung der KR C1 können Sie Programmdateien und Datenlisten erstellen. In den Programmdateien stehen Datendefinitionen und ausführbare Anweisungen, die Datenlisten enthalten nur Datendefinitionen und eventuell Initialisierungen.



Abschnitt 11 Datenlisten

Neben den Datenlisten, die Sie bei der Programmierung erstellen, gibt es auf der KR C1 noch Datenlisten, die von KUKA definiert sind und die mit der Steuerungssoftware ausgeliefert werden. Diese Datenlisten heißen vordefinierte Datenlisten und beinhalten hauptsächlich die vordefinierten Variablen.

Die vordefinierten Datenlisten können Sie weder löschen noch selbst erzeugen. Sie werden bei der Installation der Software erzeugt und sind dann immer vorhanden. Auch die Namen der vordefinierten Datenlisten beginnen wie die Namen der vordefinierten Daten mit einem \$-Zeichen.

Auf der KR C1 gibt es folgende vordefinierte Datenlisten:

- G \$OPERATE. SRC**
bildet aus Implementierungsgründen die einzige Ausnahme. Hier handelt es sich nämlich um eine vordefinierte Programmdatei und nicht um eine vordefinierte Datenliste. Programmdateien sind in KRL programmierte Einheiten (s. Abschnitt 2). In ihnen sind, wie in Datenlisten, auch Variablen definiert. Da in **\$OPERATE. SRC** nur vordefinierte Variablen enthalten sind, ist der Unterschied hier unerheblich. **\$OPERATE. SRC** gibt es sowohl im Steuerungssystem als auch in jedem Robotersystem. Steuerungsspezifische Daten sind vor allem die aktuellen Einstellungen für Bediengeräte und Kommandoumgebung, während als roboterspezifische Daten vor allem Programm- und Roboterzustandsdaten vorkommen. Sie können keine neuen Variablen erstellen oder vorhandene Variablen löschen.

Beispiele:

\$HOME

Voreinstellung der Umgebung (steuerungsspezifisch)

\$BASE

Basiskoordinatensystem (roboterspezifisch)

- G \$MACHINE. DAT**

ist eine vordefinierte Datenliste mit ausschließlich vordefinierten Systemvariablen. Mit



den Maschinendaten wird die Steuerung an den angeschlossenen Roboter angepaßt (Takte, Kinematik, Regelparameter, etc.). Es gibt sowohl eine **\$MACHINE.DAT** auf dem Steuerungssystem als auch im Robotersystem. Sie können keine neuen Variablen erstellen oder vorhandene Variablen löschen.

Beispiele:

\$ALARM_STOP

Signal für Not-Aus (steuerungsspezifisch)

\$NUM_AX

Anzahl der Roboterachsen (roboterspezifisch)

G \$CUSTOM.DAT

ist eine Datenliste, die es nur auf dem Steuerungssystem gibt. Sie beinhaltet Daten, mit denen Sie bestimmte Steuerungsfunktionen projektieren oder parametrieren können. Für den Programmierer besteht nur die Möglichkeit, die Werte der vordefinierten Variablen zu ändern. Sie können keine neuen Variablen erstellen oder vorhandene Variablen löschen.

Beispiele:

\$PSER_1

Protokollparameter der seriellen Schnittstelle 1

\$IBUS_ON

Einschalten alternativer Interbusgruppen

G \$CONFIG.DAT

ist eine von KUKA vordefinierte Datenliste, die aber keine vordefinierten Systemvariablen enthält. Dabei existiert eine **\$CONFIG.DAT** auf Steuerungsebene und auf Robotersebene. Es können darin Variablen, Strukturen, Kanäle und Signale definiert werden, die längere Zeit gültig sind und für viele Programme von übergeordneter Bedeutung sind.

Die Datenliste ist in folgende Blöcke untergliedert:

- BAS
- AUTOEXT
- GRIPPER
- PERCEPT
- SPOT
- A10
- A20
- TOUCHSENSE
- USER

G Globale Deklarationen des Anwenders sollten unbedingt in den USER-Block eingetragen werden. Denn nur hier werden die Vereinbarungen bei einem späteren Software-Update übernommen.

G \$ROBCOR.DAT

Die Datei \$ROBCOR.DAT enthält roboterspezifische Daten für das Dynamikmodell des Roboters. Diese Daten werden für die Bahnplanung benötigt. Auch hier können Sie keine neuen Variablen erstellen oder vorhandene Variablen löschen.

Tab. 13 gibt eine Zusammenstellung der vordefinierten Datenlisten.



Datenliste	System		Wertzuweisung	
	Steuerung	Roboter	bei	durch
\$OPERATE. SRC	n	n	Lieferung	KUKA
\$MACHINE. DAT	n	n	Inbetriebnahme	KUKA/Anwender
\$CUSTOM. DAT	n		Inbetriebnahme	Anwender/KUKA
\$CONFIG. DAT	n	n	Zellenauf- oder -umbau	Anwender/KUKA
\$ROBCOR. DAT	n		Lieferung	KUKA

Tab. 13 Vordefinierte Datenlisten auf der KR C1



NOTIZEN: