

eigerScript®

Evaluation mathematischer Ausdrücke

Konstante Ausdrücke

Rein konstante Ausdrücke werden zur Kompilierzeit durch den Compiler ausgewertet. Sie ergeben immer einen konstanten Wert im Datentyp, dem er zugewiesen wird.

Grundrechenarten

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
inc()	Inkrement um 1
dec()	Dekrement um 1

Stringoperationen

&	String-Concat (nur für Strings definiert!)
---	--

Transzendente und Exponentialfunktionen (reellwertig)

^	Potenzfunktion
sin()	Sinus
cos()	Cosinus
tan()	Tangens
arcsin()	Arcussinus
arccos()	Arcuscosinus
arctan()	Arcustangens
sqrt()	Quadratwurzel
sqr()	Quadratfunktion
exp()	Exponentialfunktion e^x
log()	Logarithmus zur Basis 10
ln()	natürlicher Logarithmus
pi()	Kreiszahl Pi (ohne Argumente!); entspricht Push 3.1415926535897932

Umwandlungsfunktionen

abs()	Absolutwert (negative Zahlen werden mit -1 multipliziert)
sgn()	Vorzeichen einer reellen Zahl (-1 für negative Zahlen, +1 für positive und 0 für Null)
round()	Runden auf die nächste Ganzzahl

Logische Operationen (bitweise)

and	bitweises And (für Ganzzahlentypen, NICHT boolean!)
or	bitweises Or (für Ganzzahlentypen, NICHT boolean!)
xor	bitweises Xor (für Ganzzahlentypen, NICHT boolean!)
not	bitweises Not (für Ganzzahlentypen, NICHT boolean!)



Zur Laufzeit auszuwertende Ausdrücke

Nicht konstant evaluierbare Ausdrücke werden durch den Compiler in eine Stack-Befehlskette umgesetzt, die vom Zielrechner ausgewertet wird. Der Datentyp ergibt sich grundsätzlich aus dem Zieldatentyp. Das bedeutet, dass Parameter durch den Aufruf einer Type-Cast-Funktion wenn notwendig in den Zieldatentyp gewandelt werden müssen. Wichtig zu wissen ist dabei, dass jeder Ganzzahl-Datentyp intern mit 32Bit-Signed-Integer (LONG) ausgewertet wird. Erst vor der Zuweisung wird das 32bit-Ergebnis in den Zieldatentyp gewandelt. Dies geschieht automatisch durch den Stack-Befehlsinterpreter des Zielsystems.

Die unten aufgeführten Symbole werden durch den Compiler erkannt und als Stackketten-Befehl synthetisiert. Es ist dadurch möglich, dass einige der Befehle nicht in ein ausführbares Programm übersetzt werden können, weil der Befehlsvorrat der Zielmaschine diesen nicht kennt. Der Compiler meldet einen Fehler (Stack command not found). Zum Beispiel ist der Befehl sgn() nicht implementiert:

```
Line 240 : (Error) stack command eFA_SGN_S not found.
Line 240 : Error: Assignment a <- 3*sgn(w) can't be calculated.
```

Grundrechenarten

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
inc()	Inkrement um 1
dec()	Dekrement um 1

Stringoperationen

&	String-Concat (nur für Strings definiert!)
---	--

Transzendente und Exponentialfunktionen (reellwertig)

^	Potenzfunktion
sin()	Sinus
cos()	Cosinus
tan()	Tangens
arcsin()	Arcussinus
arccos()	Arcuscosinus
arctan()	Arcustangens
sqrt()	Quadratwurzel
sqr()	Quadratfunktion
exp()	Exponentialfunktion e^x
log()	Logarithmus zur Basis 10
ln()	natürlicher Logarithmus
pi()	Kreiszahl Pi (ohne Argumente!); entspricht Push 3.1415926535897932

Umwandlungsfunktionen

abs()	Absolutwert (negative Zahlen werden mit -1 multipliziert)
sgn()	Vorzeichen einer reellen Zahl (-1 für negative Zahlen, +1 für positive und 0 für Null)



Logische Operationen (bitweise)

and	bitweises And (für Ganzzahlentypen, NICHT boolean!)
or	bitweises Or (für Ganzzahlentypen, NICHT boolean!)
xor	bitweises Xor (für Ganzzahlentypen, NICHT boolean!)
not	bitweises Not (für Ganzzahlentypen, NICHT boolean!)

