

Was ist netbasic?

netbasic ist eine Programmiersprache für einfache Steuerungsanwendungen. Es ist auch eine sehr einfache. Es besteht aus einer Teilmenge von Basic mit hardware-spezifischen Erweiterungen. Es orientiert sich am MCS Tiny Basic (im Intel 8052AH) sowie am Dartmouth-Basic von 1964. Also „old fashion“ Basic. Der in netbasic geschriebene Code wird interpretiert und ist daher nicht für Anwendungen geeignet, die hohe Geschwindigkeit erfordern. Typische Anwendungen sind einfache Steuerungen oder Regelungen zum Beispiel Alarmsysteme, Anzeigeeinheiten für Messwerte, Schaltuhren, Thermostate oder ähnl. Mit netbasic ist es möglich, mit wenigen Programmierschritten eine einfache Anwendung zu erzeugen, ohne sich in umfangreiche Programmiersprachen einarbeiten zu müssen.

netbasic wurde von einem Praktiker für kleine praktische Hobby-Projekte geschrieben, es fehlt jeder theoretische Überbau.

Die Hardware, die von netbasic unterstützt wird, ist das AVR-NET-IO-Board von Pollin Artikel-Nr.: 810 058 bzw. 810 073 (Fertiggerät bzw. Bausatz). Zusätzlich benötigt wird für die Steuerausgänge und -eingänge der Bausatz PC-Relaiskarte K8IO Artikel-Nr.: 710 722. Wer mag, kann auch noch ein LCD mit 4x20 Zeichen anschliessen.

Der Programmcode von netbasic ist sehr kompakt. Der Programmspeicher fasst ca 4 KByte. Das reicht für kleine Projekte (ca 200 Programmzeilen). Eine Zugangskontrolleinheit mit Tastencode ist mit 20 Programmzeilen ruck-zuck erstellt.

Der Original-Prozessor des Boards muss gegen einen ATMEGA1284P ausgetauscht werden, den man mit dem netbasic „gebrannt“ hat. Das Board ist zugleich auch Entwicklungsboard, sodass dies auch auf dem Board passieren kann. Dafür wird ein ISP-Programmer benötigt.

Die Basic-Programmierung wird wie bei MCS51 Basic mit einem Terminalprogramm durchgeführt. Prinzipiell kann sowohl via serieller Schnittstelle als auch via Netzwerk programmiert werden.

Es gibt zwei Betriebsmodi: Nach dem Einschalten befindet sich der Basic-Computer im *Direktmodus* = Anweisungseingabemodus. Hier können Anweisungen wie "print time" (wird sofort ausgeführt) oder auch Programmzeilen (beginnen mit einer Zeilennummer) wie "**10** let B = sin(A)" (wird im RAM gespeichert) eingegeben werden. Mit der Anweisung "run" wird das Programm im RAM gestartet und der Basic-Computer wechselt in den *Programm-Modus*. Es gibt Anweisungen, die dürfen nur im Anweisungsmodus und andere nur im Programm-Modus verwendet werden. Viele Anweisungen arbeiten in beiden Modi. Zur Eingabe verfügt netbasic über einen rudimentären "Zeileneditor" wie MCS51 Basic. - zusätzlich: Cursor links und rechts, sowie Überschreiben – auf Terminals mit VT100 Unterstützung (z.B. TKTerm oder TeraTerm) auch Einfügemodus (mit Strg + e einschalten).

Das fertige Programm kann dauerhaft im EEPROM gespeichert werden. Der Basic-Computer wird durch eine Autostartfunktion zur Applikation. Das Programm wird dann nach Einschalten des Basic-Computers sofort gestartet.

Bedienung

Die Grundeinstellungen können nur über die serielle Schnittstelle vorgenommen werden. Das Pollin-Board wird über einen USB-Seriell-Adapter mit dem PC verbunden. Es wird ein Terminalprogramm wie TeraTerm benötigt. Einstellungen:

The screenshot shows the 'Tera Term: Terminal setup' dialog box. It has a title bar with a close button (X). The dialog is divided into several sections. On the left, under 'Terminal size', there are two input fields: the first contains '80' and the second contains '24', separated by an 'X' symbol. Below these are two checkboxes: 'Term size = win size' (unchecked) and 'Auto window resize' (checked). To the right, under 'New-line', there are two dropdown menus: 'Receive:' set to 'CR' and 'Transmit:' set to 'CR'. Below these are two more checkboxes: 'Local echo' (unchecked) and 'Auto switch [VT<->TEK]' (unchecked). At the bottom left, there is a 'Terminal ID:' dropdown menu set to 'VT100' and an 'Answerback:' text input field. On the right side of the dialog, there are three buttons: 'OK', 'Cancel', and 'Help'.

The screenshot shows the 'Tera Term: Serial port setup' dialog box. It has a title bar with a close button (X). The dialog contains several settings, each with a label and a dropdown menu: 'Port:' set to 'COM1', 'Baud rate:' set to '9600', 'Data:' set to '8 bit', 'Parity:' set to 'none', 'Stop:' set to '1 bit', and 'Flow control:' set to 'none'. To the right of these settings are three buttons: 'OK', 'Cancel', and 'Help'. At the bottom, there is a section titled 'Transmit delay' which contains two input fields: the first is '0' followed by 'msec/char' and the second is '0' followed by 'msec/line'.

Welche COM-Schnittstelle einzustellen ist, bitte der Systemsteuerung entnehmen.

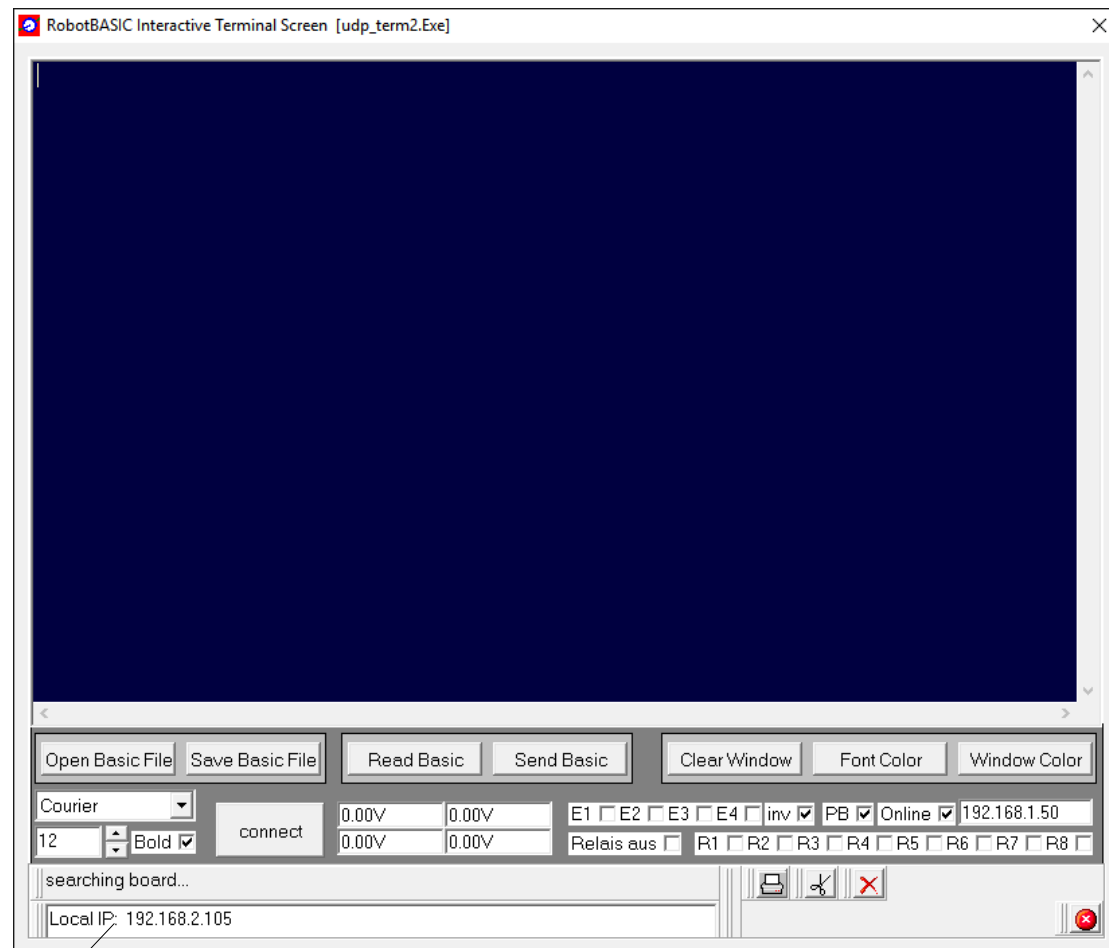
Nach dem Einschalten der Betriebsspannung wird die Startmeldung: netbasic V0.0beta 4086 bytes free angezeigt.

Vorbereitung für den Netzbetrieb

Vorbemerkung:

netbasic verwendet für den Datenaustausch reines UDP ohne Verschlüsselung ohne Sicherheitsmaßnahmen. Es ist zunächst nur für den Betrieb in einem LAN und in der ersten Betaversion nur für den Betrieb innerhalb *eines* 8-Bit-Segments gedacht.

Für die Verbindung via Netzwerk wird mein UDP-Term benötigt. UDP-Term.exe bitte in ein beliebiges Verzeichnis kopieren und evtl. eine Verknüpfung auf dem Desktop anlegen. Nach dem Start zeigt sich folgendes Bild:



Hier wird die IP-Adresse des PCs angezeigt. In diesem Beispiel: 192.168.2.105.

192.168 ist die LAN-Adresse

2 ist die Segment-Adresse

105 ist die PC-Adresse innerhalb des Segments. Netbasic ist zunächst für solche Netze gedacht. In diesem Fall kann man von einer Netzmaske: 255.255.255.0 ausgehen.

Ein Wort zur MAC-Adresse: Der Netzwerkchip auf dem Pollin-Board hat leider selbst keine eigene MAC-Adresse, netbasic liefert sie. Das heißt, vom Prinzip her, hätten alle Pollinboards mit netbasic dieselbe MAC-Adresse. Wenn nur ein Board im LAN betrieben wird, ist das noch kein Problem, bei mehreren – und das ist der Sinn – schon. Es kann daher ein Nibble der MAC-Adresse eingestellt werden, sodass der Betrieb von 16 Boards im LAN z.Z. möglich ist.

Die ersten Einstellungen:

Die ersten Einstellungen werden mit dem Basic durchgeführt:

let seg = xxx
let adr = xxx

xxx ist die Segmentadresse siehe oben

xxx ist die Adresse des Pollinboards, bitte selbst eine freie Adresse aussuchen. Oft werden die Adressen vom Router erst ab 100 oder so vergeben. xxx also z.B. 50

let gw = xxx

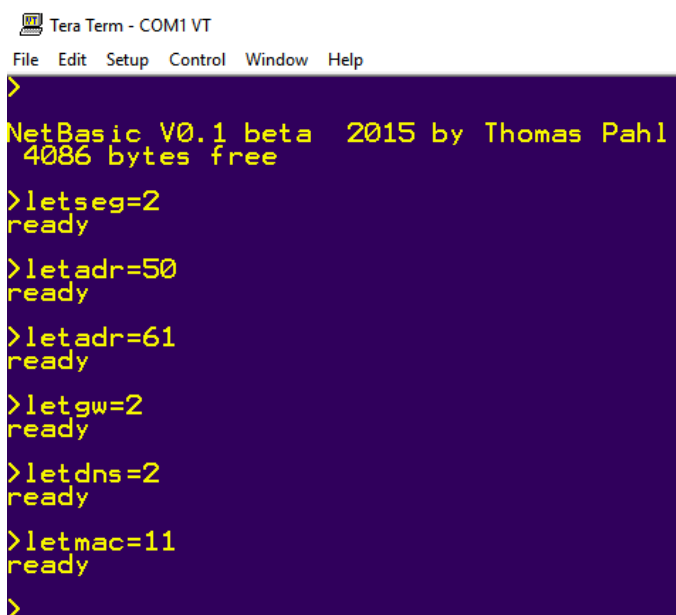
xxx ist die Adresse des Gateways – wenn unbekannt nicht eingeben.

let dns = xxx

xxx ist die Adresse des DNS-Servers – wenn unbekannt, nicht eingeben.

let mac = 0

ist die MAC-Adresse des ersten Pollinboards – muss unbedingt eingegeben werden. Erst nach dieser Eingabe werden alle Eingaben im EEPROM gespeichert.



```
Tera Term - COM1 VT
File Edit Setup Control Window Help
>
NetBasic V0.1 beta 2015 by Thomas Pahl
4086 bytes free
>letseg=2
ready
>letadr=50
ready
>letadr=61
ready
>letgw=2
ready
>letdns=2
ready
>letmac=11
ready
>
```

Bei der Eingabe müssen keine Leerzeichen eingegeben werden. Wenn man einen falschen Wert eingeben hat, kann man die Eingabe wiederholen. Als letzte Eingabe muss die let mac = 0 gemacht werden.

Nach erfolgreicher Eingabe (jede Eingabe wurde mit „ready“ quittiert) das Board bitte von der Stromversorgung trennen.

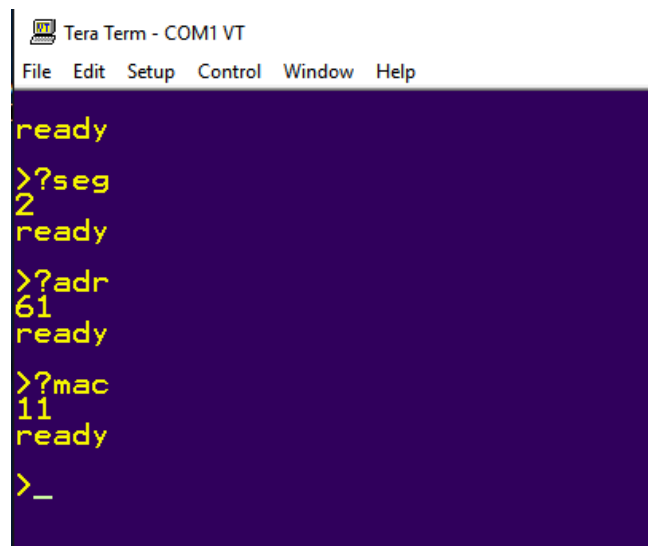
Nach dem Wiedereinschalten ist das Board für den Netzworkebetrieb bereit. Wer will kann sicherheitshalber nochmal die Werte abfragen:

? seg

? adr

? mac

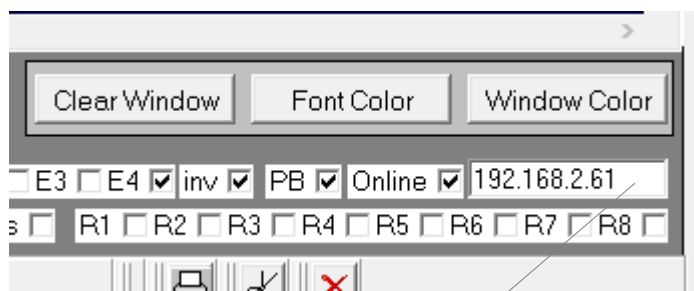
Sollte so ähnlich aussehen:



```
Tera Term - COM1 VT
File Edit Setup Control Window Help

ready
>?seg
2
ready
>?adr
61
ready
>?mac
11
ready
>_
```

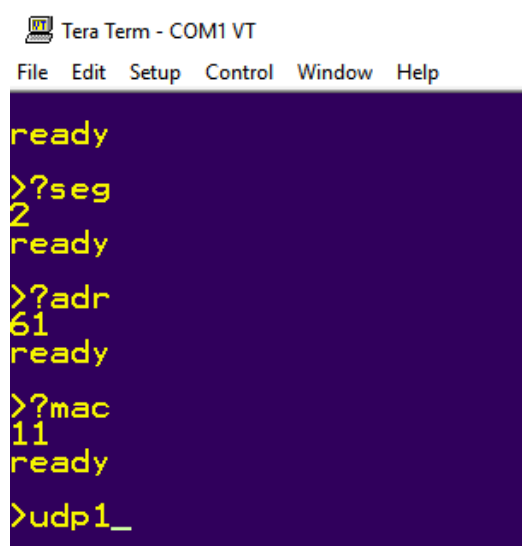
Fast geschafft! Jetzt muss UDP-Term noch die Adresse vom Board wissen



Adresse einfach überschreiben und wichtig(!) mit

Return übernehmen.

Der letzte Schritt: auf dem TeraTerm bitte eingeben:



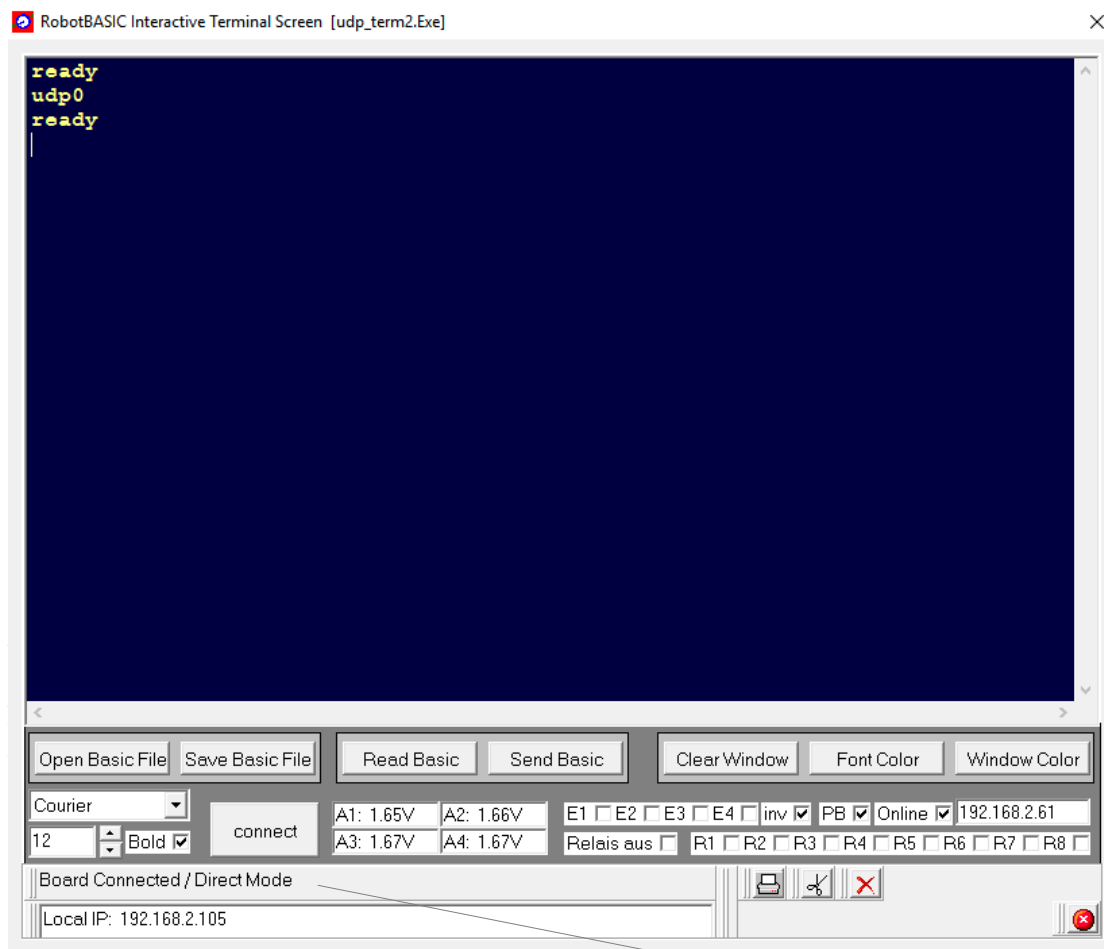
```
Tera Term - COM1 VT
File Edit Setup Control Window Help

ready
>?seg
2
ready
>?adr
61
ready
>?mac
11
ready
>udp1_
```

udp1

und Enter schaltet auf Netzwerkbetrieb um. Das „ready“ ist schon weg!

TeraTerm wird erst wieder benötigt, wenn man etwas mit der seriellen Schnittstelle macht.



Nun sollte UDP-Term so aussehen. Insbesondere die Nachricht sagt: verbunden!

Wer will, kann die Eingänge mal mit 5V oder Gnd verbinden und schauen, was sich tut... Oder die Kästchen neben R1...R8 anklicken. (Falls das Relaisboard angeschlossen ist)

Direct Mode bedeutet Eingabemodus: man kann Anweisungen eingeben, die direkt ausgeführt werden (wie bereits gemacht) oder auch Programmzeilen eintippen. Den anderen Modus nenne ich Run Mode. Das Board führt ein Programm aus. Auf Eingaben reagiert das Board nur noch, wenn im Programm entsprechende Anweisungen eingebaut wurden.

Open Basic File	öffnet ein Dateiladefenster, wie man es kennt. Es können reine Textdateien ohne Formatierungen ins Terminalfenster geladen werden. Das Fenster verhält sich, wie der Texteditor von Windows. Das Terminal schaltet offline.
Save Basic File	wie öffnen nur eben speichern.
Read Basic	holt ein Programm, welches sich im RAM des Boards befindet auf das Terminal und schaltet in den Offline-Modus.
Send Basic	Sendet das Programm im Fenster als Ganzes zum Board.
Clear Window	Löscht den Fensterinhalt. Kopiert ihn sicherheitshalber aber in die Zwischenablage. Mit Strg + v kann der Fensterinhalt wiederhergestellt werden.

Basic Anweisungen

Alle Basic-Anweisungen werden klein geschrieben. Leerzeichen zwischen den Elementen erhöhen die Lesbarkeit, sind aber nicht nötig. Variablen sind Grossbuchstaben. Siehe Variablen.

Parameter in [] sind alternativ oder können weggelassen werden. Eine Basic-Programmzeile sieht so aus:

Zeilennummer Befehl 1 [Parameter] [: Befehl 2 Parameter : Befehl 3 Parameter....]

Max.80 Zeichen.

ex:

10 for N = 1 to 10 : print N : next

Wo immer eine Zahl als Argument erwartet wird, kann auch ein komplexer Rechenausdruck stehen. (ausser list, lcd, inc, dec und save) ex.: gosub 300 + (10 * N) ist möglich!
Zeilennummern sind von 1 bis 65535 erlaubt.

baud **baud Übertragungsgeschwindigkeit**

ex.:

baud 19200

ändert die Übertragungsgeschwindigkeit der seriellen Schnittstelle. Standard nach Reset ist **9600 Baud**. **Übertragungsgeschwindigkeit** darf folgende Werte annehmen:

600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 31250

die letzten Nullen dürfen auch weggelassen werden.

Übertragungsgeschwindigkeit darf auch ein berechenbarer Ausdruck sein. Wird während einer Terminalsitzung die Übertragungsgeschwindigkeit einseitig geändert, reisst die Verbindung zum Terminal ab. Das Terminal muss ebenfalls auf den neuen Wert eingestellt werden.

clear

ex.:

clear

löscht alle Variablen und Programmschleifen. Alle Variablen erhalten den Wert 0. Alle Stringvariablen erhalten den Nullstring (""). Die Relais werden ausgeschaltet, die PWMs auf Anfangswert gesetzt. Direkt- und Programm-Modus.

close

Setzt eine "Umleitung" auf den Datenbereich einer anderen Programm-Datei durch **open** wieder zurück auf das aktuelle Programm. Siehe **open**, **read**, **write**, **restore** und **dptr**. Die Schreib-/Lesezeiger werden auf Anfang gesetzt.

ex.:

close

cls

(clear screen)

ex.:

10 cls

cls löscht den Terminalbildschirm- oder LCD-Inhalt. (abhängig vom aktuellen Ausgabemedium, das mit **lcd** eingestellt wird) Kann auch von der Befehls-eingabe aufgerufen werden. Das Terminal muss das Zeichen „Formfeed“ verstehen. Siehe auch **lcd**.

cont (continue)

ex.:

cont

setzt die Programmausführung nach Programmabbruch (mit ctrl c) fort. Nur im Direktmodus. Nach einem Fehler ist die Fortsetzung mit **cont** nicht möglich.

ctrl

Kontrolle
gibt dem Programm die Fähigkeit, den Zugriff auf die Relais vom Terminal zu unterbinden

ex.:

ctrl0 schaltet den Zugriff auf die Relais aus.

ctrl1 schaltet den Zugriff auf die Relais ein.

dec

(decrement)

dec (System)-Variable

ex.:

dec D

10 **dec time**

20 **dec stack**

Verringert den Wert der (System-)Variablen um 1. Programm und Direktmodus.

do

do Anweisungen loop [until Ausdruck]

ex.:

10 **do**

20 inc A

30 print A

40 **loop until A = 10**

oder in einer Zeile:

10 let time = 0 : **do** : inc A : **loop until time = 10** : print A

Der Anweisungsblock zwischen **do** und **loop** wird wiederholt bis der Ausdruck nach **until** wahr wird. Ohne **until** wird der Block immer wiederholt. Mehrere (5) **do-loop**-Schleifen können ineinander verschachtelt werden.

end

Programm-Ende

end beendet das Programm und kehrt zum Direktmodus zurück. **end** kann auch als Stopmarke benutzt werden. Eine extra **stop**-Anweisung wurde „eingespart“. Nach **end** kann das Programm mit **cont** fortgesetzt werden, auch wenn das vielleicht nicht immer erwünscht ist.

ex.:
1000 **end**

exit

Schleifen-Ausgang

exit dient zum vorzeitigen Verlassen einer Programmschleife. Bitte Schleifen niemals mit **goto** verlassen. Das Programm wird hinter der **loop**- oder **next**-Anweisung fortgesetzt.

ex.:
10 let A = 0 : **do**
20 if A > 5 then **exit**
30 inc A : **loop**
40 hier geht es weiter nach **exit**

for

for NumVar = Startwert to Endwert [step Schrittweite]

ex.:
10 **for** N = 2 to 24 **step** 2
20 Anweisung 1
...
50 Anweisung X
60 **next**
oder:
70 **for** A = 2 * pi to pi ^ 2 **step** pi / 100 : print sin(A) : **next**

for / next ist eine Zähl-Schleife. Der Anweisungsblock zwischen **for** und **next** wird so oft wiederholt, wie durch Startwert, Endwert und Schrittweite bestimmt wird. Als Zähler dient eine Variable (NumVar), die bei jedem Schleifendurchlauf um den Wert Schrittweite erhöht oder erniedrigt (bei negativer Schrittweite) wird, (mit dem Startwert startet) bis der Endwert erreicht ist. Defaultwert für Schrittweite (wenn Schrittweite nicht angegeben wird) ist: +1. Schleifen dürfen jederzeit mit **gosub** verlassen werden – es muss dann mit **return** wieder in die Schleife zurückgekehrt werden. Aber niemals mit **goto**! Mehrere (5) **for / next**-Schleifen können ineinander verschachtelt werden.

get

get NumVar

Eingabe eines einzelnen Zeichens

ex.:
10 **get** A : print chr A
20 if buf > 0 then **get** B else return

holt ein Zeichen von der RS232 Schnittstelle und speichert es als Zahlenwert (ASCII) in der Variablen (A). **get** wartet bis ein Zeichen empfangen wurde. Es kann vorher die Systemvariable **buf** abgefragt werden, um Blockaden zu vermeiden. Oder man fragt mit **on buf** ereignisgesteuert ab. Siehe auch **Systemvariable get**.

goto

goto Zeilen-Nummer

Sprunganweisung

ex.:
goto 100
verzweigt zur Programmzeile 100. Kann auch von der Befehlseingabe aufge-

rufen werden z.B. um das Programm ohne das Löschen der Variablen (wie mit **run**) oder ab einer bestimmten Stelle zu starten oder fortzusetzen.

gosub **gosub Zeilen-Nummer** Sprung in ein Unterprogramm

ex.:
 20 **gosub** 100 hier der Aufruf des Unterprogramms
 30 Anweisung zu der zurückgekehrt wird.
 ...
 50 **gosub** 100 wiederholter Aufruf
 60 Anweisung zu der zurückgekehrt wird.
 ...
 ...
 100 **Anweisung 1** Ab hier das Unterprogramm
 ...
 110 Anweisung x
 120 **return** Springt zurück zur Stelle, die der Aufrufenden folgt.

Oder:
 20 **gosub** 100 : Anweisung : **gosub** 100 : Anweisung : Anweisung
 ...
 ...
 100 Anweisung1 : Anweisung2 : **return**

verzweigt zur Programmzeile (hier 100) ins Unterprogramm und merkt sich die Stelle von der aus verzweigt wurde. Rückkehr mit **return**. Aus einem Unterprogramm können wieder Unterprogramme aufgerufen werden. Schachtelungstiefe max. 5 (20). Im Direktmodus wie goto (ohne Rücksprung).

inc (increment)
inc (System)-Variable

ex.:
inc F
 Erhöht den Wert der Variablen (hier F) um 1. Programm und Direktmodus.

if **if Ausdruck then [Anweisung] [:Anweisung]...[else Anweisung] [:Anw...]**

Die Anweisung für Entscheidungen im Programm: **If** prüft den **Ausdruck** und führt die auf **then** folgenden Anweisungen aus, wenn der Ausdruck wahr (<> 0) ist. Ist die **else**-Anweisung vorhanden, so werden die darauf folgenden Anweisungen bei nicht Erfüllung ausgeführt. Bei Zeichenketten ist nur "=", "<>" und "in" erlaubt. Ausserdem muss der 1. **Vergleichsoperand eine Stringvariable** sein. **If / then / else** müssen in derselben Zeile stehen. Nur eine **If**-Anweisung pro Zeile. Nur im Programm.

ex.:
 10 **if** time > 33 **then** goto 20 **else** goto 30
 20 **if** A xor (B and 7) **then** print "OK"
 30 **if** \$1 = "Hallo" **then** let A = 2 : return **else** let A = 3 : return
 30 **if** \$0 <> \$2 **then** gosub 50
 40 **if** \$5 = clock **then else** let \$5 = "00:00:00" 'der then-Teil darf fehlen.
 50 **if** \$1 in \$2 **then** return

lcd 1	20 lcd 1, 2, 16 : print "Hallo Welt"	Ausgabe auf LCD (2x16)
	Wechselt das Ausgabemedium. lcd 1 schaltet die Zeichenausgabe von RS232 auf das LCD um. Ohne die Angabe von Zeilen- und Spaltenzahl wird 4 x 20 angenommen. lcd 0 schaltet wieder zurück zum Terminal.	
lcd 3	verschiebt den Displayinhalt um 1 Position nach links.	
lcd 4	verschiebt den Displayinhalt um 1 Position nach rechts.	
lcd 5	schaltet das LCD aus. (lcd1 schaltet es wieder ein.)	
let	let NumVar = Ausdruck let StringVar1 = Stringvar2, [Startpos], [Anzahl] let Stringvar (Position) = "Zeichen" let Stringvar (Position) = ASCii-Wert ex.: 10 let A = 2 * (A + sin (pi / 8)) rechnet Ausdrücke (Formeln) aus. let N = 5 > 4 Kann vergleichen 10 let C = A and (250 or B) Logische Verknüpfungen 30 let \$0 = "Hallo Welt" Kopiert eine StringKonstante in einen String. let \$1 = \$2 Kopiert Strings: \$2 nach \$1 let \$2 = \$3, 6 Kopiert \$3 nach \$2 ab dem 6. Zeichen let \$2 = \$3, , 6 Kopiert die ersten 6 Zeichen von \$3 nach \$2 40 let \$0 = \$3, 5, 4 Kopiert 4 Zeichen von \$3 ab dem 5. Zeichen nach \$0 50 let \$4 = clock Kopiert Uhrzeit nach \$4 let rel2 = 1 Schaltet Relais 2 ein let \$4(3) = "A" Tauscht ein einzelnes Zeichen aus.	
	Weist einer String- oder (System-) Variablen einen Wert zu. Kann auch direkt (im Befehlsmodus) eingegeben werden. Negative Werte bitte in Klammern: let A = (-2.33) + (-1.4) Ausdruecke werden strikt von links nach rechts berechnet, Punkt-vor Strichrechnung ist nicht implementiert. Bitte Klammern setzen zur Änderung der Reihenfolge. let N = 3 + (4 * 9) . Max. 5 (20) Klammerebenen. Siehe auch Stringfunktionen .	
list	list Zeile - Zeile ex.: list listet das gesamte Programm list 10 nur Zeile 10 list -100 bis Zeile 100 list 100 - ab Zeile 100 list 20 - 100 die Zeilen 20 bis 100	
	Zeigt die gewünschten Programmzeilen an. list alleine listet das ganze Programm. list 10 nur Zeile 10 . list 20 - 100 die Zeilen 20 bis 100. Sollte nicht im Programm verwendet werden.	
load	load falls ein Program im EEPROM ist, wird es ins RAM geladen. Wenn es mit save1 gespeichert wurde, wird es direkt gestartet.	

locate

locate Zeile, Spalte

ex.:
10 **locate 1, 5 : print** freq
20 **locate A, B – 2 : print** ad1

Setzt den Cursor auf dem LCD auf Zeile und Spalte und schaltet auf LCD - Betrieb um. Zeile = 1...4, Spalte = 1....20

morse

morse String

ex.:
10 **morse “netbasic ist super“**
20 **morse \$0**
30 **morse date**

Gibt den String als Morsezeichen am Relay 8 aus. String kann jede in netbasic erlaubte Zeichenkette (Konstante, (System-)Variable, einzelnes Zeichen) sein. Die Ausgabegeschwindigkeit wird mit der Systemvariablen **speed** eingestellt. Siehe **speed**.

new

(alles neu)

ex.:
new

löscht das im **RAM**-Speicher befindliche Programm und alle Variablen. Nur Direktmodus. (Zum Löschen des **internen Eproms** anschliessend **save 0** ausführen.)

on

on time T-Wert gosub Zeilennummer
on dig n [, Flanke] gosub Zeilennummer
on err gosub Zeilennummer
on key gosub Zeilennummer

ex.:
10 **on time 300 gosub 100**
20 **on dig1, 1 gosub 200**
30 **on dig4, 0 gosub 1200**
40 **on buf gosub 900**

T-Wert = 1.... 65535 Sekunden, **n** = 1....4 und **Flanke** = 0 oder 1.

on verzweigt bei Auftreten eines Ereignisses zur angegebenen Programmzeile und merkt sich die Position (im Programm) für einen späteren Rücksprung (return). **on** Instruktionen *sollten* am Programmanfang stehen. Die aufgerufenen Unterprogramme *dürfen nicht* am Programmanfang stehen!

on ist nicht interrupt gesteuert, sondern wird nach jeder Anweisung geprüft. Daher können Anweisungen, die den Programmablauf blockieren (wait, get, input, inkey) das Erkennen der Ereignisse verzögern oder sogar verhindern.

on buf aktiviert ein Ereignis, sobald ein Zeichen im RS232 Empfangspuffer ist. z.B. wenn auf dem Terminal eine Taste gedrückt wurde.

on dig1 - on dig4 wird aktiviert, wenn der entsprechende Eingang **dig1** bis **dig4** den Wert **Flanke** annimmt. Logisch eins = + 5 Volt, logisch null = 0Volt. Defaultwert für **Flanke** ist **1**. (Ruhewert 0 Volt, aktiv bei Wechsel auf 5 Volt)

on err verzweigt bei Auftreten eines Fehlers zur angegebenen Programmzeile. Dort kann der Fehler „behandelt“ werden. Der normale Programmabbruch im Fehlerfall wird ausgeschaltet. Ist in der Fehlerbehandlungsroutine selbst ein Fehler, wird das Programm abgebrochen.

on time löst ein Ereignis aus, wenn **time** den Wert **T-Wert** erreicht hat. Bei Aufruf von **on time** wird die Systemvariable **time** neu gestartet. **time** ist die Systemuhr und wird nach dem Einschalten des Basic-Computers jede Sekunde eins weiter gezählt.

Für einen zyklischen Aufruf eines Unterprogramms muss die erste Anweisung des Unterprogramms „let time = 0“ sein.

```
ex.:
10 on time 1000 gosub 100    Am Programmstart wird das Ereignis
...                          definiert.
...
...
100 Anweisung 1             Ab hier wird das Ereignis on time abge-
...                          arbeitet.
110 Anweisung 2
....
150 return                  Mit return wird zu der Stelle zurück-
                              gesprungen, an der das Programm durch
                              das Ereignis unterbrochen wurde. Siehe
                              auch return.
```

poke **poke Word-Adresse, Byte-Wert**

```
ex.:
10 poke 4000, 234
20 poke A and 4096, 9 or C
```

Schreibt den Wert *Byte-Wert* in die RAM-Speicherzelle mit Adresse *Word-Adresse*. Im Programm-Modus oder direkt.

print oder ? **print NumVar**
 print StringVar
 print StringKonstante
 print Ausdruck
 print bin Ausdruck
 print hex Ausdruck

```
ex.:
? A
? not ( A or ( C and 255 ) )
10 cls : locate 1, 1 : print "Hallo"
20 print $3 ;
```

druckt den Inhalt von Variablen oder Textkonstanten oder Funktionen auf

dem Terminal oder LCD aus. **print** rechnet auch Ausdrücke wie $A + B * C$ aus. Ein Semikolon am Ende der Anweisung unterdrückt den Zeilenvorschub. (Auf dem Terminal). Ein Komma setzt den Tabulator auf die nächste Position (für Tabellen).

ex.:
 10 **print** " x", : **print** "sin(x)", : **print** "cos(x)", : **print** "tan(x)"
 20 for N = 0 to pi / 2 step pi / 20 : **print** N, : **print** sin (N),
 30 **print** cos (N), : **print** tan (N) : next

Programm- oder Befehlsmodus.

return **return [Zeilennummer]**

Rücksprung vom Unterprogramm. Es wurde zuvor mit **gosub** in ein Unterprogramm gesprungen. Mit **return** wird an die Stelle zurückgekehrt, die dem Aufruf folgt. Ist die **Zeilennummer** angegeben, wird dorthin gesprungen und die mit **gosub** gespeicherte **Rücksprungadresse** verworfen.

ex.:
 10 **gosub** 100 'Aufruf des Unterprogramms
 20 Anweisung1

 100 Anweisung2 'hier beginnt das Unterprogramm

 150 **return** 'Rücksprung (zur Zeile 20)

run **run**

ex.:
run
 startet das im Speicher befindliche Programm. **run** löscht vorher alle Variablen wie **clear**. Nur im Direktmodus.

save (sichern)
save 0 Speichert das im RAM befindliche Programm dauerhaft im internen EEPROM. Es muss mit run gestartet werden. Dient auch zum Löschen des EPROMS (nach **new**)

save 1 Wie **save 0**. Das Programm wird nach einem Reset sofort gestartet. Ist kein Programm im RAM-Speicher wird eine Fehlermeldung ausgegeben.

wait **wait numer. Wert** warten

ex.:
wait 10 Wartet 1 Sekunde.
 10 **wait A**
 20 **wait ad1 / 5** auch so etwas ist möglich

Das Programm wartet die angegebene Zeit in Zehntelsekunden. Wertebereich 1 ... 65535. Die Zeit ist nicht Timer gesteuert daher nicht exakt.

rset (reset) **rset relay x** (x = 1...8)

ex.:

rset relay 1

Schaltet Relais 1 aus. Programm und Direktmodus.

set (setze) **set relay x** (x = 1...8)

ex.:

set relay 2

Schaltet Relais 2 ein. Programm und Direktmodus.

tgl (toggle) **tgl relay x** (x = 1...8)

ex.:

tgl relay 3

Toggelt das Relais 3. War Relais 3 eingeschaltet, wird es ausgeschaltet. War es ausgeschaltet, wird es eingeschaltet. Programm und Direktmodus.

imp (impuls) **imp relay x** (x = 1...8)

ex.:

10 let **pulse** = 20

20 **imp relay 3**

Schaltet das Relais 3 für z.B. 20 Zeiteinheiten ein und danach wieder aus, die zuvor in der Systemvariablen **pulse** gespeichert wurden. Zeiteinheit ist 1/10 Sekunde. Defaultwert für pulse ist 10 (= 1 Sekunde). Programm und Direktmodus. (War das Relais vorher eingeschaltet, wird es für die Impulszeit ausgeschaltet.)

blk (blink) **blk relay x** (x = 1...8)

ex.:

blk relay 1

Schaltet das Relais1 mit Blinkrythmus (1Hz) ein. Das Relais muss mit **rset relay 1** wieder ausgeschaltet werden. Programm und Direktmodus.

www.netbasic.de

Variablen

1.) numerische Variablen

A, B, C, D, E, ... ,Z

netbasic kennt die 26 vordefinierten Variablen A,B,C....Z, die numerische Werte (Fließkomma) aufnehmen können. Genauigkeit max. 7 Stellen / max. 5 Stellen nach dem Dez.-Punkt. Wissenschaftliche Notation ist nicht implementiert. Variablen müssen vor ihrer Verwendung nicht deklariert noch initialisiert werden. Ihr Startwert ist 0.

Beispiel für einen Wandlungsfehler: $1.2345 * 100 = 123.45001$ Es werden keine „kosmetischen“ Rundungen durchgeführt.

Array:

Die Variablen **A – J** können auch als Array mit dem Namen **A** angesprochen werden. Das Array **A** besteht aus 10 Elementen mit den Indizes von **0 ... 9**.

Die Variablen **K...T** bilden das Array **B**

A(0) = A, A(1) = B, A(2) = C A(9) = J (B(0) = K)

ex.:

```
10 print "x", : print "sqr(x)"           'Quadratwurzel-Tabelle
20 for P = 0 to 9 : let A(P) = sqr ( P + 1 ) : next
30 for P = 0 to 9 : print P + 1, : print A(P) : next
```

2.) Zeichenketten Variablen

\$0, \$1, \$2, \$3, \$4, \$5, \$6, \$7 können jeweils 80 Zeichen aufnehmen.

ex.:

```
10 let $0 = "Garage" : print $0
```

3.) Systemvariablen

ad1

ADC1

Wert des 1. Analog-Digital-Wandlers. 0 = 0Volt , 1023 = + 5 Volt. Nur lesen. Auflösung 10bit.

ex.:

```
10 print ad1 / 1023 * 5; : print " Volt"
```

ad2

ADC2

Wert des 2. Analog-Digital-Wandlers. 0 = 0Volt , 1023 = +5 Volt. Nur lesen. Auflösung 10bit.

ex.:

```
10 let A = ad2 / 1023 * 5
20 if ad2 < 512 then gosub 1000
```

ad3	<p>ADC3 Wert des 3. Analog-Digital-Wandlers. 0 = 0Volt , 1023 = +5 Volt. Nur lesen. Auflösung 10bit.</p> <p>ex.: 10 let A = ad3 / 1023 * 5 20 if ad3 < 512 then gosub 1000</p>
ad4	<p>ADC4 Wert des 4. Analog-Digital-Wandlers. 0 = 0Volt , 1023 = +5 Volt. Nur lesen. Auflösung 10bit.</p> <p>ex.: 10 let A = ad4 / 1023 * 5 20 if ad4 < 512 then gosub 1000</p>
buf	<p>Buffer Anzahl der Zeichen im RS232 Empfangspuffer. Nur lesen.</p> <p>ex.: 10 if buf > 0 then gosub 300 20 on buf gosub 1000</p>
clock	<p>Uhr Uhrzeit der Systemuhr als Zeichenkette als HH:MM:SS. Nur lesen. HH = Stunden, MM = Minuten, SS = Sekunden.</p> <p>ex.: 10 let \$0 = clock ? clock 30 if \$1 = clock then imp rel1 : locate 1,1 : print „Wecken“</p>
dig1	<p>Digitaleingang 1 Zustand des digitalen Eingangs 1. 1 = +5V , 0 = 0V. Nur lesen.</p>
dig2	<p>Digitaleingang 2 Zustand des digitalen Eingangs 2. 1 = +5V , 0 = 0V. Nur lesen.</p>
dig3	<p>Digitaleingang 3 Zustand des digitalen Eingangs 3. 1 = +5V , 0 = 0V. Nur lesen.</p>
dig4	<p>Digitaleingang 4 Zustand des digitalen Eingangs 4. 1 = +5V , 0 = 0V. Nur lesen.</p>
dow	<p>Day of week Wochentag der Systemuhr. Werte: 1...7 . Montag = 1. Mit DCF77-Modul: nur Lesen (ohne Modul auch Schreiben).</p>
dow\$	<p>dow als String Statt einem Zahlenwert liefert dow\$ den Wochentag als abgekürzten String: „Mo, Di, Mi, Do, Fr, Sa, So“. Mit DCF77-Modul: nur lesen.</p>
date	<p>Datum Datum der Systemuhr. Mit DCF77-Modul: nur lesen. date ist ein String mit dem Format: Wochentag, TT.MM.20JJ</p>

day	Tag. Mit DCF77-Modul: nur lesen.
erl	Fehlerzeile Nummer der Programmzeile, in der der letzte Fehler aufgetreten ist.
err	Fehlernummer Nummer des zuletzt aufgetretenen Fehlers. err = 0 heisst: kein Fehler aufgetreten. Nach der Abfrage von err wird err gelöscht. Lesen und setzen möglich – setzen erzeugt künstlich einen Fehler. ex.: 10 on err gosub 1000 20 print "Eingabe : "; : input A : let A = 1 / A 'testweise 0 eingeben. 1000 return 20 'falsche Eingabe wiederholen.
free	freier Speicher free gibt den freien Programmspeicherplatz in bytes wieder. Nur lesen. ex.: print free
get	get ist auch eine Systemvariable. get liefert den Ascii-Wert des letzten empfangenen Zeichens von der seriellen Schnittstelle. ex.: 10 do : loop until buf > 0 : let A = get diese Zeile ahmt die Instruktion „ get A “ nach. Sie hat den Vorteil jederzeit durch on unterbrochen werden zu können.
hour	Stunden Stunden der Systemuhr. Nur 24-Stundenmodus. Lesen und setzen. (Setzen nicht mit DCF77-Modul.)
min	Minuten Minuten der Systemuhr. Lesen und setzen. (Setzen nicht mit DCF77-Modul.)
mon	Monat. Mit DCF77-Modul: nur lesen.
pulse	Impuls Schreiben in pulse: Zeitwert für die imp-Anweisung. Impulslänge in 1/10 Sekunden. ex.: 10 let pulse = 30 (=3 Sekunden, kleinster Wert: pulse = 1) 20 imp relay 2 (Relais 2 zieht für 3 Sekunden an)
relay	Relais Zustand der Relais 1. 1 = Ein, 0 = Aus. Lesen und setzen. 8-Bit Wert. Bit 0 entspricht Relais 1, Bit 7 ist Relais 8. ex.: let relay.2 = 1 schaltet Relay 3 ein

sec	Sekunden Sekunden der Systemuhr. Lesen und setzen. (Setzen nicht mit DCF77-Modul.)								
speed	Ausgabegeschwindigkeit der morse-Anweisung . Anzahl Worte pro Sekunde. Maßstab ist das Wort „Paris“. let speed = 20 Defaultwert ist 12.								
sync	synchron mit PC-Uhr								
time	Systemuhr time ist die Systemuhr. time wird nach dem Einschalten des Basic-Computers jede Sekunde um 1 erhöht. Auflösung: 32 Bit. ex.: <table> <tr> <td>? time</td><td>Lesen</td></tr> <tr> <td>let A = time</td><td>Lesen</td></tr> <tr> <td>10 let time = 0</td><td>Setzen</td></tr> <tr> <td>20 if time > A then goto 200</td><td>Lesen</td></tr> </table>	? time	Lesen	let A = time	Lesen	10 let time = 0	Setzen	20 if time > A then goto 200	Lesen
? time	Lesen								
let A = time	Lesen								
10 let time = 0	Setzen								
20 if time > A then goto 200	Lesen								
year	Jahr. Mit DCF77-Modul: nur lesen. Datum und Wochentag werden von der Systemuhr nicht fortgeschrieben. Sie werden nur vom DCF-Signal geliefert.								

Zugriff auf einzelne Bits

auf ein einzelnes Bit einer **numerischen** (System-) **Variablen** greift man mit dem Dezimalpunkt zu:

(System-)Variable . Bitnummer

ex.:

let A.15 = 1

print pwm1.1

10 for N = 0 to 15 : print C.N : next

20 if J.3 = 0 then

Die entsprechende Variable wird **vorher** in eine **16-Bit-Wort-Zahl** gewandelt. Der Wert hinter dem Punkt muss kleiner als 16 sein. (Bit 0....15) und darf eine Konstante oder Variable sein. Das Bit kann nur die Werte 0 oder 1 annehmen. (Alle Werte > 0 werden als 1 interpretiert)

Operatoren

arithmetrisch

keine Rechenregelbeachtung, bitte die gewünschte Reihenfolge mit Klammern erzeugen.

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
^	Potenz - ermöglicht auch z.B. das Ziehen der 12.Wurzel: let A = 2 ^ (1 / 12)
mod	liefert den Rest einer Division ganzer Zahlen.
<<	bitweises links-schieben (Das Ergebnis ist ein Wordwert, der rechte Operand (der angibt um wieviel Bits verschoben wird) darf nicht grösser als 16 sein) let A = 16 << 2 print B << 3
>>	bitweises rechts-schieben (sonst wie <<)
@	pointer, liefert die physische Adresse einer Variablen. print @A

logisch

not	bitweises not , die Operanden werden in 16 Bit-Wordvariablen gewandelt. not (0) = 65535 not wird vom Interpreter wie eine Funktion behandelt. ex.: 10 let A = not (B) 20 print not (1 + C)
and	bitweises and , das Ergebnis ist ein 16 Bit-Word-Wert. ex.: 10 print 3 and 255 20 let A = B and (5 or 8)
or	bitweises or , das Ergebnis ist ein 16 Bit-Word-Wert. ex.: 10 print 3 or 8 20 let A = B or (5 and 255)
xor	bitweises xor , das Ergebnis ist ein 16 Bit-Word-Wert.

Vergleich

>	größer
<	kleiner
>=	größer gleich
<=	kleiner gleich
=	gleich
<>	ungleich

Math. Funktionen

abs	<p>liefert den Absolutwert eines Wertes zurück.</p> <p>ex.: print abs ((-1.6)) Ergebnis: 1.6 10 let B = abs (B)</p>
atn	<p>berechnet den Arcustangens eines Radiant-Wertes.</p> <p>ex.: ? atn (.3)</p>
cos	<p>berechnet den Cosinus eines Radiant-Wertes</p> <p>ex.: ? cos (.234) 10 print cos ((pi / 8) + B)</p>
hi	<p>liefert das höhere byte einer 16-Bit-Wort-Zahl.</p> <p>ex.: ? hi (999)</p>
lg	<p>berechnet den natürlichen Logarithmus zur Basis e.</p> <p>ex.: ? lg (e)</p>
lo	<p>liefert das niedere byte einer 16-Bit-Wort-Zahl.</p> <p>ex.: ? lo (999)</p>
log	<p>berechnet den 10-er Logarithmus eines Wertes.</p> <p>ex.: ? log (100) 10 let B = log (A + 99)</p>
peek	<p>liest ein Byte von der angegeben (RAM) Adresse.</p> <p>ex.: ? peek (2000) 10 let B = 12 20 let A = peek (B)</p>
rnd	<p>(round) rundet den angegebenen Wert auf die nächste ganze Zahl.</p> <p>ex.: ? rnd (1.6) liefert 2 als Ergebnis 10 let A = rnd (B)</p>

```
ex.:
? sin ( .234 )
10 print sin ( ( pi / 8 ) + B )
```

```
ex.:
? sqr ( 13.56 )
10 let A = sqr ( pi / 2 )
```

```
ex.:  
? tan ( .234 )  
20 print tan ( pi / 8 )
```

an jeden analogen Eingang kann ein Sensor angeschlossen werden. Deshalb muss der Pin angegeben werden. (1...4)

ex.:
print temp(4) druckt den Wert des Sensors an
Analogeingang 4 auf dem Terminal aus.

Stringfunktionen

asc Statt `print asc ("A")` schreibt man in netbasic: `print ("A")` (d.h.: **asc** gibt es nicht)

bin **bin Zahl**
liefert eine Zeichenkette, die den Wert **Zahl** binär darstellt. **Zahl** ist eine 16-Bit-Wort-Zahl. Das Zeichen «%» kennzeichnet eine Binärzahl.

ex.:
`print bin 4 + 333` Ergebnis: %101010001
`let $2 = bin 255`
`print val $2`

chr **chr Zahl**
liefert ein Zeichen mit dem **Ascii-Wert Zahl**. **Zahl** ist ein Bytewert.

ex.:
`print chr 13` sendet Wagenrücklaufzeichen (Return)
`let $1 = chr 10` weist einem String ein nicht druckbares Zeichen zu.
`write chr 13` schreibt einen 1 Zeichen String ins EEPROM (als Fließkommazahl)

hex **hex Zahl**
liefert eine Zeichenkette, die den Wert **Zahl** hexadezimal darstellt. **Zahl** ist eine 16-Bit-Wort-Zahl. Das Zeichen «&» kennzeichnet eine Hexadezimalzahl.

ex.:
`print hex 3 * 9` Ergebnis: &1B

in **Stringvariable1 in Stringvariable2**
liefert als Ergebnis die Position in **String 2**, ab welcher **String1** dort vorkommt. Kommt **String1** nicht in **String 2** vor, liefert **in** den Wert 0.

ex.:
`10 let $1 = "1234567890" : let $2 = "67"`
`20 let A = $2 in $1`
`30 if $2 in $1 then print "gefunden an Position: "; : print A`
`40 if $2 in $1 = 6 then print "richtige Position"`

str **str Ausdruck**
str verwandelt das Ergebnis eines Rechenausdrucks in einen **String**.

ex.:
`10 let $1 = str A` verwandelt den Wert der Variablen A in einen String.
`20 let $4 = str 4 + (3 * B)` Rechenergebnis in einen String.
`30 print $4` Ausdruck des Ergebnisses

val **val Stringvariable**
val liefert den numerischen Wert, den die **Stringvariable** repräsentiert. Der Inhalt von **Stringvariable** muss ein berechenbarer Ausdruck sein.
(Umkehrung von **str**) Ist der Inhalt von Stringvariable nicht numerisch, ist das Ergebnis 0.

ex.:
`10 let $1 = "125.45"` numerische Werte ohne Klammern
`20 print val $1`

```
30 let $2 = # sin(pi / 8 )
40 print val $2
```

Rechenausdrücke müssen dem Interpreter durch # mitgeteilt werden.

Die Zeichenkette wird dann in Tokens umgewandelt. Der String sieht dadurch «komisch» aus. # ginge auch im oberen Beispiel.

Richtig Sinn macht **val** mit **input#**. Dadurch ist es möglich, zur Laufzeit des Programms Formelausdrücke einzugeben, die durch das Programm berechnet werden. Ebenso ist es möglich mit **write** und **read** mathematische Bibliotheken abzuspeichern und durch ein und dieselbe Routine berechnen zu lassen.

val kann selbst nicht in mathematischen Ausdrücken benutzt werden:

let A = 3 * val \$1 geht nicht! – statt dessen: **let A = val \$1 : let A = 3 * A**

Vergleiche:

Strings können nur auf Gleichheit „=“ oder Ungleichheit „<>“ geprüft werden. Der **erste** Vergleichsoperand muss **immer** eine **Stringvariable** sein.

ex.:

```
10 if $1 = $2 then print "Die Namen sind gleich"
```

```
20 print $2 <> "Montag"
```

```
30 if "netbasic" = $3 then.... in netbasic nicht erlaubt
```

```
40 if date = "Mo, 01.12.2011" then.... geht mit netbasic nicht, statt dessen:
```

```
40 let $1 = " Mo, 01.12.2011" : if $1 = date then...
```

Stringvariable(Position)

greift auf ein einzelnes Zeichen innerhalb eines Strings zu. Man kann es lesen oder verändern. Einzelne Zeichen sind Bytes also auch Zahlen – je nach Interpretation. Man kann mit ihnen auch rechnen. netbasic interpretiert einzelne Zeichen stets als Zahlenwert (Ascii-Wert), da alle Ausdrücke von einer Rechenroutine ausgewertet werden. (Ausgenommen sind nur Zuweisungen an eine Stringvariable.)

ex.:

```
10 let $3(1) = "E"
```

Tauscht das **1.** Zeichen in der Stringvariablen **\$3** gegen „E“ aus. (Voraussetzung: **\$3 ist nicht leer**)

```
10 let A = $2(5)
```

weist der **Variablen A** den Ascii-Wert des **5. Zeichens** der Stringvariablen **\$2** zu. hier ist „=“ das Zuweisungszeichen

```
20 print $2(5) <----
```

Unterschied zu Standart Basic:

Druckt nicht das Zeichen selbst aus, sondern seinen **ASCII-Wert**. Daher auch keine eckigen Klammern.

```
40 print chr $2(5)
```

Mit **chr** erhält man aber auch das Zeichen selbst. (**write** funktioniert wie print.)

```
50 if $2(5) = "A" then....
```

und so prüft man auf ein Zeichen an bestimmter Position
(„=“ ist hier Operator - wird vom Interpreter auch anders codiert als im 1. Beispiel)

oder:

```
60 if $2(5) = 65 then....
```

Innerhalb der Klammer dürfen nur Konstanten oder Variablen stehen.

\$n(0) ist ein virtuelles Zeichen (das erste Zeichen ist **\$n(1)**) und dient dazu, Zeichen an das Ende der Zeichenkette anzuhängen.

ex.:

let \$0(0) = "d" hängt das Zeichen „d“ an das Ende von **\$0**.

Das funktioniert auch mit leeren Zeichenketten.

ex.:

10 for N = 1 to 10 : get A : **let \$1(0) = A** : next

Liest 10 Zeichen vom Terminal ein und bildet den **\$1** daraus.

Konstanten

pi	Die Kreiszahl: 3.1415927 ex.: ?sin (pi / 4)
e	Die Euler'sche Zahl: e = 2.71828183. ex.: ?lg (e)
true	intern verwendet für das Ergebnis eines Vergleichs. true = 65535 ex.: print 4 > 3 Ergebnis: 65535
false	intern verwendet für das Ergebnis eines Vergleichs. false = 0 ex.: print 3 > 4 Ergebnis: 0

true und false sind keine Schlüsselwörter und können nicht im Basic-Programm verwendet werden.

Zahlen

ausser 10-stelligen Fließkommazahlen kennt netbasic auch Hexadezimalzahlen und Binärzahlen.

Hexadezimalzahlen werden durch ein vorangestelltes „&“ gekennzeichnet.

ex.:

&F00C (dez. 61452) max. 4 Stellen

Binärzahlen werden durch ein vorangestelltes „%“ gekennzeichnet.

ex.:

%10010011 (dez. 147) max. 16 Stellen

www.netbasic.de

Fehlermeldungen

1	illegal direct	nicht im Direktmodus (if, return,on...)
2	syntax error	eine Anweisung ist falsch (geschrieben) In vielen Fällen gibt der Interpreter einen Hinweis auf die Position des Fehlers in der Form:X wobei x die Fehlerstelle markiert. (wie mcs51 basic) bei der Eingabe, sonst zur Laufzeit.
3	expected ') ' '	eine geöffnete Klammer wurde nicht geschlossen
4	expected ' to ' '	to oder Endwert fehlt in for
5	next without for	ein "for" fehlt
6	return without gosub	ein gosub weniger als returns
7	expected ' = ' '	= Zeichen in let oder for fehlt.
8	line number?	Zeilen-Nummer fehlt, ist Null oder existiert nicht.
9	stack error	exit wurde ausgeführt, obwohl keine Adresse auf dem Stapel war
11	not in prog mode	Nur im Direktmodus erlaubt. (new, run, cont, list, save, dir, del)
12	can't continue	Nach einem Programmfehler kann nicht mit cont fortgesetzt werden.
13	bad argument	Falsches Argument oder Division durch 0, . z.B.: let A = \$0
14	out of memory	zu viele Klammern, for/next-, do/loop-Schleifen, gosubs (max 5 Ebenen) oder Programmspeicher voll.
15	loop without do	ein do fehlt.
16	no program	kein Programm im RAM-Speicher bei save
17	no ext. eeprom	kein externes EEPROM vorhanden.
18	prog not found	Programm nicht gefunden
19	number too big	Diese Zahl ist für netbasic zu gross (> 10 000 000)
20	prog too big	Das Programm im RAM passt nicht ins interne EEPROM. (Bei save)
23	user break	Das Programm wurde vom Nutzer beendet.