Minibasic



```
WHAT?

READY
>LIST
4010 O.12
4020 P."**************************************************"
4030 P."*   M A S T E R M I N D                          *"
4040 P."**************************************************"
4050 IN."HOECHSTE DER ZU ERRATENDEN ZIFFERN (2...9) "N
4060 IFN<2G.4050
4070 IFN>9GOTO4050
4080 IN."ANZAHL DER STELLEN (2...9) "X
4090 IFX<2G.4080
5000 IFX>9GOTO4080
5002 P."SOLL DER AC1 DIE ZU RATENDEN ZAHLEN BESTIMMEN ? (J,N)"
5003 Q=0;T=INC
5004 IFT='J'G.5015
5005 IFT='N'P."GIB DIE ZIFFERN EIN ! (ZIFFER= *)";;Q=1;G.5015
5006 G.5003
5015 Q=1
5020 IFQ=0@(Q)=RND(N);G.5030
5021 H=INC ;@(Q)=H-48
5022 IF(H<49)+(H>48+N)G.5021
5023 O.'*'
5030 Q=Q+1;IFQ<X+1GOT.5020
5035 IFQ=1P." ERLEDIGT"
5036 P.;P."RICHTIGE ZAHL AN RICHTIGER STELLE GIBT '*'"

READY
>
```

Mini-BASIC-Interpreter

[11] Klein, R. D.: Basic Interpreter, Franzis Verlag München

Durch eine Optimierung dieses Interpreters sowie das Weglassen der Schleifen- und der stark vereinfachten Zeichenkettenverarbeitung, die durch andere Befehle substituierbar sind, konnte der Speicherbedarf für den Mini-BASIC-Interpreter von 2,75 KByte auf knapp 2 KByte reduziert werden.

Der Ansprung des Mini-BASIC erfolgt mit dem Kennbuchstaben Z oder dem Monitorkommando J 800. Darauf erscheint auf dem vorher gelöschten Bildschirm die Überschrift „MINI-BASIC AC 1 V2.1“, zwei Zeilen weiter „READY“,

—

Funkamateur 12/84, Funkamateure entwickeln Amateurcomputer „AC1“ (11) Dipl.-Ing. F. HEYDER - Y21SO

Der Mini-BASIC-Interpreter

Viele Problemstellungen lassen sich außer mit in Assembler- bzw. Maschinensprache geschriebenen Programmen auch mit solchen Programmen lösen, die in einer höheren Programmiersprache geschrieben sind. Solche Programme sind allgemein einfacher und schneller zu erstellen und für Einsteiger übersichtlicher, da komplexe Zusammenhänge mit nur einem Befehl beschrieben werden können. Man muß sozusagen nicht jedes Bit bzw. Byte zusammenstellen. Bei Computern für Heim- und Hobbyanwendungen bedient man sich dabei meist der Programmiersprache BASIC (Beginners Allpurpose Symbolic Instruction Code). Solche BASIC-Interpreter benötigen je nach dem realisierten Sprachumfang etwa 8 bis 12 KByte (teilweise auch noch mehr) Speicherplatz für das Interpreterprogramm. Dann natürlich noch entsprechenden Speicherplatz zum Ablegen der Anwenderprogramme.

Das übersteigt natürlich die Möglichkeiten des Grundmodells des Amateurcomputers „AC1“ und bleibt einer späteren Speichererweiterung vorbehalten. Um aber erste Erfahrungen im Umgang mit dieser

Programmiersprache zu sammeln, bot sich der Einsatz des in [11] veröffentlichten Tiny-Basic-Interpreters an, dessen Befehlsumfang eine Untermenge der Sprache BASIC darstellt. Dabei fehlen die Gleitkomma-Arithmetik und die Zeichenkettenverarbeitung. Durch eine Optimierung dieses Interpreters sowie das Weglassen der Schleifen- und der stark vereinfachten Zeichenkettenverarbeitung, die durch andere Befehle substituierbar sind, konnte der Speicherbedarf für den Mini-BASIC-Interpreter von 2,75 KByte auf knapp 2 KByte reduziert werden. Dieses Programm, aufgrund des verringerten Sprachumfangs und Speicherbedarfs als Mini-BASIC bezeichnet, findet in dem restlichen 2 KByte-EPROM-Speicher des Grundmodells des „AC1“ Platz.

Dieser Mini-BASIC-Interpreter ist besonders für das Sammeln erster Erfahrungen und Fertigkeiten in BASIC gedacht und ermöglicht das Lösen einfacher Anwendungsbeispiele. Im Vergleich zum Programmieren in Maschinensprache sind z.B. Fehler leichter zu finden und ziehen auch kaum ernsthafte Folgen nach sich. Natürlich haben diese und andere Vorteile auch den Nachteil, daß solche Programme mehr Speicherplatz und längere Verarbeitungszeiten gegenüber gleichen Problemlösungen in Maschinensprache erfordern. Im Extremfall kann das dazu führen, daß besonders zeitkritische Probleme nur in Maschinensprache lösbar sein werden. Im Rahmen dieses Beitrags ist es nicht möglich, einen Grundkurs in BASIC zu realisieren. Es sollen aber einige grundlegende Bemerkungen zur Arbeitsweise und Bedienung dieses Interpreters und zu den einzelnen Befehlen sowie auftretenden Fehlermeldungen gemacht und einige Programmbeispiele gezeigt werden. Damit dürfte es leicht möglich sein, sich innerhalb kurzer Zeit mit der Anwendung des Mini-BASIC-Interpreters vertraut zu machen.

Der Ansprung des Mini-BASIC erfolgt mit dem Kennbuchstaben Z oder dem Monitorkommando J 800. Darauf erscheint auf dem vorher gelöschten Bildschirm die Überschrift „MINI-BASIC AC 1 V2.1“, zwei Zeilen weiter „READY“, und am Anfang der nächsten Zeile das Zeichen „größer“ als Promptsymbol.

Der Interpreter erwartet nun die zeilenweise Eingabe von Befehlen bzw. Programmzeilen, wobei in der Grundausrüstung des Speichers die Eingabezeile nicht länger als 70 Zeichen sein darf. Die Zeile ist wie üblich mit Wagenrücklauf abzuschließen. Enthält die eingegebene Zeile am Anfang eine Zeilennummer (1 bis 32767), wird diese Zeile als Programmzeile interpretiert und abgespeichert. Bei der Zeilennumerierung geht man zweckmäßigerweise meist in Zehnerschritten vor, so daß dann noch genügend Zeilen für eventuelle Korrekturen oder Erweiterungen frei sind. Alle anderen Eingaben werden als Befehle zur direkten Ausführung angesehen. Sind sie zulässig, werden sie ausgeführt, sonst erfolgt eine Fehlermeldung. Danach erwartet der Interpreter die nächste Eingabe. Innerhalb einer einzugebenden Zeile kann beliebig oft mit der Backspace-Taste (Rückschritt) korrigiert werden. Soll eine bereits im Programmspeicher abgelegte Zeile verändert werden, so ist sie neu einzugeben. Soll sie gelöscht werden, so ist nur die Zeilennummer einzugeben.

Alle Befehle bzw. Schlüsselworte kann man mit einem Punkt nach dem ersten oder weiteren Buchstaben abkürzen. Bei mehreren Schlüsselworten mit gleichem Anfangsbuchstaben wird das genommen, das zuerst in Tabelle 7 aufgeführt ist. Die anderen sind dann soweit abzukürzen, daß keine Verwechselung mehr möglich ist. Auch die Leerzeichen zwischen den einzelnen Elementen einer Programmzeile können weggelassen werden. Diese beiden Möglichkeiten gestatten es, den Programmspeicher optimal auszunutzen, sie gehen aber auf Kosten der Übersichtlichkeit. Es dürfen auch mehrere Anweisungen in einer Programmzeile untergebracht werden, diese sind dann durch ein Semikolon voneinander zu trennen.

Der Mini-BASIC-Interpreter realisiert eine einfache Ganzzahl-Arithmetik in den vier Grundrechenarten Addition, Subtraktion, Multiplikation und Division (+ - * /) im Zahlenbereich -32768 bis +32767. Den meisten Schlüsselworten folgt ein weiterer Ausdruck. Solche Ausdrücke sind Zahlen, Variable oder arithmetische Konstruktionen mit diesen, die mittels Klammem aufgestellt werden können. Die Klammern können dabei beliebig geschachtelt werden. Innerhalb eines Befehls kann ein weiterer

Ausdruck bzw. Argument verwendet werden.

Als Variable sind alle Buchstaben von A bis Z erlaubt. Mit dem Symbol @ bzw. ÿ ist die Nutzung eines eindimensionalen Feldes möglich, wobei @ (I) den I-ten Wert des Feldes darstellt. I steht hierbei für einen Ausdruck, wie oben erläutert. Der Mini-BASIC-Interpreter kennt drei Fehlermeldungen:

WHAT? - Das Schlüsselwort bzw. der Ausdruck sind nicht erlaubt bzw. fehlerhaft, d.h., der Interpreter versteht die Anweisung nicht.

HOW? - Die Ausführung der Anweisung ist im Rahmen der Möglichkeiten dieses Interpreters nicht möglich (z.B. Zahlenbereichsüberschreitung).

SORRY - Die Ausführung der Anweisung ist zwar möglich, aber nicht unter den gerade aktuellen Voraussetzungen (z.B. Programmspeicher erschöpft).

Tritt eine Fehlermeldung bei der Abarbeitung eines Programms auf, so wird nach der Fehlermeldung auch die Zeile ausgegeben, in der der Fehler auftrat, wobei an der entsprechenden Stelle vom Interpreter ein Fragezeichen gesetzt wird. Im Interpreter selbst befinden sich keine Befehle zum Abspeichern bzw. Einlesen von Programmen über das Kassetteninterface. Das läßt sich jedoch mit Hilfe eines Monitorprogramms ausführen, wie es im Beispiel noch gezeigt wird.

In Tabelle 7 (s. FA, Heft 1/1985) sind alle realisierten Befehle dargestellt, die im Text noch näher erläutert werden.

```
Erläuterung zum Mini-BASIC-Interpreter

Kommandos:

LIST
Auflisten des im Speicher stehenden BASIC-Programms auf dem Bildschirm,
beispielsweise zu Kontrollzwecken.
> LIST cr    gesamtes BASIC-Programm
> LIST 110 BASIC-Programm ab Zeile 110
Durch das Drücken einer beliebigen Taste kann das Listen bis zum Loslassen
dieser ausgeführt werden, CONTROL C bricht das Listen ab.

RUN
Startet ein BASIC-Programm, wobei die Abarbeitung bei der niedrigsten
Zeilennummer beginnt. Ist die letzte Zeile abgearbeitet oder der STOP-Befehl
erreicht, kehrt der Interpreter in die Eingabeschleife zurück. Wird während
der Programmabarbeitung CONTROL-C eingegeben, führt das zum Programmabbruch
und zur Rückkehr in die Eingabeschleife.

NEW
Ein im Speicher vorhandenes BASIC-Programm wird vollständig gelöscht.

BYE
Beendet die Arbeit mit dem BASIC-Interpreter und kehrt zum Monitorprogramm
zurück. Das zuletzt eingegebene BASIC-Programm befindet sich weiterhin im
Speicher. Wenn an dessen Inhalt nichts geändert wird, kann mit J 803 wieder
zum BASIC-Interpreter zurückgesprungen werden (Warmstart). Es erscheint dann
nur READY ohne Überschrift auf dem Bildschirm. Die Arbeit mit dem BASIC-
```

```
Interpreter kann nun fortgesetzt werden.

END
Dieser Befehl wird nicht, wie in BASIC sonst üblich, zur Kennzeichnung des
Programmendes genutzt, sondern zum Vergrößem des Programmspeichers. Mit END
kann das Programmspeicherende neu gesetzt werden. Lassen die aktuellen
Möglichkeiten des Amateurcomputers diesen Speicherbedarf nicht zu, weil er
real nicht existiert, wird die Fehlermeldung "SORRY" ausgegeben. Beispiel:
> END HEX(1FFF)

Programmierbare Befehle (Anweisungen):

LET Definiert den Anfang einer Ergibtanweisung, muß aber nicht mit
vorangestellt werden. Es dient eigentlich nur der besseren Verständlichkeit.
> 100 LET A=1
> 110 A=50;B=30
> 120 A=B=C+3
> 130 X=3*A+(B-3)/C
> 140 LET@(3)=24

IF
Mit Hilfe von IF lassen sich Bedingungen für das Ausführen von Anweisungen
formulieren, also auch Verzweigungen aufbauen.
>100 B=B-1;IF B=0 GOTO 150
>105 IF C=0 PRINT"FERTIG"

GOTO
Unbedingter Sprung zu der Zeile, die entweder direkt angegeben ist, oder
sich aus dem angegebenen Ausdruck berechnet. GOTO als Direktbefehl startet
das Programm ab der angegebenen Zeilennummer. In Verbindung mit IF wird GOTO
zur Konstruktion bedingter Sprünge benutzt.
> 100 GOTO 180
> 104 GOTO 180 + X
> 108 GOTO A
> 110 IF A>0 GOTO 50

GOSUB ... RETURN
Aufruf eines in BASIC geschriebenen Unterprogramms, das mit RETURN beendet
werden muß. Nach RETURN wird mit dem dem Unter-Programmaufruf folgenden
Befehl im BASIC-Programm fortgesetzt. Unterprogramme sind vor allem da
sinnvoll, wo gleiche Teilprogramme an verschiedenen Stellen benötigt werden,
da dann Speicherplatz gespart werden kann.
> 100 C=20;GOSUB 200;PRINT"ZEIT",
> 105 C=50;GOSUB 200;PRINT"SCHLEIFE"
:
> 200 REM ZEITSCHLEIFE
> 210 C=C-1;IF C#0 GOTO 210
> 220 RETURN

REM
Damit ist es möglich, in ein BASIC-Programm Kommentare zur besseren
```

```
Dokumentation zu setzen. Diese Kommentare werden bei der Abarbeitung durch
den Interpreter überlesen, sie belegen aber entsprechend ihrer Länge
Speicherplatz.
> 100 REM LET A=1234
(Diese Zeile wird nicht verarbeitet!)
> 110 REM + + + TEILPROGRAMM 1 + + +

INPUT
Ermöglicht die Eingabe von numerischen Werten, die einer Variablen
zugeordnet werden. Alle eingegebenen Zeichen werden auf dem Schirm als Echo
ausgegeben, mit der Backspace-Taste kann korrigiert werden. Die Eingabe ist
mit cr abzuschließen. Nach der INPUT-Anweisung kann in Hochkomma oder mit
Anführungszeichen geschriebener Text folgen, der dann bei der Ausführung der
INPUT-Anweisung auf dem Schirm mit ausgegeben wird. Es können mehrere
Eingaben mit einer INPUT-Anweisung erledigt werden. Bei der eigentlichen
Eingabe kann anstatt einer Zahl auch ein Ausdruck angegeben werden.
> 100 INPUT A
> 105 INPUT"BEISPIEL"B,C
> 110 INPUT"WERT",D
> RUN

A:
BEISPIEL:
C:
WERT D:

PRINT
Ermöglicht die Ausgabe von numerischen Werten und von Texten auf dem
Bildschirm. Texte sind dabei in Hochkomma oder Anführungszeichen zu setzen.
Innerhalb einer Anweisung können mehrere Ausdrücke stehen, die dann durch
Komma voneinander zu trennen sind. Wenn nicht anders angewiesen, werden
Zahlen sechsstellig mit unterdrückten Vornullen rechtsbündig ausgegeben.
Wird in die PRINT-Anweisung ein Doppelkreuz, gefolgt von einer Zahl,
gebracht, so läßt sich damit diese Formatierung ändern. Die Änderung bleibt
bis zur nächsten PRINT-Anweisung bestehen. Ist die PRINT-Anweisung mit einem
Komma beendet, wird crlf (Wagenrücklauf/Zeilenvorschub) unterdrückt, d.h.,
die nächste Ausgabe setzt in der gleichen Zeile fort.
> 100 A=2;B=10;C=123
> 110 PRINT"ZAHL A=",A
> 120 PRINT"ZAHL A=",#1,A
> 130 PRINT A,B,
> 140 PRINT C
>RUN
ZAHL A=   2
ZAHL A=2
     2    10   123

STOP Beendet die Abarbeitung des BASIC-Programms und kehrt in die
Eingabeschleife des Interpreters zurück.
> 100 A=1
> 110 STOP
```

```
> 120 A=2
> RUN
READY
> PRINT A
     1
```

CALL
Damit ist es möglich, in Maschinenkode geschriebene Unterprogramme in BASIC aufzurufen. Die Adresse des anzuspringenden Unterprogramms ergibt sich aus dem Ausdruck, der dem CALL folgt. Soll der Interpreter danach mit der nächsten BASIC-Anweisung fortsetzen, so muß das Unterprogramm mit einem RETURN-Befehl (z.B. 0C9H) enden. Während der Abarbeitung des Unterprogramms ist kein Abbruch des Interpreters mit CONTROL-C möglich.

```
> 100 CALL HEX(4000)
.
.
.

4000H   LD A,(4020H)
INC A
LD (4020H),A
RET
```

Durch die CALL-Anweisung in Zeile 100 wird der Inhalt der Speicherzelle 4020H um Eins erhöht. Auf diese Zelle könnte dann z.B. mit PEEK zurückgegriffen werden.

OUTCHAR Das dem nachfolgendem Ausdruck entsprechende ASCII-Zeichen wird auf dem Schirm ausgegeben.

```
> 100 OUTCHAR 65
> 110 B=66
> 120 OUTCHAR B
> 130 C='C'
> 140 OUTCHAR C
> 150 PRINT C
> RUN
ABC
67
```

INCHAR
Ermöglicht die Eingabe eines einzelnen Zeichens, ohne Echo und ohne cr. Der Wert des eingegebenen ASCII-Zeichens wird der angegebenen Variablen zugewiesen.

```
> 100 Z=INCHAR
```

POKE
Mit POKE läßt sich eine Speicherzelle beschreiben. Der erste Ausdruck bestimmt die Adresse, der zweite den Wert. Der Datenwert wird modulo 256 berechnet und auf die entsprechende Adresse geschrieben.

```
> 100 POKE 7167,10
```

Der dezimale Wert 10 wird unter der Adresse 7167 (=1BFFH) gespeichert.

```
TAB
TAB dient der Ausgabe von Leerzeichen, deren Anzahl sich aus dem
nachfolgenden Ausdruck ergibt.
> 100 PRINT"TEST"
> 110 TAB(10)
> 120 PRINT"TAB"
>RUN
TEST          TAB

BYTE
BYTE gibt den Wert des nachfolgenden Ausdrucks als Hexadezimalzahl auf dem
Schirm aus. Dabei werden nur dezimale Werte bis 255 in Form zweier
hexadezimaler Stellen ausgegeben.
> BYTE(17)
11

WORD
WORD funktioniert analog BYTE, nur werden hier vier Stellen ausgegeben.
> 100 N=2000
> 110 WORD(N)
> RUN
07D0

OUT
Gibt den angegebenen Wert an die zugewiesene E/A-Adresse des U-880-Systems
aus.
> 10 OUT(HEX(0F)=5
Das Bitmuster für 5 wird an die Systemadresse 0FH ausgegeben.


RND
RND erzeugt eine Zufallszahl, die zwischen 1 und dem Wert des nachfolgenden
Ausdrucks liegt.
> 100 A=RND(10)
Weist der Variablen eine Zahl zwischen 1 und 10 zu.
> PRINT RND(1000)
   723

ABS
Der absolute Betrag des nachfolgenden Ausdrucks wird gebildet.
> 100 A=ABS(-245)

SIZE
Mit diesem Befehl wird der aktuelle, für ein BASIC-Programm noch freie
Speicherplatz bestimmt und SIZE zugewiesen, so daß diese Information
weiterverwendet werden kann.
> 100 PRINT SIZE,"BYTES FREE"
> RUN
   590 BYTES FREE

PEEK
```

```
Hiermit ist der direkte Speicherzugriff möglich. PEEK erhält als Parameter
eine Adresse, von welcher ein Byte gelesen und als Wert übergeben wird.
> 100 A=PEEK(7167)

HEX
HEX ermöglicht eine Hexadezimal/Dezimal-Konvertierung.
> 100 A=HEX(4AF2)
> 110 PRINT A
> RUN
   19186

' ' (QUOTE)
Der durch '  ' dargestellte Befehl QUOTE gestattet die Zuweisung der
Dezimalzahl, der das jeweilige angegebene ASCII-Zeichen entspricht.
>100 X='A'
>110 PRINT X
> RUN
   65

TOP
Beim Aufruf wird der gerade aktuelle erste freie Speicherplatz hinter dem
eingegebenen BASIC-Programm berechnet.
> PRINT TOP
   6481

IN
Damit können Variablenwerte von einer E/A-Adresse des U-880-Systems
zugewiesen werden.
> 10 A=IN(HEX(0F))
Der Variablen A wird der Wert, der dem Bitmuster, das an der Systemadresse
0FH anliegt, zugewiesen.

Beispiele

Abspeichern eines BASIC-Programms
>WORD(TOP)
1AF9
>BYE
#S 18C0 1AF9 803 TEST.B
Beim Abspeichern von BASIC-Programmen haben das erste und das dritte
Argument immer die hier angegebenen Werte. Nur das zweite Argument muß mit
Hilfe von TOP immer neu bestimmt werden, entsprechend der jeweiligen
Programmlänge. Ein so abgespeichertes Programm wird dann einfach mit L cr
geladen, und mit J: angesprungen.

Bildschirmlöschen
100 OUTCHAR 12

Parabel zeichnen
100 N=-6
110 PRINT N, "I",
```

```
120 TAB (N*N)
130 PRINT "*"
140 N=N+1; IF N<7 GOTO 110
(150 STOP)
```

Ersatz der Anweisung FOR..TO..STEP

```
100 REM FOR I=xx TO yy STEP zz
110 I=xx; REM Anfangswert xx
120 REM Anweisungen der Schleife
130 I=I + zz; IF I < xx + 1 GOTO 120
```

Symbolumwandlung

```
10 PRINT "SYMBOL:",
20 Z=INCHAR
30 OUTCHAR(Z)
40 PRINT" DEZIMAL:", #2,Z," HEXADEZIMAL:",
50 BYTE(Z)
60 PRINT
70 GOTO 10
```

Zeichengenerator-Testprogramm

```
10 REM ZEICHENGENERATORTEST
20 OUTCHAR 12
30 PRINT
40 TAB(20)
50 PRINT"TESTPROGRAMM"
60 PRINT
70 A=' '; REM LEERZEICHEN
80 OUTCHAR A; A=A+1
90 IF A#HEX(60) GOTO 80
100 GOTO 70; REM Zeile fertig
```

Hexdump mit ASCII-Interpretation

```
200 INPUT "ANFANGSADRESSE" A, "ENDADRESSE" E
210 I=A
220 WORD(I); PRINT" ",
230 J=0
240 BYTE(PEEK(I+J)); PRINT" ",
250 J=J+l; IF J < 8 GOTO 240
260 PRINT"*",; K=0
270 S=PEEK(I+K)
280 IF S<32 GOTO 300
290 IF S<127 GOTO 310
300 S='.'
310 OUTCHAR (S); K=K+l; IF K < 8 GOTO 270
320 PRINT"*"
330 I=I+8; IF I < E+l GOTO 220
340 PRINT; GOTO 200
```

From:
https://hc-ddr.hucki.net/wiki/ - **Homecomputer DDR**

Permanent link:
https://hc-ddr.hucki.net/wiki/doku.php/homecomputer/ac1/minibasic

Last update: **2019/06/06 07:55**