POLYMORPHIC SYSTEMS

16K BASIC

Preliminary Version

This manual represents a preliminary and interim document describing briefly the features and capabilities of Poly 88 BASIC. It is applicable to versions of Poly 88 BASIC up to and including V24 (the version number is displayed at the top of the screen when BASIC starts). This manual is not for teaching the beginning user BASIC, or computer programming; it is a summary of the features and capabilities provided by Poly 88 BASIC. For information and books on BASIC, and computer programming, visit your local computer store.

Computer software has two fundamental properties: it evolves, and it may not be free of errors. BASIC for the Poly 88 is an evolving system; it is changing and growing in capabiltities, and for that reason, future versions of BASIC will probably have features not present in this version. We make the commitment that all these changes will be upward-compatible; the programs you write for Poly 88 BASIC today you will be able to run on future versions. Every large software system may contain errors; this alone is a reason for new versions of the product, fixing errors. As we find errors in BASIC, or are made aware of them, they will be fixed in future versions. If you think you have found an error in BASIC, first try it out on another Poly 88 to eliminate the possibility that the error is caused by hardware problems such as memory or a marginal cassette. If you wish to inform us of a problem in BASIC, please send us all the information on the problem that you can - listings, descriptions, and a cassette of the program that causes the problem. If we cannot reproduce the problem, we can't find or fix it. We hope that because of our testing, each version of BASIC is as free from errors as we can make it.

INPUTTING A PROGRAM:

Every program line begins with a line number. Any line of text typed to BASIC in command mode that begins with a digit is processed by the editor. There are four possible actions which may occur:

1) A new line is added to the program. This occurs if the line number is legal (range is Ø thru 65535) and at least one character follows the line number in the line.

2) An existing line is modified. This occurs if the line number matches the line number of an existing line in the program. That line is modified to have the text of the newly typed-in line.

3) An existing line is deleted. This occurs if the typed-in line contains only a line number which matches an existing line in the program.

4) An error is generated. If the line number is out of range, or the line is too long, or the memory would become full, then error messages are generated and no other action is taken by BASIC.

Blanks:

Blanks preceding a line number are ignored. The first non-digit in a line terminates the line number (even blanks). Multiple blanks are permitted anywhere in a line for indentation purposes, but not within reserved words or constants.

Multiple statements per line :

Multiple program statements may appear on a single line, if separated by a (\) back slash. A line number must appear only at the beginning of the first statement on the line.

Typing mistakes:

If a typing mistake occurs during the entering of any line of

text to BASIC, there are two possible corrective actions available:

When the user types control-x, BASIC completely ignores all input on the current line being typed in, and types a carriage return. The correct line may then be typed to BASIC.

When the user types a del, or rubout, BASIC will ignore the previously typed character.

COMMANDS

RUN <optional line number>

Begin program execution either at the first line of the program or else at the optionally supplied line number.

LIST <optional line number>, <optional second line number>

If no arguments are supplied, then print the entire existing program. If one line number is supplied, then print the program from the specified line number to the end. If two line numbers are supplied, then print the program in the region between the two line numbers.

SCR

Delete (scratch) the existing program and data, in preparation for entering a new program.

REN <optional beginning value>,<optional increment value>

Renumber the entire existing program. If the first argument is not supplied, then 10 is used as the initial statement renumber value. If the second argument is not supplied, then 10 is used as the increment value.

CLEAR

Clear all variables. This command deletes all arrays, strings and functions, and initializes all scalar variables to zero.

CONT

This command causes execution of a running BASIC program to continue after a STOP statement or after an escape stop.

LINE <number of characters>

This command defines the line length of the user terminal. No input line will be accepted longer than the specified value, and no output line will be printed longer than the specified value. The maximum value is 132. The initial value is 72.

NULL <number of nulls>

This command specifies the number of ACSII null characters to output following the output of each carriage return character. The initial value is zero.

SAVE and LOAD

SAVE and LOAD allow the storage and retrieval of named files on cassette tape. SAVE is used to save a program on tape, and LOAD is used to load the program into memory from tape. Programs may be saved in either "byte" or "Polyphase" format, and must be loaded in the same mode as they were written. The syntax for SAVE and LOAD commands are:

```
>SAVE,B,NAME          save program in "byte" format
>SAVE,P,NAME          save program in "Polyphase" format

>LOAD,B,NAME          load program in "byte" format
>LOAD,P,NAME          load program in "Polyphase" format
```

NAME is the name of the file, and may be from one to eight characters in length. Note that in the lines above illustrating the syntax, the ">" is the prompt given by BASIC and is not typed by the user. When saving or loading programs, BASIC types "Working..." to indicate that the process has begun, and types out the record number of the current tape record as it is loaded or written. A "Syntax" error may be generated if the mode specified is not "B" or "P", the file name is too long or not given, or commas are omitted. If an error occurs reading the tape, "Checksum error" is generated. The SAVE or LOAD process may be interrupted by typing the ESCAPE character (see page)

CONSTANTS:

Magnitude range: .1E-63 thru .99999999E+63

Constants appearing in programs are rounded to 8 digits if necessary. Internal representation of numbers is binary-coded-decimal.

NAMES:

All user defined names are one or two characters long: a letter of the alphabet optionally followed by any digit. For example: A, Z0, and Q9 are legal names. The same name may be used to identify different values, as long as the values they identify are of different types. For example, it is possible to have a scalar variable named A1, an array named A1, a string named A1$ and functions named FNA1 and FNA1$. There is no relationship between these entities.

OPERATORS:

Numeric: +, -, /, *, ↑ (or " on some keyboards)

Relational: =, <, >, <>, >=, =>, <=, =<

A relational operation gives a 1 (true) or 0 (false) result.

Boolean: AND, OR, NOT

A Boolean operand is true if non-zero, and false if zero.
The result of a boolean operation is 1 or 0.

STATEMENTS:

Only some statements listed below are accompanied by discussion. Consult the example programs in Appendix 1 for questions about the use of a particular type of statement.

LET

The LET is optional in assignment statements. Multiple assignments are not allowed. The statement A=B=Ø assigns true or false to A depending on whether or not B equals Ø.

IF

An IF statement may optionally have an ELSE clause. A THEN or ELSE clause may be a LET statement, a RETURN statement, another IF statement or a GOTO, for example. If either the THEN clause or the ELSE clause is a simple GOTO, then the GOTO reserved word may be optionally omitted.

```
1ØØ  IF A=B THEN 15Ø ELSE A=A-1
```

FOR

FOR loops may be multiply nested. The optional STEP value may be positive or negative. It is possible to specify values such that the FOR loop will execute zero times. For example,

```
1ØØ  FOR J=5 TO 4 \ PRINT J \NEXT
```

NEXT

A NEXT statement may optionally specify the control variable for the matching FOR statement, as a check for proper nesting.

GOTO

ON

The ON statement provides a multi-branched GOTO capability.

For example,

```
100 ON J GOTO 500,600,700
```

will branch to 500, 600 or 700 depending on the value of J being 1,2, or 3 respectively.

EXIT

The EXIT statement is identical to a GOTO except that it has the effect of terminating any active FOR loops and reclaiming the associated internal stack memory. It should be used for branching out of a FOR loop.

STOP

END

REM

READ

DATA

RESTORE

The RESTORE statement may optionally include a line number, specifying where the READ pointer is to be restored to. In the absence of the optional line number, the READ pointer is set to the first line of the program.

INPUT
INPUT1

The INPUT or INPUT1 statement may optionally specify a literal string which is typed on the terminal as a prompt for the input instead of a question mark. To inhibit the echoing of the carriage return at the end of user input, use the INPUT1 statement.

```
100 INPUT1 "TYPE VALUE:",V
```

GOSUB

RETURN

PRINT

The PRINT statement may include a list of expressions, variables, or constants separated by (,) commas. Note that if the list of variables is terminated by a comma, then a carriage return is not typed. The null PRINT statement will cause only a carriage return to be typed. A semi-colon is equivalent to a comma in the print list. All values are printed in free format, separated by a blank, unless formatting is specified. If a value will not fit on the current output line, then it is printed on the next output line. Advancement of the printer to a specified output position may be accomplished with the TAB function. Formatting may be accomplished by including a "format string" in a print statement (see below).

FILL

This statement permits filling a specified byte in the computer memory with a given expression value. For example, FILL 1ØØ,J+3 will fill memory byte 1ØØ with J+3.

OUT

This instruction permits doing an 8Ø8Ø OUT instruction. For example, OUT 5,3 will perform an OUT 5 instruction with 3 in the 8Ø8Ø accumulator.

PLOT

The PLOT statement allows the user to display data on the video display as a 128 by 48 grid. The "origin" for the display grid is the lower left corner of the display, and is addressed as point (Ø,Ø). The x coordinate (left-right) must be in the range Ø <= <=127, and the y coordinate in the range Ø <= y <= 47 or "Out of bounds" error will result. The syntax for the PLOT statement is;

PLOT xexpr,yexpr,zexpr

Where xexpr and yexpr are arbitrary expressions selecting the x and y coordinates, in the ranges mentioned above and zexpr is an arbitrary expression that will plot a bright spot if odd, and a dark spot if even. As a side effect, the PLOT statement causes cursor position to move, so that the next PRINT or INPUT statement will appear at theat cursor position. This is useful for formating the screen, or displaying text in different positions on the screen. See the sample programs in Appendix 1 for expamples of PLOT useage.

ARRAYS:

Arrays may be dimensioned with any number of dimensions, limited only by available memory, e.g.,

1ØØ DIM A(1), B7(5,2,3,4,5,6)

Array indexing starts at element Ø. Array A in the above example actually has two elements, A(Ø) and A(1). Use of an undimensioned array causes automatic dimensioning to a one-dimension, 1Ø element array. Arrays may not be re-dimensioned within a program.

STRINGS:

Strings of 8-bit characters may be dimensioned to any size, limited only by available memory, e.g.,

```
10Ø DIM A$(1),A1$(1ØØØØ)
```

Note that a string name is a variable name followed by a ($) dollar sign. Substrings may be accessed as A$(N,M) which is the substring of characters N thru M. For example, if A$ is "ABCDEF" then A$(3,5) is "CDE". Alternatively, A$(N) identifies the substring including characters N thru the last character in the string. The concatenation operator is a plus sign.

If an assigned value is larger than the destination string or substring, then it is truncated to fit. If an assigned value to a substring is shorter than the substring, then the extra characters of the substring are left unmodified. A string variable used before being DIMensioned is given the default demension of 1Ø. Strings may not be redimensioned within a program.

Strings, substrings and string expressions may be used in conjunction with: LET, READ, DATA, PRINT, IF, and INPUT statements. The string IF statement does alphabetic comparisons when the relational operators are used, .e.g.

```
1ØØ  IF A$+B$ < "SMITH" THEN 5Ø
```

When string variables are INPUT, they must not be quoted. When strings appear in DATA statements, they must be quoted.

USER DEFINED FUNCTIONS:

User-defined functions (either of type string or numeric) may be 1-line or multiple line functions. There may be any number of numeric arguments. Parameters are "local" to a particular call of a function.

That is, the value of the variable is not affected outside of the execution of the function.

Functions are defined before execution begins (at RUN time), so definitions need not be executed, and functions may be defined only once.

Multiple line functions must end with a FNEND statement. A multiple-line function returns a value by executing a RETURN statement with the value to be returned, for example:

```
100 DEF  FNA(X,Y,Z)
200 IF Z=1 THEN RETURN X
300 X=Y*Z+X*3
400 RETURN X
500 FNEND
600 PRINT FNA(1,2,X+Y)
```

BUILT IN FUNCTIONS:

```
FREE(0)      returns number of bytes remaining in free storage.
ABS(expr)    returns the absolute value of the expression
SGN(expr)    returns 1, 0, or -1 if the value is +, 0, or -
INT(expr)    returns the integer portion of the expression value
LEN(string name)      returns the length of the specified string
CHR$(expr)      returns a string with the specified character
VAL(string expr)    returns the numeric value of the string
STR$(expr)   returns a string with the specified numeric value
ASC(string name) returns ASCII code of first character in string
SIN(expr)   returns SINE of the expression
COS(expr)   returns the COSINE of the expression
RND(expr)   returns a random number between 0 and 1
LOG(expr)   returns the natural log of the expression
EXP(expr)   returns the value of e raised to the specified power
SQRT(expr)   returns the positive square root of the expression
CALL(expr, optional expr)  see below
TIME(expr) returns value of real-time clock, see below
```

EXAM(expr) return contents of addressed memory byte
INP(expr) return result of 8Ø8Ø IN to specified port

MACHINE LANGUAGE SUBROUTINE INTERFACING:

The built-in function CALL takes a first argument which is the address of a machine language subroutine to call. The optional second argument is a value which is converted to an integer and passed to the machine language subroutine in DE. The CALL function returns as value the integer which is in HL when the machine language subroutine returns.

TIME

The TIME function returns as its value the least significant 16 bits of the Poly 88 real-time clock, which is incremented every 1/6Ø of a second The syntax used for the TIME function is:

TIME(expr)

The argument (expr) is required, and if the expression evaluates to zero, the TIME function returns the current value of the timer, and then sets the timer to zero; this is useful for recording eleapsed times. If the expressions is non-zero, it is ignored. Note that since only the least 16 significant bits of the timer are returned, the value returned by the TIME function will cycle every (2↑16)/6Ø = 1Ø92 seconds = 18.2 minutes. Longer timing periods may be measured using the EXAM and FILL features to manipulate the most significant bytes of the real-time clock. See the sample programs in Appendix 1 for examples.

FORMATTED OUTPUT:

If no format string is present in a PRINT statement, then all numeric values will be printed in the "default format". (The default format is initially set to be free format.) A format string appears anywhere in the print list and must begin with a per cent (%) character, e.g.

PRINT %$1ØF3,J

A format string consists of optional format characters followed optionally by a format specification. The format characters are:

C place commas to the left of decimal point as needed
$ put a dollar sign to the left of value
Z suppress trailing zeroes
make this format string the default specification

Format specifications (similar to FORTRAN) are:

nFm F-format. The value will be printed in a n-character field, right justified, with m digits to the right of decimal point.

nI I-format. The value will be printed in a n-character field, right justified, if it is an integer. (Otherwise an error message will occur.)

nEM E-format. The value will be printed in scientific notation in a n-character field, right justified, with m digits to the right of the decimal point.

All printed values are rounded if necessary. A null format string will print values in free format.

ESCAPE

Typing the escape character (control-shift K) has the effect of prematurely interrupting BASIC from whatever it is doing. If a LIST is in progress, the listing will be terminated at the completion of the output of the current line. If a RUN or CONT is in progress, then execution will stop after the completion of the currently executing statement.

DIRECT STATEMENTS:

When BASIC is in command mode, certain statements may be typed for immediate execution. This is typically used for examining the values of certain variables to diagnose a programming error. Note that an exclamation point (!) may be used as a shorthand way of typing the PRINT reserved word. No direct statement is permitted which transfers control to the BASIC program. Also, DATA, DEF, FOR, NEXT, INPUT, and REM are forbidden.

Appendix 1: SAMPLE PROGRAMS.

```
100  REM  A NUMERIC SORT PROGRAM
110  REM
120  DIM A(15)
130  PRINT "INPUT FIFTEEN VALUES, ONE VALUE PER LINE"
140  FOR J=1 TO 15
150  INPUT A(J)
160  NEXT
170  REM  DO EXCHANGE SORT UNTIL ALL IN ORDER
175  F=0  \  REM THIS FLAG USED TO SIGNAL WHETHER ARRAY IN ORDER YET
180  FOR J=2 TO 15
190  IF A(J-1)<=A(J) THEN 220
200  T=A(J)\A(J)=A(J-1)\A(J-1)=T\REM EXCHANGE A(J) AND A(J-1)
210  F=1 \ REM SET FLAG
220  NEXT
230  IF F=1 THEN 175\REM  LOOP IF EXCHANGES HAPPENED
240  PRINT "SORTED ARRAY:",
250  FOR J=1 TO 15\PRINT A(J), \ NEXT

100  REM   CHARACTER SORT
110  REM   EXAMPLE USING STRINGS AND FUNCTION
115  DIM A$(72)
120  INPUT "TYPE A STRING OF CHARACTERS" ",A$
130  IF LEN(A$)=0 THEN 120
140  IF FNA(LEN(A$))=1 THEN 140\REM CALL FNA UNTIL IT RETURNS ZERO VALUE
150  PRINT "SORTED ARRAY: ",A$
155  END
160  DEF FNA(N)\REM CHARACTER SORT
170  REM  RETURN 0  IF A$ SORTED, ELSE RETURN 1
175  F=0
180  FOR J=2 TO N
190  IF A$(J-1,J-1)<=A$(J,J)  THEN 220
```

```
200  T$=A$(J,J)  A$(J,J)=A$(J-1,J-1)  A(J-1,J-1)=T$
210  F=1
220  NEXT
230  RETURN F
240  FNEND

100  REM INPUT A STRING AND CHECK THAT IT IS A LEGAL INTEGER
105  REM
110  DIM A$(72)
115  INPUT "TYPE AN INTEGER: ",A$
120  IF LEN(A$)=0 OR LEN(A$)=8 THEN GOTO 500
130  FOR J=1 TO LEN(A$)
140  IF A$(J,J)< "0" THEN 500
145  IF A$(J,J)> "9"  THEN 500
150  NEXT J
155  PRINT "THE INTEGER IS OK: ",VAL(A$)
160  GOTO 115
500  REM
510  PRINT "NOT AN INTEGER!"
520  GOTO 115
```

```
100 REM THIS PROGRAM PLOTS A FUNCTION ON THE VIDEO SCREEN
110 REM USING THE PLOT FEATURE OF BASIC
120 PRINT CHR$(12), \ REM CLEAR THE SCREEN
130 X=0
140 FOR I=0 TO 3*3.1415 STEP 3*3.1415/127
150 PLOT X,FNA(I),1 \ REM PLOT THE POINT
160 X=X+1 \ NEXT
170 PLOT 0,4,0 \ REM PLACE CURSOR SECOND LINE FROM BOTTOM
180 STOP
190 REM PLACE YOUR FUNCTION HERE AS FNA(X) OR WHATEVER-
200 REM IT SHOULD RETURN Y SUCH THAT 0 <= Y <= 47 FOR
210 REM AN INPUT ARGUMENT 0 <= X <= 3*PI
220 THE FOLLOWING FUNCTION GENERATES THE "BOUNCING BALL"
230 DEF FNA(X)=47*ABS(COS(X))*EXP(-X/10)


100 REM "TICK-TOCK" DISPLAY USING REAL-TIME CLOCK
110 REM NOTE- THIS DISPLAY REALLY IRRITATES SOME PEOPLE!
120 A$="TICK!" \ B$="TOCK!" \ REM INITIAL STRINGS
130 S=TIME(0) \ REM ZERO THE CLOCK, DONT CARE ABOUT S.
140 IF TIME(1)<30 THEN 140 \ REM WAIT 1/2 SECOND
150 PRINT TAB(K),A$ \ REM PRINT SOMETHING
160 T$=A$ \ A$=B$ \ B$=T$ \ REM SWAP A$ AND B$
170 IF K=20 THEN K=40 ELSE K=20
180 GOTO 130 \ REM KEEP TICKING

100 REM THE OLD RANDOM WALK ON THE SCREEN....
110 PRINT CHR$(12), \ REM PRINT FORM FEED TO CLEAR THE SCREEN
120 H=64 \ V=23 \ REM INITIAL X AND Y POSITIONS
130 A=TIME(1) \ A=A-2*INT(A/2) \ REM RANDOM 1 OR 0
140 B=INT(RND(A)*3-1) \ C=INT(RND(A)*3)-1
150 IF A THEN B=-B ELSE C=-C
160 IF H+B=128 OR H+B=0 THEN B=-B
170 IF V+C=48 OR C+V=0 THEN C=-C
180 H=H+B \ V=V+C\ PLOT H,V,A \ GOTO 130
190 REM B AND C ARE RANDOM X AND Y MOVES, EITHER (-1,0,1)
200 REM STATEMENTS ARE SQUASHED ONTO ONE LINE TO SPEED
210 REM THINGS UP!
```

Appendix II

This appendix deals with two topics: restarting BASIC, and changing the memory limits set in BASIC. These topics are mainly of interest to the programmer who wishes to connect interface assembly language routines to BASIC using the CALL interface.

RESTARTING BASIC

In the event of a catastrophic error in BASIC, it may be necessary to restart BASIC. The starting address for BASIC is (in hexadecimal) 2000. Starting the program at this address results in what is known as a "cold start" - ANY BASIC PROGRAM PREVIOUSLY LOADED WILL BE LOST WHEN BASIC IS RESTARTED AT ADDRESS 2000. To restart BASIC without losing the loaded program, the "warm start", or restart address (in hexadecimal) is 2003. When restarted at 2003, any BASIC program will be retained, but the contents of all variables, arrays, strings, etc., will be lost. Usually it is only necessary to resort to restarting BASIC when an attached assembly language program has run away, or some part of the BASIC system itself has been damaged through use of the FILL statement, or intermittant memory. If you think that the copy of BASIC you have in memory is not well, it is usually advisable to save the program you are working on cassette, and then reload BASIC and your program.

CHANGING MEMORY LIMITS

This section is of interest mainly to those who wish to interface assembly language routines to BASIC, or to change the memory limits used by BASIC to allow it to use more or less memory. When BASIC is either "cold started" or "warm started" (see above), it sets the default memory limits it will use for storage of both programs and data. These initial low and high limits are assembled into the program as the following two assembly language instructions at the locations shown:

```
2012 21PPQQ     LXI  H,QQPPH      ; first free byte
2015 11FF5F     LXI  D,5FFFH      ; end of memory
```

Note that the address given as "QQPP" is representative, and changes with each version of BASIC. The address 5FFF assumes the existance of 16K of memory starting at (hexadecimal) 2ØØØ. Changing the addresses in these two instructions, then, will change the amount of memory available to BASIC. If, for example, a user has 24K of memory instead of 16K, then the instruction at location 2Ø15 would be changed from 11FF5F using the front panel mode described in the Poly 88 user's manual, and restarting BASIC at either 2ØØØ or 2ØØ3 (hexadecimal). For interfacing assembly language programs, the user may examine locations 2Ø12-2Ø14 (again, using the front panel mode described in the user's manual) to determine the starting data address for this version of BASIC. The user then assembles the assembly language program to start after this address when running the combined program, first load BASIC from cassette. After BASIC has been loaded, return to the tape boot section of the monitor (by hitting the RESET button on the Poly 88), and load the assembly language program into memory. When the assemb language program is loaded, using the front panel mode, change the first free byte of memory pointer in location 2Ø12-21Ø4 of BASIC to point past the end of the assembly language program. BASIC may then be started at either 2ØØØ or 2ØØ3

Preliminary specifications for
PolyMorphic Systems BASIC, subject to change.
16K

Size: 10K bytes
Scientific functions: sine, cosine, log, exponential, square root, random number, x to the y power
Formatted output
Multi-line function definition
String manipulation and string functions
Real-time clock
Point-plotting on video display
Arrays of up to 7 dimensions
Cassette save and load of named programs
Multiple statements per line
Renumber
Memory load and store
8080 input and output
IF THEN ELSE

Commands:
RUN LIST SCR CLEAR REN CONT LINE NULL
Statements:
LET IF THEN ELSE FOR NEXT GOTO ON EXIT STOP END REM
READ DATA RESTORE INPUT GOSUB RETURN PRINT FILL OUT
Built in functions:
FREE ABS SGN INT LEN CHR$ VAL STR$ ASC SIN COS RND LOG TIME
EXP SQRT CALL EXAM INP PLOT

Preliminary specifications for

PolyMorphic Systems BASIC, subject to change.

8K

Size: 7½K bytes

Scientific functions: sine, cosine, square root, random number, x to the y power (y=integer)

Arrays of up to 7 dimensions

Cassette save and load of named programs

Multiple statements per line

IF THEN ELSE

Commands:

RUN LIST SCR CLEAR CONT

Statements:

LET IF THEN ELSE FOR NEXT GOTO ON STOP END REM READ DATA RESTORE INPUT GOSUB RETURN PRINT

Built in functions:

FREE ABS SGN INT SIN COS RND SQRT

Here are some better known books on BASIC. You can find some or all of these at your local computer store. Most of these are available from Peoples Computer Company (PCC):

Community Computer Company
1919 Menalto Avenue
Menlo Park, CA 94025

BASIC, BASIC
by James S. Coan
Published by Hayden Books, Rochelle Park, NJ 1970

ADVANCED BASIC
by James S. Coan
Published by Hayden Books, Rochelle Park, NJ

BASIC, A SELF-TEACHING GUIDE
by Bob Albrecht

MY COMPUTER LIKES ME WHEN I SPEAK IN BASIC
by Bob Albrecht
Published by Dymax, Menlo Park, CA 1972

BASIC
by Bob Albrecht
Published by Findel, Brown 1973

101 BASIC COMPUTER GAMES
Edited by David Ahl
1974