

Chapter 1 - BASIC Programming Rules

When you turn on the Intuition Engine, you see the BASIC prompt:

```
Ready  
.
```

On a normal start, BASIC may first print a short memory line such as 64 GiB SYSTEM RAM, 4 GiB BASIC FREE. The exact numbers depend on the machine and active memory profile. The transcripts in this guide begin at Ready unless the memory line itself is being discussed.

This is **direct mode**. Anything you type is treated as a BASIC statement and runs as soon as you press RETURN. If you begin a line with a number, the rest of the line is stored in memory as part of a program; this is **program mode**. The same statements work in both modes, with a small number of exceptions noted later.

BASIC is the front panel of the Intuition Engine. The same language that prints numbers can also read memory, write hardware registers, draw pictures, start sound, load files, and call machine code. Later chapters use those powers to reach each card on the shared bus. This chapter gives you the rules you need before you start touching that hardware.

This chapter describes how BASIC stores and manipulates information. It tells you what kinds of values BASIC understands, how to name them, how to build expressions, and how to enter a program one line at a time. The keywords themselves are described one by one in Chapter 2.

First session

If this is your first time at the prompt, type these lines before reading the vocabulary chapter. They are not a full lesson in display, sound, or files. They show the shape of the machine: the same BASIC prompt can calculate, remember a program, draw through a video card, make sound through an audio engine, and save what you typed.

First try one direct-mode calculation:

```
PRINT 2+3  
5  
Ready  
.
```

Now enter a two-line program. Numbered lines are stored until you run, list, save, or clear them:

```
10 PRINT "INTUITION ENGINE"  
20 PRINT 2+3  
RUN  
INTUITION ENGINE  
5  
Ready  
.
```

BASIC can make native IE64 programs from inside the machine. Type this while the two-line program is still in memory:

```
RUN AOT
Compiling to native code...
INTUITION ENGINE
5
Ready
.
```

The printed result is the same. The difference is the path through the machine: BASIC compiles the stored lines into native IE64 instructions inside Intuition Engine, then starts those instructions.

Make one visible mark on the VGA card:

```
SCREEN &H13
PALETTE 1,63,0,0
PLOT 160,100,1
```

You should see a red pixel near the centre of the screen. Now make one sound through the SoundChip:

```
SOUND 0,440,200,0,128
```

You should hear a steady tone. Finally save the little program and ask the machine to list the files it can see:

```
SAVE "FIRST.BAS"
```

Then DIR (direct mode only) lists the filenames the machine can see. Its output depends on the current file area, so it is not shown here.

Later chapters explain every register behind those statements. For now, remember the important rule: BASIC is not just a calculator. It is the first way into the shared bus.

1.1 Direct mode and program mode

To use direct mode, type a statement at the prompt and press RETURN. For example:

```
PRINT 2+3
5
Ready
.
```

To enter a program, begin each line with a line number. BASIC keeps the lines in memory in ascending order of line number, no matter what order you type them in:

```
20 PRINT "WORLD"
10 PRINT "HELLO"
LIST
10 PRINT "HELLO"
20 PRINT "WORLD"
Ready
.
```

Type RUN to execute the stored program from the lowest-numbered line. Type LIST to see the program. Type NEW to clear the program from memory.

Line numbers are 32-bit unsigned integers, so any value from 0 to 4294967295 is legal. By convention, you number lines in steps of 10 (10, 20, 30, ...) so there is room to insert new lines later.

To replace a line, type the same line number with new text:

```
10 PRINT "HI"  
10 PRINT "HELLO"  
LIST  
10 PRINT "HELLO"
```

To delete a line, type its line number on its own:

```
10 PRINT "HELLO"  
20 PRINT "WORLD"  
10  
LIST  
20 PRINT "WORLD"
```

1.2 Statements, lines, and statement separators

A BASIC line is a sequence of statements separated by colons. Both of these are equivalent:

```
10 A=1:B=2:PRINT A+B  
20 A=1  
30 B=2  
40 PRINT A+B
```

There is no fixed limit on the number of statements per line beyond the size of the line buffer. A program line ends at the end of the typed line. There is no continuation character.

1.3 Numeric values

Every ordinary numeric calculation in BASIC uses IEEE double-precision floating point. There is **one** visible numeric type at the prompt. The same variable can hold 1, -1, 3.14159, or 1.23E+10.

The approximate range of a numeric value is from about 2.2×10^{-308} to about $1.8 \times 10^{+308}$, with about fifteen significant decimal digits of precision. Operations that need an integer (POKE, PEEK, CHR\$, ASC, array subscripts, etc.) truncate the value toward zero. The explicit 64-bit helpers such as PEEK64, POKE64, and MEMALLOC preserve exact integer payloads where the hardware interface requires them.

You may write numeric literals in four forms:

Form	Example
Plain decimal	42
Decimal with point	3.14159
Scientific notation	2.5E+10
Hexadecimal	&H710000

The &H prefix introduces a hexadecimal literal. The digits after &H are 0-9 and A-F (upper or lower case). Hexadecimal literals are useful for addresses and MMIO register values: PEEK (&H001F0000) and &HFF00 are equivalent to

PEEK(2031616) and 65280. There is no separate integer literal syntax. 42 and 42.0 are the same value.

1.4 Numeric variables

A numeric variable name is a sequence of letters and digits beginning with a letter. Letters may be entered in either case; they are stored in upper case. For example, count, Count, and COUNT all refer to the same variable.

Examples of valid names:

```
| A | K | X1 | | COUNT | score | A2B3C4 | | TOTAL | index | xyz789 |
```

Examples of invalid names:

```
| 1A | (begins with a digit) | | A-B | (- is not a letter or digit) | | MY VAR | (spaces are not allowed inside a name) |
```

The first four characters of a name determine its identity for short names. Names longer than four characters are distinguished by their full content, so COUNT and COUNTER are different variables.

If you have not used a variable yet, BASIC creates it with the value 0 the first time you read it.

1.5 String values

A string is a sequence of bytes. String *literals* are written between double-quote characters:

```
A$ = "HELLO"  
PRINT A$  
HELLO
```

A string literal in source text may not contain a double quote. To include a quote, use CHR\$(34) and string concatenation:

```
PRINT "HE SAID "; CHR$(34); "HI"; CHR$(34)  
HE SAID "HI"
```

The bytes in a string are not interpreted as anything in particular. They round-trip through ASC and CHR\$ unchanged. See Appendix C for the character code table.

1.6 String variables

A string variable has the same naming rules as a numeric variable but ends with a \$:

```
A$ = "HELLO"  
NAME$ = "WORLD"  
LONGNAME$ = "GOOD MORNING"
```

The \$ is part of the name; A and A\$ are two different variables. Until you assign to a string variable, its value is the empty string "".

1.7 Arrays

You declare a numeric array with DIM. Subscripts begin at 0. The bound you give to DIM is the highest legal subscript, so DIM A(10) creates an array with eleven elements A(0) through A(10):

```

10 DIM A(10)
20 FOR I=0 TO 10:A(I)=I*I:NEXT
30 PRINT A(5)
RUN
25

```

Only numeric arrays may be dimensioned. There is no string-array form in this BASIC. If you need a sequence of strings, use several simple string variables (N0\$, N1\$, N2\$, ...) or pack the strings into a single string with a separator and split them with MID\$.

Arrays of more than one dimension are supported:

```

10 DIM GRID(7,7)
20 GRID(0,0)=1:GRID(7,7)=99
30 PRINT GRID(0,0)+GRID(7,7)
RUN
100

```

You may only DIM an array once. To resize, use CLEAR to discard all variables and arrays, then DIM again.

1.8 Expressions and operator precedence

A BASIC expression combines values, variables, and operators. When several operators appear, BASIC applies them in order of **precedence**. Higher-precedence operators are applied first. Operators of equal precedence are applied left to right.

The precedence table below lists the operators from highest to lowest. The lowest-precedence operators (OR, EOR) are the last to be applied.

Level	Operators	Meaning
1 (highest)	(. . .)	grouping
2	^	raise to a power
3	unary -, +	sign
4	*, /	multiply, divide
5	+, -, <<, >>	add and subtract; + also concatenates strings; integer shift left and shift right
6	<, <=, =, >, >=, >	compare
7	NOT	bitwise complement (integer)
8	AND	bitwise AND (integer)
9 (lowest)	OR, EOR	bitwise OR, bitwise exclusive-OR (integer)

Use parentheses to override the default order. The expression 2*3+4*5 evaluates to 26. The expression 2*(3+4)*5 evaluates to 70.

The comparison operators return -1 for TRUE and 0 for FALSE. This is useful in arithmetic with conditions:

```

10 A=5:B=3
20 PRINT (A>B)
RUN
-1

```

Multiplying by a comparison's result turns the comparison into a selector. With A=5 and B=3, A>B is -1, so 10 * (A>B) is -10:

```
A=5:B=3
PRINT 10 * (A>B)
-10
```

The bitwise operators AND, OR, EOR, NOT, <<, and >> operate on the 32-bit integer representation of their operands. The operands are truncated to integer, the bitwise operation runs, and the result is converted back to a number.

1.9 Assignment

To give a variable a value, use the = sign:

```
A = 5
B$ = "DOG"
A(3) = 99
```

The keyword LET may precede an assignment but is not required:

```
LET A = 5
```

You may assign one variable's value to another:

```
A = 5
B = A
PRINT B
5
```

When you assign to a string variable, BASIC copies the bytes of the right-hand side into the internal arena and stores the new pointer in the variable's slot.

1.10 The PRINT statement

PRINT displays values on the terminal. Items separated by semicolons are printed adjacent. Items separated by commas are printed with a TAB character (CHR\$(9)) between them; the visible effect of TAB depends on the active text mode. A trailing semicolon suppresses the final newline; a trailing comma does not:

```
PRINT "A"; "B"; "C"
ABC

PRINT "A", "B", "C"
A  B  C

PRINT "WAIT...";
WAIT...
```

PRINT followed by no arguments prints a blank line. After every PRINT that does not end with a semicolon, BASIC sends CHR\$(13) followed by CHR\$(10) to the terminal.

The question mark ? is short for PRINT. Typing ? 2+3 and typing PRINT 2+3 produce identical results. When you LIST a program, the question mark is shown as PRINT.

1.11 Comments

REM introduces a comment. Everything from REM to the end of the line is ignored. The comment is stored as literal text in the program (it survives LIST and SAVE):

```
10 REM THIS IS A COMMENT
20 PRINT "HELLO" : REM TRAILING COMMENT
```

A comment placed after `:` only ignores the rest of the current line; the next program line still executes.

1.12 Editing a program

While you build a program, you can use the following commands at the direct-mode prompt:

Command	What it does
LIST	List the whole program. Arguments after LIST are ignored.
NEW	Delete the program and all variables.
CLEAR	Delete all variables and arrays; keep the program.
RUN	Run the program from the lowest line. Numeric line arguments are not parsed, so <code>RUN 100</code> behaves like <code>RUN</code> .
RUN AOT	Compile the stored program to native IE64 code, then run it.
CONT	Continue from where STOP halted.
SAVE "name"	Save the program to a file (Chapter 35).
LOAD "name"	Load a program from a file (Chapter 35).
COMPILE "name"	Write the stored program as a standalone IE64 image (Chapter 35).
TRANSPILE "name"	Write the stored program as IE64 assembly text (Chapter 35).
ASSEMBLE "name"	Assemble IE64 source text into a standalone image (Chapter 35).
DIR	Show available filenames (direct mode only).
TYPE "name"	Print a text file from the disk volume (direct mode only).

DIR, TYPE, RUN AOT, COMPILE, TRANSPILE, and ASSEMBLE are direct-mode forms. They cannot appear inside a program line. See Appendix A for the list of words that work only at the prompt.

1.13 Stopping a program

A program stops in two ways:

1. Running off the last line.
2. Executing an END or STOP statement.

After STOP, you can examine variables in direct mode and then type CONT to resume from the next statement. After END, you must RUN again to start over.

The same rule applies to a program started with RUN AOT: a top-level STOP saves a native IE64 continuation, and CONT re-enters the compiled code. Editing the program, NEW, LOAD, or a fresh RUN or RUN AOT discards that continuation. If a

compiled STOP is reached inside an active GOSUB, the following RETURN cannot be resumed as a subroutine return; use top-level STOP points when you want to continue a compiled run.

1.14 Error handling

When a statement fails, BASIC prints an error message and returns to the direct-mode prompt. In program mode the line number of the offending statement is also shown:

```
10 PRINT 1/0
RUN
?DIVISION BY ZERO ERROR IN 10
Ready
.
```

Appendix I lists every error message and its meaning.

1.15 What to do next

Chapter 2 lists every BASIC keyword in alphabetical order, with its syntax, an example, and the chapter where the related hardware appears. You do not need to memorise it. Skim it once, then keep going. Once you can write a short program in direct mode and understand what each operator does, Chapters 3 through 10 show how to use the display cards, and Chapters 11 through 21 show how to use the sound engines on the same machine.