CAMBRIDGE

Cambridge University Press
978-0-521-21704-0 - Illustrating Basic: (A Simple Programming Language)
Donald Alcock
Excerpt
More information

1

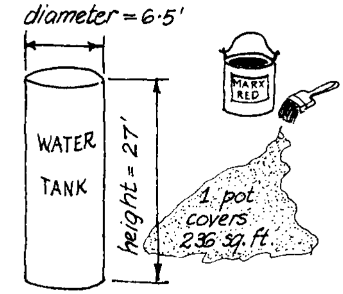
COMPONENTS OF THE LANGUAGE

CAMBRIDGE

Cambridge University Press
978-0-521-21704-0 - Illustrating Basic: (A Simple Programming Language)
Donald Alcock
Excerpt
More information

PROBLEM

HOW MANY POTS OF PAINT DO YOU NEED TO PAINT THE ROOF AND WALL OF THIS WATER TANK?



WE COULD GO STRAIGHT AT IT LIKE THIS:

roof area, $T = \pi \times 6{\cdot}5^2 \div 4 = 33{\cdot}2$
wall area, $S = \pi \times 6{\cdot}5 \times 27 = 551$
total area, $A = T + S = 584{\cdot}2$
number of pots, $G = A \div 236 = 2{\cdot}48$
rounding up, $R = 3$
∴ *you need 3 pots of paint*

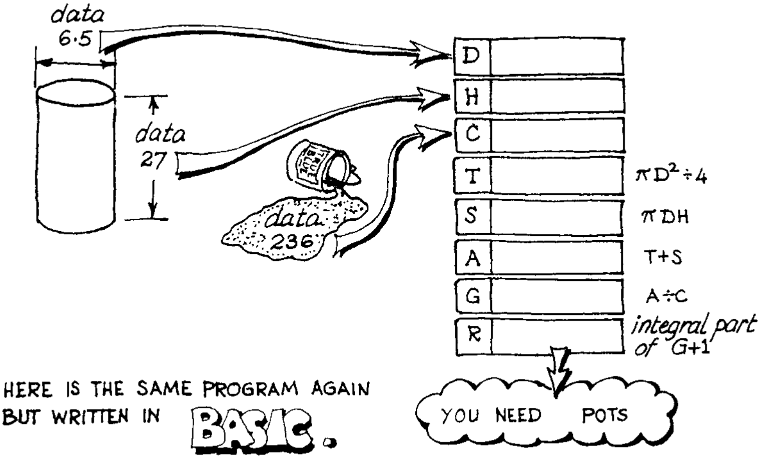
OR WE COULD WRITE A *PROGRAM* (IN ENGLISH) TO SOLVE THE PROBLEM.

```
1. REMARK:  A PROGRAM IN ENGLISH
2. THE FOLLOWING NUMBERS ARE THE DATA  6·5, 27, 236
3. READ THE DATA, CALLING THEM D, H & C RESPECTIVELY
   (think of this as putting the data into little boxes
   labelled D, H & C respectively – see opposite page)
4. WORK OUT 3·14×D²÷4 AND LET THE RESULT BE CALLED T
   (i.e. put the result in a little box labelled T)
5. WORK OUT 3·14×D×H AND LET THE RESULT BE CALLED S
6. ADD T TO S AND LET THE RESULT BE CALLED A
7. WORK OUT A÷C AND LET THE RESULT BE CALLED G
8. ROUND G TO THE NEXT WHOLE NUMBER AND
   LET THE RESULT BE CALLED R
   (i.e. add 1 to G and take the integral part of the result)
9. PRINT "YOU NEED" ; R ; "POTS"
   (i.e. print whatever whole number R turns out to be)
10. THE END
```

THIS HAS THE ADVANTAGE OF BEING GOOD FOR ANY SIZE OF TANK AND PAINT POT ⇝ YOU NEED ONLY REPLACE THE DATA ON LINE 2.

NOW

TRY *OBEYING* THE ENGLISH PROGRAM OPPOSITE & FEEL WHAT IT WOULD BE LIKE TO *BE* A COMPUTER & *DEFILE* THIS PAGE BY WRITING NUMBERS IN THE LITTLE BOXES BELOW.



HERE IS THE SAME PROGRAM AGAIN BUT WRITTEN IN BASIC.

COMPARE IT CAREFULLY WITH THE *ENGLISH* VERSION OPPOSITE.

```
1   REM   A PROGRAM IN BASIC
2   DATA  6.5, 27, 236
3   READ  D, H, C
4   LET   T = 3.14*D↑2/4
5   LET   S = 3.14*D*H
6   LET   A = T + S
7   LET   G = A/C
8   LET   R = INT (G + 1)
9   PRINT "YOU NEED"; R ; "POTS"
10  END
```

notice
* meaning multiply
↑ meaning raise to a power
/ meaning divide

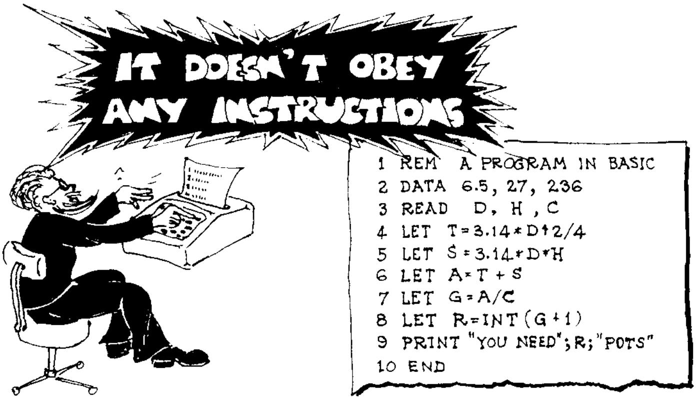
AND THIS, WHEN OBEYED, WOULD PRODUCE :

```
YOU NEED 3 POTS
```

FIRST

PREPARE YOUR PROGRAM BY TYPING INSTRUCTIONS AT THE KEYBOARD ☞ THE COMPUTER SIMPLY *STORES* THE PROGRAM AT THIS STAGE :

IT DOESN'T OBEY ANY INSTRUCTIONS



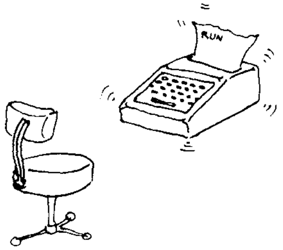
```
1  REM  A PROGRAM IN BASIC
2  DATA  6.5, 27, 236
3  READ  D, H , C
4  LET  T=3.14*D↑2/4
5  LET  S=3.14*D*H
6  LET  A=T+S
7  LET  G=A/C
8  LET  R=INT(G+1)
9  PRINT "YOU NEED";R;"POTS"
10 END
```

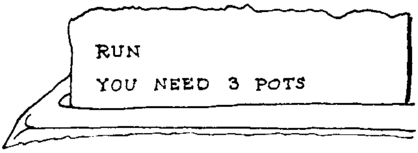
THEN

TYPE RUN

WHICH SETS THE COMPUTER TO WORK *OBEYING* THE STORED INSTRUCTIONS ONE AFTER THE OTHER ☞ IN NUMBERED SEQUENCE ☞ WHILST YOU RELAX .

EVENTUALLY THE COMPUTER WILL OBEY THE INSTRUCTION END . THAT MAKES IT STOP .





```
RUN
YOU NEED 3 POTS
```

BUT

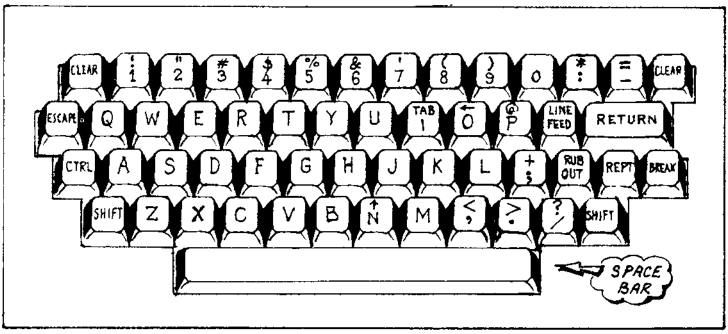
BEFORE YOU CAN TAKE THE FIRST STEP AND START TYPING THE PROGRAM YOU HAVE TO GO THROUGH THE RITUAL OF *SIGNING ON* AND TELLING THE COMPUTER YOU WANT TO USE *BASIC*.

DIFFERENT COMPUTERS (EVEN IDENTICAL COMPUTERS RUN BY DIFFERENT ORGANISATIONS) OFTEN HAVE DIFFERENT WAYS OF DOING THESE THINGS , SO IF YOU WANT TO TRY THE PROGRAM NOW GET SOMEONE WHO "KNOWS THE SYSTEM" TO SIGN ON FOR YOU AND CALL UP *BASIC*.

KEYBOARD

EVERY PROGRAM IN *BASIC* HAS TO BE TYPED ON A KEYBOARD

PROBABLY SOMETHING LIKE THIS


CLEAR ! 1 " 2 # 3 $ 4 % 5 & 6 ' 7 (8) 9 0 * : = - CLEAR
ESCAPE Q W E R T Y U TAB I ← O @ P LINE FEED RETURN
CTRL A S D F G H J K L + ; RUB OUT REPT BREAK
SHIFT Z X C V B ↑ N M < , > . ? / SHIFT
SPACE BAR

ALTHOUGH POSITIONS OF LETTERS & DIGITS ARE THE SAME ON MOST KEYBOARDS, KEYS LIKE RUB OUT & BREAK IN THE PICTURE ABOVE VARY IN NAME, POSITION AND FUNCTION FROM ONE INSTALLATION TO ANOTHER.

NOTICE ALL LETTERS ARE CAPITAL LETTERS. NOTICE ALSO THERE IS A KEY FOR 1 AND A KEY FOR ZERO (BOTH IN THE TOP ROW). NEVER PRESS THE LETTERS I AND O IN THEIR PLACE.

AS ON AN ORDINARY TYPEWRITER, PRESSING SHIFT AT THE SAME TIME AS ANOTHER KEY GIVES THE CHARACTER SHOWN ON THE UPPER HALF OF THAT KEY: THUS # 3 TOGETHER WITH SHIFT GIVES # WHEREAS # 3 ALONE, 3.

THE "BACK ARROW" ← SERVES TO DELETE THE CHARACTER ON ITS LEFT FROM THE COMPUTER'S MEMORY; TWO OF THEM DELETE THE PREVIOUS TWO CHARACTERS, AND SO ON. THUS IF YOU TYPE PRIMP←←NT THEN *BASIC* RECEIVES THE WORD PRINT. (REMEMBER THIS BY EXCLAIMING *OH SHIFT!* WHEN YOU HIT THE WRONG KEY.) SOME *BASICS*, HOWEVER, USE AN UNDERSCORE CHARACTER FOR THIS PURPOSE: PRIMP__NT.

MOST *BASICS* USE A KEY (PERHAPS "RUBOUT") WHICH, WHEN PRESSED, DELETES THE WHOLE OF THE LINE YOU ARE TYPING FROM THE COMPUTER'S MEMORY; ANOTHER (PERHAPS "BREAK") STOPS A PROGRAM RUNNING.

FOR A *NEW LINE* IN *BASIC* PRESS RETURN

WHEN TYPING "OFF LINE" PRESS RETURN then LINEFEED

TYPING

IF YOU INTEND TO USE *BASIC* A LOT, LEARN TOUCH TYPING. TEN FINGERS ARE FASTER AND LESS FRUSTRATING THAN TWO.

THERE IS A LIMIT TO THE LENGTH OF A TYPED LINE ☛ MOST *BASICS* ALLOW LINES UP TO 72 CHARACTERS LONG. SOME ALLOW LONGER LINES BUT IT IS BEST TO ACCEPT A LIMIT OF 72.

SOME *BASICS* ALLOW GREAT FREEDOM WITH THE SPACE BAR; SOME DISREGARD SPACES EXCEPT THOSE BETWEEN QUOTATION MARKS. THUS IT WOULD BE ALLOWABLE TO TYPE:

```
8FORD=STOP
```

INSTEAD OF:

```
8  FOR  D = S  TO  P
```

BUT IT IS OBVIOUSLY SILLY TO OBSCURE THE MEANING OF THE PROGRAM IN ORDER TO SAVE A FEW TAPS ON THE SPACE BAR.

SOME *BASICS* REFUSE TO ALLOW SPACES WITHIN THE CONTROLLING WORDS OF THE LANGUAGE. THUS THE FOLLOWING WOULD BE WRONG:

```
23 L E T  A = B+C
```

SOME *BASICS* DEMAND AT LEAST ONE SPACE BEFORE EACH CONTROLLING WORD, OR AFTER IT, OR BOTH:

```
20DATA   6.5, 27, 236
80 PRINT"YOU NEED";R;"POTS"
```

SOME *BASICS* REFUSE TO ACCEPT SPACES WITHIN LINE NUMBERS BUT DO NOT OBJECT TO THEM INSIDE OTHER NUMBERS:

```
1 000  LET  A = 1 000 . 0
1000  LET  A = 1000.0
```

SOME BASICS OBJECT TO THESE TOO

SOME *BASICS* DO NOT ALLOW SPACES IN FRONT OF LINE NUMBERS:

```
 95 LET  A = B
100 LET  C = D*F + G
```

SPACES OPTIONAL HERE

GENERALLY WHEN *ONE* SPACE IS ALLOWED (OR DEMANDED) THEN *SEVERAL* ARE ALLOWED. AND GENERALLY A SPACE IS OPTIONAL ON EITHER SIDE OF THESE ⇒ (, ; * + / − = ↑ > <) BUT NOT IN 1.5E2 (SEE PAGE 9) NOR BETWEEN > AND = (SEE PAGE 41).

A PROGRAM WHICH ACCEPTS ALL THESE RESTRICTIONS SHOULD BE ACCEPTABLE TO ANY VERSION OF *BASIC*.

LINE NUMBERS

LEAVE GAPS IN THEIR SEQUENCE THUS:

```
10  REM    A PROGRAM IN BASIC
20  DATA   6.5,  27,  286←←36
30  READ   D, H, C
40  LET    T = 3.14 * D↑2/4
50  LET    S = 3.14 * D * H
60  LET    A = T + S
70  LET    G = A/C
80  PRINT "YOU NEED"; R; "POTS"
90  END
```

USE 10s OR 5s

THIS IS 236. SEE PAGE 5

THERE IS A *MISTAKE* IN THIS PROGRAM: THE LAST LET WAS FORGOTTEN. INSERTING IT IS SIMPLE; JUST TYPE:

```
75  LET  R = INT (G+1)
```

AND THE COMPUTER PUTS LINE 75 BETWEEN LINE 70 & LINE 80. IT MAKES NO DIFFERENCE IN WHAT ORDER YOU *TYPE* THE LINES; THE COMPUTER SORTS THEM INTO ASCENDING ORDER OF LINE NUMBER.

IF YOU TYPE SEVERAL LINES WITH THE *SAME* LINE NUMBER THE COMPUTER OBLITERATES EACH PREVIOUS VERSION THUS ACCEPTING THE LINE TYPED LAST. IF THE LINE TYPED LAST IS *JUST* A LINE NUMBER WITH NOTHING AFTER IT THEN THE *WHOLE LINE VANISHES* FROM THE COMPUTER'S MEMORY — INCLUDING THE LINE NUMBER. THAT IS HOW TO DELETE UNWANTED LINES. THUS:

```
120 LET  A = B + C
125 LET  E = F
120 LET  A = B + G
125
120 LET  A = B
```

RESULTS IN THE COMPUTER REMEMBERING ONLY:

```
120 LET  A = B
```

THE FIRST LINE NUMBER IN A PROGRAM MUST BE GREATER THAN 0. THERE IS ALWAYS A LIMIT TO THE HIGHEST LINE NUMBER: SOME *BASICs* STOP AT 9999, SO IT IS BEST TO ACCEPT THIS AS THE LIMIT.

THE *LAST* STATEMENT OF EVERY PROGRAM MUST BE: **END**
(NO OTHER STATEMENT BUT THE LAST MAY SAY END).

STATEMENTS

A *BASIC* PROGRAM IS A SEQUENCE OF NUMBERED LINES CALLED *STATEMENTS*.

A STATEMENT MAY SIMPLY *STATE* SOMETHING

```
110 DATA 1, 2, 4
120 END
```

```
30 READ A,B,C
40 LET G=A*B↑2 +C
50 PRINT "ANSWER IS"; G
```

OR IT MAY *INSTRUCT* THE COMPUTER TO *DO* SOMETHING. A COMMON SYNONYM FOR *STATEMENT* IS *INSTRUCTION*; THE STATEMENTS THAT *DO* THINGS ARE *EXECUTABLE* INSTRUCTIONS.

THE COMPUTER FINDS OUT WHAT IS STATED OR WHAT TO DO BY LOOKING AT THE FIRST WORD: DATA, END, READ, LET *etc.*

OR SOMETIMES AT THE FIRST *TWO* WORDS: MAT READ, MAT PRINT *etc.* (WE MEET MAT ON PAGE 76).

BUT THERE IS AN IMPORTANT EXCEPTION:

THE WORD **LET** MAY BE *OMITTED* IN MOST VERSIONS OF *BASIC*.

```
40 G = A*B↑2 + C
```

REM

REM STANDS FOR *REMARK*. REM STATEMENTS CAUSE NO ACTION BY THE COMPUTER; YOU INCLUDE THEM TO CLARIFY YOUR PROGRAM.

```
10 REM          *** WATER TANKS ***
20 REM
30 REM   A PROGRAM TO ILLUSTRATE BASIC
40 REM   -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
50 DATA    6.5,     27,     236
60 REM    DIAM, HEIGHT, COVERAGE
```

REM FOR BLANK LINES

REM FOR EMBELLISHMENT

REM FOR CLARITY

THE EXAMPLES IN THIS BOOK DO NOT HAVE MANY "REM" STATEMENTS BECAUSE I HAVE ANNOTATED PROGRAMS WITH LITTLE ARROWS AND CLOUDS SO AS TO SAVE SPACE.

NUMBERS

YOU CAN TYPE NUMBERS THREE WAYS ☞ AS *INTEGERS*, AS *REALS* OR IN *E-FORM*.

INTEGER FORM
(WHOLE NUMBERS)

```
160 DATA  0, 2, +4, 1000, -30
```

REAL FORM
(DECIMAL NUMBERS)

```
170 DATA +0.70, 4., .6, -1.3, 987.65
```

4. MEANS 4.0

E-FORM
(EXPONENT FORM)
WHERE *E* SAYS: "TIMES TEN TO THE..."

```
190 DATA 1E3, 13.6E-4,-13.6E6,-.0136E9
```

1.0×10^{3} $=1,000$

13.6×10^{-4} $=0.00136$

-13.6×10^{6} $=-13,600,000$

$-.0136\times10^{9}$ $=-13600,000$

E INTRODUCES AN *INTEGER* SAYING HOW MANY PLACES TO SHIFT THE DECIMAL POINT. SHIFT TO THE RIGHT IF THE INTEGER IS POSITIVE; OTHERWISE LEFT.

```
190 DATA   E3, 13.6E1.2 , 13.6 E 2
```

IN THE *E* FORM THERE MUST ALWAYS BE A NUMBER IN FRONT OF THE *E* AND AN *INTEGER* AFTER IT. SOME *BASIC*S ALLOW SPACES WITHIN AN *E* FORM BUT IT IS BEST NOT TO HAVE THEM.

$\pm10^{38}$

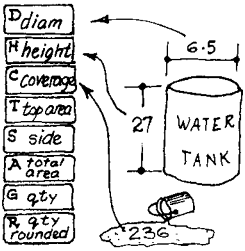
IN SOME *BASIC*S THE BIGGEST NUMBER THAT CAN BE STORED IS APPROXIMATELY $\pm10^{38}$ (*BIG* MEANS FAR FROM ZERO ON EITHER SIDE; *SMALL* MEANS CLOSE TO ZERO ON EITHER SIDE). OTHER *BASIC*S CAN HANDLE MUCH BIGGER NUMBERS THAN $\pm10^{38}$; IT DEPENDS ON THE COMPUTER'S "WORD LENGTH" AND WHETHER THE "WORDS" ARE USED SINGLY, IN PAIRS, OR IN MULTIPLES. BUT ***NO*** *BASIC* SHOULD REFUSE TO HANDLE A NUMBER AS BIG AS ±100,000,000,000,000,000,000,000,000,000,000,000,000.

6~7 SIG. FIG.

IN SOME *BASIC*S THE PRECISION OF STORAGE AND ARITHMETIC IS BETWEEN 6 AND 7 SIGNIFICANT DECIMAL DIGITS ☞ 987,654,321 WOULD BE STORED AS APPROXIMATELY 987,654,000. OTHER *BASIC*S OFFER MUCH HIGHER PRECISION, 15 SIGNIFICANT FIGURES BEING TYPICAL. AGAIN IT DEPENDS ON THE COMPUTER'S "WORD LENGTH" AND HOW THE "WORDS" ARE USED. BUT ***NO*** *BASIC* SHOULD WORK TO LESS PRECISION THAN 6 TO 7 SIG. FIGS. (THE VAGUENESS OF "6 TO 7" IS BECAUSE MOST COMPUTERS USE BINARY ARITHMETIC, NOT DECIMAL. A MORE PRECISE RENDERING WOULD BE "24 BINARY DIGITS FOR POSITIVE NUMBERS; 23 FOR NEGATIVE; OR VICE VERSA" BUT THESE IMPLICATIONS NEED NOT BOTHER THE NOVICE TO *BASIC*.)

VARIABLES

THERE ARE 286 *SIMPLE* NUMERICAL VARIABLES IN *BASIC*.



WE SAW SOME OF THESE ON PAGE 3. THEY ARE THE LITTLE BOXES USED TO HOLD NUMBERS.

THE FULL 286 ARE SHOWN BELOW. IT IS USEFUL TO KEEP A LARGE-SCALE CHART LIKE THIS AND MAKE A PHOTOCOPY FOR EACH NEW PROGRAM. AS YOU USE EACH VARIABLE WRITE A NOTE IN ITS BOX SAYING WHAT YOU USE IT FOR. THIS STOPS YOU USING VARIABLES ALREADY USED FOR SOMETHING ELSE A COMMON SOURCE OF TROUBLE.

A	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9
B	Bo	B1	B2	B3	B4	B5	B6	B7	B8	B9
C	Co	C1	C2	C3	C4	C5	C6	C7	C8	C9
D	Do	D1	D2	D3	D4	D5	D6	D7	D8	D9
E	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9
F	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9
G	G0	G1	G2	G3	G4	G5	G6	G7	G8	G9
H	H0	H1	H2	H3	H4	H5	H6	H7	H8	H9
I	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9
J	Jo	J1	J2	J3	J4	J5	J6	J7	J8	J9
K	K0	K1	K2	K3	K4	K5	K6	K7	K8	K9
L	Lo	L1	L2	L3	L4	L5	L6	L7	L8	L9
M	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9
N	N0	N1	N2	N3	N4	N5	N6	N7	N8	N9
O	O0	O1	O2	O3	O4	O5	O6	O7	O8	O9
P	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9
Q	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
R	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
S	So	S1	S2	S3	S4	S5	S6	S7	S8	S9
T	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9
U	U0	U1	U2	U3	U4	U5	U6	U7	U8	U9
V	Vo	V1	V2	V3	V4	V5	V6	V7	V8	V9
W	W0	W1	W2	W3	W4	W5	W6	W7	W8	W9
X	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9
Y	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9
Z	Zo	Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8	Z9