

Cambridge University Press  
978-0-521-31495-4 - Illustrating BBC-Basic  
Donald Alcock  
Excerpt  
[More information](#)

---

1

## INTRODUCTORY EXAMPLE

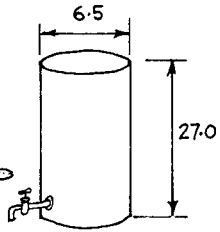
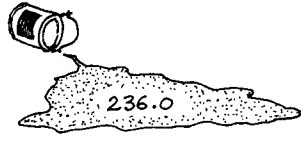
A PROBLEM  
SOLUTION IN ENGLISH  
SOLUTION IN BBC-BASIC  
TYPE THE PROGRAM  
EDIT  
LIST, RUN, RENUMBER  
AUTO  
SAVE AND LOAD

# A PROBLEM

ONLY THOSE MEETING A COMPUTER FOR THE FIRST TIME NEED READ THIS

All my introductory books start with this problem: how many pots of paint are needed to paint this water tank? The paint manufacturer claims that every pot has enough paint to cover an area of 236.

Recall that the area of a circle is given by  $\pi r^2$  (where  $r$  is its radius) or  $\pi d^2/4$  (where  $d$  is its diameter). The value of  $\pi$  is about 3.14:



$$\text{AREA OF LID} = 3.14 \times 6.5^2 \div 4 = 33.17$$

Recall that the circumference of a circle is given by  $\pi d$  (where  $d$  is its diameter as before). Therefore:

$$\begin{aligned} \text{AREA OF WALL} &= \text{CIRCUMFERENCE} \times \text{HEIGHT} \\ &= 3.14 \times 6.5 \times 27.0 = 551.07 \end{aligned}$$

The area to be painted is the sum of the two areas computed above:

$$\text{TOTAL AREA} = 33.17 + 551.07 = 584.24$$

Into this must be divided the coverage of a pot of paint so as to give the number of pots needed:

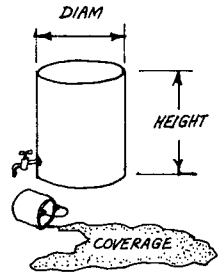
$$\text{POTS} = 584.24 \div 236.0 = 2.48$$

But you can't buy a fraction of a pot of paint, so the number calculated above must be rounded up to the next whole number. Look at it another way; take the *integral part* of 2.47558 (which is 2) and add 1 (which makes 3).

$$\text{NUMBER OF POTS} = \cancel{2.48} + 1 = 3$$

Of course, if the number of pots had worked out at 3.0000 then this approach would yield  $3+1=4$  which would be wrong mathematically but nevertheless a more practical answer than 3.

**NOW** suppose the painter wanted to set down this method of calculation so as to be able to calculate the number of pots of paint (having perhaps a different coverage) for a tank of any specified diameter and any specified height.



A possible list of instructions is set out on the opposite page.

# SOLUTION IN ENGLISH

INSTRUCTIONS GOOD FOR  
 ANY SIZE OF TANK

**Instruction 1.** "Input" three numbers representing the diameter and height of a particular tank and the coverage of a pot of paint. To "input" a number, draw a box for it and give each box a name thus:

$d$    $h$   coverage

then "put" the number "in" the box.

≈ example ≈  $d$    $h$   coverage

**Instruction 2.** Work out the area of the lid of the tank by the formula  $\pi d^2 \div 4$  where  $d$  is the number to be found in the box named  $d$ . Draw a little box; name it *lid*; write the answer inside the box

*lid*  ←  $\pi \times 6.5^2 \div 4$  ≈ example ≈

**Instruction 3.** Work out the area of the vertical wall of the tank by the formula  $\pi \times d \times h$  where  $d$  is the number in the box named  $d$ ;  $h$  is the number in the box named  $h$ . Draw a new little box; name it *wall*; write the answer inside the box.

≈ example ≈ *wall*  ←  $\pi \times 6.5 \times 27.0$

**Instruction 4.** Add the areas found in the boxes named *lid* and *wall*; divide this sum by the number found in the box named *coverage*. Draw a new little box; name it *paint*; write the answer inside this box.

≈ example ≈  $(33.17 + 551.07) \div 236.0$  → *paint*

**Instruction 5.** Take the integral part of the number found in the box named *paint* and add 1. Draw a new little box and name it *pots%* (the % in this context does not mean *per cent*; when appended to the name of a box it is a signal to say that the box is for storing whole numbers ≈ *integers*). Write the answer inside this box.

≈ example ≈  $2 \text{ (integral part)} + 1 \text{ (fractional part)}$  → *pots%*

**Instruction 6.** Print the value found in the box named *pots%* between the phrases "You need " and " pots of paint".

≈ example ≈

You need 3 pots of paint

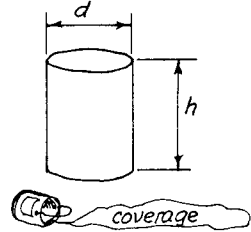
In computer jargon a set of instructions like this is called a *program*. This program is in English; on the next page it is translated into BBC-BASIC.

## SOLUTION IN BBC-BASIC GOOD FOR ANY SIZE OF TANK

Here is the water-tank program in BBC-BASIC:

```

1 INPUT d, h, coverage
2 LET lid = PI * d^2 / 4
3 LET wall = PI * d * h
4 LET paint = (lid + wall) / coverage
5 LET pots% = INT(paint) + 1
6 PRINT "You need "; pots%; " pots of paint"
```



The English instructions on the previous page have a one-to-one correspondence with the statements above (a *statement* is computer jargon for *instruction*). But before typing the program and making the computer obey it it is worth looking at some of the niceties.

```
1 INPUT d, h, coverage
```

Every statement in this example begins with a number  $\approx$  a *line number*  $\approx$  followed by a *keyword*. In statement 1 the keyword is "INPUT". Keywords make up the vocabulary of BBC-BASIC; all are defined in this book. The keyword after the line number tells the computer what sort of thing to do. "INPUT" tells the computer to draw some little boxes and give them the names listed (in this case the names  $d$ ,  $h$  and *coverage*). The computer jargon for "little box" is "variable", so  $d$ ,  $h$  and *coverage* are names of *variables*. The keyword "INPUT" also tells the computer to display a question mark on the screen, wait for the operator to type a number, then PUT that number into the next variable. The computer must do this for each nominated variable of the list in turn.

```
2 LET lid = PI * d^2 / 4
```

Statement 2 begins with the keyword LET. When LET is the keyword there is always an equals sign as well. On the left of the equals sign is the name of a variable (in this case *lid*); on the right is an *expression* (in this case  $PI * d^2 / 4$ ). The computer is told to evaluate the expression and assign the resulting value to the nominated variable. So this kind of statement is called an *assignment* statement.

The expression  $PI * d^2 / 4$  needs clarification:

- PI could be replaced by 3.14159265 without altering the effect of this program. PI (denoting  $\pi$ ) when written as part of an expression simply implies 3.14159265 (more accurate than 3.14 as used earlier)
- \* says *multiply by*
- ^ says *raise to the power of* (thus  $^2$  says "squared")
- / says *divide by*
- $d$  implies the number to be found in the variable named  $d$   $\approx$  this is a fundamental concept.

Cambridge University Press  
 978-0-521-31495-4 - Illustrating BBC-Basic  
 Donald Alcock  
 Excerpt  
[More information](#)

3 LET wall = PI \* d \* h

Statement 3 is another assignment statement. The computer is told to multiply 3.14159265 by the number found in the variable named *d*, then multiply the product by the number found in the variable named *h*, then assign the result to a variable which the computer is to create and give the name *wall*.

4 LET paint = (lid + wall) / coverage

Statement 4 is another assignment statement. The computer is told to add the numbers found in the variables named *lid* and *wall*, then divide this sum by the number found in the variable named *coverage*, then assign the result to a new variable to be named *paint*. Notice the effect of the brackets; they tell the computer to do the addition before the division. Without the brackets the division would be done first because it has higher *precedence*. The precedence of operators is:

- ^ is applied first. In statement 2 the  $d^2$  is done before the multiplication or division as one would expect (( $\pi d^2$  means  $\pi \times (d^2)$  not  $(\pi d)^2$ ))
- / \* are applied next, from left to right unless brackets signify otherwise. In statement 2 the \* is applied before the /
- + are applied last, from left to right unless brackets signify otherwise.

5 LET pots% = INT(paint) + 1

Statement 5 is another assignment statement. The expression involves INT() which is a *function*; it tells the computer to take the integral part of what is inside the brackets. (There are many other useful functions provided by BBC-BASIC; they are all explained in this book.) The value in the brackets is, in this example, simply the number to be found in the variable named *paint*. The computer is told to add 1 to the resulting integer and assign the result to a new variable to be named *pots%*. The % says this new variable is to be a variable for storing integers (you cannot store 2.4 in a variable whose name ends in%).

Although all the above assignments begin with the keyword LET, it is allowable in BBC-BASIC (as in most other BASICs) to omit the word LET.

6 PRINT "You need "; pots%; " pots of paint"

Statement 6 begins with the keyword PRINT which tells the computer to display something on the screen (the word PRINT having been adopted as a keyword in BASIC in the days of clattering typewriter terminals). The computer is told to copy anything in quotes exactly as quoted, but to replace the *names* of variables outside the quotes with the *contents* of those variables. So in this example the computer would make the screen display:

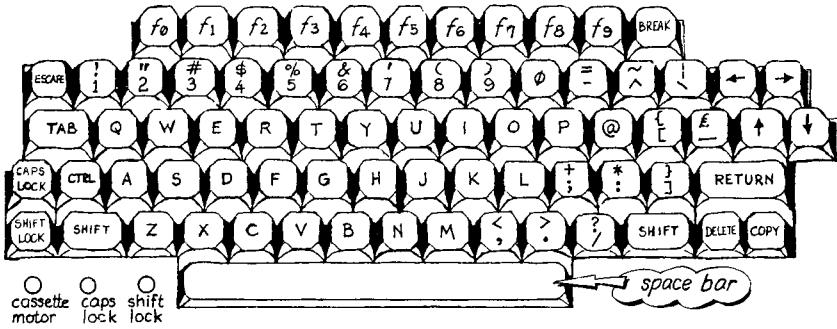
You need 3 pots of paint

Cambridge University Press  
 978-0-521-31495-4 - Illustrating BBC-Basic  
 Donald Alcock  
 Excerpt  
[More information](#)

## TYPE THE PROGRAM

FIRST PLUG IN & SWITCH ON  
 AS EXPLAINED IN USER GUIDE

The keyboard is arranged like this:



The screen should now show > which is a prompt saying "Type something."

The middle light (caps lock) should be on; if not press **CAPS LOCK** and the light should glow ☺ this is a "push on, push off" switch.

Now type this program. Some freedom is allowed with spacing, but for the time being put spaces where shown. (To type a space press the space bar as on an ordinary typewriter.) After typing each line press **RETURN** to get the > prompt at the start of the next line.

```
>1 INPUT D, H, COVERAGE
>2 LET LID = PI * D ^ 2 / 4
>3 LET WALL = PI * D * H
>4 LET PAINT = (LID + WALL) / COVERAGE
>5 LET POTS% = INT(PAINT) + 1
>6 PRINT "YOU NEED "; POTS%; " POTS OF PAINT"
>:≡
```

This is the same as the program on previous pages except that everything is in capitals to make typing easier. However, names of variables *may* be typed in lower-case letters. To do this press **CAPS LOCK** which makes the middle light go out; then type the name (it appears in small letters); then press **CAPS LOCK** again. But be consistent; don't type "WALL" on line 3, for example, and "wall" on line 4 because "WALL" and "wall" and "Wall" are all distinct names. Also, don't type "LET" as "let" because *all keywords must always be in capitals*.

In the rest of this book capital letters are used except for remarks and texts; I adopted this policy because I found it impractical to press **CAPS LOCK** twice for every name, and would always forget whether the light was on or off with infuriating results.

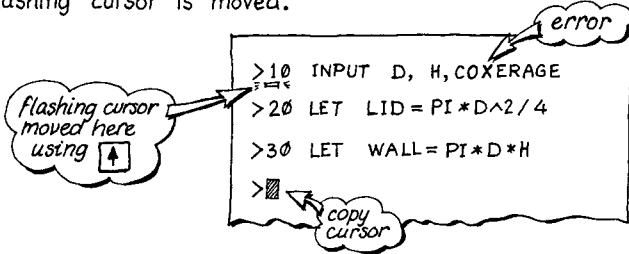
To type % hold down **SHIFT** and press **%/5**, or press **SHIFT LOCK** and then **%/5**. When **SHIFT LOCK** is pressed the third light comes on and the middle light, if on, goes out. On pressing **SHIFT LOCK** again, the third light, if on, goes out. To touch typists these lights are inconveniently placed, being always shielded by the left hand.

# EDIT

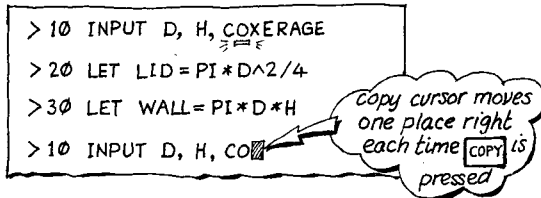
HOW TO COPY AND CORRECT ANY NUMBERED LINE USING ← → ↑ ↓ AND COPY

There are three ways of correcting a typing mistake. If the mistake is noticed straight away, press **DELETE** which erases the character to the left of the cursor. The whole line may be deleted by holding this key down and letting it repeat. If a mistake is seen on a previous line then simply type that line again, but correctly, making sure that it has the same line number as before. (A line may be erased completely by line number and all by typing just the line number and pressing **RETURN**.) The third method involves the arrow keys and is explained below.

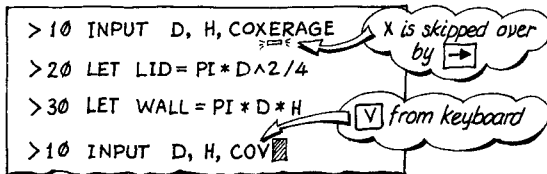
Pressing an arrow key **← → ↑ ↓** causes an extra cursor to appear. It is called the *copy cursor*. Whereas the usual cursor is a flashing underscore the copy cursor is a stationary rectangle. The flashing cursor may be moved about the screen by the arrow keys; the copy cursor stays put whilst the flashing cursor is moved.



Pressing **COPY** makes the character above the flashing cursor reappear under the copy cursor. The copy cursor then moves one position to the right.



By pressing **COPY** again and again the erroneous line may be copied until the flashing cursor reaches the error. Then the correct character ( or characters ) may be typed at the keyboard. Then the erroneous characters may be skipped by pressing **→**.



Finally the rest of the line may be copied by pressing **COPY** as before.

When **RETURN** is pressed, the copied ( and corrected ) line 10 replaces the erroneous line 10 in the computer's memory. With a little practice this "screen editor" becomes automatic to the fingers.

# LIST, RUN, RENUMBER COMMANDS OF BBC-BASIC (SEE ALSO CHAPTER 13)

Having typed the program opposite, type **CLS** RETURN. CLS is an abbreviation of "Clear the Screen". Its effect is to make the screen turn black. But a clear screen does not imply a vacant memory; type **LIST** followed by **RETURN** to prove the program is still there.

```
>CLS
Clear Screen
>LIST
1 INPUT D, H, COVERAGE
2 LET LID=PI*D^2/4
3 LET WALL=PI*D*H
4 LET PAINT=(LID+WALL)/COVERAGE
5 LET POTS%=INT(PAINT)+1
6 PRINT "YOU NEED "; POTS%; " POTS OF PAINT"
>
```

*LIST the program in memory*

Now the big moment has arrived; type **RUN** RETURN to make the computer begin executing the program in other words obeying the instructions in sequence.

The first statement is **1 INPUT D, H, COVERAGE** which causes a question mark to appear on the screen. The question mark means the computer is waiting for data. Type the diameter of the water tank: **6.5** RETURN. Another question mark appears. Type the height of the water tank: **2.7** RETURN. Yet another question mark appears. This time type the coverage of a pot of paint: **2.36** RETURN.

The **INPUT** statement on line 1 of the program has now been satisfied; the computer goes on to execute the remaining statements. The final statement makes the screen display the result. which is **YOU NEED 3 POTS OF PAINT**. The screen now looks like this:

```
>RUN
?6.5
?27
?236
YOU NEED 3 POTS OF PAINT
>=
```

*the result*

Now type **RENUMBER** RETURN then **LIST** RETURN. The result:

```
>RENUMBER
renumber as 10, 20, 30, ...
>LIST
10 INPUT D, H, COVERAGE
20 LET LID=PI*D^2/4
30 LET WALL=PI*D*H
40 LET PAINT=(LID+WALL)/COVERAGE
50 LET POTS%=INT(PAINT)+1
60 PRINT "YOU NEED "; POTS%; " POTS OF PAINT"
>=
```

*commands*

*statements*

*prompt for next line or next command*

Now save the program on cassette or disk as explained opposite.



Cambridge University Press  
 978-0-521-31495-4 - Illustrating BBC-Basic  
 Donald Alcock  
 Excerpt  
[More information](#)

## AUTO

MAKE THE COMPUTER PROVIDE LINE NUMBERS  
 AUTOMATICALLY

When typing a program it is not necessary to type line numbers, the computer may be made to provide them automatically. All that is necessary is to type the command AUTO and press **RETURN**.


```
>AUTO
10 ;==<
```



Unless instructed otherwise the computer offers 10, 20, 30,... as the line numbers.


To make the computer stop offering line numbers, press **ESCAPE**:

```
10 INPUT D,H,COVERAGE
20 LET LID=PI*D^2/4
30
Escape
>:==<
```



Automatic line numbering can be restarted at any line number:

```
>AUTO 30
30 ;==<
```



## SAVE AND LOAD

JUST THE ESSENTIALS: SEE CHAPTER 13  
 ALSO THE USER GUIDE, CHAPTER 5

If the computer is fitted with a disk, to save the program type the command SAVE followed by a name in quotes. Then type the command \*CAT (short for catalogue) to ensure the program has been properly saved:

```
>SAVE "TANK1"
>*CAT
```

*list of programs saved on disk; should now include a program named TANK1*

To save a program on cassette, rewind the tape and set the counter to 0000. Type SAVE "TANK1" and press **RETURN**. The message "RECORD then RETURN" should now appear. Fast forward the tape to the place where the program is to go, preferably with a multiple of 100 on the counter. Then press the RECORD button(s) on the cassette recorder, then the **RETURN** key on the computer. Press **ESCAPE** and start again should anything go wrong.

To bring a program back into the computer's memory, use LOAD:

```
>LOAD "TANK1"
```

If using cassette tape, locate the tape just before the start of TANK1 and so avoid a long delay whilst the tape is being searched for the program with this name.

Cambridge University Press  
978-0-521-31495-4 - Illustrating BBC-Basic  
Donald Alcock  
Excerpt  
[More information](#)

---

## EXERCISES

1. Use `AUTO` and type the example program. Try the program with data provided in this book; try with other data too.
2. Practise with the screen editor
3. Practise with `SAVE` and `LOAD`