# TEXAS INSTRUMENTS
## HOME COMPUTER

# STARTER PACK 1

## CASSETTE SOFTWARE
### WITH MANUAL

An integrated pack containing a series of programs on cassette that develop and graphically display major ideas covered in the accompanying book. Enables any user to progressively understand and make full use of this computer.



COLLINS
MICROSOFTWARE
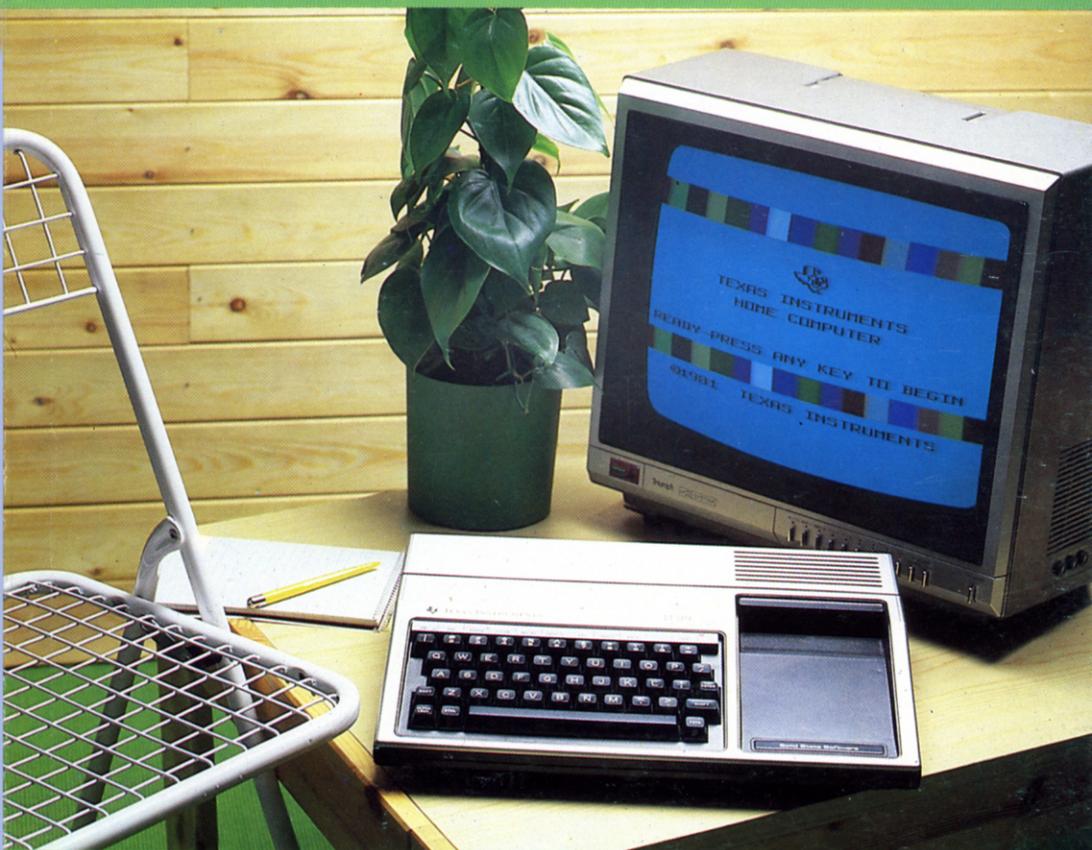
# TEXAS INSTRUMENTS
## HOME COMPUTER

# Starter Pack 1
## PK McBride

**COLLINS
MICROSOFTWARE**

# Contents

# Introduction

This Pack is the first of a series of five which together form a complete course in programming in **TI BASIC** using the TI-99/4A home computer.

BASIC (Beginners All-purpose Symbolic Instruction Code) is one of the easiest computer languages to learn, and the language of almost all home computers. **TI BASIC** is a version specially developed for Texas Instruments. It is slightly different from other forms of the language in that it has a number of special routines built into it to make programming easier. However, you will find that once you have learned **TI BASIC**, you can easily transfer to the versions that other machines use. You will also find that the programming skills you have mastered will make it easier to learn other computer languages if you ever want to.

This Pack teaches the techniques and routines needed for writing a wide variety of programs using simple BASIC. Note that simple does not have to mean short. The program TRANSPORT on the cassette uses little more than those BASIC commands that are covered in this book.

Starter Pack 2 will take your understanding of BASIC a lot further, and will help you to explore deeper into the Colour and Sound capabilities of the 99. By the time you have finished that part of the course, you will have all the skills needed to write programs as complex as any that you see on the cassettes in these Packs.

The two Gamewriter's Packs deal with the particular techniques needed for writing many types of computer games, and include a number of example games.

The last part of the course, the Record Keeper's Pack looks at the way in which computers handle information, and will enable you to use the 99 for keeping your accounts, records

of collections, lists of addresses or other information. It also deals with the analysis and presentation of statistics on the 99.

## Using the book

Work steadily through the book, trying out all the short demonstration programs that are included and writing your own programs using techniques as they are covered. Take your time, and play with each new idea until you are sure that you have mastered it.

Don't worry if things go wrong. It happens all the time when you are learning. You will find a list of common errors in Appendix D which should help you to sort out any problems.

Don't be afraid to experiment. **TI BASIC** will not let you make many mistakes, and nothing you type in will damage the computer. At the very worst, you can always switch off and start again.

Remember that you are learning a language, and that the best way to do this is by using it. Remember also that there are often several different ways of saying the same thing in any language. BASIC is the same. What matters at this stage is that the computer does what you want it to do.

## Using the cassette

The programs on the cassette are intended to be used with particular chapters in the book. KEYS (chapter 23), CHARLIES (chapter 20) and TRANSPORT (chapter 22) demonstrate the uses of particular techniques. EFFECTS (chapters 5, 13 & 22) and GRAPHICS (chapter 19) are demonstrations, and also utilities. You may wish to use some of the colour and sound special effects in your own programs, and GRAPHICS provides you with a set of 32 graphics characters which you can use to make your own pictures on screen.

For advice about the use of a cassette recorder, see Appendix C at the back of the book.

# 1
# Getting started

Set your computer up using the instructions in the "READ
THIS FIRST" handbook that came with the machine. Check
that the computer and T.V. are connected as shown in
figure 1, and that both are plugged into the mains and
switched on. The little red light at the front right of the 99
should be on.



**Figure 1**

ON/OFF switch
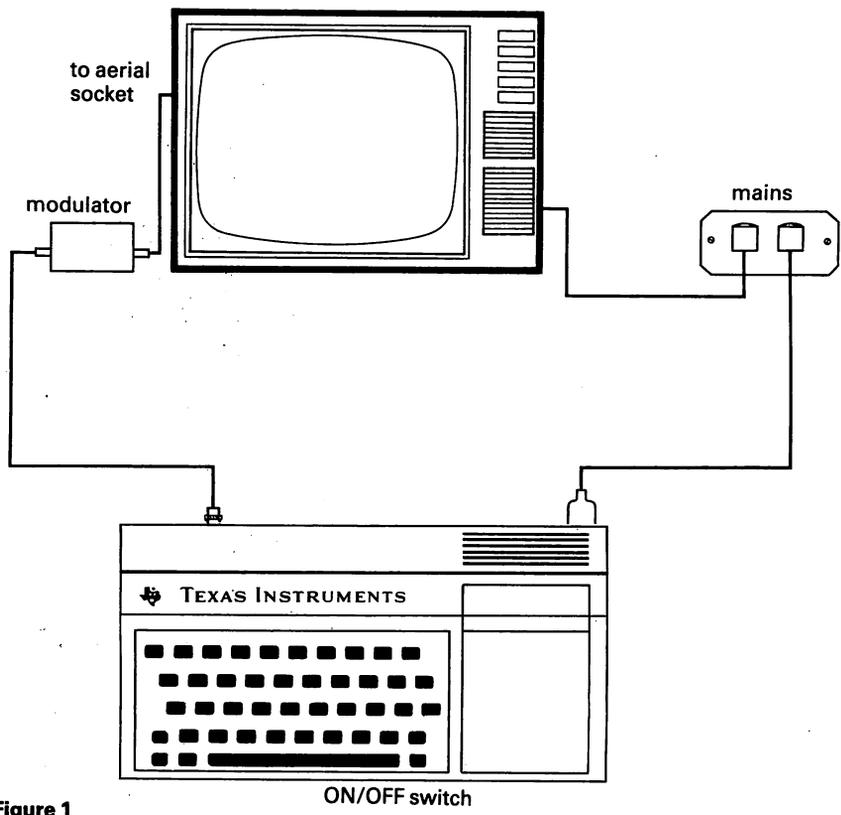
Choose a channel of your T.V. set that is not used for anything else (if possible) and tune it into the computer. You should get the master title screen (figure 2). The background colour is cyan (pale blue.)
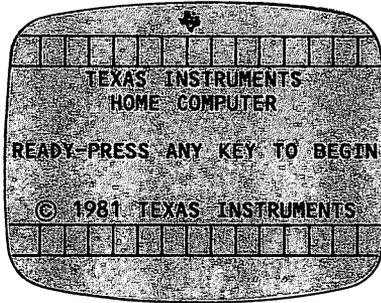
**Figure 2**

Press any key and you will move to the starting screen (figure 3). If you ever have a Solid State Software module plugged into the 99, this is the time that you will pick whether to use the module, or to work in **TI BASIC**.
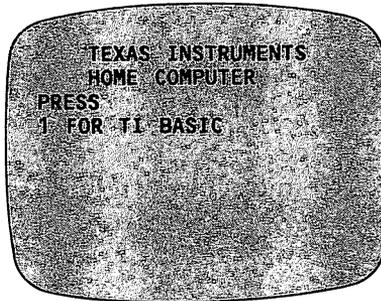


**Figure 3**

Press ⊡ now and you will hear a short high beep and the screen will change to figure 4. If you haven't heard any beeps, it's because you haven't turned up the T.V. volume. All the sounds from the computer are directed through the T.V. loudspeaker.
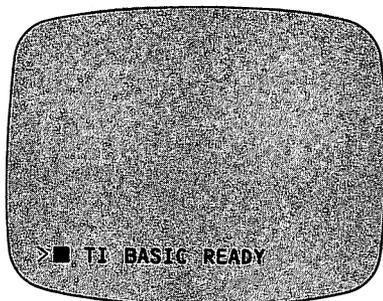
The 99 is now ready to start work. Are you?

## Trouble tuning in?

If you cannot find the picture from the computer check that all the leads from the TI-99 to the modulator and onto the T.V. are plugged in. Now try this. Turn up the volume on the T.V. until you get a reasonably loud hiss. Now keep tapping the 99's space bar while you tune in the set. Every time you tap this key the computer will beep. These beeps can be picked up over a wider range than the picture's signal. Once you hear the beeps you will know that you are nearly there.

Just one point though – all that tapping has moved you past the master title page. To get back to this hold down the FunCTioN key (FCTN) and press ⊟ (QUIT).

## The keyboard

The keyboard of the 99 is more or less the same as that of a normal typewriter, and like a typewriter each key can do more than one thing.

Each letter key can give you either a large or a small capital letter. The small capitals are what will normally appear. To get the large capitals, hold either SHIFT key (they are both

the same) and press the letter. To lock onto the large capitals, press down the ALPHA LOCK key. This affects the letters keys only. The number keys will still give you the number, unless you hold SHIFT when you press them.

Try typing a few things on the screen to get the feel of the SHIFT and ALPHA LOCK keys. It doesn't matter if it doesn't make sense to the computer. The 99 is programmed to ignore anything it doesn't understand. Press the ENTER key after typing in your name, or other message, and it will give you the nonsense beep, and print up * INCORRECT STATEMENT.

| | | | | | | | | | | | • |
| DEL | INS | ERASE | CLEAR | BEGIN | PROC'D | AID | REDO | BACK | | QUIT | • |

| ! 1 | @ 2 | # 3 | $ 4 | % 5 | ∧ 6 | & 7 | * 8 | ( 9 | ) 0 | + = |
| Q | W | E | R | T | Y | U | I | □ | P | − / |
| A | S | D | F | G | H | J | K | L | : ; | ENTER |
| SHIFT | Z | X | C | V | B | N | M | < , | > . | SHIFT |
| ALPHA LOCK | CTRL | | | | | | | | FCTN | |

*EXAMPLES*

Key 4
Normal  − 4
SHIFT   − $
FCTN    − CLEAR

Key P
Normal  − small
SHIFT   − large
FCTN    −  "

**Figure 5**

When you are typing in your programs later, it makes no difference whether you use large or small capitals. The 99 will automatically turn all BASIC words into large capitals. You may find it useful to leave the ALPHA LOCK on, and turn it off only when you particularly want small letters.

There are two other keys which change the way the number and letter keys behave. FCTN (FunCTioN, on the right of the space bar) gets you to the symbols printed on the fronts of the keys, and also to the words on the slide-in overlay above the number keys. Hold FCTN and press ⊞ and you get QUIT which takes you back to the master title page.

The CTRL (ConTRoL on the left of the space bar) key does nothing – at the moment. We will put it to use later on.

Last, but by no means least, there is the ENTER key. Press this when you have finished typing your instruction, or message, or whatever. The 99 will do nothing until you ENTER.

## Disappearing screens

If your picture suddenly disappears, don't worry. This is simply the 99 looking after your T.V. for you. If you leave the computer turned on, but do not use it, then after about 10 minutes it shuts off the screen display. This prevents damage to the T.V. screen. To get the picture back – press any key.

# 2
# Hands on

Some people only buy computers so that they can get their name on the television screen. Just in case you are one of those people, this is the first thing we will do
Type in:

**PRINT**

Check your spelling carefully. If you have made a mistake then press ENTER. The 99 will print up "* INCORRECT STATEMENT". Ignore it and start again. PRINT tells the computer to put something on the screen. Any words you want written there must be put in quotes. Press FCTN and ⦿ to get the quote marks and type in your name. Now press FCTN and ⦿ again to put another set of quotes at the end. You should have something like this:

**PRINT"ROGER"**

Now press ENTER and the screen should look something like figure 6.



**Figure 6**

11
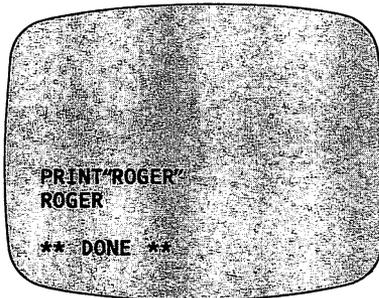
You will always get the DONE message when the 99 has finished a job. PRINT some more messages and names. Don't forget the quotes. If you get an * INCORRECT STATEMENT message, then check that the quotes are there, and that PRINT is spelt correctly. Start again, and do it right this time.

NOTE:   SPACES.
Obviously you can put spaces between words in your messages. You can also leave a space between PRINT and the first quotes. In fact, it is a good habit to get into. All BASIC words on the 99 must be followed by a space, or some sort of punctuation (",;:), with a few odd exceptions.

# 3
# Hello, hello

Time for a program, but first, is your screen cluttered up with old messages? If it is then type in :

**CALL CLEAR**

This CALLs up a special built-in routine that wipes the screen clean. The 99 has a number of these built-in routines, and we will meet more of them later.

Now type in:

**10  PRINT  "HELLO"**     (and press ENTER)
**20  PRINT  "GOODBYE"**     (and press ENTER)

Make sure that you leave a space between the number and the word PRINT. If you miss it out the 99 will be confused. All commands start after a space. The numbers are very important. They tell the 99 not to do anything yet, and they keep the instructions in the right order.

Check your typing and then add:

**RUN**     (and press ENTER)

There will be a short pause, while the computer checks the program, then the screen will turn light green while it prints. The screen returns to cyan when the program has finished. You should get a screen like figure 7.

RUN told the 99 to go to the first numbered line, do what it said, and move onto the next line, do that, move on until it reached the end.

If there is a mistake in your program, then you will get a message which says "* INCORRECT STATEMENT IN . . .". Look closely at the line numbered whatever-it-was and try and see what you have done. Have you mistyped "PRINT"?
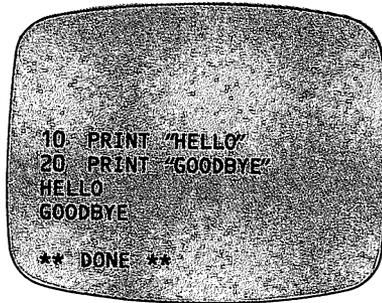
**Figure 7**

It is also possible that you used a letter O and not zero in the line number. If you have then the 99 will read the number 1 or 2 and try and make sense of the letter O that comes after it.

Whatever the mistake, the best thing to do now is to retype the line. When you ENTER the line, that new (good) line will replace the old (bad) line in the 99's memory. The rest of the program is left as it was. To see the new program type in:

**LIST**    (and press ENTER)

The program will be listed on the screen.

You can RUN a program as often as you like. When you get tired of "HELLO" and "GOODBYE" then write a new PRINTING program

First type in

**NEW**    (and press ENTER)

NEW wipes out the old program, ready for a new one. It also clears the screen and prints up the READY message.

Your PRINTing program can have as many lines as you like. Make sure that each line is typed correctly, and is numbered in the right order. You can number your lines 1, 2, 3, 4, etc, but going up in 10's is usual. There is a very good reason for this. Suppose you wanted to add "How are you?" to the "Hello" program. The new line has to go between

"Hello" and "goodbye". If you had numbered them 1 and 2, then you would have to retype half the program to make your change.

You will very often find that you want to add to a program, and numbering in 10's leaves you room to slip extra lines in easily.

Any time that you want to see your program lines again (and they may well have disappeared off the top of the screen after a few RUNS), then type:

**LIST**     (and press ENTER)

The lines are printed up, in the right order, at the bottom of the screen. Figure 8 shows the "Hello" program, after the extra line was added.
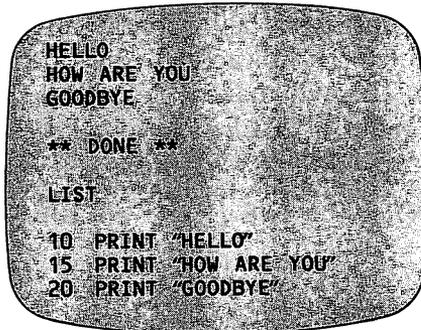


**Figure 8**

# 4
# Going round in circles

This program could run forever.

```
10 PRINT "HELLO AGAIN"
20 GO TO 10
```

Type it in and run it. You will see that the GO TO in line 20 sends the 99 back to line 10 and prints the message again. The screen fills, and the flickering that you see on the bottom line is where the message is being continually reprinted. You can't see it, but the top line is actually disappearing off the top of the screen. This will make it clear. Stop the program by pressing FCTN and 4 (CLEAR). You will get the message:

```
* BREAKPOINT AT LINE 10     (or 20)
```

Now add:

```
15 PRINT "GOODBYE AGAIN"
```

Run this and you will have a continually moving display. Break out of that program, (by pressing FCTN and 4) and NEW it.

Now type in a line to print your name, and put a comma after the last set of quotes. Add a GO TO line, and you should have something like this:

```
10 PRINT "SUSAN",
20 GO TO 10
```

Run the program and you should see a screen like figure 9.

Normally after a print message, the next message appears on the next line. If you add a comma though, the next message is printed half a line further on.
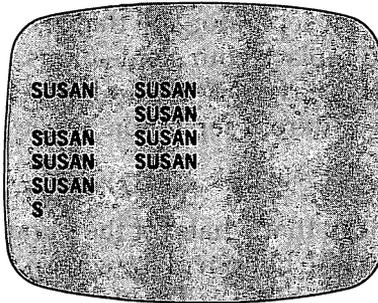
**Figure 9**

What happens when you put 2, or 3 commas after the print quotes?

The comma is what is known as a PRINT SEPARATOR. There are several others and we will find out what they do in a moment, but rather than retype that first line, we will EDIT it.

Type in:

**EDIT 10**   (and ENTER)

Line 10 is printed up at the bottom of the screen. Notice the flashing block, called the CURSOR. We can move this over the character we want to change by pressing FCTN and ⒟→. Hold those keys down for a moment and the 99's automatic repeat comes into play. The cursor whizzes along the line, until you lift your fingers off. If you overshoot, then press FCTN and ⒮← which moves the cursor left. Type a semi colon (;) over the first comma, and spaces over any other you have there, then ENTER the new line.

You should now have something like this:

**10 PRINT "SUSAN";**
**20 GO TO 10**

Run this. Where is the next message printed after a semi-colon?

Try it with 2 or 3 semi-colons. Is it any different?

There is one last print separator and that is the colon. (:)

   EDIT line 10 and replace your semi-colons with one colon.

What effect does this have: Try 2, 3 and more.

Try combining the different separators to see what effect
   they have.

You will have noticed that whatever the punctuation, all
printing starts from the bottom of the screen and moves up
as new lines are printed. We can hold the printing on the
bottom line by clearing the screen each time round. Add in
this line:

```
15 CALL CLEAR
```

Run it now. You should have a flickering name at the bottom
left. CALL CLEAR clears the screen, and puts the print
position back to the bottom.

NOTE:   GO TO can also be typed GOTO: the 99 doesn't
mind. This is one of the odd cases where a space can be
missed from a BASIC word.

## Even tighter circles

If you type in:

```
10 GO TO 10
```

and run it, what happens?

   You have got the 99 running round in a very tight, and
totally useless circle. Well, not totally useless. The line can be
used to hold the computer at one point in the program. To
break out of this hold FCTN and press 4 (CLEAR). There will
be times when you are trying to get the layout of the screen
right, and it will be hopeless if as soon as the screen's
printing is done the whole lot gets shuffled up to make space
for the * DONE * message.

```
100 GO TO 100
```

will give you time to see what you are doing. When you are
happy with the rest of the program and want to get rid of this
line, then type in:

**100**

The new line replaces the old line 100. As the new line is an
empty one, it is then ignored.

Here is a program that uses a GO TO line to confuse
people.

```
10 CALL CLEAR
20 PRINT "TI BASIC READY"
30 PRINT ">"
40 GO TO 40
```

Type the program in and run it.

The first three lines produce a screen almost like the
normal starting screen (see figure 10). Anyone coming to the
computer now would think that it was ready for use. It all
goes to prove that you should never believe anything you
see on television.

There is one difference between your screen and the
normal starting screen – it is coloured light green, rather than
cyan. Don't worry, we can fix that, and will do in the next
chapter.
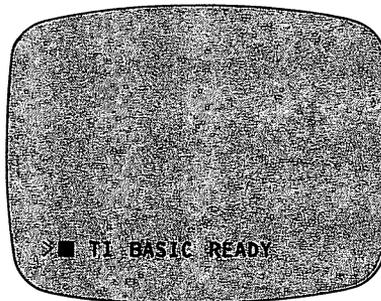


**Figure 10**

# 5
# Coloured paper

Next time you are just starting a programming session, or just after you have QUIT (FCTN and ±), look closely at the title screen. You will be able to count 15 different colours on the screen. (There are actually 16 colours, but one of these is Transparent and therefore you can't see it.)

There are two colours at each character space. One of these is the background colour – which you can think of as the paper, and the second is the foreground (ink) colour. Controlling the ink colours of different characters is a bit fiddly, so we will leave that until later, but it is a simple matter to change the colour of the screen.

The 99 knows its colours by their numbers, and here they are.

| Colour code | Colour |
|---|---|
| 1 | Transparent |
| 2 | Black |
| 3 | Medium Green |
| 4 | Light Green |
| 5 | Dark Blue |
| 6 | Light Blue |
| 7 | Dark Red |
| 8 | Cyan |
| 9 | Medium Red |
| 10 | Light Red |
| 11 | Dark Yellow |
| 12 | Light Yellow |
| 13 | Dark Green |
| 14 | Magenta |
| 15 | Grey |
| 16 | White |

To change the screen colour you use one of the 99's built in routines:

```
10 CALL SCREEN(10)
```

This will change the screen colour to Light Red (code 10), but you won't get much time to see it, as the screen will go back to cyan almost immediately. Hold the program by adding:

```
20 GO TO 20
```

Change the number in brackets in line 10 and see what you think about the screen colours. You can add variety to your programs by fixing a different screen colour at the start of each, or you may decide that there is one colour that you like best, and you will add a CALL SCREEN line at the start, to fix that colour.

The actual quality of the screen colours depends very largely on the type of T.V. set that you own. Some give much sharper colours than others.

## Ready?

You can now alter your trick READY program to make it more realistic. Add

```
15 CALL SCREEN(8)
```

and you will have the normal cyan starting screen.

## Easy editing

When you altered the code number in line 10, you probably did it by typing

```
EDIT 10
```

There is another way to get lines back for editing, and it takes less typing. Type in the line number only

```
10
```

and then hold down FCTN and press E ↑ . Line 10 now appears at the bottom, with the cursor in place ready for editing.

# 6
# Questions and answers

You are already using the computer's memory to store your program lines. Now we are going to use it to store data – numbers, names, answers to questions and other bits of information.

Computers treat words and numbers differently. We will start with words. This next program asks "WHO'S THERE?", takes in the name, and prints a friendly "HELLO" to whoever has answered.

**10 PRINT "WHO'S THERE?"**   FCTN and $\boxed{\text{I}}$
FCTN and $\boxed{\text{O}}$

**20 INPUT NAME$**   SHIFT and $\boxed{4}$

INPUT allows information to be typed into a program.
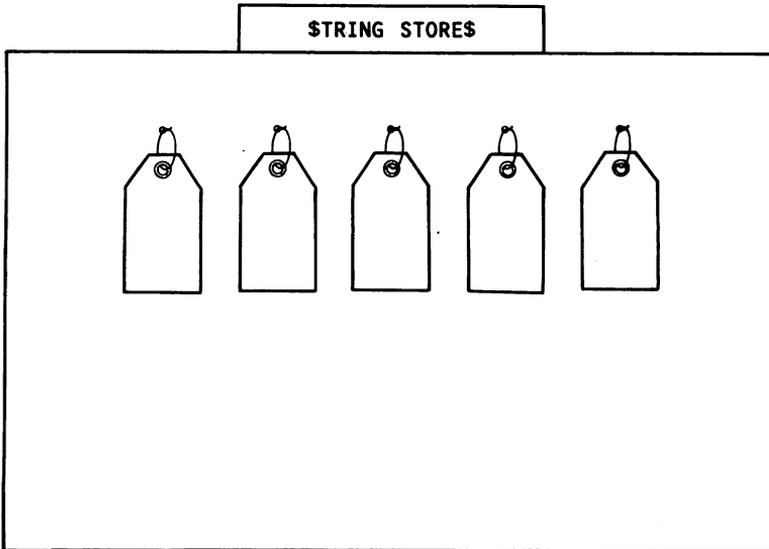


**Figure 11**

The $ sign stands for STRING. In the computer's memory are string stores where words are stored. Think of each string store as a luggage label (the sort you tie on with string). See figure 11. You can then mark each label with its own special name. Here we have called it NAME$. We could have called it N$, or ANSWER$ or A$. You can use any letter or group of letters for the string store labels (or STRING VARIABLES as they are properly called) – but follow these rules.

You must end with a $

you must start with a letter

you must not use spaces

Single letters or short words are best, as they save typing errors.

Let's finish the program:

### 30 PRINT "HELLO THERE ";NAME$

(Notice the semi-colon, and the space after THERE. What happens if you miss it out?)

You may not realise it but there is a very hardworking Chip in your computer. In figure 12 you can see him working through this program.

At line 20 (INPUT NAME$) he marks up a label in the string stores ready for later use

When the name is actually entered, he writes it on the label.
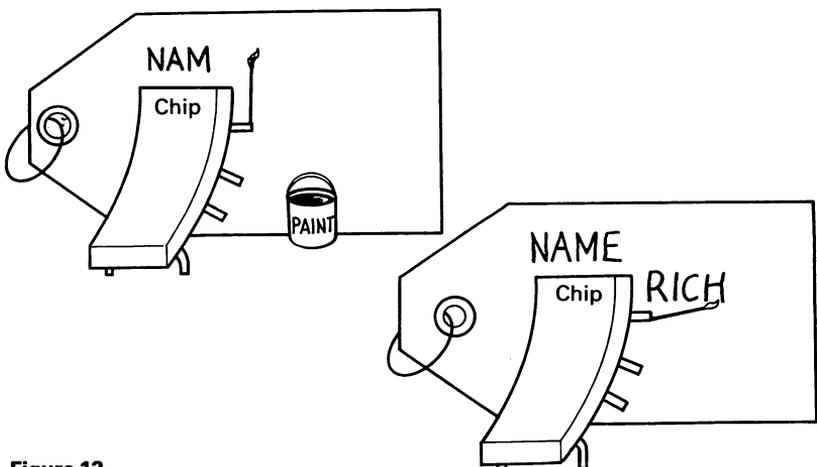


**Figure 12**

Later, at line 30 (**PRINT"HELLO THERE ";NAME$**) he goes back to the store to see what was written there, so that he can print it out.
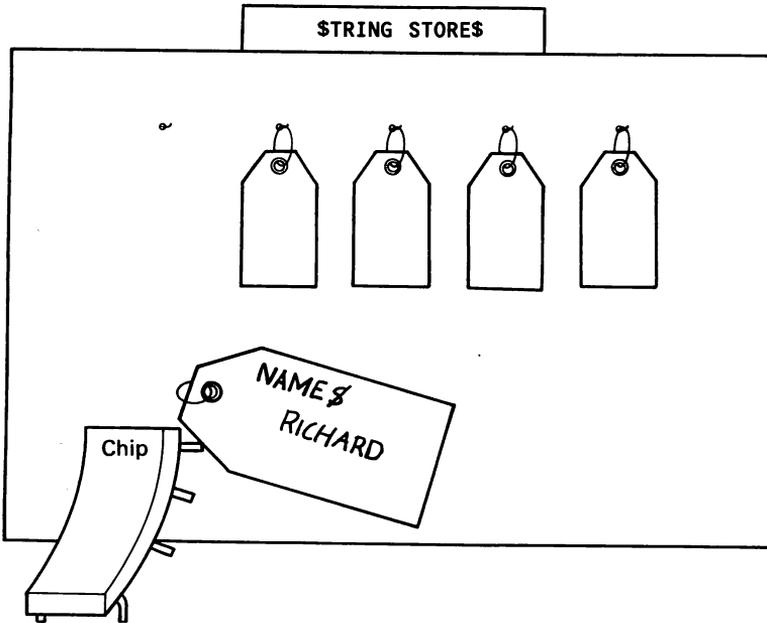


**Figure 13**

Run it a few times and type in different names. (Don't forget to press ENTER after you have finished typing.)

Figure 13 shows what happens when first Richard, and then Amanda answers the computer's question.

Whatever is written on a string store label stays the same until something new is written there, or until the whole program is wiped clean with NEW. You can see this if you run the program, and then, after the * DONE * message, type in (no line number):

**PRINT NAME$**

You should see the last name that was entered.

Let's add one more line, to keep the program running:

```
40 GO TO 10
```

Now invite the rest of the family in, and you should finish with a screen something like figure 14.

When you run out of relatives and friends, press FCTN and 4 CLEAR.
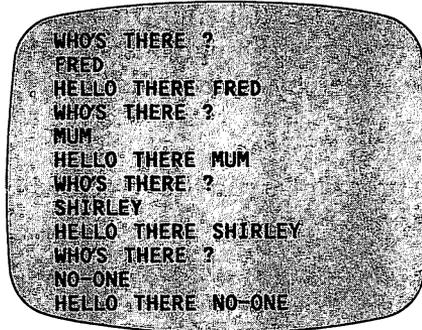


**Figure 14**

Notice that last INPUT answer. This shows an important point about computers. They are very stupid and do not think about what they are doing, unless you make them. That is what we are going to do now.

Add these two lines to the program.

```
25 IF NAME$ ="NO-ONE" THEN 50
50 STOP
```

IF and THEN mean the same in BASIC as they do in ordinary English. At line 25 the computer will compare the word on the NAME$ label with "NO-ONE" and IF they are the same THEN it will jump to line 50. IF they are not the same it will simply carry on to line 30.

You can change this program to set a trap for a friend. EDIT line 25 and type a friend's name in place of NO-ONE. If this name is shorter than "NO-ONE" then you will need to

rub out the remaining letters using the DELETE key. (Hold down FCTN and press ⊡.)

If your friend's name is longer, you will need to make more space. You can do this by using INSERT (hold down FCTN and press ⊡) when you run out of room. Now type in the rest of the name and the second half of the line will shuffle up to make space.

Change line 50 so that there is a special message for your friend. Make it quite different from the normal "HELLO THERE", and make sure that your friend sees a few normal runs round the loop first. It might also be useful to wipe your program off the screen by adding this line:

```
5 CALL CLEAR
```

Now when the program is run the screen will look like figure 15 at the start.
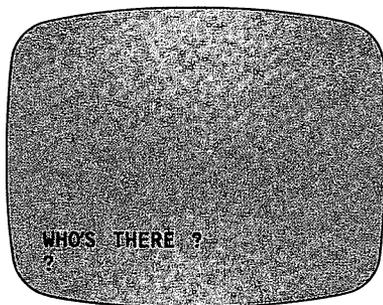


**Figure 15**

NOTE: Don't be too disappointed if the trick doesn't work. To make it work, you must have in line 25 exactly the name which your friend will use. Even if he types the name the same, but adds a space, it won't work. As far as the computer is concerned "FRED" and "FRED " are two different things. To get a "thinking" computer you need a very clever program.

# 7
# Introducing flowcharts

As your programs become more complicated, so you will need to plan them more carefully. To help us with our program planning, computer scientists have developed FLOWCHARTS. A Flowchart is a diagram that shows the different steps a program must take.

Figure 16A shows the flowchart of the "WHO'S THERE?" program, as it was when we first wrote it.

```
   ( START )
       ↓
  PRINT
  "WHO'S THERE?"
       ↓
  INPUT
  NAME$
       ↓
  PRINT
  "HELLO THERE";
  NAME$
       
   ( STOP )
```
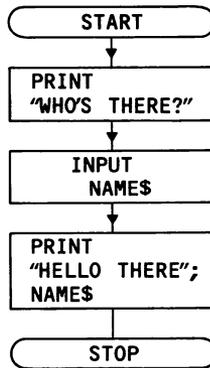
**Figure 16A**

The ends of the program are shown in oval shapes:

( START )   ( STOP )

Simple instructions are put in boxes:

PRINT "HELLO THERE ";NAME$

27
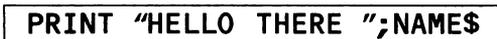
When we actually wrote the program it looked like this:

```
10  PRINT "WHO'S THERE?"
20  INPUT NAME$
30  PRINT "HELLO THERE "; NAME$
```

We didn't write START and STOP into our program, because we didn't need to. Somtimes you will need to write in STOP. It is a good habit to use them in your flowcharts, as they do make things clearer.

Here you see the flowchart for the second version of the program, where we had added a GO TO 10 line, so that it looped round to the beginning after each run.
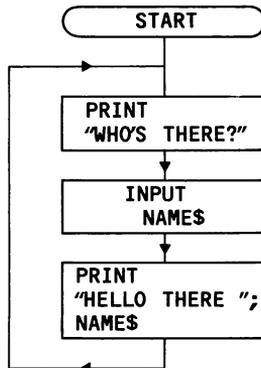


**Figure 16B**

In the third version of the program we added an IF. . .THEN. . . line. Here we are asking the computer to make a decision. To show this on a flowchart we use a diamond shape.
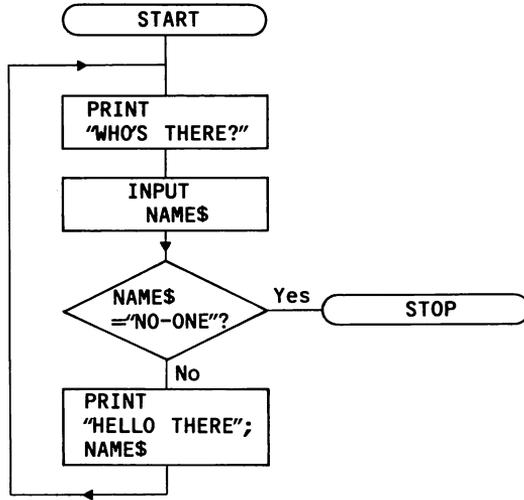
**Figure 17**

Notice that there are two lines leading from the diamond. The computer goes one way if the INPUT word is the one it is looking for, and the other way if it isn't.

Here's that version of the program. Compare it with the flowchart.

```
10 PRINT "WHO'S THERE ?"
20 INPUT NAME$
25 IF NAME$ ="NO-ONE" THEN 50
30 PRINT "HELLO THERE "; NAME$
40 GO TO 10
50 STOP
```

The flowchart in figure 18A is for a program that asks if a person likes computers. If the answer is "YES" the computer prints a special message; any other answer gets a different message. Work out the program lines using the flowchart.
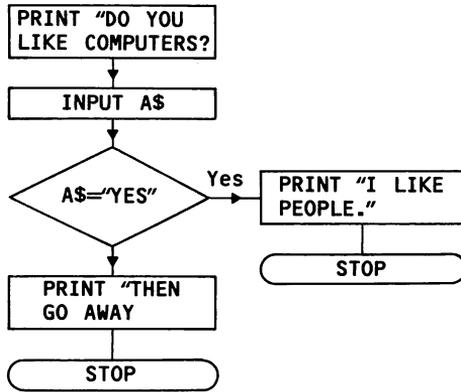
**Figure 18A**

Figure 18B is the same program with a little extra added. Now it checks to see if the answer is "YES" or "NO". If the answer is neither of these, it goes back to the INPUT line. Change your program to make it run this way.
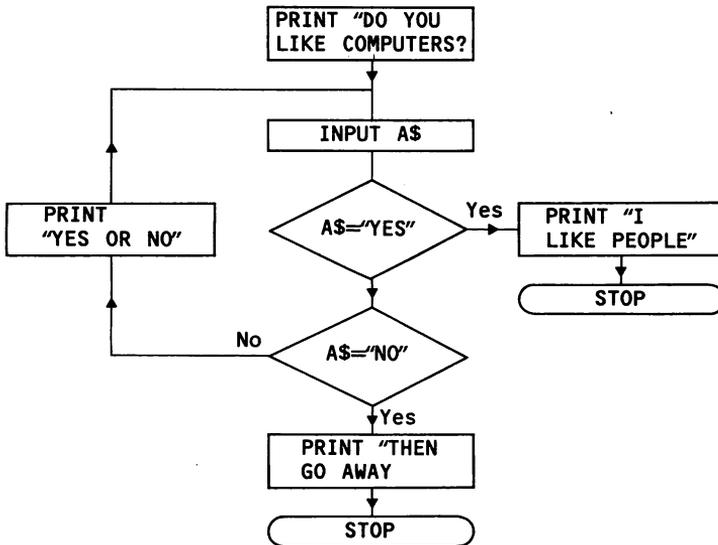


**Figure 18B**

# 8
# Working out sums

You can use the 99 as a calculator. If you tell it to PRINT and follow this with the sum WITHOUT QUOTES then it will simply print the answer. Try this. Type in:

**PRINT 2+2**    SHIFT and ⊞

Press ENTER and you should have a screen like figure 19.
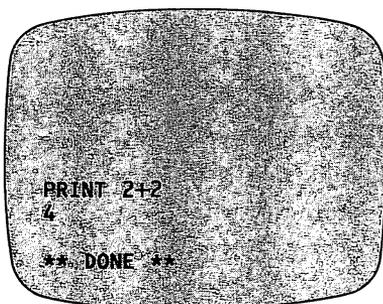


**Figure 19**

Try some sums of your own. Remember NO QUOTES. (See what happens when you do put quotes round the sum.)

**The symbols**
ADD (+)        SHIFT and ⊞
SUBTRACT (−)        SHIFT and �7
MULTIPLY (*)        SHIFT and ⬚8
DIVIDE (/)        �7 only

The 99 can also do far more complicated mathematics. If you are interested in this you will find out more about it in Pack 2.

WARNING!!
When you want the add sign, take great care to press SHIFT and ⊞. If you press FCTN and ⊞ you will get QUIT and lose your program!

# 9
# The number stores

You have already met the String Stores (or VARIABLES) where the computer keeps a note of words. There is another set of variables for numbers. You can think of number stores as a set of pigeon holes, like figure 20. Notice that the number stores are simply labelled with a single letter, or a group of letters, or letters followed by a number.

| A | AGE | A1 | WEIGHT |
|---|-----|-----|--------|
| HEIGHT | CLASS | Z | Z1 |
| | | | |

**Figure 20**

This program shows the use of those stores:

```
10 PRINT "HOW OLD ARE YOU ?"
20 INPUT AGE
30 PRINT "YOU ARE ";AGE
```

When you run the program, the number you type in at line 20 is stored in the box marked AGE. Here is what happens when first Richard, and then his grandfather type in their ages.

When Richard answers . . . . .



When Grandfather replies . . . . .



**Figure 21**

You will see that a variety of store names have been used in figure 20. What name you give to a store is up to you, as long as you follow these simple rules.

Don't use spaces in the name   (HOW MANY won't work)
Don't use the $ sign
Keep the name to less than 16 letters
Don't use any BASIC words.   (check with the list in Appendix B)

If the 99 doesn't like the name it will either tell you "* INCORRECT STATEMENT" when you try to enter the line, or you will get a "* BAD NAME" when you try to run the program.

The name you give to a store should mean something to you, and single letters or short words save typing errors. N1 for the first number; WT for weight; AGE; COST and so on.

# 10
# Comparing numbers

IF and THEN can also be used for checking numbers. They are a key part of any SUMS program. Here is the simplest type of sums program, where one person INPUTS two numbers and a second person INPUTS the answer. The computer then checks to see if it is right.

```
10  PRINT "FIRST NUMBER"
20  INPUT N1
30  PRINT "SECOND NUMBER"
40  INPUT N2
50  PRINT N1;"+";N2
60  INPUT A                    (A for Answer)
70  IF A=N1+N2 THEN 100
80  PRINT "WRONG.TRY AGAIN"
90  GO TO 60
100  PRINT "WELL DONE"
```

Notice how in line 70 we get the computer to find the right answer by adding the two numbers together.

You can merge lines 10 and 20 into one line:

```
10  INPUT "FIRST NUMBER ":N1
```

This will print the words "FIRST NUMBER" instead of a question mark in front of the INPUT cursor. Note the colon (:) after the prompt: before the store name. You might like to merge the other two PRINT and INPUT pairs into prompted INPUTS. If you want a question mark to appear, you must write it into the prompt. It will make for a neater screen if you also include a final space in the prompt to separate it from whatever is typed in.

## Greater or less?

When you compare numbers, you can also get the computer to check if one number is greater or less than the other. The signs > (SHIFT and [ · ]) and < (SHIFT and [ , ]) are used for this.

9 > 7   means 9 is greater than 7
2 < 5   means 2 is less than 5

We can get a game out of this sort of comparison. The first player types in a number, without the other player seeing. The screen is cleared, and then the second player tries to guess what the number was. This is what the program's flowchart looks like.
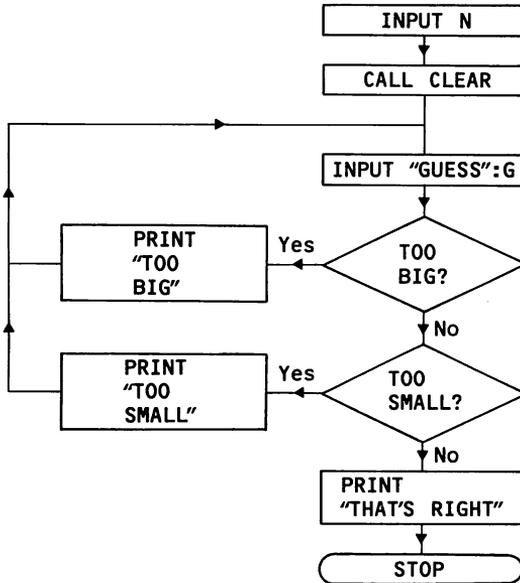
**Figure 22**

And here is the program:

```
10  INPUT "NUMBER ?":N      (Note you can't use
                            NUMBER as a store
                            name)

20  CALL CLEAR
30  INPUT "HAVE A GUESS ":G
40  IF G>N THEN 60
50  IF G<N THEN 80 ELSE 100
60  PRINT "TOO BIG"
70  GO TO 30
80  PRINT "TOO SMALL"
90  GO TO 30
100 PRINT "THAT IS RIGHT"
110 STOP
```

Look closely at line 50. If G is less than N then the computer jumps to line 80, otherwise (ELSE) it jumps to line 100.

IF. . .THEN. . .ELSE. . . is very useful if you have two different jumps for the computer. You would get the same effect if you missed off the ELSE 100, and wrote in an extra line:

```
55  GO TO 100
```

Notice also that you do not need a line to check if the Guess is the same as the Number. If it is neither greater nor less, then it must be the same.

This sort of game is O.K. if you have a second player at hand, but what if you haven't? You can turn the computer into the second player, and that is what we will do next.

# 11
# Random numbers

First a small detour. So far, when you have wanted to put a
number in a store you have used an INPUT line. You can
also give a value to your variables by writing it into the
program. Try this:

```
10 LET A= 99
20 PRINT A
```

Line 10 means "put 99 in the store labelled A".
   The "LET" is not actually needed. If you change line 10 to:

```
10 A=99
```

the program works just the same. Use LET if it makes the
line easier to find in a long program, but miss it out if you
want to save a little bit of memory space.

This line puts a RANDOM NUMBER into a store marked X.

```
10 X=RND
```

RND is short for RaNDom number. Add another two lines:

```
20 PRINT X
30 GO TO 10
```

Run the program, and BREAK when the screen starts to fill.
   You will see a lot of long numbers, all of 10 figures, and all
between 0 and 1. These are not nice numbers to try and
guess, but we can make them friendlier. Change line 20 to
this:

```
20 PRINT X*10
```

Run it again, and you will see the same numbers, but with
the decimal point moved up so that they are now between

0 and 10. All we really want right now is the whole number. We can get rid of the decimal part by using the INT command. This chops the decimals off, leaving just the INTEGER (the whole number). Change line 20 again:

**20 PRINT INT(X*10)**    (don't forget the brackets)

Run the program again, and you should get something like figure 23.
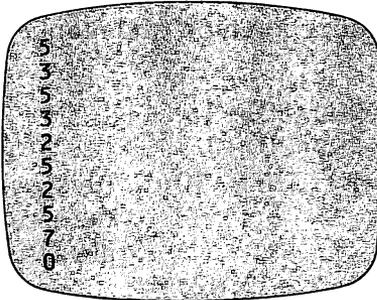


**Figure 23**

Run it again, and you will get exactly the same set of numbers. They are not exactly random are they?

This is because the random numbers are actually part of a very long sequence. The problem is, every time you run the program the sequence starts again at the beginning. Do not despair, there is an answer. Add an extra line at the beginning.

**5 RANDOMIZE**    (watch your spelling!)

Now run it again a couple of times. Better? RANDOMIZE gets the computer to pick a different place in the sequence to get the Random number from.

Change the number guessing game in chapter 10 so that the computer picks the number you must guess. Don't forget to add a RANDOMIZE line!

If you make line 10:

```
10 N =INT(RND*20)
```

you will be guessing at a number between 0 and 19. You will never get 20 because the INT command always rounds the number down to the nearest whole number. If you want your numbers in the range of 1 to 20 then make line 10:

```
10 N=INT(RND*20)+1
```

You can use random numbers in sums practice programs. The routine below is the basis of such a program. Here, the sums are of the subtraction type, and numbers up to 20 are used.

```
10  N1=INT(RND*20)+1
20  N2=INT(RND*20)+1
30  IF N2>=N1 THEN 10      (this makes sure that
                           the second number is
                           always smaller)
40  PRINT N1;" - ";N2;"="
50  INPUT A
60  IF A = N1-N2 THEN 90
70  PRINT "WRONG"
80  GO TO 40
90  PRINT "RIGHT"
100 GO TO 10      (this program goes on for ever)
```

Notice the double check in line 30:

```
IF N2>=N1 THEN...
```

>= means 'is greater than or equal to'
You can combine other comparisons.
<= means 'is less than or equal to'.
<> means 'is greater than or less than'.
That last one is very important. If a number is greater or less than another, then it is not equal to. Use <> when you want to check that two numbers are not the same.

Write a program of your own that will compare the user's age with your own, and print out a message something like

this – "YOU ARE OLDER THAN ME, JIM"
Here is a flowchart to help you.
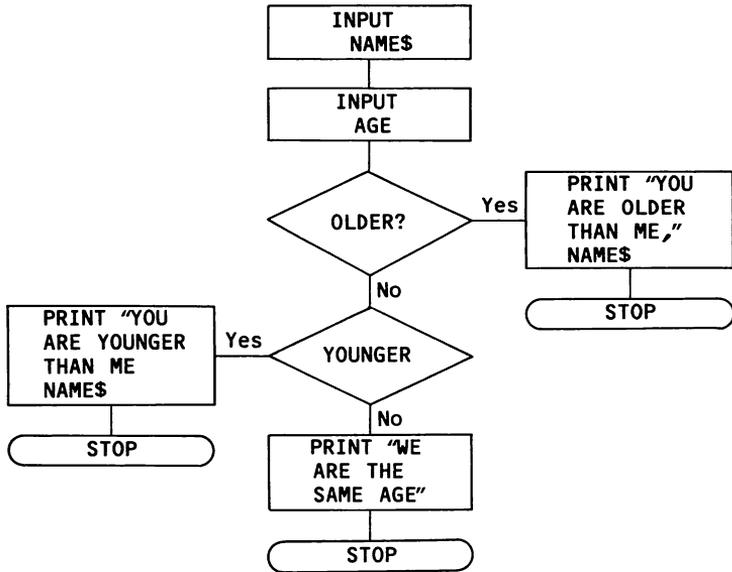


**Figure 24**

# 12
# Keeping count

In chapter 5 we had a version of the "WHO'S THERE?" program that made the computer say "HELLO THERE" to each new person, until "NO-ONE" was typed in. We can add a few more lines to that program so that the computer keeps a count of how many people there are. To do this we need to use another number store in which to keep count. We can watch a counter at work in this program.

```
10  LET C=0     (C for Count)
20  PRINT C     (so you can see C)
30  LET C=C+1
40  GO TO 20
```

You can see Chip keeping count in figure 25.

He starts by labelling a store, and putting a zero there.

LET C=0
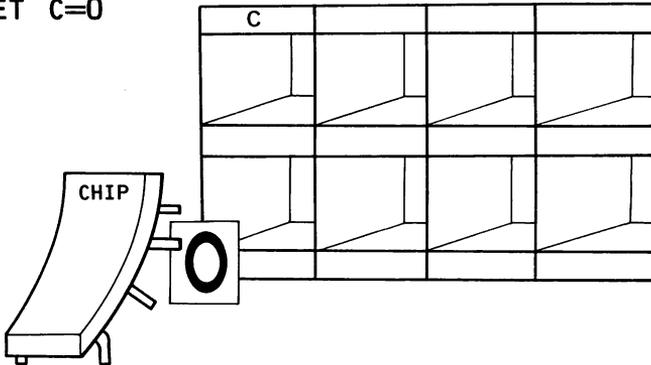


Then, each time round the loop, he changes the number in the C store, to make it one more.

LET C=C+1
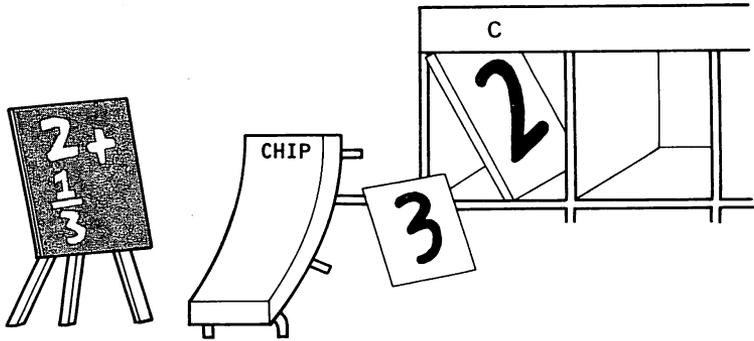
After 10 trips it looks like this.
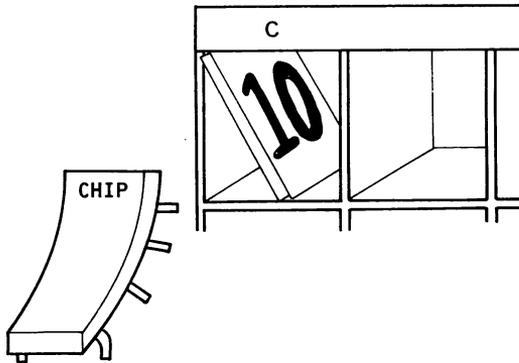


**Figure 25**

We can combine this with the original "WHO'S THERE?"
like this. The new lines are numbers 5, 35 and 50

```
 5  LET C=0
10  PRINT "WHO'S THERE ?"
20  INPUT NAME$
25  IF NAME$="NO-ONE" THEN 50
30  PRINT "HELLO THERE ";NAME$
35  LET C=C+1
40  GO TO 10
50  PRINT "THERE ARE ";C;"PEOPLE HERE."
```

Notice how the add-one-on line (35) is fitted into the loop so that it only works if there is some-one there. If it was fitted in between 10 and 25 somewhere, then it would also count "NO-ONE"! At line 50 the computer goes back to the C store and prints whatever number it finds there.

You could also add a counter to the number guessing game, so that when the number is guessed right the computer tells you how many guesses it took.

NOTE:  You already know that you can miss out a LET statement, and that "C=0" will work without it. You could miss out the whole of the line and the program would still work. If you don't open up a store, then Chip will do it for you automatically. Prove this to yourself. Type in (no line number):

**PRINT N**

What do you get? Now try "PRINT A", "PRINT B", "PRINT COUNT". The computer assumes you want number stores with those names, and labels them up. The value in the store will always be 0 at the start.

It is a good habit though to write in a "LET C=0" line. When you get to write more complicated programs you will find it very useful to have the point where the store is set up clearly marked by such a line.

# 13
# Times tables and other things

One of the beauties of computer programming is that there is never a single right answer to anything. There are always several ways of getting the same result. You can see this in the following examples. Each of these programs produces a screen like figure 26.

```
1        2
2        4
3        6
4        8
5        10
6        12
7        14
8        16
9        18
10       20
** DONE **
```
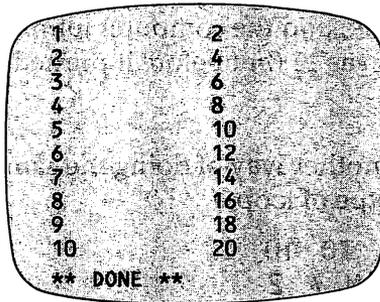
**Figure 26**

Here is the first:

```
10 LET N =1      (N for Number)
20 LET T=2       (T for Two Times)
30 PRINT N,T     (comma for half screen spacing)
40 LET N=N+1
50 LET T=T+2
60 IF N=11 THEN 80
70 GO TO 30
80 STOP
```

Notice that the stores N and T are given values of 1 and 2 when they are first opened. In the next version, only one number store is used, but the number is doubled in the PRINT line, so that the result is the same.

```
10  LET N=1
20  PRINT N,N*2
30  IF N=10 THEN 60
40  LET N=N+1
50  GO TO 20
60  STOP
```

Two points to note here.
1   You can do more or less anything to a number in a store,
    add, subtract, multiply or divide by any other number.
2   Here the IF. . .THEN. . . line that gets you out of the
    loop comes directly after the PRINT line, so it stops after
    10. In the first of these programs, the line came after the
    add-one-on line, and the computer jumped out of the
    loop when N was 11 (but before it printed it). The result is
    still the same.

And here is yet another way of doing the same thing. This
uses a different type of loop.

```
10  FOR N=1 TO 10
20  PRINT N,N * 2
30  NEXT N
```

This is called a FOR. . .NEXT. . . loop. What happens here is
that the computer takes every number from 1 to 10 in turn,
and does the same thing to each. When it runs out of
numbers, it stops. You do not need an IF. . .THEN. . . line
to check that a certain number has been reached.
    FOR. . .NEXT. . . loops are very useful wherever you
want something to happen for a set number of times. When
the line reads "FOR. . .TO. . ." the computer will always
work through the numbers one at a time, but we can make it
take bigger, or smaller steps if we add a little more. Look at
this:

```
10  FOR N=0 TO 20 STEP 2
20  PRINT N
30  NEXT N
```

Type it in and run it. Now change line 10, so that when you run the program you finish up with a screen like figure 27.
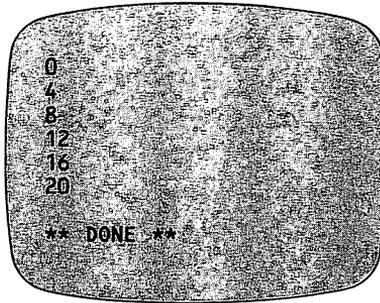


**Figure 27**

See what happens when you change line 10 to:

```
10 FOR N=0 TO 10 STEP .5
```

You can also STEP backwards.

```
10 FOR N=10 TO 0 STEP -1
20 PRINT N
30 NEXT N
40 PRINT "BLAST OFF"
50 PRINT
60 GO TO 50
```

What happens when you run it?

You probably found that the program went through too fast for you to see it very well. We can slow things down by using a delay loop. What we are doing here is asking the 99 to do nothing, but it does nothing so quickly that we need to ask it to do nothing lots of times to get any real delay. Add in these lines:

```
22 FOR D=1 TO 100      (D for Delay)
24 NEXT D
```

This is what is known as an 'empty loop'. By making the 99 whizz round doing nothing 100 times every time it gets to that part of the program, we slow things down nicely. You might like to add another pair of lines after the BLAST OFF to slow down the scrolling.

NOTE – WATCH YOUR STEP.
Two minor points about using STEPs.
1    It doesn't matter if your last STEP takes you past the last number in the loop.

```
FOR N=1 TO 10 STEP 2
```

will go through 1, 3, 5, 7, 9 and then stop.
2    It does matter if you STEP the wrong way.

```
FOR N= 1 TO 10 STEP -1
```

will work once only (when N is still 1). After the first STEP N will be 0, which is outside the range of the loop.
    And a last point about FOR. . .NEXT. . . loops. The variable name need not be a single letter.

```
FOR TIMES = 1 TO 10
```

will work, as will

```
FOR T = 1 TO 10.
```

# 14
# Sound effects

The 99 can produce a variety of sounds, both musical and otherwise. You have probably already had a look at the EFFECTS program, but if you haven't, load it in now and have a listen to some of the possible sound effects you can get from the 99.

These effects are all produced using the CALL SOUND routine. To make it work you need to tell the computer three things about each sound you want – how long it is to last, what type of sound you want, and how loud it should be.

## Duration

A sound can last anything between 1/1000th of a second, and 4¼ seconds. Each unit of time for the computer is 1/1000th of a second, so the number to type in for one second of a sound is 1000.

## Type or pitch

Sounds can be of two types, noises, and musical tones. We will look at these in the programs below.

## Volume

You can fix the volume by typing in a number between 0 and 30. 0 is the loudest, and 30 is completely silent. (Sometimes silent noises are useful!) As the sounds are actually produced through your T.V. set, you can further adjust the volume by using the T.V. controls.

The three numbers fit together like this:

```
CALL SOUND(1000,200,10)
```

This sound lasts for 1 second, has a pitch of 200 Hertz (which is just above the G below middle C) and has a volume level of 10.

## Noises

There are 8 types of noises available, and they have code numbers between −1 and −8. Note that they are negative numbers. Musical tones are positive numbers. You can listen to these sounds by running the 8 numbers through a loop.

```
10  FOR N= −1 TO −8 STEP −1        (you must
                                   STEP
                                   backwards
                                   through
                                   negative
                                   numbers)
20  CALL SOUND(1000,N,10)
30  NEXT N
```

Type this in and run it.

Noises −1, −2 and −3 are all steady beeps of different pitches. −4 gives you short regular beeps.

−5, −6 and −7 produce what is known as 'white noise'. You have probably come across it as a sound effect on games of the space invader type. −8 is also a 'white noise', but this comes as a series of short crackles.

These noises can be combined with musical tones to produce different effects.

## Musical tones

These will be dealt with in much more detail in Pack 2, but let us have a quick look at the range of sounds that can be produced. Type this in:

```
10  FOR P = 110 TO 40000 STEP 100 .
                                   (P for Pitch)
20  CALL SOUND(100,P,10)
30  NEXT P
```

That really is 40000 in line 10! The 99 can make sounds as high as 44733 Hertz. This is way above anything any person can hear. Because the range is so huge we will include a STEP in line 10. Without it this program would take over an hour to run!

When you run this program have your fingers poised over the FCTN and CLEAR keys. Break into the program at the point where you can no longer hear the tone. (There will still be the 'click' which marks the end of each separate sound.) Now type in:

**PRINT P**   (no line number)

What number do you get? It should be something between 10000 and 18000. Different people have different top limits to their hearing, and children can usually hear very high sounds better than older adults.

By fixing the time and pitch of the notes carefully, you can get the 99 to play tunes. This is very fiddly though without using more complicated programming techniques, and we will leave it until Pack 2. If you do fancy struggling through some short tunes, here are the numbers you need for the notes of the scale of C.

| **Pitch** (frequency in Hertz) | **Note** |
|---|---|
| 262 | C |
| 294 | D |
| 330 | E |
| 349 | F |
| 392 | G |
| 440 | A |
| 494 | B |
| 523 | C |

These can be combined to give chords of 2 or 3 notes. Fix the time for the chord, and then the pitch and volume of each note.

**CALL SOUND(1000,262,5,330,5)**

This plays one second of C (at level 5) and E (at level 5).

The next line plays 2 seconds of F, A and C, all at level 8.

```
CALL SOUND(2000,349,8, 440,8,523,8)
```

You can also combine musical tones and noises. This sounds like a car horn.

```
CALL SOUND(1000,523,5,-1,5)
```

Your maximum at any one time is three notes and one noise.

## Fading away

If you control the volume of the sound by a loop, you can produce a fading effect, though not a completely smooth one. Try this:

```
10  FOR V = 0 TO 30
20  CALL SOUND(100,262,V,330,V,392,V)
30  NEXT V
```

Now change line 20 to this:

```
20  CALL SOUND(500,-5,V)
```

Could that be bursts of fire from a rocket ship as it climbs into the distance? Change line 10 to:

```
10  FOR V = 30 TO 0 STEP -1
```

and your noise gets steadily louder.

Look again at the sounds section on EFFECTS and take note of the lines which produce the particular effects. Experiment with some sounds of your own.

One point that you should notice from that program, and which you could put to good use yourself, is this – the 99 can do other things (printing on screen, for example) while a sound is being produced. The only thing it can't do is to produce another sound. You can make this work for you. If you want the program to wait until the end of a sound before it goes on to the next part, then put a silent noise at the end of your sound.

```
10  CALL SOUND(1000,262,10)
20  CALL SOUND(1,262,30)
30  PRINT"NEXT ITEM"
```

Line 10 produces one second of middle C. Line 20, with the volume set to 30 (silence) produces 1/1000th of a second of nothing. (There still must be a figure for pitch, even though nothing sounds).

Try those three lines, and then take out line 20, and try again.

# 15
# Remember, remarks can remind you

If you LIST any of the programs on the cassette, you will find in them a number of lines that start with REM. These lines all have some sort of note or REMARK written in them:

**100  REM THESE LINES PRINT THE ANSWERS**

. . . or something similar.

The computer ignores the REM lines. The only reason they are there is to make the program easier to read.

There are three times when easy reading is important in a program LIST.

1   When you are DE-BUGGING the program – that is sorting out the various mistakes that almost always find their way into everybody's programs.

2   When you come back to the program after a few weeks, or months. You will almost certainly have forgotten which store was used for which reason, and how the different routines fastened together. If there are no REMs to help you, it will take much longer to make any changes or additions.

3   When someone else looks at your program. That is the main reason why the REMs are in the programs on the cassette.

The REMarks should be close to the lines they refer to.

```
100  LET C=0
101  REM C IS THE COUNTER
. . . . .
. . . .
250  GO TO 500
251  REM WRONG GUESSES GO TO 500
```

A REM line can be as long as any other line in a program –
that is, it can actually take up to 4 lines of print on the screen.
Like everything else in a program though, REMs take up
memory space, so keep them short if you are writing a
lengthy program.

# 16
# Neater printing

You are already using PRINT SEPARATORS (;,:) in your
PRINT lines, to give some sort of spacing on the screen. You
have probably got double spacing in your lines of print by
including two colons at the end. In case you haven't try this:

```
10 FOR N=1 TO 10
20 PRINT " A FRIENDLY MESSAGE"
30 NEXT N
```

Run it, then EDIT 20, and add two colons at the end.

```
20 PRINT "A FRIENDLY MESSAGE"::
```

Now run it.

The extra spacing improves the appearance of the screen
and makes reading it a little easier.

Now here's another way to improve your print layouts.
Suppose you have written a program to print out a variety of
times tables (×2, ×3). You want the screen to look like
figure 28.

You could get this layout with a PRINT line like this

```
30 PRINT N;" ";N * 2;" ";N * 3
```

– except that it wouldn't quite work. When the computer
printed two-figure numbers it would push the spacing
slightly out of line.

Here is a program which does it, very neatly. It uses the
TAB instruction. TAB is short for TABULATOR, and is used
for printing TABLES.

```
10 PRINT "NUMBER"; TAB(15);" ×2";TAB(25);"×3"
```

TAB(15) means 'start at Column 15'. Notice the semi-colons

separating the TAB instructions from the things that need printing ("×2", "NUMBER", etc).

That first line will give us the headings to the table. Three more lines will produce the numbers.

```
20 FOR N=1 TO 10
30 PRINT TAB(5);N; TAB(15);N*2; TAB(25);N*3
40 NEXT N
```

When typing this in take great care over the punctuation, brackets and semi-colons, otherwise it won't work. An "* INCORRECT STATEMENT" message will mean that you have probably missed out a semi-colon.

Use the grid in figure 29 for working out PRINT lines using TAB. You will see that the actual range of column numbers is 1 to 28, but if you type PRINT TAB(30), it will still work. The computer will see that it is over 28, and take 28 off to produce TAB(2). The program below shows how the computer keeps the TAB numbers within the range allowed.

```
10 FOR N=1 TO 100
20 PRINT TAB(N);N
30 NEXT N
```

Run it and watch what happens. Add a delay loop between 20 and 30 to slow it down if you like.

This program also shows up another minor point about printing numbers. All positive numbers are printed with a space in front of them. (Negative numbers have a – sign there instead). This means that if you want a column of numbers to start at column 10, you must write PRINT TAB(9);. . . to allow for the space. You will also notice that when the print position gets close to the right hand side, so that the number should appear half on and half off the screen, the 99 pushes the print position back to the start of the line, so that the number can be printed in one.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Figure 29**

# 17
# Running totals

You read earlier how you can do anything to a number variable that you can do to a number. This includes adding (or subtracting, multiplying or dividing by) the number in another store. You can see it in this program.

```
10 LET T=0
20 INPUT N
30 LET T=T+N
40 PRINT T
50 GO TO 20
```

Run it, and type in the numbers from 1 to 6. You should get a screen like figure 30.

(1+2=3)
(3+3=6)
(6+4=10)
(10+5=15)
(15+6=21)
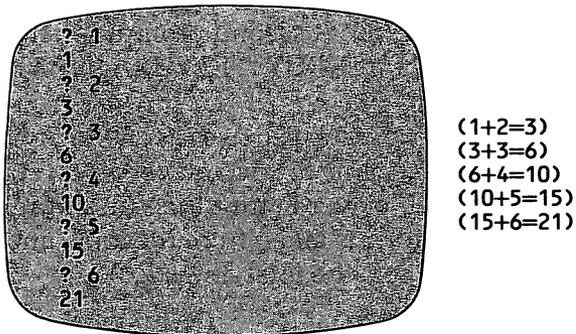
**Figure 30**

You could work this kind of totalling into a program that asked how many sweets you ate each day for a week (or how many hours each day you spent computing), and gave you a total for the week at the end.

A flowchart for this sort of program is shown in figure 31.

```
                    ┌─────────────┐
                    │   T = 0     │
FOR D = 1 TO 7      └──────┬──────┘
              ┌────────────┤
              │     ┌──────▼──────┐
              │     │ PRINT "DAY";D│
              │     └──────┬──────┘
              │     ┌──────▼──────┐
              │     │   INPUT N   │
              │     └──────┬──────┘
              │     ┌──────▼──────┐
              │     │  T = T + N  │
              │     └──────┬──────┘
NEXT D    ◄───┘            │
              ┌────────────▼──────┐
              │ PRINT "TOTAL      │
              │ THIS WEEK=";T     │
              └─────────┬─────────┘
                   ┌────▼────┐
                   │  STOP   │
                   └─────────┘
```
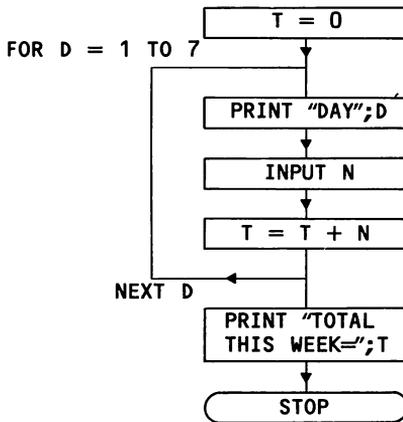
**Figure 31**

Notice how in this program you only want a limited number of INPUTS (7 Days in the week). It makes sense then to include the INPUT and totalling lines in a FOR. . .NEXT. . . loop. It is useful to have a prompt printed before the INPUT line –

PRINT "DAY ";D. A second prompt can then be included in the INPUT line – INPUT "HOW MANY THAT DAY ?":N.

The following program also uses a totalling type line, but in reverse. This is a game for two players. You start with a total of 100, then each player in turn types in a number between 1 and 9. This number is taken from the total, and the winner is the player who gets the total down to 0.

```
10  REM TAKING GAME
20  LET T=100
30  INPUT N
40  LET T=T−N
50  IF T=0 THEN 70
60  GO TO 30
70  PRINT "THE WINNER "
```

This basic game can be improved and altered in several ways.

You could add a line after the INPUT to make sure that the number is no more than 9, and a second check line to make sure that it is no less than 1.

You can vary the rules of the game. Reaching 0 could lose the game rather than win it. The range of permitted numbers could also be changed.

It might be useful to add a section at the beginning to print up the rules of the game.

All these changes could really mess up your line numbering, so it might be worth looking at another new command.

## Resequence

Try this when you have got a program in the computer. Type in (no line number):

**RESEQUENCE** (or RES – the short version works just as well)

You will find that your program has been renumbered for you. The lines are still in the right order, and the GO TO's still go to the right lines, even though they now have new numbers. These new numbers start at 100 and go up in 10's.

If you don't want your lines to be numbered this way, then you can control the renumbering.

**RES 10,10**

Will renumber so that the first line is 10 and the numbering goes up in 10's.

**RESEQUENCE 100,5**

Will start from 100 and go up in 5's.

However you want the numbers to run, the general shape of the command is always:
RESEQUENCE (first line),(interval)
or RES (first line),(interval)

While we are on the subject of line numbers, here's a tip for when you are next typing in a program. The 99 will do the numbering for you if you ask it. All you have to do is type in

**NUMBER**    (or NUM which works the same)

before you begin your program.

As soon as you press ENTER the number 100 will appear. Type your first line and enter it, and 110 pops up ready for the next line. New line numbers, spaced in 10's will keep coming as long as you want them. When you have finished, press ENTER after the next line number appears, and your program is ready to run.

If you don't want to start at 100 and work up in 10's, then you can fix your own numbering style just as you can with RESEQUENCE.

**NUM 10,5**

Will start at 10 and go up in 5's.

If you want to add lines to the end of a program, type in NUM followed by the next line number that you want. It will then number for you (in 10's) from that point.

You may find this very useful, especially when typing in programs from the book.

# 18
# The character set

Computers cannot think in words and letters. Numbers are all they can handle, (and binary numbers at that – see 'Creating your own characters' in Pack 2.)

Humans like to use words. BASIC translates words into numbers for the computer, and back for us. It does this by giving each letter and symbol a special code number. These codes are more or less the same on every computer, and are known as the ASCII codes. (American Standard Code for Information Interchange).

If you type in:

```
10  FOR N=1 TO  127
20  PRINT N, CHR$(N)      (CHR$(N) means the
                          character with code (N))
30  NEXT N
```

The complete set will be printed out on screen.

Notice that some of the characters do not appear, or are printed as blocks of dots at random. This is because some of the characters carry information for the computer's use only – where the cursor should move, and things like that. You will find a complete ASCII list in Appendix A.

You will notice that if you type in:

```
PRINT CHR$(65)
```

you get exactly the same as if you type in:

```
PRINT "A"
```

so you won't normally bother then to use the ASCII codes to print characters that you have got on the keyboard. You might use them though for characters that don't appear on the keys. We will come to them shortly. The character codes

are used in writing on the screen when you don't want your words to scroll up from the bottom.

You will see some printing like this in the cassette programs, and the techniques are covered in Pack 2.

Character codes are also used for crashproofing inputs, and for various other special effects. These are dealt with in Pack 2, and in the Games Packs. Finally, they can be used for coding messages, if you fancy going into the spying business. Here's how.

## Secret codes

Type this in:

```
PRINT ASC("A")
```

and you get – 65, unless you used a small "A", in which case you get – 97. ASC gives you the ASCII code number of the letter (or other character) in the quotes. It works with string variables as well, as you will see with this program.

```
10 INPUT C$    (any Character)
20 PRINT ASC(C$)
30 GO TO 10
```

Run this, and enter S,P,Y. You should see this:



**Figure 32**

64
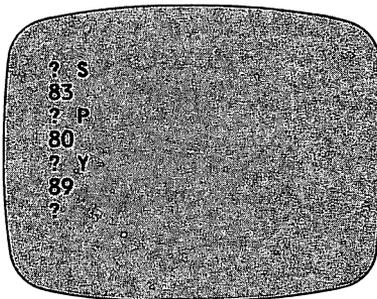
Use this program to turn a message into a series of code numbers. Write down the code numbers and keep them for later. You will need another program to turn your numbers back into letters. Here it is:

```
10  INPUT N      (the code Number)
20  PRINT CHR$(N)
30  GO TO 10
```

Use this to decode your message from earlier. You can also use it to decode the message that Chip has received.



8 7 6 9      7 2 6 5 8 6 6 9

6 6 8 2      7 9 7 5 6 9 7 8

8 9 7 9 8 5 8 2      6 7 7 9 6 8 6 9
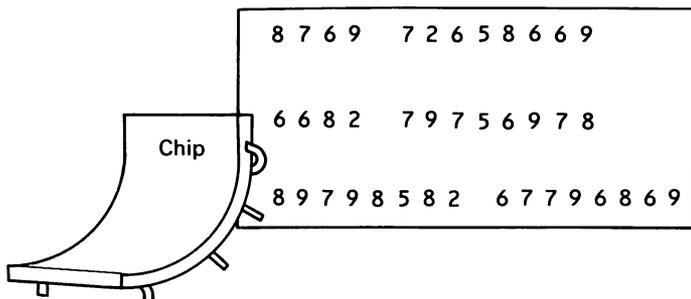
Chip

Why should Chip be worried?

**Figure 33**

Why should Chip be worried?

You can improve the security of your codes with a bit of number juggling. Change the encoding program to this:

```
10  PRINT C$
20  PRINT ASC(C$)+5      (extra number outside
                          brackets)
30  GO TO 10
```

Now when you enter "A" you will get 70 and not 65 as your special code number.

The decoding program needs the opposite.

```
10 INPUT N
20 PRINT CHR$(N—5)    (extra number inside
                       brackets)
30 GO TO 10
```

This 'code key number' can be anything you like. It could be written into the program as a variable, so that the same program could be used for different coded messages.

Alter the encoder program to this:

```
 5 INPUT "CODE KEY NUMBER":K
10 INPUT C$
20 PRINT ASC(C$)+K
30 GO TO 10
```

Write a decoder program to work the same way, and then try and decode this message from Chip. He seems to have left his code key number lying around. No wonder people keep breaking his codes!

| 9587938476 | 978793 | 84818377 |
| 73    828774 | 95819280 | 928077 |
| 917775907792 | 91779094817577 | |
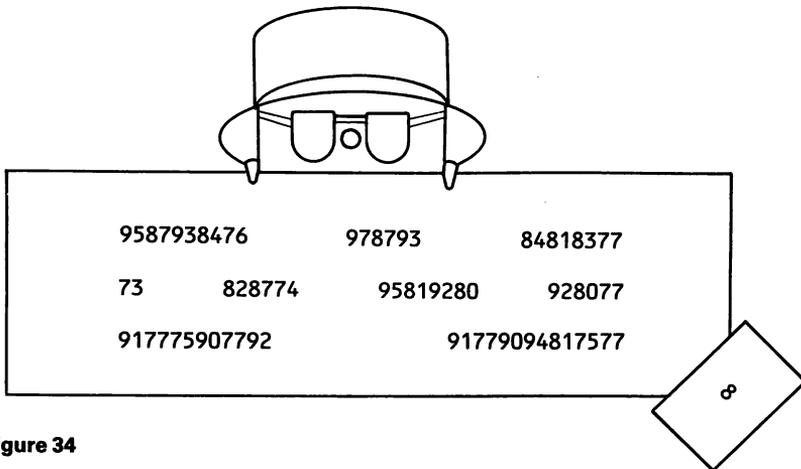
**Figure 34**

# 19
# Graphics

Some home computers have sets of block graphics built into them. The 99 doesn't. There are, however, 32 character codes available at the end of the ASCII set which can be defined to print new graphics characters using the CALL CHAR routine. (See Pack 2). The GRAPHICS program does just that. LOAD and run the program and you will have, for your use, the characters shown below. Once the 99 is switched off, these characters are, of course, lost. They can be restored by LOADing the program again.

| Character | Code | Character | Code | Character | Code | Character | Code |
|-----------|------|-----------|------|-----------|------|-----------|------|
|           | 128  |           | 136  |           | 144  |           | 152  |
|           | 129  |           | 137  |           | 145  |           | 153  |
|           | 130  |           | 138  |           | 146  |           | 154  |
|           | 131  |           | 139  |           | 147  |           | 155  |
|           | 132  |           | 140  |           | 148  |           | 156  |
|           | 133  |           | 141  |           | 149  |           | 157  |
|           | 134  |           | 142  |           | 150  |           | 158  |
|           | 135  |           | 143  |           | 151  |           | 159  |

**Figure 35**

There is a short demonstration written into the program to give you a few ideas of the sort of things which are possible. To convert the program for your own use, rub out the lines indicated by the REMs, and add your own program from line 30 onwards. There is well over 8k of memory left for your program, but if you do need more (it must be a long program!) then rub out the demonstration lines. The REMs will tell you when to stop.

## Using the graphics set

You can get to the graphics characters in three different ways. You can use the ASCII codes:

**PRINT CHR$(153);CHR$(154)**

. . . will print the two parts of the aeroplane.
    You can also find the characters on the keyboard, by holding down the CTRL key and pressing the letter keys.

**PRINT "ᴿ"**
CTRL and Y          CTRL and Z

. . . prints the aeroplane as well.
    Figure 36 shows where the different characters are on the keyboard.
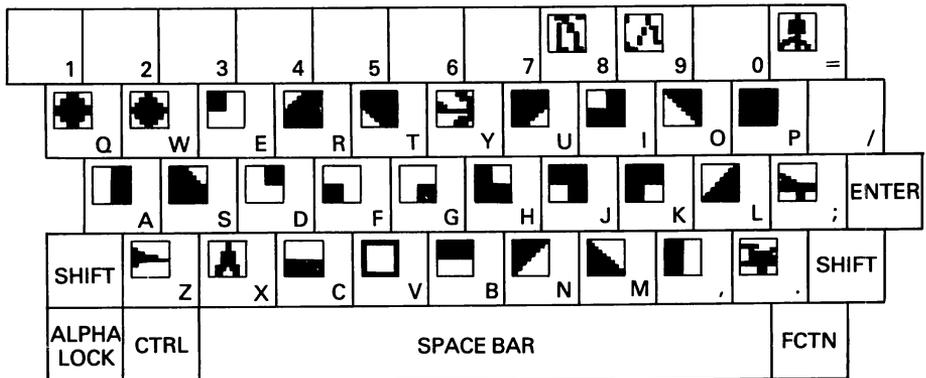


**Figure 36**

The third way is to use HCHAR or VCHAR commands. We will come to that later.

Any graphics printing starts on squared paper. Sketch a rough outline first, and then find the graphics shapes that fit best with what you want. Figure 37 shows the Mars 12 spaceship.
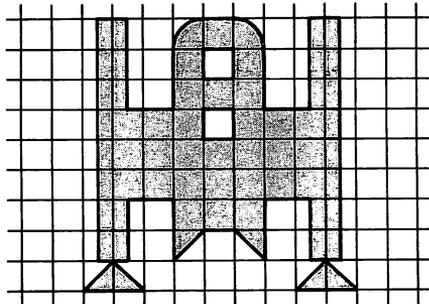


**Figure 37**

The picture now needs to be turned into a set of Print instructions. If you use CTRL and the letter keys, then the shapes will show up in the print lines. This makes it easier to check that you are typing it in correctly.

Work from the top down.

| | | |
|---|---|---|
| 100 | PRINT "▮▮ ◢▰▶ ▮▮" | ( A , space R P S space A , ) |
| 110 | PRINT "▮▮ ▪ ▪ ▮▮" | ( A , space P V P space A , ) |
| 120 | PRINT "▮▮ ▪▪▪ ▮▮" | ( A , space P P P space A , ) |
| 130 | PRINT "▮▪▪▪ ▪▪▪▮" | ( A P P P V P P P ,) |
| 140 | PRINT "▮▪▪▪▪▪▪▪" | ( A P P P P P P P ,) |
| 145 | PRINT "▮▪▪▪▪▪▪▪" | |
| 150 | PRINT "▮▮ ▪▪▪ ▮▮" | ( A , space P P P space A , ) |
| 160 | PRINT "▮▮ ◤ ◥ ▮▮" | ( A , space N space 0 space A , ) |
| 170 | PRINT "◣ ◢" | ( L M five spaces L M ) |

Figure 38 shows a patrol vessel of the Korth Imperial Space Navy. Work out the print lines needed to get this on screen.

69
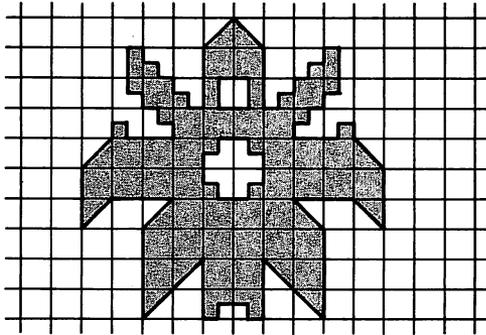
**Figure 38**

You can make your space ships take off up the screen by writing in a set of empty print lines after the graphics lines. The easiest way to do this is to use a loop.

```
200  FOR N= 1 TO 24
210  PRINT     (change the line numbers if your
                program goes beyond 190)
220  NEXT N
```

Hold it for a moment at the bottom of the screen by slipping a delay loop in before the empty print lines.

```
190  FOR D = 1 TO 500
195  NEXT D
```

You can slow the travel up screen by including a short delay between two of the lines of the Print loop.

```
214  FOR D=1 TO 50
216  NEXT D
```

Putting a Sound in the loop will also have a slow effect. Use this, instead of the delay.

```
215  CALL SOUND(100,-5,5)     (change the
                               numbers to suit
                               yourself)
```

The PRINT command will move pictures up the screen very nicely, but is little use for movement across. You can see it in this program. It runs a car across the bottom of the screen.

```
100  FOR C=1 TO 27    (Column)
110  PRINT TAB(C);"▄▄▄"    (CTRL ⊡ and ⨀)
120  CALL CLEAR    (to keep the printing on the
                    bottom line)
130  NEXT C
```

Runs across the screen? More like bounces! Do not despair, help is at hand.

# 20
# Putting things in the right place

So far, everything you have printed on the screen has scrolled up from the bottom. It is possible to put any character you like, letter, number, symbol or graphic anywhere you like on the screen. To do this we use one of two special routines, CALL HCHAR and CALL VCHAR. If you have not yet looked at the CHARLIES program, now is the time to do it.

Type this in:

```
10 CALL CLEAR
20 CALL HCHAR(12,16,42)
```

You should see a single asterisk in the middle of the screen. How did it get there? Look carefully at the three numbers in brackets in line 20. The first number (12) is the row; the second number (16) is the column; and the third number (42) is the ASCII code for asterisk.

The screen is divided into 24 rows and 32 columns for these routines. The numbering is shown in figure 39.

The question mark at the bottom of the figure was put there by this line.

```
CALL HCHAR(22,16,63)
```

Two lines were needed to get the "HI" at the top.

```
CALL HCHAR(1,15,72)
CALL HCHAR(1,16,73)
```

What three lines would you need to get the "END"?

Remember (Row, Column, Code number)

**Figure 39**

you might think that you need 10 lines to produce that sleepy line of Z's. You don't.

The HCHAR sub-program is specially designed to allow you to get lines of the same character. To do this, you need to tell the computer where to start, what character to print, and how many you want. (If you don't tell it how many, it only prints one, which is what happened earlier.)

```
CALL HCHAR(18,10,90,10)
```

produces the line of Z's.

HCHAR is short for HORIZONTAL CHARACTER REPETITION and the lines are always Horizontally across the screen. To get a Vertical line, you need VCHAR (VERTICAL CHARACTER REPETITION).

Try this:

```
CALL VCHAR(1,16,42,24)
```

You should have a line of asterisks down the middle of the screen.

HCHAR and VCHAR can be used together to draw pictures. The short program below produces the picture in figure 40. (If you have got the GRAPHICS program loaded in at the moment, change the code number from 42 to 144. Your 'box' will then have solid edges, rather than asterisks.)

```
100  CALL CLEAR
110  CALL HCHAR(5,5,42,10)
120  CALL VCHAR(6,5,42,9)
130  CALL VCHAR(6,14,42,9)
140  CALL HCHAR(15,5,42,10)
```
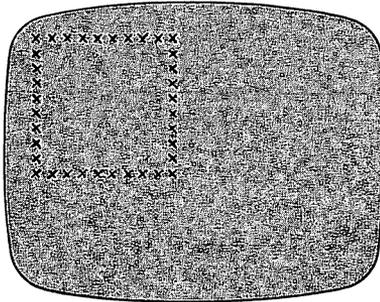


**Figure 40**

Change the Row and Column numbers so that the same box is drawn in the middle of the screen. Now add five more lines to get the robot's face in figure 41.

Check your program with the one in figure 42. Don't worry if your row and column numbers are slightly different.
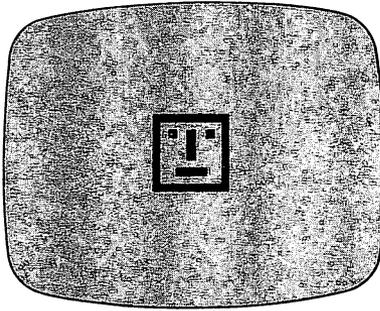
**Figure 41**

```
100 CALL CLEAR
110 CALL HCHAR(8,11,144,10)
120 CALL VCHAR(8,12,144,9)
130 CALL VCHAR(8,20,144,9)
140 CALL HCHAR(18,11,144,10)
150 CALL HCHAR(11,13,144)
160 CALL HCHAR(11,18,144)
170 CALL VCHAR(11,15,144,3)
180 CALL VCHAR(11,16,144,3)
190 CALL HCHAR(15,14,144,4)
```

character 144 from the GRAPHICS set use 35 or 42 if GRAPHICS not there

(one character only, VCHAR works just as well)

(nose)

(mouth)

**Figure 42**

# 21
# Coloured pictures

You saw earlier in 'Coloured Paper' how to change the screen colour, and you read then how each character space has two colours, its foreground (ink) colour, and a background (paper) colour. These colours can be any of the available 16.

For colour changing purposes the characters are grouped in sets of 8. To fix the colour of any one character, you have to fix the colour for all the characters of that set. However, the sets can all be very different from each other, which means that you can have 16 different colour combinations on screen at any one time. The program below shows this, but first, the character sets.

| Set number | ASCII codes | |
|---|---|---|
| 1 | 32–39 ⎫ | all punctuation signs or |
| 2 | 40–47 ⎭ | symbols |
| 3 | 48–55 | — numbers 0 to 7 |
| 4 | 56–63 | 8, 9 and punctuation |
| 5 | 64–71 ⎫ | |
| 6 | 72–79 ⎬ | large capital letters |
| 7 | 80–87 ⎭ | |
| 8 | 88–95 | X, Y, Z, a few symbols |
| 9 | 96–103 ⎫ | |
| 10 | 104–111 ⎪ | |
| 11 | 112–119 ⎬ | lower case letters |
| 12 | 120–127 ⎭ | |
| 13 | 128–135 ⎫ | blanks for your own |
| 14 | 136–143 ⎪ | characters. |
| 15 | 144–151 ⎬ | These are the ones used by |
| 16 | 152–159 ⎭ | the GRAPHICS program |

Next, the routine that changes the colour. It is one of the

TI BASIC sub-programs. To change the colour of Set 3 (numbers) to Red ink (code 9) on yellow paper (code 12) you would use this line:

```
CALL COLOR(3,9,12)
```
(note the spelling of COLOR. This is the American way)

And now, to save you turning back to the 'Coloured Paper' chapter.

| Colour code | Colour |
|---|---|
| 1 | Transparent |
| 2 | Black |
| 3 | Medium Green |
| 4 | Light Green |
| 5 | Dark Blue |
| 6 | Light Blue |
| 7 | Dark Red |
| 8 | Cyan |
| 9 | Medium Red |
| 10 | Light Red |
| 11 | Dark Yellow |
| 12 | Light Yellow |
| 13 | Dark Green |
| 14 | Magenta |
| 15 | Grey |
| 16 | White |

At last, the program to explore these colours.

```
10 PRINT .....
```
(a very long message using as many different characters as possible.)

```
20 INPUT "SET":S
```
(S = number of the character set)

```
30 INPUT "INK ":I
```
(I = code for foreground colour)

```
40 INPUT "PAPER ":P
```
(P = code for background colour)

```
50 CALL COLOR(S,I,P)
```
(watch the spelling)

```
60 GO TO 20
```

Run this and see how the colours change. You will notice that the actual characters remain the same (apart from scrolling steadily up the screen) while their colours are changed. You can fix the colours for a set of characters at any point in a program that you like. The character does not have to be on screen at that time. You can also make characters disappear by giving them the same colours as the screen. Some of the special effects you can achieve using the CALL COLOR routine are covered in the EFFECTS program. Others will be dealt with in Pack 2.

For the moment you may wish to try some coloured pictures using the GRAPHICS program's characters. They have been grouped to keep the same kind of character in each set for ease of colour changing. Fix the colours of each set separately, or make them all the same colour using this routine:

```
FOR S = 13 TO 16
CALL COLOR(S,16,1)
NEXT S
```

This makes them white (16) on a transparent (1) background. After these numbers to suit your pictures.

And finally, before we leave colour, you might like to try this routine from the EFFECTS program. It prints all the characters from 32 to 126 on the screen 7 times, so that the screen is more or less full. It then goes through the entire range of colour combinations, changing the colours of every set. Notice how the FOR. . .NEXT. . . loops are nested together. Whenever you have more than one loop running at any time, you must always close the last loop first.

```
     ┌──── 10  FOR T= 1 TO 7          (we'll do it 7 times.)
     │ ┌── 20  FOR N=32 TO 126        (the range of
     │ │                               characters we can
 T loop │                              print)
     │ N loop 30  PRINT CHR$(N);       (the semi-colon is
     │ │                               VITAL)
     │ └── 40  NEXT N                  (last loop first)
     └──── 50  NEXT T
     ┌──── 60  FOR P= 1 TO 16         (Paper =
     │                                 background)
     │ ┌── 70  FOR I=1 TO 16          (Ink = foreground)
 P loop │ ┌ 80  FOR S=1 TO 16          (every Set)
     │ I loop │
     │  S loop 90  CALL COLOR(S,I,P)
     │    └100  NEXT S                 (last loop first)
     │ └──110  NEXT I
     └────120  NEXT P                  (first loop last)
```
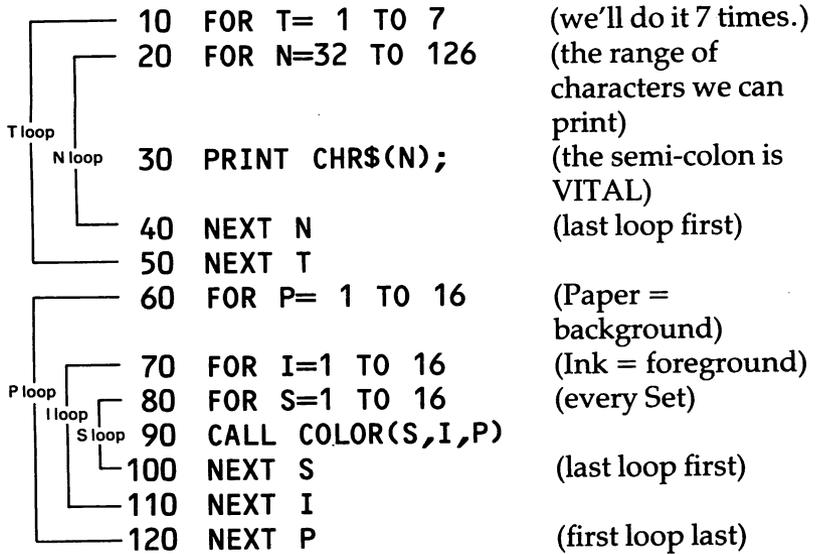
Run it. Fascinating, isn't it?

# 22
# Branching programs

Computers are much used for sorting and classifying information. They usually do this through some form of 'branching program' – that is, a program where the computer can go along any one of many different paths. At each dividing point on those paths is a question, and the way the computer goes depends upon the answer to that question. In figure 43 you can see the flowchart for a branching program that can identify (some) common pets. While the program itself is fairly long (see figure 44), it is actually made up of a series of simple stages. This is one of the attractions of branching programs. You can write very long and detailed programs that will sort a great many objects into their types, and it does not require any complicated programming skills. What it does take is an understanding of your subject, and a lot of patience.

Have a look at the way the PETS program was put together.

The first thing to decide was what categories the animals were to be sorted into. To keep it simple the program would only classify cats, dogs, rabbits, birds, monkeys and fish.

We now need a few key questions. The obvious one to start with was 'How many legs has it?'. There are 4 possible answers to this:

4 – so it must be either a cat, a dog, or a rabbit
2 – which leads to either a bird or a monkey
0 – so it must be a fish
Any other – the program won't recognise.

The key question for the 2-legged beast was whether or not it had feathers, to sort 'bird' from 'monkey'. There were 3 possible 4-legged animals, so 2 key questions are needed.

'Does it eat carrots?' – must be a rabbit.

'Does it bark?' – must be a dog if it does, and a cat if it doesn't.

Obviously, this is a very crude program, and you can think of all sorts of animals that it will not work for, but by adding more questions you could cover a wider selection of animals. The ideal PETS program would have a branch that lead to every possible type of pet. It could then be used to find out the name of an animal that the user did not already know himself. (You, the programmer, must have known it to be able to include it.)

```
START
  |
INPUT "HOW
MANY LEGS?"; L
  |
4? ──Yes──────────────────────────────────┐
  |                                         |
  No                                      REM
  |                                      4 LEGS
2? ──Yes──→  REM                            |
  |          2 LEGS                      EAT ──Yes──→
  No            |                     CARROTS?        "RABBIT"
  |         FEATHERS? ──Yes──→                          |
0? ──Yes──→                    |                       STOP
  |            |               |
  No           |               |
  |            |               |            BARK? ──Yes──→
"DON'T KNOW"  "FISH"  "MONKEY"  "BIRD"                   "DOG"
  |            |        |        |            |            |
 STOP         STOP     STOP     STOP         "CAT"        STOP
                                              |
                                             STOP
```
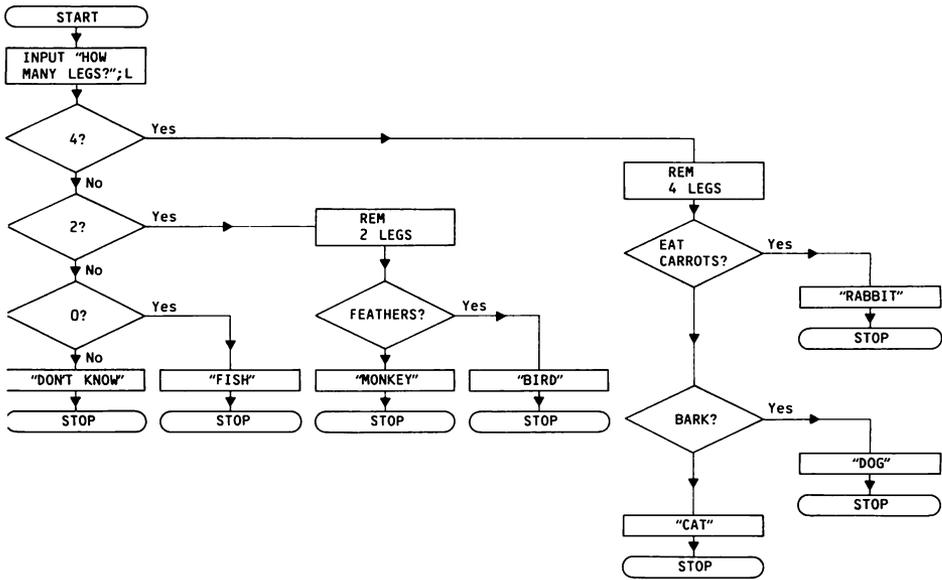
**Figure 43**

```
10  REM PETS
20  INPUT "HOW MANY LEGS HAS IT GOT ?":L
30  IF L=4 THEN 80
40  IF L=2 THEN 210
50  IF L=0 THEN 290
60  PRINT "I DON'T KNOW ONE WITH ";L;" LEGS"
70  STOP
80  REM 4-LEGGED BEASTS
90  INPUT "DOES IT EAT CARROTS?(Y/N)":A$
100 IF A$="Y" THEN 120
110 IF A$="N" THEN 140 ELSE 90
120 PRINT "IT'S A RABBIT"
130 STOP
140 INPUT "DOES IT BARK ?(Y/N)":A$
150 IF A$="Y" THEN 170
160 IF A$="N" THEN 190 ELSE 140
170 PRINT "IT'S A DOG"
180 STOP
190 PRINT "IT'S A CAT"
200 STOP
210 REM 2-LEGGED BEASTS
220 INPUT "HAS IT GOT FEATHERS ?(Y/N)":A$
230 IF A$="Y" THEN 250
240 IF A$="N" THEN 270 ELSE 220
250 PRINT "IT'S A BIRD"
260 STOP
270 PRINT "IT'S A MONKEY"
280 STOP
290 PRINT "IT'S A FISH"
300 STOP
```

**Figure 44**

## Note:   INPUT routines

Notice how the INPUT routines starting at lines 90, 140 and
220 all follow the same style. The INPUT prompts include
the acceptable answers (Y for Yes, N for No). This allows for
much easier checking. If you let the user reply anyway he

wanted, then you could have all sorts of replies
"Y","YES","SOMETIMES","ONLY IF I LET IT",
"N","NO"."NEVER". etc. You need an IF. . .THEN. . . line
to check every acceptable answer. Life is much easier if you
are only looking for "Y" and "N".

As the program stands, any answer apart from "Y" or "N",
will simply sent the computer back to the INPUT line.

You should also notice that each INPUT is taken into the
same string store (A$). There is no special reason for this.
You could just as well have used three different stores for the
different questions. There is, however, no harm in using the
same store several times, for different things, as long as you
don't want to keep the answer for later use in the program. It
is in fact quite useful to always use the same variable names
for the same things in all your programs, so that, when you
look through an old program and see A$ you will know it is
used for a "Y/N" answer. N$ could be the user's name; N the
number in a FOR. . .NEXT. . . loop; S the score in a game;
G the number of goes you have had.

## Branching by numbers

Suppose your program had a key question which asked
"What is the animal covered in?", and the acceptable
answers were "fur", "feathers" or "scales". You could follow
the INPUT line with check lines like this:

```
IF A$="FUR" THEN 200
IF A$="FEATHERS" THEN 250
IF A$="SCALES" THEN 300
```

These lines will work very well, as long as your users can
spell properly. You can avoid the problems caused by poor
spelling, or typing errors, by asking them to enter a number.

```
PRINT " WHAT SORT OF COVERING HAS IT ?"
PRINT " ENTER 1 FOR FUR"
PRINT "ENTER 2 FOR FEATHERS"
PRINT "ENTER 3 FOR SCALES"
INPUT C     (C for Covering)
```

The check lines are now simpler, and less likely to produce problems.

```
IF C=1 THEN 200
....
```

Now here's a command which will save a little typing. You can replace those three check lines by one instruction.

```
ON C GO TO 200,250,300
```

What happens now is that the computer goes to 200 if C is 1, 250 if it is 2 and 300 if it is 3.

To make this work the numbers that can be input must start from 1 and run in sequence. If 4 is entered the computer will stop with a * BAD VALUE report. To avoid this you will need check lines, immediately after the INPUT.

```
100  INPUT C
110  IF C<1 THEN 100
120  IF C>3 THEN 100
```

Where you have only 3 branches, and you want to prevent accidental errors, then ON. . .GO TO. . . is hardly worth using, but it can be very handy where there are many possible places for the computer to go to.

NOTE: If you use ON. . .GO TO. . . you will not need to worry about the awkward customer who argues that chickens have got scales and feathers, and enters 2.5. Chip will deal with them.
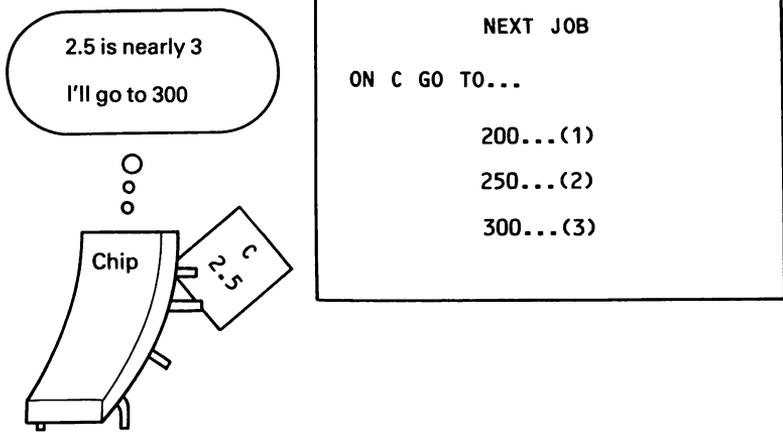
**Figure 45**

In cases like this the 99 automatically round up or down to the nearest whole number.

You will find another example of a branching program on the cassette. It is called TRANSPORT, and it will give you the names of a number of different vehicles in English, French or German. All you have to do is to answer the questions which let the computer classify the vehicle you are thinking about. It is put together in exactly the same way as the PETS program. The only complications are those needed for the three languages.

# 23
# Keyboard tricks and games

One of the 99's built-in routines reads the keyboard directly, without waiting for the ENTER instruction. This allows the computer to tell if a key is being pressed, and which key it is. The routine has several different forms, but right now we are only concerned with two of these. Look at this first:

```
10  CALL CLEAR
20  CALL KEY(3,K,S)
30  IF S=0 THEN 20
40  PRINT CHR$(K)
50  GO TO 20
```

Type it in and run it. Here is what is happening. The computer checks the keyboard, (line 20). It looks to see what key is pressed (K), and what the Status (S) of the keyboard is. There are three possible statuses; either no key is pressed, in which case S=0; or a new key has been pressed (S=1) or the same key is still being pressed (S=−1). In this program if no key is pressed the computer just waits until one is. The K variable here collects the ASCII code of the key which is touched, and this is turned back into the right letter or symbol by the CHR$ in line 40. Add an extra line to the program:

```
35 PRINT S
```

Now run it. Hold the same key down and you will see "−1" printed before the letter as it reappears each time round. This is the status part of the routine at work. Constant pressing gives the −1 report. Press the same key repeatedly, but taking your finger off in between, and you will see "1" printed for the status report. You will probably realise that this has important implications for games playing.

The next time that you know someone else will be using the computer, get there first, type this program in, run it, and leave it. When they ask you what you have been doing, say you were just checking something out. Watch what happens when they try to write a program!

```
10  CALL SCREEN (4)
20  CALL CLEAR
30  PRINT "TI BASIC READY"        (the screen now
                                   looks like a
                                   normal starting
                                   screen)

40  CALL KEY(3,K,S)
50  IF S=0 THEN 40
60  PRINT
70  PRINT "JUST A MOMENT"         (or any other
                                   suitable message)

80  PRINT
90  PRINT "I AM THINKING"
100 PRINT
110 FOR D=1 TO 5000
                                  (thoughtful delay)
120 NEXT D
130 PRINT "NEARLY READY"
140 PRINT
150 FOR D=1 TO 5000               (more thinking time)
160 NEXT D
170 CALL CLEAR
180 PRINT "TI BASIC READY"        (starting screen
                                   again)
190 GO TO 190     (this locks everything up)
```

You can make the joke as elaborate as your imagination will allow, and time it to last as long as the other user's good humour! (See the program KEYS).

Lines like those at 40 and 50 can be usefully added to many programs, where you want to let the user work through the program at his own pace. Whenever the computer comes to lines of that sort it will wait for a key contact. You will find them in most of the cassette programs.

There is one more thing on the CALL KEY brackets that

needs to be thought about. That is the 3. We are here using the third of the different CALL KEY routines. In this form the whole keyboard is seen in its normal state, with the full range of characters. Go round the keyboard trying the keys. Hold SHIFT down and press some more.

There is another version of CALL KEY which splits the keyboard into two parts. Change line 20 to:

```
20 CALL KEY(1,K,S)
```

Now run it. You should notice two things: nothing happens when you touch the right hand side of the keyboard, and pressing a key on the left side will give you a status report (1 or −1) but nothing else.

```
CALL KEY(1....
```

collects information from the left only, and it does not use ASCII codes. The keys are still coded, and you can see these by changing line 40:

```
40 PRINT K
```

The numbers you get will be between 0 and 19.

The right hand side of the keyboard is checked in exactly the same way, but using the line:

```
CALL KEY(2,...)
```

CALL KEY (1 . . .          CALL KEY (2 . . .

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | = |
| 19 | 7 | 8 | 9 | 10 | 19 | 7 | 8 | 9 | 10 | |

| Q | W | E | R | T | Y | U | I | O | P | / |
| 18 | 4 | 5 | 6 | 11 | 18 | 4 | 5 | 6 | 11 | 16 |

| A | S | D | F | G | H | J | K | L | ; | ENTER |
| 1 | 2 | 3 | 12 | 17 | 1 | 2 | 3 | 12 | 17 | |

| SHIFT | Z | X | C | V | B | N | M | , | . | SHIFT |
| | 15 | 0 | 14 | 13 | 16 | 15 | 0 | 14 | 13 | |

| ALPHA LOCK | CTRL | SPACE BAR | | | | | | | FCTN | |

**Figure 46**

Once again the codes for these keys are between 0 and 19. In figure 46 you can see how the 99 codes the keyboard when it is working in a half-board mode.

You can use CALL KEY(1. . . and CALL KEY(2. . . in the same program, next to each other. This allows you to use the 99 for two-player games. Here is a simple example of a split-keyboard game. Play it against a friend, or play your left hand against the right!

```
10  FOR D=1 TO 200        (short delay, much needed
                           on repeats)
20  NEXT D
30  LET A=INT(RND*2)+1     (either 1 or 2 at
                           random)
40  LET B=INT(RND*2)+1
50  CALL CLEAR
60  PRINT A,B
70  FOR T=1 TO 10          (ten times round
80  CALL KEY(1,K1,S1)      check left
90  CALL KEY(2,K2,S2)      check right
100 IF S1=1 THEN 140       go if touched)
110 IF S2=1 THEN 190
120 NEXT T                 otherwise . . .
130 GO TO 10      (two more numbers)
140 IF A=B THEN 170        (are the numbers the
                           same?)
150 PRINT "LEFT WRONG"     (. . . they weren't)
160 GO TO 10      (for another pair of numbers)
170 PRINT "LEFT RIGHT"
180 GO TO 10
190 IF A=B THEN 220
200 PRINT "RIGHT WRONG "   (this routine is the
210 GO TO 10               same as line 140 to
220 PRINT " RIGHT RIGHT"   180, except for the
230 GO TO 10               right side keys)
```

It's basically computerised 'Snap'. If the two numbers are the same, then the first player to press a key (any key on his half)

wins. Notice how lines 100 and 110 only accept new key touches. If a key is held down constantly, the Status (S1 and S2) report will be −1.

Variations:

1 Add two more number variables to keep track of the scores. (LEFTS and RIGHTS perhaps). You will also now need to add some means of escaping from the loops. A check line after the first CALL KEY will do the job:

```
85  IF K1=15 THEN 240      (15 is the code for
                            "A")
```

Here are the other lines you will need to add:

```
175  LET LEFTS= LEFTS+1
225  LET RIGHTS=RIGHTS +1
240  PRINT "LEFT ";LEFTS:" RIGHT ";RIGHTS
```

2 Instead of boring old numbers, why not make some nice graphics. Store them at Character 128, 129 and 130. To get the computer to pick one of those numbers at random use this sort of line

```
LET A= INT(RND*3)+128
```

Obviously, you can have as many characters as you like. The more there are, the less often the computer will come up with a matching pair. With only two characters, or numbers, they will match, on average, half the time. With three to choose from, one third of the pairs will match; one quarter with four, and so on.

3 Reduce the time the players have by putting a smaller number in the loop at line 70.

4 Add a set of instructions so that players who are new to the game can find out what they are supposed to do. RESEQUENCE the program, with the first line at 200, and you have got lots of room before the game itself, in which you can print out the rules.

# Appendices

# A
# ASCII codes

The set numbers are for use with the CALL COLOR
command. See chapter 21.

Code   Character

| | | | |
|---|---|---|---|
| **Set 1** | | | |
| 32 | | (space) | |
| 33 | ! | (exclamation mark) | |
| 34 | " | (quote) | |
| 35 | # | (hash – number sign) | |
| 36 | $ | (string – dollar sign) | |
| 37 | % | (per cent) | |
| 38 | & | (and) | |
| 39 | ' | (apostrophe) | |
| | | | |
| **Set 2** | | | |
| 40 | ( | (open bracket) | |
| 41 | ) | (close bracket) | |
| 42 | * | (asterisk – multiply) | |
| 43 | + | (plus) | |
| 44 | , | (comma) | |
| 45 | – | (minus) | |
| 16 | . | (full stop) | |
| 47 | / | (divide) | |
| | | | |
| **Set 3** | | | |
| 48 | 0 | | |
| 49 | 1 | | |
| 50 | 2 | | |
| 51 | 3 | | |
| 52 | 4 | | |
| 53 | 5 | | |
| 54 | 6 | | |
| 55 | 7 | | |
| | | | |
| **Set 4** | | | |
| 56 | 8 | | |
| 57 | 9 | | |
| 58 | : | (colon) | |
| 59 | ; | (semi-colon) | |
| 60 | < | (less than) | |
| 61 | = | (equals) | |
| 62 | > | (more than) | |
| 63 | ? | (question mark) | |
| | | | |
| **Set 5** | | | |
| 64 | @ | (at sign) | |
| 65 | A | (Upper case capitals) | |
| 66 | B | | |
| 67 | C | | |
| 68 | D | | |
| 69 | E | | |
| 70 | F | | |
| 71 | G | | |

| Code | Character | | | |
|------|-----------|--|------|---|
| **Set 6** | | | 100 | D |
| 72 | H | | 101 | E |
| 73 | I | | 102 | F |
| 74 | J | | 103 | G |
| 75 | K | | | |
| 76 | L | | **Set 10** | |
| 77 | M | | 104 | H |
| 78 | N | | 105 | I |
| 79 | O | | 106 | J |
| | | | 107 | K |
| **Set 7** | | | 108 | L |
| 80 | P | | 109 | M |
| 81 | Q | | 110 | N |
| 82 | R | | 111 | O |
| 83 | S | | | |
| 84 | T | | **Set 11** | |
| 85 | U | | 112 | P |
| 86 | V | | 113 | Q |
| 87 | W | | 114 | R |
| | | | 115 | S |
| **Set 8** | | | 116 | T |
| 88 | X | | 117 | U |
| 89 | Y | | 118 | V |
| 90 | Z | | 119 | W |
| 91 | [ | (square bracket) | | |
| 92 | \ | (slant) | **Set 12** | |
| 93 | ] | (square bracket) | 120 | X |
| 94 | ∧ | (exponential power) | 121 | Y |
| 95 | — | (underline) | 122 | Z |
| | | | 123 | { | (left brace) |
| **Set 9** | | | 124 | ¦ |
| 96 | \ | (grave) | 125 | } | (right brace) |
| 97 | A | (Lower case capitals) | 126 | ~ | (tilde) |
| 98 | B | | 127 | | (this is DELETE) |
| 99 | C | | | |

The characters at ASCII 128 to 159 (sets 13 to 16) are left blank
to be defined by the user for his own graphics.

# B
# BASIC words

This is a complete list of the words used in TI BASIC. Almost half of these have been dealt with in this Pack, the others are covered in other books in this series. After each word you will see either a number or an abbreviation. These tell you where the word is first used. The numbers refer to the chapters of this book. The abbreviations are : P2 Starter Pack 2; G1 Game Writer's Pack 1; RK Record Keeper's Pack.
    None of these words may be used as variable names.

| | | | |
|---------|-----|----------|------------|
| ABS | P2 | EOF | RK |
| APPEND | RK | EXP | P2 |
| ASC | 18 | FIXED | RK |
| ATN | P2 | FOR | 13 |
| BASE | P2 | GO | 4 |
| BREAK | P2 | GOSUB | P2 |
| BYE | P2 | GO TO | 4 |
| CALL | 3 | IF | 7 |
| CHR$ | 18 | INPUT | 6 |
| CLOSE. | RK | INT | 11 |
| CON | P2 | INTERNAL | RK |
| CONTINUE | P2 | LEN | P2 |
| COS | P2 | LET | 11 |
| DATA | P2 | LIST | 3 |
| DEF | G2 | LOG | P2 |
| DELETE | RK | NEW | 3 |
| DIM | P2 | NEXT | 13 |
| DISPLAY | P2 | NUM | 17 |
| EDIT | 4 | NUMBER | 17 |
| ELSE | 10 | OLD | Appendix C |
| END | P2 | ON | 22 |

| OPEN | RK |
|------|-----|
| OPTION | P2 |
| OUTPUT | RK |
| PERMANENT | RK |
| POS | P2 |
| PRINT | 2 |
| RANDOMIZE | 11 |
| READ | P2 |
| REC | RK |
| RELATIVE | RK |
| REM | 15 |
| RES | 17 |
| RESEQUENCE | 17 |
| RESTORE | P2 |
| RETURN | P2 |
| RND | 11 |
| RUN | 3 |
| SAVE | Appendix C |
| SEG$ | P2 |
| SEQUENTIAL | RK |
| SGN | P2 |
| SIN | P2 |
| SQR | P2 |
| STEP | 13 |
| STOP | 6 |
| STR$ | P2 |
| SUB | P2 |
| TAB | 16 |
| TAN | P2 |
| THEN | 7 |
| TO | 13 |
| TRACE | P2 |
| UNBREAK | P2 |
| UNTRACE | P2 |
| UPDATE | RK |
| VAL | P2 |
| VARIABLE | 6 |

# C
# Using the cassette

## Connecting the machines

The 99 has routines built into it to take much of the sweat out of recording programs on tape and loading them back into the computer at a later date. Connect you machinery up properly to start with and then follow a few simple rules.

You will need a TI Dual Cassette Interface Cable and a reasonable cassette recorder.

The machine MUST have

Sockets for MICROPHONE

   EARPHONE (or external speaker)

   REMOTE CONTROL

Controls for Volume

   and Tone (you might survive without this)

a DIGITAL TAPE COUNTER is very useful for finding programs, but is not essential.

The majority of recorders will work perfectly well, but if you do seem to be having trouble in saving or loading, then check with your Texas dealer.

Notice that the cassette lead has two separate cables, one ending in three plugs, the other ending in two. You are only interested in the three plug end at the moment. The other lead is for connecting up a second cassette for particular types of file handling programs. (See the Record Keeper's Pack.)

Plug the flat 9-pin end into the socket next to the mains lead at the back of the 99 and connect the three jack plugs to the cassette recorder like this:

 RED lead to MICrophone socket

 WHITE lead to EARphone socket

 BLACK lead to REMote control socket (this is a mini-jack plug.)

T.V. lead

mains

black

red

white

REM
MIC
EAR

lead for second cassette
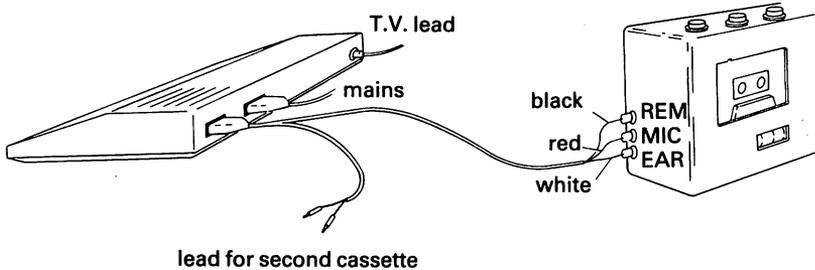
For best results position your recorder at least two feet away from the T.V. set, as the magnetic fields from the T.V. may interfere with the magnetic fields of the recorder itself and spoil your recordings.

Tune the tone control of the recorder up to maximum treble, and turn the volume control about half way up. Check that your recorder batteries are in good condition, or that the machine is plugged into the mains, and you are ready to go.

## Loading a program from a cassette

The first thing to do is find your program. When you record your own programs you will do well to keep a careful note of the tape counter position at the start of each new program. Counter numbers are given on the cassette labels in this Pack, but use them as a guide only. Tape counters differ slightly. You will find it worth while to unplug cassette leads and play through the cassette once, listening to the tape and noting counter numbers as you go. At the start of each recording you will hear a steady high tone. This makes sure that the tape has had time to settle into its proper running speed. There then follows a minute or so of noisy crackling. This is the program itself. Aim to start loading your program · at the start of that high tone. Once you have got the counter numbers noted, reconnect your leads and rewind the tape.

Everything connected properly?

Tape wound to the program you want?

T.V. on and TI BASIC READY?
Type in:

`OLD CS1`

OLD tells the 99 you want to load an OLD program.
CS1 tells it that you are loading from CaSsette recorder 1.
 Press ENTER and you will see:

\* **REWIND CASSETTE TAPE CS1**
 **THEN PRESS ENTER**

You have already rewound, so just press ENTER. You get
this:

\* **PRESS CASSETTE PLAY CS1**
 **THEN PRESS ENTER**

Do it. It doesn't matter how slowly you do this. The tape
recorder is being controlled by the 99, so it won't actually
start to play until you press ENTER. You should now see this

\* **READING**

You should hear, through the T.V. speaker, those noises you
listened to earlier. READING will take a minute or so for
most of the programs on the cassette, so be patient. All being
well, the next message you get will be:

\* **DATA OK**
\* **PRESS CASSETTE STOP CS1**
 **THEN PRESS ENTER**

If things are not well you will get this, after about
20 seconds:

\* **ERROR — NO DATA FOUND**
 **PRESS R TO READ**
 **PRESS C TO CHECK**
 **PRESS E TO EXIT**

There is not a lot of point in asking it to CHECK, because if it
didn't read the first time, it is unlikely to the next. Do your
own checking instead. Are the jacks in the right sockets?
Were you at the right point on the tape? If you started too far

back on the tape the 99 would have got tired of waiting for the program.

If the answer to each is yes, then try again with the volume turned up. As a rough guide to the volume you need try this. Take out the remote and earphone jacks and play part of the program to yourself. If you can stand the level of noise, then it's too quiet.

EXIT from the routine, rewind and start again. If you continue to have trouble – and it is very unlikely that you will – then check with your Texas dealer. You may need a new recorder.

Once the program has been loaded into the 99, then type RUN and sit back. At first nothing will appear to happen. In fact the 99 is very busy checking through the program and sorting itself out ready to run. This takes a few seconds, and the screen remains cyan at that time. When the program actually starts to run the screen will normally turn light green (unless another screen colour has been written into the program).

The programs on the cassette are all written in TI BASIC and are intended to be looked at. Some of them have LIST INDEXES written in. These will tell you which lines to list to see particular routines. If you wanted the lines from 1000 onwards you would type in:

LIST 1000–

and then wait with your fingers poised over FCTN and 4 (CLEAR). When the lines you want are all on screen, press and stop the listing.

## Saving programs on tape

When you have spent time working out a program and typing it in and you like it – or you don't like it but haven't got time to sort out its faults – SAVE it.

Connect up the recorder as before, put in a cassette wound to a blank spot and note the counter number. Now type in:

SAVE CS1

You will see this:

**\* REWIND CASSETTE TAPE CS1**
**THEN PRESS ENTER**

Your tape is alright, so press ENTER. You see this:

**\* PRESS CASSETTE RECORD CS1**
**THEN PRESS ENTER**

Do it. The next message is:

**\* RECORDING**

There is nothing to hear while the program is being recorded, and nothing to see except for the tape wheels turning round. The length of time taken to record a program depends directly on how long the program is. After some seconds you should see this:

**\* PRESS CASSETTE STOP CS1**
**THEN PRESS ENTER**

and then:

**\* CHECK TAPE (Y OR N)?**

Press Y (it must be LARGE CAPITAL Y). It is always worth checking that the program has been recorded properly. More instructions will appear. Follow these carefully.

**\* REWIND CASSETTE TAPE CS1**    (you did note
**THEN PRESS ENTER**                             the counter
**\* PRESS CASSETTE PLAY CS1**      number didn't
**THEN PRESS ENTER**                             you?)
**\* CHECKING**

This will take as long as the recording did. The 99 is comparing the program on tape with the one in its memory. If they are exactly the same you get this:

**\* DATA OK**
**\* PRESS CASSETTE STOP CS1**
**THEN PRESS ENTER**

If they are not the same you will get one of these messages:

```
* ERROR — NO DATA FOUND
* ERROR IN DATA DETECTED
```

If there is no data for the 99 to find then the first thing to do is to listen to that bit of the tape to hear if anything was recorded. Do you need to set recording volume levels on your recorder? Are your jack plugs in the right sockets?

If there is a program there then adjust the volume controls and the tone setting (maximum treble) and try checking again.

If your cassette recorder will load in the programs from the cassette in this pack, then it should save your own programs perfectly well. It may just take a little experimenting to get the levels right.

## Error messages

You may occasionally come across some error messages when using the recorder. They will all start like this:

```
* I/O ERROR...
```

I/O means Input and Output. The message will end with two numbers. The first of these numbers will be either 5 or 6. The second number will probably be 3 or 6.

```
I/O ERROR 5...    refers to an OLD command
I/O ERROR 6..     refers to a SAVE command
```

If you see:

```
   I/O ERROR 53
Or I/O ERROR 63
```

then check your typing. The command must be either "OLD CS1" or "SAVE CS1", typed in large capitals.

If you see

```
   I/O ERROR 56
or I/O ERROR 66
```

then either your cassette recorder is not connected properly, or the volume is too low.

# D
# Some common errors

This is by no means a complete list of the possible error reports that you might see, but it does include all those that you might meet using the BASIC commands and statements that are covered in this book. A more complete account of errors is given in Pack 2.

**\* BAD LINE NUMBER** – you have probably told the computer to GO TO a line that doesn't exist. Check the number after THEN.

**\* BAD NAME** – either you are trying to enter a line which includes a variable name of more than 15 letters, or the computer has run up to a CALL. . . line, and the routine name is mis-typed, or the name doesn't start with a letter.

**\* BAD VALUE** – check that the numbers you are using in the line are within the possible ranges for that instruction. In COLOR lines, all the numbers must be between 1 and 16, either as Set numbers, or as colour codes. In HCHAR and VCHAR the ranges are 1 to 24 (for rows) and 1 to 32 (for columns).

This error can also occur in SOUND lines, and with CHR$ and TAB. Check your typing, and check the numbers again.

**\* CAN'T DO THAT** – either the computer has found a NEXT. . . line, and there is no FOR. . . line to match it earlier,

– or you are trying to LIST, RUN or SAVE and there is no program in the memory.

– or you have got COMMANDS and STATEMENTS mixed up.

There are two sorts of instructions in TI BASIC:

COMMANDS are entered directly, without line numbers. EDIT, LIST, NEW, NUMBER, OLD, RUN and SAVE are all commands. You cannot use these in a program line.

STATEMENTS may only be used in a program, and will not work if entered directly. FOR, GO TO, IF, INPUT, NEXT and ON are the ones you have met so far.

Some instructions can be used both as commands and as statements. PRINT and all of the CALL. . . routines are examples of these.

**\* FOR. . . NEXT ERROR**   This error might be reported during the checking stage, after you have entered the RUN command, but before the program starts. The computer has noted a FOR. . . line, but cannot find a NEXT. . . to match. Either the line is missing, or you have used a different variable name at the other end of the loop.

**\* INCORRECT STATEMENT**   The most likely cause here is that you have missed out the final quotes in a PRINT line. You might also get this if you have used a BASIC word as a variable name. There are many other causes, but you are not likely to come across them at this level.

The only thing to do is to look closely at the line you tried to enter (it is still there on the screen) and retype it correctly. You cannot pull the line down for editing, as the 99 never accepted it.

The report might also occur during a program's run. Look at the line number given in the report, and list that line by typing in:

**LIST**

followed by the line number. Check the line carefully. If it is a CALL SOUND line, then perhaps you have tried to use too many sounds at once. If the line is part of a FOR. . .NEXT. . . loop, then check that the variable name is right, that you have included an = sign, and that the numbers that you are working though are correct. FOR N=1 TI 10 is a fairly common typing error, and would produce this error report.

**\* LINE TOO LONG**   Your maximum line length is always 112 characters, which take up 4 lines on the screen. You are limited here by the size of the *INPUT BUFFER* where information is processed before going into memory. It might crop up if you are trying to PRINT a very long message. Split

it up into several shorter PRINT lines instead. Short lines are much easier to edit if you need to later.

**\* MEMORY FULL** You have written an incredibly long program. The 99 has 16k of memory available. 16k means 16 kilobytes, or 16 × 1024 bytes. (= 16384 bytes). Each letter in your program takes one byte, and numbers and variables take a little more, but on average each line of the program takes up 20 to 25 bytes. So, to fill the memory you would need to have written a program of about 700 lines or more!

There are other ways in which memory can get swallowed up, but none that you will come across at the level of this book.

**\* WARNING: INPUT ERROR IN. . .TRY AGAIN:** You will see this if you try to enter a letter when a number is wanted. If the line was:

**100 INPUT N**

then the 99 will only accept a number entry.

# English
# Limited warranty — U.K.

This Texas Instruments (TI) warranty extends only to the original consumer purchaser.

1. The electronic and mechanical components of the product are warranted for a period of twelve (12) months from the date of original purchase under normal use and service against defective materials or workmanship. This warranty is void if the product has been damaged by accident, unreasonable use, neglect, improper service or other causes arising out of defects in materials or workmanship.

2. Any implied warranties arising out of the sale are limited in duration to the above twelve (12) month period.

3. ** During the above twelve (12) month period, the product will be repaired or replaced with a new or reconditioned one of equivalent quality at Texas Instruments option, without charge to the purchaser when the product is returned, with proof of purchase date to a Texas Instruments retailer. The repaired or replacement product will continue to be warranted until the end of the original twelve month period or ninety (90) days from the date of repair or replacement, whichever comes later.

4. Important notice of disclaimer regarding the software programs and book materials — read this carefully before purchasing the console and/or programs.

   T.I. does not warrant that the software programs and book materials will be free from error or will meet your specific requirements. Each user is notified that the programs may contain errors and assumes sole responsibility for any decision made or actions taken based on information obtained from using the programs. No information given concerning the utility of the programs is to be construed as an express or implied warranty.

5. ** In no event shall T.I. be liable to anyone for special, incidental, or consequential damages in connection with or arising out of the purchase or use of the console. Hardware and/or programs and the sole and exclusive liability of T.I. shall not exceed the purchase price of the console, hardware and/or programs. T.I. shall not be liable for any claim of any kind whatever against the user of the programs by any other party.

** **Paragraphs 3 and 5 shall not affect the statutory rights of the consumer as defined in the consumer transactions (restrictions of statements) order 1976, as amended.**

✂ • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • •

## TEXAS INSTRUMENTS

| | | 1 □ MR. | 2 □ Miss, Mrs | 3 □ Company |
|---|---|---|---|---|
| Model | Serial N⁰ | Herr | Frau, Fräulein | Firma |
| Modell | Serien Nr. | Monsieur | Mme, Melle | Societé |
| Modèle | N⁰ de série | Sig. re | Sig. ra, Sig. na | Ditta |
| Modello | Numero di serie | Dhr. | Mevr., Mej. | Bolag. |
| Malli | Serie nr. | Herr | Fru., frk. | Yhtiö |
| Modelo | Sarja No | Hr. | Nti, Rva | Empresa |
| | No de serie | Hra. | Senhora, Menina | |
| Last Name | N⁰ de serie | Senhor | Sra. o Srta. | |
| Familienname | | Sr. | | |
| Nom | | | | |
| Cognome | | | | |
| Achternaam | | First Name | | |
| Efternamn | | Vorname | | |
| Efternavn | | Prénom | | |
| Sukunimi | | Nome | | |
| Ultimo nôme | | Voornaam | | |
| Appellidos | | Förnamn | | |
| | | Fornavn | | |
| | | Etunimi | | |
| Address | | Primeiro nome | | |
| Adresse | | Nombre | | |
| Indrizzo | | | | |
| Adres | | | | |
| Gatuadress | | | | |
| Osoite | | | | |
| Endereco | | | | |
| Dirección | | | | |

| Date, Datum, Data, Pävämäärä, Dato, Fecha | Town | P.O. Code | Country |
|---|---|---|---|
| | Ort | Postleitzahl | Land |
| | Ville | Code Postal | Pays |
| | Città | Codice Postale | Paese |
| | Stad | Postcode | Maa |
| | By | Postnr. | Pais |
| | Kaupunki | Postinumero | |
| | Citade/Vila | Zona postal | |
| | Ciudad | D. Postal | |

# TEXAS INSTRUMENTS
## HOME COMPUTER

1103213 - 0000