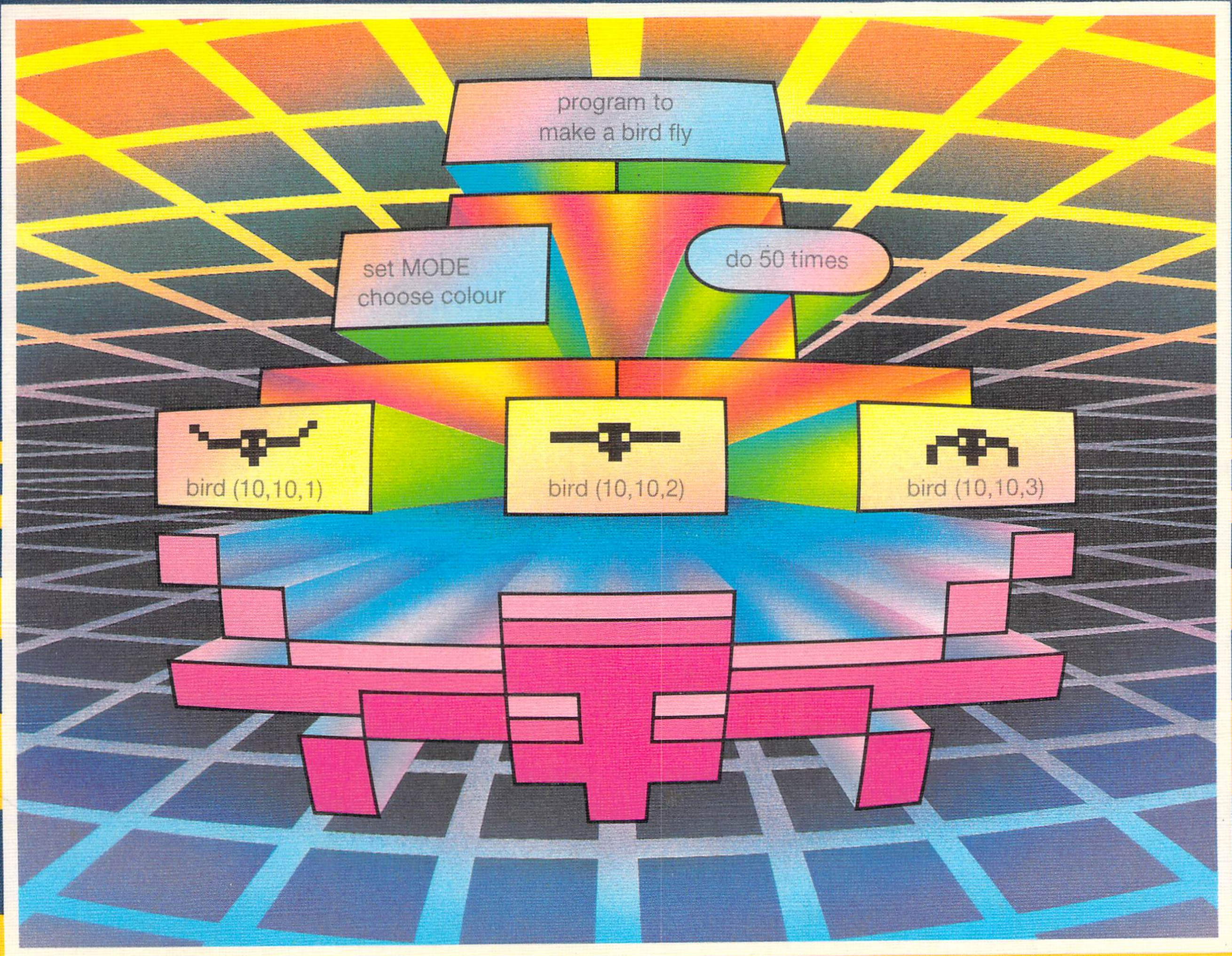
COMPUTING STUDIES IN·CONTEXT

COMAL

PROGRAMMING GUIDE



Marcus Bowman and Stephen Pople

FOR BBC MICRO

COMPUTING
S·T·U·D·I·E·S
IN·CONTEXT

COMAL

PROGRAMMING GUIDE

Marcus Bowman and Stephen Pople

H·E·B

Heinemann Educational Books

TO THE TEACHER

COMAL demon disc

To make full use of this programming guide you will need a copy of the **COMAL demon disc**. The disc will run on both stand-alone and networked BBC microcomputers.

For Standard Grade Users

If you are using this guide for the programming part of a Standard Grade Computing Studies course, the main ideas are covered in these sections:

- Foundation – Part 1
- General – Parts 1 and 2 together
- Credit – Parts 1, 2 and 3 together

Part 4 of the guide provides some suggestions for projects graded at 3 levels:

- Foundation – Level 1
- General – Level 2
- Credit – Level 3

Components of the Scheme

Computing Studies for GCSE and Standard Grade

- A full colour pupil textbook.
- Detailed case studies on: automated systems; electronic document processing; information systems; commercial data processing.
- Questions on each self-contained spread.
- Includes summary of computing principles.

COMAL Demon Disk

- The disk consists of program procedures known as DEMONS (*demon*stration software).
- A DEMON can be used 'off the shelf' to carry out simple tasks without the need for detailed programming.
- DEMONS can be joined together to make more complex programs, the ideal way to design solutions to the problems in the programming guides.

Teacher's Resource Pack

- Teaching notes for the course.
- A wide range of practical computing activities.
- Extension work related to the pupil textbook.
- Activities and extension work supplied in a ring binder as copyright-cleared A4 masters.

Heinemann Educational Books Ltd.
22 Bedford Square, London WC1B 3HH
London Edinburgh Melbourne Auckland
Singapore Kuala Lumpur New Delhi
Ibadan Nairobi Johannesburg
Portsmouth (NH) Kingston

ISBN 0 435 77055 1
© Marcus Bowman and Stephen Pople, 1988
First published in 1988

Originated by Gravure Engraving Services, Bristol
Printed and bound in Great Britain by
BAS Printers Ltd, Over Wallop, Hampshire

Acknowledgements

The Authors and Publishers would like to thank the following for permission to reproduce photographs:

1.1 Brenda Prince/Format, *Female mechanic*; 1.2 Ann Ronan Picture Library, *Player piano*; 1.3, All Sport, *Commonwealth Games*; 1.5 David Redfern Photography, *Herbie Hancock*; 1.6 Ford, *Ford car*; 1.7 J Allan Cash, *Cow and calf*; 1.8 J Allan Cash, *Geodesic building*; 1.11 Judy Harrison/Format, *Computer operator*; 2.5 Financial Times Photography, *Counting votes*; 2.7 J Allan Cash, *Golfer*; 2.9 Granada TV, *TV quiz show*; 2.10 Barnabys, *Greyhound bus*; 2.11 EHS Photography, *Cashpoint machine*; 2.12 Sally & Richard Greenhill, *School kids*; 3.2 Telefocus, *Directory enquiries*; 3.3 Sporting Pictures (UK) Ltd, *Scottish footballer*; 3.5 J Allan Cash, *Humming bird*; 3.8 Pan Am, *Airplane*; 3.9 Jenny Mathews/Format, *Young people*; 4.4 J Allan Cash, *Male mechanic*; 4.5 J Allan Cash, *Library and Swan*; 4.5 Laurie Sparham/Network, *Teenagers*; 4.5 EHS Photography, *House*.

All others taken by Chris Ridgers.

TO THE PUPIL

About this guide . . .

This guide will help you learn to solve problems using a BBC micro-computer and the COMAL programming language.

To go with the guide there is a disc of **COMAL demons**. Demons are short pieces of COMAL which you can join together to make programs. You can start using the demons to solve problems straight away. As you work through the guide you can learn how to write your own COMAL programs too.

. . . and how to use it

To help you learn to use COMAL easily, this guide is presented in two-page units. Each unit introduces a new idea, starting with a problem for you to solve. Then there are notes on planning a program to solve the problem, and details about writing, running and testing your program. Most units also have a **Help** box containing hints to help you solve the problem.

Solving problems is a good way of learning and can be fun too, so try the problems in the **For you to do** boxes in every unit. If you are keen, there are more problems for you to try, and tips on how to solve them between pages 71 and 72.

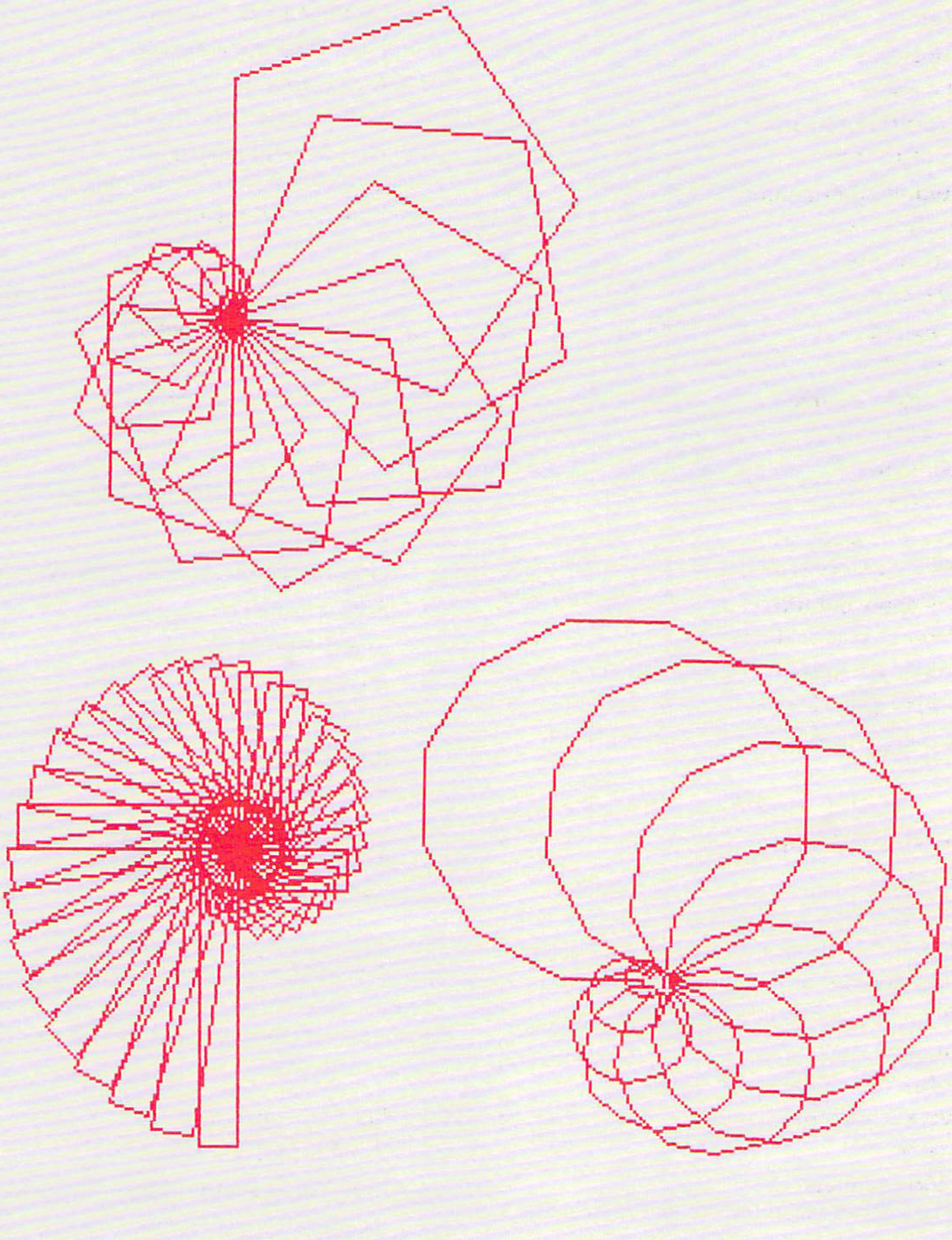
Some useful features of the guide

- **summary pages** tell you what you should be able to do if you work through the units in the guide. See pages 24, 50 and 70.
- **demon documentation pages** list the COMAL demons you can use and explain what they do. See pages 83 to 90.
- **index pages** help you to find examples of how COMAL demons and COMAL key words are used, and let you check up on important ideas in the guide. See pages 91 and 92.

This guide will help you to learn to solve problems using COMAL, but it should be fun to use too. We hope you enjoy using it.

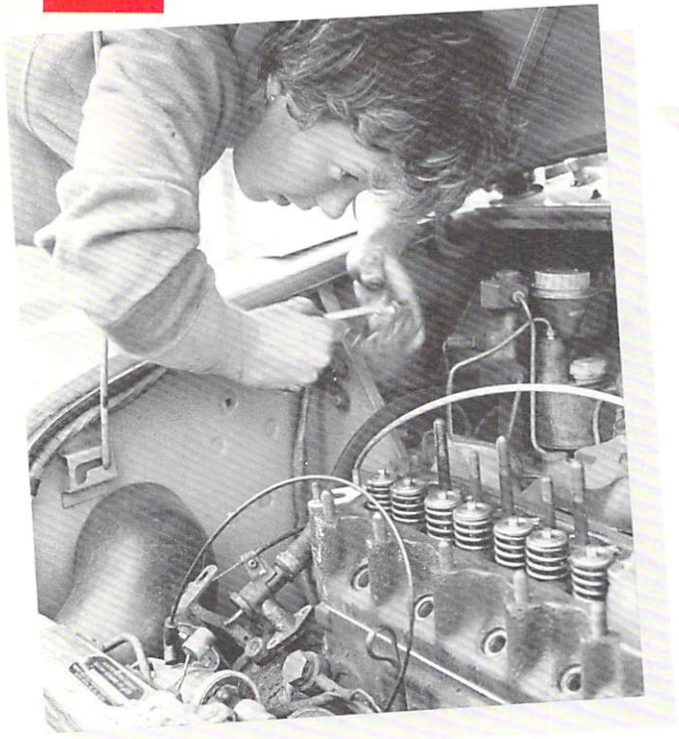
C CONTENTS

STARTING COMAL



You can draw shapes like these using some of the techniques you will learn in Part 1 of the guide.

1.1 CREATING A PROGRAM



To complete a job like this you need to follow the instructions in a car repair manual. And you have to follow them in the right order.

A computer also needs a list of instructions or **commands** before it can do a job. And it needs to be told the order in which you want them carried out.

1 Commands

In the computer language COMAL, simple command words or **keywords** are used to tell the computer what do. `PRINT` and `CLS` are two useful keywords. For example:

`PRINT "My name is Sue"` RETURN will make the computer print the message in the quotes on the screen.

`PRINT` RETURN will make the computer print a blank line.

`CLS` RETURN will make the computer clear the screen.

You have to press the RETURN key after each command. This tells the computer that you have finished typing in the command.

2 Planning a program

A program is a list of commands which the computer will carry out in order.

Before you type in a program on the keyboard, you must plan out exactly what you want the computer to do. For example, if you want this to appear on the computer screen. . . .

```
My name is Sue McKay

I live at

6 Castleton Road
Glasgow
```

you will need a program which does this. . . .

```
clear the screen
print My name is Sue McKay
print a blank line
print I live at
print a blank line
print 6 Castleton Road
print Glasgow
```

This is called a **program plan.** Don't type it on the keyboard, because the computer won't understand it.

The instructions in a program must be written using COMAL commands. Each line of commands must be given a number. Find out how to do this on the next page. But first get the computer ready . . . .

3 Preparing the computer

Before you start typing in the program, type

`NEW` RETURN

This removes any other program you might have already typed. It gets the computer ready to receive your new program.

4 Writing the program

To turn your program plan into an actual program, type this on the keyboard:

```
100 CLS                          RETURN
110 PRINT "My name is"           RETURN
120 PRINT                        RETURN
130 PRINT "Sue McKay"            RETURN
140 PRINT                        RETURN
150 PRINT "I live at"            RETURN
160 PRINT                        RETURN
170 PRINT "6 Castleton Road"     RETURN
180 PRINT "Glasgow"              RETURN
190 END                          RETURN
```

Don't forget to press RETURN at the end of each line. You've been reminded of this above, but most program listings don't show it.

There is a **line number** at the beginning of each command. This tells the computer that the command is part of a program. Without the number, the computer would carry out the command as soon as you pressed RETURN

The line numbers show the order in which the computer should carry out the commands. The computer starts at the lowest number and works through to the highest.

You could number the lines 1, 2, 3, 4 and so on. But it's better to start at 100 and go up 10 each time. This gives you the chance to add extra lines to the program later.

The keyword `END`, at line `190`, tells the computer that the program is complete.

5 Listing the program

You can check your typing by asking the computer to list the lines in the program. Do this by typing `LIST` RETURN

Don't use a line number for `LIST`, because it isn't part of the program.

When you list your program, any extra lines you added will be slotted in so that they appear in the correct order.

6 Corrections and changes

If you make a mistake, just type the line again. If you use the same line number, the computer will automatically replace the old line by the new line.

If you want to remove or **delete** a line, just type the line number and press RETURN

To add an extra line, choose a suitable line number and type in the command. For example, if you type:

```
175 PRINT "Bearsden" RETURN
```

the extra line will be put in the program between lines `170` and `180`.

Check any changes you make, by listing the program.

7 Running the program

To tell the computer to carry out the commands in the program, type:

```
RUN RETURN
```

The computer will start at the lowest line number and work through to the highest.

8 Testing the program

It is important to check that your program has done its job properly. Look at what the computer printed. Make sure that the words are printed correctly, and that the blank lines have been printed too. If anything is wrong, list the program and check each line carefully.

For you to do

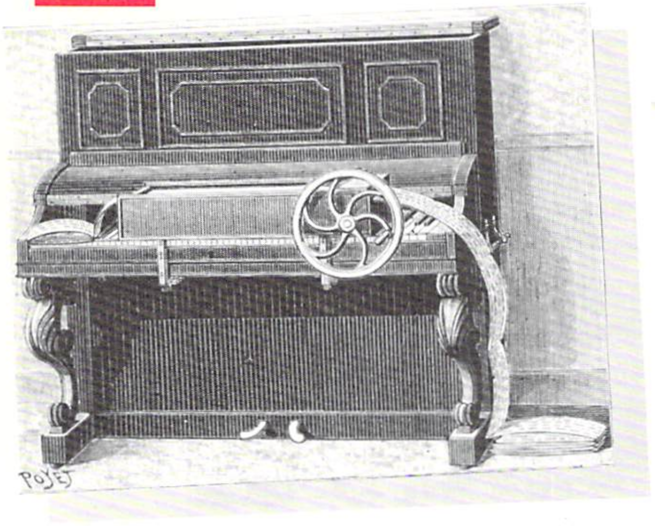
Plan and write programs to print:

1 Your name, age and address – with a blank line between each.

2 A list of football teams or pop groups.

3 The titles of five TV programmes – with two blank lines between each.

1.2 TUNING UP



This old piano can play a tune by itself. Holes on the cards tell it what to play. You can write a program to make your computer play a tune. To make the job easier, we've already written some bits of program for you.

1 Demons

One way of writing a program is to break it down into sections so that each section has a different job to do.

The sections of program we've already written are called **demons**. They each **demons**trate how to do a different job. And they each have a name. For example:

`set_up_notes` is a demon which tells the computer how to play different notes.

`tune` is a demon which makes the computer play a tune.

2 Bringing the demons into the computer

Before you can use the demons, you have to load them from your demons disc. To do this:

- Put the disc in the disc drive.
- Hold down SHIFT, press and release the BREAK key, then let go of SHIFT
  A **menu** should appear on the screen.
- Select the `Sounds` demons from the menu by typing their menu number and RETURN
  The computer will print a message when this set of demons has been loaded.

3 Procedures

Any section of program which has a name and does a particular job is called a **procedure**. The demons `set_up_notes` and `tune` are both procedures.

You can tell a procedure to do its job by typing its name:

If you type `set_up_notes` RETURN

the computer will get ready to play notes. (Note that the lines between the words in `set_up_ notes` are underscore characters, and not hyphens.)

If you then type `tune` RETURN

the computer will play a tune.

4 Playing a tune – planning the program

First, write down exactly what you want the computer to do:

set up the notes

play a tune

The next step is to change this plan into the program itself.

5 Writing the program

If you haven't already done so, load the set of `Sounds` demons from disc. You will load several demons. But in this program, only two of them are going to be used.

Type in the main program, using line numbers from `100` onwards:

```
100 set_up_notes
110 tune
999 END
```

The `END` statement tells the computer that the whole job is finished. It's usual to give the `END` statement the line number `999`.

6 Listing the program

When you have loaded the demons from disc, you have part of a program. It looks like this:

7000 to 20999	`set_up_notes`, `tune` and other ready-made demons
21000 to 29999	list of demons loaded from disc

When you have finished typing in the rest of the program, it looks like this:

100 110 999	main program which you typed in
7000 to 20999	`set_up_notes`, `tune` and other ready-made demons
21000 to 29999	list of demons loaded from disc

If you use `LIST` RETURN to list the program, the *whole* program will appear – including all the instructions in the demons. This makes the program very difficult to read and check.

It's much more convenient to list one part of the program at a time. You can do this using the orange function keys. When you load the demons from the disc, the keys are automatically set up to do these jobs:

f0 lists the main program (that's the lines with numbers up to `999`).

f2 lists any demons you typed yourself (none, this time).

f4 lists the names of all the demons loaded from disc.

f8 takes you back to the menu, if the demons disc is in the drive.

(Be careful! When you press this key, you lose your program).

7 Running the program

To run the program, type `RUN` RETURN

When the program runs, the computer starts at the lowest line number. Then it carries out its instructions line by line.

When the computer comes across the name of a procedure:
it searches the program for that procedure,
it carries out those instructions
then it goes back to where it left off in the main program.

In your program, the computer first carries out the instructions in the `set_up_notes` procedure. Then it carries out the instructions in the `tune` procedure. When it reaches line `999`, it stops.

8 Testing the program

This program is easy to check. If it works properly, you should hear a tune.

For you to do

Plan and write programs to solve the problems below. Remember that if your program uses any sound demons, `set_up_notes` must be used first.

1 Make the computer play the tune twice.

2 Make the computer:

print **Listen to this tune**

play a tune

3 One of the sound demons is called `beep`
It makes the computer 'beep' just like it does when you switch on.
Make the computer:

beep

print **Listen to this tune**

play a tune

print **End of tune**

beep

1.3 EVERYTHING IN ORDER

Many of the world's countries take part in the Commonwealth Games. But in which order should the contestants parade around the stadium before the games begin? Usually, it's decided by putting the names of the countries in alphabetical order. It's a job which can be done by the computer.

1 Putting names in order – planning the program

Here is a plan for a program to sort a list of names:

get ready
ask for the names
put the names in order
print the names

These four parts of the program can be done by four of the demons on your demons disc:

`set_up` gets the computer ready to handle lists of words, numbers, or both.

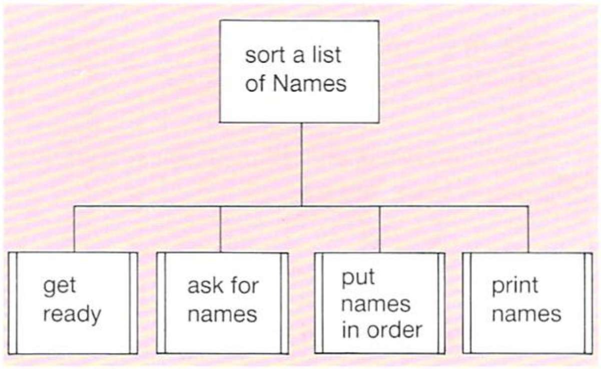
`ask_for_words` asks you to type in your list of words. Each time it asks you for a word, you type in the name of a country and press RETURN Names like **New Zealand** count as just one word.

`order_words` sorts your list of words into alphabetical order.

`output_words` makes the computer print the list of words on the screen.

2 Drawing a picture of the plan

To help your planning, it is a good idea to draw a diagram showing the different sections of the program. A diagram like this is called a **structure diagram**:



The box at the top describes what the whole program does.

The boxes underneath are arranged in order from left to right. Each shows a different section of the program.

Boxes with double lines on each side show jobs which can be done by ready-made demons.

HELP

Turn back to 1.1 'Tuning up' if you want full details on how to load demons from disc.

Remember to load all the demons you want *before* typing in your program.

SHIFT BREAK loads the demons menu. Select the demons you want by choosing a number on the menu. You want `Words and Numbers – Level 1` demons this time, so type their menu number and press RETURN

f0 lists the main program (the lines which you typed in).

f4 lists the names of all the ready-made demons you loaded. There isn't space on the screen for the full list. Press SHIFT to see more.

3 Writing the program

First load the `Words and Numbers - level 1` demons from your disc. The **HELP** box at the bottom of the page shows you how.

Then type in the main program:

```
100 set_up
110 ask_for_words
120 order_words
130 output_words
140 END
```

When you type in the names of the demons, remember that the words are joined by underscore characters and not hyphens.

4 Running the program

When the program runs, `set_up` puts a menu on the screen. It will ask you whether you want to deal with numbers, words, or words and numbers. You want words this time, so select number `2` on the menu. Do this by typing `2` RETURN

Next, the computer asks you how many words you want to deal with. This is so that it can reserve space in its memory for all the names in the list. To enter 5 names, type `5` RETURN

`ask_for_words` makes the computer ask you for your words one at a time. Each time `Word:` appears on the screen, type a name and press RETURN. If you reserved space for 5 words, this will happen 5 times.

`order_words` puts the words in alphabetical order.

`output_words` makes the computer print the words in the list.

5 Testing the program

It is important to check that the computer is doing the job properly.

Before you run the program, write down the 5 names which you are going to type into the computer. Be sure that they aren't in alphabetical order. That would make it too easy!

Work out the alphabetical order yourself. Then run the program – typing in the names in the same order as you first wrote them down.

When the program ends, check that the computer put the names in the same order as you did.

- Use line number `100` for the first line of your program.

  Use line number `999` for the `END` statement.
- To *change* a line of your program: type the line number and the new line. Then press RETURN. The new line will automatically replace the old one.
- To *delete* a line from your program: type the line number and press RETURN
- To *run* the program: type `RUN` RETURN
- To *stop* the program before it ends: press ESCAPE
- To clear the screen: type `CLS` RETURN

For you to do

Plan and write programs which will:

1 ask for the names of 6 TV programmes
put them in order
print the list

2 ask for the names of 8 models of cars
put them in order
print the names

3 sort these names, then print the list:

A. Smith	**F. Smithers**
K. Smith	**A. T. Smith**
A. Smythe	**A. Smithers**

Is there a better way of writing these names so that the computer can sort them?

4 print the names of ten pop groups in alphabetical order.

1.4 TELLING OTHER PEOPLE

When you watch TV, you can find out more about the programme by reading the notes in the Radio or TV Times. Notes like this are called **documentation**.

Documentation is also useful for computer programs. It can be written on paper. Or it can be written in the program itself.

1 Documenting your program

You can use **comment statements** to write notes inside your program. Comment statements don't show up when the program runs, but you can see them when you list the program.

In COMAL, comment statements begin with two slash characters, //. For example, you could start a program with comment statements like this:

```
10 // Sort a list of numbers
20 // by M. Ferguson
30 // 21st September 1988
```

2 Documenting demons

At the end of the book, you'll find documentation on all the demons used in the book. Read it to find out what the following demons do. They are going to be used in the next program:

```
set_up
ask_for_numbers
order_low_to_high
output_numbers
```

3 Titles and instructions

When you write a program, it is a good idea to make some documentation appear on the screen:

`Sort a list of numbers`	the program title
`—by M. Ferguson—`	the writer's name
`21 September 1988`	the date

`Instructions`
`You will be asked how`
`many numbers you wish`
`to sort. Then you will`
`be asked to type them in.`
	some instructions telling the user what the program does and how to use it

There are two demons for doing this. They are called `title` and `instructions`. They are in the `Words and Numbers - level 1` demons. Load this set of demons from your disc now.

Before you can use `title` and `instructions`, you have to change them so that they give your own title and instructions instead of the ones already there.

`title` is a demon which prints a program title, the writer's name, and a date on the screen. It uses line numbers from `11800` to `11900`. List these lines by typing:

```
LIST 11800,11900
```

Then change lines `11830`, `11850` and `11860` so that they give *your* title, *your* name and *today's* date.

`instructions` is a demon which prints 5 lines of instructions on the screen. It uses line numbers from `12000` to `12080`.
List these lines by typing:

```
LIST 12000, 12080
```

Then change lines `12010` to `12060` so that *your* instructions are put between the quotes.

You are going to write a program that puts some numbers in order. The program will ask for the numbers to be typed in one after another. The instructions should explain what is going to happen.

4 Planning the program

Here's a plan for a program to sort a list of numbers:

```
print a title
print instructions
get ready to deal with a list of numbers
ask for the numbers
sort the list of numbers
print the list of numbers
```

5 Writing the program

Be sure you have loaded the `Words and Numbers – Level 1` demons. Then type in your program:

```
 10 // Sort a list of numbers
 20 // by M. Ferguson
 30 // 21st September 1988
100 CLS
110 title
120 instructions
130 set_up
140 ask_for_numbers
150 order_low_to_high
160 output_numbers
170 END
```

The program starts with a few comment statements. The main program begins at line `100`. It uses 6 demons.

Remember to change the demons `title` and `instructions` so that your own title and instructions are used.

HELP

SHIFT BREAK loads the demons menu from disc. Select a set of demons by typing a number from the menu and pressing RETURN

f0 lists the main program.

f4 lists the names of the demons loaded. There isn't room on the screen for the full list. Press SHIFT to see more.

■ To *change* a line of your program: type the line number and the new line. Then press RETURN

6 Running the program

When the program runs, the computer will start by printing a title and some instructions on the screen.

Next, it will ask you whether you want to deal with numbers, words, or words and numbers. You want numbers this time. Select this option by typing 1 RETURN

The computer will then ask you how many numbers you want to deal with. When you have typed in your reply, it will ask you for your numbers one at a time.

Finally, the computer will sort these numbers and print them on the screen in order – from the lowest to the highest.

7 Testing the program

Sorting numbers is rather like sorting words. You can use the same kind of test to check the program.

Write 5 numbers on paper. Put them in order (smallest to largest). Then use the computer to sort the same 5 numbers. The computer's order should be the same as yours.

For you to do

1 Draw a structure diagram for the program on this page.

Plan and write programs to solve the following problems. Be sure each program prints a title and some instructions.

2 Print these numbers in order:

120	21	10	121	12	22
221	211	101	112	11	212

3 Print these numbers in order:

142 14.2 0.142 1.42 0.0142 1420

4 Use the computer to sort and print the heights of 12 people you know.

5 Use the computer to sort and print the telephone numbers of 8 people you know.

1.5 HIT SOUNDS

This electronic keyboard is programmed to sound notes. But it won't work until you tell it which notes you want. To do this, you have to press the keys. The further up the keyboard you press, the higher the **pitch** of the note.

Using two demons, you can make the computer sound a note:

`set_up_notes` is a demon which gets the computer ready to sound notes.

`note` is a demon which makes the computer sound a note.

1 Parameters

`note` is a procedure which isn't quite complete. It can't do its job until you tell it which note you want. You do this by putting a letter in brackets after the demon name. Like this:

```
note(c)
```

The letter in brackets controls the pitch of the note. The letter could be `c`, `d`, `e`, `f`, `g`, `a` or `b`.

`c` gives the lowest note, `b` gives the highest note.

The letter in brackets is called a **parameter**. If you make the letter `c`, then `c` is the **value** of the parameter. When you put the letter in brackets, you are **passing a parameter** to the procedure.

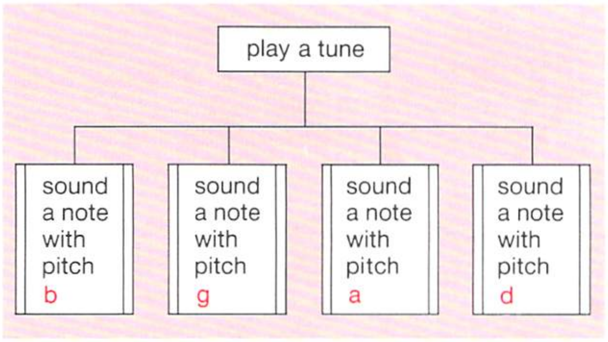
With parameters, you can use the same procedure over and over again to produce lots of different effects.

2 Making a tune – planning the program

To play a tune, the computer needs to sound several different notes one after another. Here is a plan for a program which plays a tune with 3 notes:

```
print a title
sound a note with pitch b
sound a note with pitch g
sound a note with pitch a
```

Here's the structure diagram:



play a tune
sound a note with pitch b
sound a note with pitch g
sound a note with pitch a
sound a note with pitch d

In the structure diagram:

You can only start a box when the box on its left has been completed.

A box isn't completed until its lower branches have all been completed.

HELP

SHIFT BREAK loads the demons menu from disc. Select the `Sounds` demons by typing their menu number and pressing RETURN

f0 lists the main program.

f4 lists the names of the demons you can use in your program.

- To *delete* a line from your program: type the line number and press RETURN
- To *change* a line: type the line number and the new line. Then press RETURN

3 Writing the program

First, load the `Sounds` demons from your disc. Then type in your program:

```
 10 // Three notes
 20 // by K. Patel
 30 // 26 Oct 88
100 title
110 set_up_notes
110 note(b)
120 note(g)
130 note(a)
999 END
```

The values `b`, `g` and `a` set the pitches of the notes. Use different values if you prefer. Some pitches produce sweeter notes than others, but the demon `note` will sound a note whatever pitch you choose. It's your choice!

If you want your own title to appear in the program, you will need to change some of the lines in the demon `title`. List the lines in the demon by typing:

```
LIST 11800,11900
```

Then change lines `11830`, `11850` and `11860` so that they give *your* title, *your* name and *today's* date.

`note(c)` plays a single note, a musical C. For a different note, put `d`, `e`, `f`, `g`, `a` or `b` in the brackets instead of `c`.

`chord(c)` is rather like `note(c)` except that it mixes two extra notes with the note C to sound a chord.

You don't have to write a program to try out these demons. For example:

To make the computer sound the note C:

type: `set_up_notes` RETURN

then type: `note(c)` RETURN

4 Running the program

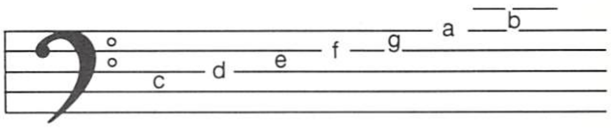
When the program runs, it uses the same procedure, `note`, three times. A different parameter value is passed to the procedure each time. The procedure makes the computer sound a different note each time.

5 Testing the program

It is important to check that the program does what it is supposed to do. Listen carefully, and make sure you can hear 3 notes.

It is more difficult to tell if the pitch is correct, but you can check to see if the pitch is going up or down in the right order. A parameter value of `c` gives the lowest note. The pitch gets higher if `c` is changed to `d`, `e`, `f`, `g`, `a` or `b`.

Musicians may find this useful:



For you to do

Plan and write programs to solve these problems:

1 Play a tune using these values for the pitches of the notes:

`b g a d d a b g`

2 Play a scale, using these values for the pitches of the notes:

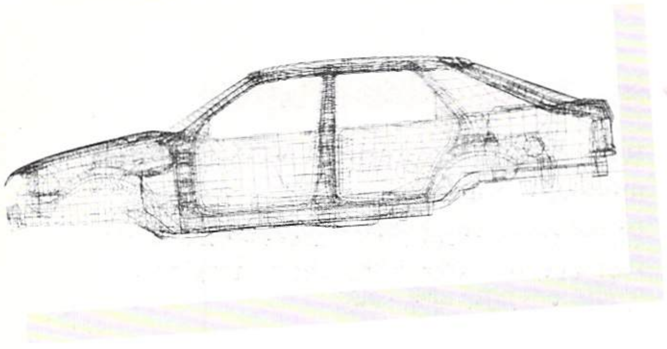
`c d e f g a b`

3 `chord` is a demon which plays a chord (look at the HELP? box for more details). Play a tune by using these values for the pitches of the chords:

`f f g e f g`

4 Write your own tune, using the demons `note` or `chord`. Use both demons in the same program, if you want!

1.6 DRAWING



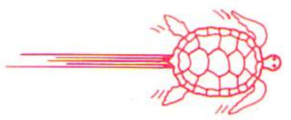
The Ford Granada – as drawn by computer. The car was designed using **computer graphics** like this. The picture looks complicated. But it's based on simple shapes which you can draw using your computer.

1 Turtlegraphics demons

There are several demons to help you draw shapes on the screen. These are `Turtlegraphics` demons. One demon puts a small turtle in the middle of the screen. Other demons tell it where to crawl. As the turtle moves about, it leaves a trail behind it.

You'll find a full description of the `Turtlegraphics` demons at the back of the book. Here is a list of their names:

```
set_up_graphics          pen_down
forward(distance)        pen_up
back(distance)           home
left(angle)              hide_turtle
right(angle)             show_turtle
end_graphics
```



The demons with brackets after their names use parameters. They won't work until the parameter has been given a value. You do this by putting a number in the bracket instead of the word. For example:

```
right(90)
```

will make the turtle turn right 90°

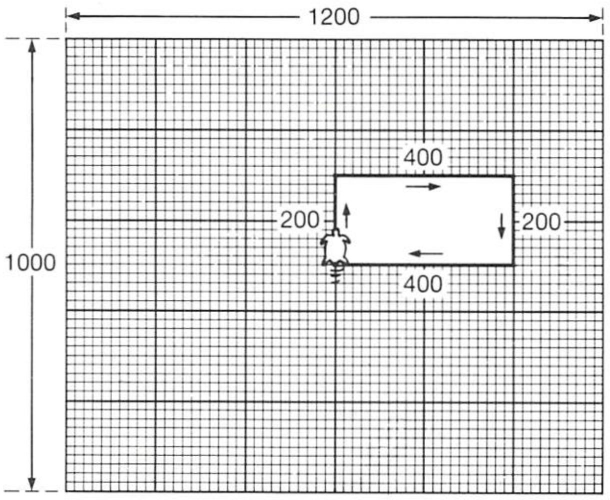
Now try using turtlegraphics to draw a rectangle.

2 Drawing a rectangle – planning the program

Imagine that the screen is a large piece of graph paper, 1200 squares across and 1000 squares from top to bottom. In the middle of the screen is a small turtle pointing upwards. To draw a rectangle, you could make the turtle follow a set of instructions like this:

move forward 200 squares
turn right 90 degrees
move forward 400 squares
turn right 90 degrees
move forward 200 squares
turn right 90 degrees
move forward 400 squares
turn right 90 degrees

This is the trail the turtle would leave:



You can write a program to make the turtle draw this rectangle on the screen. Use the program plan as a guide.

HELP

SHIFT BREAK loads the demons menu from disc. Select the `Turtlegraphics` demons by typing their menu number and pressing RETURN

3 Writing the program

First, load the `Turtlegraphics` demons from your disc. Start with a few comment statements. Then type in the main program:

```
100 set_up_graphics
110 forward(200)
120 right(90)
130 forward(400)
140 right(90)
150 forward(200)
160 right(90)
170 forward(400)
180 right(90)
999 END
```

The demon `set_up_graphics` gets the computer ready to draw, and puts the turtle on the screen.

The demon `forward` is used 4 times. Each time, the parameter value (in brackets) is changed.

The demon `right` is also used 4 times. But the parameter value, `90`, doesn't change. This is because the turtle has to turn 90° to the right after drawing each line.

4 Running the program

When the program runs, a small arrow (that's the 'turtle') will move quickly over the screen. It should draw a rectangle.

When the program has finished, you may want to get rid of the drawing and return the screen to normal. To do this:

type `end_graphics` and press **RETURN**

`end_graphics` undoes the effect of `set_up_graphics`. It doesn't need a line number because it isn't part of the program.

- To *clear the graphics* from the screen: type `end_graphics` and press **RETURN**

5 Testing the program

If you are quick, you can follow the turtle as it draws the shape on the screen.

Check that it carries out the commands in the correct order. Check that the rectangle looks twice as long as it is high.

One way of checking the program is to do a **dry run**. Follow the instructions in the program, line by line, using paper and pencil:

Take a piece of graph paper.
Place your pencil in the middle.
Move the pencil, following each instruction.
Check that you have drawn a rectangle.

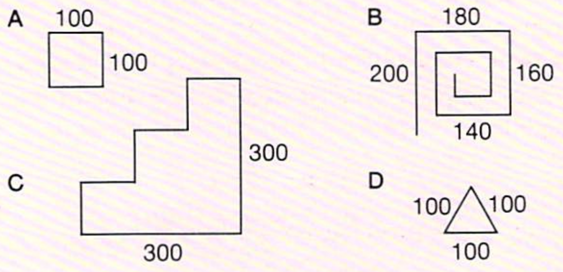
For you to do

1 Draw a structure diagram for the program described on this page.

2 What shape do you get when you follow these instructions on graph paper?

move forward 200
turn left 90°
move forward 100°
turn left 90°
move forward 100
turn right 90°
move forward 200
turn left 90°
move forward 100
turn left 90°
move forward 300

3 Plan and write programs to draw each of these shapes.



1.7 CHANGE THE SIZE

This little calf will grow in size.

You can change the size of your turtlegraphics pictures as well. It can be done by changing just one line in a program.

Start with a simple shape like the letter L.

1 Using a variable

Here are two sets of instructions for drawing the letter L:

```
left(90)              left(90)
forward(100)          forward(length)
right(90)             right(90)
forward(100)          forward(length)
back(100)             back(length)
left(90)              left(90)
back(100)             back(length)
right(90)             right(90)
```

The first set of instructions draws the letter L with legs 100 long. But if you want a different size of letter, you have to change several lines in the program.

The second set of instructions doesn't have a value for the length of the sides. Before it will work, you have to tell the computer what `length` stands for, like this:

```
length := 100
```

Whenever the computer meets `length` it uses the number `100` instead. If you want a different size letter, just make `length` stand for a different number.

In this example `length` is a **variable**. When you put `length := 100`, you are **assigning a value** to the variable.

2 Variable names

In the last example, the variable name was `length`.

Variable names can be almost any word, or any combination of letters and numbers. But: *Always* start the name with a letter. *Don't* use COMAL keywords as names, in case you confuse the computer!

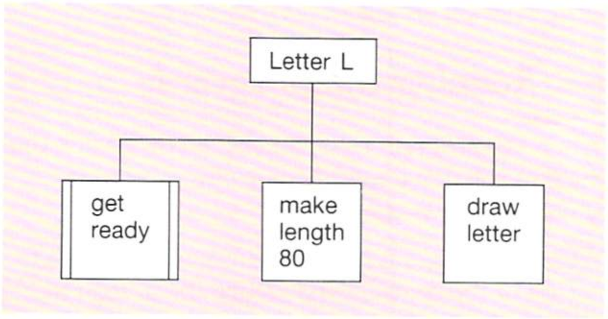
The only punctuation symbol you can use in a variable name is the underscore sign _.

3 Drawing the letter L – planning the program

To draw the letter L with legs 80 units long, you could use a program plan like this:

```
get ready to use turtlegraphics
make length 80
draw the letter
```

Here's its structure diagram:



HELP

SHIFT BREAK loads the demons menu. Select the `Turtlegraphics` demons by typing their menu number and pressing RETURN

f0 lists your program.

f2 lists any procedures you have written yourself.

f4 lists the names of the demons you can use in your program.

4 Writing the program

Load the Turtlegraphics demons.

Use set_up_graphics to get the screen ready for drawing.

State the length of the letter's legs by giving the variable length a value of 80. After that, the computer will use 80 whenever it meets length.

```
 10 // Letter L
 20 // by A. Name
 30 // 1st October 1987
100 set_up_graphics
110 length := 80
120 left(90)
130 forward(length)
140 right(90)
150 forward(length)
160 back(length)
170 left(90)
180 back(length)
190 right(90)
999 END
```

5 Running the program

When the program runs, the computer should draw the letter L with legs 80 long. Now try drawing the same letter with longer legs, by changing line 110:

```
110 length := 160
```

Changing the value of length in this line is all that is needed to produce a different size of letter.

- Use lines 10 to 999 for your procedures.
- To delete a line from your program: type the line number and press RETURN
- To change a line: type the line number and the new statement. Then press RETURN
- To clear the graphics, so that you can continue work on your program: type end_graphics and press RETURN

6 Testing the program

Did the computer draw the correct shape? When you changed line 110, did the computer draw the letter with legs twice as long?

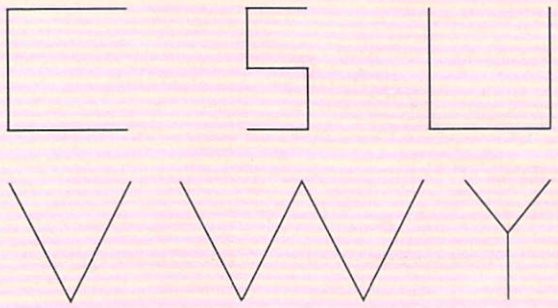
Testing the program with every possible value of length is called **exhaustive** testing. It would take too long to test the program by making it draw every possible size of letter L, so just try a few carefully chosen lengths. For example, large, middle-sized and small. If the computer draws these properly, it will probably manage any other sizes as well.

Testing the program with a few carefully chosen values is called **selective** testing.

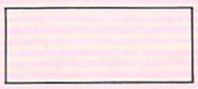
For you to do

Plan and write programs to solve these problems:

1 Draw the following letters. In each letter, make all the lines the same length. Write each program in such a way that it will draw a different size of letter if you change just one line of the program.



2 Draw a rectangle. The length of the rectangle should be a variable, and so should the height.



Run your program three times, drawing a different size of rectangle each time. Then make your program draw a square.

1.8 MORE SHAPES

This building looks complicated, but it is actually made up of many simple shapes.

With **turtlegraphics**, you can draw a complicated shape like a house by drawing a few simple shapes one after another.

There are several ready-made turtlegraphics procedures on your disc. But none of them draw whole shapes like squares and triangles. If you want procedures for these, you have to write them yourself.

1 Writing your own procedures

To draw a square, you plan and write a procedure like this:

First, choose a name – let's call it `square`. Start your procedure with `PROC` and the name of the procedure.

Next, write the commands to be used in the procedure.

Finally, end the procedure with `END PROC` and the name.

```
1000 PROC square
1010   right(90)
1020   forward(100)
1030   right(90)
1040   forward(100)
1050   right(90)
1060   forward(100)
1070   right(90)
1080   forward(100)
1090 END PROC square
```

2 Procedure names

These can be any combination of letters and numbers. But don't begin the name with a number. And don't use COMAL keywords like `PRINT` or `COUNT`.

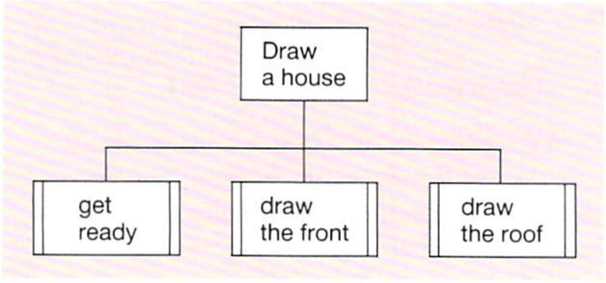
You can use several words in the name, but they must be linked by the underscore sign like this: _ . The sign is below the £ on the keyboard. Don't confuse it with the hyphen.

3 Drawing a house – planning the program

First, break the job down into stages:

get ready to draw
draw the front
draw the roof

The structure diagram for the program plan looks like this:



4 Writing the program

First, load the `Turtlegraphics` demons from disc. Then type in the program:

```
 10 // House
 20 // by A.Name
 30 // Spring 1988
100 set_up_graphics
110 front
120 roof
999 END
```

You must add the instructions for `front` and `roof` yourself using line numbers between `1000` and `6999` for these procedures. That way, you won't interfere with any ready-made demons.

5 Running the program

First the program tells the computer to find and carry out the instructions for `set_up_graphics`. This puts the turtle on the screen and gets it ready for drawing.

Next, the program makes the computer find and carry out the instructions for `front`.
Then it does the same for `roof`.

6 Testing the program

If the program runs correctly first time – good!

If there are any mistakes, you will have to test your program bit-by-bit. Removing the mistakes (or **bugs**) from your program is called **debugging**.

Once you have run your program, COMAL remembers the names of your procedures, and where to find them. To check a procedure: type its name and press RETURN
You don't need a line number because you aren't writing a program. You are using **immediate mode** to make the computer carry out the procedure straight away.

Check `front` first. Make sure that the turtle draws the shape properly. It should end up in the right place to start drawing the roof. If it doesn't, you will have to change some of the instructions in the procedure.

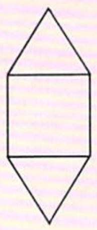
When `front` works, check `roof` in the same way.

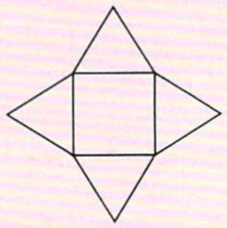
Now run the program again to see if it does the whole job properly.

For you to do

1 Complete the example program which draws a house. You must add procedures to draw the front and the roof.

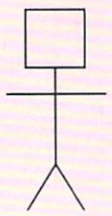
2 Plan and write separate programs to draw these shapes:

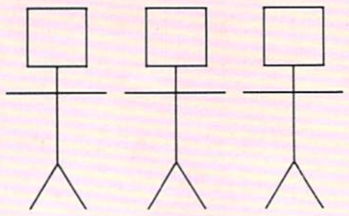




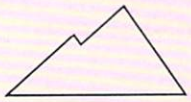
3 Plan and write a procedure to draw:

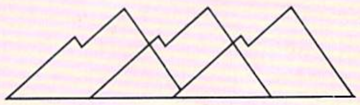
a a person **b** three people in a row





4 Plan and write a procedure to draw:
a a mountain **b** a range of mountains





HELP

SHIFT BREAK loads the demons menu.

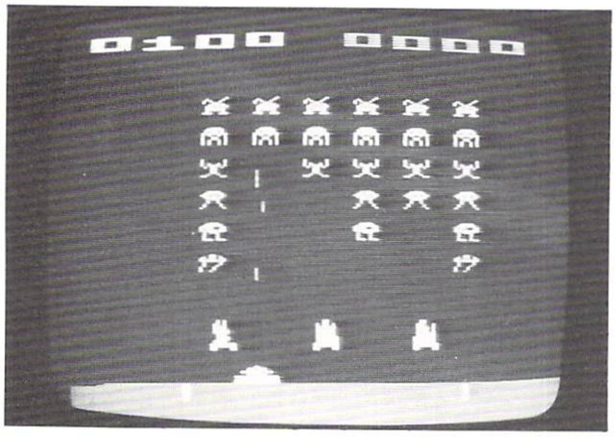
f0 lists your program.

f2 lists any procedures you have written yourself.

f4 lists the names of the demons you can use in your program.

- Use lines `1000` to `6999` for your own procedures
- To test a procedure:
  run the whole program first, then type the procedure name and press RETURN
- To clear the graphics:
  type `end_graphics` and press RETURN

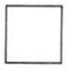
1.9 ROUND AND ROUND

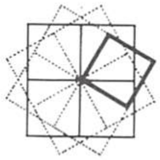


This looks like a whole invasion! In fact, it's just one set of shapes repeated on the screen in lots of different positions.

1 Repeating a set of instructions

Making a computer repeat a set of instructions can be very useful. For example, you could turn

this square . . . 

into this pattern . . . 

To draw the pattern, the turtle has to draw a square, turn a little, draw another square, turn a little – and so on until it has drawn 12 squares. You could make it do this by typing out the same set of instructions 12 times. But there would be a lot of typing to do!

Fortunately, there is an easier way: type the instructions just once, then tell the computer to repeat the same instructions, 12 times in all.

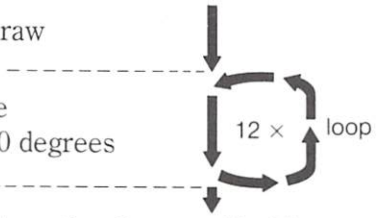
Any section of program which is repeated over and over again in this way is called a **loop**.

2 Drawing the pattern – planning the program

Twelve squares have to be drawn. Each time the turtle finishes one square, it has to turn 30° before starting the next one.

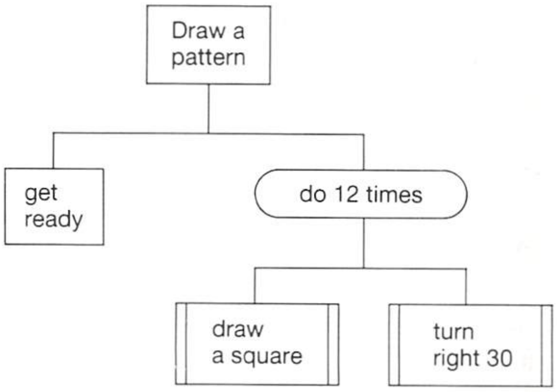
The plan for the program looks like this:

```
get ready to draw
do 12 times
   draw square
   turn right 30 degrees
end loop
```



The loops starts where the plan says 'do 12 times'. The loop contains the instructions to be repeated. The computer must count the number of times it goes round the loop. It must stop going round the loop after 12 times.

Here's the structure diagram of the program:



In the structure diagram:
The start of the loop is written in a box with rounded ends. The box isn't completed until its lower branches have been completed.

HELP

SHIFT BREAK loads the demons menu.

f0 lists your program.

f2 lists any procedures you have written yourself.

- To make the computer wait until you press the spacebar at line `115`, type in:

```
115 wait
```

- To delete a line from your program: type the line number and press RETURN

3 Writing the program

First, load the Turtlegraphics demons. Then type in the main program:

```
100 set_up_graphics
110 FOR turns := 1 TO 12 DO
120   square
130   right(30)
140 NEXT turns
999 END
```

The loop used in this program is a FOR...NEXT loop. It begins with a FOR statement and ends with a NEXT statement.

turns is called the **control variable** because it controls the number of times the computer goes round the loop. The control variable doesn't have to be called turns. Any variable name will do.

The control variable is given a starting value (1 in this case) and a maximum value (12 in this case).

The instructions for drawing the square and turning the turtle are written inside the loop, so that they will be repeated.

When you have typed in the main program, you must type in your own procedure for drawing a square. Use line numbers between 1000 and 6999 for this.

Finally, remember to add a few comment statements at the beginning of the program.

4 Running the program

When the computer first sees line 110 it gives the variable turns the value 1.

The computer then carries out the instructions in lines 120 and 130. When it meets NEXT turns the computer gives turns the value 2 and goes back to line 110.

The computer goes round and round the loop. Each time, it carries out the instructions in lines 120 and 130, and adds 1 to the value of the variable turns.

When turns is more than 12 the computer skips straight to the instruction after the end of the loop.

5 Testing the program

To check how the computer draws the pattern, make it wait until you press the spacebar, between lines 110 and 120. You can do this using the ready-made procedure wait, like this:

```
115 wait
```

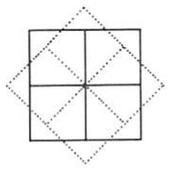
Now when you run the program, you can count the number of times the turtle draws the square then turns. This should be 12.

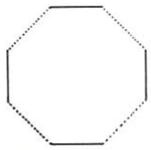
You can remove line 115 later, when you have finished testing the program.

For you to do

Plan and write programs to solve these problems:

1 Use turtlegraphics to draw these shapes:



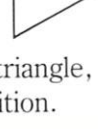


2 On page 16, there is a procedure for drawing a square. In it, the instructions for moving and turning the turtle are repeated several times. Rewrite the procedure in a shorter form, using a FOR...NEXT loop.

3 Here is a plan for producing a shape:

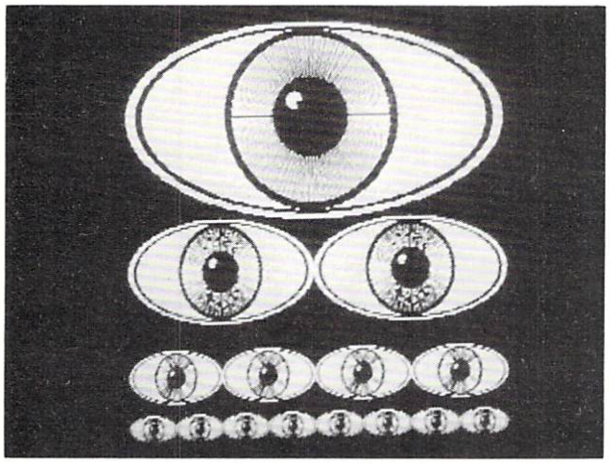
```
get ready to use turtlegraphics
do 36 times
   draw a triangle
   turn right 10 degrees
end loop
```

Use turtlegraphics to draw the shape. You can use a triangle like this:



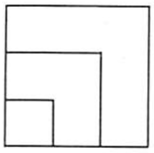
Each time the turtle draws the triangle, it should return to its starting position.

1.10 THE SAME BUT DIFFERENT



To produce this picture, the computer drew the same shape in several different sizes.

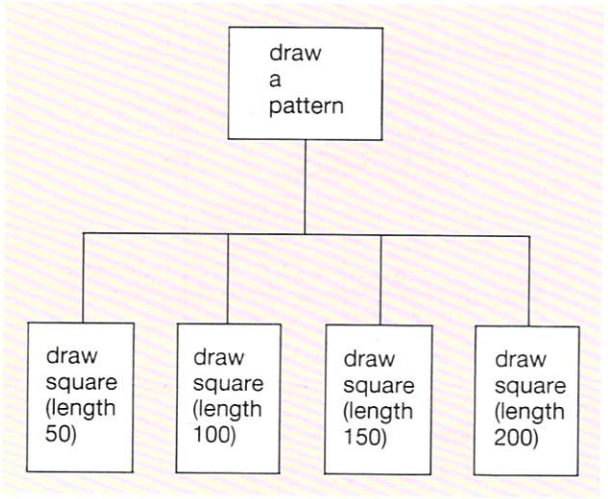
1 Repeating and getting larger



To produce this pattern, you could draw a square over and over again, making it larger each time. Here is a program plan:

```
get ready to use turtlegraphics
draw a square of length 50
draw a square of length 100
draw a square of length 150
```

The structure diagram looks like this:



2 Procedures and parameters

There's only one difference between the squares in the pattern – the length of their sides.

You can use the same procedure for drawing all the squares, provided you don't actually write the length of the side in the procedure. Instead, use a **variable** in the instructions. Call the variable `side`, or some other name:

```
PROC square(side)
  FOR round := 1 TO 4 DO
    forward(side)
    right(90)
  NEXT round
END PROC square
```

When you put `side` in brackets after the procedure name, you are making `side` a **parameter**. The procedure won't work until the parameter has been given a value. To use the procedure in the main program, replace the word in brackets with a number, like this:

```
square(50)
```

This will pass the parameter value, 50, into the procedure. The procedure will draw a square with sides of length 50.

3 Drawing the pattern – writing the program

First, load the `Turtlegraphics` demons.

Put your comment statements in lines `10`, `20` and `30`. Then type in the rest of the program:

```
100 set_up_graphics
110 square(50)
120 square(100)
130 square(150)
999 END
```

Finally, type in the procedure for drawing a square. Start this at line `1000`.

In all your programs, number the lines like this:

`10 to 99`	comments
`100 to 998`	main program
`999`	`END`
`1000 to 6999`	your own procedures
`7000 to 29999`	leave free for the ready-made demons

4 Running the program

When the computer meets the instruction `square(50)` it looks for the commands in the procedure `square`.

`PROC square(side)` tells the computer that it must give `side` a value before it can carry out the procedure square.

`square(50)` makes `side := 50`. Every time the computer meets `side` in the procedure, it will use the value 50 in its place.

5 Testing the program

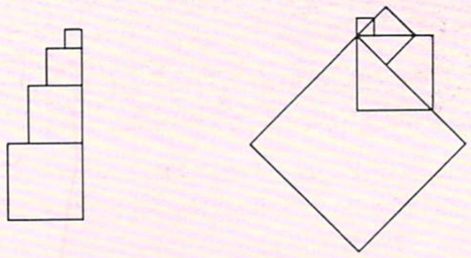
The procedure `square` is used 4 times, so check that 4 squares are drawn.

It is difficult to measure the size of the squares accurately, but the sides should get bigger by the same amount each time. Make sure that the pattern looks the same on the screen as on paper.

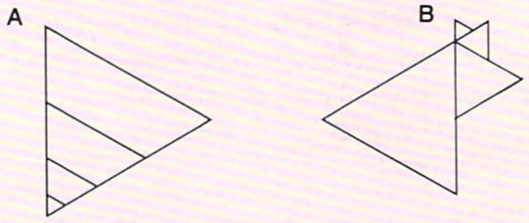
For you to do

Plan and write programs to solve these problems, using turtlegraphics.

1 Use your procedure `square(side)` to draw these shapes:



2 Write a procedure called `triangle(side)` to draw equilateral triangles (all sides the same size, and all angles the same size). Then use your procedure to draw shape **a** below.



3 Use your procedure `triangle(side)` to draw shape **b** above.

HELP

SHIFT BREAK loads the demons menu.

f0 lists your program.

f2 lists any procedures you have written yourself.

f2 lists the names of the ready-made demons.

- To *delete* a line from your program: type the line number and press RETURN
- To *change* a line: type the line number and the new statement. Then press RETURN

Start every procedure with `PROC`, then the name of the procedure, then the name of the parameter in brackets:

```
PROC square(side)
```

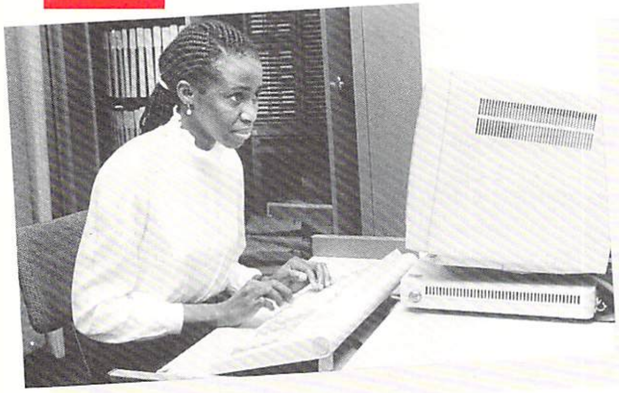
End your procedure with `END PROC` followed by the procedure name:

```
END PROC square
```

To use the procedure in the main program, write its name, then the value of the parameter in brackets:

```
square(50)
```

1.11 INVITATION TO A PARTY



The operator here is typing a name on the keyboard. She is making an **input** into the computer. The computer uses the name to print a personal invitation to the office Christmas party. The name is held in a variable, only this time, it is a word rather than a number.

1 Variables and words

When a variable stands for a word, you must put a `$` sign at the end of the variable name. For example:

```
name$ := "Lauren"
```

When a word like `Lauren` is assigned to a variable, the word must be written in quotes. Then the computer knows where the word begins and ends.

A variable, like `name$`, which stands for a word is called a **string variable**.

2 Fetching information from the keyboard

The keyword `INPUT` is used to fetch words or numbers from the keyboard. `INPUT` is always used with a variable. For example:

```
INPUT name$
```

will wait for someone to type something on the keyboard and press RETURN. `name$` will then stand for whatever is typed.

If you typed `Lauren`, then `name$` would stand for `Lauren` in the rest of the program.

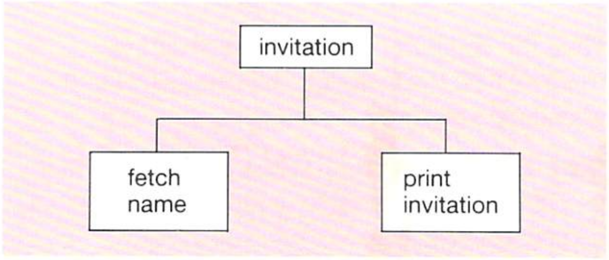
`INPUT name$` does the same job as `name$ := "Lauren"` except that `Lauren` is typed on the keyboard.

3 A party invitation – planning the program

The plan for a program to print a party invitation would look like this:

```
fetch the name from the keyboard
print the invitation
```

Here's the structure diagram:



4 Writing the program

You don't need to load any demons this time. Just type in the program:

```
 10 // Invitation
 20 // by Mike Stewart
 30 // Christmas 1988
100 PRINT "Please type a name."
110 INPUT name$
120 CLS
130 PRINT "Mike Stewart"
140 PRINT "would like to invite"
150 PRINT name$
160 PRINT "to his party"
170 PRINT "on Christmas day,"
180 PRINT "at 11 a.m."
190 PRINT
200 PRINT "R.S.V.P."
999 END
```

The computer waits at line `110` until something has been typed on the keyboard. Line `100` puts a message on the screen, so that you know what to type.

`CLS`, at line `120`, clears the screen. This gets the computer ready to start printing the next message in the top left-hand corner of the screen.

5 Running the program

When the program runs, the computer will print

```
Please type a name
```

Then it will wait until you type a name and press RETURN.

`name$` will stand for whatever you type. Later, when told to `PRINT name$`, the computer will replace `name$` by the name you typed.

You can type someone's first name, or their whole name – or any other word you like!

6 Testing the program

It is important to check that the name you type is being printed properly on the invitation.

After you type the name and press RETURN, the computer clears the screen. This means you can't compare the name you typed with the name the computer prints.

If you remove line `120`, the name you typed in will stay on the screen. So you can compare it with the name the computer prints later.

The computer should print exactly what you type. If you make a spelling mistake, the computer should make one as well!

When you are satisfied the computer is printing the name properly, put line `120` back.

You don't have to get rid of line `120` to test the program. You can disable the line instead. Just retype it as a comment statement, like this:

```
120 // CLS
```

The computer won't treat this as a command. And you won't forget what was on this line.

For you to do

Plan and write programs to solve these problems:

1. Make the computer ask you to type a name, street and town. The computer should clear the screen and print the name, street and town. Then it should stop.
   Check carefully that the computer prints exactly what you type.
2. Make the computer ask for the name of a disco, and then print a ticket for this disco.
3. Make the computer print the railway notice below. The computer must ask for a time (like 11.30 am), and put it at the end of the last line.

```
Scotrail wish to announce that
the Highland Express
will depart from platform 5
at
```

HELP

- To remove an old program from the computer and prepare it for a new one: type `NEW` RETURN
- To *list* your program: type `LIST` RETURN
- To *delete* a line from your program: type the line number and press RETURN
- To *change* a line: type the line number and the new statement. Then press RETURN
- To *clear* the screen: type `CLS` RETURN

A **string variable** is a variable which stands for a word. The name of the variable must have a `$` sign at the end, like this: `town$`

`INPUT` lets you put your own words (or numbers) into a program while it is running:

```
110 INPUT name$
```

will make the computer wait at line `110` until a word has been entered on the keyboard. The variable `name$` will then stand for that word in the rest of the program.

S Summary of Part I

If you have completed all the work so far, you should know how to:	*Covered in section:*
write a simple COMAL program	1.1
add change or delete program lines	1.1
plan, write, run and test a program	1.2
load a collection of demons from the demon disc	1.2
use demons and other procedures in your program	1.2
draw a structure diagram as part of a program plan	1.3
document your program	1.4
pass a parameter to a program	1.5
use turtlegraphics to draw shapes	1.6
use a variable	1.7
write your own procedures	1.8
use a `FOR...NEXT` loop to repeat a set of instructions	1.9
use a box in a structure diagram to show where a loop begins	1.9
write a procedure which uses a parameter	1.10
make a program fetch information from the keyboard	1.11
use a string variable	1.11

Useful line numbers

You should be able to use these line numbers in your programs correctly:

`10 to 99`	comment statements
`100 to 998`	main program
`999`	`END`
`1000 to 6999`	procedures you type in yourself
`7000 to 29999`	leave free for any ready-made demons

HELP

You should be able to use these keys for loading and listing:

SHIFT BREAK loads the demons menu from disc. Select the set of demons you want by typing a number from the menu and pressing RETURN.

f0 lists the main program (lines with numbers up to `999`).

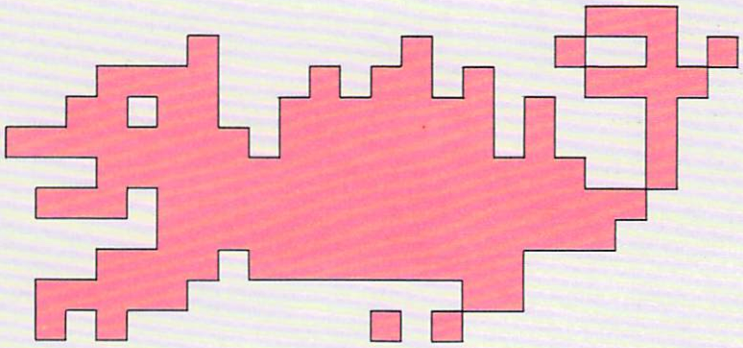
f2 lists any procedures you typed yourself.

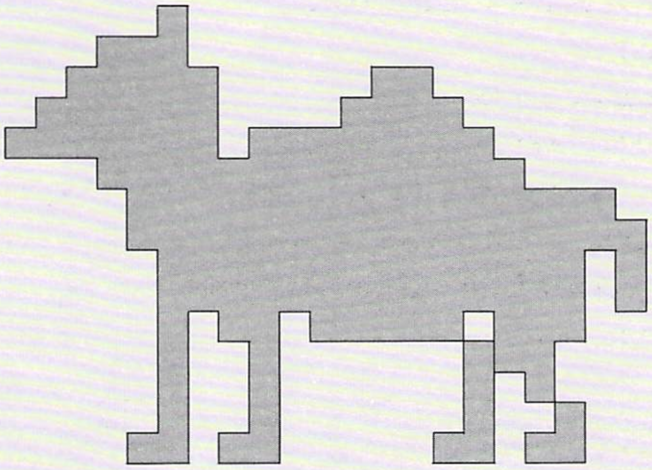
f4 lists the names of all the demons loaded from disc.

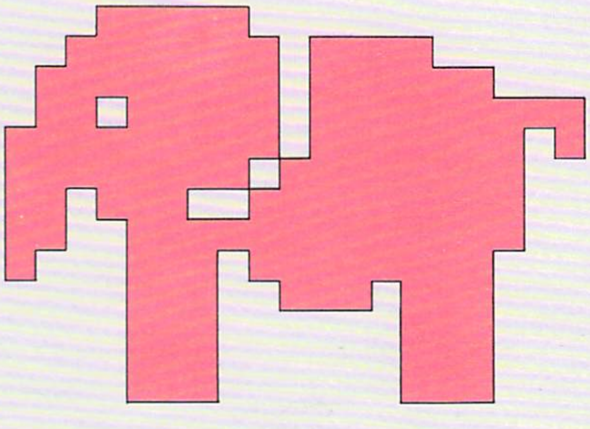
f8 takes you back to the menu, if the demons disc is in the drive.

2

TAKING COMAL FURTHER







You can use the camel, elephant and dragon demons, which produce these pictures, to advertise a zoo in unit 2.4.

2.1 LOOKING DIFFERENT

Salt!
spray – that's the message from the AA, now urging
readers to act fast and stop the rot caused by
conditions. 'CONDIMENT OF THE
SEASON' WILL RUST
YOUR CAR

Newspapers can be printed in many different styles. So can the letters on a monitor screen.

1 Modes

The BBC computer can print on the screen using 8 different styles or **modes**. These range from `MODE 0` to `MODE 7`. The number of characters across the screen, and the number of lines down the screen, are different in each mode:

Mode	Characters across the screen	Lines down the screen
0	80	32
1	40	32
2	20	32
3	80	25
4	40	32
5	20	32
6	40	25
7	40	25

You tell the computer which mode to use by using the keyword `MODE`, like this:

```
MODE := 1
```

`MODE := 1` makes the computer turn to `MODE 1`, clear the screen, and put the cursor in the top left-hand corner.

Be careful when you use `MODE 7`. This mode is used to display `TELETEXT` characters. Some of these are different from the normal characters used in the other modes.

Let's try out the different modes on a ready-made procedure – a demon called `bomb`

2 Demons in the toolbox

`bomb` is one of the ready-made procedures in the `Graphics and Sound Toolbox`. The toolbox contains all the demons in the `Sounds` collection, and many more besides. It takes a little longer to load demons from the toolbox. But you have a much wider selection to choose from.

To select demons from the toolbox:

- Use SHIFT BREAK to load the menu, then choose the toolbox.
- Load the demons you want to use by selecting numbers from the toolbox menu.
- When you have finished your selection, press `0` to exit from the menu.
- You can now start work on your program.

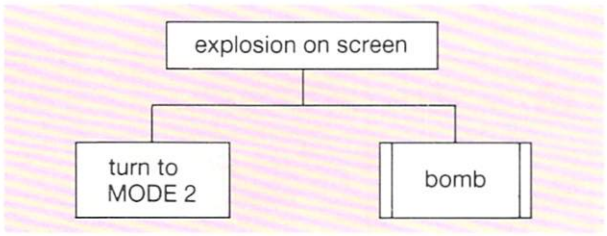
3 Explosion on screen – planning the program

Start by trying out the procedure `bomb` in `MODE 2`. The program plan is quite simple:

turn to `MODE 2`

use the ready-made procedure `bomb`

Here's the structure diagram:



4 Writing the program

First, load `bomb` from the `Graphics and Sound Toolbox`. Then type in the program:

```
 10 // Explosion on screen
 20 // by A. Name
 30 // 21st September 1987
100 MODE := 2
110 bomb
999 END
```

5 Running the program

To run the program, type `RUN` RETURN

When the program has finished, count the number of characters across the screen. Estimate the number of lines down the screen.

Try changing line `100` to

```
100 MODE := 1
```

and then running the program.

You should also try

```
100 MODE := 0
```

and note the difference.

6 Testing the program

Before running your program, list it to check that you have typed everything correctly.

The results of the program will look different in different modes, but the program should always do the same job.

If you test the program using `MODE 7` you will not be able to see the picture of the bomb properly. This is not a mistake. `MODE 7` is different from the other modes, and it can't display this picture.

HELP

- To load demons from the toolbox disc:

  use SHIFT BREAK to load the demons menu, then choose the toolbox.

  Load the demons you want by selecting their numbers on the toolbox menu.

  Choose `0` when you have finished your selection.

- To *list* your program: press f0
- To *delete* a line from your program: type the line number and press RETURN
- To *change* a line: type the line number and the new line. Then press RETURN

Modes

Modes give different sizes of letter, and different numbers of lines on the screen:

`MODE 2` and `MODE 5` give the largest letters,
`MODE 0` and `MODE 3` give the smallest letters,
`MODE 7` won't work with graphics

To select mode 1:

type `MODE := 1` and press RETURN

or add a line to your program:

```
100 MODE := 1
```

For you to do

Plan and write programs which print the following notices. Use the largest characters you can, without changing the lengths of any of the lines.

```
Dirt and dust
can damage
your computer.
Keep it covered
when not in use.
```

```
Don't touch the surface of
your discs. You may cause
damage and lose your data.
```

```
Don't switch off your computer
while you still have a disc
in the disc drive.
```

```
Keep discs in
their paper sleeves
when not in use.
```

```
Keep your computer and disc drive away from sources of heat.
```

ADDING COLOUR

The Collection.
Only 4.99 album

A computer can print white letters on a black background like this. But if you have a colour monitor, it can print in a range of other colours as well.

1 Choosing colours

On a screen, the letters and other characters are printed in the foreground colour. The screen behind them is in the background colour. Two of the demons in your `Graphics and Sound Toolbox` can be used to change these colours:

`foreground(shade$)` is used to set the foreground colour.

`background(shade$)` is used to set the background colour.

When you use these procedures, the colours on the screen don't change straightaway. But anything printed after them will use the new colours.

Each mode has a different selection of colours for you to choose from:

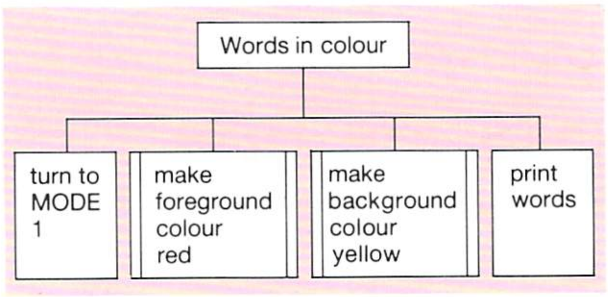
Mode	Colour
0	black, white
1	black, white, red, yellow
2	black, white, red, yellow, green, blue, magenta, cyan
3	black, white
4	black, white
5	black, white, red, yellow
6	black, white
7	the procedures `foreground (shade$)` and `background (shade$)` have no effect in `MODE 7`

2 Words in colour – planning the program

Try using the two colour demons to print some words in colour. Here is one way of doing the job:

turn to `MODE 1`
make the foreground colour red
make the background colour yellow
print the rhyme

The structure diagram looks like this:



3 Writing the program

First, select `fore and background colours` from the toolbox menu:

Then type in your program:

```
 10 // Words in colour
 20 // by R. Logan
 30 // 22nd September 1988
100 MODE := 1
110 foreground("red")
120 background("yellow")
130 PRINT "Old computers"
140 PRINT "never die."
150 PRINT "They just lose"
160 PRINT "their memories."
999 END
```

The parameters needed by `foreground(shade$)` and `background(shade$)` are words. When a parameter is a word, like `red` or `yellow` it must be written inside quotes. The computer then knows where the word begins and ends.

4 Running the program

When the program runs, and the computer reaches line `110`, it will search for the instructions for the procedure `foreground`. In this case, the word `red` inside the brackets is passed to the procedure.

A similar thing happens in line `120`. The word `yellow` is passed to the procedure `background`.

The foreground and background colours don't change straight away, but the computer uses these colours when it prints on the screen.

5 Testing the program

Go through lines `10` to `999` and write down what the computer should do at every line. This is called a **dry run**. Its a good way of working out what a program should do.

A dry run is like a rehearsal. You run through the program to see if there are any snags before the main performance begins.

When the program runs, make sure it does all the jobs listed in your dry run.

Keep the dry run you have written. You may want to refer to it later.

For you to do

Plan and write programs to solve these problems (you will find the demons in the `Graphics and Sound Toolbox` useful when you solve each one):

1 Print the name of a person, team or rock group in blue letters on a white background.

2 Print the name of your town four times. Put it on a different line each time, using four different foreground colours and four different background colours.

3 If you set the background colour, then clear the screen, the background colour of the whole screen will be changed. For example:

```
background("yellow")
CLS
```

will colour the whole screen yellow.

Change the colour of the whole screen to red, then print `COMAL COLOURS` in yellow in the centre of the screen.

HELP

- To load demons from the toolbox:

  Use SHIFT BREAK to load the demons menu, then choose the toolbox.

  Load the demons you want by selecting their numbers on the menu.

  Choose `0` when you have finished your selection.

- To *list* your program: type `LIST,999` RETURN
- To *delete* a line from your program: type the line number and press RETURN
- To *change* a line: type the line number and the new statement. Then press RETURN

`foreground(shade$)` sets the colour of any letters printed on the screen. For example:

```
110 foreground("red")
```

will make letters be printed in red at all `PRINT` instructions after line `110`.

`background(shade$)` sets the colour of the screen behind the letters. For example:

```
120 background("yellow")
```

will turn the screen yellow from the first `PRINT` instruction after line `120` onwards.

The procedures will only work with colours given in the table on the opposite page.

2.3 LIBRARY TICKETS

The library ticket on the right was printed by computer. Any name and address could have been used. And they would have been printed in exactly the same positions. Positioning the words is all part of the program.

```
-----------------------
          LAMOC LIBRARY
Name      Leela Chopra
Address   4,Bridge Lane
          Ayr
          Scotland
-----------------------
```

1 Printing positions

To control where the computer starts printing along a line, you can use
a comma `,`   a semicolon `;`   or a `TAB` command

A **comma** makes the computer move a few spaces across the screen before printing the next item:

```
PRINT "Name","Leela"
```

puts this on the screen:

```
Name    Leela
```

A **semicolon** doesn't make the computer move. So the next item is printed immediately after the last:

```
PRINT "Name";"Leela"
```

puts this on the screen:

```
NameLeela
```

A `TAB` command makes the computer move to any position you choose. For example, in `MODE 1`, there are 40 printing positions on each line. These are numbered 0 to 39.

```
PRINT TAB(11);"Leela"
```

makes the computer move to position 11 before printing:

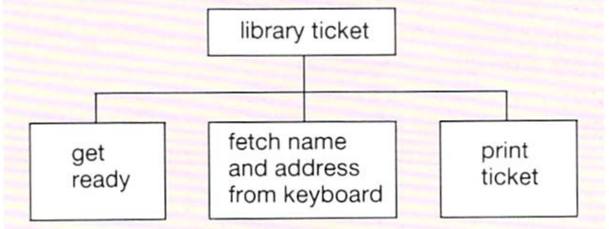
```
           Leela
```

2 Printing the ticket – planning the program

The program plan is quite simple:

get ready
fetch name and address from keyboard
print ticket

The structure diagram looks like this:



3 Writing the program

The program is listed at the top of the next page.

At the start of the program, the name, street, town and country are assigned to string variables, because words are being used.

The top and bottom edges of the ticket are each printed using `23` hyphens.

In line `210`, nothing follows the `PRINT` command. So a blank line is printed. The same thing happens in line `280`.

In line `240`, the computer is told to print two items. The first is the word `Name`, starting at printing position 2. The second is the word which `name$` stands for. This starts at printing position 11. The semicolons after each item stop the computer moving across the screen before carrying out the next bit of the `PRINT` command.

```
 10 // Library ticket
 20 // by A. Reader
 30 // October 1987
100 MODE := 1
110 PRINT "Type the name"
120 INPUT name$
130 PRINT "Type the street"
140 INPUT street$
150 PRINT "Type the town"
160 INPUT town$
170 PRINT "Type the country"
180 INPUT country$
190 CLS
200 PRINT "-----------------------"
210 PRINT
220 PRINT TAB(9); "LAMOC LIBRARY"
230 PRINT
240 PRINT "Name";TAB(9);name$
250 PRINT "Address";TAB(9);street$
260 PRINT TAB(9);town$
270 PRINT TAB(9);country$
280 PRINT
290 PRINT "-----------------------"
999 END
```

4 Running the program

When the program runs, question marks appear on the screen. A reply has to be typed in for each, like this:

```
Type the name
?Leela Chopra
Type the street
?4, Bridge Lane
Type the town
?Ayr
Type the country
?Scotland
```

When all the information has been entered, the computer should print out the ticket on the screen.

HELP

- To *list* your program: type `LIST` RETURN
- To *clear* the screen: type `CLS` RETURN

5 Testing the program

Use a piece of graph paper to test the program. Mark out 40 boxes across, then 32 boxes from top to bottom. That's the same as the printing positions on the screen in `MODE 1`.

Carry out a dry run, doing exactly what the program says on graph paper. You should end up with a library ticket on the graph paper.

Run the program. Type in the same details as you used for the dry run. Check that you end up with a library ticket on the screen which looks the same as the ticket on your graph paper.

For you to do

Plan and write programs to solve these problems:

1 Make the computer print a label for a disc box. The computer should ask for

    the box number
    contents (first line)
    contents (second line)
    contents (third line)

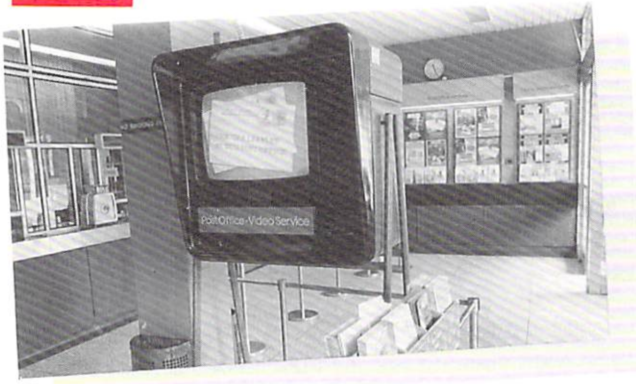
Your label should look something like the one below. You can use exclamation marks to print the lines at the sides of the ticket:

```
 -----------------------
!                       !
!     Disc Box 14       !
! Contents: COMAL games !
!           for the     !
!           BBC computer!
!-----------------------!
```

2 Karen owns a mobile disco. She often has to print tickets showing where the disco is to be held (that's the **venue**), the time it begins and ends, and the price of admission.

Design a ticket. Make the computer ask for the venue, starting and finishing times, and the price. Then print ten tickets.

2.4 COME TO THE ZOO



As you stand in the queue at the Post Office, the adverts flash up on the screen one after another.

With the demons in your `Graphics and Sound Toolbox`, you can make animals appear on the computer screen one after another. Try making your own advert – for a zoo.

1 More parameters

`camel`, `elephant` and `dragon` are demons which draw animals on the screen. Each demon uses *two* parameters:

the position across the screen

the position down the screen

The demons can't do their job until their parameters have been given values. This is done by placing numbers in brackets after the demon's name. For example:

`camel(18,8)` will draw a camel 18 printing positions across the screen, and 8 printing positions down.

`centre` is a demon which you can use to print the name of the animal under the drawing. It uses *two* parameters: the word to be printed, and the position down the screen

The first parameter value is a word. So it must be put between quotes. For example:

`centre("CAMEL", 24)` will print the word `CAMEL` in the middle of line 24.

You can use the demon `pause` to delay the computer before it changes words or pictures on the screen. For example:

`pause(1)` will make the computer pause for 1 second. Higher numbers give a longer pause.

2 Advertising the zoo – planning the program

The plan below is for a program which shows three animals, one after another. It also prints messages about the zoo:

```
print a message
show the camel
    draw the camel
    print the camel's name
show the elephant
    draw the elephant
    print the elephant's name
show the dragon
    draw the dragon
    print the dragon's name
print a message
```

HELP

`elephant(18,8)` will draw an elephant 18 printing positions across the screen, and 8 printing positions down.

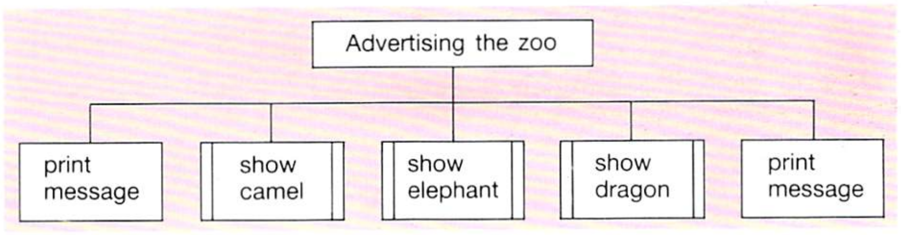
`centre("ELEPHANT", 24)` will print the word `ELEPHANT` half way across the screen, and 24 printing positions down.

`pause(3)` will make the computer pause for 3 seconds before moving on to the next line of the program.

`gallop(20, "forward")` will make a horse and rider move across the screen from left to right, 20 printing positions down. For movement from right to left, use "`backward`" instead of "`forward`".

`drive(20, "car", "forward")` will make a car travel across the screen from left to right, 20 printing positions down. You can use "`bus`" instead of "`car`", and "`backward`" instead of "`forward`".

Here is the structure diagram:



3 Writing the program

First, load the demons you need from your `Graphics and Sound Toolbox`. Do this by selecting the following options on the menu:

`pause`

`dragon, camel and elephant`

`centre a message`

Start your program with a few comment statements. Then type in the main program:

```
100 MODE := 1
110 PRINT "Visit Ayr zoo"
120 pause(3)
130 show_camel
140 show_elephant
150 show_dragon
160 PRINT "Open every day"
170 PRINT "except Christmas day"
999 END
```

Next, type in the procedures `show_camel`, `show_elephant` and `show_dragon`. Use line numbers between 1000 and 6999 for these.

Each procedure should draw an animal and print its name underneath. Here is the procedure for showing a camel:

```
1000 PROC show_camel
1010 CLS
1020 camel(18,8)
1030 pause(3)
1040 centre("CAMEL",24)
1050 pause(3)
1060 END PROC  show_camel
```

The procedures for showing the elephant and the dragon are similar. Type these in next.

4 Running the program

When the program runs, the message `Visit Ayr zoo` is printed. A camel, an elephant and a dragon then appear on the screen one after another, with their names printed underneath. The screen display changes every 3 seconds.

At the end of the program, another message is printed on the screen.

For you to do

Plan and write programs to solve the problems below. You will need demons in your `Graphics and Sound Toolbox` for this. Select these options from the menu before typing in your programs:

`pause`

`centre a message`

`gallop and drive`

The demons all use parameters which must be given values. The **HELP** box shows you how.

1 `gallop` is a demon which makes a horse and rider gallop across the screen.

Make the computer print `Horse and rider` in the centre of the top line of the screen. Then make it show the horse and rider galloping across the bottom of the screen and back again.

2 `drive` is a demon which makes a bus or car travel across the screen. Use this demon to show a bus moving from left to right across the screen, then a car moving from right to left.

2.5 FUN WITH WORDS

Election time. Boxes of voting papers are delivered by collectors. These tellers then count the votes and send their results to the Returning Officer.

In computers, some procedures are like collectors. They do a particular job. But some procedures are like tellers. When they have finished their job, they send a result back to the main program.

Procedures which send back or return a result are called **functions**.

1 Using functions

Some functions return a number. The built-in COMAL function `LEN` is one of these. `LEN` counts the number of letters in a word. For example:

```
length := LEN("Hazel")
PRINT length
```

will make the computer print the number **5**. The function `LEN` works out the result **5**. So the variable `length` takes the value **5**.

Some functions return a string (a word or letter). The demon `first_letter$` is one of these. It returns the first letter of a word. For example:

```
first$ := first_letter$("Hazel")
PRINT first$
```

will make the computer print the letter `H`. The function `first_letter$` works out the result `H`. So the variable `first$` takes the value `H`.

The functions `first_letter$` and `LEN` each use one parameter. They can't do their job until the parameter has been given a value, like this:

```
first_letter$("Hazel")
LEN("Hazel")
```

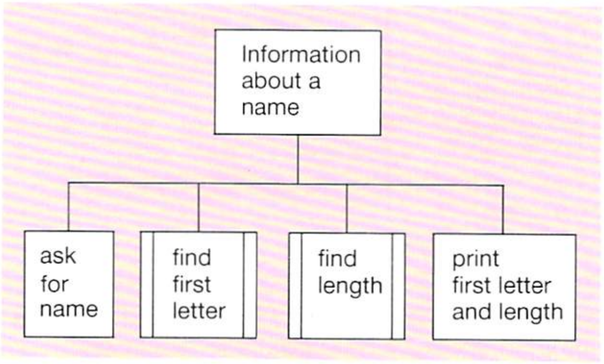
Functions, like `first_letter$`, which return a string all have a `$` at the end of their name.

2 Information about a name – planning the program

The plan below is for a program which will ask you for someone's name, then print the first letter and the length:

```
ask for name
find first letter
find length
print first letter and length
```

Here's the structure diagram:



HELP

SHIFT BREAK *loads* the demons menu from disc.

f0 *lists* your program.

f4 *lists* the names of the ready-made demons you can use. There isn't room on the screen for the full list. Press SHIFT to see more.

variables which stand for strings (words or letters) must have a `$` at the end of the variable name. For example: `first$`

3 Writing the program

First, load the `Words and Numbers – level 2` demons from disc. Start your program with a few comment statements. Then type in the main program:

```
100 MODE := 3
110 PRINT "Type your first name"
120 INPUT name$
130 first$ := first_letter$(name$)
140 length := LEN(name$)
150 PRINT "Your name begins with "
160 PRINT first$
170 PRINT "Number of letters is"
180 PRINT length
999 END
```

4 Running the program

When the program runs, the name you type in at line `120` replaces the variable `name$` in the rest of the program.

The function `first_letter$(name$)` works out the first letter of this name. This letter now replaces the variable `first$` in the rest of the program.

The function `LEN(name$)` counts the number of letters in the name. This number replaces the variable `length` in the rest of the program.

When the computer reaches the instructions `PRINT first$` and `PRINT length`, it prints the first letter and the length of the name.

```
result$:= last_letter$("Hazel")
PRINT result$
```
will make the computer print the letter `l`

```
result$:= first_letter$("Hazel")
PRINT result$
```
will make the computer print the letter `H`

```
result$:= backward$("Hazel").
PRINT result$
```
will make the computer print `lezaH`

5 Testing the program

You can't carry out an exhaustive test of this program. There would be too many names to try. Instead, test the program selectively. Choose just a few different kinds of names.

Try a long name, of up to 40 letters. Then try a very short name with 0 letters. Don't type any letters, just press RETURN

After that, try names which start with a capital letter or a lower case ('small') letter.

For you to do

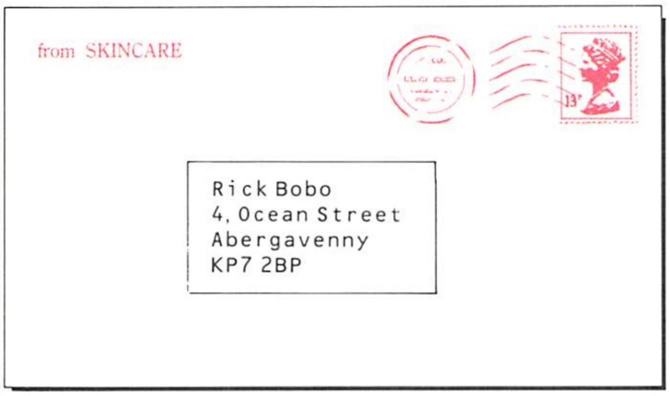
Plan and write programs to solve the problems which follow. Each of the functions mentioned is stored in `Words and Numbers – level 2.` Read the **HELP** box first.

1 The function `last_letter$` returns the last letter of a word.
Make the computer ask for a word, then print its last letter and its length.

2 The function `backward$` returns a word with its letters in reverse order.
Make the computer:
ask for a word
print its first letter and its length
print the word spelt backwards
print the first letter and the length of the word spelt backwards

3 The function `die` returns a whole number between 1 and 6, just like tossing dice. The function `odd_or_even$` returns the word `ODD` or the word `EVEN`, according to whether the number you put in the brackets is odd or even.
For example: `odd_or_even$(5)` will return the word `ODD`.
Make the computer behave like electronic dice. It must print a number between 1 and 6, and state whether the number is odd or even.

2.6 ADDRESS LABELS



Skincare is a a small company, selling cosmetics by mail order. Every month, they have to send out a new price list to their customers. So they need a regular supply of printed address labels.

Fortunately, their secretary doesn't have to type out a whole new set of labels each time. The job can be done by computer.

1 Storing information inside a program

The secretary only wants to type out the names and addresses once – when the program is written. After that, the computer should be able to print out a full set of labels every time the program is run.

`INPUT` statements won't do the job, because the names and addresses would have to be typed each time.

`PRINT` statements could be used for the names and addresses. But that would make the program long, and difficult to type in.

There is a better way:

`DATA` statements can be used to store the information inside the program.

`READ` statements can be used to recall the information stored in the `DATA` statements.

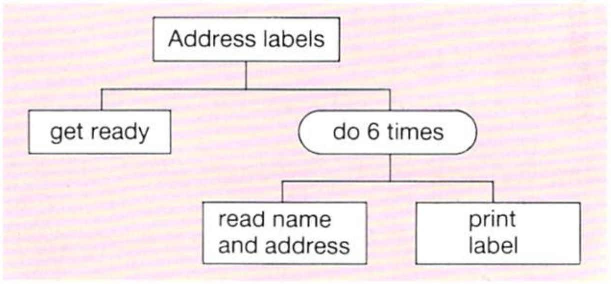
To find out how `READ` and `DATA` statements work, start with a program for printing just six names and addresses.

2 Address labels – planning the program

The program plan looks like this:

```
get ready
do 6 times
   read name and address
   print label
end loop
```

Here's the structure diagram:



3 Writing the program

The program is listed at the top of the next page.

This program prints the labels on the screen. If you want it to print the labels on a printer as well: first, load the `switch_printer_on` and `switch_printer_off` demons from your **toolbox**; then, type in the program with two extra lines added like this:

```
105 switch_printer_on
210 switch_printer_off
```

The `DATA` statements are given line numbers from `30000` onwards. That way, they won't interfere with any demons being used.

HELP

- To *load* the `switch_printer_on` and `switch_printer_off` demons from your toolbox:

  select '`Printer on and off`' from the menu

```
   10 // Address label
   20 // by Skincare
   30 // 21 March 1988
  100 MODE := 1
  110 FOR label := 1 TO 6 DO
  120    READ name$, address$, town$, postcode$
  130    PRINT "--------------------"
  140    PRINT name$
  150    PRINT address$
  160    PRINT town$
  170    PRINT postcode$
  180    PRINT "--------------------"
  190    PRINT
  200 NEXT label
  999 END
30000 DATA "R. Bobo","4 Ocean St.", "Ayr", "KP7 2BP"
30010 DATA "M. Mill", "12 Bell St.", "Bath", "BI4 7KJ"
30020 DATA "Z. Mohamed", "23 Canter St.", "Hull", "HP7 6YT"
30030 DATA "D. Rix", "121 Mill St.", "Aber", "DF5 6RT"
30040 DATA "S. Smith", "3 West St.", "Troon", "LPO 4TR"
30050 DATA "D. Adams", "45 Cook St.", "London", "WC4 5HU"
```

4 Running the program

At line `120`, the computer is told to `READ name$, address$, town$, postcode$`. It begins its search for `DATA` and first finds it at line `30000`. It assigns the first item to `name$`, the second to `address$`, the third to `town$` and the fourth to `postcode$`.

The next time round the loop, the computer is told to `READ` data again. The computer has its own built in 'pointer' to mark where it last got to in its search. So, it reads `DATA` at line `30010`. And so on.

5 Testing the program

When the program runs, the computer should print the labels on the screen. If the printer demons are being used, it should print the labels on the printer as well. Using the printer is a good idea, because it makes it easier to compare the labels with the data.

Examine each label to make sure that the names and addresses have all been printed correctly.

For you to do

Plan and write programs to solve these problems:

1 Alter the address label program above so that it will print ten labels.

2 Make the computer print a list like this:

```
Skincare price list: November
1987

Product            Size   Price(£)
-------------      -----  -------

Cleansing cream   100 ml  1.95
Night cream        50 ml  2.95
Throat cream       50 ml  3.65
Facial cocktail   250 ml  1.95
Eye cream          20 ml  1.55
Moisturiser       200 ml  2.50
Hand cream        250 ml  1.25
Toner             150 ml  1.75
```

3 Make the computer print five short letters. The contents of each letter should be the same, but the name of the person on each letter should be different.

2.7 IN THE SWING

There are ten players in Angie's golf team. When they all play a practice round, she needs to know their average score. That way, she can keep a check on their performance. Angie keeps all the score cards. So she can use a computer to work out the results.

1 Signs

Computers are good at arithmetic. They can add, subtract, multiply and divide. But some of the signs they use are different from the ones you write on paper. On the keyboard:

+ means 'add'	* means 'multiply'
– means 'subtract'	/ means 'divide'

Here is how you could solve four simple problems using the computer:

Problem	On the computer

What is 6 + 2 ?	`answer := 6 + 2`
	`PRINT answer`

What is 6 – 2 ?	`answer := 6 - 2`
	`PRINT answer`

What is 6 × 2 ?	`answer := 6 * 2`
	`PRINT answer`

What is 6 ÷ 2 ?	`answer := 6 / 2`
	`PRINT answer`

2 Totals and averages

Here's one way of finding the total of 10 numbers:

```
write down 0 (the total so far!)
for each score
   add the score to the total
   write down the result (the total so far)
next score
```

When you have added each of the scores, the last number you wrote down is the **total** score.

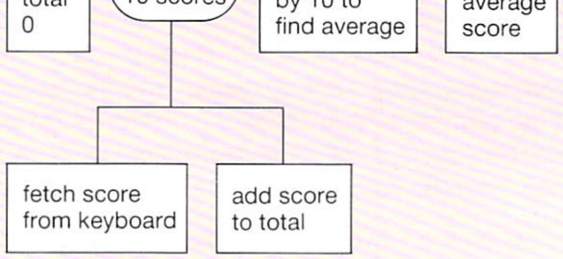
To work out the **average** score, you divide the total score by the number of golfers (10).

3 Finding an average score – planning the program

Here's one way of doing the job on the computer:

```
make the total 0
for 10 scores
   fetch a score from the keyboard
   add the score to the total
next score
divide the total by 10 to find the average
print the average score
```

The structure diagram looks like this:



4 Writing the program

No demons are needed, but you should get the computer ready for a new program by typing NEW RETURN

Start with a few comment statements. Then type in the main program:

```
100 MODE := 6
110 total :=0
120 FOR golfer := 1 TO 10 DO
130   INPUT "Type next score ": score
140   total := total + score
150 NEXT golfer
160 average := total / 10
170 PRINT "Average score is "; average
999 END
```

5 Running the program

When the program runs, the total is first set to 0. Then the computer is told to go round a loop 10 times. `golfer` is used as a control variable, to keep track of the number of times the computer has gone round the loop.

The instructions inside the `FOR...NEXT` loop are repeated, 10 times in all. Each time round the loop, the computer asks for the next score. At line `140`, it adds this score to the total:

```
total := total + score
```

This line gives `total` a new value each time. It's the old value plus the last score added.

After all 10 scores have been added, the total is divided by 10. This gives the average score.

6 Testing the program

Use a dry run to check that the program is working properly.

Begin with a list of the 10 scores you will use when the program runs. Work through the program line by line. Write down the value of the total every time it is changed by the program.

When you have finished the dry run, check your final answer by another method. You could use a pocket calculator! When the program runs, check the computer's answer.

For you to do

Plan and write programs to solve these problems:

1 Use the computer to work out the average height of 12 people.

2 A disc jockey in a local radio station often plays several short jingles one after the other, to fill up time. The DJ needs to work out how long this will take.

Use the computer to calculate the total time it takes to play the jingles. The computer should print the answer twice; first in seconds, then in minutes.

3 A store assistant in a dairy looks after crates of eggs. Every morning she starts with 250 crates. She sells some of these to 12 milkmen who load their floats at the dairy.

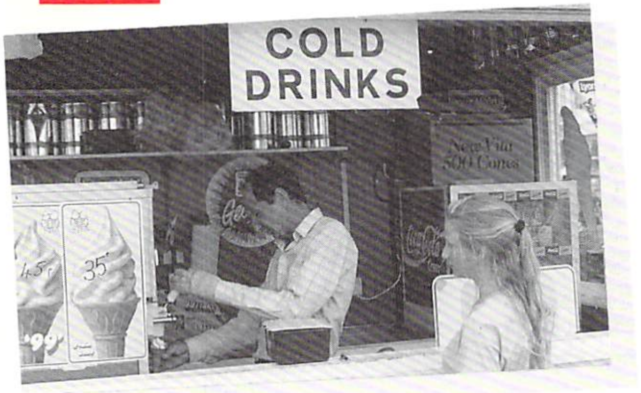
Use the computer to print the number of crates left in the store as each milkman buys some eggs.

HELP

- To *clear* the screen: type `CLS` RETURN
- To *list* your program: type `LIST` RETURN
- To *delete* a line: type the line number and press RETURN
- To *change* a line: type the line number and the new statement. Then press RETURN
- To prepare the computer for a *new* program: type `NEW` RETURN
- *Start* your program with comment statements:

```
10 // the name of the program
20 // the author's name
30 // the date when it was written
```

2.8 THE ICE CREAM PARLOUR



Mike runs an ice cream parlour. In it, he sells vanilla, raspberry and chocolate flavoured cones. The cones cost him 6p each to make. Here are the amounts people have to pay for them:

Ice cream cones price list

Flavour	Cost
vanilla	20p
raspberry	25p
chocolate	30p

1 Working out the profit

At the end of each day, Mike needs to know how much profit he has made. He must work out the total amount paid for all the cones, then take away the total cost of making them. For example:

Total amount paid for cones	£72.50
Total cost of making cones	−£18.00
Profit	£54.50

2 Planning the program

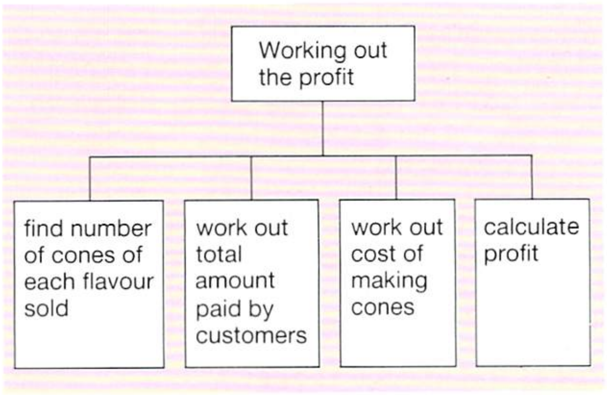
Here are Mike's sales figures for one day last June:

Flavour	Number sold
vanilla	100
raspberry	150
chocolate	50

You can use these numbers in a program which works out Mike's profit for the day. Here is one way of tackling the problem:

find the number of cones of each flavour sold
work out how much the customers paid
work out how much the cones cost to make
calculate Mike's profit

Here's the structure diagram:


Working out the profit
find number of cones of each flavour sold
work out total amount paid by customers
work out cost of making cones
calculate profit

HELP

- To prepare the computer for a *new* program: type `NEW` RETURN

In your programs, number the lines like this:

`10 to 99`	comment statements
`100 to 998`	main program
`999`	`END`
`1000 to 6999`	procedures you type in yourself
`7000 to 29999`	leave free for any ready-made demons
`30000 to 32767`	information stored in DATA statements

- To *list* the main program: type `LIST,999` RETURN
- To *list* any procedures you typed in yourself: type `LIST 1000,6999` RETURN
- To *clear* the screen: type `CLS` RETURN
- To *delete* a line from your program: type the line number and press RETURN

3 Writing the program

Don't use any demons. Just type this program in.

```
  10 // Working out the profit
  20 // by Mike
  30 // 6 June 1988

 100 MODE := 6
 110 how_many_cones
 120 how_much_paid
 130 cost_to_make
 140 calculate_profit
 999 END

1000 PROC how_many_cones
1010 PRINT "Type the number of"
1020 PRINT "vanilla, rasp and choc"
1030 PRINT "flavoured cones sold"
1040 INPUT van, ras, choc
1050 END PROC how_many_cones

1100 PROC how_much_paid
1110 vanpaid := van * 20
1120 raspaid := ras * 25
1130 chocpaid := choc * 30
1140 paid := vanpaid + raspaid + chocpaid
1150 END PROC how_much_paid

1200 PROC cost_to_make
1210 numbersold := van + ras + choc
1220 cost := numbersold * 6
1230 END PROC cost_to_make

1300 PROC calculate_profit
1310 profit := paid - cost
1320 pounds := profit / 100
1330 PRINT
1340 PRINT "Profit is £"; pounds
1350 END PROC calculate_profit
```

4 Running the program

The procedure `how_many_cones` asks you to type the numbers of vanilla, raspberry and chocolate flavoured cones sold. These numbers replace the variables `van`, `ras` and `choc` in the rest of the program.

The procedure `how_much_paid` calculates the total amount paid for the cones. For each flavour, the number of cones is multiplied by the cost of each cone. These amounts are then added to find the total amount paid.

The procedure `cost_to_make` calculates the total cost of making the cones. It multiplies the total number of cones sold by the cost of making each one.

The procedure `calculate_profit` subtracts the total cost of making the cones from the total amount paid for them. It divides the answer by 100 to give the profit in pounds. Then it prints the result.

5 Testing the program

Do a dry run to check that the computer is doing the calculations properly.

Test the program using small numbers of cones as well as large numbers.

Make sure the program works properly when no cones at all are sold. Check that the program can still produces the right answers if some flavours of cones are not sold at all.

For you to do

Plan and write programs to solve these problems.

1 A takeaway shop sells portions of chicken and fried rice for £2.15 each.

Make the computer:
ask how many portions have been sold
print the total amount of money taken

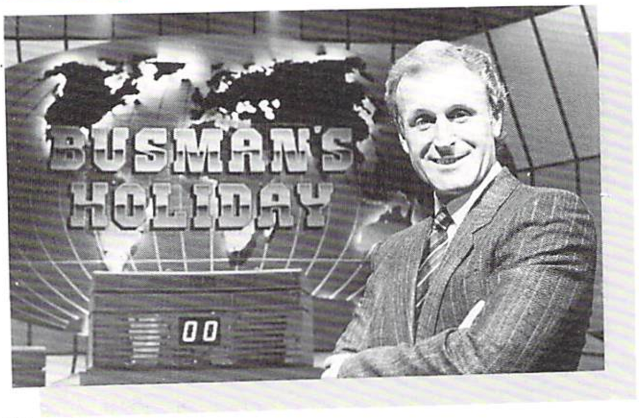
2 Five people make corkscrews in a factory. They are paid 12p for each corkscrew they make.

Use `DATA` statements to hold each person's name, and the number of corkscrews they make. Then make the computer calculate and print their pay.

3 A 'phone bill is made up of:
a system charge of £14.00 for being connected
an apparatus charge of £7 for the telephone
a unit charge of 5p for each call made

Make the computer:
ask for the number of calls made
calculate and print the 'phone bill

2.9 QUIZ PROGRAM



Here is one type of quiz programme. The host puts the questions, and decides whether the contestants' answers are right or wrong.

1 Deciding on the answer

A computer can make decisions using `IF... THEN...ELSE` statements.

The computer might ask
`What is the capital of Scotland?`

`IF. . . .` your answer is `Edinburgh`
`THEN..` the computer prints `Correct`,
`ELSE..` it prints `Wrong`.

When the computer checks your answer against `Edinburgh`, it is **testing a condition**.

`END IF` must be used to tell the computer that it has finished testing the condition.

2 Capitals quiz – planning the program

The job of the program is to print the names of 10 different countries on the screen, one after another. It must ask for the capital of each. When you type in each answer, it must tell you whether you are right or wrong.

The countries and correct answers for capitals can be stored as `DATA`.

A `READ` statement can be used to bring data into the main program.

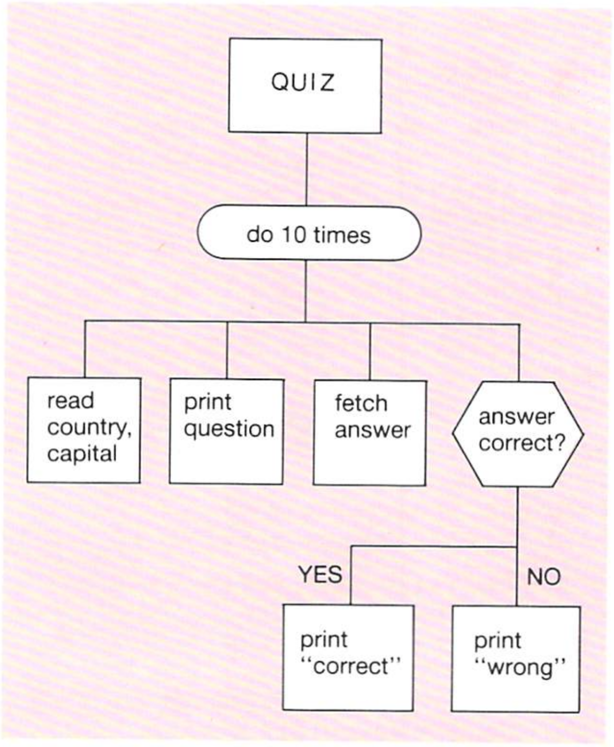
A loop can be used to put each country on the screen in turn.

An `IF...THEN...ELSE` statement can be used to check each answer against the correct one and tell you whether you are right or wrong.

Here is a program plan:

```
do 10 times
   read country and capital
   print question
   fetch answer
   if answer is correct, print Correct
   else print Wrong
end loop
```

The structure diagram looks like this:



Questions are written in **decision boxes**. A decision box has pointed ends and two exits – one for YES and one for NO.

HELP

- To *list* the main program: type `LIST,999` RETURN
- To *list* the `DATA` statements: type `LIST 30000,` RETURN
- To *clear the screen:* type `CLS` RETURN

3 Writing the program

Here is the program itself:

```
100 MODE := 1
110 FOR turn := 1 TO 10 DO
120   READ country$,capital$
130   PRINT "Type capital of";country$
140   INPUT guess$
150   IF guess$ = capital$ THEN
160   PRINT "Correct"
170   ELSE
180   PRINT "Wrong"
190   END IF
200 NEXT turn
999 END

30000 DATA FRANCE, PARIS, SPAIN, MADRID
30010 DATA RUSSIA, MOSCOW, ITALY, ROME
30020 DATA TURKEY, ANKARA, U.K., LONDON
30030 DATA GREECE, ATHENS, CUBA, HAVANA
30040 DATA PERU, LIMA, JAPAN, TOKYO
```

A `FOR...NEXT` statement is used to create the loop.

Inside the loop, `READ` is used to fetch each country and its capital, from `DATA` statements.

The `INPUT` statement is used to assign your answer to the variable `guess$`. The computer decides whether the answer was correct or wrong, by testing the statement

`guess$ = capital$`

It prints `Correct` if the statement is `TRUE`, or `Wrong` if the statement is `FALSE`.

(Note: When testing whether two values are equal, use = by itself, not := .)

If you get an `Out of data` message:

check the `DATA` statements for missing words or missing commas.

Try checking the next country and capital after the last one the computer used. That is probably where the mistake will be found.

4 Running the program

When the program runs, the computer should repeatedly print `What is the capital of` and the name of a country. It will then wait until a name is typed. The name is checked, and `Correct` or `Wrong` will be printed.

When the computer reads the names of the countries, and their capital cities, they are in capital letters, so you must type your answers using capital letters.

5 Testing the program

First, check the loop. Run the program and make sure that the computer asks the question `What is the capital of...` ten times.

Next, check the `IF...THEN...ELSE` by making sure the computer prints `Correct` and `Wrong` at the right times.

For you to do

Plan and write programs to solve the following problems.

1 Create a quiz in which you have to tell the computer the names of the groups who sang various songs.

The computer should ask you about 12 songs. For each song, the computer should print `Well done!` if you type the correct group, or `No, that's not right` if you type the wrong group.

2 Robotic Assemblies is a large company which uses robots to assemble industrial goods. Most of the people who work in the factory are paid £4 for each hour they work. Supervisors are paid £5 per hour.

Write a program which will:

ask for the number of hours a person has worked in a week
ask if the person is a supervisor
calculate their pay

2.10 DRIVING A BUS

This 'superbus' can travel very fast. But it's still not as fast as the bus on the computer screen!

1 The bus demon

In your `Graphics and Sound Toolbox`, there is a demon called `bus(x,y)`. If this demon has been loaded, and you type:

```
MODE := 1 RETURN
bus(0,0) RETURN
```

a small bus will appear in the top left hand corner of the screen.

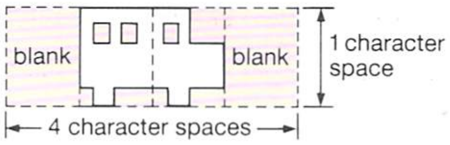
`x` and `y` are parameters which control the position of the bus. In `MODE 1`:

`x` can be given values from 0 to 36. The higher the number the further across the screen the bus will be printed.

`y` can be given values from 0 to 31. The higher the number, the further down the screen the bus will be printed.

You can move the bus by writing a program which makes the value of `x` change, by one each time, from 0 to 36.

The bus itself takes up four character spaces on the screen. A blank character is printed at the front of the bus, and another at the back. These blanks act as **erasers**. They print over any 'old' images left on the screen when the bus moves.



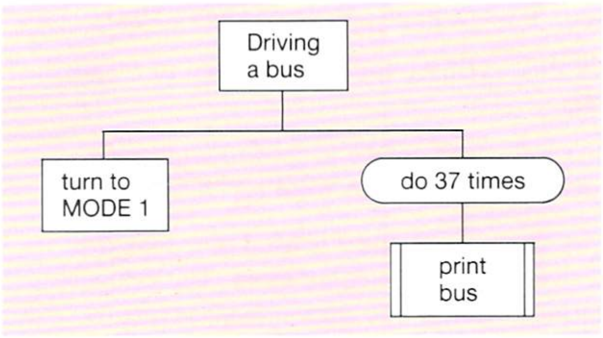
2 Moving the bus – planning the program

To make the bus move, you need a program with a loop in it, so that the bus is printed on the screen over and over again very quickly. If each position is a little different from the one before, you see the bus moving. The effect is called **animation**. It's used in cartoons on TV.

Here is a possible program plan:

```
turn to MODE 1
do 37 times
  print bus
end loop
```

The structure diagram would look like this:



A `FOR..NEXT` loop is useful for this program because its control variable can do two jobs at once. It can count the number of times the computer goes round the loop. And it can also control the position of the bus by giving the parameter `x` its value.

HELP

- To *exit* from a program while it is running: press ESCAPE
- To *list* the main program: type `LIST,999` RETURN
- To *clear* the screen: type `CLS` RETURN
- To *delete* a line from your program: type the line number and press RETURN

3 Writing the program

First, load the bus demon from your Graphics and Sound Toolbox. Do this by selecting the bus and car demons from the menu. Select the hide and show cursor, and the pause demons as well. You may want to use these later.

Start off with some comment statements. Then, type in the main program:

```
100 MODE := 1
110 FOR across := 0 TO 36 DO
120   bus(across,20)
130 NEXT across
999 END
```

In the FOR...NEXT loop, across is the control variable. As the loop is repeated, across takes the values 0, 1, 2. . . . ..36, one after the other.

The value of across is also used to position the bus across the screen. The bus is printed in position (0,20), then (1,20), then (2,20), then (3,20) . . . ... and so on.

The bus finally stops at (36,20), because the loop is carried out for the last time when across is 36.

When you are using the control variable to do a job like this, it is very important that you do not alter its value. Otherwise the computer will not be able to keep proper count as it goes round the loop, and might go round the wrong number of times.

bus(x,y) draws a bus x character spaces across the screen and y character spaces down.

car(x,y) and para(x,y) are similar to bus(x,y)

hide_cursor makes the cursor invisible. Use hide_cursor *after* a MODE command.

show_cursor makes the cursor visible again.

4 Running the program

The bus should move quickly across the screen, from left to right.

As is usual when printing shapes quickly, the cursor may flash from time to time. If you find this annoying, use hide_cursor immediately after turning to mode 1; and show_cursor at the end of the program.

5 Testing the program

It is important to check that the bus is printed in each position. To do this you need to slow the bus down, and make the computer print the value of the control variable each time round the loop. You could add these lines to your program:

```
122 PRINT TAB(0,0); across
123 pause(1)
```

Line 123 makes the computer pause so that you can see the bus and read its position each time round the loop.

When you are satisfied that the bus is being printed properly, you can remove these lines.

For you to do

Plan and write programs to do the following jobs below. They all use demons in your Graphics and Sounds Toolbox. The demons are loaded by selecting these options on the menu:

bus and car   parachutist

1 car(x,y) prints a car which is 4 characters wide and one character high. Make the computer move this car across the top line of the screen, from left to right.

2 para(x,y) prints a parachutist who is 1 character wide and 4 characters high. Make the computer move this parachutist down the middle of the screen from top to bottom.

3 Make the computer move a car and a bus across the screen at the same time.

2.11 PASSWORDS

Cash dispensers are useful machines. To obtain cash, first you put your bank card in a slot. Then the machine asks you for your Personal Identity Number (or PIN for short). If you type the correct number, the machine will let you withdraw cash. It's a good system because, without the secret number, no one else can use your card.

1 REPEAT...UNTIL loops

When you put a card in a cash dispenser, the dispenser works through a set of instructions like this:

```
REPEAT
  ask for the Personal Identity Number
UNTIL the correct number is typed
```

REPEAT and UNTIL are COMAL keywords. In a program, you can use them to create loops. The computer will go on repeating the loop until a certain condition is true.

For example:
you want a message to appear on the screen, but only when the correct password, **MAYFLY**, is typed in. The computer needs to follow instructions like this:

```
REPEAT
  ask for a word
UNTIL the word is MAYFLY
then print message
```

The condition in this case is that the word is **MAYFLY**. Once the condition is true, the computer stops repeating the loop and the message is printed.

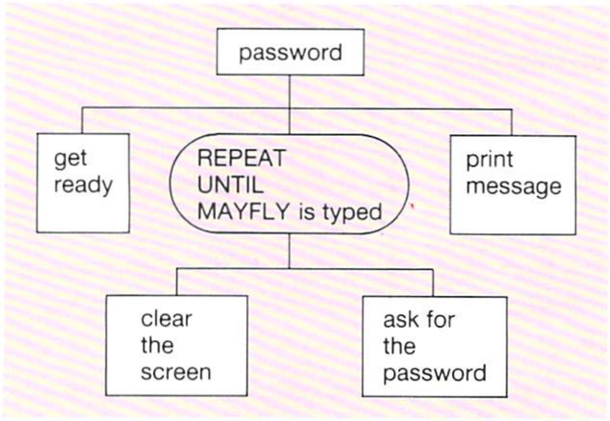
2 Asking for a password – planning the program

The job of the program is to repeatedly ask for a password until the correct one is typed. Then a message must be printed.

Here is one possible plan:

```
get ready
repeat
  ask for password
until MAYFLY is typed
print a message
```

The structure diagram would look like this:



In the structure diagram:

You can only start a box when the box on its left has been completed.

A box isn't completed until its lower branches have all been completed.

HELP

- To *delete* a line from your program: type the line number and press RETURN
- To *change* a line: type the line number and the new statement. Then press RETURN

3 Writing the program

Add a few comment statements at the start. Then type in the rest of the program:

```
100 MODE := 1
110 REPEAT
120   CLS
130   PRINT "Please type password"
140   INPUT password$
150 UNTIL password$ = "MAYFLY"
160 PRINT "You have clearance"
999 END
```

At line 140, the word you type in is assigned to the variable password$.

The loop starts at line 110.
It begins with a REPEAT statement.
It ends with UNTIL and a condition.
The condition in this program is

password$ = "MAYFLY"

Every time the computer reaches the end of the REPEAT...UNTIL loop, it tests the condition. If the condition is false, the computer goes back to the beginning of the loop, so the instructions in the loop will be repeated until you type in MAYFLY.

4 Running the program

Once the computer has begun the loop, it will repeat the instructions inside the loop until the statement

password$ = "MAYFLY"

is true.

Only then will the rest of the program be carried out.

- To *list* the main program: type LIST,999 RETURN
- To *clear* the screen: type CLS RETURN

5 Testing the program

Trace the path through the program. The instructions in the REPEAT...UNTIL loop should be carried out at least once.

Test the condition after UNTIL, by typing the wrong password. The computer should repeat the instructions in the loop. Then it should clear the screen and ask for the password again.

Try typing Mayfly or mayfly. These are nearly correct. But they aren't exactly right. So the computer should carry on repeating the loop.

Then type in MAYFLY. The computer should stop repeating the loop and go on to the rest of the program.

Check that the program prints the correct message.

For you to do

Plan and write programs to solve these problems:

1 Repeatedly ask for a password, until your first name is typed, then print a message.

2 Repeatedly ask for a code number, until 1476 is entered, then make a sound.

(You may find the procedure note(pitch) useful. This procedure is in the Graphics and Sound Toolkit.)

3 Repeatedly ask for a code number, until the number 67 is typed. Then make a car move across the screen.

(You may find the procedure car(x,y) useful. This procedure is in the Graphics and Sound Toolkit.)

4 Rewrite the quiz program on page 43 so that a REPEAT...UNTIL loop is included. The computer should repeatedly ask for each capital until the correct answer is given.

2.12 HIGHEST AND LOWEST

The picture shows a classful of people. But who is the tallest, and who is the shortest? Type their heights on the keyboard and the computer can work it out for you.

1 Making space for a list

Whenever the computer has to deal with a list of words or numbers, you have to tell it to reserve space in its memory.

Do this at the start of the program, by using the keyword `DIM`. Follow this with the name of the list and the maximum size of the list.
For example:

```
DIM heights(30)
```

tells the computer to reserve space for a list called `heights`, with up to 30 numbers in it.

2 Finding the highest and lowest

There are two ready-made demons for this:

`max(nos(),qty)`	finds the highest number in a list
`min(nos(),qty)`	finds the lowest number in a list

Each demon uses two parameters: the name of the list, and the number of items in the list. Each demon is a **function** because it works out a result for you. For example

```
min(heights(),30)
```

will look for a list called `heights` with 30 numbers in it, and find the smallest number.

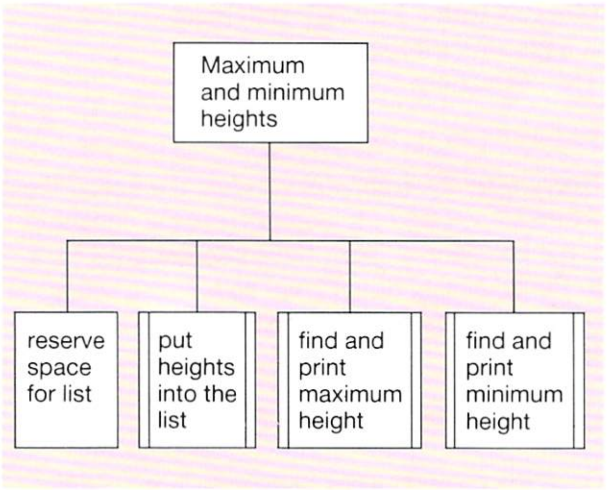
3 Tallest and shortest – planning the program

In a list of heights, the tallest person has the maximum (highest) height. The shortest person has the minimum (lowest) height.

Here's one way to find the highest and lowest heights in a list:

```
    reserve space for the list
    put heights into the list
    find and print the maximum height
    find and print the minimum height
```

Here's the structure diagram:


Maximum and minimum heights
reserve space for list
put heights into the list
find and print maximum height
find and print minimum height

HELP

SHIFT BREAK loads the demons menu.

f0 lists your program.

f4 lists the names of the ready-made demons. There isn't space on the screen for the full list. Press SHIFT to see more of it.

- To *clear* the screen: press `CLS` RETURN
- To *delete* a line from your program: type the line number and press RETURN
- To *change* a line: type the line number and the new line. Then press RETURN

4 Writing the program

First, load the `max` and `min` demons from your disc. You will find them in `Words and Numbers - Level 2`.

Start your program with a few comment statements. Then type in the rest of the program:

```
100  DIM heights(30)
110  MODE := 6
120  take_in_numbers(heights(),30)
130  PRINT "The tallest measures ";
140  PRINT max(heights(),30);"cm"
150  PRINT "The shortest measures ";
160  PRINT min(heights(),30);"cm"
999  END
```

`DIM heights(30)` makes the computer reserve space for a list of up to 30 heights.

`take_in_numbers` will ask for each number in the list, and store it.

This demon takes two parameters; the name of the list and the number of items in the list.

The functions `max` and `min` are used to find the maximum and minimum heights. The brackets are put after `heights`, like this:

```
heights()
```

to tell the computer that `heights` is a list.

`DIM size(10)` makes the computer reserve space in its memory for a list called `size` with up to 10 numbers in it.

Once space has been reserved in the memory, these demons can be used:

`take_in_numbers(size(),10)` asks for 10 numbers and stores them in the list called `size`.

`max(size(),10)` finds the largest of the first 10 numbers in the list called `size`.

`min(size(),10)` finds the smallest of the first 10 numbers in the list called `size`.

5 Running the program

First, the computer reserves space for a list called `heights`. Then, it takes in numbers from the keyboard and stores them in the list.

When the computer is told to

```
PRINT max(heights(),30)
```

`max(heights(),30)` is replaced by the highest number in the list of heights.

When the computer is told to

```
PRINT min(heights(),30)
```

`min(heights(),30)` is replaced by the lowest number in the list of heights.

6 Testing the program

Check that the computer correctly finds the maximum and minimum heights. If you don't agree with the computer, make it print the full list of heights. Then check for typing errors. You can use the demon:

```
display_numbers(heights(),30)
```

for this job. Just type in its name when the program has finished running. The demon is being used in immediate mode, so it doesn't need a line number.

For you to do

Plan and write programs to solve these problems:

1 Use the program on this page to work out the maximum and minimum heights in any group. Remember to change the program to suit the number of people on your list.

2 Put the ages (in months) of the people in your class into a list. Then find the ages of the youngest and oldest people.

3 Measure the hand span of each of your friends. Then use the computer to find the widest and narrowest spans.

S Summary of Part 2

If you have completed all the work so far, you should know how to:	*Covered in section:*
use different modes to display information in different styles	2.1
make your programs change the colours used on the screen	2.2
pass a parameter which is a string	2.2
use `,;` and `TAB` to position printed messages on the screen	2.3
pass several parameters to a procedure	2.4
use a function	2.5
store information in a program, using `READ` and `DATA` statements	2.6
make the computer do simple arithmetic	2.7 and 2.8
use an `IF...THEN...ELSE` statement to make a decision	2.9
use a decision box in a structure diagram	2.9
use a loop control variable	2.10
use a `REPEAT...UNTIL` loop	2.11
set aside space for a list, using `DIM`	2.12

Useful line numbers

You should be able to use these line numbers in your programs correctly:

`10 to 99`	comment statements
`100 to 998`	main program
`999`	`END`
`1000 to 6999`	procedures you type in yourself
`7000 to 29999`	leave free for any ready-made demons
`30000 to 32767`	information stored in `DATA` statements

HELP

You should be able to use these keys for loading and listing:

SHIFT BREAK loads the demons menu from disc. Select the set of demons you want by typing a number from the menu and pressing RETURN.

If you select `Toolbox` from the main menu, the Toolbox menu should appear.
Load the demons you want by selecting their numbers on the Toolbox menu.
Choose `0` when you have finished your selection.

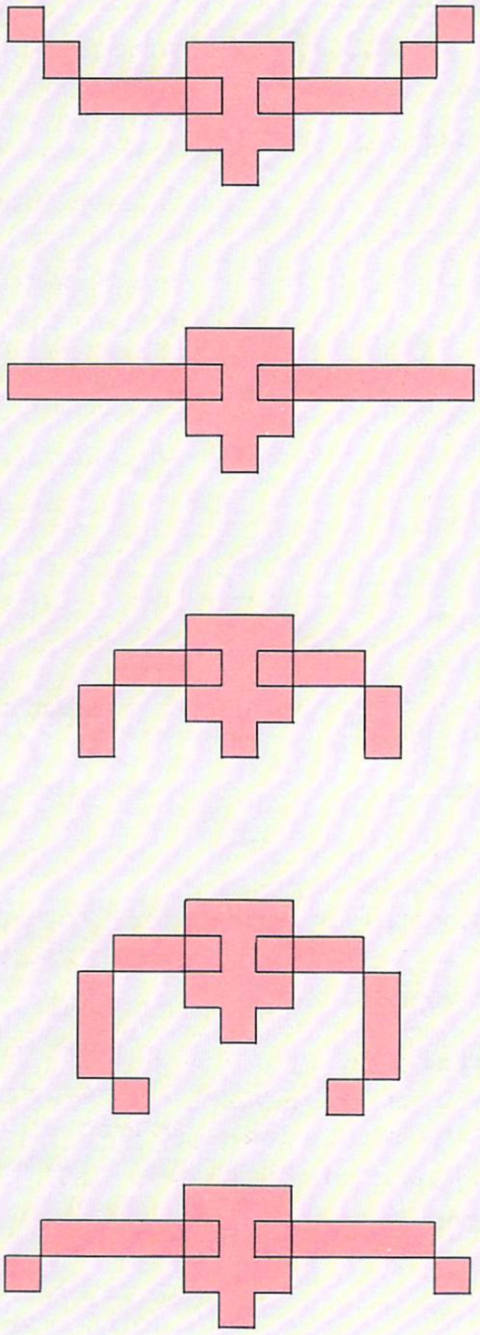
f0 lists the main program (lines with numbers up to `999`).

f2 lists any procedures you typed yourself.

f4 lists the names of all the demons loaded from disc.

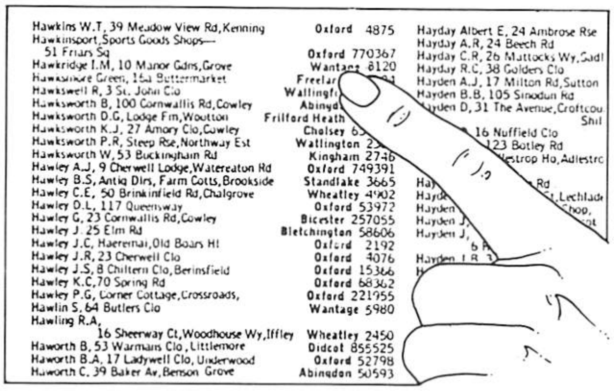
f8 takes you back to the menu, if the demons disc is in the drive.

FOR COMAL ENTHUSIASTS



You can make a bird appear to fly by printing these shapes, in the right order, in unit 3.5.

3.1 MAKING A 'PHONE BOOK



Hawkins W.T, 39 Meadow View Rd,Kenning Oxford 4875
Hawkinsport,Sports Goods Shops—
51 Friars Sq Oxford 770367
Hawkridge I.M, 10 Manor Gdns,Grove Wantage 8120
Hawksmoore Green, 16a Buttermarket Freelar
Hawkswell R, 3 St. John Clo Wallingf
Hawksworth B, 100 Cornwallis Rd,Cowley Abingd
Hawksworth D.G, Lodge Fm,Wootton Frilford Heath
Hawksworth K.J, 27 Amory Clo,Cowley Cholsey 6
Hawksworth P.R, Steep Rse,Northway Est Wallington 2
Hawksworth W, 53 Buckingham Rd Kingham 2746
Hawley A.J, 9 Cherwell Lodge,Watereaton Rd Oxford 749391
Hawley B.S, Antiq Dlrs, Farm Cotts,Brookside Standlake 3665
Hawley C.E, 50 Brinkinfield Rd,Chalgrove Wheatley 4902
Hawley D.L, 117 Queensway Oxford 53972
Hawley G, 23 Cornwallis Rd,Cowley Bicester 257055
Hawley J. 25 Elm Rd Bletchington 58606
Hawley J.C, Haeremai,Old Boars Hl Oxford 2192
Hawley J.R, 23 Cherwell Clo Oxford 4076
Hawley J.S, 8 Chiltern Clo,Berinsfield Oxford 15366
Hawley K.C,70 Spring Rd Oxford 68362
Hawley P.G, Corner Cottage,Crossroads, Oxford 221955
Hawlin S, 64 Butlers Clo Wantage 5980
Hawling R.A,
16 Sheerway Ct,Woodhouse Wy,Iffley Wheatley 2450
Haworth B, 53 Warmans Clo ,Littlemore Didcot 855525
Haworth B.A, 17 Ladywell Clo, Underwood Oxford 52798
Haworth C, 39 Baker Av,Benson Grove Abingdon 50593

Hayday Albert E, 24 Ambrose Rse
Hayday A.R, 24 Beech Rd
Hayday C.R, 26 Mattocks Wy,Sadl
Hayday R.C, 38 Golders Clo
Hayden A.J, 17 Milton Rd,Sutton
Hayden B.B, 105 Sinodun Rd
Hayden D, 31 The Avenue,Croftcou
Shil
16 Nuffield Clo
123 Botley Rd
estrop Ho,Adlestrc
Ha Rd
Hayd t,Lechlade
Hayden hop,
Hayden J, nt
Hayden J,
6
Hayden L.B 3
H

Here are just a few of the 200 000 or so entries in a typical 'phone book. In fact, there are two **connected lists** in the book: names with addresses, and telephone numbers.

1 Sorting connected lists

You can use the computer to make a personal 'phone book. You type in a list like this:

```
HUGHES          835256
DAVIES          924127
PATEL           732994
SPYROS          655428
CARR            538962
```

The computer can sort the names into alphabetical order. Then it can rearrange the 'phone numbers so that they reappear opposite the correct names like this:

```
CARR            538962
DAVIES          924127
HUGHES          835256
PATEL           732994
SPYROS          655428
```

There are some useful demons for handling connected lists. Here are their names:

`take_in_words_and_numbers`

`sort_words_and_numbers`

`display_words_and numbers`

The name of each demon must be followed by three parameters. The **HELP** box tells you more.

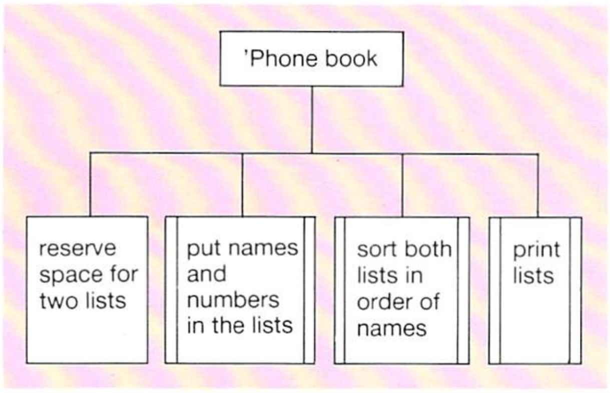
2 Making a 'phone book – planning the program

The computer can be used to store lists of names and 'phone numbers. Pairs of names and numbers can be typed, in any order, and the computer can quickly put them in order.

Here's how to create lists of names and telephone numbers:

reserve space for two lists
put names and numbers in the lists
sort both lists, in order of names
print the lists

The structure diagram looks like this:


'Phone book
reserve space for two lists
put names and numbers in the lists
sort both lists in order of names
print lists

HELP

`DIM names$(5), numbers(5)` makes the computer reserve space for two lists:

- a list called `name$`, holding up to 5 words (the '$' tells the computer to expect words and not numbers).
- a list called `numbers`, holding up to 5 numbers.

Once space has been reserved in the memory, the demons on the right can be used. Each demon uses three parameters:

the name of the list of words,
the name of the list of numbers,
the number of pairs of words and numbers.

3 Writing the program

Start with 5 names and 'phone numbers only. First, load the demons in `Words and Numbers – level 2`. Then type in the program, adding a few comment statements at the start.

```
100 DIM names$(5),numbers(5)
110 MODE := 6
120 take_in_words_and_numbers(names$(),numbers(),5)
130 sort_words_and_numbers(names$(), numbers(),5)
140 CLS
150 PRINT "Names and numbers"
160 PRINT
170 display_words_and_numbers(names$(),numbers(),5)
999 END
```

4 Running the program

The `DIM` statement reserves space for up to 5 names and 5 'phone numbers.

`take_in_words_and_numbers`, at line `120`, asks for 5 pairs of words and numbers. You type in 5 names and 'phone numbers. The computer puts them in two lists, called `names$` and `numbers`.

You won't see anything on the screen while the lists are being sorted, but the demon `display_words_and_numbers` prints the names and numbers opposite each other.

`take_in_words_and_numbers(names$(), numbers(),5)` will ask for 5 pairs of words and numbers. It will put the words in a list called `name$`, and the numbers in a list called `numbers`.

`sort_words_and_numbers(names$(), numbers(), 5)` will put the first 5 words in `name$` in alphabetical order. Then it will rearrange the list of numbers so that they are in the same order as the words.

`display_words_and_numbers(names$(), numbers(),5)` will print the first 5 words and numbers from `name$` and `number$`.

5 Testing the program

First, write down the lists of names and telephone numbers which you will use.

Run the program, and type the same names and telephone numbers as are on your list.

Rearrange the lists you wrote down without using the computer. Compare the result with the lists that the computer prints. If there is a difference, either you or the computer hasn't done the job properly!

For you to do

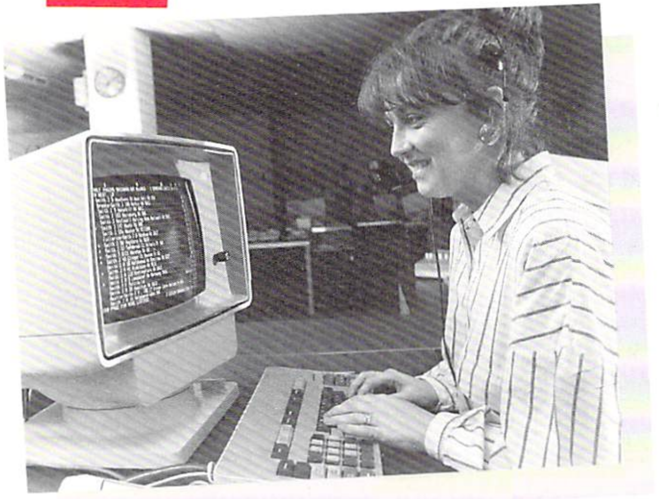
Plan and write programs to solve these problems:

1 Alter the 'phone book program so that it does the same job for 12 names and telephone numbers.

2 Write down the names and heights of 10 people you know. Then use the computer to print lists of names and heights, in alphabetical order of their names.

3 Choose 15 people and find out how long each took to travel to school this morning. Use the computer to print a table of results, with the names in alphabetical order.

3.2 USING THE 'PHONE BOOK



Dial 192 for Directory Enquiries and within seconds they can find you any 'phone number in the country. It happens very quickly because a computer does the searching.

1 Searching lists

The computer 'phone book on page 52 stores two lists – a list of names, and a list of numbers in the same order:

HUGHES	835256
DAVIES	924127
PATEL	732994
SPYROS	655428
CARR	538962

To find the telephone number of PATEL

```
go down the list of names until it finds the name PATEL
count how many places down the list the name is (3)
go down the list of numbers to the same position (3)
read off the number (732994)
```

You can use the ready-made demon:

`search_for_word`

to find the position of a name in a list. This demon will search for a list, find a particular word, then give you the position of the word down the list (3, in the case of PATEL).

This demon is a function because it works out a result for you (a number). It uses three parameters. See the **HELP** box for details.

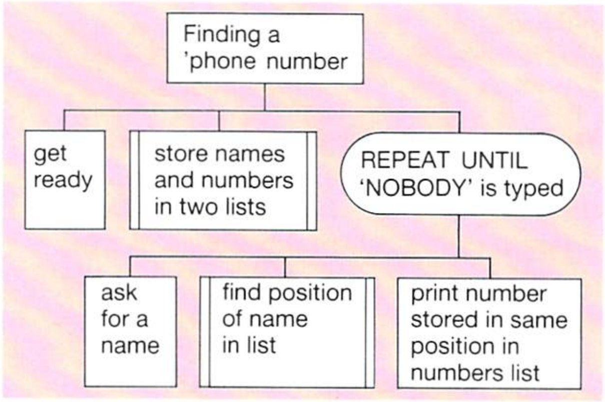
2 Finding a 'phone number – planning the program

The plan below is for a program which asks for names and 'phone numbers, stores them in the computer, and finds the 'phone number of any name you choose.

The REPEAT...UNTIL loop means that you can find several numbers, one after another.

```
get ready
store the names and numbers in two lists
REPEAT
  ask for a name
  find position of name in list of names
  print 'phone number stored in same
    position in list of numbers
UNTIL the word NOBODY is typed
```

Here's the structure diagram:



Finding a 'phone number
get ready
store names and numbers in two lists
REPEAT UNTIL 'NOBODY' is typed
ask for a name
find position of name in list
print number stored in same position in numbers list

HELP

search_for_word(names$(),person$,5) will look for a list called name$ with 5 words on it, look down the list for the word which person$ stands for PATEL, for example) and work out the position of this word on the list (3 for example)

PRINT names$(3) will make the computer print the 3rd word down in the list called names$.

3 Writing the program

First, load the Words and Numbers - level 2 demons. Then type in the rest of the program:

```
100 DIM names$(5),numbers(5)
110 take_in_words_and_numbers(names$(),numbers(),5)
120 CLS
130 REPEAT
140   PRINT "Enter the name ";
150   INPUT person$
160   position = search_for_word(names$(),person$,5)
170   PRINT names$(position), numbers(position)
180 UNTIL person$ = "NOBODY"
999 END
```

4 Running the program

The DIM statement reserves space for two lists, holding up to 5 words and numbers. Then take_in_words_and_numbers fills the lists with the names and 'phone numbers you type.

When you want to find a 'phone number, INPUT person$ lets you put a name into the computer. This name then replaces person$ in the program.

search_for_word finds the name in the list of names. It works out the position of the name, (3 for example). This number then replaces position in the rest of the program.

The PRINT statement makes the computer print a word and a number. If, say, position has the value 3, the computer will print the 3rd name down and the 3rd number down.

search_for_number(numbers(), phone, 5) will look for a list called numbers with 5 items on it, look down the list for the number which the variable phone stands for and work out the position of this number.

PRINT numbers(3) will make the computer print the 3rd number down in the list called numbers.

5 Testing the program

Take a note of the names and numbers before you type them in. Ask the computer to find a 'phone number. Then check that it finds and prints the right one.

Check that the computer goes on finding numbers correctly. The loop should continue until you type in the word NOBODY.

For you to do

Plan and write programs to solve these problems:

1 Change the program on this page so that the computer :

stores 10 names and 'phone numbers;

prints a suitable message if you try to find a number which hasn't been stored.

2 Use these demons from Words and Numbers - level 2:

take_in_words_and_numbers
search_for_number

to help create a program in which the computer asks for a phone number, then prints the name of the person with that number.

3.3 GOAL!

Celtic
Rangers
Aberdeen
Dundee Utd
Hearts
Dundee
St Mirren
Hibernian
Motherwell
Falkirk
Clydebank
Hamilton

This is how twelve Premier Division teams stood in the points order one weekend in February 1987.

The computer can sort these teams into alphabetical order. But first you have to reserve space for a list in the memory. And then the list has to be filled. . .

1 Reserving space

```
DIM teams$(12)
```

tells the computer to reserve space for a list called `team$` which can hold up to 12 words. In this list,

`team$(1)`	stands for the 1st word
`team$(2)`	stands for the 2nd word
`team$(3)`	stands for the 3rd word and so on, up to
`team$(12)`	which stands for the 12th word

2 Filling the list

To fill a list, each item in it has to be given a value. For example

```
team$(3) := "Aberdeen"
```

makes the 3rd word in the list `Aberdeen`. And so on.

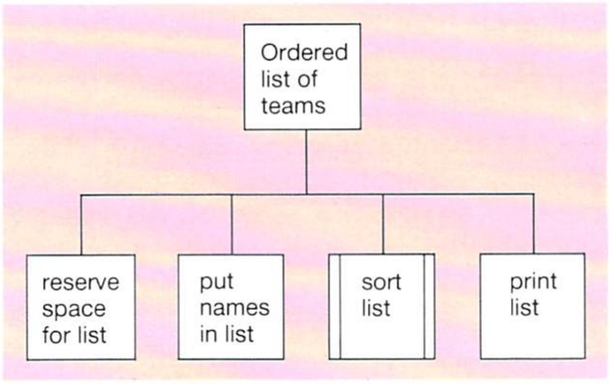
To the computer, names like `Dundee Utd` count as single words.

3 Ordering the teams – planning the program

Here's how to put 12 teams in a list, sort them into alphabetical order, then print them out.

```
reserve space for the list
put the names in the list
sort the list
print the list
```

Here's the structure diagram:



HELP

SHIFT BREAK loads the demons menu.

f0 lists your program.

f4 lists the names of the ready-made demons you can use. There isn't room on the screen for the full list. Press SHIFT to see more.

- To *delete* a line:
  type the line number and press RETURN

New demon

```
sort_words(team$(), 12)
```

will search for a list called `team$` with 12 items in it. Then it will sort the list so that all the items are in alphabetical order.

Each item, `team$(1)`, `team$(2)`, `team$(3)`.. and so on, will therefore be given a new value.

4 Writing the program

First, load the `Words and Numbers – Level 2` demons. Then type in the rest of the program:

```
  100 MODE := 6
  110 DIM team$(12)
  120 FOR position := 1 TO 12 DO
  130   READ team$(position)
  140 NEXT position
  150 sort_words(team$(),12)
  160 FOR position := 1 TO 12 DO
  170   PRINT team$(position)
  180 NEXT position
  999 END
30000 DATA "Celtic", "Rangers", "Aberdeen"
30010 DATA "Dundee Utd", "Hearts", "Dundee"
30020 DATA "St Mirren", "Hibernian"
30030 DATA "Motherwell", "Falkirk"
30040 DATA "Clydebank", "Hamilton"
```

The `DIM` statement reserves space for the list.

The `FOR...NEXT` loop `READ`s the names from `DATA` and puts them in the list. As the control variable `position` changes its value to 1, 2, 3 and so on, the list is filled with the different names from `DATA`:

```
team$(1) = "Celtic"
team$(2) = "Rangers"
team$(3) = "Aberdeen"
```
...and so on

`sort_words` is a demon which sorts the list, in alphabetical order. See the **HELP** box for details.

When the list is in alphabetical order, another `FOR...NEXT` loop is used to print it. As the control variable `position` changes its value to 1, 2, 3 and so on, the different items on the list are printed. For example:

when `position = 3`, the computer obeys the instruction

```
PRINT team$(3)
```

It prints the word `Clydebank`, because this is the 3rd item down in the new alphabetical list created by the demon `sort_words`.

5 Running the program

The program doesn't print anything while it is filling and sorting the list. Unless you have added a title or some instructions, the only information to appear on the screen will be the names of the teams, in alphabetical order. The names of the teams should each be printed at the start of a new line. That's because there is no comma or semicolon at the end of the `PRINT` statement in line `170`.

6 Testing the program

Keep a written list of the names you type in the `DATA` statements.

You can check the list the computer stores, by adding this line to your program:

```
145 display_words(team$(),12)
```

This will make the computer display the original list of teams. You can remove the line later.

Put the teams in alphabetical order without using the computer. Then check that the computer sorts and prints the names in the correct order.

For you to do

Plan and write programs to make the computer do the following:

1 create a list of 10 names of pop groups
  print the list

2 create a list of 10 record titles
  print the list
  sort the list
  print the titles in alphabetical order

3 create a list of groups and a list of record titles
  print both lists, side by side
  ask for the name of a group
  find the position of the group in the list
  print the title of their record

  (Do this without using demons, if you can.)

3.4 KEEP IT SECRET

CDGTFGGP
EGNVKE
JGCTVU
HCNMKTM
JCOKNVQP
FWPFGG
TCPIGTU
OQVJGTYGNN
JKDGTPKCP

This list is in code. But what are the words? Work through the next two pages and you should be able to solve the mystery.

Putting information into code is one way of stopping unauthorised people from reading it. The computer can do this job because it already uses one code itself.

1 The ASCII Code

Inside the computer, the letters and numbers you type at the keyboard are often stored using ASCII code numbers. ASCII stands for **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange.

You can find the ASCII code number of any of the symbols on the keyboard by using the built-in COMAL function `ORD`. For example, if you give the computer this command:

```
PRINT ORD("A")
```

the computer will print the number 65. 65 is the ASCII code number for `A`.

Try the command on other letters of the alphabet to find out their ASCII code numbers.

2 Letters from code numbers

The built-in COMAL function `CHR$` does the opposite job to `ORD`. It finds the letter or symbol given by any ASCII code number. For example, if you give the computer this command:

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

The computer will print the letter `A`, because 65 is the ASCII code number for `A`.

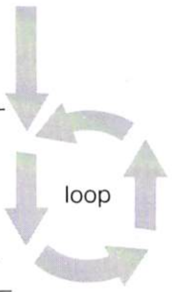
3 Coding a word – planning the program

To change a whole word into a coded word, you could make the computer:

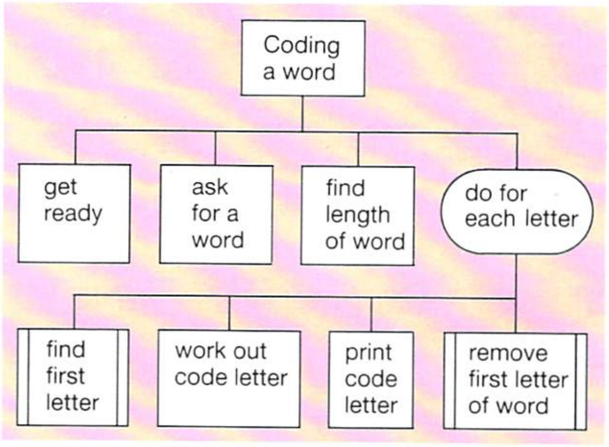
take each letter in turn
use `ORD` to find its ASCII code number
add a number (say 4) to the ASCII code number
use `CHR$` to turn this new number back into a letter

Here is a program plan for putting a whole word into code and printing the result:

```
get ready
ask for a word
find its length
for each letter — — — — — — —
   find first letter in word
   work out code letter
   print code letter
   remove first letter from word
end loop — — — — — — — — — —
```



The structure diagram looks like this:



HELP

If `w$= "GLASGOW"`

`all_but_first_letter(w$)` removes the first letter from the word which `w$` stands for.
Result: `LASGOW`

4 Writing the program

First, load the `Words and Numbers – level 2` demons. Then type in the main program:

```
100 MODE := 6
110 PRINT "Please type your word ";
120 INPUT w$
130 PRINT "The coded word is ";
140 length := LEN (w$)
150 FOR letter := 1 TO length DO
160   first$ := first_letter$(w$)
170   number := ORD (first$)
180   newnumber := number + 4
190   newletter$ := CHR$ (newnumber)
200   PRINT newletter$;
210   w$ := all_but_first_letter$(w$)
220 NEXT letter
999 END
```

`INPUT` takes your word from the keyboard and assigns it to the variable `w$`. There is a `$` at the end of this variable name because it stands for a word.

`LEN` is a built-in COMAL function. It finds the number of letters in your word.

The `FOR...NEXT` loop goes round once for each letter in your word.

The demon `first_letter$` finds the first letter of your word.

`ORD` finds the ASCII code number of this letter. The program adds 4 to this number. Then `CHR$` turns the new number back into a different letter.

The demon `all_but_first_letter$` removes the first letter from your word before the loop is repeated. Now the computer gets to work on the next letter... and so on, until the whole word has been coded.

`LEN(w$)` finds the number of letters in the word which `w$` stands for. Result: `7`

`first_letter$(w$)` finds the first letter of the word which `w$` stands for. Result: `G`

5 Running the program

Type a word or phrase when the computer tells you to. The computer will then print the coded letters, one by one.

While it was coding the message, the computer should have removed all the letters in the word or phrase you typed. By the end of the program, `w$` should have no letters left in it at all. Make sure this has happened, by asking the computer to: `PRINT w$`

6 Testing the program

A good way to test this program is to carry out a dry run. A table of ASCII code numbers is useful. You should find one in your computer manual.

When you have carried out the dry run, run the program and enter the word to be coded. Carefully compare your answer with the computer's result.

For you to do

Plan and write programs to solve these problems:

1 Decode a word, by changing the program above so that 4 is *subtracted* from the ASCII code number of every letter.

2 Change a word typed in capitals, into the same word typed in lower case (small) letters. You can do this by adding 32 to the ASCII code number of every capital letter.

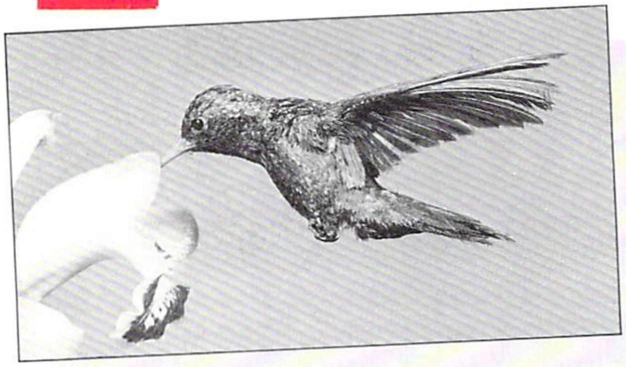
3 Print the ASCII code of every capital letter.

4 Decode this message: **aovv Nyxo6**

5 Solve the mystery of the list at the top of the opposite page.

Clues: the words are all in the same code, the code is similar to the one used on this page, but the number added to each ASCII code number is *not* 4.

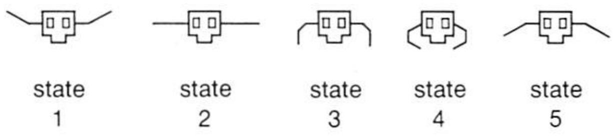
3.5 FLYING



This humming bird moves its wings so fast, you can't see them, unless you take a photograph. The bird on your demons disc is slower. But it can flap its wings and fly – provided you give it the right instructions.

1 Flapping

`bird` is a demon which draws a bird on the screen. The bird's wings may be drawn in 5 different ways. These are **states** 1, 2, 3, 4 and 5:



`bird(10,15,1)` will draw the bird 10 printing positions across, 15 printing positions down, with its wings in state 1.

`bird(10,15,2)` will draw the bird in the same position, but with its wings in state 2.

The bird will appear to **flap** its wings once, if it is drawn in the same *position* on the screen five times – but in a different *state* each time. This can be done using a loop.

2 Flying

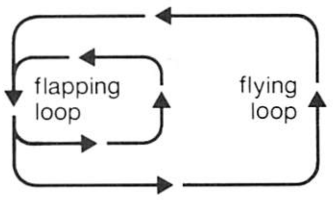
The bird will appear to **fly** if it makes lots of *whole* flaps (say 50 or so). This can be done using another loop.

The loop for flying must have the loop for flapping inside it. Writing one loop inside another like this is called **nesting**.

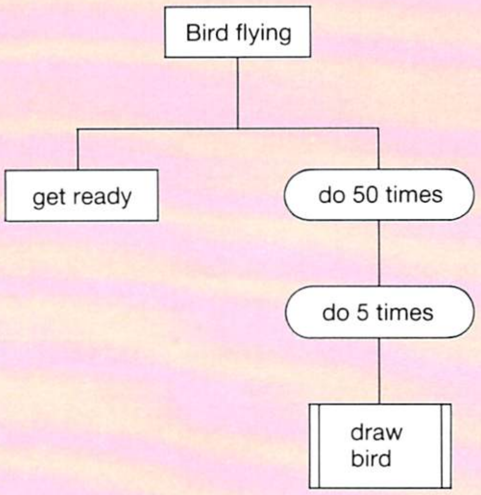
3 A bird flying – planning the program

Here's one way to make the bird fly:

```
get ready
do 50 times
   do 5 times
      draw bird
   end loop
end loop
```



The structure diagram looks like this:



HELP

To *load* demons from the toolbox:

Use SHIFT BREAK to load the demons menu, then choose the toolbox.

Load the demons you want by selecting their numbers on the menu.

Choose option 0 when you have finished your selection.

Remember to load all the demons you need before typing in your program.

4 Writing the program

First, load the `bird` demon from your `Graphics and Sound Toolbox`. Do this by selecting the `bird` option on the main menu. Select these options as well:

```
hide and show cursor
pause
```

You may want to use them later.

Start your program with a few comment statements. Then type in the main program:

```
100 MODE := 2
110 FOR flap := 1 TO 50 DO
120   FOR state := 1 TO 5 DO
130     bird(10,10,state)
140   NEXT state
150 NEXT flap
999 END
```

In the inner loop, the control variable `state` selects the way in which the bird is drawn. This loop is completely contained inside the flying loop.

5 Running the program

The mode used will affect the size of the bird, and its position on the screen.

When the bird is drawn, the cursor may flash occasionally. If this is annoying, use the demon `hide_cursor` at the beginning of the program, after the `MODE` statement. Then use `show_cursor` at the end.

6 Testing the program

To see if the bird flaps, you need to slow the action down. Add this line to your program:

```
135 pause(1)
```

This makes the computer pause for 1 second after drawing the bird. Check that the wings are drawn in 5 positions during each flap. Then check that there are 50 complete flaps.

For you to do

Plan and write programs to solve the problems below. All the demons mentioned are in your `Graphics and Sounds Toolbox`. Find out how to use them in the **HELP** box.

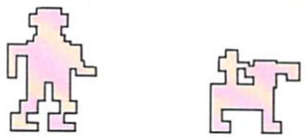
1 The demon `spaceship` draws a spaceship pointing in five possible directions.
The spaceship will rotate once if it is drawn in these five directions, one after another.
Make the spaceship spin by rotating 50 times.

2 The demon `man` draws a person in three possible poses. Make the person take a walking step by drawing these three poses one after another.
Make the person walk across the screen by letting one of the control variables select the position of the drawing on the screen.

3 Use the demon `horse` to make a horse canter across the screen.

`spaceship(10,15,1)` draws a small spaceship 10 printing positions across the screen, 15 printing positions down the screen. To point in different directions: use 2, 3, 4 or 5 instead of 1.

`horse(10,15,1)` draws a horse with rider 10 printing positions across the screen, 15 printing positions down the screen. For different leg angles, use 2 instead of 1. To 'rub out' the drawing, use 3 instead of 1.



`man(10,15,1)` draws a person 10 printing positions across the screen, 15 positions down.

Instead of 1, use 2 or 3 for different poses, or 4 to 'rub out' the drawing.

`pause (1)` makes the computer pause for 1 second. Higher numbers give a longer pause.

3.6 CARLA THE COOK

Carla sells cakes she bakes herself. She stores all her recipes in a small computer.

1 CASE

You can use `CASE ... OF WHEN` statements to make the computer select one part of a program and not others.

Say you wanted to know the ingredients for **ginger cake**. The program could work like this:

```
INPUT cake$ (Here, you type in GINGER CAKE)
CASE cake$ OF
  WHEN = "ORANGE CAKE"
            print ingredients for orange cake
  WHEN = "APPLE CAKE"
            print ingredients for apple cake
  WHEN = "GINGER CAKE"
            print ingredients for ginger cake
  OTHERWISE
            print Sorry, not available
END CASE
```

The variable `cake$` stands for whatever you type in. In this case, it's `GINGER CAKE`.

The first `WHEN` statement doesn't match this. The second `WHEN` statement doesn't either. So the instructions to print the ingredients for orange cake and apple cake are ignored.

The third `WHEN` statement *does* match. So the instructions to print the ingredients for ginger cake are carried out.

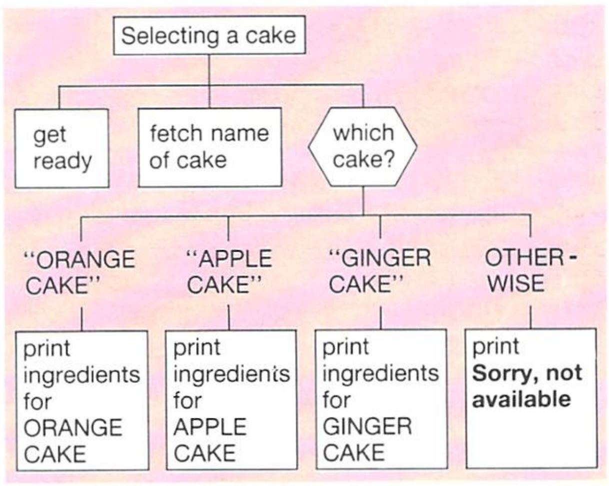
If none of the `WHEN` statements matched, the `OTHERWISE` instructions would be carried out.

2 Selecting a cake – planning

Here is a program plan for making the computer select a list of ingredients:

```
get ready
fetch the name of the cake from the keyboard
select and print the list of ingredients
if none of the selections is chosen, print a message
```

Here's the structure diagram:


Selecting a cake
get ready
fetch name of cake
which cake?
"ORANGE CAKE"
"APPLE CAKE"
"GINGER CAKE"
OTHERWISE
print ingredients for ORANGE CAKE
print ingredients for APPLE CAKE
print ingredients for GINGER CAKE
print Sorry, not available

A selection is shown by using a decision box with several exits.

HELP

- If there is a program in the computer already, type `NEW` RETURN before you start.
- *Start* your program with a few comment statements like this:

```
10// Cake ingredients
20// by Carla
```

- To *list* your program: type `LIST` RETURN
- To *delete* a line from your program: type the line number and press RETURN

3 Writing the program

No demons are needed. Just type the program straight in:

```
100 MODE := 0
110 PRINT "Type the name of the cake"
120 INPUT cake$
130 PRINT

140 CASE cake$ OF
150 WHEN = "ORANGE CAKE"
160   PRINT "150g flour, 100g butter"
170   PRINT "100g sugar, 2 eggs"
180   PRINT "2 tablsp. marmalade"
200 WHEN = "APPLE CAKE"
210   PRINT "200g flour, 100g butter"
220   PRINT "100g sugar, 1 egg"
230   PRINT "4 tablsp. milk"
240   PRINT "200g sliced apple"
250 WHEN = "GINGER CAKE"
260   PRINT "200g flour, 150g butter"
270   PRINT "100g brown sugar"
280   PRINT "2 eggs,4 tablsp. milk"
290   PRINT "1 teasp. ground ginger"
300 OTHERWISE
310   PRINT "Sorry, not available"
320 END CASE

330 PRINT "Program over"

999 END
```

Each of the `WHEN` statements is followed by a set of `PRINT` statements. These contain the ingredients in each recipe.

4 Running the program

When you type in the name of the cake, this replaces `cake$` in the program.

`CASE cake$ OF` tells the computer to make a selection. It selects whichever `WHEN` statement is followed by the words you typed for `cake$`. Then it prints out the correct ingredients.

If `cake$` isn't any of these names, the computer tells you that the information isn't available.

Once the selection process is over, the computer carries out the statement after `END CASE`.

5 Testing the program

To test the selection process, you must type the name of each cake at least once. Then you must type a name which isn't on the list.

Typing the correct names tests the `WHEN` statements.

Typing a name which isn't selected by a `WHEN` statement tests the `OTHERWISE` statement. This should make the computer print the apology.

For you to do

Plan and write programs to solve these problems:

1 Make the computer ask for the name of a food. The computer should print the maximum storage time for that food in the freezer.

Food	Storage time
bread	6 months
rolls	4 months
mushrooms	1 month
chicken	12 months
beef	8 months
sausages	2 months

2 Write a program which can be used as an electronic appointment book.

When you type in a day of the week, the computer should print a list of appointments for that day.

If you type in some other word, instead of a day, the computer should print

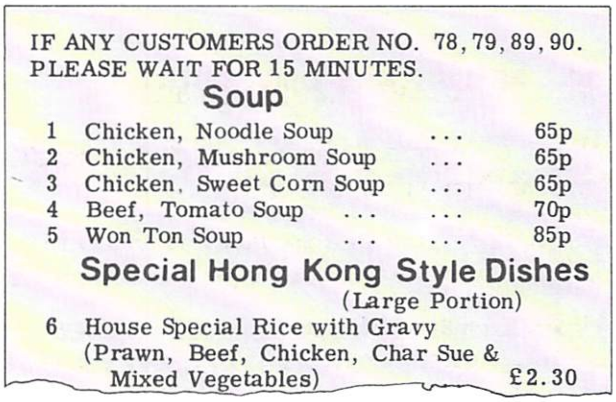
`That is not the name of a day!`

3 Write a program which will:

ask for a person's name
print their address and telephone number

Your program should hold the addresses of 4 people. If you type in a name which doesn't match any of those in the program, the computer should print a suitable message.

3.7 IT'S ON THE MENU



IF ANY CUSTOMERS ORDER NO. 78, 79, 89, 90.
PLEASE WAIT FOR 15 MINUTES.

Soup

1	Chicken, Noodle Soup	...	65p
2	Chicken, Mushroom Soup	...	65p
3	Chicken, Sweet Corn Soup	...	65p
4	Beef, Tomato Soup ...	...	70p
5	Won Ton Soup ...	...	85p

Special Hong Kong Style Dishes
(Large Portion)

6 House Special Rice with Gravy (Prawn, Beef, Chicken, Char Sue & Mixed Vegetables) £2.30

Fast-food restaurants often use menus with numbers on. You tell them the number of your choice of food. The restaurant gives you the food to match.

Computer menus work rather like this. You have already used them on your demons disc. Now try creating your own menu. Start with a menu which lets someone choose which type of animal they would like to see on the screen.

1 Creating a menu

Here's one way of creating a menu:

1 Show the choices.
2 Wait for a choice.
3 Check the choice is actually on the menu.
4 If the choice is on the menu, carry it out. Otherwise go back to **2**.

2 Setting out the screen

When a menu is printed on the screen: the choices should be clearly stated, there should be some instructions about how to make a choice:

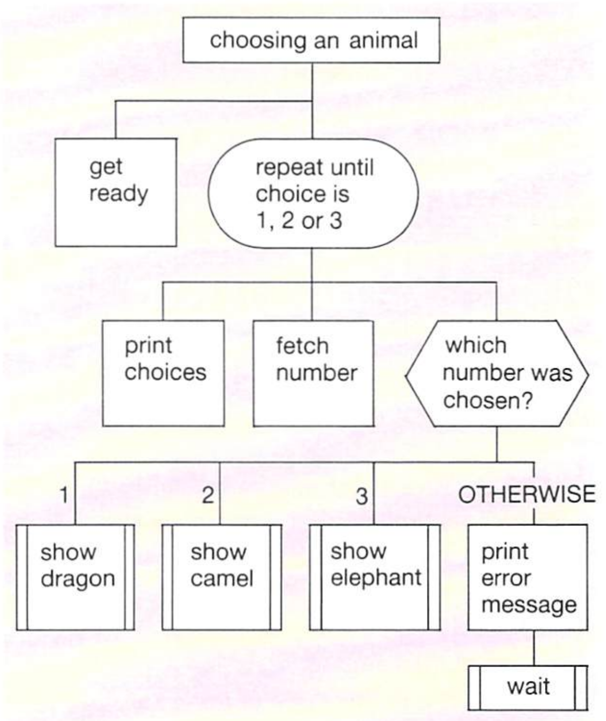
```
2 - camel
3 - elephant

Choose an animal by
typing the number then
pressing RETURN.
```

3 Choosing an animal – planning the program

Here's one way of using a menu to show a dragon, a camel or an elephant:

```
get ready
repeat
   print choices
   fetch choice
   if choice is 1, show dragon
   if choice is 2, show camel
   if choice is 3, show elephant
   otherwise print an error message
until choice is 1, 2 or 3.
```

The structure diagram looks like this:



HELP

- *Remember* to load all the demons you need before typing in your program.
- To *list* your own procedures: type `LIST 1000, 6999` RETURN
- To *clear* the screen: type `CLS` RETURN

4 Writing the program

First, load the demons you will need from your `Graphics and Sound Toolbox`. Do this by choosing these options on the menu:

```
dragon, camel and elephant    wait
centre a message
```

Start your program with a few comment statements. They type in the main program:

```
100 MODE := 1
110 REPEAT
120   show_choices
130   INPUT num
140   CASE num OF
150     WHEN = 1
160       show_dragon
170     WHEN = 2
180       show_camel
190     WHEN = 3
200       show_elephant
210     OTHERWISE
220       PRINT "That's not allowed"
230   END CASE
240   wait
250 UNTIL num=1 OR num=2 OR num= 3
999 END
```

Write these procedures yourself:

```
show_choices        show_dragon
show_camel          show_elephant
```

Use line numbers between `1000` and `6999` for this. For example:

```
1000 PROC show_dragon
1010 CLS
1020 dragon (18,8)
1030 centre("DRAGON",24)
1040 END PROC show_dragon
```

Your procedure `show_choices` should print out instructions like those on the opposite page. Start the procedure like this:

```
1300 PROC show_choices
1310 CLS
1320 PRINT "Instant animal"
```

.... and so on

Remember to end each procedure with an `END PROC` statement.

5 Running the program

When the program runs, a set of choices should appear. You type in 1, 2 or 3.

If you type in one of these numbers an animal should appear, with its name printed underneath.

If you type in some other number, a message should appear. The `REPEAT...UNTIL` loop gives you the chance to choose again.

6 Testing the program

There are several parts to this program, and each must be tested carefully.

First, make sure that the list of choices has been printed correctly.

Next, check that the `CASE` works properly. When a correct choice has been made, the right animal should be shown. This also tests the `UNTIL` statement, because the program should then end.

Finally, check that `CASE` works properly when an incorrect choice is made. This tests two parts of the program; the `OTHERWISE` statement, and the `REPEAT...UNTIL` loop. This time, the program should loop back to the `REPEAT` statement.

For you to do

Plan and write programs to solve these problems:

1 Complete the 'Choosing an animal' program.

2 Create a menu which can be used to choose and play a low, medium or high pitched note. You might find the procedure `note(pitch)` useful; see page 10.

3 Create a menu which allows you to select a program which the computer will run. Save your menu program before you run it, otherwise it will be lost!

3.8 FREIGHT WEIGHT

This Pan Am 747 can carry 20 000 kg of freight. During loading, a computer keeps track of the total weight in the hold. As each item is loaded, its weight is typed into the computer and added to the previous total. While the total is less than 20 000 kg, the computer accepts new items. But when the 20 000 kg limit is passed, the computer prints a warning.

1 WHILE

A `WHILE` loop can be used in the plane loading program. It works like this:

```
WHILE total weight is less than 20 000 kg DO
   add weight of next item to total
   INPUT weight of next item after that
   (Here, you type in a weight)
END WHILE
print Sorry, that's too heavy
```

The `WHILE ... DO` statement tests a **condition** – that the total weight is less than 20 000 kg. While this condition is TRUE, the loop repeats, carrying out the instructions inside it each time.

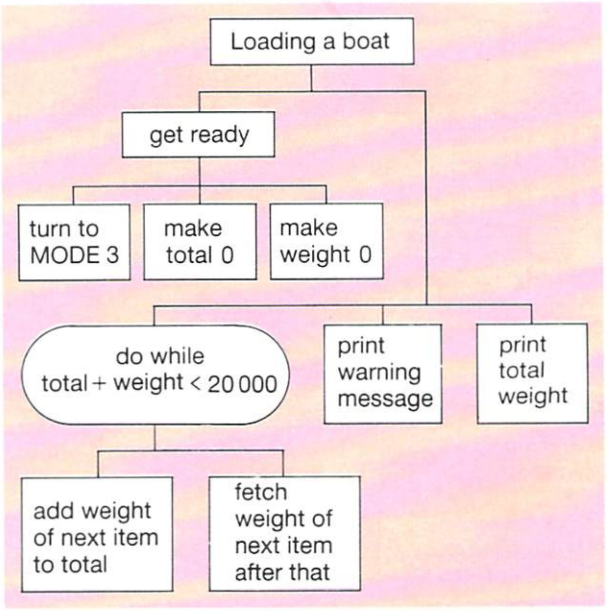
When the total weight is 20 000 kg or more, the condition becomes FALSE. Now, the computer won't carry out the instructions inside the loop. The loop ends, and the computer goes on to the next line.

2 Loading a plane – planning the program

Here's a plan for a program to keep track of the weight being loaded in the plane:

```
get ready
   turn to mode 3
   make total weight 0
   make weight of next item 0
while total weight + weight of next item <20 000
   add weight of next item to total
   fetch weight of next item after that
end loop
print warning message
print the total weight in the hold
```

Here's the structure diagram:



HELP

- `WHILE` loops must
  start with a `WHILE .... DO` statement
  finish with `END WHILE`
- < is the computer symbol for 'less than'
  > means 'greater than'
  <> means 'is not equal to'

3 Writing the program

No demons are needed. Just type the program straight in.

```
100 MODE := 3
110 total := 0
120 weight := 0

130 WHILE total + weight <> 20000 DO
140   total := total + weight
150   PRINT "Enter next weight"
160   INPUT weight
170 END WHILE

180 PRINT "Sorry, that's too heavy."
190 PRINT "The hold is full."
200 PRINT "Cargo weighs ";total;"kg."
999 END
```

Two variables are used in the program:

`total`	the total weight in kg
`weight`	the weight of the next item in kg

Both these variables are given zero values at the start of the program.

`INPUT` weight gives weight a new value each time round the loop.

Line `140` tells the computer to give the variable `total` a new value. It's the old value, plus the last weight typed in.

4 Running the program

While the computer is repeating the `WHILE` loop, it keeps asking for the weight of the next item.

The loop only repeats while `total + weight` is *less* than 20 000. When `total + weight` is *equal to* or *more* than 20 000, the loop ends. A warning message is printed, along with the total weight loaded so far.

5 Testing the program

You will need a pencil, some paper and perhaps a calculator to test this program.

Write down a list of weights in a table like the one at the top of the next column.

weight of item (kg)	total weight loaded so far (kg)
3 000	3 000
1 000	4 000
7 000	11 000
8 000	19 000
------------	------------ weight limit
2 000	21 000

To find the total weight loaded so far, add the weight of the *next* item to the *last* value of the total weight loaded so far. When the total weight comes to 20 000 or more you can stop.

Use the same weights when the program runs. Check that the program prints the same total weight when it stops (19 000 in the example).

For you to do

Plan and write programs to solve these problems:

1 Every morning, a tanker collects milk from several farms. The tanker can hold 10 000 litres of milk. It arrives at the first farm empty.

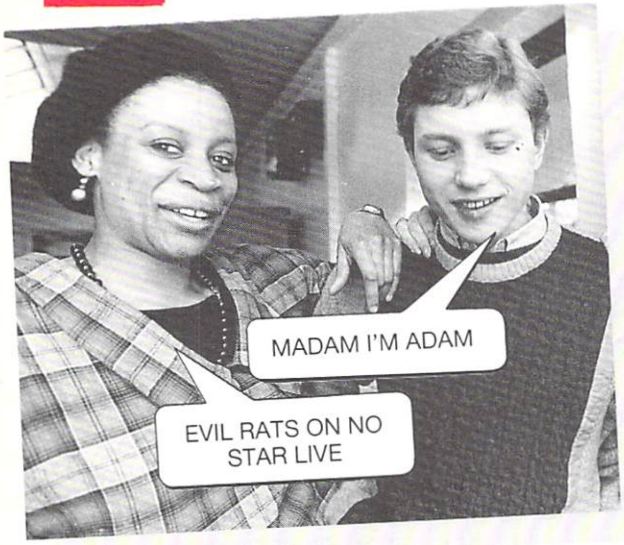
Use the computer to keep a check on the total amount of milk loaded. The computer must warn the driver when the next load will make the tanker overflow.

2 Imagine you have £1000 in a bank account. Every time you withdraw money, the total in the account drops. Write a program which:

asks for the amount of money you want to withdraw
prints the amount left in the account
repeats until you are about to run out of money
then prints a warning that you will need an overdraft

3 Use the computer to present the questions in a quiz to test someone's adding up. When a wrong answer is typed in, the computer should repeat the question.

3.9 PALINDROMES



There's something special about this strange conversation. The two sentences spell the same backwards as forwards. Words and sentences which do this are called *palindromes*. **MUM** and **DAD** are simple examples. You can use the computer to find out if words are palindromes.

1 Reversing words

`backward$` is a demon which spells words backwards for you. It works like this:

`INPUT word$` (You type in a word; say, `MONITOR`)

`pal$:= backward$(word$)`

`PRINT pal$` (The computer prints out your word in reverse: `ROTINOM`)

2 Palindrome or NOT?

`word$` stands for the word you type in.
`pal$` stands for the word spelt backwards.

If the word is a palindrome, the statement `pal$ = word$` will be *true*.

If the word isn't a palindrome, the statement `pal$ = word$` will be *false*. Put another way:

`NOT(pal$ = word$)` will be *true*.

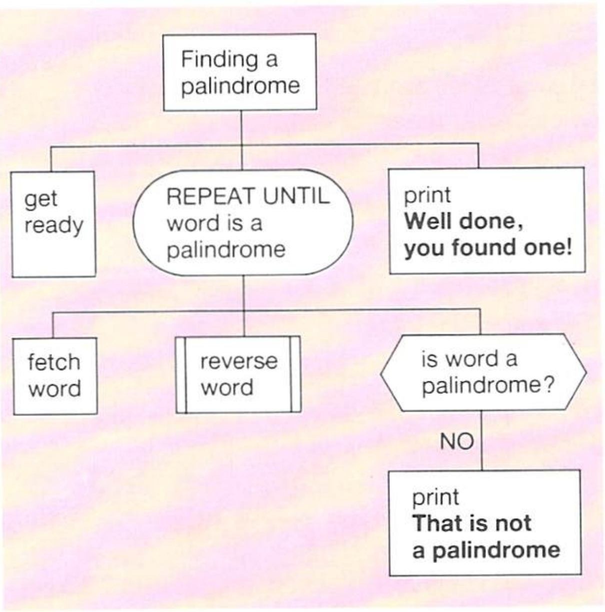
The COMAL keyword `NOT` inverts the logic, making *true* into *false* and *false* into *true*.

3 Finding a palindrome – planning the program

Here's one way to find a palindrome:

```
get ready
repeat
  fetch word
  reverse word
  if not a palindrome,
  print **Not a palindrome**
until the word is a palindrome
print **Well done, you found one!**
```

The structure diagram looks like this:


Finding a palindrome
get ready
REPEAT UNTIL word is a palindrome
print Well done, you found one!
fetch word
reverse word
is word a palindrome?
NO
print That is not a palindrome

HELP

SHIFT BREAK loads the demons menu.

f4 *lists* the names of the ready-made demons you can use. There isn't room on the screen for the full list. Press SHIFT to see more.

- To *delete* a line from your program: type the line number and press RETURN
- To *change* a line: type the line number and the new statement. Then press RETURN

4 Writing the program

First, load the `Words and Numbers – level 2` demons from disc. Start your program with a few comment statements. Then type in the main program:

```
100 MODE := 3
110 REPEAT
120   PRINT "Please type your word."
130   INPUT word$
140   pal$ := backward$(word$)
150   IF NOT (pal$ = word$) THEN
160     PRINT "Not a palindrome."
170   END IF
180 UNTIL pal$ = word$
190 PRINT "Well done, you found one!"
999 END
```

`backward$(word$)` reverses the spelling of the word you type in, `word$`. This 'reversed' word now replaces `pal$` in the program.

The `IF... THEN` statement tests the condition `NOT(pal$ = word$).`
If the condition is *true*, a message is printed saying that the word isn't a palindrome.

The `REPEAT` loop goes on repeating until `pal$` and `word$` are the same.
The loop ends when `pal$ = word$`
This statement is a test, so = is used on its own instead of :=

When the loop ends, a message is printed telling you that a palindrome has been found.

`backward$(word$)` is a demon which looks for a word called `word$` and puts the letters in reverse order.

For example:

```
word$ := "MONITOR"
PRINT backward$(word$)
```

will make the computer print `ROTINOM`, so will:

```
PRINT backward$("MONITOR")
```

5 Running the program

The program asks for a word.

If the word is not a palindrome, a message is printed. The program loops, and asks for another word.

If the word is a palindrome, a suitable message is printed, and the program ends.

6 Testing the program

It would take too long to test this program exhaustively, by typing in every possible word. Test it selectively by choosing just a few words.

First, type in a word which isn't a palindrome. Check that the computer prints `That is not a palindrome`. After this, the program should loop and ask you for another word.

Next, type a palindrome. Make sure that the computer prints the correct message before stopping.

For you to do

1 If `first := 6` and `second := 6` write down whether each of these statements is true or false:

a `first = second`
b `NOT(first = second)`
c `NOT(NOT(first = second))`

2 Write a program to carry out this plan:

```
repeat
  ask for a number
  if the number is not 0 or greater,
  print **Sorry, I wanted a positive number**
until the number is 0 or greater
```

3 Write a program to carry out this plan:

```
repeat
  ask for a word
  if the word does not have 6 letters or
    more, print **I want a longer word**
until the word has 6 letters or more
```

S Summary of Part 3

If you have completed all the work so far, you should know how to:	*Covered in section:*
sort a pair of connected lists	3.1
print individual members of a list	3.2
make your own list	3.3
use the ASCII code	3.4
nest two loops	3.5
make a selection	3.6
show a selection in a structure diagram	3.6
create a menu	3.7
use a `WHILE...DO` loop	3.8
invert the logic of a statement, using `NOT`	3.9

Useful line numbers

You should be able to use these line numbers in your programs correctly:

`10 to 99`	comment statements
`100 to 998`	main program
`999`	`END`
`1000 to 6999`	procedures you type in yourself
`7000 to 29999`	leave free for any ready-made demons
`30000 to 32767`	information stored in `DATA` statements

HELP

You should be able to use these keys for loading and listing:

SHIFT BREAK loads the demons menu from disc. Select the set of demons you want by typing a number from the menu and pressing RETURN.

If you select `Toolbox` from the main menu, the Toolbox menu should appear.
Load the demons you want by selecting their numbers on the Toolbox menu.
Choose `0` when you have finished your selection.

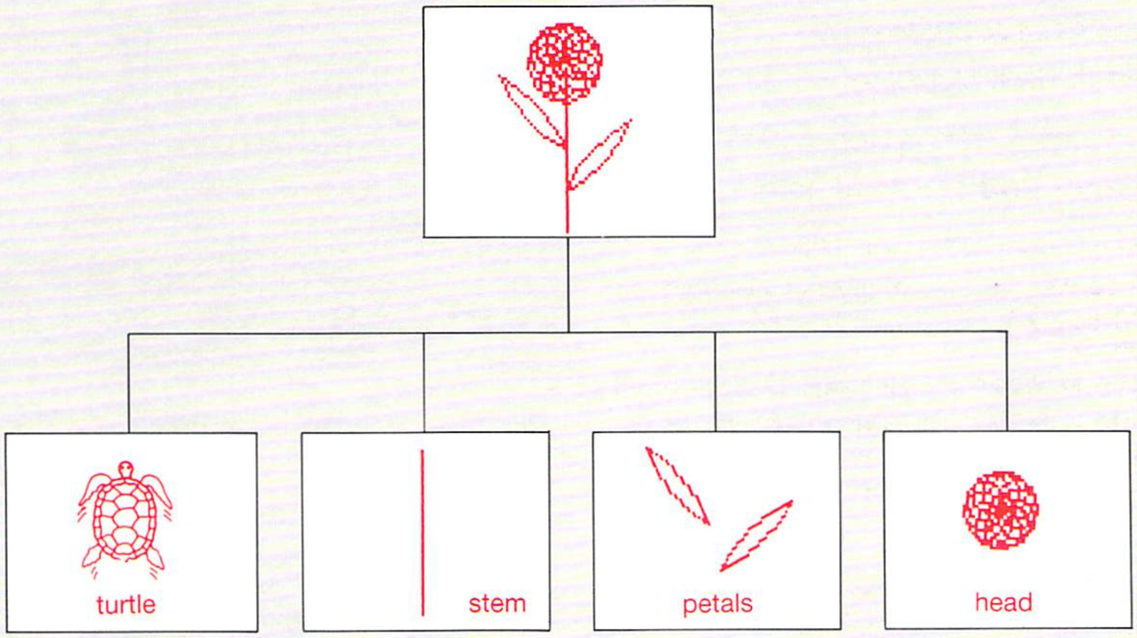
f0 lists the main program (line numbers up to `999`).

f2 lists any procedures you typed yourself.

f4 lists the names of all the demons loaded from disc.

f8 takes you back to the menu, if the demons disc is in the drive.

PROJECTS, AND HOW TO TACKLE THEM



Use the skills you have learned to solve the problems in Part 4. Project 7 on page 77 asks you to use Turtlegraphics to draw a flower.

4.1 PROBLEM SOLVING I

What's the problem?

When you have to solve a problem, first stop and think. You need to ask yourself some questions.

1 What exactly is the problem?

Start by writing the problem down. Most problems can be explained in a few short sentences. The shorter the better – but don't miss out any important details. Try to get to the heart of the problem.

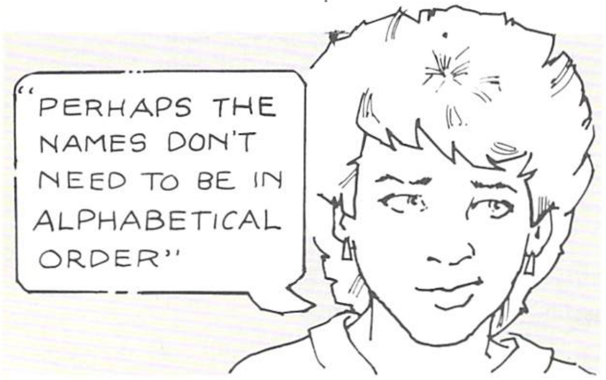
Problem
To store names and numbers of swimming club members so that I can find them quickly

2 Is there already a way of solving this problem?

A lot of problems may have already been solved. Find out if this one has already been solved. Is there a way of solving the problem without using the computer? Is using a computer the best way of solving the problem?

3 Can the problem be simplified?

Sometimes, you only need to solve the important part of the problem, especially when you are using a computer. Many details can best be dealt with by some other method.



4 What information must be given to the computer?

Does the information change each time the computer is used to solve the problem? Or will you be using the same information every time?

The answer to this question affects the way in which the information is given to the computer. Information which is always the same can be stored inside the program itself. Information which needs to be changed every time might be better typed on the keyboard.

"I DON'T WANT TO HAVE TO TYPE A NEW LIST EVERY TIME I RUN THE PROGRAM"

5 How will the computer solve the problem?

Exactly what must the computer do with the information it is given, to solve the problem? Is there already a way of solving the problem without using the computer? Can the same method be used on the computer? Is there another method which is more suitable for solving the problem by computer?

6 What information must the computer supply, when it gives the answer?

How will the information be sent out of the computer – text, graphics or sound? How realistically must the information be presented? Are graphics really necessary? Or will a few words and numbers do?

7 Is speed important?

How quickly does the problem have to be solved?

Planning a solution

Once you know what the problem is, you can plan how to solve it.

1 Write down the main steps in solving the problem, in ordinary words

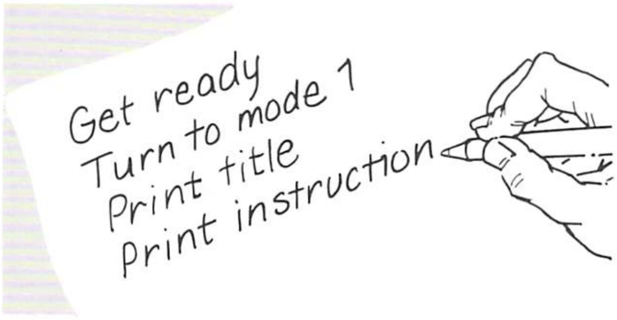
Don't worry about the detail at first. Just concentrate on the important steps.

Get ready
Ask for name
Search names stored in program

Some of these steps might involve a lot of work, but don't worry about that.

2 Divide each step into smaller steps

Underneath each main step, write down the smaller steps needed to do this step.


Get ready
Turn to mode 1
Print title
Print instruction

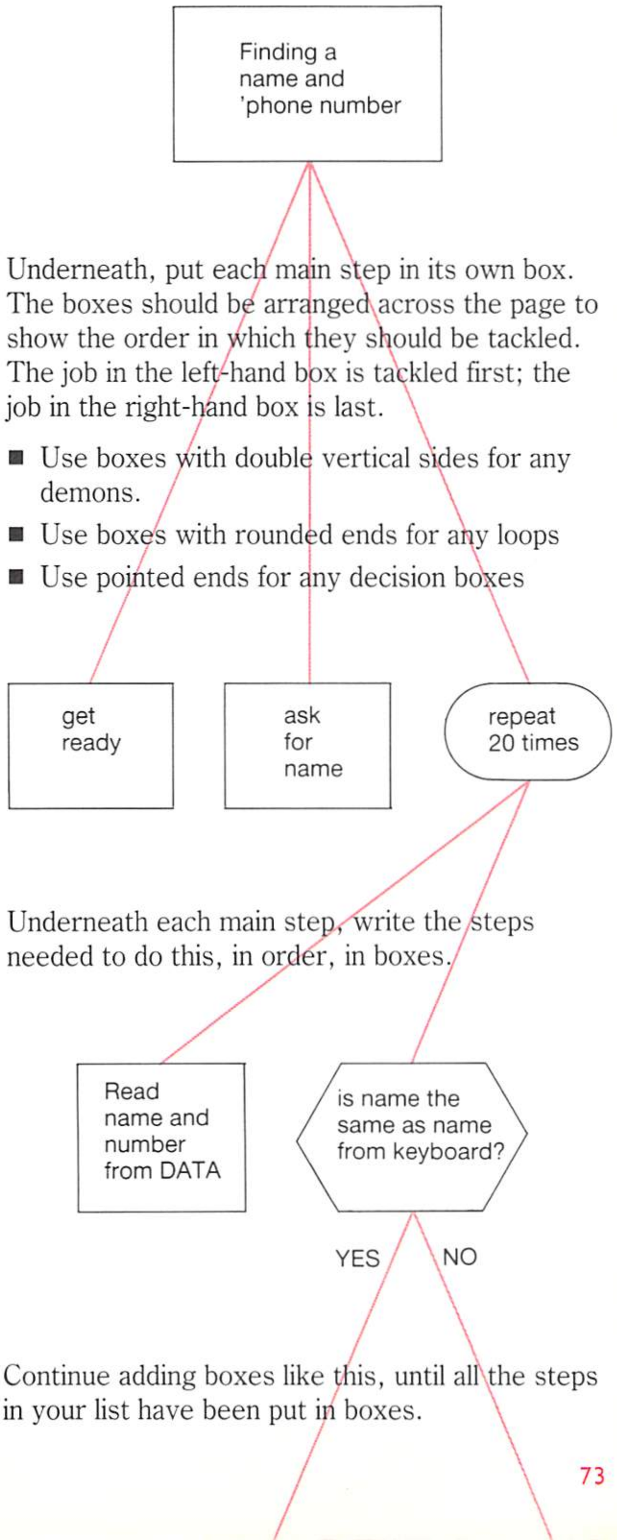
Carry on doing this, dividing each main step into smaller steps, until you have written a complete list of all the steps needed to solve the whole problem.

Don't use COMAL commands in your plan at this stage. It's not supposed to be a program yet. It's a plan which tells you how to solve the problem.

When each step in your program can't be divided up any further, your plan is finished.

3 Use your list of steps to help you draw a structure diagram

Start by writing a very brief description of the whole job in a box at the top of the diagram.


Finding a name and 'phone number
get ready
ask for name
repeat 20 times
Read name and number from DATA
is name the same as name from keyboard?
YES
NO

Underneath, put each main step in its own box. The boxes should be arranged across the page to show the order in which they should be tackled. The job in the left-hand box is tackled first; the job in the right-hand box is last.

- Use boxes with double vertical sides for any demons.
- Use boxes with rounded ends for any loops
- Use pointed ends for any decision boxes

Underneath each main step, write the steps needed to do this, in order, in boxes.

Continue adding boxes like this, until all the steps in your list have been put in boxes.

4.2 PROBLEM SOLVING II

Solving the problem

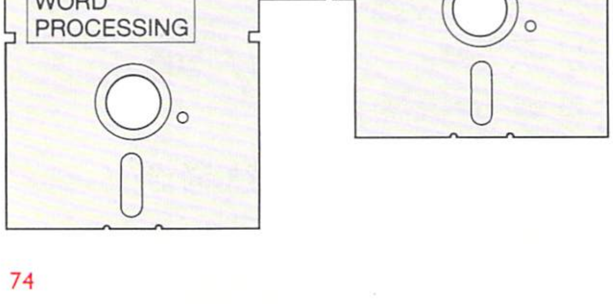
Once you have a good plan, you can solve the problem.

1 Is there an existing program you can use?

Some problems are so common that there are already programs you can use to solve them.

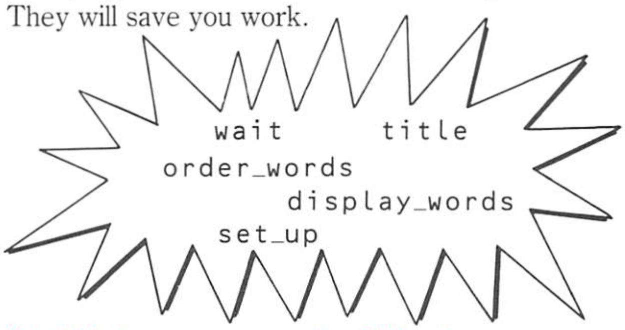
- **Wordprocessing programs** can be used whenever you need to create documents (letters, essays, notes).
- **Spreadsheets** can be used to do mathematical calculations.
- **Databases** can be used to store, recall and manipulate information.

These three kinds of programs can solve many problems easily. Use them if you can.



2 If you need to write a program, are there any demons you can use?

There are lots of useful demons you can add to your programs. Use demons whenever you can. They will save you work.



3 Write a program by following these steps:

- Make a note of any steps which can be done by demons.
- Work through your plan, turning each step into programming statements. It's a good idea to use procedures wherever possible. This breaks the program into manageable chunks.

```
100 set_up
110 title
120 instructions
130 wait
140 what_name
```

```
1000 PROC what_name
1010   CLS
1020   PRINT "Please type a name"
1030   INPUT name$
1040 END PROC what_name
```

Most main steps can be procedures. Many of the other smaller steps can also be procedures. Give your procedures sensible names that suggest what each procedure does.

Testing the program

Every program must be tested, otherwise you won't know whether it does its job properly.

1 Choose suitable information to test the program

Sometimes you can test a program with all the information to be be used. Usually, this takes too long, so you have to be content with using a few pieces of information, which must be *carefully chosen*.

Choose information that will test every kind of information that could be used with the program.

```
FIND A - PHONE NUMBER
This program contains the names
and 'phone numbers of all members
of the Swimming Club.
You can use it to find a 'phone
number when you know a name.
Press SPACEBAR to continue
```

What information would you choose to test this program?

2 Keep a printed record showing the test results

You can often get this by selecting the printer, then running the program. Or it may be more convenient to turn the printer on by using a demon inside the program.

3 Correct any mistakes in your program

Carry out a dry run and see if you can find where the program is going wrong. Use the same information you used to test the program earlier.

4 Test the program again

Keep testing, altering and re-testing until the program does its job properly.

Documenting the program

Documentation is important. It provides a record of what the program does, how it does its job, and how it should be used.

1 Give clear information on the screen

It's a good idea to make your program print a title when it runs.

If you expect someone to type information while the program is running, give clear instructions on the screen.

2 Write documentation for anyone who wants to use your program

This should include:

- your description of the problem.
- your explanation of the method used to solve the problem.
- clear instructions about how to run the program, with details about any information that has to be given to the computer as the program runs.
- an explanation of the results the computer should produce.

3 Write documentation for programmers

Use comment statements in your program, to explain what each bit does.

Use comment statements at the beginning of the program, to hold

- the program name
- the author's name
- the date when the program was written

```
 10 // Finding a phone number
 20 // by S. Gregory
 30 // 15 July 1988
100 set_up
110 title
120 instructions
130 wait
140 // ask for name to be typed
150 what_name
```

4.3 PROJECTS – LEVEL I

I Instruction book

■ Write a program to print three pages of instructions on the screen, one after another. Use the instructions given below as your text.

Load the `Words and Numbers – Level 1` demons before you begin. Then you can use the `wait` demon to give people time to read one page before going on to the next.

page 1

Operating your video cassette recorder

1 To turn the unit on, press the ON button. Press it again to turn the unit off.

2 Insert a cassette into the front of the unit.

3 If you wish to remove the cassette, press the EJECT button.

page 2

4 To play back a previously recorded cassette, press the PLAY button.

5 For sharper sound, press the TONE button.

6 If streaks or snow appear during play-back, adjust the TRACKING control.

7 To advance the tape rapidly press the FF button. Keep this button pressed if you wish to 'peep search' the tape.

8 To rewind the tape, press the REW button. Keep this button pressed if you wish to 'peep search' the picture during rewind.

page 3

9 To record a program: insert a cassette into the unit and select the channel by pressing the PROGRAM button. Press the record button.

10 To set the timer for automatic recording: Use the TIMER START and TIMER END buttons to set the start and end times of the program. Then press the TIMER REC button.

2 Sorting names

■ Write a program which will let you:

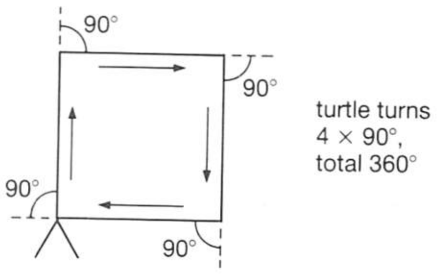
type a list of names into the computer
clear the screen
print `Here is the list you typed:`
print the list
wait until the spacebar is pressed
sort the list
clear the screen
print `Here are the names in order:`
print the names in alphabetical order

3 Pretty poly

■ Load the `Turtlegraphics` demons. Then type in this procedure, called `poly`, and try it out.

```
1000 PROC poly
1010 FOR side := 1 TO 4 DO
1020   forward(100)
1030   right(90)
1040 NEXT side
1050 END PROC poly
```

The procedure should draw a shape with 4 sides. The shape is closed because the last side joins onto the first side. The turtle makes 4 turns as it draws the shape (line `1010`). Each turn is 90° (line `1030`). So the turtle turns a total of 360°.



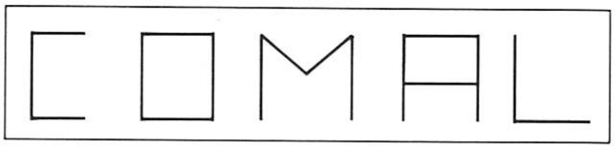
■ Change the numbers in line `1010` and line `1030` to draw shapes with

a 8 sides **b** 6 sides
c 12 sides **d** 36 sides

How many sides do you need to make the shape look like a circle?

4 Quick word

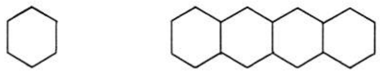
- Use the `Turtlegraphics` demons to draw this word:



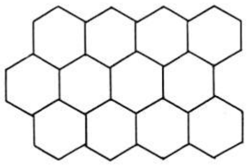
Load the `Turtlegraphics` demons first. Write a procedure for each letter. Write a procedure to draw the box around the word. Then use your procedures to draw the whole word.

5 Tiles

- Load the `Turtlegraphics` demons. Then write a procedure to draw a six-sided shape. Use it in another procedure to draw a row of six-sided shapes:

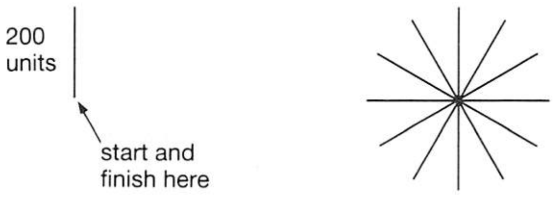


- Draw 3 rows, fitting one on top of another. Can you fill the screen with identical six-sided shapes?



6 Sunshine

- Load the `Turtlegraphics` demons. Make the turtle draw a line 200 units long. Use several lines to draw a sun. Then make the computer draw many more lines so that the sun becomes a solid disc.



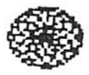
7 Flowers

- Load the `Turtlegraphics` demons. Add these procedures and try them out:

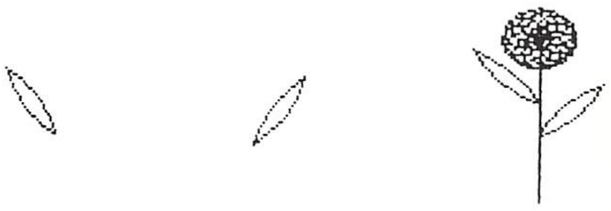
```
PROC small_circle
  hide_turtle
  FOR side := 1 TO 36 DO
  forward(4)
  right(10)
  NEXT side
  show_turtle
END PROC small_circle
```

```
PROC petal
  hide_turtle
  FOR side := 1 TO 4 DO
  forward(30)
  left(10)
  NEXT side
  left(140)
  FOR side := 1 TO 4 DO
  forward(30)
  left(10)
  NEXT side
  left(110)
  show_turtle
END PROC petal
```

- Write your own procedure, called `head`, to draw the head of a flower. You could use `small_circle` several times to turn the turtle a bit each time you use the procedure.



- Use the procedure `petal` to draw petals which lie to the right and to the left. Then write a procedure to draw a flower.



4.4 PROJECTS – LEVEL II

1 Paying the workers

Staff in a garage are paid every Friday.

Name	Job	Pay per hour
Abe Melzer	chief mechanic	£7.50
John Jackson	mechanic	£5.10
Myra North	mechanic	£5.10
Jane Brown	salesperson	£4.70
Peter Dykstra	salesperson	£4.70
Julie Young	manager	£8.00

There are 40 hours in a normal working week. Overtime is paid at one and a half times the normal amount per hour.

- Write a program that does this job for each person in the garage:

  reads the person's name and pay per hour, from DATA

  asks how many hours they have worked at the normal rate
  asks how many hours
  overtime they have worked

  calculates their pay

2 Square or triangle?

- Write a program using **Turtlegraphics** which follows this plan:

  clear the screen

  print `Do you want to see the square?`

  fetch the answer from the keyboard

  if the answer is `YES` draw a large square

  otherwise, draw a large triangle

3 Whirlybird

In the `Graphics and Sound Toolkit` there is a demon called

```
helicopter(x,y,blades)
```

This demon prints a helicopter on the screen, with its blades in two positions, numbered 1 and 2. For example:

```
helicopter(20,10,1)
```

prints the helicopter at `(20,10)` with its blades in position 1.

When the helicopter is printed in the same place on the screen, with its blades in position 1 then in position 2, it looks as though the blades are turning.

- Write a program which uses a loop to make the blades turn 50 times.

4 Train trips

Steam Spectaculars Ltd run steam train trips every summer. The cost of their latest trip is:

	single	return
adult	£4.80	£7.50
child	£2.40	£3.90

- Write a program to help calculate the cost of tickets for a party of thirty. They will need the following tickets:

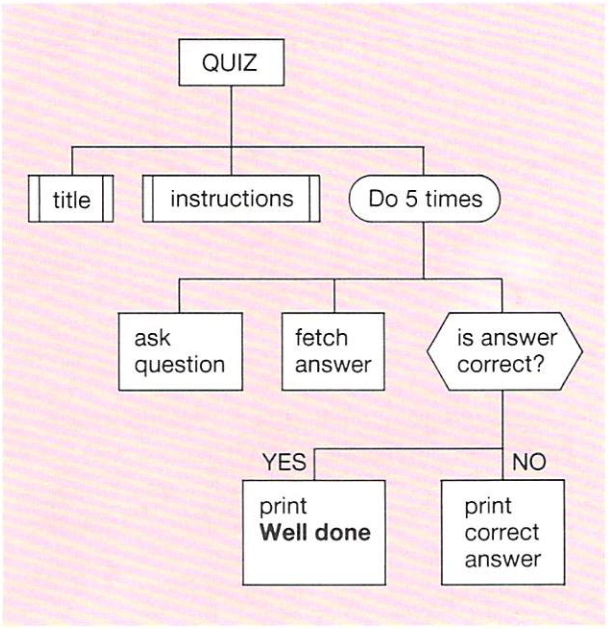
	single	return
adult	2	16
child	4	8

Your program should ask how many of each ticket are needed. Then is should print the cost, like this:

```
adult singles cost £.....
      returns cost £.....
child singles cost £.....
      returns cost £.....
                   ------
              TOTAL £.....
```

5 Choice quiz

Here is a plan for a quiz program:



When the questions are asked, the screen should look something like this:

```
Which soccer team is going to win the
next World Cup Series?
    A SCOTLAND
    B WALES
    C NORTHERN IRELAND
    D ENGLAND
Type the letter of the correct
answer:
```

- Use the plan to help you write a quiz on a subject of your own choice.

6 Cassette labels

- Print ten labels for music cassettes, complete with the details for each tape, like this:

```
----------------------------
Tape No: 147
Length: 60 minutes
Side A: Elvis on tour
Side B: Madonna's Greatest Hits
----------------------------
```

7 Pick a number

The built-in COMAL function `RND(0,9)` returns a positive whole number between 0 and 9. It selects the number at random, rather like dice. For example:

```
digit := RND(0,9)
```

will make `digit` some number between 0 and 9.

- Write a program which will:

```
repeat
   use RND(0,9) to produce a three-digit
   number between 000 and 999
   print the three digits on the screen,
   one underneath the other
until the number ends in a 9
```

8 Sounding off

- Load the `Sounds` demons. Then add this demon:

```
1000 PROC music(pitch,duration)
1010 SOUND 1,1,pitch,duration*10
1020 END PROC music
```

- Write a program to play a tune. Choose your own tune, or use the one in the chart below.

  Use the `set_up_notes` demon first. When you use `music(pitch,duration)`:
  `pitch` can be `c`, `d`, `e`, `f`, `g`, `a` or `b`.
  `duration` can be a number between `1` and `25`; this is the number of half-seconds the note will sound for. For example:

```
music(b,4)
```

will sound a note with pitch `b`, for `4` half-seconds i.e. 2 seconds.

These notes will play a tune:

pitch	c	f	a	a	g	f	d	c
duration	1	2	1	2	1	2	1	2

- Make your program print the title of the tune on the screen. If you are choosing your own tune, try making the computer print the words of each verse just before the verse begins.

4.5 PROJECTS – LEVEL III

1 Looking in the library

A small library has copies of the following books:

Title	Author	ISBN
The History Man	Bradbury	0099149109
BBC User Guide	Coll	0563165588
The Young Astronomer	Ridpath	0600309150
Coldwater Fish	Pinks	0702883603
London Transport	Day	0706360737
The Prophet	Gibran	0330262203

- Create a program which can find and print the details about each book if you know its title, *or* author, *or* ISBN.

  Begin by storing the details about the books in three lists of words. (It's easier to treat the ISBNs as words instead of numbers).

  Use a menu to allow someone to choose whether to find the book by typing the title, author or ISBN.

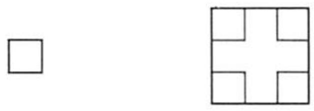
  Search one of the lists to find where to get the missing information. Then print all the details about the book.

  You may find the `search_for_word` demon in `Words and Numbers - level 2` useful.

The more details you have about a library book – title, author's name, ISBN (International Standard Book Number) – the easier it is to find.

2 Flag

- Load the `Turtlegraphics` demons. Write a procedure to draw a solid square. Then make the computer draw a shape like a flag.



3 Bird data

- Write a program which displays a page of information about each of 6 different kinds of birds (or other subjects of your choice).

  Your program should begin by asking which bird is to be described. When a choice is made, the corresponding page should be displayed. If there is no information about the bird chosen, a suitable message should be printed.

  Choosing to see a bird called `ALL` should show all the pages of information, one after the other.

Which features of this swan would you describe using your program?

4 Quiz counts

- Alter the quiz program you wrote for Project 5 on page 79, so that it keeps count of the number of correct answers given. It should print this number at the end of the program.

5 Sorting sexes

How can you use your computer to sort Jan, Sue and Alan's names into alphabetical order?

- Write down the names of six boys and six girls, in any order. Write the code letter B or G opposite each name, to show whether the name belongs to a boy or a girl.
- Write a program which:

  asks you to type the names and code letters in any order.

  puts the names into two separate lists, in alphabetical order. One list should hold boys' names and the other girls' names.

  prints the lists.

You may find the demon `sort_words (wrd$,qty)` useful. It is one of the demons in `Words and Numbers – level 2`. This demon sorts the words in a list, arranging them in alphabetical order. For example:

```
sort_words(boy$(),6)
```

will sort the first 6 names in the list called `boy$`.

6 Once upon a time . . .

- Write down 10 short sentences from a story about a person and a dog. Keep each sentence to 40 characters or less.
- Write a program which makes the computer select these sentences one by one at random, so that a story is produced. Some stories will be better than others!

  Hints: The built-in COMAL function `RND(10)` returns a positive whole number between 1 and 10. If the sentences are stored in a list, `RND(10)` could be used to decide which sentence to print next.

  It is important that each sentence is only printed once. One way of ensuring this is to store each number returned by `RND(10)` in a list. When a number is returned, the list should be searched. If the number is not already on the list, it can be used. Otherwise, ignore the number.

7 Curtains

A shop sells material for making curtains. There are several different styles:

Style	Colour	Pattern
Regency	red	flowers
Arbroath	yellow	dots
Queen	green	lines
Thrush	oatmeal	flecks
Plain	orange	none
Air	blue	clouds

- Store the styles, colours and patterns in three connected lists.
- Sort the lists and print the information in alphabetical order by style.

  There are no demons for sorting three connected lists, but you might find two of the demons in `Words and Numbers – level 2` useful. These are:

```
sort_words(wrd$(),qty)
search_for_word(wrd$(),target$,qty)
```

PROJECTS – LEVEL III continued

8 Estate agent

An estate agent uses a computer to store details of houses for sale. Each house has:

a reference code	e.g.	A14
a type	e.g.	detached
a number of rooms	e.g.	5

Customers can ask to see information arranged in various ways:

by reference code: Typing a reference code makes the computer print all the details about that property.

by house type: Typing `detached`,`semi`, `flat` or `sheltered` makes the computer print all the details of houses of that type.

by number of rooms: Typing a number of rooms makes the computer print details about houses with that number of rooms.

- Write a program which shows details of these houses:

Reference code	Type	Number of rooms
A16	detached	4
A21	semi	3
A22	detached	5
A31	sheltered	3
B1	detached	4
B12	semi	5
C43	flat	3
C92	flat	2

These three-bedroom houses are semi-detached.

9 Biscuits

- Create two lists to hold the names of these biscuits and their prices:

Name	Price per packet (p)
Tasty Crunch	45
Ginger Nips	60
Cream Snaps	40
Crumbly Crunch	80
Mint Oats	35
Brandons	40
Super Snaps	82
Malt Fingers	68
Chocolate Nibbles	84

- Write a program which will:

  calculate the average price of the packets

  print the names and prices of the packets

  print `more`, `same` or `less` to tell you how each price compares with the average.

10 Takeaway

Here is a small selection of dishes from a takeaway food shop:

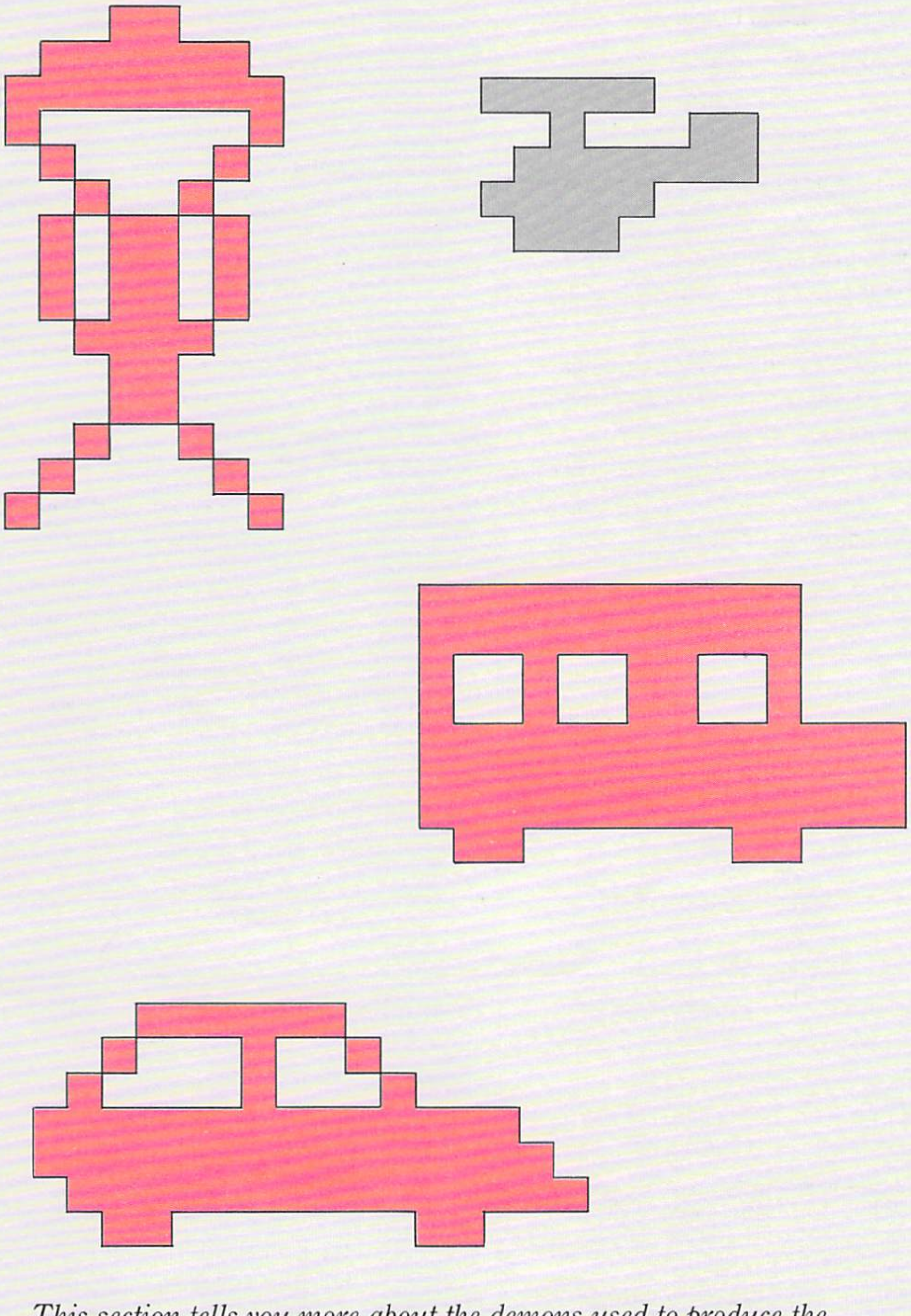
Dish	Price(p)
Curried chicken	285
Chicken fried rice	190
Chop suey	220
Spring pancake	90
Spare ribs	240
Sweet and sour chicken	310
Beef and boiled rice	265
Vegetable curry	175

- Create a program in which the names of the dishes and their prices are stored in two lists.

  Use a menu so that someone can choose whether to see the lists in alphabetical order of names, or in order of prices. The program should return to the menu so that it can be used repeatedly. The program should end when the word `FINISHED` is typed at the menu.

DEMON DOCUMENTATION

D



This section tells you more about the demons used to produce the helicopter, bus, car and parachutist, and about all the other demons used in this guide.

D Demon documentation

Turtlegraphics

Procedure name	Parameters	Description	Example
`set_up_graphics`	none	Must be used before any other turtlegraphics demons. Turns on graphics, places the turtle in the centre of the screen, and gets ready for drawing.	`set_up_graphics`
`forward (distance)`	`distance` the distance to move	Makes the turtle move forwards.	`forward(200)`
`back (distance)`	`distance` the distance to move	Makes the turtle move backwards.	`back(150)`
`left(angle)`	`angle` the angle through which to turn	Makes the turtle turn to the left through `angle` degrees.	`left(45)`
`right(angle)`	`angle` the angle through which to turn	Makes the turtle turn to the right through `angle` degrees.	`right(90)`
`home`	none	Moves the turtle to the starting position in the centre of the screen.	`home`
`pen_down`	none	Lowers the pen. Any moves after this will leave a trail.	`pen_down`
`pen_up`	none	Lifts the pen. Any moves after this will not leave a trail.	`pen_up`
`hide_turtle`	none	Makes the turtle invisible.	`hide_turtle`
`show_turtle`	none	Makes the turtle visible.	`show_turtle`
`end_graphics`	none	Returns to text, and clears the screen.	`end_graphics`

Sounds

Procedure name	Parameters	Description	Example
`beep`	none	Makes the computer beep.	`beep`
`chord(pitch)`	The pitch of the chord; `c, d, e, f, g, a` or `b`	Makes the computer play a chord.	`chord(d)`
`note(pitch)`	the pitch of the note; `c, d, e, f, g, a` or `b`	Makes the computer play a note.	`note(f)`
`set_up_notes`	none	Must be used before `note (pitch), chord(pitch) or tune.` Defines the notes `c, d, e, f, g, a` and `b.`	`set_up_notes`
`tune`	none	Makes the computer play a tune.	`tune`

Words and Numbers – level 1

Procedure name	Parameters	Description	Example
set_up	none	Must be used before any of the Words and numbers-level 1 demons. Finds out whether words, numbers or words and numbers are to be used; and how many will be used.	set_up
ask_for_words	none	Takes words from the keyboard and stores them in the list created by the set_up demon.	ask_for_words
order_words	none	Sorts the words in the list created by the set_up demon, arranging them in alphabetical order.	order_words
output_words	none	Prints the list of words created by the set_up demon, on the screen.	output_words
ask_for_numbers	none	Takes numbers from the keyboard and stores them in a list created by the set_up demon.	ask_for_numbers
order_low_to_high	none	Sorts the numbers in the list created by the set_up demon, arranging them in ascending order.	order_low_to_high
output_numbers	none	Prints the list of numbers created by the set_up demon, on the screen.	output_numbers
title	none	Prints a title page. Altering the contents of lines 11810 to 11880 changes the title.	title
instructions	none	Prints instructions for using the program. Altering the contents of lines 12010 to 12060 changes the instructions.	instructions

Demon documentation

Words and Numbers – level 2

Procedure name	Parameters	Description	Example
`take_in_numbers (nos(),qty)`	`nos()` the name of the list of numbers `qty` the number of numbers in the list.	Takes numbers from the keyboard and puts them into the named list.	`take_in_numbers (weights(),20)`
`take_in_words_and_numbers (wrd$(),nos(),qty)`	`wrd$()` the name of the list of words `nos()` the name of the list of numbers `qty` the number of items in each list	Takes in pairs of words and numbers from the keyboard, and puts them into the named lists of words and numbers.	`take_in_words_and_numbers (record$(), tracks(),30)`
`sort_words (wrd$(),qty)`	`wrd$` the name of the list of words `qty` the number of words to be sorted	Sorts the first `qty` words in the named list, arranging them in alphabetical order.	`sort_words (record$(),30)`
`sort_words_and_numbers (wrd$(),nos(),qty)`	`wrd$()` the name of the list of words `nos()` the name of the list of numbers `qty` the number of pairs of words and numbers to be sorted	Sorts the first `qty` pairs of numbers and words, arranging them in alphabetical order of the words.	`sort_words_and_numbers (record$(), tracks(),30)`
`search_for_number (nos(),target,qty)`	`nos()` the name of the list of numbers to be searched `target` the number to search for `qty` the number of items to be searched	Searches the first `qty` items in the named list of numbers. Returns the position of the first match in the list, or `0` if no match is found.	`search_for_number(weights (),100,20)`

Words and Numbers – level 2 *continued*

Procedure name	Parameters	Description	Example
`search_for_word (wrd$()target,$qty)`	`wrd$()` the name of the list of words to be searched `target$` the word to be searched for `qty` the number of items to be searched	Searches the first `qty` items in the named list of words. Returns the position of the first match in the list, or `0` if no match is found.	`search_for_word (record$(), "Computer Songs",30)`
`display_numbers (nos(),qty)`	`nos()` the name of the list of numbers `qty` the number of items to be displayed	Displays the first `qty` numbers from the named list, on the screen.	`display_numbers (tracks(),30)`
`display_words (wrd$(),qty)`	`wrd$()` the name of the list of words `qty` the number of items to be displayed	Displays the first `qty` words from the named list, on the screen.	`display_words (record$(),30)`
`display_words_and_numbers (wrd$(),nos(),qty)`	`wrd$()` the name of the list of words `nos()` the name of the list of numbers `qty` the number of pairs of words and numbers to be displayed	Displays the first `qty` pairs of words and numbers from the named lists, on the screen.	`display_words_ and_numbers (record$(), tracks(),30)`
`all_but_first_letter$ (word$)`	`word$` the word from which the first letter is to be removed	Returns the given word after removing its first letter.	`all_but_first_ letter$ ("shush")`
`backward$(word$)`	`word$` the word to be spelt in reverse.	Returns the given word after reversing the order of its letters.	`backward$ ("raw")`
`first_letter$(word$)`	`word$` the word whose first letter is to be identified.	Returns the first letter of the given word.	`first_letter$ ("rhododendron")`
`last_letter$(word$)`	`word$` the word whose last letter is to be identified.	Returns the last letter of the given word.	`last_letter$ ("petunia")`

Words and Numbers – level 2 *continued*

Procedure name	Parameters	Description	Example
`die`	none	Returns a positive whole number between 1 and 6.	`die`
`max(nos(),qty)`	`nos()` the name of the list of numbers `qty` the number of items to be examined	Examines the first `qty` items in the named list, and returns the maximum value.	`max(weights(), 20)`
`min(nos(),qty)`	`nos()` the name of the list of numbers `qty` the number of items to be examined	Examines the first `qty` items in the named list, and returns the minimum value.	`min(weights(), 20)`

Graphics and Sound toolbox

Procedure name	Parameters	Description	Example
`background(shade$)`	`shade$` black, red, green, yellow, blue, magenta, cyan or white (depending on the mode) the colour of the background	Sets the background colour to be used in any printing.	`background ("yellow")`
`bird(x,y,state)`	`x,y` the printing position of the left side of the bird `state` the way in which to draw the bird's wings 1, 2, 3, 4 or 5)	Prints a bird with its wings in one of five states.	`bird(12,14,2)`
`bus(x,y)`	`x,y` the printing position of the left side of the bus	Prints a bus.	`bus(8,10)`
`bomb`	none	Prints a message, and a sequence showing a bomb exploding.	`bomb`
`camel(x,y)`	`x,y` the printing position of the top left corner of the camel	Prints a camel.	`camel(5,5)`

Graphics and Sound toolbox *continued*

Procedure name	Parameters	Description	Example
`car(x,y,)`	`x,y` the printing position of the left side of the car	Prints a car.	`car(12,20)`
`centre(message$,y)`	`message$` the message to be printed `y` the number of lines to move down the screen, from the top, before printing the message	Prints a message, centred on a line, `y` lines from the top of the screen.	`centre ("Hello there!",10)`
`dragon(x,y)`	`x,y` the printing position of the top left corner of the dragon	Prints a dragon.	`dragon(10,12)`
`drive(y,vehicle$,way$)`	`y` the position of the line on which to print the vehicle `vehicle$` `bus` or `car`, the vehicle to be printed `way$` `backward` or `forward`, the direction in which to move the vehicle	Moves a bus or car across the screen.	`drive (10, "bus", "forward")`
`elephant(x,y)`	`x,y` the printing position of the top left corner of the elephant	Prints an elephant	`elephant(20,20)`
`foreground(shade$)`	`shade$` `black, red, green, yellow, blue, magenta, cyan` or `white` (depending on mode) the colour of the foreground	Sets the foreground colour to be used in any printing.	`foreground ("red")`
`gallop(y,way$)`	`y` the position of the line on which to print `way$` `forward` or `backward` the direction in which to move the shape	Moves a horse and rider across the screen.	`gallop(8, "forward")`
`hide_cursor`	Makes the cursor invisible.	none	`hide_cursor`

Graphics and Sound toolbox *continued*

Procedure name	Parameters	Description	Example
`helicopter(x,y,blades)`	`x,y` the printing position of the helicopter blades; 1 or 2. the position of the blades	Prints a helicopter.	`helicopter (12,10,1)`
`horse(x,y,legs)`	`x,y,` the printing position of the left side of the horse `legs` the position of the horse's legs; 1, 2 or 3	Prints a horse.	`horse(0,0,1)`
`man(x,y,pose)`	`x,y` the printing position of the top left corner of the shape `pose` the position of the person's legs; 1, 2 or 3	Prints a person in one of three poses.	`man(0,12,3)`
`new_page`	none	Clears the screen. If the printer is selected, this procedure also makes the paper feed to the start of a new page.	`new_page`
`parachutist(x,y)`	`(x,y)` the printing position of the top of the shape	Prints a parachutist.	parachutist `(10,0)`
`pause(seconds)`	`seconds` length of the pause	Makes the computer pause.	`pause(3)`
`spaceship (x,y,state)`	`x,y` printing position of left edge of the shape `state` the direction in which to point the spaceship; 1, 2, 3, 4 or 5	Prints a spaceship pointing in one of 5 directions.	`spaceship (0,0,1)`
`show_cursor`	none	Makes cursor visible.	`show_cursor`
`switch_printer_off`	none	Stops information being sent to printer.	`switch_printer _off`
`switch_printer_on`	none	Allows information to be sent to the printer.	`switch_printer_ on`
`wait`	none	Makes the computer print a message at the bottom of the screen, then wait until the spacebar is pressed, before continuing.	`wait`

I INDEXES

Index of COMAL demons

Index of COMAL key words

I INDEXES

Index of main ideas

This is an innovative scheme which provides you with all you need for GCSE and Standard Grade computer studies courses. By dealing with computers and computing in situations relevant to everyday life, all aspects of computer studies are covered in a stimulating and informative way. The detailed description of real-life contexts means the scheme is also well suited to Information Technology examination courses.

The scheme consists of four components: the pupil textbook, a programming guide, a disk containing program procedures and a teacher's resource pack.

- The textbook provides a stimulating framework for the course.
- The programming guide encourages each pupil to adopt a problem-solving approach.
- The disk supplies the programming 'tools' with which to solve the problems in the guide.
- The teacher's resource pack offers a wide range of practical activities and extension work with which to develop the course.

Both the programming guide and the disk are available in BBC BASIC and BBC COMAL versions. Further details are included in the 'To the Teacher' section at the start of this book.

COMAL Programming guide

- **Each spread starts with a real-life problem for pupils to solve.**
- **Pupils are guided through the problems analysing the problem, designing a solution and then testing that solution.**
- **Additional exercises on each spread.**

ISBN 435 770551

Heinemann Educational Books