Mein LASER Home-Computer

• b
Hamburg
SANYO

Video Vertne ,

Tips and tricks
for beginners

My

LASER

Home Computer

Tips and Tricks
for beginners

Introduction to Programming with
Switching Images

SANYO Video Sales, Hamburg

Imprint:

COPYRIGHT 1984 by SANYO Video Distribution, Hamburg.
Publishing: Roland Löhr, Ahrensburg.
Photo Set: Telesatz Infora GmbH, Norderstedt near Hamburg. Book cover and sketches designed by Cathrin Utescher, Hamburg. Printing and production: Kuncke Druck GmbH, Ahrensburg.

The text of this book was captured on a personal computer. A TerminalComputer transferred it to the Light Set Machine, whose computer also performed the hyphenation. Apart from minor irregularities, for which we ask for understanding, the interaction between humans and computers worked well.

All friends and owners of the Horne-Computer LASER and the VZ 200 should be informed about the start of working with the computer in an easily understandable way.

For the LASER 110 (4 KByte RAM, monochrome), the LASER 210 (8 KByte RAM, 8 colours) and the new LASER 310 (8 KByte RAM, 8 colours, OWERTY typewriter keyboard) as well as for the VZ 200 (4 KByte RAM, 8 colours) there are a wide range of user software and a wide range of peripherals, including 64 KByte RAM expansion module, Lightpen, FloppyDisk drive and more, which makes the application so appealing.

In this book we had to limit ourselves to describing and explaining the BASIC part. The series continues, however, with an edition that deals intensively with the technology of the devices, deals with the possibilities of programming in machine language and contains a variety *of* special *switching* images for advanced users.

But now 'get to work and have fun with your Horne computer!

Hamburg, June 1984

Table of Contents

1 The ancestors of the computer

Isn't it exciting? You just brought your new home computer home. And the first thing you want to know is how to use it. Well, we'll get to that in a moment. First, you should know what your new machine actually represents.

Everybody knows that computers are called electron brains, right? No wrong!
Actually, they have no brains at all.

Forget what you've seen in movies about it, amazing machines that have intelligent conversations, ideas or plans to take over world domination. A computer is more like a trained parrot. They can teach him to do tricks or obey orders and even "talk" to him. But he certainly can't think like you and me! So that's what a computer is not. To explain what a computer is, consider the predecessors of your computer.

Since the beginning of history, man has sought ways and means to make life easier for him. He discovered that there were limits to his physical ability. So he started to look for ways. that allowed him to expand his physical abilities. This was done with tools.

Humanity today relies on many different tools. The motor vehicle, for example, is a tool that allows us to move much faster than our legs can.

The phone is a tool that allows us to talk to someone much further away than our voice is audible.

A microscope is a tool by which we can look at things that are not recognisable by the naked eye.

And the computer, after all, is a tool that allows us to remember a lot and do calculations much faster than our brains can do on their own. In its case, it began thousands of years ago with Chinese Abacus.

What we're trying to say is: You don't need to be afraid of your computer.

You just need to remember that this is a tool to enhance your mental abilities. Without your help, he can

do not think; it has no independent mind; and he will certainly not laugh at you if you do something wrong!

Speaking of remembering, here's another point you should remember:

No matter what you type in your computer, you cannot infringe it. You can play around with it, type in as much nonsense as you want. They will never end up in a situation that is hopeless. Because when it gets to the worst, they can easily turn the device off and start again. So experiment as much as you want. Why worry about what might happen if you press this instead of the other key. Try it and see what comes out?!

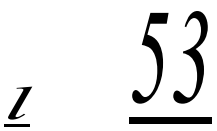
2 Commissioning of the computer

To connect your computer, you must turn off your TV. On the back of the TV you will find the antenna port. This port is used to receive images from the TV.

But for now, we want your TV to receive information from your computer!

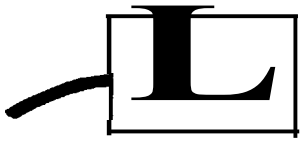
So: Remove the antenna cable from the connector jack. Instead, they use the thick cable that's inside the box on your computer. The thickest end of this cable will be inserted into the socket on your TV.

So much for one end of the cable. The other end fits into the outermost right-hand jack on the back of your computer. (If you look carefully, you can see the letters "Tv." just below this jack.)

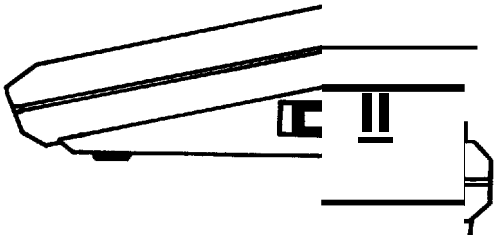


The computer package also includes a black device with a thin cable. This device is called a power supply ([redacted]use it converts the mains current of a standard house connection (220v, 50 Hz) into a power type suitable for your computer. First, plug the usual power cord into the wall. All right, you'll end up with the long, thin cable and a little round

See the plug. Plug it into the far left jack on the back of your computer. (Below this port on the computer is "DC9V".

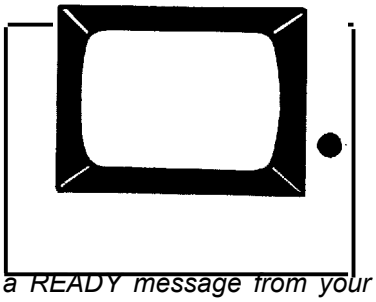


The computer powers on by pressing the toggle switch located on the right. (If all is well, the little red power light will light up on the top of the computer.)



Turn your TV back on and then select Channel 1. That's it, done!

Everything is now ready for operation. And look. If you've done everything right, your computer will have started talking to you. He tells you that he is "READY" (ready) to listen to everything you tell him. The cursor, the flashing square on the third line, should also be visible. The cursor has the function to tell you where the next location is on the screen.



If you do not receive a READY message from your computer, check the ports you have made. The error may only be on a loose contact or a wrong port.

A big difference between the computer and the human When people talk to each other, they need two things.

> Ears to hear what is being said and a mouth to answer, ask questions and so on.

Look at the computer. Sure he has neither ears nor a mouth, but computers are nevertheless able to "talk"! The computer's keyboard is, in a sense, his "ears." If you enter questions or instructions, he will "hear" you. And if the computer wants to "talk" to you, it uses the screen (in this case, your TV set.) Can you imagine what that means? Right - in order to talk to a computer, you have to use your senses a little differently than usual.

In other words, when you talk to a computer, your fingers will be your mouth and your eyes will be your ears!

"But how do I use the keyboard?"

If you are confused by the number of individual buttons at this time, this is no problem. First of all, we want to call the entire keyboard area "the keyboard." Secondly, we want to explain how easy keyboards are to use.

There are many letters, numbers, symbols, commands, etc. that your computer understands. Of course we could have assigned a key to each character. But this would have made the computer's keyboard quite large and cluttered!

It was better to add multiple functions to each key, called *multi-functions.* Some of these multi-functions can have up to four different applications!

To show you how this affects, we choose any key, e.g. the "P" key. Here's what it looks like on the keyboard.

If you want to write the letter "P", just press the button. If you want to have the square clamp, you must first find the "SHIFT" key. (It is located in the lower left corner of the keyboard.) You got it? Now press and hold the SHIFT key until you press the P key.

Above and below the "p" key, you see two more symbols.

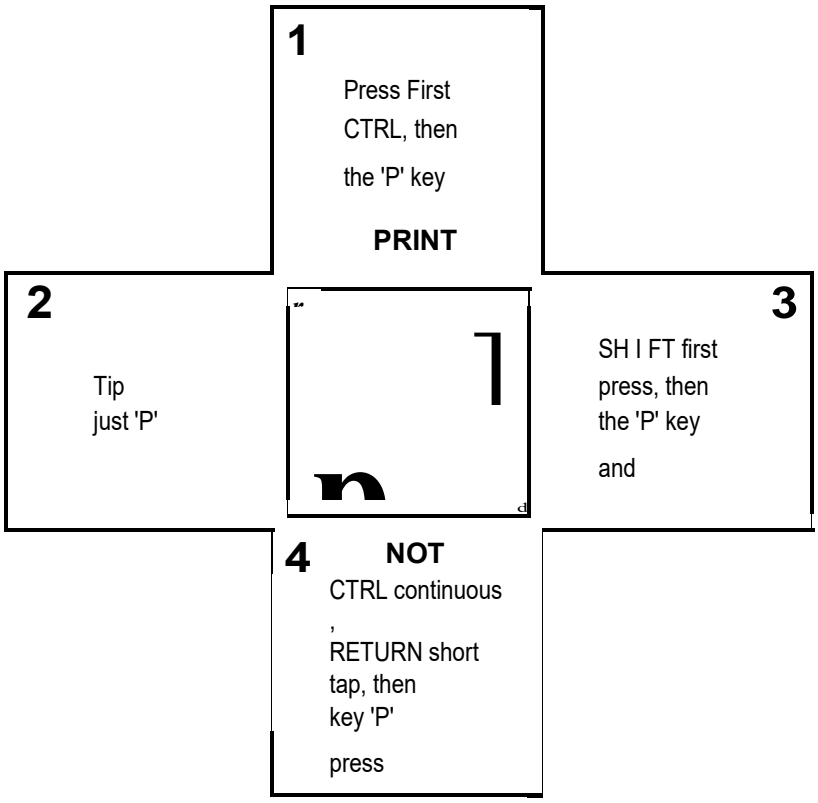
> **To** get **"PRINT"**, hold down the "CTRL" key (located just above SHIFT) while pressing the P key.

If "NOT" is desired, two things must be done:

1. CTRL is pressed until RETURN is pressed and released.

Then press the P key.

The above rules must be observed with each multi-function key.



1. Application: Hold down the CTRL key while pressing for PRINT
2. Application: Press only the key for P
3. Application: Press and hold SHIFT until the key is pressed
4. Application: Hold down CTRL, press Return, press the key for NOT

By the way, the peculiar box-shaped characters on some buttons are called graphic characters. They will be discussed later in the graphic display on the screen.

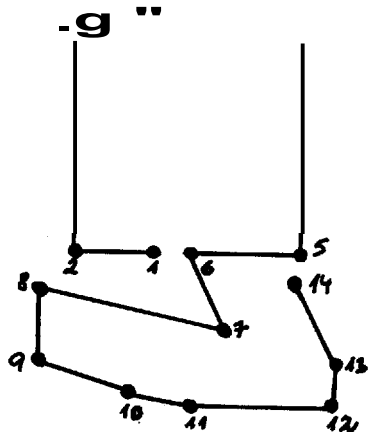
3 Programming short!

We say: "Computer Programming"

They think: "HELP!" Well, that's an understandable response to something you don't know about.

But wait, don't look black! It's far from as complicated as you thought. On the contrary, it is indeed very easy to understand. Especially if you think of it this way:

Suppose they wanted to solve a *numbers-to-numbers game*, as illustrated by magazines (everyone knows how easy it is!) You naturally start at the first step (this is the point with the number 1). Then you draw the line to the point with the number 2, from there to point 3 and so on, until you arrive at the last figure.



There's nothing hard about that, right? You only have to pay attention to an already given write sequence. And this created a picture, even though you didn't know what the image would look like at the end. In other words: *You followed a programme!*

So your activity as a programmer is simply to set a sequence of steps for your computer in advance.

There is something important to note here: What would happen if the points of the Numbers game were misstated? Or if

What are some points that would be missing? Yes, it would produce a structure that would make the wrong or no sense!

In order for the programme to make sense to the computer, *you must make sure that the individual steps are entered in the correct order.So,* and now you already know much more about the "mysterious" computer programming. (It's not that wild, is it?)

In just a few minutes, we learned the basics of how to get the computer to do what we want.

* It needs the *steps* it needs to follow to complete the task
* and the *order* of steps to be processed.

So far so good! But there's something else you need to know to tell the computer what to do, which is how to *talk to* it!

Overcoming the Language Hurdle

Recall the last chapter on the differences between computer and human. Here's another difference.
Computers are not intelligent enough to understand our complex language! *To overcome this "language barrier", humans use a special language called "BASIC".*

You're probably assuming that something called *"BASIC"* must be pretty *simple*. Well, that's really how it goes. You can compare this language very well with English, but it has a much lower vocabulary.

You can apply a variety of things to a fellow human. E.g.:
"Make me something to eat!" or "Climb the flagpole!" Of course, we need many different words to give such instructions. By contrast, a computer is only capable of doing relatively few tasks. That is why he can understand few words. (Try telling your computer to prepare something for you to eat, it won't understand).

And now some computer psychology

As I mentioned, a computer is not an electronic brain; it has no brain at all. The computer is actually only capable of *remembering* (using memory) and *executing commands* (with its computer).

But humans are equipped with brains. It allows us to think, to come up with ideas, to work out solutions to problems, to take initiative, and to do a whole bunch of other smart things. So if you want to have a "conversation" with the computer, it is important that you are always aware of these differences. Because the computer will happily do every task assigned to it exactly, very, very quickly. *But he will do nothing without your prompting!*

There you might encounter difficulties, because after all, most people are used to talking to others, and it's easy to forget how stupid the computer really is.

For example: Suppose you added a friend 2 and 2. Your friend would think for a moment, and then he would answer. Why? Because he would automatically assume that you want to know the answer. Difference: If you ask your computer, they would wait all day for an answer without receiving one. *Your computer won't give you an answer until you tell it to! This is because, unlike your friend, he is unable to make assumptions.* Your computer won't understand you unless you express yourself clearly.

Second example: Suppose you say to your friend: "Please go and get me my bar. It's in one of the drawers over there." Probably your friend would understand that it is the ruler you are looking for. And he would look in the drawers for it until he found it.

But if your friend's brain worked like a computer, he wouldn't be able to figure out what you want him to do! Instead, he would say something: "I don't understand! What's a bar? How do I get to the drawers? Which drawer should I look in? Or he would sit and look at you with an empty look!

Notice one thing. Computers are not as intelligent as we are, but they rarely make mistakes. So what to do if the computer does not run a programme to the result you want? Or even refuses to do anything at all? Well, one thing you shouldn't do is cover it with desolate abuse. Not because he is delicate, but because he is most likely not to blame.

> The probability that you - the programmer - have patted is 1: 1,000,000 against them, because the computer works almost flawlessly.

Therefore, you must be very careful that the instructions given are

in detail. Because if there's even a small thing missed, your computer will never find out what it was.

4 Our first programme

Now that you already know a lot about computers, we want to programme something. Programming your computer is as easy as saying "hello". And to prove that, "HELLO" will be the first word we teach computers.

The use of a row number

Remember the Numbers Pin game. A number was assigned to each point. These digits ordered the steps. Your computer needs to know the order in which it should perform the individual steps. So we give it a number for each step (or line).

Of course, one could easily use the digits 1, 2, 3, etc. but a good plan is not. Because when you create longer programmes, it may well happen that one or the other line is omitted by mistake (this happens to everyone once). With paragraphs 1, 2, 3, etc. But you have no possibility to insert the forgotten lines later.

It is therefore better to add more distances to the initial specification, e.g.: 10, 20, 30 etc. This will give you nine unused intermediate lines (in all cases).

Okay, computer say "HI!"

Type the lines you see below as you see them. (What? Did you say you don't know how? All right, for the moment, it's enough if you keep the lines in mind. We go line by line, button by key, so you can see how easy it is.

Press 10 PRINT "HELLO" RETURN button
Press 20 GOTO 10 RETURN button

So, the first thing that should be on the top line is the number 10 (that is, explained above, the line number). Simply press the 1 key on it and then the 0 key on it. Now you see the 10 on the screen. And the little blinking cursor is immediately behind the 0. It marks the point where the next character will appear.

By the way: Have you heard a beep after each key? This is the message from your computer that says: "Yes, I took the push of a button."

What's next in our first line? The word PRINT, isn't it? So no, not quite. First, you must enter a space. *Empty spaces are entered on the computer using a special key, the Space key.* (See this key? It is located in the lower right corner of the keyboard.) Try it now. Press the key once and release it. You will notice that the cursor has jumped one place to the right.

Now it's PRINT's turn. There are several ways to enter PRINT. The usual method is to type letter by letter. But there's a *shortcut* you might prefer to avoid the typing, which is this. on the P key, you see the word PRINT. And you already know how to type in the shortcut. (What? You've already forgotten that? If so, go back to the end of the 2nd. Chapter.) We try the shortcut and hold down the CTRL key while pressing P.

Well, that's pretty convenient. As fast as your computer could beep, the word PRINT appeared on the screen. It may look like magic, but it's just a simplification. Your computer offers you a variety of such abbreviations. Almost all of the buttons (above or below) on your keyboard have words that are quite useful.

To separate PRINT from the next word, we use the Space key again. Has the cursor jumped to the right one spot? Good! You'll notice that "Hello" is within *quotation marks (*"). The character is located in the upper corner of the 2-key. Press and hold SHIFT while pressing 2. You got it? Good! And now type in Hello, then another quotation mark at the end. Done!

Finally, a RETURN is at the end of the first line of the programme. This is really the easiest. Press the key and watch what happens. Nothing new appears on the screen, but the cursor jumped to the top of the next line.

Now that we have convinced you of the ease of use, you can start the next line without our help:

Tap 20 then GOTO, just above the G key, then 10 and finally the RETURN key.

Do you actually know what you just did? It's really exciting. You just talked to your computer using a keyboard. And you used two special words from the BASIC language mentioned in chapter three.

We're now translating what's been programmed so far and showing you exactly what prompted you.

> PRINT means: "Write this on the screen." (Never forget the double quotation marks that must include the end of the print.)
> GOTO means: "Go on to the line I'm about to enter." This command is always followed by a line number, because you must always tell the computer exactly where to continue editing. (In this case, we wanted it to go back to line 10. So we said, GOTO 10. All right?
> The RETURN button simply tells the computer: That's all in this line. Now look at the next one. This button closes almost every line you type in your computer.

We congratulate you! Your first programme is currently sitting in the memory of your computer. Fantastic, you might say. But why is nothing happening? Well, it's about time... as soon as we have taught you some BASIC words.

> The words you're about to learn are called *direct commands.* You do not need a line number (even though the computer would run it). When your computer reads a direct command, it knows that you expect a specific execution, and that is immediately.

The first direct command is LIST and means: "Show me the programme the way I typed it." If you type LIST, the entire programme will appear on the screen. Try this now. (Nothing happened? If not, it's probably because you failed to type RETURN. And until you do, your computer will think you're not done with your command input.)

The second direct command is RUN and means: "Execute the programme instruction." If you type RUN and then RETURN on your computer, it will run the programme you entered.

Are you ready to see what your little programme will do? Okay. Then type RUN and RETURN and watch what happens. Great! If you couldn't imagine what was going to happen, you'll have been surprised by the many "HALLOS" appearing on the screen. But it's actually not that surprising when you look at how the programme works.

You remember that the computer starts with the lowest row digit and usually works on the next one, when it does not meet a GOTO.

> So the computer first took a line 10. This line caused him to print "HELLO". Then he went on to the next line (20). Line 20 tells the computer to go back to line 10. And that makes it start all over again.

This clever trick is called *looping.* Why? Because a loop is endless, which is the case with this kind of programme. It's kind of like sending your computer on a never-ending carousel ride.

> If your computer refused to say HELLO, don't blame it. Type LIST again (you haven't already forgotten what that means, have you?) and verify that the programme on the screen is exactly the same as in this book. You may discover an error. Also verify that all connection cables are seated correctly.

"But how do I stop him?"

In the meantime, while we've been busy, the computer has become a real "HI-Dri" (it doesn't stop printing "HI-O" anymore). And not until we slow down or cancel the programme. Here's another BASIC command that will do just that. It is called "BREAK" (Cancel) and it means: "Stop where you are." The BREAK button is located in the top row of keys on the far right. To apply, press the BREAK key while holding the CTRL key. And now comes a surprise: RETURN does not have to be typed after BREAK. Commands that do not need to enter RETURN are very thin (extremely thin, because BREAK is the only command that does not require RETURN!).

With BREAK, you're guaranteed to quit. You should see the last HALLOS on your screen, followed by: BREAK IN LINE 10 (or BREAK IN LINE 20, that depends on where the computer was when you "slowed down").

If you have enjoyed the HALLOs and want to see more, you can enter CONT and RETURN on the computer. CONT is an abbreviation for "continue" and instructs the computer to continue where you interrupted the programme.

Of course, you can easily stop your programme by typing another BASIC statement, namely END. A programme that allows you to print your computer "HELLO" just once looks like this:

```
10 PRINT "HELLO"
20 END
```

END means: "This is the end of the programme".

Are you ready to go on? Then we now learn two new BASIC commandos, which would also cancel the HALLOs: CLS and NEW

CLS means: "Delete Screen"

Try it now

CLS RETURN

See? The screen is deleted and the computer tells you that it is READY (ready) again.

NEW means: "Delete Saved Programme".

This command deletes the screen and memory. This is comparable to immediate memory loss.

So type them once

NEW RETURN

Nothing will change on the screen, but if you try to instruct RUN now, the computer will just tell you that it is READY.

BASIC commands we have learned so far:

PRINT' ...." Show what is inside the" ...." is on the screen. GOTO go to the line (never forget to enter a line number immediately after this command!)

END End of programme

LIST Show my programme on the screen to RUN

Run the programme

BREAK Interrupt the programme

CONT Continue the programme

CLS Deletes the screen

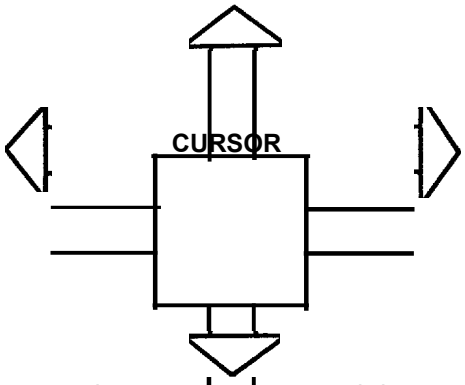
NEW delete the programme

And never forget: At the end of each line, tap the RETURN key to let the computer know that you are completing a command.

How and hint error Corrected: The Editing

No matter how careful you were about typing, sooner or later you will make a typo by typing the wrong button. Don't despair. Patters happened quickly. You wouldn't be human, you wouldn't make mistakes. And besides, your computer is so *clever* that it's as easy to fix bugs as it is to make them!

If you look at the last four keys in the bottom row of the keyboard, what do you see? These keys matter. Above each is an arrow. An arrow points to the left, one to the right, one to the top and one to the bottom.



You can use these arrows to control the cursor on the screen. The idea of the story is to place the cursor directly on the error.

"Yes, but how do the arrow keys work?"

When you press and hold the CTRL key, each single whistle key press will move the computer's cursor one stiffness toward the arrow. (Up, down, left, right, it depends which arrow you choose.)

If you are not happy with it, just one more place at a time, then try to keep your finger on the arrow button for longer. The cursor will move (beeping) over the screen until you release the key.

A little warning: If you try this trick, you could shoot over the target and miss the target. But don't worry, just use the opposite arrow key to bring the cursor back to your destination. (To control the cursor around the screen is about like driving a car scooter, you can drive as you like.)

A warning: Do not forget to hold down the CTRL key throughout the time you use the arrow key. Because if you don't, the cursor on the screen will completely disrupt everything.

There are several ways to make a splash, but fortunately there are also several ways to fix it.

"I typed one letter too many!"

Suppose you typed line 10 as follows:

10 PRINT "HAALLO"

That looks a little strange! We need to delete the superfluous A. To erase a letter (or a space, a number, or any character) is easy, because there is a special command for it, namely: RUBOUT! This command is located above the ";" key.

Troubleshooting:

Use the arrows to place the cursor on the superfluous "A" (it does not matter on which). Now hold down the CTRL key and press RUBOUT. The cursor "swallows" the letter below.

"I left out a letter"

Also there is a command that allows you to create a forgotten sign two

see two others. This command will: INSERT. You can find it above the "L" key.

What if you had written line 10 like this:

10 PRINT "ALLO"

Fixing such a bug:

Place the cursor over the character that follows *after* the omitted character. (In this case, it would be the letter "A"). Holding down the CTRL key and pressing INSERT, the cursor will move the letter underneath it to the right. (Why? To make room for the character you want to insert.) Insert the forgotten "H" into the resulting space.

> Unfortunately, INSERT does not allow you to make shortcuts. To insert more than one character, press INSERT before inserting.

"I really fucked that line!"

Sometimes, a command line is totally wasted. For example, you might end up with the following:

1 & NELOO"

What a disaster! In such a case, it is easiest to rewrite the whole line. To do this, move the cursor to the beginning of the line and just type across the whole and then RETURN.

After you have made such a correction, it would be quite useful to list your programme. Why would you do such a thing? Simply search for the wrong lines that may still be stored in your computer.

If at least the line digit was correct, there is a more appropriate solution for the mangled line. Simply re-type the entire line using the same row number. (But this time without errors!) Your computer knows that there cannot be two lines with the same line number. So he will replace the first line with the second (correct) line.

> We have an important rule to tell you: *After any kind of corrections (whether RUBOUT, INSERT, or re-typing a whole line), press RETURN. It does not matter where you are in the line.* Because it is the case that with RETURN, you will be able to

puter say that he has to observe the corrections just made.

5 And now some math magic!

Wonderful! Now you've done your first piece of programming. And it worked. This programming business is certainly quite useful - right? But your computer also has another page, and that's at least as useful.

It's just a matter of moments, and your computer evolves to become a

Super calculator!

You don't even have to send him to university, but you just have to type the instructions a little differently.

When we wrote our first programme, each step had a line number. For your computer, placing a number at the beginning of each line means something like: "Remember these steps, but don't follow them yet." The computer takes your instructions into its memory so it doesn't forget what you wanted.

These are called conditional instructions.

In addition, there is another type of command that you can give to your computer. These are: "Take this step now!"

Such commands are called unconditional instructions.

If we apply these unconditional instructions, then we are in desktop computer mode and can now count on a calculator. *Commands in desktop computer mode are written without a line number and are executed immediately after pressing the RETURN key.*

Now we've told you how to turn your computer into a SuperComputer. Maybe it would be appropriate to show you how to expect it next.Imagine you wanted to pull two numbers together. We are used to writing numbers on a piece of paper to add up. And what that means is that most of us are used to seeing these symbols for mathematical functions.

Addition or plus, represented by +
subtraction or minus, represented by

Multiplication, represented by X
Division, represented by :

If you want to calculate with your computer, you will have no problems with addition and also with subtraction, since the function signs are the same. But what if you made the usual multiplication sign "X" in an input? Right, the computer would consider the character the letter "X" and would get confused by it. So if you want to multiply the computer, you have to use the following character: "" (The star also looks a bit like an "X").

The computer's divider also looks slightly different, like this: "/". For example, if you wanted the number 56 divided by 8, this would look like this: 56/8.

Your computer can even potentiate! Now, most of you know how to potentiate with pencil and paper. For example, the fourth power of ten is written like this:

$+o'$

Where 10 is the base and 4 is the exponent. (This is actually only an abbreviation for 10X10X 10X 10.)

It would be difficult for the computer to display such a calculation on the screen. So we use a special character to abbreviate the compute operation, which is: **"t"** (high arrow). This symbol is inserted between the base and the exponent (in our example between the 10 and the 4).

Typing the following characters will cause the computer to calculate the fourth power of 10: 10 1 4. (Think of it this way: The arrow is used to tell the second number where it belongs.)

We mentioned that the computer can be turned into a super computer. In fact, he is able to perform "super" -complicated bills. But in solving complex arithmetic operations, it follows a specific order of operations:

First of all: It potentiates from left to right.

E.g.: in Operation $3t212$
The computer calculates 312 first (that equals 9);
then he calculates $9t2$ (that's 81).

Secondly: It multiplies or divides if such operations occur (again, from left to right).

E.g.: For operation 6 + 3*4 + 6/3
The computer calculates 34 (returns 12) and 6/3 (returns 2). Then (!) it adds 6 + 12 + 2 (and gets 20 as the final total).

By the way, you might want to work with brackets. If so, then what is written within the bracket will be considered a self-contained partial operation and will always be performed (!) before all other operations.

E.g.: Operation 18/(3 + 3).
The computer first calculates the bracket (3 + 3) and receives 6. The remaining operation is 18/6 (returns 3).

How to get your computer to tell you the results

In the last chapter, we asked the computer to show something on the screen. Do you remember the BASIC word for it? It's PRINT, and we put quotes in front of and behind what we wanted to show.

Let's try a simple calculation, two + four. First, type:

PRINT "2 + 4" RETURN

See what happened? Your computer did exactly what you wanted. He printed what's inside the quotation marks. But the problem is, we do not want to see the task but the result!

To get the computer to perform the computational operation and display the result, simply use the PRINT statement without the quotation marks.

You want to try it? Type CLS to delete the screen (it's almost like using a blackboard sponge). Now, type your operation again, and it will:

PRINT 2 + 4 RETURN

A 6 has now appeared on the screen, and that is of course your result.

Do you want to bring some elegance to solving your mathematical problems? We can do that by using both types of PRINTAannotation. It's just a matter of putting your

Operation in a short programme, such as the following: 10 PRINT "2 + 4="

20 PRINT 2 + 4
30 END

Can you imagine what that will do? Tap it in your computer. Don't forget to apply the SPACE key for spaces and the RETURN key at the end of each line!

You got it? If you don't trust your own typing, then type LIST to check it (Isn't it convenient to have these helpful commands at hand?). If everything is clear, you can start the programme with RUN. (Oh yes, if you want to enter CLS (delete screen) first, please. A lot of people like to do that, because it looks a bit nicer.)

Now the screen should display:

READY
RUN
2+4= 6
READY

The programme caused the computer to:

Row 10 said: "Show the text between the quotation marks exactly as it is on the screen." Row 20 said: "Run the calculation and show the result." Row 30 said: "That's it for today."

That's pretty sharp. But do you think there is a way to make both the task and the result appear on a line? Do you think we would have mentioned that if it weren't possible?

To return your programme to the screen, type LIST again. Good! And can you still remember the editing arrows we were talking about? Now apply it to move the blinking cursor *to the end of line 1 O after the second quotation marks.* (Don't forget to hold down the CTRL key all the time you move the cursor!)

When the cursor is where you want it to be, type a barb (this looks like this: ‘;’). Ready? Now, type RETURN. The cursor should have jumped to the beginning of line 20. Use your arrows again to move the cursor to the right

Location below the bottom line (READY).

And just tap the CLS key and you are ready to run your fried programme with RUN. As soon as you do, it looks like this.

```
READY
RUN
2+4=6
READY
```

Yeah, that's much better. Who would have thought that a simple stick could be such an important tool? He caused the computer to continue writing on the same line after displaying row 10.

Now packed and the adventure lust

The more we tell you about this wonderful little computer, the more you will want to try out new applications yourself. Let's try to solve the same math problem again together, but this time in a much more detailed way.

Tap NEW to delete the saved programme. And now type this programme:

```
10 PRINT "COMPUTER, ADD 2 AND 4"
20 PRINT
30 PRINT "OF COURSE, MASTER!" 40 PRINT
"THE RESULT IS";
50 PRINT 2+4
60END
```

Clear the screen (just for order), tap RUN and hold fast! Who would have thought a computer could be so polite? Look at what the computer says now:

```
READ
Y RUN
COMPUTER, ADD 2 AND 4 OF
COURSE, MASTER! THE RESULT IS
6
READY
```

Did you notice the prank we played for you? There, where line 20 was active, nothing is printed. When you typed line 20, it looked like this.

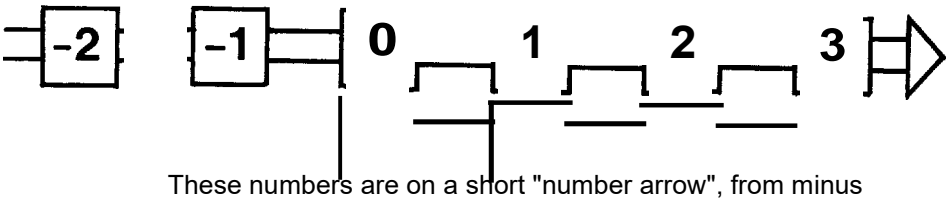
20 PRINT In other words, you caused the computer not to display anything(!) in this line, and that's exactly what it did. He made an empty line appear. You can use this trick to give the print image a nice shape.

6 Constant and variable

Sooner or later, every programmer will have to know the difference between constant and variable things. We should go ahead and know a little earlier.

What is a constant?

If you call something constant, you mean it doesn't change: The value of the whole remains the same. Here is a small example of better illustration:



These numbers are on a short "number arrow", from minus three to plus three.

You notice that each number has a traditional place on the number line. Where they sit depends on their value, and they can never leave their place! Take the number one, for example. The 1 always has the same value and always sits exactly where it is now. There is nothing to change.

Maybe you will say: "I can change the value already, I just need to add a 1 to the 1." All right, let's see what happens when we try.

The representation "141" is actually only an abbreviation for: "Start at number 1 and shout for a number along the number line." Of course, if you do that, you get to number two.

But you have not made the 1 a 2 for a long time. They just moved along the line to a new number. The 1 is still sitting there and has not changed at all.

Constants can be added, subtracted, multiplied, and divided. You can actually do all the computational operations with them. But the constants do not change as a result, only new constants emerge!

And the variables?

Have you guessed what we'll tell you about Variable? *A variable is something whose value can change very well. And* it's you who makes these changes as often as you want.

Imagine that your computer's memory is made up of lots of individual car parks, like a high-rise car park. Suppose everyone in the garage could park their car where they wanted to. This would, of course, lead to a scramble and the best parking spots (where the exit is closest). In order to avoid this and not to disadvantage anyone, the individual parking spaces are usually marked, i.e. assigned to the individual owners. So with the marking of the parking spaces, it's determined where the car is parked.

When you create a variable, do the following:

1. You *mark* the space in the computer's memory by assigning it a *label*.
2. You park something in the car park.

The markers for the variables can be:+ Any letters (from A-Z).
+ A combination of several letters of AB, IQ, MT, etc.)
+ A combination of a letter and a digit of 0-9.
(A3, YO, R2, etc.) Only note that the letter is the first in the order. 4U, 2Z and the like are not recorded by a computer (!).

Marking a variable can be longer than two characters, but the computer will read and distinguish only two.

> For example, if you mark with BEAUTIFUL, the computer will read only SC. So the variable is marked by the computer as SC. Other examples include: Labels such as PETER, ANNA or ABRAKADABRA enter the computer as PE, AN or AB.

Warning: The computer has a certain number of "reserved" words. These special words, such as reserved tables, are already

other applications. The use of such a designation is not allowed! (See Appendix C for a list of the names already used.)

We recommend that you use single letters if you do not have an immense variety of different variables. It's actually much easier as well.

Keeping memory in the computer

We're using a new BASIC word called LET. Consider the following example:

LET A=4 The first part of this statement says: "Mark a disk space named A." The second part says: "Save the value of 4 to disk A."

Do you remember what I said about variable? Your value can be changed at any time. The value of A can be changed to, for example, 7:

```
LET A=7
```

Your computer will locate the space in its memory called A. There is only one space in the space. So the computer will save the new value (7), but forget the old one.

Simple, right? *The value of a variable can therefore be changed at any time by assigning it a new value.*

And now some surprising news: There are some abbreviations for a programmer, and here's one of them.

If the computer receives instructions to create a variable, *the LET command is not necessarily necessary.* It can be neglected. The computer will understand you without LET. LET is only there to tell us humans what the command does. So it's equivalent.

```
LET A=7 A=7
```

7 More Math Magic with INPUT

It's good that you made it through the last chapter. We are pleased to hear that, because we now know that you are familiar with the LET statement. This statement is used not only to mark memory in the computer's memory, but also to assign value to that memory.

This is just one way to create a variable. In this chapter, we'll hear about another BASIC statement that does something similar. Only the way it does it is different from the way it has been treated. The expression for this statement is INPUT. That's the difference.

LET A=T7 says:1. "Mark a disk with A." 2. "How to change storage space A to 7, immediately."

INPUT A says:1. "Mark a space with A".2. "Waiting for the value assignment to disk space A, wait where you are until a value is typed from the keyboard."

Do you see the difference between the two? If the computer encounters an INPUT statement while it is running a programme (RUN}), it will stop and wait (quite patiently) until it receives the value assignment for A from you via the keyboard.

The Knowing instructions is extremely useful. Want to know Why? Then we'll show you now!

A dance with a once

Almost everyone we know hates having to say one thing. You remember, it looks like this.

1x4 = 4
2x4 = 8
3x4=12
4x4 = 16, etc.

General moaning. But don't be afraid! Your loyal computer is ready to use. After we have finished this chapter, you will never have to do a one-off, because we will have trained your computer to do it for you.

So let's start! The programme is not difficult at all. We will work with you step by step. Before we start, we need a multiplier. Let us first take something simple, for example the 4.

The first for our programme will be to have a number multiplied by the multiplier (4 in this case). Wouldn't it be nice if your computer asked you about it politely? We already thought so. Maybe the first line of the programme should look something like this:

10 PRINT "WHAT NUMBER, PLEASE" **31**

Now to line 20: Now it is time to activate our new BASIC statement "INPUT". Type the following line:

20 INPUT A

(Please note: When the programme starts in the computer, a stop occurs at this line, where we later enter a value assignment for A.

The next line should show the operation and the result on the screen. That's easy! We have already done so in chapter 5. Type the following in your computer:

30 PRINT A;' X 4 =';

There are two sticks in this line. If you noticed them, so much the better. We know what the second does, but what about the first? Relax, it's just another trick to make things easier. In this row, the computer should display (with PRINT):

The number that you assigned to Aper INPUT.

The small thing, which, clamped by the quotation marks, follows. Now, by placing a bar between these two, you can use the same PRINT statement for both, which will make them appear in the same row one at a time.

You probably know what's next without us having to tell you. It is the line that causes your computer to calculate and tell you the result. And this looks like this.

40 PRINT A*4

Of course, this says nothing more than "multiply what's in the Aist store by 4 and show me the result on the screen."

But this should be a complete one-off and we will not settle for just one result! You will probably want to multiply other numbers by 4. So why not have the computer ask us for the next number? Here's how we do it.

50 PRINT "ONE MORE NUMBER"

How do we get the new number in storage space A? Said, done, the next line of our programme will do that. In line 60, we will use our old friend, the GOTO statement. Tap:

```
60 GOTO 20
```

This GOTO statement is really something wonderful, isn't it? It will send your computer back to line 20 in a flash, and there it will wait until you come up with a new number to put in space A.

> Remember what happens when you tell your computer to add a new value to a specific location. First, it creates a new value for the variable. Then he removes the previous value and replaces it with the new one.

It's time! Your own programme to calculate the 4-series one-off. But why should you rely on our word and believe that it works if you can convince yourself of it in-, which you start with RUN. Then go ahead and try (and don't forget to delete the screen with CLS first).

Wait, what's that? The only thing you see on the screen is:

```
READY
RUN
WHAT NUMBER,
PLEASE?
```

We warned you! The question mark in the second line of your programme will only show the INPUT message you should note. All you want your computer to do is give it a number that it wants to multiply by four. Well, there's no reason to wait. Let's figure out a number. How about two? Type it and press RETURN.

```
READY
RUN
WHAT NUMBER,
PLEASE ?2
2X4 = 8
ANOTHER
NUMBER?
```

So it works! Think about what happened. The programme has pulled a loop back to the IN PUT message in line 20 and now the computer asks for a new number, which it wants to multiply by 4. And he'll keep pulling loops, and he'll keep pulling loops.

ask for another number until you are tired of the game. (Or until dinner calls).

So try some more lines just for fun. You may want to try another one, the one-off nine, for example. easier done than said. You only need to change some lines of the programme.

First tap BREAK, which stops the computer immediately. Then LIST your programme on the screen. Now use the edit arrows to convert the "X 4=" in line 30 to "X 9=". Then change line 40 to A9 instead of A4.

Do not forget to press the RETURN key after each line change.

Everything done? Move the cursor down again (so that it flashes under READY). Type CLS and then RUN. Then your new programme will cause the computer to multiply each number you enter by 9.

Now a little programme for the often inventive, sluggish types. You know who we mean. These are the ones who think it's too much effort to have to keep typing a new number with INPUT. Believe it or not, this small programme will cause your computer to increase the value of A on its own initiative each time you loop again. And that is fully automatic.

```
10A=1
20 PRINT A; ‘X 4=’;
30 A*4
40 A=A+1
50 GOTO 20
```

If you run this programme with RUN, you don't have to do anything! (Except, of course, to sit back and look in amazement.) Here's how it works.

Row 10 stores the value 1 in a variable named A. Row 20 shows the calculation task on the screen.
Row 30 performs the operation by multiplying A (initially = 1) by 4, and displays the result.
Row 40: here is the secret of our automatic one-off. This line says: "Add 1 to the value stored in A."

Row 50 returns the programme in a loop back to line 20. When you reach line 20 for the first time, your computer will add 1 to 1, giving A the new value 2. Next time, the value changes from A to 3. Then on four, then on five, then on six...etc. The computer will continue to increase the value of A by 1 until you cancel the programme with BREAK.

But you have taught your computer a very practical trick.No matter what number you want to create a one-off, it will always work and you can be proud of yourself for having mastered it, but now it's enough, this chapter was really quite difficult. You probably want to rest a bit now (and play a little with your new programming skills). So let's leave you alone for a moment.

8 Decision - making

Now it's time for us to turn to something very exciting. We will learn about one of the most powerful BASIC instructions you can use in your programmes. This statement is called IFTHEN-ELSE (if-then-otherwise).

As you expand your knowledge of programming, you will find that this statement offers amazing possibilities.

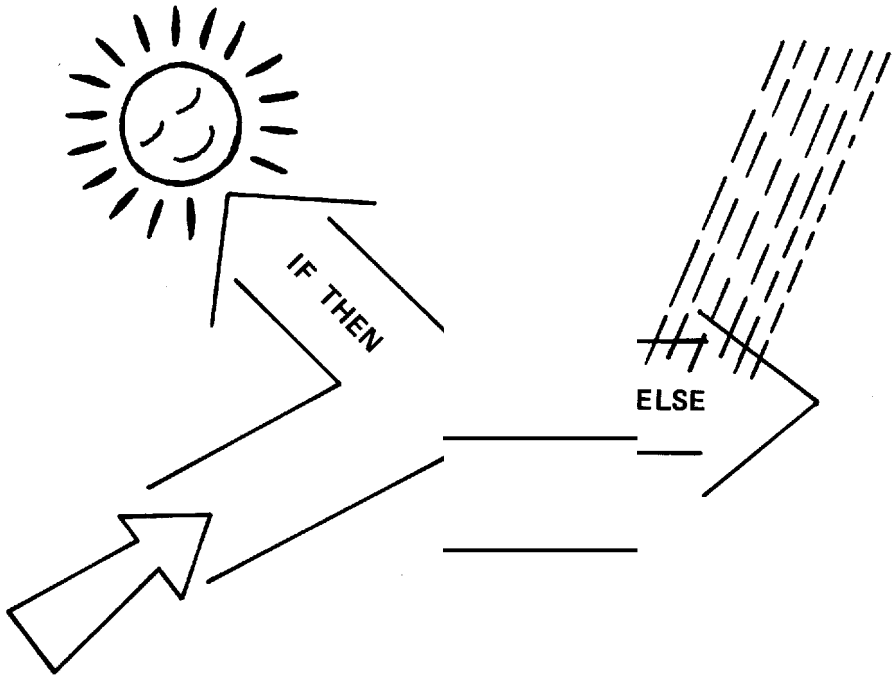
Saving a programme to your computer is like writing a jogging path on which your computer will run. And if you tell him to run (RUN), he does just that. It starts at the beginning of the programme (remember? This is the line with the smallest digit.) and runs and runs until it reaches the end of the programme.

If you include an IF-THEN-ELSE statement in your programme, the computer will detect an fork in the path on which it is jogging. Now this is a little confusing for the computer, Suddenly there are several ways in which he could continue jogging and how should he know which way to continue running? He doesn't know that by himself. You have to help him with information to make a 'decision'.

Here's a simple example that makes it easy to understand. Let's say you're walking on a suburban street, on your way to a friend. The area is still unknown and you

So, you have to follow the signs to get your bearings and find the right path.

So, you're walking there. But suddenly, you're faced with a fork in the road that goes left into a dirt road and right into a tarred road. Which way is the right one? You need additional information to decide. Unfortunately, you have no idea and need to get information from somewhere. Don't worry! Because right at the crossroads there is a sign that says:



IF (IF) it does not rain, THEN (THEN take the dirt road.
OTHERWISE (ELSE) take the tarred road.

And there's your answer! To find the right way, all you have to do is investigate the circumstances, in this case the weather conditions. So, how do you study the circumstances? Of course by checking you!

If you look closely at the message on the board, you'll see that it's divided into three parts. You see them? One part starts with IF (IF), the other with THEN (THEN) and the last with ELSE (OTHERWISE).

IF (IF) it does not rain,
THEN (THEN) commit the dirt road (=GOTO),
ELSE (OTHERWISE) walk the tarred road (=GOTO).

To solve the problem of which path to take, do one of the following:

Part one:
If it doesn't rain. ..

You should ask yourself: "Am I getting wet?" If not, it is not difficult to conclude that it is not raining. And if it doesn't rain, then that part of the message is correct. So you have to go directly to:

Part two: ... THEN commit (GOTO) the dirt road.

And you do exactly what this part tells you to do.

But what if you asked yourself: "Am I getting wet?" And suddenly you realise: "Man, that's right! It has to rain!" So, if it's really raining, the first part of the message can't be right. And if it is, then you must not follow the instructions in the second part, you must do something else, namely:

Part Three: ... ELSE (OTHERWISE) commit (=GOTO the tarred way.

And you have to do what this part tells you to do instead.

So far, we have dealt with the case of a human being on the forked path. We have a good reason to explain this to you. We do not want to say anything other than that the computer would react similarly (with some exceptions) if it were in the same situation.

Difference No.1: If we want to use the IF-THEN-ELSE statement to force your computer to make a choice, then we have to give it two things against which it can draw a comparison. These things can be either variables or numbers. Computer people call them expressions.

Difference No.2: When you were on your way to see your friend, you had checked the weather conditions. There are a variety of other conditions that could influence your decision in other situations, the choice can be infinite.

However, the computer has a very limited choice. We call them 'relations' (compare

che), because what your computer really does is it makes comparisons between two things. Here is a list of his comparison options:

= equal
<> unequal
< = less than or equal
to >= greater than or
equal to < less than
> greater than

The feeling of power is wonderful! Now you have a means in your hand to force your computer to 'get over something' without first asking you. As you might have guessed, this tool also helps to relieve you of some strain while the computer does more work. What are electronic slaves for?

If you build a fork into a programme, then IF and THEN are absolutely necessary. *But with ELSE, it's different. This statement can be omitted.* The ELSE statement belongs in the same programme line immediately after IF-THEN. If the IF part of the statement turns out to be incorrect, the computer will ignore the THENT part and instead it will fall down to the next line of the programme and continue editing there.

Here's how it works.

10 If it does not rain THEN GOTO the dirt road
20 GOTO the tarred way

Now rest a bit and read this chapter again, just to make sure you understand the IF-THEN-ELSE statement. Because in the next chapter we will show you how you can unfold your added power!

9 games

If you've gone all the way with us so far, you've already learned a lot about your little computer. You probably don't see your computer as a mystery anymore, but as a friend. Anyway, we hope so. Well, why do you need friends? For one thing, you can have a lot of fun with them. So we're going to tell you in this chapter how to play with your computer.

We have to admit, however, that there are many games where the computer plays quite badly. For example, you would never see a computer at the on-hockey or ping-pong. Monopoly is a little too heavy for him. But there's a game that the computer likes to play, and that's a guessing game.

Select a number, select a number

Finally a chance to try the new powerful IF-THEN statement (no, this time we don't need the ELSE). Before we go any further, figure out a number, any number. You got it? Well, keep them to yourself. This will be our secret number, and nobody, except your computer and yourself, will know it. (Don't you like secrets too?!)

And on in the text! We're going to write a programme where one of your friends is going to guess your secret number. Look at the programme below. There are no pits in there! We have already discussed every command that is found there. You may already know with a glance how it will work. We're going to do it line by line. Tap each line as specified.

```
10 A=7
20 PRINT "GUESS MY NUMBER"
30 INPUT B
40 IF B=ATHENS GOTO 70
50 PRINT "WRONG! TRY IT AGAIN" 60 GOTO
30
70 PRINT" YEAWEEL, THIS IS
HER" 80 END
```

Ready? You get faster and faster with the keyboard, right? Now, because this is the longest programme you've written, it might be a good idea to LISTen it. You do that now.

Here is another practical advice. So far, you have always pressed CLS to delete it after the on-screen LISTen of your programme. Wouldn't it be better if we put the command CLS in the first line of your programme? In this way, the computer will first delete the screen when you start the programme with RUN. Yeah, we thought you'd like that!

Tap the following line under line 80 already on your screen:

```
5CLS
```

Now, see why we leave gaps between the lines? 5 is less than 10, so our new line will become the first line of our programme. You don't believe that? Then just press LIST and look at it. See? At the top, the first line of our programme is CLS. Let's turn to our game again.

programme analysis

Row 10: In this line, we whisper our secret number to the computer and tell it to put it in storage space A.

Row 20: Prints a message into the screen that asks your friend to guess the secret number.

Row 30: The computer will mark an additional space with B. Then he waits for your friend to come up with a number. Then, when he types the number, the computer saves it to disk B.

Row 40: The IF-THEN statement becomes active! There are two variables in the computer's memory. In A is the secret number, B contains the number of your friend. When the computer reaches line 40, it stops to compare the content of A with that of B. Remember: The IF-THEN statement forms a fork in the programme and the computer must decide which way to go. If B is Aist, the computer will continue to process the programme in line 70, with a message appearing on the screen saying that it was correctly guessed.

But if B is unequal then the IF part of the IF THEN statement is incorrect and the computer will ignore the THEN part and continue in programme line 50.

Row 50: Prints a message on the screen that tells your friend that he has misadvised and prompts him to try again.

Row 60: This line will take the programme back to line 30 on a loop, ready for the next input of a guessed number, which will be taken back into space B. If your friend keeps misadvising, this line will always draw a loop back to line 30, ready for the next attempt. Your friend can advise as often as he wants.

When he finally pulls the big draw and the number in memory space B is equal to the one in disk space Aist, (GOTO) the computer jumps to line 70, prints the corresponding message and then ends the programme in line 80 and simultaneously the game.

You can't wait to try this game with someone? Never mind, we didn't expect you to wait. You should try to find a friend (or some) who wants to try the game.When you are done with the game, we will still be here. The IF-THEN statement has made this programme really great. And in doing so, we applied only one of the comparison options that your computer has that it can use to determine its path. How about we make some improvements in the guessing game? We already have a certain feeling that you will agree. Press LIST to return your programme to the screen.

First: Modify line 50 as follows

```
50 IF B>A THEN GOTO 56
```

(To change it, simply move the cursor to the beginning of the existing line 50 and tap over it. Do not forget the RETURN at the end of the line.}

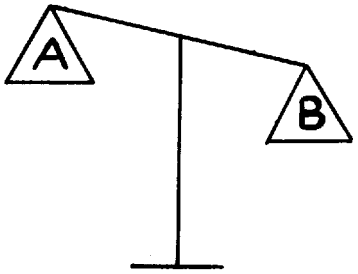
Once you have that, move the cursor down to the last line and type the following lines:

```
52 PRINT "TOO LOW!" 54
GOTO 30
56 PRINT "TOO HIGH!"
```

Again, the remaining gaps between the line numbers prove to be very practical. Like we said. Type LIST again. Your computer will sort the rows according to your numbering so that your new programme should look like this:

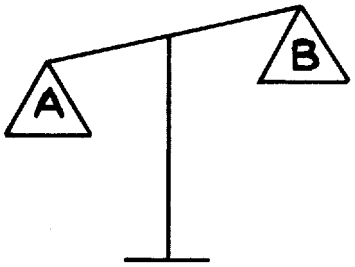
```
5CLS
10 A=7
20 PRINT "GUESS MY NUMBER"
30 INPUT B
40 IF B=ATHENS GOTO 70
50 IF B>ATHENS GOTO 56
52 PRINT "TOO LOW!" 54
GOTO 30
56 PRINT "TOO HIGH!"
60 GOTO 30
70 PRINT "YAWOHL! THAT'S HER!"
80 END
```

This programme will run the same way as the last programme, until line 50. If the number in B is equal to the secret number in Aist, your computer will immediately break to line 70. But if B is not equal, the computer goes to the next line of programmes. And this is where it gets interesting! Since we changed the programme slightly, we have a new modified line 50.



In our high-frictioned programme, line 50 still contains an IF-THENE fork and the computer still has to make a comparison between B and A to make its way decision. This time we want to check if the number in B is greater than the number in A. IF SO, the computer will continue with line 56. 56 prints a message saying the election was *too high*. Then it goes one line further to line 60. 60 performs a loop to 30 to wait for your friend's new choice.

But if B is not greater than A, then the IF part of the second IFTHEN statement is incorrect. So the computer will continue its way to line 52.



Now we know that B is unequal to Aist. And that belly is not greater than A. So when your computer is done with line 52, B can only be one relative to A, which is smaller than A. In this line, the computer is told to display a message saying that the choice was *too low*.

What's after line 52? Row 54. And this line brings the computer back to 30 on a loop, ready to receive the next input.

Now you have a programme for a rate game, which helps the guessing even with small hints!

10 More games

No matter how many friends you have, it will always be the case that you will not find anyone who wants to play with you and your computer. Well, you don't necessarily have to wait for someone to find you to play with.

The rate game from the last chapter is not really fun playing alone. Not for long, anyway. Because you know the secret number. So you could call the right solution right at the first guess and the game would be over immediately. But we could fix that little beauty bug. We would just have to make a small change to the Chapter 9 programme.

> And again, it's time to learn a new BASIC word RND. This command means: "Imagine a number, but don't tell me which."

We already know what you're thinking. And if this new trick can make the computer think, it must be magic! Well, it's not quite, but it's pretty close. The RND command asks the computer to pick a number randomly from the air (like a magician pulling a rabbit from a cylinder hat).

If there's something very numerous, it's numbers. Numbers are like sand by the sea, more than that! There are millions, billions, trillions! So what if your computer picked out any number? You could spend months figuring it out.

Fair is fair, so let's narrow the range of numbers to choose from.

Immediately after the RND command you have to insert a number in brackets. To illustrate, let's just take the number 50. The RND command looks like this:

```
RND (50)
```

Do you know what this command commands to your computer? It says:
"Choose any number, it only has to be between 1 and 50!"

In other words, your computer now has 50 numbers in its hat, and it will draw one of them, which will serve as its secret number.

Oh, that makes things a little easier. Now you know the number you're supposed to guess is between one and 50. Of course, it is up to you what number you put in the bracket. If you want to make it a little easier, you could take a smaller number, 10 for example. If the guessing game is to be a bigger challenge, then take a bigger one, for example 500.

Now back to our programme. If it's still in the computer's memory, so much the better. We just have to change row 10, and this way.

```
10 LET A=RND(50)
```

The new line tells your computer:

"First, mark a disk space with A. Then select a number between 1 and 50 and save it there."

If you had turned off your computer in the meantime, just scroll back to the last chapter and type it again. Do not forget to enter new line 10.

And now you have it! A rate game that you can play with your computer when no one else is around. And you can play it as often as you like, it just requires a RUN, and your electronic friend reimagines a secret number that you can guess.

Some about 'Features'

The word RND can make the whole thing much more interesting. After we have already introduced it to you, we also say what it is. RND is a 'feature'.

What would be your answer to the question of what a function is? If you're not mathematics-orientated, you might say it might have something to do with radios or broadcasters, because they're also irrelevant.

Like with radio! But if you're mastering some mathematics, you'd probably say it has something to do with activity, efficacy or dependence.

There are different functions. And they serve different purposes. If we use a number in function, the rule of the function in question, which will perform its own calculation operation with the number, will then (except for certain exceptions) come out a different number or value.

RND is just one of these different features that you can apply to your computer. And so, many of these functions can be very useful, especially for people who do higher mathematics.

But of course, not everyone will want that. Have you always been a passionate math hater? If so, just turn the page very quickly. But please, not yet. Listen for a moment.

We're going to tell you about a different function. Because at some point this function will be useful to everyone (yes, to you too). It's called SQR, and it's used to pull the square root.

Take a number, say 16. The square root is drawn with SQR like this:

```
PRINT SQR(16)
```

If we type this line, your computer will display the result 4 on the next bottom line.

Now we will show you the functions you can have in the following list. If you understand math, read it through. If not, make yourself a cup of coffee by now!

A list of numeric functions

Function	Operation
ABS (X)	Is the absolute (positive) value of X
SGN (X)	-1 if X negative +1 if X positive 0 if X = 0
SQR (X)	
LOG (X)	
EXP (X)	Makes the values high X, where e = 2,71828
INT (X)	Returns the integer part of X
RND (X)	X must not be negative.

SIN (X) COS (X) TAN (X) ATN (X)	the value of the trigonometric functions is is expressed in rad, where X may be in the range of values from -9999999 to 999999. Gives ARC TANGENS in rad.

11 How to break off activities?

The more we discover about computer programming, the more exciting it becomes. For breathtaking sharks, we want to move back to normal ground in this chapter. So relax, take a deep breath, because now we'll learn a little more about our old friend "Looping".

But before we tell you what you don't know, let's think again about what we already know.

> The first loop that we met was applied to the non-ending carousel with GOTO to the programme start, by applying the BREAK statement, you know, the way that goes to no more loops pulls. And if we hadn't shown you how to pull the emergency brake, your computer would still pull loops today We've also explained the STOP statement that causes the computer to pull the brakes itself.
>
> Now, how was that with the loop when we wrote our rate game programme? Yeah, right, the loop came to an end, and last but not least. If the first INPUT was wrong, you could always make new entries until you had guessed the secret number. And when you found it, that was the end of the ride.
>
> But since we did not know how many attempts would fail until the right number was guessed, we could not say how many times the loop would pass!

Wouldn't it be convenient if you could limit the loops in their number? It's like giving your computer a ticket to tell the computer how many loops to pull until it has to "get off." Now it happens that you can programme loops in two ways that have a number boundary.

Limited loops using IF-THEN

Yes, where would we be without the wonderful instruction called IF-THEN. Here's another way it can help us. As you know, IFTHEN makes your computer make a decision. So why not order your computer to decide how often it has to pull the loop?

We promised to stay on the ground in this chapter. So we're going to try something we already know to see how the new trick works on our one-offs. (The last programme in Chapter 7).

Type the programme back into the memory of your computer: 5CLS

```
10A=1
20 PRINT A; " X4 = ";
30 A*4
40 A=A+1
50 GOTO 20
```

You know exactly what would happen if this programme was running? Of course you know! It will increase the value of variable A by 1 each time it comes through the loop.

Now let's assume we only wanted to go 12x4. That's easy. At the end of the programme type this line:

```
45 IF A=13 THEN END
```

As you know, row 45 inserts between rows 40 and 50. Now this is what our programme looks like.

```
5 CLS
10 A=1
20 PRINT A;' X 4 =';
30 A*4
40 A=A+1
45 IF A=13 THEN END
50 GOTO 20
```

Do you want to see it in action? You know what to do: Just tell him RUN. (And prepare for a leap of joy).

It works!

Thank you IF-THEN, we just have our first limited loop

programme. And how did we do that? You might have guessed it.

> By adding line 45, we have caused the computer to check each new value of A there with IF. While A was still smaller than 13, your computer simply rushed through the loop again.But: IF A (when) was no longer unequal 13, THEN (then) he decided to be the loop and thus simultaneously the programme.

See what this programme has done? It gave your computer a ticket for the Looping Carousel, and that ticket reads: "Turn 12 times and then get off."

> So much for method 1. It worked wonderfully, didn't it? You're probably so impressed that you can't wait to know Method 2.

But, don't get impatient! Before we show you the second method, we have to teach you three brand new BASIC Statements. And this has to wait until the next chapter, because you've done enough for now.

12 Still with "LOOPING": The FOR-TO-NEXT

See the BASIC statements at the beginning of this chapter? In this chapter we will show you the second way how you can give your computer with FOR TO NEXT a ticket that tells it to draw a limited number of loops.

First of all: FOR-TO

The first thing the FOR statement says is: "Mark a space in your memory with the following".

> (In this exercise, we call the variable number A. But you can add any other letter.)

Does that sound familiar to you? Well, it should, because the good old LET statement does exactly that! But there the similarity stops.

The LET statement gives the computer only one number or value to *store*, such as:

```
LET A=7
```

On the other hand, the FOR statement can give the computer an entire range of numbers (e.g. the numbers from 1-12) at the same time.

> You may now try to write the FOR statement as follows:
>
> FORA = 12 3 4 5 6 7 89101112
>
> Please don't do it. It just won't work. And if you were to do it anyway, it would be rather burdensome. Especially if you took a large number range. (Imagine writing a FOR statement for the numbers from 1-100.) No, we have a better idea.Would it not be easier to use the word TO to write your FOR statement like this:
>
> FORA=1 TO 12

Yes, that makes a lot more sense. "Sensible or not, it won't work!" some of you will say. "All these numbers won't fit in one memory space, will they?"

You're right, it wouldn't work. A variable can never have more than one value, at least not at the same time. But don't panic yet! What FOR A=1 TO 12 really means is this:

> "Okay, computer, listen well. I have 12 numbers that I want to save on location A. You should change the value of A 12 times, where A is the next (!) number in the group for each change."

E.g.: The first value in A is 1. Next time he's 2. Then equal to 3. The values are placed one after the other on the storage place until A=12.

> (Do you realise that? It's like the computer is going up a 12-step staircase, one step at a time.)

Next up is NEXT

Aha! Now the price question: How do we make the change of that value in variable A?

We're glad you asked. See, the FOR-TO statement "opens" a loop. So at the end of the loop, we need something like a "shutter," where at the same time, the computer is sent back to the beginning, as long as it hasn't finished its job.

So far, we had used the GOTO statement to stop the loop. As you know, it's a pretty practical instruction. And if we were to use it at the end of our FOR-TO loop, it would definitely send the computer back to the beginning. There's an "but" and it's a pretty big one! The GOTO statement does not tell the computer to change the value of A.

Obviously, we need a new type of instruction. And now it is the case that NEXT is perfectly suited here.

We use NEXT to terminate the loop with the running variable A, as follows:

```
NEXTA
```

It tells your computer:

> "Go back to FOR-TO at the beginning of the loop, select the next (NEXT) value of A and start again."

Quiet in the studio, recording!

Well, now we know FOR-TO-NEXTew. Do you want to see it in action? Yeah, we thought so. We want to create a new programme for this. Why don't we brace the computer to calculate the one-time-four again, just like we learned in the last chapter? Only this time we will use the FOR-TO-NEXT statement instead of IFTHEN. Agreed? Get the mail! Type this programme:

```
5CLS
10 FOR A=1 TO 12
20 PRINT A; ‘X4=’; 30
PRINT A4
40 NEXT
50END
```

Now is the big moment! Type RUN and watch:

If you were right, then the computer should have reckoned the one-off four to4x12, just as we did with IF-THEN. Are you impressed? If so, then stay on this channel, because the next prank follows immediately (at least the next chapter!) Because there is still a lot to learn about FOR-TO-NEXT.

13 More about FOR-TO-NEXT

When we first mentioned FOR-TO-NEXT, you probably don't have

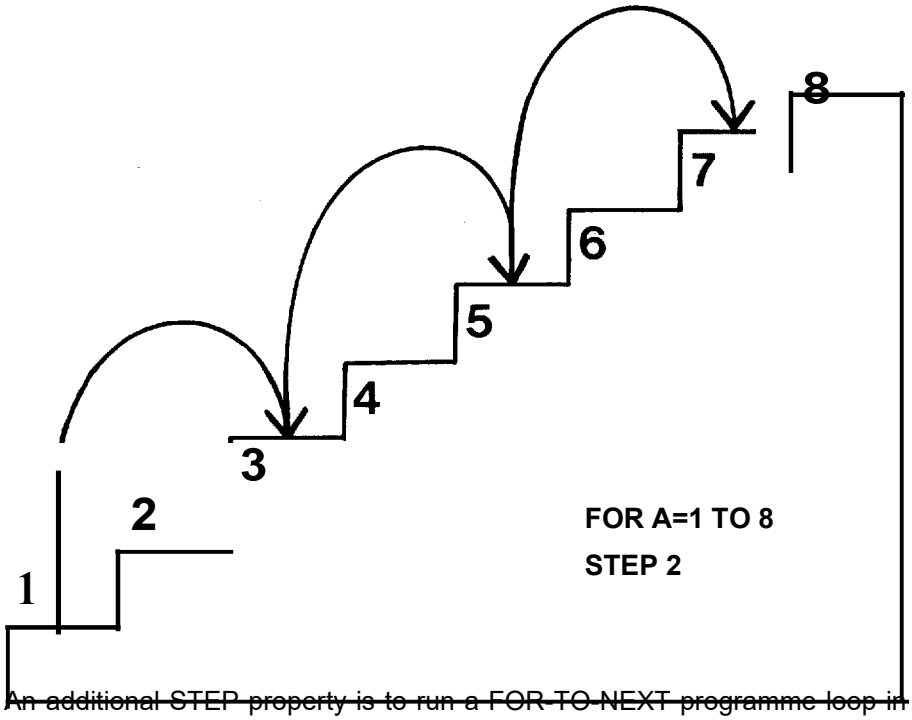
shows how powerful this statement is. And here's another thing to surprise you: By adding another BASIC word, we can make FOR-TO-NEXT beat even more amazing caprioles.

Don't you sometimes take two steps when you walk up stairs? Well, our next word is STEP, and this will enable the computer to do the same and even better than a human being can! With STEP, you can have your computer take two, three, ten or more steps with one step, if you want.

Here is an example of how to use STEP:

FOR A=1 TO 12 STEP 2

The computer then does the following:



An additional STEP property is to run a FOR-TO-NEXT programme loop in reverse. The highest number is first addressed and then further in descending order the values are changed until the smallest number, the given lower limit, arrives. Amazing! Here's an example.

FOR A=12t01 STEP-1 **51**

And this is what your computer does.

A reverse loop? We have to look at that! Type this programme:

```
 5 CLS
10FORA=12 TO 1 STEP-1
20 PRINT A; "COLOURED CONE,"
30 PRINT" THEY STAND AROUND,"
40 PRINT A; "COLOURED CONE,"
50 PRINT "THEY ARE STANDING AROUND,"
60 PRINT "AND WHEN A CONE SHAKES," 70
PRINT "THEN IT ALSO FALLS AROUND,"
80 PRINT "THEN THERE ARE "; A-1; "COLOURFUL
CONE," 90 PRINT" THEY STAND AROUND LIKE THIS!"
100 NEXT A
110 END
```

Of course, not everyone knows this little song. But on some "colourful" bowling evenings it is quite audible. It starts with 12 colourful cones, with verse after verse falling over a cone (the later the cone evening, the "more random" the falling over can be!) Every new verse starts with a cone less than the previous one and when all 12 cones have fallen over, the song ends.

For more information:

We already know what the FOR-TO statement in line 10 will do. The first value in variable A is 12, the next 11, etc. to A=1.

The following 4 programme lines show the current value of A, followed by two lines.

Lines 60 and 70 tell the observer that a cone is about to fall. Row 80 has

several tasks:

* Calculates the remaining number of cones if 1 is subtracted from A and * then displays the number in the middle of a verse line.

Line 100 completes the loop and sends the programme back to the FOR statement at the beginning, where the next value is stored on A.

If the computer has gone through the loop 12 times (and

where he has printed all 12 verses of the rhyme) he ends the programme with line 110.

So what are we waiting for? Let's do a RUN and see how a reverse FOR-TO-NEXT loop works.

Oh, that was a little disappointing! It works, but maybe it works a little too well, what do you think? When all these lines flash across the screen, it's hard to read the verses at all. Unfortunately, this is one of the disadvantages of computers. They're doing exactly what we're telling you, but sometimes they're so freakishly fast that we humans can't compete at all.

In order to slow down this programme a little, we have to make sure that the computer breaks down occasionally. In order to "buy" additional time, we have to give the computer something that it's dealing with in the meantime. During this process, we can see what's on the screen.

There's a way. Look at the line below:

```
FOR T=1 TO 3000:NEXT T
```

We know that a FOR-TO statement opens a loop and a NEXT statement closes it. So the loop shown above is empty(!), because as soon as we open it, we close it again. It has no particular executive task. It just sends the computer "up a flight of stairs" — 3,000 steps. But because your computer is so fast, we need a long staircase to keep it busy long enough.

Where do we put this loop? Well, it would be logical to use them after each line. So LIST your programme so that we can find the job.

There it is! Row 90 prints: "THEY'RE STANDING AROUND." This is the end of the verse. So we insert our void loop between lines 90 and 100 and call the programme line 95. This will cause the computer to pause before it retracts its loop to display the next line.

Try RUN to see if this helps.

Yes! At the end of each verse, as the computer spins up the 3,000 steps and nowhere to go, you have time to read the verse.

These loops are called *delay loops,* because they delay processing in the computer.

Now something about "nested FOR-TO-NEXT loops".

But that sounds awfully technical and confusing? We have a surprise for you: We have already done something like this, and we have done it now! Our "delay loop" is a "nested loop" (loop within another loop) at the same time.

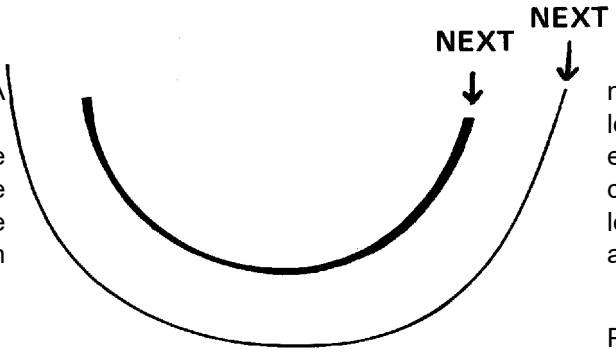
"Nested" is meant to express that one thing is entirely contained in another.

> Do you notice where we're headed? A nested FOR-TO-THEN loop is located within another FOR-TO-NEXT loop.

If we try to draw one, it would look something like this.

FOR ... **TO FOR**... **TO**

V



(By the way: A nested loop FOR-TO-NEXT loop may not be empty. It may well have content, but the whole loop must lie within another one.

Price question: Have you noticed anything unusual about line 95? No? Take a closer look:

FOR T=1TO 3000: NEXT T

Resolution: Row 95 contains 2 BASIC statements. The first is FOR-TO, the second is NEXT. In most cases, both statements appear separately in each line. But if both should appear in a line, they must be separated by a colon!

14 programmes within other programmes

Although the computer is quite clever, its capabilities have limits. In fact, the number of programme lines he can remember is limited. Because each line occupies space in the computer memory.

And if there's one thing a computer programmer hates, it's to waste space. (Especially when the storage capacity is needed for a very, very long programme.) Now that you're becoming a pro, you'll probably agree.

Do you know how best to waste space? By repeating yourself unnecessarily. You will notice occasionally, when you type a programme into your computer, how repetitions creep in.

For example, our "KEGEL" programme from chapter 13. What do you say? You thought it was a pretty great programme? We too. And it really is! But perfect is not yet, as you'll see when you look at lines 20, 30, 40 and 50.

> Really bad! Twenty and 30 do exactly the same thing as 40 and 50. This is now a good example of the waste of programme lines.

This chapter will show you how to avoid such repetitions by using the BASIC GOSUB and RETURN commands.

Close your eyes and think hard, try to remember Chapter 3. Remember how we described the programme to you? Exactly: We said that a programme is a sequence of individual commands that perform a specific task.

> Here's a very interesting thought. Lines 20 and 30 are actually a sequence of steps, although they are only two lines.

Do you see the specific role of these two steps in the programme? They tell the computer to show two lines of the "KEGEL" song on the screen. (And lines 40 and 50 are nothing more than a repetition of lines 20 and 30, because they contain exactly the same instructions.)

A mini sequence of steps? It's kind of a mini programme!

Actually, a waste of typing the same steps twice. Are you thinking what we're thinking? Maybe you could just do the same with a hidden mini-programme. Yeah, that sounds good! But for a programme within a programme, there has to be a label — what do you think it's called? A sub-programme, perhaps? Come close! It's called "Subroutine" in English, German *subroutine.*

A subprogramme is part of the main programme, but it is also separate. Subprogrammes can be installed anywhere in a programme, depending on what they are designed to do. We need to get the computer to do this.

jump to the subprogramme and perform the function to be performed there, and then

back to the next line of the main programme.

We almost forgot to tell you something: You probably can't imagine how to separate a subroutine from the main programme.

The simplest way to keep two things apart is to create an obstacle in between. Accepted? To prevent a subprogramme from conflicting with the main programme, assign line digits to the subprogramme that are much higher than the last line digit of the main programme. (The last line of the main programme contains the END statement.) Try starting your subroutine at the bottom of line 3000. There can be no conflicts because the computer beENDet is the main programme before the subprogramme is reached.

Thus, the END statement of the main programme acts as an obstacle to prevent the subprogramme from causing damage.

The creation of subprogrammes using GOSUB and RETURN.

Getting your computer to jump to a subroutine and back is no problem, you have to

just call it. There is only one catch - you have to call correctly. And that's where our two BASIC words come in - GOSUB and RETURN.

Yes, it is! GOSUB is a special type of GOTO command and causes your computer to go to the beginning of the subprogramme. And because there's no point in telling the computer to go, if you don't tell the computer where to go at the same time, *GOSUB always has to have a line number.*

To send your computer to a subroutine in line 3000, type:

GOSUB 3000

You may be surprised that we are dealing with GOSUB if GOTO would do the same.

There's a reason for everything in the programming world. In this case GOTO will not work! GOTO only says: "Jump to line 3000 - now!" But GOTO does not warn the computer that it will be sent to a subprogramme.

Because the computer is so cooperative, it jumps immediately. Without remembering where he jumped from. That would be a problem, because how would he know how to get back?

It looks like we need GOSUB, because this statement warns the computer. She says: "Watch out! You're now sent to a subroutine, make sure you find your way back here."

Now we know how to send the computer to the small separate programme. And what happens when the subprogramme is finished? Everyone knows that he will jump back to the main programme.

Well, not everyone. You know, we know, only the computer doesn't know. Like I said, the computer only knows what they're told. Therefore, you must complete your subprogramme with a RETURN command. It's like saying:

"This is the end of this subprogramme. Return to the main programme, to the line immediately following the GOSUB command that sent you here."

Now that we have understood the principles of the subprogramme (and how to apply them), we want to test our new skills and improve our "KEGEL-LIED" programme.

A subprogramme that would perform the task of lines 20 and 30 (and lines 40 and 50) would look like this:

```
3000 PRINT A; "COLOURED CONE,"
3010 PRINT "THEY ARE STANDING
AROUND," 3020 RETURN
```

When we do our song programme "up to date", the new version looks like this:

```
5 CLS
10 FORA=10 TO 1 STEP-1
20 GOSUB 3000
30 GOSUB 3000
40 PRINT "AND WHEN A CONE SHAKES," 50
PRINT "THEN IT ALSO FALLS AROUND,"
60 PRINT "THEN THERE ARE STILL"; A-1 ;"COLOURFUL
CONES," 70 PRINT "THEY STAND AROUND LIKE THIS!"
100 NEXT A
110 END
3000 PRINT A;' COLOURFUL CONE,'
3010 PRINT "THEY ARE STANDING
AROUND," 3020 RETURN
```

See how this is going to work? To make sure that you can imagine it, we would like to show you a picture!

Look at the diagrams below (you should make everything seem clear, i.e. pretty clear!).

Rows 20,30,40, and 50 of the old programme have been replaced by two lines - a new line 20 and a new line 30. Moreover, these replacement lines are shorter, which only includes GOSUB 3000.

When you start your computer with RUN for (On the Ready-to-RUN) the new programme, it starts the same way as with the old programme until line 20!

Line 20 sends the computer to subroutine 3000. Your computer flips through the subroutine and then returns to line 30 of the main programme.

Row 30 also contains a GOSUB 3000 command. So the computer goes back to the subroutine. This time he returns to line 40, ready to continue with the rest of the programme.

That's it! Subprogrammes are really easy to use. And they're a good thing, as we've just seen, to deal with the waste of storage. (It is true that subprogrammes can provide priceless services when certain sequences of steps occur again and again!)

Even better:

What if you have several different steps in a programme? It's very simple, you make subroutines to do these things as well. Subprogrammes can be used in unlimited numbers. (You only need to change each subroutine to a different row number. For example, the first subprogramme with 3000, the second with 4000, the next with 5000!)

Best: Do you remember what we said about the nested FOR-TO-NEXT loops? The same applies to subroutines, which then results in *"nested subroutines."* Example:

You can cause your computer to run from the main programme to subprogramme 300. And Sub-programme 3000 can contain a GOSUB command, which includes a forwarding to Subprogramme 4000. Another GOSUB command in subprogramme 4000 leads to subprogramme 5000!

The same GOSUB-RETURN rules apply no matter how many subprogrammes are nested: A RETURN command sends the computer back to the line immediately following the GOSUB command that sent it to the subprogramme!

15 String Variable

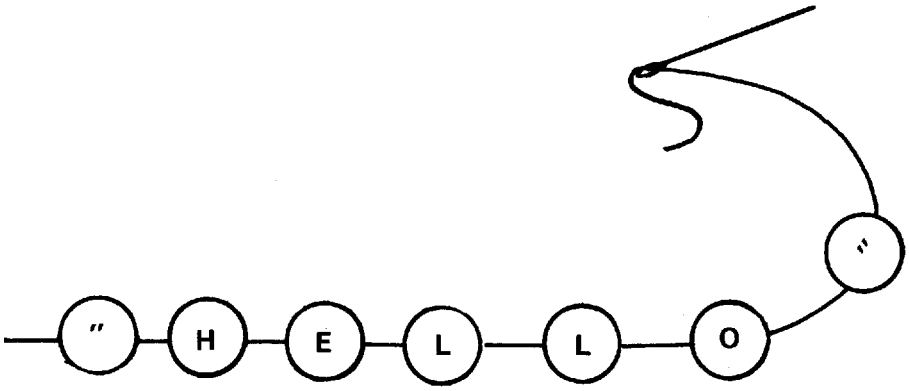
They'll love this chapter. And what it can do for your programmes, you will also love! Because now we're going to learn something new and exciting about a new type before variables.

We already know that the "parking spaces" in the computer's memory are specifically designed for storing various things. We have already saved numbers with success. But it would be a great pity if we were to use them only for this purpose.

Have you ever wondered if these storage locations can be used to store

Can I use other things? Like words or even sentences? I'm sure that would be very practical.

Now here's good news. Yes, you can! That's quite amazing, but it's certainly possible. There is even a special name for such variables. They're called the string variable. (String is called 'Chain'. We say German 'Textkette'!)



The term may seem a bit strange at first. But if you think about it carefully, you'll see that it makes perfect sense. If you put these variables in a form that's suitable for processing, you're doing nothing but putting beads on a string together into a chain. And that's pretty simple.

A data chaining is also pulled together. You just chain a certain amount of letters, spaces and other characters, like beads in a chain.

In order to create a variable concatenation, you must additionally mark a disk space on which you can then save the concatenation!

Rules and rules

As with the other new things we have learned, certain rules must be observed here. But don't worry, they're not hard to learn.

A text chain must not exceed 255 characters. (incl. Empty). Why, you ask? Well, the memory slots also have a limited amount. A lorry would not find enough space in an underground car park!

The name you assign to the text chain must always end with a $ character. (No! That doesn't mean they have anything to do with money! The $ sign simply says that there is a text chain on this location, not numbers.)

Example: A$, B$, C$ and D$are possible text chain names.

A text string must always be enclosed by quotation marks when typing into the computer! Example:

```
A$="ENIE"
B$="MEENIE"
C$="MINIE"
D$="MO"
```

The new variables must be processed in the same way as ordinary variables, an exception:

You can only apply one mathematical operation to it, which is addition, that is: Chain, concatenation.

If you want to add text chains, the computer is instructed to:

```
PRINT A$+B$+C$+D$
```

And after RUN, the result looks like this:

EENIEMEENIEMINIEMO

Do you see how the computer chained together without leaving any spaces between the variables? If you want to have spaces, then you must insert them within the quotation marks

You've probably come up with all sorts of things about how to use this trick. After all, when you mention certain variable names, your computer will "speak" whole words or phrases.

Would you like to try it? We too. Let's try to get the computer to "have a conversation" with a third person.

Type this programme:

```
 5CLS
10 PRINT "HELLO! I'M YOUR computer."
```

```
  20 PRINT "WHAT IS YOUR
  NAME"; 30 INPUT N$
  40 PRINT "HELLO, ";N$
  50 PRINT "I'M VERY HAPPY TO GET TO KNOW YOU.
NO."
  60 PRINT "IN WHICH DISTRICT" 70
  PRINT "LIVE";
  $80 INPUTS
  90 PRINT "LIKE TO LIVE IN" 100
PRINT S$;
A$110 INPUT
120 IF A$="NO" THEN GOTO 150
130 PRINT "BEAUTIFUL! IT'S MOST BEAUTIFUL AT HOME!"
140 END
150 PRINT, "OH, YES! MAYBE THEY SHOULD MOVE!"
160 END
```

If you are working through the programme, you should be able to imagine how it works.

Some tips: Line 20 asks your friend to enter his name (INPUT). Line 30 "listens" to his answer and stores the information in the text chain named N$. Lines 50, 60 and 70 do roughly the same. Your friend will be asked where he lives, the answer will be stored in S$.

If this information is securely stored in the computer, the computer uses it to conduct a fairly personal "conversation." (Maybe you'll fool your friend into thinking that the computer really has its own mind!)

A well-intentioned warning

The courtesy of your computer to impress your friends is good and beautiful. But be careful! If the computer becomes too charming, you might have difficulty separating it from it.

16 Variable in READ and DATA

Before we say more about computers, we want you to do a little exercise.

Are you prepared for anything? Then follow the instructions exactly:

1. Stand on two. Tap your shoulder. (Yes, it's a bit difficult, but it can be done if you develop some joint).
3. Throw yourself in the chest, smile and shout: "I did it!"

Doesn't that do well? And don't you think you deserve it? Don't be modest: Praise him who deserves praise. You started off with shaky steps to explore the world of computers, and by now you've worked your way through 15 whole chapters of this book.

And now... continue in the text

Since one thing usually leads to the next, we go back to our underground car park in this chapter. And what's next on our "Learning List"? In the last chapter, we talked about a new thing that we put down in the car park, the chain of text. Now we're going to talk about a new way of storing it.

"Another kind? But we already know two different species with LET and INPUT. Surely there is no extra?"

Don't be so sure. There's one more filing system, one that will prove very practical later.

You see, both LET and INPUT are very, very useful. But one day, you'll want to store a lot of information. (Especially when your programmes get longer and more complicated.) Suppose you wanted to create five variables, where you wanted something in each one.

You could do it like this.

```
LET  A=5
LET  B=4
LET  C=3
LET  D=2
LET E=1
```

... but it would take too much time and space!

So we try something completely different using just two lines.

```
DATA 5,4,3,2,1
READ A,B,C,D,E
```

It probably leaves you in the dark. Allow us to shed some light on the mystery.

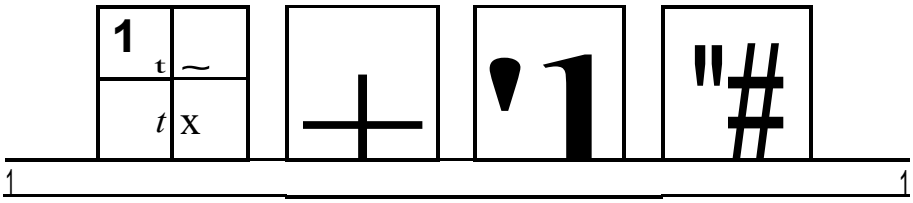
Some time ago we mentioned an important rule regarding the variables. Remember this?:

> The computer will not put anything into its memory until we assign a label to it to mark the space.

So far, we have always *provided a name for a variable* before telling the computer what to store on it. But this time, it will be something different.

The first line in the pair above contains 5 entries (in this case they are numbers). The typing in of this line is as if you were going to deliver a lot of packages to a hotel reception and say:

> "Here. .. Be careful in the meantime. Later, I'll tell you where to park them." These packages need to be placed somewhere. For example, on a shelf.



> A line that starts with a DATA statement is just like a "package shelf." This is a place where things are kept before they get to their final place. (By the way, did you notice the comma in the DATA line? They are used to keep the individual entries apart).

When it is time to remove the packages from the Regeal, the computer always starts from the left and works its way to the right. Computers prefer to work in that order, they're actually pedantic people. (read: machines.)

And now to the second line, which starts with READ.

This line is actually just a list of variable names. We want the computer to use every single name to mark an individual space. Again: *The commas are there to separate each entry from its neighbour.*

When your computer reads the READ list, it thinks: "Oh, how beautiful. Now I can start sorting the mail." So let's look over the shoulder of the computer and watch it work with this filing system.

The name at the top of the READ list is A. So: Your computer uses this name to mark a space with it. Then he takes the first package from the DATA shelf and saves it to disk space A.

Right, now to the second name on the list. This is B. And your computer will store the second package from the shelf on the storage space called B. Did you guess what lands on disk C? Yeah, the third package. And so it goes on until the last variable name on the READ list was used.

> Can you see how efficient this system works? The computer will work through the READ list name by name. Where he saves the next package from the DATA shelf to each new space he marks.

By the way, you don't have to use every single package that is on the DATA shelf. For example, let's assume that there are six entries in your DATA line, but there are only five names on your READ list. It doesn't matter to your computer. He will simply record the five packages in his memory and ignore the remaining ones.

> If we had put these two lines into a small programme, we could then add an additional line that causes the computer to express the values
>
> PRINT A;B;C;D;E
>
> And when we ran our programme, the following would appear on our screen:

```
54321
```

Remember what? If you told your computer to PRINT E; D; C; B; A, we would have seen the following on the screen

```
12345
```

There is something to keep in mind: You must ensure that you have enough data packages to store in the designated marked locations. Do you want to see an example? Then replace the second line of our example with the following

```
20 READ A,B,C,D,E,F
```

Now try running the programme again with RUN. Everything will start out just right. Until your computer gets to the last variable name in the list. It will obediently mark a disk space with F. But if he tries to take another package from the DATA shelf. Oh, dear! The shelf is empty! They named six parking spaces, but there were only five DATA packages. There is no more left for space F.

If something like this happens, your computer will say: "How am I going to do that? Can't you see I'm OUT OF DATA?"

The moral of the story':

Never use more names on your READ line than entries on your DATA line. (If you need more than one DATA shelf to save your packages, do it. You can use as many as you want.)

Here's another piece of useful information: If you want, you can only use one variable name (say D) on the READ line. Then you can drag the computer back to the beginning of the programme. (How to use a FOR-TO-NEXT loop, of course.)

Each time your computer then comes to a READ line, it will replace from earlier, the data present in D with the next from the DATA shelf.

And how do you figure out how many times the computer has to grind through? Simple! Your computer must loop for each entry in the DATA line.

To prove to you how useful this new system can be,

We'd like you to type the following programme as an example. But first we want to introduce you to our imaginary friend:

Mr Caution

Mr. Caution is a shopkeeper. Like most businessmen, he checks whether a customer is reliable or not. (i.e. he checks the customer to see if he pays his bills.)

This programme is designed to do this check for him. Because it is the case that Mr Caution allocates a special customer number to each customer. And this programme lists the customers that are called low-paying customers.

Mr. Caution can use INPUT to check every customer number he wants to have checked. And the computer will tell it whether the customer is on the "blacklist" or not.

Sounds smart, doesn't it? Let's see if it works.

```
  5CLS
 10 DATA 11067, 12549, 22871, 57824, 86256
 20 INPUT ‘CUSTOMER NUMBER’;N
 30 FOR L=1 TO 5
 40 READ D
 50 IF D=N THEN GOTO 90
 60 NEXT L
 70 PRINT "THIS CUSTOMER NUMBER IS OK"
 80END
 90 PRINT "NO! THIS CUSTOMER NUMBER IS WRONG!"
100 END
```

Something confused? Don't bother yourself. When you analyse it, you'll notice how easy it is.

Row 5:

It gives us a nice, clear screen, with which we will see after RUN all clear.

Row 10:

Our DATA shelf: There are 5 entries saved. And remember: Every "data package" is really the number of a customer who has not paid his bills.

Row 20:

We haven't had a line like that yet. This is really just an ordinary INPUT statement. But we only have it

some spice mixed by using an insert line that prompts the user to reply to INPUTs.

Row 30:

Here the loop of our data import is started. Because we have five different entries on the DATA line, we will have to grind through just as often. (We assigned the name L not for a particular reason, but only because L stands for LOOP.)

Row 40:

We've just explained what this line does. On the first loop, the READ line causes the computer to save the first packet from the DATA shelf to a space marked D. The second time it saves the second package and so on.

Row 50:

By using our proven IF-THEN statement, the customer number is saved to N for verification. If it is equal to the customer number in D, THEN then your computer will go to line 90 with GOTO. Line 90 prints a message as information that it is on the "black list". And immediately after this line, follows row 100, which the programme beENDet. But if N is not D, the computer will go to the next line and continue the programme.

Row 60:

Here is the NEXT statement that closes the loop. It sends the computer back to the beginning of the FOR-TO-NEX loop. Then it replaces the number already in D with the second package from the DATA shelf, ready to compare it with N. This line keeps the computer on the loop until it circles for the fifth time. Only then is he free to go to line 70.

Row 70:

When your computer is ready, all bad customer numbers are checked. This line will print a message saying the number is OK.

Row 80:

This one speaks for itself.

Try it yourself. The first RUN of the programme you think

If you have a customer number that you enter (INPUT):

Does it work? Great! Now try again. But this time, you enter a number that you know is on Mr. Caution's "blacklist." (Just to make sure that this part of the programme also works.)

17 Additional information on DATA and READ

We're going to tell you some helpful things about DATA and READ.

After we drove Mr Caution's review programme, what do you think happened to the five packages? We know that they were all taken from the DATA shelf and were individually stored in D. But what on earth happened to the individual packages after they were removed from D and replaced by the next?

Good question. We didn't tell the computer to put the packages back on the DATA shelf, so they're nowhere at the moment.

If we wanted to check another customer, we need to save the packages back to their original locations on the shelf. This might require a repeat of the whole programme, but it would be rather burdensome.

Here's a better idea: We could learn a new BASIC word that would store the packages back, without having to start over. The new word is called (logically) RESTORE. And for the application in Mr Caution's programme, we must add the following four lines.

```
72 RESTORS
74GOTO5
92 RESTORS
94 GOTO 5
```

Now our programme looks like this: 5CLS

```
 10 DATA 11067, 12549, 22871, 57824, 86256
 20 INPUT ‘CUSTOMER NUMBER’;N
```

```
 30 FOR L=1 TO 5
 40 READ D
 50 IF D=N THEN GOTO 90
 60 NEXT L
 70 PRINT" THIS CUSTOMER NUMBER IS OK!" 72
 RESTORS
 74 GOTO 5
 80 END
 90 PRINT "NO, THIS CUSTOMER NUMBER IS LAZY!" 92
 RESTORS
 94 GOTO5
100 END
```

Guess what it does now. When the computer finds a RESTORE statement, it puts all packages back on the DATA shelf in the original order. The GOTO command immediately after the RESTORE line simply says "Go back to the beginning of the programme".

And when the computer is back to the beginning, we can add another number to check INPUT.

Why do we need two pairs of RESTORE and GOTO lines? If you look closely at the programme, you will notice that it can be terminated in two ways. This is why we had to insert both a RESTORE and a GOTO command in the following lines.

* By line 70 (the end of the programme if the number to be checked is OK) and

* By row 90 (the end of the programme if the number is lazy) Try your

modified programme now!

By the way: Have you noticed the END statements in lines 80 and 100 of your new programme? And have you also noticed that you are not really necessary, because the programme goes back to the beginning in a loop before it gets there? To tell the truth: We have included them only because it is a good habit to close every programme. (Even if sometimes the statement is not required at all.)

18 music in your ears!

Music has been produced in many ways since the earliest days of mankind. For several centuries, we have been using music notation to write down music permanently.

> Our system of writing music is quite efficient. It allows us to come up with almost any sound we want and put it on paper. Well, this system works great for people. If a musician wants to play a melody (that's a sequence of sounds) written on paper, all he has to do is look at this:
>
> 1. The position of the notes, and
> 2. their form

And he'll know exactly what sounds to make.

Unfortunately, when it comes to reading music, your computer has a handicap for understandable reasons. To see something, people have two eyes. And the computer doesn't, of course. Anyway, even if he could watch our music, he wouldn't understand our pictorial writing system. (Computers can't think like this.)

So since the computer can't see the notes of a song, we have to find a way to describe each note individually.

The simplest way to do this is to use a coded number system with pairs of numbers. The first thing you need to tell your computer is: The note position on the note lines (the pitch).

The computer can play 31 different notes (pitches), plus one "rest". (This is called Pause in Music). In musical notation it is represented by a variety of symbols (different forms for breaks of different lengths).

Each note (pitch) has a code number for itself. The complete list of notes and their code numbers is shown below.

FREQUENCY (pitch)

Code	Sound	Code	Sound
0	rest	16	C4
1	**A2**	17	C#4
2	**A#2**	**18**	04
3	B2	19	D#4
4	C3	20	E4
5	C#3	21	F4
6	D3	22	F#4
7	D#3	23	G4
8	E3	**24**	G#4
9	F	25	**A4**
10	F#3	26	**A#4**
11	G3	27	**B4**
12	G#3	28	C5
13	**A3**	29	C#5
14	**A#3**	30	05
15	B3	31	D#5

The second number in the number pair is the note length, which is specified here as the count time in the code. Well, there you have your pair of instructions. Together they form a description of a single note.

SOUND

Code	note	note length
	J=	1 8
2	;=	1 4
3	/:"	3 8
4	F	1 2
5		3 4
6		1
7		1! 2
8	d	2
9	d.	3

It wasn't that hard, was it? Simply use the two tables above to describe the music to your computer. Well, we certainly don't recommend Beethoven's 5. Symphony to play (the computer is not so cultivated again). Please just stick to small melodies. You're really just so much fun! These are preferably found in children's singing books.

To "translate" a piece of music into a language that the computer understands, please note:

1. Determine where the individual note is on the note line and find the corresponding code number of that position from the pitch table.
2. Determine what shape this note has and find the corresponding code number from the timed table.

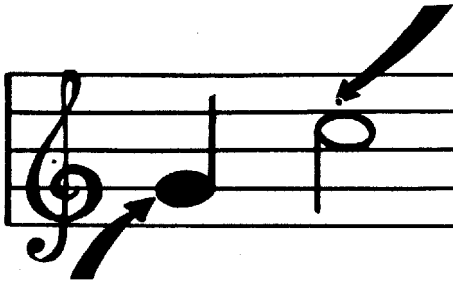
The piece of music that you describe to your computer will consist of rows of different code-number pairs, instead of rows of differently shaped notes, with different positions on the sheet of notes.

We almost forgot something very important: All these number pairs don't mean anything to the computer until you tell it what it means to do. So we're going to have to put up some kind of sign before each pair of numbers. A sign that says to your computer: "These two numbers represent a musical note. Please play it for me!"

To do that, we need to learn a BASIC word. And what's it called? You guessed it: SOUND!

The SOUND command precedes each code-number pair. Here's an example.

The code for the pitch of the first note is 28 and the code for the note form is 2. To describe it, say SOUND 28,2



The code for the pitch of the second note is 23 and the code for the note form is 6. So, to describe it, let's say SOUND 23.6

The position of the tadpole tail of the note is not important. It can hang down from the note or stand vertically up. So, what really matters is whether the note is filled in or not, whether there are any "flags" attached to the tadpole tail, or whether there is a dot immediately behind it.

From sounds to songs

How to teach your computer how to sing.

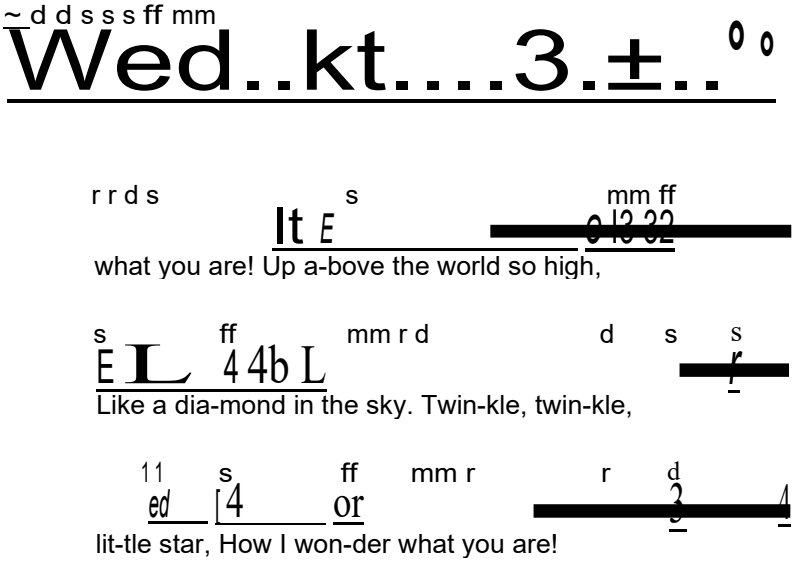
MELODIES

In the example below, a piece of music is converted into a computer programme. Track the implementation using the table.

TWINKLE, TWINKLE, LITTLE STAR

Nursery Rhyme

Key F



Because you now know how to describe individual notes to the computer, nothing can stop you! In order to use your newly developed ability, we want to try something particularly exciting. Let's try to teach your computer to "sing a song." How about a beautiful and old nursery? "Twinkle, twinkle little star" would be ideal.

What do we need to do before your computer can perform for us? We have to give him a description of every single note of the song. We use:

If you add up all the notes in this song, you'll find it's exactly 42. Does this mean that we have to use 42 SOUND commands in our programme? No, calm down, there's a convenient way out. These many SOUND commands are not necessary at all. Instead, we're using a little clever trick, one that we showed you before.

> Did you guess what we mean? Here's a little note: Two BASIC statements are affected (and, they go hand in hand). When we first mentioned them, we said they were best for storing a lot of information.

If you advised DATA and READ, you were right on the first attempt! These two statements will certainly save us a lot of unnecessary work. (Because we really have a lot of information to store.) You wanna see how? Then enter the programme below carefully.

```
5CLS
10 DATA 21.4.21.4.28.4.28.4.30.4.30.4.28.6.26.4.26.4.25.4
20 DATA 25.4.23.4.23.4.21.6.28.4.26.4.26.4.25.4.25.4.25.4.23.6 30
DATA 28.4.28.4.26.4.25 1.4.25.4.23.6.21.4.21.4.28.4.28.4 40 DATA
30.4.30.4.28.6.26.4.26.4.25.4.25.4.23.4.21.4 6
50 FOR L=1 TO 42
60 READ P,D:SOUND P,D:NEXT L:END
```

programme analysis

Rows 10, 20, 30 and 40: As you can see from the DATA statements right at the beginning, we use these 4 lines as "memory shelves". (We need 4 because we need to store so many DATA.) And what exactly do we save? Code number pairs! Each pair is a description of a note of our song. (The first number of our couple describes the pitch. The second number describes the sound time.) Count it out: You will notice that there are 42 pairs on our DATA shelf. Example:

10DATA 21,4,21,4,28,4, 28,4, 30,4, etc.

We want the computer to read these numbers (two each) (READ) and to put each on a variable storage space. Once our two code numbers are safely stored, we proceed with only one SOUNDKommando followed by the addition of the variable pair. The computer will play the sound described by the respective two numbers of memory slots.

Row 50: Our number loop. The computer has to loop 42 times because it's 42 pairs of code numbers. (For each loop, a pair of numbers is replaced by the next one on the DATA shelf.)

Row 60: This is one of the clever lines containing multiple commands, remember that it is OK as long as one command is separated from the other by a colon(:).

First command to colon: This is our READ statement. It stores the code numbers, pair by pair, on two memory slots. The first one is marked with P (here comes the pitch code); and the second one is marked with D (here is the code for the sound duration.)

Second command from SOUND: tells the computer to play the note described by P and D.

Third command from NEXT: sends your computer to a loop where the next code numbers are stored in P and D.

Fourth and last command: beENDet the song after the loop has been pulled through for the 42nd time.

Have you already typed the programme into your computer? Well, it would be a good idea to LIST the programme just to be sure that you have every single number right. (Because if you don't have it, your computer might sing something "wrong.")

And now the big moment has come, time to exit the programme (RUN). Isn't that wonderful? If you believe that standing ovations are appropriate, please! And if you want an encore, just run the programme again.

And now a particularly interesting (if completely useless) information: Your computer played you a song written by a very, very famous composer.

"Twinkle, twinkle little Star" was composed by none other than Wolfgang Amadeus Mozart.

19 Fun with graphics

Will your little computer ever stop surprising you with its capabilities? In this chapter we will reveal the hidden artistic ability of your computer. We will now learn to draw on the screen of your computer. But don't expect to be able to produce a Mona Lisa!

The art of creating images or patterns with the computer is called computer graphics. Becoming a master in this requires time and patience. But it's probably a lot of fun!

Your computer has two different modes (modes) in which it can produce graphics. The first mode is simple mode (0). In this case, it is immediately and automatically located on power-on.

Mode (0) writes the keyboard characters to the screen.

And what exactly are keyboard characters? Quite simply, these are all characters (and commands) that are on the computer keyboard. Including these weird things called "graphic characters."

These different graphic signs are similar to tiles (the kind found in bathrooms). Each one looks different, and it's 15 pieces. By arranging them on the screen in a certain way, you can create a mosaic (pretty much).

Unfortunately, the tiles (or, if you want, tiles) are quite large and clunky. This means that the images you put together with them will also be roughly rastered.

As we said, "computer graphics" is a strange term. If you look at the word graph, you might find another word in it. One that describes the whole idea quite well. It's called the graph. And it's a very useful description, because the screen is nothing but a big graph.

In Mode (0), the graph is 64 blocks wide and 32 high.

A keyboard character requires four blocks of space. And so in text mode, we can only make very rough drawings. Wouldn't it be wonderful if we could divide the screen into much smaller boxes? And that would allow us to produce drawings that aren't that clunky. You only need to switch the computer to the *second mode (Mode 1)*.

easier done than said! Simply type MODE (1) and press RETURN.
In MODE (1), the graph is 128 blocks wide and 64 high, and this is already considerably finer.

In MODE (1) the editing is slightly different. It starts with the fact that you can't type text characters with the keyboard. What is this mode for then?

We're going to use a BASIC command now to bring something to the screen. It's called SET.

SET tells your computer to fill one of these blocks with colour.

Each block has its own position on the screen. So, before we can explain to the computer exactly what point it should beSETzen, we have to put "coordinates" on it.

"Coordinates? Help! It's getting a little too technical!"

Please do not be deterred by this term. Because you probably know coordinates better than you think, think about how to deal with a map. You only look in the table of contents for the street name. In addition to the street name, there is usually a letter and a number.

Most city maps are as split as your computer's screen, with lines running horizontally and vertically. The letters determine the horizontal and the numbers the vertical. To

To reach the goal, you only have to follow the two lines until they meet. There you will find the street you are looking for. The letter and the digit in the road map are nothing but coordinates.

And to define a point on your screen with SET is just as easy. You just need to type SET and then type a set (2) of coordinates in your computer. The first indicates the vertical line of the graph (and may lie between 0 and 127). The second, of course, is the name of the horizontal graph line (it lies between 0 and 63).

How do you delete a block that you already coloured with SET? No problem! You can use a BASIC command to do this, which causes the opposite of SET, namely RESET. This command must be combined with the same coordinates that the SET command was previously associated with.

It may take hours to colour a larger field with SET, as long as you have to do this block by block. Not really. There's a little trick there that saves you that effort. This trick is not new. We used the same thing when we taught the computer how to sing.

We only use one SET command, followed by two variables. In the first disk space (let's call it V) we save the vertical coordinate. And on the second storage space (called H) we set the horizontal coordinate. Do you want to see how it works? Then type this programme:

```
5CLS
10 MODE (1)
20 FOR V=0 TO 127
30 FOR H=0 TO 63
40 SETV,H
50 NEXT H
60 NEXTV
```

programme analysis

Row 10: Switches to the correct mode (this is Mode (1).) Row 20: Is the beginning of a FOR-TO loop. For each new loop, the value of V is increased by 1.

Row 30: We are opening a new FOR-RO loop. As with the first loop, it will increase the value by 1 (this time H).

Row 40: And here's our SET command. Each point defined by the coordinates in the memory locations is coloured. Row 50: Says NEXT dog will end one of our FOR-TOS loops. Remember: The H loop is nested inside the V loop, so we have to finish the H loop first. When the computer comes from this line, it flips back to line 30 - the beginning of this FOR-TO loop and increases H by 1.

The computer continues in the loop until the value 63 is stored in H. Then he continues:

Row 60: This is NEXT V. This marks the end of our "outer" loop. So the computer will go back to the beginning of this loop (that's line 20) and increase the value of V by 1.

That's all! Are you ready to try it out? Then type RUN.

> The Vz-200 has begun to "paint" the entire screen. A little patience, please. It will take some time to get it done.

Do you want to get rid of the "painting" again? That's simple enough.
When your computer is finished, change line 40 to:

> RESETV, H

You should see how the computer wipes the varnish when you run this modified programme with RUN.

We make rainbows! (It's not called *"Colour Computer"* for nothing.)

There's something about your computer that's probably been a mystery to you right from the start. And this thing is the set of colour markers on the keyboard. (See these? You are above the first eight buttons on the top row.)

> The marker above the button 1 is green. Above the 2 is yellow and above the 3 is blue. Then follow red, light brown, light blue, magenta red until the button 8 with orange. You look beautiful. But they are not there for decoration alone.

The number on each key represents the code for the colour. These codes, along with a brand new BASIC statement (please be patient, we'll hear more about it soon), we use to produce colours on the screen.

So, by now, you have certainly recognised how logical the names of most BASIC statements are. We don't really have to tell you what our new BASIC command is called. His name is, who would have thought that, COLOUR!

The first number describes the foreground colour and the second number the background colour.

Do you remember the two different modes we told you about? Now, when it comes to colour, each mode has a slightly different set of rules.

Mode (0)

You have the choice in this mode between two different background colours. The first one is green. When used, it has a special code number, namely 0. (We're all used to seeing a green screen, because the computer automatically selects that colour when it's turned on.)

Example: If you want the computer to have a green background and blue foreground on the screen, type the following statement:

COLOUR 3, 0

In this case, three, the code for blue, comes first. The (background) code for green is 0 and is therefore the second number of the pair. (By the way, did you notice that we keep the individual code numbers separate with a comma?)

But there is an alternative. We choose the second orange as the background colour, the code number for it is 1. Some variety can't hurt. Let's try our new COLOUR statement:

Strangely enough: Because we are currently only dealing with the background colour, we can omit the first (foreground) code number and just say:

COLOUR, 1 RETURN

Of course, the place where the foreground colour code comes in is empty. But we still have to type in the comma that would otherwise be listed afterwards. This tells the computer that it only gets one code for the background colour.

Are you sure you're ready now? Although it is simple, this is probably the most spectacular thing we have seen so far

. Now, if you think you can stand the excitement, tap COLOUR, 1.

Have you ever seen such an orange? We warned you it would be quite a difference. And switching back to green is just as easy. Just type: COLOUR,0 and there you go.

Let's get a little bit smarter and bring some foreground colour into the picture. How about a red foreground and a green background?

Deal? Just type:

COLOUR 4.0

"I see the green, but where's the red? It didn't work!" Would we give you a "dud" statement? Of course not! It actually worked. There's a good reason why you can't see the foreground colour. And the reason is that there's nothing on your screen but text.

The only keyboard characters that can be represented in a different colour are the GRAPHIC CHARACTERS.

Try some graphic characters. You'll see they're as red as possible.

Mode 1

The colour selection in this mode is somewhat limited. (Unfortunately, but you can't have everything!)

Your background colour can be either green (whose code is still 0) or light brown (code number 1).

[eeN] [euow] ue] [eo [surr [cvaN] bacwra] [oavce]

LILIE LIEILI

Green, yellow, blue, red: 1 2 3 4.
If you use green as the background, you can only use this colour palette.

Light brown, light blue, magenta, orange: 5 6 7 8.
And if you choose light brown as the background, you have to stick to this colour palette.

Now we don't want to emphasise one thing that's obvious. But in all cases:

For the COLOUR statement to work, you need a colour TV.

20 Latest information

And now, as the sun slowly tilts to the west, we're finishing our little book. Did you enjoy your first escapades into the world of computers? We accept that, because we enjoyed showing you the way.

And it's certainly been a long, long road. Since the moment you opened this book for the first time, we have faced challenges and defeated them without much effort. For the sake of the good old days, let's go back to some of the vanquished hurdles:

"What? You expect me to use this computer? I don't know what the fucking thing is!"

This was probably the biggest worry for some of you.

The computer/human language barrier. Have you realised that you are now bilingual? You can now understand yourself almost fluently - in BASIC. (Really! Just think of all the words you've learned.)

The horrible thing called "computer programming."

Remember when we first mentioned this? It probably scared most of you. And yet, here you are, ready to write your own programmes!

The obstacles were quite difficult and there were only a few! (After you have so much practice in hurdle-taking, you are probably Olympic-ready.)

The world of computers is a very extensive field. Far too extensive to be dealt with briefly (or in this little book). What we could hope for was to give you basic knowledge about computers in general and computers in particular. We want you to be well equipped for your further development as a programmer. And if this manual made that possible, we could be more satisfied.

You know that a good, solid friendship costs nothing but time. So be prepared to accept the effort required to get to know your computer more closely. Take time to experiment, time to practise and time to play. (We promise you: The effort is worth it!)

And now a final piece of advice. If you ever have trouble, don't hesitate to go back to BASICs. Is something going wrong? Anything? Just come back to this little manual. It is always there to explain it again from the beginning.

And: Don't be afraid to experiment. Trial and error are the best teaching methods, and by trying your ideas with the computer, you cannot possibly harm it.

So: Goodbye, now and good luck! An in-depth book for your computer will be published in the summer of 1984.

How to Treat Your Computer Well

It's not hard. It's not hard. Simply read the notes below. It's good advice, and if you follow it, your computer will be grateful for it.

Use it carefully.

To be dropped is to have a traumatic experience for the computer. Please do not try to push it or push a door open with it. If someone did that to you, you'd be working just as badly as your computer would be.

Heat also interferes with your computer. Therefore, you should not place it on a window sill or on the terrace or elsewhere, • where it is strong sun-

is exposed to radiation, leave. No, he wouldn't get a sunburn, but he would get sick.

Nowhere is it as beautiful as home.

For your computer, the place at home (a safe, of course) where you put it when you're done with it. If it's connected to the grid all the time, it would be disrupted by the overheating that follows.

Some etiquette, please!

Unlike humans, your computer won't mind if you eat in front of it. But please always remember your table manners. Because your computer will definitely be offended if you spill orange juice or breadcrumbs onto the keyboard. It's really a good idea to stay away from the computer with food. (Just to be sure.)

Find a safe place

You certainly wouldn't want to put your computer in the middle of a freeway. Unfortunately, there are equally dangerous places in the household. So you should look for a quiet place, where you are not in the way of anyone, if you find yourself for a longer session with your computer. (Believe us, there's nothing more frustrating than someone stumbling across the power cord!)

Don't play doctor.

Would you have surgery from one of your friends if you were sick? No - we did not accept that. If we humans get sick, we go to the doctor...and if computers are sick, they must go to the computer "doctor"! So if your computer isn't working properly (and you're sure it's not up to you), take it to the dealer. With one of our "computer doctors," the computer will be ready for use again soon. Please do not try to do "operations" yourself!

Annexe A

The BASIC commands we learned Chapter 4

PRINT (with applications) GOTO END LIST RUN BREAK CONT CLS NEW

Chapter 5

PRINT (excluding

applications) Chapter 6

LET

Chapter 7

INPUT

Chapter 8

IF-THEN ELSE, etc.

Chapter 10

RND SQR

Chapter 12

FOR-TO NEXT

Chapter 13

STEP Chapter 14

GOSUB RETURN

Chapter 16

DATAREAD

Chapter 17

RESTORS

Chapter

18

SOUND

Annexe B

Reserved words

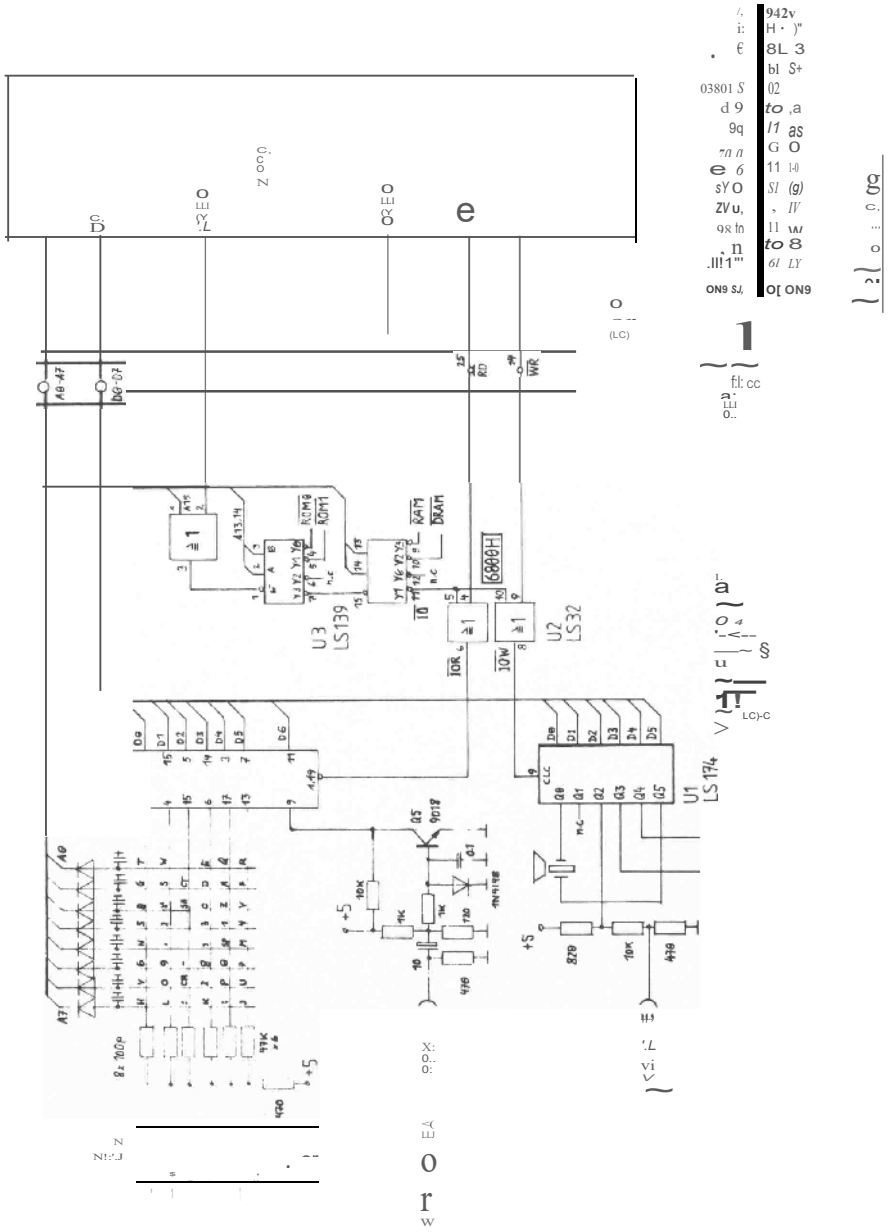
The following words may not be used when marking the variable parking spaces! The computer has reserved it for special purposes ... If you try to use them anyway, you will cause a lot of confusion!

Be sure to note: Do not use:

words that are on the reserved list; or
those whose first two characters are the same as the first two characters of the reserved words!

(Why? Because the computer only reads the first two characters of the variable name. The names, DATA and DAVID mean the same to your computer!)

ABS AND ASC ATN
CHRS CLOAD CLS COLOR CONT COPY COS CRUN CSAVE DATA DIM
EXP FOR
GOSUB GOTO
IF INKEYS INP INPUT INT
LEFTS LEN LET LIST LOG LLIST LPRINT MODE
MIDS
NEW NEXT NOT
OR OUT
PEEK POKE POINT PRINT
READ REM RESET RESTORE RETURN RUN SET SGN
SOUND SIN SOR STEP STOP STRS
TAB TAN TO THEN
USING USR


U3
LS139
U2
LS32
U1
LS174
6000H
ROM0
ROM1
RAM
DRAM
IOR
IOW
RD
WR
+5

P1
NC
NC
NC
NC
NC
NC
+5V
A0
A1
A2
A3
A4
A5
A6
A7
D0
D1
D2
D3
D4
D5
D6
D7
IORQ
RD
WR
P2
NC
+5V
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
D2
D7
RESET
RFSH
M1
WAIT
NMI
RD
IORQ
THE PROPERTY OF THE COMPANY
NO REPRODUCTION OR UNAU-
THORIZED USE IN PART OR IN
WHOLE SHALL BE MADE WITH-
OUT WRITTEN CONSENT OF THE
COMPANY AUTHORITY.
DIMENSIONS ARE IN MM INCHES
AND TOLERANCES ARE
XXX XX ANGULAR
ALL MACHINED SURFACE
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING
MATERIAL
FINISH
NEXT ASSY
USED ON
DWN BY
CHECK BY
ENGR
ENGRG AUTH

Notes

Notes

Notes

Notes

Introduction to Programming with Switching Images