PROGRAMMING WITH YOUR JOHN SANDS SEGA PERSONAL COMPUTER

John Sands
SEGA®

Written and
published by
John Sands Electronics

Written and published by
John Sands Electronics
Division of John Sands Limited
6 Bay Street
Port Melbourne Vic
3207

First published 1984

Typeset in Australia by
Typecast Pty Ltd, South Melbourne
Printed in Australia by
Valentine Graphics, Clayton, Vic

ISBN 0 9590880 0 8

FIRST THINGS FIRST

Owning a John Sands Sega Computer puts you in excellent company. Whether you are a first time user, or an experienced operator, you will find that your John Sands Sega is the best computer in its class. And one which will give you many years of service.
Although your computer has many unique and futuristic features, you will find that this guide will help you understand how to get the best from it – with simplicity of operation being the keynote.
If this is the first computer you have ever laid hands on, please don't think you're in for a tough time learning how to 'drive' it. You'll find that operating your Sega gets easier and easier everytime you use it. Once you have mastered each stage and each function, it will become as simple as riding a bike.
The best way we know of getting to know your John Sands Sega is to play with it for a couple of hours before starting to read this guide. Use the keys in any order you wish and see what happens! You can't do any damage and no harm will result. You might see things appearing on the screen and hear sounds that you don't yet understand – but you will later!
Before you start working through this book, remember that your John Sands Sega is for you to command. It cannot think for itself and needs your instructions to tell it what to do. If you always keep this in mind, you will find its uses to be unending!

Communicating with your John Sands Sega

The word 'programming' means communicating with your computer and giving it instructions which you want it to carry out. Once it has been programmed the way you want it to be, it will obey any orders which can be related to that particular programme.
The number of programmes you can use is virtually endless. You can create programmes which apply to mathematics, calculations, graphics,

composing music, games, budgeting, charts, diagrams, sporting statistics, word processing, the calories you eat in your daily diet, how much you spend each week, month or year – and so on and on and on!
Everybody in the world speaks their own specific language.
Many different types and makes of computers speak many different languages. The particular language your John Sands Sega knows about is called BASIC.
BASIC stands for Beginners All-Purpose Symbolic Instruction Code.

Let's get things underway

By referring to the instruction leaflet with your John Sands Sega, you will find how to power up your computer.
Once you have done this you are ready to begin.
Right?

Contents

CHAPTER 1

LET'S START PROGRAMMING!

Having powered up your computer, you're now ready to start to put it through its paces.
On the screen, you will see the number of bytes – or characters – which are available for you in the computer's memory.
Under this figure you will see the word READY.
And beneath READY a square which blinks on and off.
What the READY really means is that the computer is telling you that it's ready for your command.

So now let's begin by telling it to do something.
Type your command exactly like this:

PRINT "WELCOME TO TOMORROW'S WORLD TODAY"

Don't worry if the words continue over to the next line.
They're supposed to.
The blinking square is the CURSOR and wherever it is placed on the screen when you type on the keyboard, the letter, symbol or figure you typed, will appear in that spot.
Right. Now just check that the line you typed is exactly as it is supposed to be. If you've made a mistake, don't worry. Just press the DEL key and the last character you typed will disappear. Another press will delete the previous character, and so on.
If you have made no mistakes, your screen will now look like this:

READY
PRINT "WELCOME TO TOMORROW'S WORLD TODAY"

Now. Press the CR key.
The screen will now show you this:

READY
PRINT "WELCOME TO TOMORROW'S WORLD TODAY"
WELCOME TO TOMORROW'S WORLD TODAY
READY

Your John Sands Sega obeyed your command and printed the message you had written in quotes.
The final READY told you that the computer is now awaiting your further instructions.

So let's give it another command!
Type:

PRINT "3 + 2"

Then press the CR key.
The computer replies with:

3 + 2

So what you have told it to do it has carried out perfectly! But that's not all it can do.
This time type:

PRINT 3 + 2

Then press the CR key.
The reply will be:

5

What you have just learned is that the " (quotation) mark at the beginning and end of an instruction must have a special meaning to the computer. We'll take it a little further by doing the following exercise.

Type the following:

PRINT 6 + 3 CR
PRINT "6 + 3" CR
PRINT "6 + 3 EQUALS" ; 6 + 3 CR
PRINT "6 / 2" CR
PRINT 6 / 2 CR

See what happened?
Anything written between two " (quotation or quote) marks is printed exactly as it appears.
Anything not in quotes can be added, subtracted, multiplied, or divided by the computer's mathematical capability.
Smart eh?

It's all done by strings and numbers!

Absolutely everything you type, is seen by your computer as either a STRING or a NUMBER.
By beginning and ending with " (quote) marks, the computer recognises it as a string and prints it exactly that way.
If there are no before and after quotes, it sees it as a number, and figures out the answer on that basis.

Now we'll turn your computer into a calculator

Feel like a spot of long division without a headache?
Here goes!

Type:

PRINT "70 DIVIDED BY 1.25 IS" ; 70 / 1.25 CR

Your computer answers:

56

Multiplication? No sweat!
Type:

PRINT 185 * 28 CR

And the answer is: 5180
In case you're wondering why you had to type * instead of the usual x you're used to, that answer is simple too. Because your computer always sees things exactly as they appear, it could mistake the alphabetical letter x for the multiplication sign x and not know what to do. To save any confusion, we always use the sign * to indicate mathematical multiplication.

Let's keep thinking about maths.
Type these four lines:

PRINT "100 * 5 =" ; 100 * 5 CR
PRINT 12.67 / 4.354 CR
PRINT 3 * 3 * 3 ; " IS THE CUBE OF 3" CR
PRINT 84 / 21 CR

Is it all starting to make sense to you?
As you are finding out, your John Sands Sega computer is an extremely logical and smart thinking friend.
Have another go for yourself:
Programme the computer with two command lines which will print the following two problems together with their answers.
Ready?

14.37 * 18.654 =
791.3 / 16.4 =

Now. Did your command lines look like this?

PRINT "14.37 * 18.654 = " ; 14.37 * 18.654 CR
PRINT "791.3 / 16.4 = " ; 791.3 / 16.4 CR

If so, you should have got the following answers:

14.37 * 18.654 = 268.05798
791.3 / 16.4 = 48.25

Everyone makes mistakes... (but not your John Sands Sega Computer)

While you've been using your computer, it may have printed some strange messages on the screen from time to time. If this hasn't happened before, we're going to make it happen now.

Type in a deliberate mistake as follows:

PRINNT "SMARTIE PIE" CR

Your computer replies with:

? Syntax error

What SYNTAX ERROR means is that the word PRINNT is not in the computer's vocabulary or language. It just doesn't understand what you're talking about!

Every time you see the words SYNTAX ERROR on your screen, it is because you've made some kind of mistake or have used a wrong key or two.

Sometimes also, your computer will flash up the ERROR message when it can understand what you want it to do – but it feels it's impossible or doesn't make proper sense. Here's an example:

Type:

PRINT 7.26 / 0 CR

The computer will reply:

? Division by zero error

In the nicest possible way it's trying to tell you that it's pointless trying to divide a number by 0!
Which is fair enough when you come to think about it.
Incidentally, if you are ever presented with an error message you can't understand, just refer to Appendix I at the rear of this book. You'll find the explanation there.

Short cutting and saving finger wear!

Until now, if you wanted your computer to PRINT, you have typed the word PRINT. Which has taken you five keystrokes. Here's how to do it with one keystroke!
With one finger keep the FUNC key depressed. And with another, press the PRINT key.
Brilliant eh? This goes for a lot of other commands too and means that you can really save hours and hours when you get deeper into programming. (Think of FUNC as short for FUNCTION – which is what you are telling the computer to carry out.)

Chapter 1 Summary

This is what you have basically learned from Chapter 1:
How to use the PRINT command and the FUNC key.
You have learned how to correct a typing error with the DEL key and how to enter your commands with the CR key.
You also have an understanding of STRINGS and NUMBERS and why ERROR messages will sometimes appear on the screen.
And it wasn't hard at all, was it?
If there's anything you don't properly understand, go back over the chapter and refresh your memory.

Notes:

remember = sign then ; then :

CHAPTER 2

GOING DEEPER...

Your John Sands Sega has a brain which allows it to remember anything you wish.
Now we're going to prove it.

Type:

X = 6

Leave that in the brain and start typing a few lines of anything that comes into your head. Words, single letters, numbers – just anything. Then press the CR key.
Finished?
Chances are, the computer will reply with ? Syntax error.

Now, type this:

PRINT X

See! It replied with 6. And will continue to do so until you either turn the computer off or do the following:

Type:

X = 600

This time, if you ask it to PRINT X it will reply with 600.
What you have done is to cancel your original statement that X = 6, and tell the computer that from here on X = 600.
This doesn't only work with the letter X, it will do so with any letter in the alphabet. And it doesn't only work with one letter either; it can do so with two letters, or a letter then a number. Which opens up a lot of possibilities doesn't it?

Here we go again.

Type the following:

```
X  = 600
F9 = 100
M  = 25
CS = 51
```

Now type:

```
PRINT X, F9, M, CS
```

Your John Sands Sega will reply:

```
600      100
25       51
```

The power of the dollar!

If you want to command your computer to remember a string of either letters or numbers,

simply type:

```
A$ = "DON'T FORGET"
B$ = "THIS"
```

Numbers and strings stored by your computer are called VARIABLES. Let's check and see that your John Sands Sega has remembered them all.

Type in the following:

```
PRINT X, F9, M, CS
PRINT A$, B$
```

Things your John Sands Sega can do with variables. And things it can't!

Numeric Data:

1. Numbers which are not in quotes are NUMERIC data.
2. NUMERIC data can only be assigned to variables WITHOUT A $ SIGN.
3. You can use NUMERIC data in a mathematical expression.

String Data:

1. STRING data can be any data in quotes.
2. STRING data can only be assigned to variables WITH A $ SIGN.
3. You cannot use STRING data in a mathematical expression.

Here are some examples of what you can and cannot do –

Q = 6 / 123	CORRECT
L = ''34''	NO NO
SHORT$ = ''A''	CORRECT
Z$ = 3.14	NO NO
M = G$ + 5	NO NO
A = B / 2	CORRECT

Switching off can be the biggest turn off of all!

Please, please, please remember this.

WHEN YOU TURN OFF THE POWER TO YOUR COMPUTER YOU WILL ALSO REMOVE EVERYTHING IN ITS MEMORY!

There's nothing worse than spending hours of enjoyment and achievement in programming your John Sands Sega computer and then unintentionally turning off the power!

All your programmes can be saved on a Data Tape as will be explained later – but in the meantime, please do remember that if you want to save what you've already got, don't turn off the power!

Chapter 2 Summary

Now we're really getting somewhere aren't we?
What you've learned in Chapter 2 is an understanding of VARIABLES – both STRING and NUMERIC.
You've also been warned not to switch the power off to your computer, unless you wish to erase everything in its memory!

Notes:

Numeric & string on different PRINT commands

CHAPTER 3

LET'S DO SOME REAL PROGRAMMING

Now we're really getting down to showing your John Sands Sega that it's in the hands of a fast learning programmer!

Here we go.

Type this command:

10 PRINT "JOHN SANDS SEGA FOREVER"

Although you pressed the CR key, the computer did not print those words on the screen did it? Don't worry, there's nothing wrong.
What you actually have done is to write your very first programme. The number 10 at the beginning of the line has a special command meaning.

Now type:

RUN

Wow! The computer is RUNNING your programme!

Type it again:

RUN

There it goes again.
(To save your fingers, next time you are asked to type RUN, depress the FUNC key with one finger and the RUN key with another.)

Now, type this:

20 PRINT "TELL ME YOUR NAME"

Now, type:

LIST

(Use the FUNC and LIST keys this time.)

And there's the LIST of your complete programme. Your screen now looks like this:

10 PRINT "JOHN SANDS SEGA FOREVER"
20 PRINT "TELL ME YOUR NAME"

Now let's RUN the programme and see what we've got.

Type:

RUN

(Use the FUNC and RUN keys.)

Here's what we have:

JOHN SANDS SEGA FOREVER
TELL ME YOUR NAME

Have you given the computer your name, as it asked you to do?
Type it in, then press CR.
Hello! Hello! There's that ? Syntax error.
Do you know why that appeared?
It was because you didn't speak to your Sega in language that it can understand.
So let's now communicate properly.

This time type:

30 INPUT A$

You have now told your John Sands Sega to stop and wait for you to type a STRING which it will call A$.

Now add another line to your programme:

Type:

40 PRINT "GOOD DAY, " ; A$

Just to check that everything's as it ought to be, we'll just ask the computer to list the programme. Here goes!

Type:

LIST

Is this what your programme looks like?

```
10 PRINT "JOHN SANDS SEGA FOREVER"
20 PRINT "TELL ME YOUR NAME"
30 INPUT A$
40 PRINT "GOOD DAY, " ; A$
```

If it doesn't, it should – so go back and make any corrections necessary!

Now, take a deep breath and type the following command:

RUN

There you are!

Is this what you got?

```
JOHN SANDS SEGA FOREVER
TELL ME YOUR NAME
? (THE NAME YOU ENTERED)
GOOD DAY , (THE NAME YOU ENTERED)
```

Great!
Now run the programme a few more times, changing your name when the computer asks you what it is.
See what happens.

Now we'll show you a short cut which will let you run your programme over and over without stopping each time.

Type this:

50 GOTO 10

Now RUN the programme.
Do you understand what is happening?
When the computer reached the command 50, it was told to GOTO line 10. (Or to go back to line 10).
This means that every time it gets to line 50, the computer goes back to line 10.
This is called a LOOP. And the only way you can ever stop it is to press the BREAK key.
Do that now or you'll get dizzy!
If you want to delete that 50 GOTO 10 command, simply type 50 followed by the CR key. It is now removed.
This applies to any command line you wish to take out of any programme you are devising.
You're now going to write two of your favourite words – your name!
And we're going to give it the works!
First of all, we need to change your commands.
If you want to change (but not completely delete) any command line in a programme, all you have to do is type it over again.
Your Sega will then only remember the last instruction you have given it for that particular command line.

So now type this:

50 GOTO 40

Check to see what your programme looks like now. Hopefully, it will be like this:

```
10 PRINT "JOHN SANDS SEGA FOREVER"
20 PRINT "TELL ME YOUR NAME"
30 INPUT A$
40 PRINT "GOOD DAY,  " ; A$
50 GOTO 40
```

Now enter RUN and see what happens.
When you've had enough of looking at that wonderful sight, just press the BREAK key.

Right. Now we'll do something else.

Enter line 40 again, but this time use a comma, like this:

```
40 PRINT A$,
```

Now RUN the programme.
The comma has ordered the computer to print everything in two columns!
Enter BREAK and now try the semi colon:

```
40 PRINT A$ ;
```

Now RUN the programme again.
See what's happening?
Enter BREAK and you'll see that by using the semi colon, the computer is condensing everything together.

So.

What we've just proved is that:

A COMMA IN A PRINT LINE ORDERS THE COMPUTER TO PRINT IN TWO COLUMNS.

A SEMI COLON IN A PRINT LINE ORDERS THE COMPUTER TO CONDENSE THE PRINTING TOGETHER.

Chapter 3 Summary

Here is a list of BASIC words you have encountered and used in this chapter –

INPUT
GOTO
RUN
PRINT,
PRINT;

You have also learned how to alter or delete a programme line.
And you have used the BREAK key to break into a programme.
If you want to write yourself a couple of notes about any of the above, now's the time to do it.

Notes:

CHAPTER 4

OF HUES AND HEARS AND IFS AND THENS!

You probably already know that your John Sands Sega can make colors and sounds.
You may not know what it can do with IF's and THEN's.
All will be revealed in this chapter.

We'll start with colors

Until now, you've been working with a green screen and black lettering. How would you like to see things with blue lettering on a green background? Here's what to do:

Type:

COLOR 4 , 3

Feel like something different?

Then type this:

COLOR 6 , 10

Your John Sands Sega offers you a choice of 15 individual colors. Each one is represented by a number from 1 to 15.
When you just gave your Sega the color command, the first number (or parameter) you typed, was the printing color. And the second number was the screen color.
Try yourself out on other color combinations now.
It's like playing with your own rainbow!
(You can check the Appendix for the full list of colors and their appropriate command numbers.)

The impressive sounds of Sega

You'll have heard the beep sound every time you press the keys of your John Sands Sega.
These beeps are just to let you know that your instructions are being received, or that the computer is obeying your commands.
Now, we'll go a stage further.

Type:

SOUND 2 , 110 , 15

If you didn't hear anything, turn up the volume on your television set.
What the computer is doing, is playing to you the lowest note it can make!

Now type this:

SOUND 2 , 110 , 0

Now we'll explain what those numbers are all about.
Your John Sands Sega has three 'voices' – each one of which can produce sound independently of the others.
The first number, or parameter, indicated whether you want voice 1, 2 or 3.
The second number, or parameter, relates to the tone – or frequency – of the note. This may be from 110 to as high as you wish. (If it's too high, you'll never hear it with humanoid ears; your dog might be able to though!)
The last number, or parameter, is the volume. Number 0 will produce no volume – number 15 will produce the highest volume.

Let's go again.

Type:

```
SOUND 2 , 500 , 10
SOUND 3 , 600 , 15
```

Now type:

```
SOUND 0
```

You've just discovered that your last command turned off the sound!
Now programme some different notes and volumes for yourself.
And try using more than one voice while you're at it.
Beethoven or the Beatles never had it so good!

Now surround yourself with color and sound!

First, clean the memory of your John Sands Sega.

To do this, type:

```
NEW
```

Right?
Let's go.

Enter this programme:

```
10 PRINT "TO ALTER THE TONE"
20 PRINT "ENTER A NUMBER BETWEEN 110
AND 1000"
30 INPUT M
40 SOUND 2, M, 10
50 GOTO 10
```

Now RUN the programme and hear something of what the John Sands Sega can do.

*** Important**

If a ? Statement parameter error appears on the screen when you RUN this programme, it is because you entered a number other than one between 110 and 1000.
Naughty!
This was an error, and any error will not allow the computer to continue to RUN your programme.

Remembering what you've been through before, what do you think would happen if you changed the line 40 command to this:

40 SOUND 2, M, 15

You're right!
By entering that change, you have programmed the computer to produce the tone at a different volume level.
Now, for something to do by yourself.
Press the BREAK key and then erase the current programme from the memory, by hitting the NEW key.
This time, write your own programme based on the one above and command the computer to bring up the color you ask for.
You have 15 choices, numbered from 1 to 15.
Good luck.

Write your programme here:

Here's our programme:

```
10 PRINT "TO CHANGE THE COLOR"
20 PRINT "ENTER A NUMBER BETWEEN 1
AND 15"
30 INPUT T
40 COLOR 1, T
50 GOTO 10
```

Try it out.

Starting to programme like the pro's.

You've already begun to realise that there are often several ways of programming your John Sands Sega to do the things you wish it to do. Let's take the example of commanding the programme to stop by touching the BREAK key.
A better idea is to get your Sega to ask you if you're ready to end the programme.

You can do it this way, by changing line 50 to the following:

```
50 PRINT "LIKE TO SEE ANOTHER COLOR?"
```

Now add the following two programme lines:

```
60 INPUT F$
70 IF F$ = "YES" THEN 20
```

And now RUN the programme.

This is how the programme would look:

```
10 PRINT "TO CHANGE THE COLOR"
20 PRINT "ENTER A NUMBER BETWEEN 1
AND 15"
30 INPUT T
40 COLOR 1,T
50 PRINT "LIKE TO SEE ANOTHER COLOR?"
60 INPUT F$
70 IF F$ = "YES" THEN 20
```

Just to refresh your mind on what we have actually just added, we'll run through what each of the new lines has achieved.
Line 50 simply asked a question.
Line 60 has ordered the computer to stop and wait until it receives an answer.
Line 70 programmed the computer to go back to line 20 IF the answer to the question was YES. Any other answer would make the programme end, as there are no more lines in the programme.
That's enough for Chapter 4.
If you've absorbed it all you're doing extremely well. If you haven't got everything absolutely perfect, don't worry. You've covered a lot of ground already and it will all come to you after a little more practice.

Chapter 4 Summary

How are you feeling now?
Home computing on the John Sands Sega really is fun, isn't it?
And you've discovered that you don't have to be some sort of mathematically minded whiz kid to work it.
In this chapter you have learned about making and changing COLOR and making and changing SOUND.
You've also done some work with the IF command.
So IF you want to jot down a few notes about anything, you've arrived at the right spot.

Notes

CHAPTER 5

MEET THE COUNT!

Right.
Now you're going to teach your John Sands Sega to count!

So type this:

```
10 FOR X = 1 TO 10
20 PRINT "X  =" ; X
30 NEXT X
40 PRINT "I HAVE STOPPED COUNTING"
```

RUN the programme a few times.
Now RUN it a few more times, but each time you do so, replace the command line 10 with one of these lines:

```
10 FOR X = 4 TO 75
10 FOR X = -6 TO 6
```

Do you see what FOR and NEXT are causing your Sega to do?
They're making it count!

Let's look closely at this programme.

```
10 FOR X = 60 TO 87
20 PRINT "X  =" ; X
30 NEXT X
40 PRINT "I HAVE STOPPED COUNTING"
```

Line 10 has told your computer that the first number will be 60 and the last one 87. It stores these numbers in the variable X.

Line 30 programmes the computer to keep returning to line 10 for the NEXT number – the NEXT X – until it arrives at the last number which is 87.
Line 20, being between the FOR and NEXT lines, tells the computer to PRINT the value of X every time it counts.

Now we'll add another instruction. Type this line which will appear between FOR and NEXT –

15 PRINT '' – I'M WORKING – ''

Now RUN the programme.
Every time it counts, your John Sands Sega will obey any lines you choose to insert in between FOR and NEXT.
So now write a programme which makes the computer print the name of your street 20 times.
Then write a programme which prints the multiplication table for the number 6 – from 6 X 1 to 6 X 10.

Write your first programme here:

And write your second programme here:

Done all that?
Compare your programmes with these and see how well you did!

Street Name Programme:

```
10 FOR X = 1 TO 20
20 PRINT "YOUR STREET"
30 NEXT X
```

Multiplication X 6 Programme:

```
10 FOR X = 1 TO 10
20 PRINT "6 * " ; X ; " =" ; 6 * X
30 NEXT X
```

Multiple counting

There's more to life than counting by ones. So now we'll show you how to make your computer count in multiples.
First, erase the last programme.

Now type the following:

```
10 FOR X = 5 TO 50 STEP 5
20 PRINT "X =" ; X
30 NEXT X
40 PRINT "I HAVE STOPPED"
```

RUN the programme.
As you will see, the computer counted in 5's.
Line 10 told your Sega that the first X was a 5.
That the last X was a 50.
And all the X's in between 5 and 50 were 5 numbers apart.
That they increased by STEPS of 5.
Now make the computer count by eights. Which you can do by changing line 10 to the following:

```
10 FOR X = 8 TO 25 STEP 8
```

When you RUN the programme, it should look like this:

```
X = 8
X = 16
X = 24
```

It ignored that last X (25) because 24 + 8 = 32.
Try a few more programmes so you can really get the idea.

Here are some suggestions:

```
10 FOR X = 7 TO 49 STEP 7
10 FOR X = 5 TO -2 STEP -1
10 FOR X = 8 TO 64 STEP 4
```

Sound counting

Now we're going to add some sound!

Erase your previous program and enter this one:

```
10 FOR X = 110 TO 210
20 PRINT "TONE" ; X
30 SOUND 2, X, 15
40 NEXT X
50 SOUND 0
```

Your programme will instruct the computer to count from 110 to 210 in steps of 1. Each time it counts, it also does what you have told it to do in lines 20 and 30.

So it will PRINT X – the correct COUNT
And it will SOUND X's particular TONE
When the computer got to FOR in line 10 it made X = 110.
Then it moved to line 20 and printed the value of X, which was 110.
Then it moved to line 30 where it made the SOUND TONE of 110.

Then it went back to line 10 again and made X = 111.
And so on until it reached 210.
Now make a change.

This time try:

10 FOR X = 210 TO 110 STEP -2

Using the FOR command, how would you change line 10 to make your John Sands Sega:

Sound every ninth note down from 1000 to 500?
Sound every ninth note from 1000 to 2000?
Sound every fifth note from 110 to 250?

Write your answers below:

10 For X = 1000 to 500 Step -9
10 " " 1000 to 2000 Step 9
16 " " 110 to 250 Step 5

Is this what you wrote:

```
10 FOR X = 1000 TO 500 STEP -9
10 FOR X = 1000 TO 2000 STEP 9
10 FOR X = 110 TO 250 STEP 5
```

Here endeth Chapter 5!

Chapter 5 Summary

In this Chapter you have learned how to programme your John Sands Sega to COUNT in ones and multiples.
You have also learned how to combine this ability with the SOUND command to create sound effects.
If there's anything you wish to make a note about, do it here. Now.

Notes:

CHAPTER 6

KEEPING AND LOADING YOUR PROGRAMMES

Before we tell you how to LOAD your programme in order to keep it for further use at any time, we'll explain the way in which it is kept and used in Random Access Memory – or RAM as it is more frequently called.
All information entered into the computer either by you using the keyboard or from a cassette tape (or Micro disk) is held in the form of BINARY NUMBERS. These numbers are represented by either ONES or ZEROS.
The computer does this by turning on or off literally tens of thousands of electrical switches known as TRANSISTORS which are inside INTEGRATED CIRCUITS or MICRO CHIPS.
The BINARY CODE is something like Morse Code – which has combinations of dots and dashes to represent letters of the alphabet; numbers as well.
With your computer, once the power is turned off all the switches revert to zeros and your programme is lost forever! If we want to keep a permanent record of the programme, it must be transferred to cassette tape.
When you have completed a programme on your John Sands Sega and wish to keep it, you use the SAVE command. When this occurs, the computer converts the BINARY CODE into audible sound tones which range from frequencies of around 3000 Hertz (or cycles per second) which represent the zeros, and around 5000 Hertz which represent the ones.
On a typical SAVE command, a constant 'leader' tone of 4 seconds is heard, followed by a burst of File Header Data which contains the File Name, followed by another 4 seconds of constant tone. This is then followed by the programme code being SAVED in bursts of 600 bits per second. (A bit is one eighth of a byte or character).

You can hear these sounds occuring with the John Sands Sega Data Recorder, or through the loudspeaker of a cassette recorder if you disconnect the earphone jack.
If we wish to reload the programme into the Sega from the cassette tape, we use the BASIC language code LOAD. The computer will then search over the tape for the leader tone. When it finds the File Name you require, it will announce the fact on the screen by printing the words FOUND FILE (NAME). It then continues to LOAD the programme code for up to several minutes. When it has completed the job, the words LOADING END will appear on your screen.

The programme is now in the computer's Random Access Memory (RAM). The tape can be stopped, and by pressing the RUN key you will start the programme.
If your Sega cannot find the leader tone, or the particular File Name, it will keep on looking for ever! During the LOAD or SAVE activities, there will be no response from any of the keys on the keyboard, except for RESET.
If you know your programme is near the beginning of the tape, you should have a FOUND FILE (NAME) response in about 20 seconds. If this doesn't happen we suggest you rewind the tape and try again; this time starting with the volume setting at the mid point and gently increasing it. Hopefully, you will hear something!
When the computer does find the leader tone it will then attempt to LOAD. At any time during the LOAD process, you may see a TAPE READ ERROR appear on the screen. This means the computer has lost the signal which is coming from the tape and the LOAD operation has been discontinued.

There are several causes of this infuriating problem! So it's best to make sure they never, or hardly ever, occur. The most common are these:

Poor quality tape. Always use Low Noise 60 minute, or shorter tapes. Never use Chrome or Metal tapes.
Kinks in the tape. Never leave the Play button on when you're not recording.
Poor quality original recording. Make sure you verify all your SAVES. Make a back up tape of everything too.
Bad electrical connections. Make sure your leads are not faulty from the recorder to your John Sands Sega.
Poor alignment of recorder head. This can result in signal loss which is definitely what you don't want! Make sure you use a well known and reliable recorder.
Signal recorded too low. Adjust the VU recording level on your cassette deck to give the maximum signal power without distortion.

From all the above, you can see that to SAVE and LOAD computer programmes is quite a critical process. More so, than for voice or music recording. A small loss of signal might not be even noticeable by the human ear – but it's absolutely vital for your Sega.
You cannot load just part of a programme either; it's all or nothing!
It's a funny thing, but some sophisticated hi-fi cassette decks do not make as reliable computer recorder units as less expensive, portable models. This is because the costly units very faithfully reproduce high frequency sounds as well as any other noise which may be on the tape. All this does is to corrupt your data signal and the result is not the one you're looking for!
The best of all cassette data units are those which have been produced especially for use with computers. These are called 'dedicated' models,

and contain special filters which restrict the recorded signals to the frequencies we want.
You can also expect potential problems if you SAVE a programme on one recorder and LOAD it from another. Nearly all mass produced cassette recorders have slightly different head alignment and signal characteristics.

The equipment you will need

You'll need a recorder compatible with your John Sands Sega. If you already own a portable or hi-fi cassette recorder, try it out anyway.
Then you'll need at least one connection cable with standard 3.5mm Earphone Jack on both ends – or an RCA jack on the recorder end if you are using a hi-fi deck.
And, naturally, a good quality blank audio cassette tape.
Once you've got that list organised, you're on your way!

How to save your programmes

1. Connect the cable from the OUT socket on the back of your John Sands Sega to the IN socket of your John Sands Sega Data Recorder or the MIC or MICROPHONE socket of your cassette recorder.
2. Advance the tape until the brown, magnetic section is visible on both reels.
Hold down the FUNC key and press the SAVE key on your Sega.
The screen will indicate SAVE.
Now type a File Name – which can be up to 16 characters long.
3. Press the SAVE key of your Data Recorder, or the Record and Play keys of your cassette recorder; then press the CR key on your Sega.
On the screen you will now see SAVING START.
This means that your data is now being written to the tape, and can take from one to several minutes, depending on the length and complexity of your programme. When the SAVE is completed you will hear a BEEP and the screen message will change to SAVING END.

4. Now we're going to check that all the information has been SAVED on the tape.
Rewind the tape.
Connect the cable from the OUT socket of your John Sands Sega Data Recorder, or the Earphone or External Speaker socket of the cassette recorder to the cassette IN socket on your Sega. If you are not using a Data Recorder, adjust the volume level to 3/4 of its maximum.
5. Hold down the FUNC key of the Sega and at the same time press the VERIFY key.
The screen will indicate VERIFY.
Now type in your File Name and press the CR key.
The screen will then show VERIFYING START.
Now press LOAD on the John Sands Sega Data Recorder, or Play on the cassette recorder.
Within 20 seconds you should hear a BEEP and the screen will read FOUND FILE (NAME).
If the words SKIP FILE (NAME) appear, you may have made a mistake in the spelling of your FILE (NAME). Check it to make sure. If you did make a mistake, press RESET, rewind the tape and try again – this time spelling the file correctly. (You can omit the file name completely if it is the first file on the tape.)
If no file name is found within 20 or 30 seconds, go through the procedure again – this time increasing the volume on the recorder.
If you still have no success, try another SAVE, or another brand of audio tape. If all else fails, try another cassette recorder.

How to load your programmes

The LOAD procedure is much the same as when you VERIFY.
Here we go –
1. Connect the cable from the OUT socket of the John Sands Sega Data Recorder, or the earphone or external speaker socket on your recorder to the cassette IN socket on your John Sands Sega.
Adjust the volume level to 3/4 of its maximum.
2. Hold down the FUNC key and press the LOAD key.
The screen will respond with LOAD.
Now type in your File Name and press the CR key.
The screen will now show LOADING START.
Now press LOAD on the Data Recorder, or Play on the cassette recorder.
Within 20 seconds you should hear a BEEP and the screen should show FOUND FILE (NAME).
If the message SKIP FILE (NAME) appears, you may have made an error in the typing of the File Name. If you did, press RESET, rewind the tape and retry the LOAD with the correct spelling of the File Name. (You can omit the File Name if it is the first one on the tape.)
If no File Name is found within 20 to 30 seconds, start the procedure over again with more volume on the recorder. If you're still having no luck, try another cassette tape or recorder.
If there are several programmes on your tape, the Sega will search for the particular File Name – showing on the screen the file names it has skipped over as it keeps searching.
A good idea, which saves time, is to make a note of the correct position of each particular programme by using the counter of your cassette deck. You can then Fast Forward to just before this position when you begin to LOAD.
Now that was really quite simple wasn't it?

Chapter 6 Summary

You're now not far away from being a capable programmer.
The only thing which can ever hold you back is your own imagination.
The better that is, the more enjoyment and success you are going to have with your John Sands Sega.
Chapter 6 has taught you how to SAVE all your programmes.
And how to LOAD them back into the computer.
Once they're saved, you can turn off your Sega and when you come back later, you know that everything is safe on tape for LOADING.
As we said before, it is a good idea to make duplicate copies of all your programmes. Just in case...
If you feel like writing yourself a note or two, now's the time.
And here's the spot.

Notes

CHAPTER 7

SEND YOUR SEGA FOR A LOOP!

We're going to show you how to make your Sega loop the loop! A cunning idea if ever there was one.

So now, type this new programme:

```
10 FOR L1 = 1 TO 4
20 PRINT ''L1 = '' ; L1
30 FOR L2 = 1 TO 3
40 PRINT, ''L2 = '' ; L2
50 NEXT L2
60 NEXT L1
```

When you RUN it, your screen should look like this:

```
L1 = 1
                L2 = 1
                L2 = 2
                L2 = 3
L1 = 2
                L2 = 1
                L2 = 2
                L2 = 3
L1 = 3
                L2 = 1
                L2 = 2
                L2 = 3
L1 = 4
                L2 = 1
                L2 = 2
                L2 = 3
```

This sort of thing is officially called a NESTED LOOP. Or you could call it A COUNT WITHIN A COUNT or even A LOOP WITHIN A LOOP.
What the programme does is this:

It counts L1 from 1 to 4.
Every time it counts:
It PRINTS the value of L1.
It counts L2 from 1 to 3.
Every time it counts:
It PRINTS the value of L2.

Any time you place a loop inside another loop, you must close the inner loop before closing the outer loop.
Here's an example:

The following program is CORRECT:

```
10 FOR A = 1 TO 5
20 FOR B = 1 TO 2
30 NEXT B
40 NEXT A
```

The following program is INCORRECT:

```
10 FOR A = 1 TO 5
20 FOR B = 1 TO 2
30 NEXT A
40 NEXT B
```

See the difference?
Lines 30 and 40 are transposed.
Incidentally, here's a little trick which may come in handy. Lines 30 and 40 of the first programme can be combined into one line:

```
30 NEXT B, A
```

Whenever two or more NEXT statements are together, they can be combined in this way. Clever!

Measuring within a measure

Now we're going to turn your John Sands Sega into a measuring device!
You know that 10 MILLIMETRES = 1 CENTIMETRE
and that 100 CENTIMETRES = 1 METRE
Now you're going to impart this information to your computer.
Start by typing this:

```
10 FOR MM = 0 TO 9
20 BEEP
30 PRINT MM ; '' MILLIMETRES''
40 NEXT MM
50 PRINT '' 1 CENTIMETRE''
```

Now RUN the programme.
Got the picture?
When it reached 10 millimetres, your Sega indicated that it had reached 1 centimetre.
We've now created the basic idea for turning your Sega into a measuring device.

Now type the following programme:

```
10 FOR CM = 0 TO 99
20 FOR MM = 0 TO 9
30 BEEP
40 PRINT CM ; '' CENTIMETRES, '' ; MM ; ''
MILLIMETRES''
50 NEXT MM
60 PRINT
70 NEXT CM
80 PRINT '' 1 METRE''
```

Now RUN it.
Press BREAK when you've had enough.
You'll see what it does is this.
It counts centimetres from 0 to 9.
Every time it counts:
    It counts millimetres from 0 to 99.
    Every time it counts:
        It beeps.
        It prints the centimetres and millimetres.
    It prints a blank line.
It prints the fact that 1 metre has been reached.
Now we'll improve our programme.
    Type this line:

35 CLS

Then RUN the new programme.
Now, it clears the screen before printing the millimetres and centimetres.
Try some more modifications of your own.
For instance, you might like to put your LOOPing skills into the aeronautical field. You could devise a programme based on flying exercises in which 1 'Master Loop the Loop' consisted of 3 'Loop the Loops' and 1 'Grand Fantasmogoric Loop' consisted of 2 'Master Loops'! There'd be quite a bit of looping going on in such a programme. Could be fun if you don't crash!
If you're feeling athletic, you could devise a programme in which you're going to run the marathon in a certain time. And instead of using the roads, you're going to do it on a track. The track circuit might be 450 metres and you will need to run around it so many times to reach the distance which is the metric equivalent of 26 miles

365 yards. You could create a LOOP situation which would tell you how many circuits you would have to do and what time you would need to average for each circuit. This one is a bit tricky, but if you understand the basics of this chapter it won't be too hard for you to work it out!

Chapter 7 Summary

In this chapter you have learned about NESTED LOOPS (alias loops within loops).
You have also learned to command your John Sands Sega to use the BEEP and CLS commands.
If this chapter has been a bit much for you, don't worry too deeply about it. Everything will come together the more you use your John Sands Sega.

Notes

CHAPTER 8

IF, THEN, GOTO AND WHAT HAVE YOU?

It is possible to ask your John Sands Sega certain questions, to which it will give you logical answers.
We'll show you what we mean.
IF I type CAR – THEN you (the computer) type ENGINE.
Or,
IF I type COW – THEN you (the computer) type MILK.

So let's do it.
Type the following:

```
10 PRINT ''CAR OR COW?''
20 INPUT C$
30 IF C$ = ''CAR'' THEN 100
40 IF C$ = ''COW'' THEN 200
100 PRINT ''ENGINE''
110 END
200 PRINT ''MILK''
```

Now RUN the programme a few times, changing CAR to COW when you are asked the question.
Here's what's going on inside your Sega.
IF you type CAR THEN...
Line 30 directs your programme to line 100.
This line prints ENGINE.
As this is the correct decision, we don't want the programme to continue to line 200. So when it gets to line 110 the computer is told to END.

However, IF you type COW THEN...
Line 40 sends the Sega down to line 200 and it prints MILK.
As this is the last line of the programme, we do not have to tell the programme to end. It will do so automatically on the last line.
Now let's assume that after you'd RUN the above programme, you typed in CAT. Your Sega will reply ENGINE!
The reason is that IF the condition in line 40 is not true, the THEN section of the line is totally ignored and the Sega will move on to the next programme line.
By adding two more lines to your programme, your Sega will ask to retype the word if you type anything but CAR or COW.
These lines are:

50 PRINT ''NO.NO.TYPE CAR OR COW''
60 GOTO 20

Instead of ending the programme after the computer replies, add further commands which will make it go back and ask you to type CAR or COW again.
(A clue. You'll need to change line 110 and add another line 210.)

Write your programme here:

What did your programme look like?
Ours looked like this:

```
10 PRINT ''CAR OR COW?''
20 INPUT C$
30 IF C$ = ''CAR'' THEN 100
```

```
40 IF C$ = ''COW'' THEN 200
50 PRINT ''NO.NO.TYPE CAR OR COW''
60 GOTO 20
100 PRINT ''ENGINE''
110 GOTO 10
200 PRINT ''MILK''
210 GOTO 10
```

Study this programme and try and understand the differences between the IF/THEN and GOTO lines.

Things to remember for all time about IF/THEN and GOTO.

1. IF/THEN is ALWAYS conditional.
   In other words, you only follow the commands if the condition (such as C$ = CAR or C$ = COW) is true.
2. GOTO is NEVER conditional.
   Wherever and whenever GOTO occurs, it has to be obeyed.

So much for Chapter 8.
In which you have learned some vital programming information.
This is the stuff of which really smart programmes are made and if you understand these concepts, you're soon going to find yourself becoming an equally smart programmer!

Chapter 8 Summary

You have learned the important basics of IF/THEN and GOTO.
(You'll find these commands can also be used with your FUNC key too.
Have you been doing this all the time anyway?
We thought you might!)
You have also learned about the END statement.

Notes:

CHAPTER 9

RANDOM IDEAS CAN BE FUN

There's a word in the BASIC language called RND – which is short for RANDOM.
You can have a lot of fun with this word, and even if you don't intend on using your John Sands Sega for game playing purposes, it's good to know how to use it.
There's another word which is often used in programmes, and that's CURSOR.
This chapter will introduce you to both of these words and the things they can do.
Let's get on with it!

Type:

```
10 PRINT RND (1)
```

Now RUN the programme.
Now RUN it again. And again.
The computer is picking a random number between 0 and 1.
The number which comes up each time, follows no pattern.
It is always totally unpredictable.

Now type this:

```
10 PRINT RND (1)
20 GOTO 10
```

RUN the programme. When you've had enough, press the BREAK key.

Now ask your Sega to pick random whole numbers from 1 to 100.

To do so, write this programme:

```
10 PRINT INT (RND (1) * 100) + 1
20 GOTO 10
```

If you would like your Sega to choose a random whole number between 1 and n (where n is any number), use this command:

INT (RND (1) * n) + 1

So. Try this for size:

```
10 PRINT INT (RND (1) * 6) + 1
20 GOTO 10
```

Now RUN it.
When you've seen enough, press BREAK.
As you may have realised, you've just created a computerised dice for yourself!
If you don't understand how line 10 works, don't worry. You're not really expected to. But now you know that it does work – regardless of how – and you can use it in future when creating your own programmes.
Here's a fairly easy question.
How would you change line 10 to choose a whole number between 1 and 10?
Write your answer here:

Our answer was:

```
10 PRINT INT (RND (1) * 10) + 1
```

Is that what you wrote?

Let's make your Sega sing!

We'll now command your Sega to make music for you – at random!

Type this for starters:

```
10 T = INT (RND (1) * 100) + 1
20 SOUND 2, T + 110, 15
30 FOR N = 1 TO 100
40 NEXT N
50 GOTO 10
```

Right. Now RUN it.
Press BREAK when you've heard enough.
This time we'll add a touch of the visual colors to your tune. So we'll command your Sega to produce a random color before it sounds each random tone.

It's done this way:

```
10 T = INT (RND (1) * 100) + 1
13 C = INT (RND (1) * 15) + 1
16 COLOR 1,C
20 SOUND 2,T + 100, 15
30 FOR N = 1 TO 100
40 NEXT N
50 GOTO 10
```

Now RUN.
And enjoy the fun!

Guess what!

This shows another facet of your Sega's capacity to generate random numbers.
We'll make our Sega choose a random number between 1 and 5.
See if you can guess it.

So now enter this:

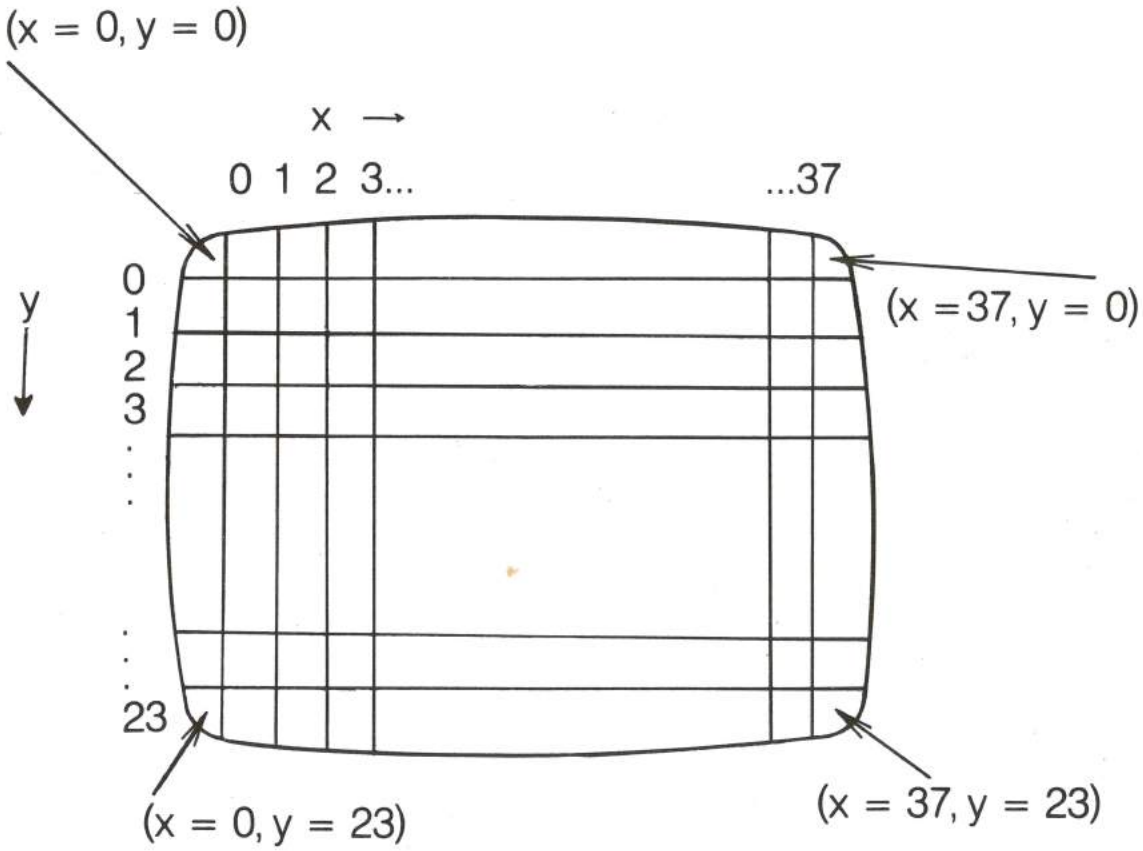
```
10 CLS
20 INPUT "PICK A NUMBER BETWEEN 1 & 5 "; N
30 IF N = INT (RND (1) * 5) + 1 THEN 100
50 PRINT "BAD LUCK, YOU'RE WRONG"
60 INPUT "PRESS CR TO PLAY AGAIN" ; Q$
70 GOTO 10
100 PRINT "RIGHT!!!!"
110 GOTO 60
```

Let's analyse what the programme is doing –
In line 20, the player decides his (or her) own INPUT – which is X – a number from 1 to 5. Your Sega then compares X with a random number from 1 to 5.
If X is not the same as the random number, the computer enters the 'Bad Luck' mode and goes back to line 10 to offer another chance.
Now we'll improve on our line 100 situation.

So write this:

```
10 CLS
20 INPUT "PICK A NUMBER BETWEEN 1 & 5"; N
30 IF N = INT (RND (1) * 5) + 1 THEN 100
50 PRINT "BAD LUCK, YOU'RE WRONG"
60 INPUT "PRESS CR TO PLAY AGAIN" ; Q$
70 GOTO 10
100 FOR X = 1 TO 10
110 CURSOR 0,10
120 PRINT "                    "
130 COLOR 1, INT (RND (1) * 13) + 2
140 CURSOR 0,10
150 PRINT "RIGHT!!!!"
160 T = INT (RND(1) * 5) + 1
170 SOUND 1,150 * T,15
180 FOR D = 1 TO 20
190 NEXT D
200 SOUND 0
210 NEXT X
220 GOTO 10
```

Now RUN the programme a few times and we'll go through what is happening.
Lines 100 to 210 are making your Sega generate sounds and colors and print RIGHT!!!! over and over again on the screen.
Lines 110 and 140 involve the use of the CURSOR. Which means the following:
The screen of your television set, or monitor, is divided into 38 characters per line horizontally (across) and 24 lines vertically (down), as shown in our diagram.



When you use the CURSOR command, it allows you to position printing in any spot on the screen. The first parameter specifies how many characters ACROSS to start printing from, the second, how many lines DOWN to start printing from.

There's a little idea we should mention here to do with the LISTING of your programme. From time to time we need to check our programme and make sure that it's coming along the right way. As you can imagine,

programmes can get very long when called upon to do complicated tasks. Rather than have to run right through the entire programme, you can ask your Sega to show it to you in sections or parts only. As an example, if you write LIST 50 – 130, that's what you'll get on the screen. Nothing before 50 and nothing after 130.
And here's another hint.
If you're listing a long programme, press the SPACE BAR when the listing begins. This will cause your Sega to pause the listing. When you want to see more, press the SPACE BAR again and a further section will appear on the screen.

Do it with money!

We're about to play a game of chance, but just because it's a game doesn't mean you're not learning anything!
You're gathering principles which will stay with you forever when programming your computer.
To play properly, we need two coins.
So we're going to have to programme your Sega to throw two coins at the same time and come up with two random numbers – each either 1 or 2.

So here we go:

```
10 CLS
20 X = INT (RND (1) * 2 + 1
30 T = INT (RND (1) *2 + 1
40 CURSOR 10,10
50 IF X = 1 THEN PRINT "HEADS"
60 IF X = 2 THEN PRINT "TAILS"
70 CURSOR 20,10
80 IF Y = 1 THEN PRINT "HEADS"
90 IF Y = 2 THEN PRINT "TAILS"
```

```
100 CURSOR 0,20
110 INPUT "LIKE ANOTHER THROW? " ; A$
120 IF A$ = "YES" THEN 10
130 END
```

RUN the programme a few times and then we'll have a closer look at what's happening.
Line 10 tells your Sega to CLEAR the screen.
Line 20 commands the computer to select a random number from 1 to 2 for one of the coins.
Line 30 does the same as line 20, but for the other coin.
Lines 40 to 90 tell the computer to PRINT the results on the screen.
Line 110 allows you to either keep the game going – by answering YES – or stopping it at line 130 if you answer anything else but YES.

Ever played two-up before? Here's your chance!

How about writing your own programme for a Two-Up game? You've just been through the BASIC commands you'll need to use. Combine them with the following information and see how well you do. The rules of Two-Up are as follows:

> The player throws two coins.
> If the coins land with a head and a tail showing, it is counted as 'no spin', and the player throws again.
> If the coins land with two heads showing, the player wins.
> If the coins land with two tails showing, the player loses.

This really is not complicated and you now have all the facts and programming details which will enable you to write the programme. Do it now. Not later! And write it here –

Here's our programme:

```
10 CLS
20 X = INT (RND (1) * 2) + 1
30 Y = INT (RND (1) * 2) + 1
40 CURSOR 10,10
```

```
50 IF X = 1 THEN PRINT "HEADS"
60 IF X = 2 THEN PRINT "TAILS"
70 CURSOR 20,10
80 IF Y = 1 THEN PRINT "HEADS"
90 IF Y = 2 THEN PRINT "TAILS"
100 CURSOR 0,20
110 IF X = Y THEN 140
120 INPUT "NO SPIN ... PRESS CR TO THROW
AGAIN " ; A$
130 GOTO 10
140 IF X = 1 THEN 170
150 PRINT "BAD LUCK, YOU LOSE"
160 GOTO 180
170 PRINT "YOU WON!!!!"
180 PRINT
190 INPUT "LIKE ANOTHER THROW?" ; A$
200 IF A$ = YES THEN 10
210 END
```

Chapter 9 Summary

In this chapter you've started to get yourself familiar with some pretty cunning programming concepts.
You've learned about RND which is fantastic and about CURSOR too, which may not be as fantastic, but is very necessary when you're working your way around the screen.
If there are any notes which come to mind now, jot them down.
Or things which you might wish to check up on later. Remember, there's no time like the present!

Notes:

CHAPTER 10

YOUR PERSONAL TUTOR

You can call upon your John Sands Sega to be your personal tutor! And it's nice to know that he (or she!) will never get tired, never make a mistake, never even take a lunch break unless you say so! Depending on you, the programmer, it is possible to create your personal tutor to be anything you wish! We'll show you what we mean. Using the RND command, we're going to get the computer to give us a few mathematical exercises in succession. Let's go!

Type this:

```
10 CLS
20 X = INT (RND (1) * 15) + 1
30 Y = INT (RND (1) * 15) + 1
40 PRINT "WHAT IS" ; X ; " * " ; Y
45 INPUT A
50 IF A = X * Y THEN 90
60 PRINT "THE ANSWER IS" ; X * Y
70 PRINT "OOOPS! TRY AGAIN!"
80 GOTO 100
90 PRINT "YOU GOT IT RIGHT!!!"
100 PRINT "PRESS CR WHEN YOU WANT
MORE!"
105 INPUT A$
110 GOTO 10
```

Now RUN the programme.
As you will see, it will test you on your multiplication tables from 1 to 15. It will also check your answers.
After you've spent some time at this, write a programme which will make the computer test you on mathematical additions using numbers from 1 to 100.

Write your programme here:

This is how we changed it –

```
20 X = INT (RND (1) * 100) + 1
30 Y = INT (RND (1) * 100) + 1
40 PRINT "WHAT IS" ; X ; " +" ; Y
45 INPUT A
50 IF A = X + Y THEN 90
60 PRINT "THE ANSWER IS" ; X + Y
```

If you'd like to add more interest into the programme by asking your John Sands Sega to keep a running total of all your correct answers, type this:

```
15 T = T + 1
95 C = C + 1
98 PRINT "YOU HAVE" ; C ; " OUT OF" ; T ;
" RIGHT ANSWERS"
```

T is counting the total number of questions the computer is asking you. When you first start to run the programme, T equals zero. Then every time the line 15 is reached, another 1 is added to T.
The same goes for C. It counts the number of times you produce a correct answer. As C is on line 95 it can only increase by number if your answer is correct.
The principle this programme is based on allows you to create many more for yourself.

Add some lines to the programme which will command your John Sands Sega to carry out one or more of the following:

Make the screen change color whenever an answer is correct.
Create a special sound each time you get five consecutive answers correct.
Keep a count of all the questions you answer correctly, and those which are not answered correctly.

These are just some of our ideas. You can think of more yourself.

Write your own programme here –

Give your John Sands Sega the good word!

Let's teach your computer some words from the dictionary!
So first of all, type this:

```
10 DATA CAR, HOUSE, BOAT
20 FOR N = 1 TO 3
30 READ A$
40 NEXT N
```

Nothing happened did it?
Or nothing that you could see anyway.
But it did really.

To find out what your Sega was actually doing, type this line too.

```
35 PRINT "A$ = " ; A$
```

Line 30 has told the computer to –

1. Search for a DATA line.
2. READ the first item on the list (CAR).
3. Make the variable A$ equal to "CAR".
4. "Cross out" CAR.

The second time the computer reached line 30, it was commanded to do the same things –

1. Search for a DATA line.
2. READ the first item on the list. (This time it was HOUSE).
3. Make the variable A$ equal to "HOUSE".
4. "Cross out" HOUSE.

And it will then do the same thing to BOAT.
Now. If you want the computer to READ the list over again, there could be a problem because it has already crossed everything out.

Let's check it. Type:

60 GOTO 10

Now RUN the programme.
See what happens?
It printed ? Out of data error in 30
And it's out of data because you asked it to cross out those words.

Now type this:

50 RESTORE

Once you have done this, the computer will act as if it has not crossed out anything and will keep READING the list over and over again until you tell it to stop.
An interesting thing about DATA lines is that you can place them anywhere you want to within the programme.

Here are three examples which show you what we mean.

```
30 FOR N = 1 TO 4
40 READ A$
50 PRINT A$
60 NEXT N
70 DATA ROSES ARE RED
80 DATA VIOLETS ARE BLUE
90 DATA IF I HAD YOUR FACE
100 DATA I'D JOIN THE ZOO!
```

```
30 FOR N = 1 TO 4
35 DATA ROSES ARE RED
40 READ A$
45 DATA VIOLETS ARE BLUE
50 PRINT A$
55 DATA IF I HAD YOUR FACE
60 NEXT N
65 DATA I'D JOIN THE ZOO!
```

```
10 DATA ROSES ARE RED
20 DATA VIOLETS ARE BLUE
30 FOR N = 1 TO 4
40 READ A$
50 PRINT A$
60 NEXT N
70 DATA IF I HAD YOUR FACE
80 DATA I'D JOIN THE ZOO!
```

See how they all achieve the same effect?
You'll find this principle helpful for things you do in the future.

Now test your geography!

Type this list of countries and their capitals:

```
10 DATA AUSTRALIA, CANBERRA
20 DATA ENGLAND, LONDON
30 DATA U.S.A., WASHINGTON D.C.
40 DATA CHINA, PEKING
50 DATA NEW ZEALAND, AUCKLAND
```

Now we'll tell the computer to select a random word from the list. There are 10 words in the list, so try this –

```
60 N = INT (RND (1) * 10) + 1
70 FOR A = 1 TO N
80 READ X$
90 NEXT A
100 PRINT "THE COUNTRY IS" ; X$
```

RUN it a few times and see how you go.
It's not really doing what we want it to is it? This is because your Sega is just as likely to choose one of the CAPITAL words instead of the others. What we want it to do is only to choose a random word from the first, third, fifth, seventh or ninth word. Or AUSTRALIA, ENGLAND, U.S.A., CHINA, NEW ZEALAND.
So now we'll make it do just that. By using our IF/THEN and INT commands.

Let's type this:

```
65 IF INT (N / 2) = N / 2 THEN N = N - 1
```

Now RUN the programme several times.
Eureka! You have it!
What INT does is to instruct the computer to only look at the whole part of a number, and to ignore the decimal part of it.
Your Sega, for instance, would see INT (4.2) as 4.
This is what's happening in line 65:

Say the computer picks the random number 8.
It then carries out this calculation –

INT (8 / 2) = 8 / 2
INT (4) = 4
4 = 4

As it is TRUE that 4 does equal 4, your Sega goes on to complete the THEN section of the line and makes N equal to 7(8–1).

On the other hand, if the computer had picked 5 as its random number, this is what would happen –

INT (5 / 2) = 5 / 2
INT (2.5) = 2.5
2 = 2.5

Because it is NOT TRUE that 2 equals 2.5, the computer does not subtract 1 from N – and 5 remains 5.

If all that seems a little involved for you, go back over it again until you do understand. It's worth the little time it will take.
Now that we've established that your Sega is able to READ a random word, you will realise that it can also READ the definition of the word.

So we'll now add these lines to your programme:

```
110 READ Y$
120 PRINT "ITS CAPITAL IS" ; Y$
```

Now RUN it a few times.

If you'd like the computer to print one random word and its meaning after the next, add these two lines:

```
130 RESTORE
140 GOTO 60
```

This is how your programme should look now:

```
10 DATA AUSTRALIA, CANBERRA
20 DATA ENGLAND, LONDON
30 DATA U.S.A., WASHINGTON D.C.
40 DATA CHINA, PEKING
```

```
50 DATA NEW ZEALAND, AUCKLAND
60 N = INT (RND (1) * 10) + 1
65 IF INT (N / 2) = N / 2 THEN N = N – 1
70 FOR X = 1 TO N
80 READ X$
90 NEXT A
100 PRINT "THE COUNTRY IS: " ; X$
110 READ Y$
120 PRINT "ITS CAPITAL IS; " ; Y$
130 RESTORE
140 GOTO 60
```

Now we want you to add a few more commands of your own. Instruct your Sega to carry out the following instructions when it is running the above programme:

1. PRINT the country only.
2. Ask you for the capital.
3. Compare your capital with the correct random capital.
4. Inform you if your answer is correct. If it is not, make your Sega print the correct word.

Write your programme here –

This is how ours finally worked out –

```
10 DATA AUSTRALIA, CANBERRA
20 DATA ENGLAND, LONDON
30 DATA U.S.A., WASHINGTON D.C.
40 DATA CHINA, PEKING
50 DATA NEW ZEALAND, AUCKLAND
60 N = INT (RND (1) * 10) + 1
65 IF INT (N / 2) = N / 2 THEN N = N - 1
70 FOR X = 1 TO N
80 READ X$
90 NEXT A
100 PRINT "THE COUNTRY IS: " ; X$
110 READ Y$
120 INPUT "ITS CAPITAL IS " ; R$
130 RESTORE
140 IF R$ = Y$ THEN 180
150 PRINT "NO! NO! NO!"
160 PRINT "THE RIGHT ANSWER IS " ; Y$
170 GOTO 60
180 PRINT "YOU'RE RIGHT!!!!"
190 GOTO 60
```

If you'd like to improve your programme with other lines, go for it! Have it create a light show, or use sounds.
Your final programme can be as exciting as your imagination!
You can also use the principle we have demonstrated here to create endless other programmes. You could create foreign word dictionaries, famous rivers and their countries, well known songs and their singers, films and actors/actresses, books and their authors, and dozens of others.
Come on – now you think of some for yourself.

Chapter 10 Summary

You've made another giant programming step in this chapter.
The BASIC words you've learned about include:

DATA
READ
RESTORE
INT

Remember, too, what we said about entering DATA lines?
They can be anywhere in the programme.
Don't forget that you don't have to follow every single programme we suggest. Once you have understood what is happening, it's a good idea to add some refinements of your own within our suggestions.

Notes:

CHAPTER 11

MEET THE MATHEMATICAL WIZARD!

The more complicated mathematical formulae become, the more help your John Sands Sega will appreciate in trying to understand them. Before we go into this, we'll just explain how your Sega goes about solving arithmetical problems. It is always in this order:

1. Exponentially! Wow – what a word. It means it solves problems relating to the power of numbers first.
2. Multiplication or Division operations next.
3. Addition and Subtractions last.
4. In the case of a "dead heat" – such as more than one Multiplication/Division or Addition/Subtraction the operations are carried out from left to right.

Here we go then.
We want your Sega to solve this problem:

Divide the sum of 8 + 2 by 5

One way of expressing this is as follows:

8 + 2/5 = 10/5 = 2

But your Sega won't solve it that way.

So type this:

PRINT 8 + 2/5

According to your Sega's rules what happened was this –
First it carried out the division operation, dividing 2 by 5 and arriving at 0.4. Then it did the addition by adding 8 together with 0.4 to make 8.4.

To make your Sega solve the problem another way, use parenthesis.

So type this line:

PRINT (8 + 2) / 5

Every time your Sega sees something in parenthesis, it carries out that function first.

Knowing what you now do, write down how you think your computer will answer the following problems:

PRINT 12 + 4 – 1 / 5
PRINT (12 + 4 – 1) / 5
PRINT 12 + (4 – 1 / 5)
PRINT 12 + (4 – 1) / 5

Once you've written the answers, insert each command line and see how you went.
Interesting eh?
Now then. How would you go about asking your Sega to solve this problem:

Divide 12 plus the difference of 4 minus 1 by 5

What you're asking your computer to do is this:

(12 + (4 – 1)) / 5

When more than one set of parenthesis is involved, your Sega solves the inside one first, then moves to the outside one. In the above problem it works this way –

(12 + (4 – 1)) / 5
4 – 1 = 3
(12 + 3) / 5
12 + 3 = 15
15 / 5
15 / 5 = 3

Here's one for you to work out.

Add your own parentheses (that's the plural of parenthesis!) to the following numbers so that the computer will print 38 as the answer:

PRINT 45 – 8 – 2 – 5 – 4

And the answer is:

PRINT 45 – (8 – (2 – (5 – 4)))

First of all your Sega subtracted 4 from 5 – making 1.
Then it subtracted the 1 from the 2 – leaving 1.
Then it subtracted the 1 from the 8 – leaving 7.
Then it subtracted the 7 from the 45 – leaving 38.
Q.E.D.!!!!!

Smart clues and shortcuts

To help you programme the computer without spending hours at the keyboard, we'd like to introduce you to some pretty smart clues and shortcuts to make things happen even more quickly!
There's a cunning way to handle complicated formulae – using what are called SUBROUTINES.

Before we explain, type this:

```
10 PRINT "CARRYING OUT MAIN
PROGRAMME"
20 GOSUB 500
30 PRINT "I'M BACK IN MAIN PROGRAMME"
40 END
500 PRINT "CARRYING OUT SUBROUTINE"
510 RETURN
```

Right.
Line 20 is telling your Sega to GO to the SUBroutine which begins at line 500.
Line 510 RETURN is telling the computer to return back to the BASIC word following GOSUB.
Now delete line 40 and RUN the programme.
The following should now appear on your screen –

CARRYING OUT MAIN PROGRAMME
CARRYING OUT SUBROUTINE
I'M BACK IN MAIN PROGRAMME
CARRYING OUT SUBROUTINE
? RETURN without GOSUB error in 510

Do you understand why deleting line 40 END caused this error?
This was because your Sega carried out the programme just like it did before you deleted the END line. It was sent to line 500 by the GOSUB command and then it returned to the BASIC word which follows GOSUB. Because you deleted END, it moved to the next line of the programme which was the SUBroutine. When it arrived at RETURN it did not have a clue where to return to; as it had not been sent to the SUBroutine by a GOSUB.

Think about that for a little while and then type the following:

```
10 INPUT "CHOOSE A NUMBER " ; N
20 GOSUB 1000
30 PRINT "THE FACTORIAL OF" ; N ; " IS" ; A
40 PRINT : GOTO 10
1000 REM CALCULATE FACTORIAL
1010 A = 0 : IF N < 1 THEN 1060
```

```
1020 A = 1 : IF N = 1 THEN 1060
1030 FOR X = 2 TO N
1040 A = A * X
1050 NEXT X
1060 RETURN
```

We added two new ideas into this programme.
Into lines 40, 1010 and 1020 we combined more than one command by using a colon to separate them.

In effect, line 40's commands are these:

PRINT
and
GOTO 10

Also in line 1000 you have met another word – REM
REM has no significance to your Sega – in fact it will ignore any line which starts with REM. You can insert one or more REM lines anywhere you so desire to help you REMember what your programme is doing. The REM lines have no effect at all on the programme. They can be inserted just for your own help.

To illustrate this, type the following, then RUN it.

```
10 REM WHAT'S GOING ON?
20 REM I'LL FIX YOU
30 REM KEEP AWAY FROM THE SUBROUTINE
```

So that takes care of that!
Now it's time for more brainwork from you.
Write a programme to make the computer print a table of squares (which is a number to the power of 2) for each number from 4 to 12.
Fair enough?

Write your programme here –

Here's ours:

```
10 FOR X = 4 TO 12
20 GOSUB 1000
30 PRINT X ; " SQUARED EQUALS" ; X X
40 NEXT X
50 END
1000 REM CALCULATE SQUARE
1010 X X = X * X
1020 RETURN
```

How much will your savings grow?

The things you have learned in this chapter will enable you to let your Sega do all the hard work, by programming complicated formulae in your SUBroutines.

Here's a programme that contains two SUBroutines, and if you put an equal amount of money into your Building Society each month, you can see how much you're going to have at the end of the year!

So let's type this:

```
10 INPUT "MONTHLY DEPOSIT  " ; D
20 INPUT "ANNUAL INTEREST RATE  " ; I
30 I = I / 12 * .01
40 INPUT "NUMBER OF DEPOSITS  " ; P
50 GOSUB 1000
60 PRINT "YOU'LL GET $" ; FV ; "  IN" ; P ;
" MONTHS"
70 END

1000 REM COMPOUND MONTHLY INTEREST
FORMULA
1010 N = I + 1
1020 GOSUB 2000
1030 FV = D * ((E - 1) / I)
1040 RETURN

2000 REM FORMULA TO RAISE NUMBER TO A
POWER
2010 E = 1
2020 FOR X = 1 TO P
2030 E = E * N
2040 NEXT X
2050 IF P = 0 THEN E = 1
2060 RETURN
```

You'll see that we've got one of our SUBroutines contacting another one. As long as you have a GOSUB to send your Sega to each SUB-routine, you can have as many as you will ever need! But you must remember to have a RETURN in each SUBroutine to make sure your computer returns to the BASIC word which follows each GOSUB.

Chapter 11 Summary

This chapter really got you going in the mathematical field with SUBroutines. Not forgetting the order in which your Sega carries out mathematical operations.

You've learned about the BASIC words of –

GOSUB
RETURN
REM

Remember about REM?
It's just for you – not the computer.
You've learned, too, about these symbols –

( )
:

Anything you feel like jotting down? Do it now.

Notes:

CHAPTER 12

THE WONDERS OF WORDS

Your Sega's brilliance extends to being able to perform wonderful things with letters, words and sentences.
You can even teach your computer to read and write!
So let's get underway and show you what we mean.

Type this:

```
10 PRINT ''WRITE A SENTENCE''
20 INPUT A$
30 PRINT ''THAT SENTENCE HAS'' ; LEN (A$) ;
'' CHARACTERS''
40 INPUT ''FEEL LIKE ANOTHER ONE?'' ; Y$
50 IF Y$ = ''YES'' THEN 10
```

See that LEN we put into line 30?
This command tells your Sega to compute the LENgth of the string A$ – your sentence. The computer then counts each individual character in the sentence, including spaces and punctuation marks.

Now erase that programme and type this one:

```
10 J$ = ''I LOVE YOU''
20 K$ = '' ''
30 L$ = ''AND YOU''
40 M$ = K$ + L$
50 N$ = ''AND ESPECIALLY YOU''
60 F$ = J$ + M$ + K$ + N$
70 PRINT F$
```

When you RUN the programme you will see that your Sega is combining strings. The plus sign (+) commands it to do this.
Having seen how strings are combined, let's get your computer to take a string apart.

Here we go:

```
10 INPUT "ENTER A WORD" ; Z$
20 PRINT "THE FIRST LETTER IS :" ; LEFT$
(Z$, 1)
30 PRINT "THE LAST THREE LETTERS ARE :" ;
RIGHT$ (Z$, 3)
40 GOTO 10
```

Here's what's happening –

In line 10 you INPUT a string for Z$. We'll assume the string is COMPUTER.
Your Sega then made several calculations in lines 20 and 30 to find the first left letter and the last 3 right letters of the string.

C O M P U T E R

LEFT$ (Z$, 1) **RIGHT$ (Z$, 3)**

Now RUN through this a few times until you perfectly understand what's going on.
When you do, change lines 20 and 30 to give you the first 3 letters and the last 7 letters of your string.
Write your answer here –

Our answer looks like this:

```
20 PRINT "FIRST THREE LETTERS ARE: " ;
LEFT$ (Z$, 3)
30 PRINT "LAST SEVEN LETTERS ARE: " ;
RIGHT$ (Z$, 7)
```

We'll move on to other things.
First, erase your last programme.

Now enter this one:

```
10 INPUT "WRITE A SENTENCE" ; A$
20 PRINT "ENTER A NUMBER FROM 1 TO " ;
LEN (A$)
30 INPUT M
40 PRINT "MIDSTRING WILL START WITH
CHARACTER" ; M
50 PRINT "ENTER A NUMBER FROM 1 TO" ;
LEN (A$) – M + 1
60 INPUT N
70 PRINT "MIDSTRING WILL BE" ; N ;
"  CHARACTERS LONG"
80 PRINT "THIS MIDSTRING IS: " ; MID$
(A$, M, N)
90 GOTO 10
```

Now RUN the programme quite a few times and try to understand what is going on with the MID$ command.
How the programme works is like this –

Let's suggest that when you were asked to write a sentence, your INPUT was "I LOVE MY SEGA"

Because you wrote the ; A$ after the sentence, the computer's memory said – "this is a string".
So in line 20, your Sega first calculated the LENgth of A$ – which is 14 characters.
Then it asked you to choose a number from 1 to 14. We'll say you chose number 7.
In line 50 you were asked to pick another number from 1 to 8 (14 – 7) plus 1. Say you chose 5.
In your Sega's memory, it would say to itself that M represented 7 and that N represented 5.
In line 80 your Sega gave you a MID string of A$ – which starts at character 7 and is 5 characters long.
Like this –

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14
I   L O V E   M Y    S E G A
MID$ (A$, 7, 5)
```

There are other ways to use MID$ too!

Erase the last programme and write this one:

```
10 INPUT ''WRITE A SENTENCE'' ; A$
20 INPUT "WRITE ONE WORD IN THE
SENTENCE'' ; Z$
30 L = LEN (Z$)
40 FOR X = 1 TO LEN (A$)
50 IF MID$ (A$, X, L) = Z$ THEN 90
60 NEXT X
70 PRINT "THAT WORD NOT IN SENTENCE"
80 END
90 PRINT Z$'' .... STARTS AT CHARACTER
NO..." ; X
```

Run the programme several times, then we'll tell you what's happening.

If you typed in the same sentence you used before, and the word you used for the INPUT Z$ was MY in line 30, your Sega counts the LENgth of Z$ – which in this case is 2 characters. The computer then moves to the FOR/NEXT loop of lines 40 to 60, counting every character in A$. It begins with character 1 and ends with character LEN (A$) – which is 14.

Each time your Sega counts a new character, it looks at a new MID string. Every MID string begins at character M and is L (or 2) characters long.

When M equalled 1, your Sega looked at the following MID string –

```
1 2 3  4  5  6 7 8  9 10 11 12 13 14
I  L O V E  M Y  S E G A
```

MID$ (A$ 1, 2)

The fourth time it went through the loop, when M was equal to 4, the Sega looked at this spot –

```
1 2 3  4  5  6 7 8  9 10 11 12 13 14
I  L O V E  M Y  S E G A
```

MID$ (A$, 4, 2)

When M equalled 6, it found the MID string it was looking for!

This device is a terrific help when you want to sort through a large amount of information. For instance, if you had a list of addresses of several hundred people living all over Australia, you could ask your Sega to just list those in a particular town, or by postcode numbers. Or whatever break-up you wish to use.

Editing your work, Sega style

Your John Sands Sega can really save you hours of typing and rewriting time.
Let's assume you have written a phrase to which you would like to add something else.

Type this:

10 A$ = ''MY SEGA''

Now add to the programme to make your Sega add the following words to the beginning of your sentence:

I LOVE

and then to PRINT the new, completed sentence.

I LOVE MY SEGA

Write your programme here –

This is how we programmed the Sega:

```
10 A$ = ''MY SEGA''
20 B$ = ''I LOVE''
30 C$ = B$ + ''  '' + A$
40 PRINT C$
```

Now add to this programme and make your Sega do the following –

1. Find the start of the MID string – SEGA
2. Delete the word SEGA, and form a new string –
   I LOVE MY
3. Add the following words to the new string –
   HOME COMPUTER
4. Print the new string –
   I LOVE MY HOME COMPUTER

Write your programme here –
(A clue. To create the string I LOVE MY, you have to get to the LEFT part of the string I LOVE MY SEGA. Right?)

This is how our programme looks.
Hope yours does too!

```
10 A$ = ''MY SEGA''
20 B$ = ''I LOVE''
30 C$ = B$ + ''  '' + A$
40 PRINT C$
50 N = LEN (''SEGA'')
60 FOR M = 1 TO LEN (C$)
70 IF MID$ (C$, M, N) = ''SEGA''
THEN 90
80 NEXT M
90 D$ = LEFT$ (C$, M – 1)
100 E$ = D$ + ''HOME COMPUTER''
110 PRINT E$
```

What you've just done is to utilise the basis of a true word processing programme. There are many more variations of this principle which you will come across later, as well as being able to develop new applications yourself.
Now we're going to get you really thinking!
We'd now like you to try and use what you have just learned and practised to write a programme which will do the following –

Instruct your Sega to ask you to:

INPUT a sentence.
DELETE a phrase within that sentence.
REPLACE that phrase with a new one.
PRINT the new sentence with your changes therein.

This might seem a little tricky at first, but if you make sure you have understood almost everything we have discussed in this chapter, you'll be able to do it.
If it doesn't work for the first couple of times, keep going.
Now write your programme here –

Here's our programme:

```
10 INPUT "SENTENCE: " ; A$
20 PRINT: PRINT A$ ; : PRINT
30 INPUT "REPLACE: " ; X$
40 INPUT "WITH: " ; Y$
50 LA = LEN (A$) : LX = LEN (X$)
60 FOR N = 1 TO LA-LX + 1
70 IF MID$ (A$, N, LX) = X$ THEN 110
80 NEXT N
90 PRINT " ' " ; X$ ; "  '  NOT FOUND"
100 END
110 L$ = LEFT$ (A$, N-1)
120 R$ = RIGHT$ (A$, LA-LX-N + 1)
130 A$ = L$ + Y$ + R$
140 PRINT: PRINT A$
150 END
```

Chapter 12 Summary

You've come a long way since Chapter 1, haven't you?
What we've covered in this chapter involves the use of the following BASIC words –

LEN
LEFT$
RIGHT$
MID$

And the BASIC string Operator – Plus sign (+)
There are bound to be some notes you want to jot down while your memory is still fresh. Do it here and now, while you remember!

Notes:

CHAPTER 13

SOMEONE TO WATCH OVER YOU!

That's the name of an old song, but we'll get around to musical notes in a minute or two.
Before we do, we'll tell you how to get your Sega to 'watch over you', to see if you are pressing anything on the keyboard.
The magic word for this, is INKEY$.

So now type this:

```
10 A$ = INKEY$
20 IF A$ < > "" GOTO 50
30 PRINT "YOU DIDN'T PRESS ANYTHING"
40 GOTO 10
50 PRINT "THE KEY YOU PRESSED WAS ...";A$
```

Did you remember that < > means 'not equal to'?
Now RUN the programme, and press any key while you're doing so.
The command INKEY$ orders your Sega to look at the keyboard and check whether you have pressed anything. This check is carried out at amazing speed.
Your Sega gives A$ the value of this key.
Then the computer decides what happened.
If A$ = "" – in other words, nothing – your Sega answers you with YOU DIDN'T PRESS ANYTHING and then goes back to line 10 to check again.
If, however, the A$ does equal something, your Sega will go to line 50 and print the key which was pressed.
If you require a continual 'watch', add this to your programme and then RUN it.

```
60 GOTO 10
```

You'll never beat your Sega in speed!
To see what keys you are pressing, simply remove line 30 and it will tell you.

Your Sega Piano!

Now we're going to make use of INKEY$ to turn your Sega into a piano! There's a table in the Appendix (Page 155) representing Musical Tone Numbers. The table lists the tonal numbers from which your Sega produces sounds. The octave ranging from middle C to the C above, is as follows:

C – 262	D – 294	E – 330	F – 349
G – 392	A – 440	B – 494	C – 523

It is a simple matter to command your Sega to SOUND one of these tones when a certain key is pressed.

So erase your last programme and type this one:

```
10 A$ = INKEY$
20 IF A$ = "" THEN 10
30 IF A$ = "A" THEN T = 262
40 IF A$ = "S" THEN T = 294
50 IF A$ = "D" THEN T = 330
60 IF A$ = "F" THEN T = 349
70 IF A$ = "G" THEN T = 392
80 IF A$ = "H" THEN T = 440
90 IF A$ = "J" THEN T = 494
100 IF A$ = "K" THEN T = 523
110 IF T = 0 THEN 10
120 SOUND 2, T, 15
130 T = 0
140 GOTO 10
```

Now RUN your programme.
Want to hear something? Then type any of the keys on the third row of your keyboard – from A to K.
Maybe you can remember the melody of 'Someone to watch over you'!

Do you know why this programme would not work if you had used INPUT instead of INKEY$?
By using INPUT, your Sega would wait for you to press CR before indicating what you are entering into the computer. When you use INKEY$, it recognises everything you type without you pressing the CR key.
You can also write this type of programme another way.
Which is by using READ and DATA lines.
READ and DATA lines also make it easier for you to add additional tones to your repertoire!

Here's how such a programme would look –

```
10 A$ = INKEY$
20 FOR X = 1 TO 8
30 READ B$, T
40 IF A$ = B$ THEN SOUND 2, T, 15
50 NEXT X
60 RESTORE
70 GOTO 10
80 DATA A, 262, S, 294
90 DATA D, 330, F, 349
100 DATA G, 392, H, 440
110 DATA J, 494, K, 523
```

Now RUN this one and see what happens.

Race your Sega Computer

First, type this programme:

```
10 X = INT (RND (1) * 4) + 1
20 Y = INT (RND (1) * 4) + 1
30 PRINT "WHAT IS" ; X ; " + " ; Y
40 T = 0
```

```
50 A$ = INKEY$
60 T = T + 1
70 SOUND 2, 110, 8
80 IF T = 15 THEN 200
90 IF A$ = "" THEN 50
95 SOUND 0
100 GOTO 10
200 CLS
210 SOUND 2, 500, 15
220 PRINT "NO TIME LEFT!!!"
230 SOUND 0
```

This is what is happening with the above programme –

Lines 10, 20 and 30 are getting your Sega to select two random numbers and asking you what their sum total adds up to.
Line 40 is setting T to 0 – because we're going to use T as a stop watch.
Line 50 is offering you the opportunity to answer the question – and giving you exactly 1 second to do so.
Line 60 is adding 1 to the timing device. So now T is equal to 1. The next time your Sega reaches line 60 , it adds 1 to the time and T then becomes 2. Each time the computer arrives at line 60 it will add another 1 to T.
We've put in line 70 just to keep you on your toes!
Line 80 instructs your Sega that you have got 15 chances to answer the problem. Once T is equal to 15, you've had it! No more time.
Lines 200, 210 and 220 are there to keep you in the picture!
Line 90 tells you that if you have not answered the question, you can go back and have another chance.
Only when you do answer the problem will your Sega go back to line 100 and set you another problem.

Now that you understand all that, make a change to the programme which will give you three times as much time to answer each question. Write your answer here –

The answer, of course, lies in line 80 – which should now appear as follows:

```
80 IF T = 45 THEN 200
```

Are you sure your answers are correct? Ask your Sega!

It's easy to make sure you haven't made a mistake! Just ask your John Sands Sega to check your answers for you.
Add these additions to your programme and see how well it does the job:

```
100 IF A$ = X + Y THEN 130
110 PRINT "INCORRECT", X ; " + " ; Y;
" = "; X + Y
120 GOTO 10
130 PRINT "CORRECT!"
140 GOTO 10
```

When you RUN this programme, and give your answer on time, this message will appear on the screen of your monitor –

? Type mismatch error in 100

Remember what we said about mixing strings with numbers in Chapter 2? That it's not allowed?
The ERROR message above, came about because we cannot make A$ (a string) equal to X + Y (numbers). So we've got to change A$ to a number.

Here's how we can do it:

100 IF VAL (A$) = X + Y THEN 130

By writing line 100 this way, VAL (A$) turns A$ into its numeric VALue.
If A$ equals the string "5", then VALue equals the number 5.
If A$ equals the string "C", then VAL (A$) must equal 0 – because "C" has no numeric value.

Chapter 13 Summary

We've learned about another two very important BASIC words in this chapter.

They are –

INKEY$ and VAL

We've made music and used a timing programme – not forgetting the fact that we've learned how to ask our Sega to check answers for mistakes.
It sure is a smart computer!

Notes:

CHAPTER 14

GETTING EVERYTHING IN ORDER

We often need ways to organise large amounts of information.
Your Sega has its own organisation programming system to make the job far easier than it's ever been!
Suppose, for instance, that you want to keep a record of your expenses on a four day holiday, as follows:

DAY 1 $82
DAY 2 $75
DAY 3 $87
DAY 4 $79

To get your Sega to remember these figures, type:

A = 82: B = 75: C = 87: D = 79

But there's a better way. Now type:

A (1) = 82: A (2) = 75: A (3) = 87: A (4) = 79

These two lines work in the same way.

We'll prove it. Type:

PRINT A; B; C; D

Now type:

PRINT A (1); A(2); A (3); A (4)

See! The effect is the same.
So why is the second method better than the first?

Take a look at the following two programmes.

```
10 DATA 82, 75, 87, 79
20 READ A, B, C, D
30 INPUT "DAY (1–4): "; X
40 IF X < 1 THEN 30
50 IF X > 4 THEN 30
60 IF X = 1 THEN PRINT "$" ; A
70 IF X = 2 THEN PRINT "$" ; B
80 IF X = 3 THEN PRINT "$" ; C
90 IF X = 4 THEN PRINT "$" ; D
100 GOTO 30
```

```
10 DATA 82, 75, 87, 79
20 DIM A (4)
30 FOR N = 1 TO 4
40 READ A (N)
50 NEXT N
60 INPUT "DAY (1–4): " ; X
70 IF X < 1 THEN 60
80 IF X > 4 THEN 60
90 PRINT "$" ; A (X)
100 GOTO 60
```

Both of those programmes do the same thing, but the first one uses ordinary variables. The second doesn't. The advantage of this method is that it makes it easier for you to programme for large amounts of information.

Now RUN the second programme.

This is how it works –

LINE 20 tells the Sega to save enough room in its memory to store a list, or ARRAY, of numbers.
The array has four items and is called A.
(You should always use the DIMension statement for every array you use).
LINES 30 and 50 are a LOOP to read the data into the array.
LINE 60 asks you the day, or the array item which you require.
LINES 70 and 80 checks the validity of your input.
LINE 90 prints the information you requested.

Now that you have your holiday expenses stored in an array, it is very easy to use them.

For instance, if you would like your programme to be able to alter the figures, add these lines:

```
92 INPUT "WANT TO ALTER THIS? " ; R$
94 IF R$ = "NO" THEN 60
96 INPUT "NEW VALUE: " ; B
98 A (X) = B
```

Or you may wish to print a list of your holiday expenses.

To do this, add these lines:

```
52 INPUT "LIST ALL VALUES? " ;R$
54 IF R$ = "YES" THEN GOSUB 200
200 PRINT "DAY", "AMOUNT"
210 FOR N = 1 TO 10
220 PRINT N, A (N)
```

230 NEXT N
240 RETURN

And change line 100 to:

100 GOTO 52

Hooray for arrays!

Now let's try something else.
Arrays are commonly used to store accounting information.
So command your Sega to help a company keep track of its sales figures on ten products.
Write your programme here –

Here's how ours looks:

```
10 DATA 107, 22, 45, 80, 138
20 DATA 83, 165, 46, 144, 72
30 DIM S (10)
40 FOR N = 1 TO 10
50 READ S (N)
60 NEXT N
70 INPUT "ITEM NO: " ; X
80 IF X < 1 THEN 70
90 IF X > 10 THEN 70
100 PRINT "TOTAL SALES FOR ITEM" ; X ;
" IS" ; S (X)
110 GOTO 70
```

How does that compare with the one you wrote?
Right.

Now back to your holiday again.

When you print the expenses for the day, you might like to print the name of the day too.
How to do that? Simple!
You can create an array of string variables in the same way that you create an array of numeric variables.
So all you need to do is create a string array which contains four items – the name of each day.
Then simply print the relevant item from this array when you print the expense value.

To do this, enter this modified version of the programme:

```
10 DATA 82, 75, 87, 79
15 DATA FRIDAY, SATURDAY, SUNDAY, MONDAY
20 DIM A (4)
30 FOR N = 1 TO 4
40 READ A (N)
50 NEXT N
52 DIM D$  (4)
54 FOR N = 1 TO 4
56 READ D$ (N)
58 NEXT N
60 INPUT "DAY (1–4): "; X
70 IF X < 1 THEN 60
80 IF X > 4 THEN 60
90 PRINT D $ (X) ;":$" ; A (X)
```

It's time to put on your thinking cap!

Try and instruct your Sega to –

1. Print each grocery item from the list below, asking you if you require it.
2. Print a shopping list of requested items. The items are –

MILK	CHEESE
BREAD	BUTTER
YOGHURT	MUESLI
ICE CREAM	MEAT
RICE	FISH
ENERGY BAR	THYME

Write your programme here:

This is the way we wrote it:

```
10 DATA MILK, BREAD, YOGHURT, ICE CREAM
20 DATA RICE, ENERGY BAR, CHEESE, BUTTER
30 DATA MUESLI, MEAT, FISH, THYME
40 DIM Z$ (12)
50 FOR N = 1 TO 12
```

```
60 READ S$
70 PRINT "DO YOU NEED" ; S$ ;
80 INPUT A$
90 IF A$ = "YES" THEN X = X + 1 : Z$ (X) = S$
100 NEXT N
110 PRINT
120 PRINT "* * SHOPPING LIST * *"
130 PRINT
140 FOR N = 1 TO 12
150 PRINT Z$ (N)
160 NEXT N
```

Another Dimension

Let's get back to your expenses programme again.
What if you also wanted to store your companion's expenses?
You could always use two arrays instead of one – or you could do it a better way.
Each item in an array is represented by an array name, and a number in brackets – called a subscript.
By adding another subscript, we can store expense information for two people.
The first subscript notes the person to whom you are referring, the second, the day.
This is how your Sega would store the numbers –

	FRIDAY	SATURDAY	SUNDAY	MONDAY
YOU	A (1, 1) 82	A (1, 2) 75	A (1, 3) 87	A (1, 4) 79
YOUR FRIEND	A (2, 1) 74	A (2, 2) 80	A (2, 3) 83	A (2, 4) 86

To programme your Sega to do this, here's what you should type:

```
10 DATA 82, 75, 87, 79
20 DATA 74, 80, 83, 86
30 DATA FRIDAY, SATURDAY, SUNDAY, MONDAY
40 DIM (2, 4)
50 FOR M = 1 TO 2
60 FOR N = 1 TO 4
70 READ A (M, N)
80 NEXT N, M
90 DIM D$ (4)
100 FOR N = 1 TO 4
110 READ D$ (N)
120 NEXT N
130 INPUT "PERSON (1–2): " ; W
140 IF W < 1 THEN 130
150 IF W > 2 THEN 130
160 INPUT "DAY (1–4): " ; X
170 IF X < 1 THEN 160
180 IF X > 4 THEN 160
190 PRINT: PRINT "PERSON" ; W
200 PRINT "   " ; D$ (X); ": $" ; A (W, X)
210 PRINT: GOTO 130
```

In this programme, the array named A is technically called a TWO DIMENSIONAL ARRAY, because it uses two subscripts.
The array named D is a ONE DIMENSIONAL ARRAY as it only uses one subscript.
Storing information in a two dimensional manner gives you two ways of analysing the data.

Or, in other words, it can give you two perspectives of the data.
For instance, the above programme enables you to calculate total expenses for either yourself or your friend.
Or, you could calculate expenses for you both on any one specific day.
Got all that?
Great!
You can also store numbers in a THREE DIMENSIONAL ARRAY. That is, an array with three numbers or subscripts within the brackets. To see this, we'll expand your expenses programme even more. Now we'll analyse your expenses into travel, food and others.

Here's how:

```
10 DATA 51, 15, 16, 50, 10, 15
20 DATA 55, 16, 15, 48, 17, 14
30 DATA 50, 14, 10, 53, 17, 15
40 DATA 54, 17, 17, 55, 17, 19
50 DATA FRIDAY, SATURDAY, SUNDAY, MONDAY
60 DATA TRAVEL, FOOD, OTHERS
70 DIM A (2, 4, 3)
80 FOR M = 1 TO 2
90 FOR N = 1 TO 4
100 FOR P = 1 TO 3
110 READ A (M, N, P)
120 NEXT P, N, M
130 DIM D$ (4)
140 FOR N = 1 TO 4
150 READ D$ (N)
160 NEXT N
170 DIM E$ (3)
180 FOR N = 1 TO 3
```

```
190 READ E$ (N)
200 NEXT N
210 INPUT "PERSON (1-2): "; W
220 IF W < 1 THEN 210
230 IF W > 2 THEN 210
240 INPUT "DAY (1-4): " ; X
250 IF X < 1 THEN 240
260 IF X > 4 THEN 240
270 INPUT "CATEGORY (1-3): " ; Y
280 IF Y < 1 THEN 270
290 IF Y > 3 THEN 270
300 PRINT: PRINT "PERSON" ; W ;" " ; E$ (Y)
310 PRINT "    " ; D$ (X) ;" $" ; A (W, X, Y)
320 PRINT: GOTO 210
```

To make the best use of all three dimensions, delete lines 210-320 and add the following:

```
210 CLS; CURSOR 0, 2
220 PRINT " 1. FOR PERSON"
230 PRINT " 2. FOR DAY"
240 PRINT " 3. FOR CATEGORY"
250 PRINT: INPUT "ENTER SELECTION: " ; N
260 ON N GOSUB 1000, 2000, 3000
270 GOTO 210

1000 CLS
1010 INPUT "PERSON (1-2): " ; W
1020 IF W < 1 THEN 1000
1030 IF W > 2 THEN 1000
1040 FOR M = 1 TO 3
1050 CURSOR M * 11-2, 4: PRINT E$ (M)
```

```
1060 NEXT M
1070 FOR N = 1 TO 4
1080 CURSOR 0, N * 2 + 4: PRINT D$ (N)
1090 NEXT N
1100 FOR M = 1 TO 3
1110 FOR N = 1TO 4
1120 CURSOR M * 11–2,N * 2 + 4
1130 PRINT "$" ; A (W, N, M)
1140 NEXT N, M
1150 GOTO 5000

2000 CLS
2010 INPUT "DAY (1–4):" ; X
2020 IF X < 1 THEN 2000
2030 IF X > 4 THEN 2000
2040 FOR M = 1 TO 2
2050 CURSOR M * 11, 4: PRINT "PERSON" ; M
2060 NEXT M
2070 FOR N = 1 TO 3
2080 CURSOR 0, N * 2 + 4: PRINT E$ (N)
2090 NEXT N
2100 FOR M = 1 TO 2
2110 FOR N = 1 TO 3
2120 CURSOR M * 11 + 2, N * 2 + 4
2130 PRINT "$" ; A (M, X, N)
2140 NEXT N, M
2150 GOTO 5000

3000 CLS
3010 INPUT "CATEGORY (1–3): " ; Y
3020 IF Y < 1 THEN 3000
```

```
3030 IF Y > 3 THEN 3000
3040 FOR M = 1 TO 2
3050 CURSOR M * 11, 4 : PRINT "PERSON" ; M
3060 NEXT M
3070 FOR N = 1 TO 4
3080 CURSOR 0, N * 2 + 4: PRINT D$ (N)
3090 NEXT N
3100 FOR M = 1 TO 2
3110 FOR N = 1 TO 4
3120 CURSOR M * 11 + 2, N * 2 + 4
3130 PRINT "$"; A (M, N, Y)
3140 NEXT N, M
3150 GOTO 5000

5000 CURSOR 0, 20
5010 PRINT "PRESS SPACEBAR TO CONTINUE
. . ."
5020 IF INKEY$ < > "  " THEN 5020
5030 RETURN
```

Chapter 14 Summary

In this chapter you have mainly learned about ARRAYS and the way they can be used singly, or in multiples, to provide all kinds of analysed information.
Anything you especially want to remember? Jot it down here.

Notes:

CHAPTER 15

FINISHING TOUCHES

In this chapter, we're going to tell you about a few more BASIC words which will let you add the professional touch to all your programmes. They will also make every task easier too!
The first new word is STOP.

Enter this programme:

```
10 A = 1
20 A = A + 1
30 STOP
40 A = A * 2
50 STOP
60 GOTO 20
```

Now RUN it.

Your John Sands Sega replies with –

```
Break in 30
Ready
```

The computer was ordered to STOP carrying on with the programme when it reached line 30.
Here, you can type a command line to check what your programme has done up to that stage.

Now type this:

```
PRINT A
```

Your Sega prints 2.
Which is the value of A when it reached line 30 of the programme.

Now type this:

```
CONT
```

See?
Your Sega CONTinues to run the programme from the point where it STOPped.
It ran it at line 40.
Then it came to line 50 where it was ordered to STOP again.

So it printed –

Break in 50

Now type:

PRINT A

This time, the Sega prints 4.
Which is the value of A at line 50.
Now type CONT again and the computer will go back to line 30.
If you ask it to PRINT A, it will give you the reply, 5.
Which is the value of A at line 30 on the second run through the programme.
The BASIC words STOP and CONT are to use when your programme isn't working as you hoped it would.
When you place STOP lines in your programmes you can analyse step by step what's going on.
Once you've corrected the programme, simply delete the stop lines!

A typing tutorial

Enter the following programme:

```
10 INPUT "WRITE 1, 2 OR 3" ; N
20 ON N GOSUB 100, 200, 300
30 GOTO 10
100 PRINT "YOU WROTE 1"
110 RETURN
```

```
200 PRINT "YOU WROTE 2"
210 RETURN
300 PRINT "YOU WROTE 3"
310 RETURN
```

Now RUN the programme.

We really could replace line 20 by the following three lines:

```
18 IF N = 1 THEN GOSUB 100
20 IF N = 2 THEN GOSUB 200
22 IF N = 3 THEN GOSUB 300
```

But it saves lines when we use ON/GOSUB
ON/GOSUB commands your Sega to look at the number which follows ON, which in this example is number N.
If N is a 1, the Sega moves to the subroutine starting at the 1st line following GOSUB.
If N equals 2, then it goes to the subroutine starting at the 2nd line number.
If N equals 3, the Sega moves to the third line number.
Because there is no 4th number in the above programme, the Sega then moves on to the next line and continues.

Now we'll really use ON/GOSUB in a more complex programme.

```
10 CLS
20 A = INT(RND(1) * 100) + 1
30 B = INT(RND(1) * 100) + 1
40 PRINT "<1> ADDITION"
50 PRINT "<2> SUBTRACTION"
60 PRINT "<3> MULTIPLICATION"
70 PRINT "<4> DIVISION"
80 PRINT: INPUT "WHICH ONE (1-4)?";X
```

```
90 ON X GOSUB 1000, 2000, 3000, 4000
100 FOR N = 1 TO 500: NEXT N
110 GOTO 10

1000 CLS
1010 PRINT "WHAT IS";A;"  + ";B;"?"
1020 INPUT X
1030 IF X<>A+B THEN 1060
1040 PRINT "RIGHT!!"
1050 RETURN
1060 PRINT "NO! NO! NO!"
1070 RETURN

2000 CLS
2010 PRINT "WHAT IS";A;"  -";B;"?"
2020 INPUT X
2030 IF X<>A-B THEN 2060
2040 PRINT "RIGHT!!"
2050 RETURN
2060 PRINT "NO! NO! NO!"
2070 RETURN

3000 CLS
3010 PRINT "WHAT IS";A;"  *";B;"?"
3020 INPUT X
3030 IF X<>A*B THEN 3060
3040 PRINT "RIGHT!!"
3050 RETURN
3060 PRINT "NO! NO! NO!"
3070 RETURN

4000 CLS
4010 PRINT "WHAT IS";A;"  /";B;"?"
```

```
4020 INPUT X
4030 IF X<>A/B THEN 4060
4040 PRINT "RIGHT!!"
4050 RETURN
4060 PRINT "NO! NO! NO!"
4070 RETURN
```

ON/GOTO can be used just like ON/GOSUB. The difference being that ON/GOSUB sends the Sega on to a subroutine and ON/GOTO sends it to another line number.
The following is part of a programme which uses ON/GOTO.

Try it now.

```
10 CLS
15 CURSOR 5, 0
20 PRINT "(1) ALIEN ZAP!"
25 CURSOR 5, 2
30 PRINT "(2) SHARK WAR"
35 CURSOR 5, 4
40 PRINT "(3) LOVE MATCH"
45 CURSOR 0, 7
50 PRINT "NAME YOUR GAME"
60 INPUT A
70 CLS
80 ON A GOTO 300, 400, 500
300 PRINT "ALIEN ZAP! GAME"
310 END
400 PRINT "SHARK WAR GAME"
410 END
500 PRINT "LOVE MATCH"
510 END
```

Got the hang of it?

Got enough room?

It's always a good idea, especially when you get into writing long programmes, to check that there's enough room in your Sega's memory to hold everything you write.
Here's how to do it.

Type:

PRINT FRE

Your Sega will then print out how much storage space remains in the memory for you to work with.
You can type PRINT FRE at any time during your programming work to check how much memory is left. But don't type NEW!!!! Or you will erase your programming up to that point.
Every time you depress a key, you are using part of the memory. That goes for spaces too – even if you cannot see anything on the screen.
You can save memory by leaving out unwanted spaces in your programme – such as before and after BASIC words, punctuations and symbols.

For instance, the following programming line –

50 X = INT (RND (1) * 4) + 1

will work just as well if it is written like this –

50X=INT(RND(1)*4)+1

In this book, we have spread our commands out in order that it's easier for you to read and understand them. When you get more and more familiar with command lines, it won't worry you at all to keep them fairly condensed.

STR$, SGN, ABS and other mathematical magic

Now we're going to show you how to convert a number to a string.

Here's an example:

```
10 INPUT "WRITE A NUMBER" ;N
20 A$ = STR$(N)
30 PRINT A$ + " IS NOW A STRING"
```

RUN this one too and you'll soon understand how it all works.

Some numbers are positive, others are negative and yet others are represented by 0.

So type the following:

```
10 INPUT "WRITE A NUMBER";X
20 IF SGN(X) = 1 THEN PRINT "POSITIVE"
30 IF SGN(X) = 0 THEN PRINT "ZERO"
40 IF SGN(X) = -1 THEN PRINT "NEGATIVE"
50 GOTO 10
```

Now RUN your programme.

Now INPUT the following numbers and see what happens –

28 –14 0 –1.250 14.6 0.2234

It's time now to talk about ABS.

ABS is a BASIC word for ABSolute and it tells you the ABSOLUTE value of a number – in other words, the magnitude of a number without respect to its sign (or SGN).

Enter the following :

```
10 INPUT "TYPE A NUMBER" ;N
```

```
20 PRINT ''THE ABSOLUTE VALUE IS'' ; ABS(N)
30 GOTO 10
```

Now RUN the programme and INPUT any numbers you feel like, using positive, negative or zero signs.

Before we leave this chapter, please will you write and RUN the following programme:

```
10 X = 1
20 PRINT X
30 X = X*10
40 GOTO 20
```

Sometimes a number becomes so small or so large that the computer cannot print out all the numbers or decimal points involved and will handle the situation by printing it in 'exponential' notation. (If you've used a calculator, you will no doubt have encountered the situation where it cannot handle more than a certain amount of numbers before it runs out of space.)

When your Sega needs to write a number such as 1 billion (which is 1000000000) it does so by exponential notation – represented by – 1E + 09. This means, the number 1 followed by 9 zeros.
If your Sega can't fit in all the required numbers, it will produce an overflow error.
If your Sega writes 7E – 09, we must then move the decimal point which comes after the number 7, 9 places to the left, inserting zeros as required.

What 7E – 09 means is 7 × (10 to the power of minus 9) or 0.000000007

This is not all that hard when you study it for some time, and it really is a lot easier to know what your numbers are all about without losing the right spot for the decimal place.

The and's and the or's

You know the difference between AND and OR.
So does your John Sands Sega.
We'll prove it now.
Let's pretend that there's a vacancy on the next moon flight.
To be considered for the job an applicant has to:

have been on a space
mission before
AND
possess a pilot's licence.

Now type this programme:

```
10 PRINT ''HAVE YOU ...''
20 INPUT ''BEEN ON SPACE MISSION?'' ; D$
30 INPUT ''A PILOT'S LICENCE?'' ; E$
40 IF D$ = ''YES'' AND E$ = ''YES'' THEN 70
50 PRINT ''BAD LUCK, CAN'T HELP YOU''
60 GOTO 10
70 PRINT ''THE JOB IS YOURS''
80 GOTO 10
```

Now RUN the programme.

You might answer the questions the following way –

HAVE YOU ...
BEEN ON SPACE MISSION? YES
A PILOT'S LICENCE? NO
BAD LUCK. CAN'T HELP YOU.

Let's suppose there were no applications who could fulfil both requirements. The job specifications might then be changed to:

have been on a space
mission before
OR
possess a pilot's licence.

We just changed one small word.
The AND became an OR.

Now let's change our programme to:

40 IF D$ = "YES" OR E$ = "YES" THEN 70

Now RUN the programme and find out for yourself just how much difference that one word makes.

HAVE YOU ...
BEEN ON SPACE MISSION? NO
A PILOT'S LICENCE? YES
THE JOB IS YOURS!

So. As you can see, your Sega understands the meaning of both AND and OR – which means you can use them whenever you feel like it in any of your own programmes.

Chapter 15 Summary

We've covered quite a few things in this last chapter.
The BASIC words you have encountered are:

STOP	FRE
CONT	STR$
ON/GOSUB	SGN
ON/GOTO	ABS

You've learned about your Sega's ability to distinguish between AND and OR.
And you're well up on the concept of Exponential Notation.
Aren't you?
Here's your chance to make notes!

Notes:

CHAPTER 16

DAZZLING DISPLAYS!

Now, with the help of your John Sands Sega, we're going to turn you into a programming 'artist'.
By the time you're through, you'll be able to create all sorts of shapes, sizes and patterns on your monitor screen.
The first thing to know, is that your Sega has two screens – a text screen and a graphics screen. Until now, you have always been using the text screen.

So now, RESET your Sega and type this:

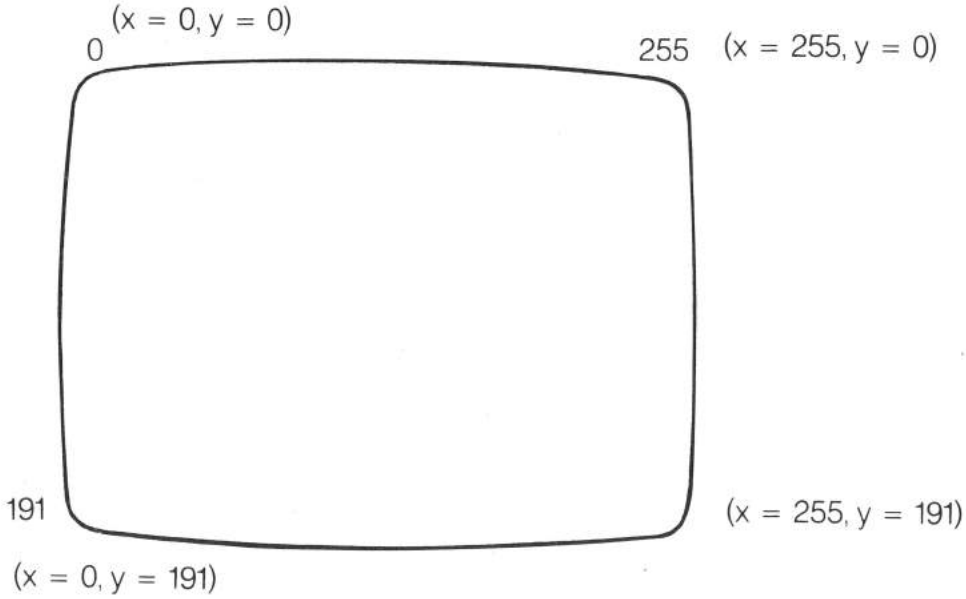
```
10 SCREEN 2, 2
20 GOTO 20
```

When you RUN this programme, you should see a white screen with a blue border around it.
Correct?
You can use the white part for graphic displays, but all you can do to the border is to change its color.
Line 10 of your programme switches the display to the graphics screen.
The first number (2) of the SCREEN command tells the Sega which screen is to be used for PRINT, CLS and so on.
The second number (2) specifies which screen is to be displayed on the TV or monitor set.
For both parameters, the number 1 means the text screen and the number 2 means the graphics screen.
Got that?
Good.

The layout of the graphics screen

The graphics screen consists of 256 individually addressable points in a horizontal direction and 192 similar points vertically.

Like this –



To address the top left hand corner, you would use 0, 0.
To address the bottom right hand corner, you would use 255, 191.
The very centre of the screen lies at 127, 95.

Your Sega is a colorful character

The color statement can also be used with the graphics screen.

To prove it, enter this:

```
10 SCREEN 2, 2
20 COLOR 1, 7, (0, 0) – (255, 191), 6.
30 GOTO 30
```

When you've been dazzled enough, use BREAK.

In conjunction with the graphic screen, the COLOR command works in the following way –

The first number specifies the PRINTING COLOR.
The second number specifies the BACKGROUND COLOR.
The numbers in brackets specify the top left hand and the bottom right hand points of the SCREEN AREA to be colored.
The last number specifies the BORDER or BACKDROP COLOR.

Now replace line 20 with the following lines and see what happens:

```
20 COLOR 5, 11, (40, 20) – (80, 80), 3
20 COLOR 15, 14, (127, 0) – (255, 191), 8
20 COLOR 8, 13, (0, 95) – (100, 191), 4
```

Satisfied with what's going on?
Good.
Now get the point!

Type:

```
10 SCREEN 2, 2
20 CLS
30 COLOR 1, 15, (0, 0) – (255, 191), 15
40 X = INT (RND (1) * 255)
50 Y = INT (RND (1) * 191)
60 C = INT (RND (1) * 13)
70 PSET (X, Y), C
80 BEEP
90 GOTO 40
```

RUN it and watch for a while. Then press BREAK when you want it all to stop!
In case you're wondering what it was, the command PSET is used to set individual points on the graphics screen. The programme you just wrote selects a random point and displays it in a random color.
If you have set a point using PSET, you can switch it back to the background color with another command – PRESET.
If you don't want any color, use this command – PRESET (x, y)

Getting from A to B

Shapes aren't the only things you can draw on the graphics screen. You can create lines too.

Like this:

```
10 SCREEN 2, 2
20 CLS
30 LINE (50, 50) – (50, 100), 8
40 GOTO 40
```

Now RUN it and BREAK when you've finished.

Here's how the line command works –

The first set of numbers in brackets set the STARTING POINT of the line.
The second set of numbers in brackets set the FINISHING POINT of the line.
The last number specified the COLOR.

Now change line 30 to this:

```
30 LINE (50, 50) – (100, 100), 8, B
```

RUN it and see what happens. Then BREAK.
By using the B at the end of the command, your Sega draws a BOX – where the first set of numbers in the bracket is its top left hand corner, and the right set of numbers is its bottom right hand corner.
Now change line 30 again.

To this:

30 LINE (50, 50) – (100, 100), 8, BF

RUN it.
The box is colored in.
The F you added to line 30 did this. It FILLed it in.
Here's another thing. You can ERASE any graphics created by the LINE command, through using another statement – BLINE.
The format is the same as for the LINE statement, but the color assignment is ignored.

Going round in circles

Now that you can draw lines, let's move on to circles!

Enter this:

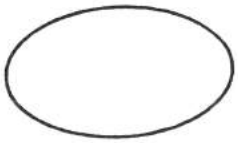
```
10 SCREEN 2, 2
20 CLS
30 CIRCLE (127, 95), 20, 4
40 GOTO 40
```

RUN this, then BREAK when you're ready.
In that programme, the set of numbers within the brackets is the CENTRE OF THE CIRCLE. The second number (20) is the RADIUS.
The third number is the COLOR.

Now change line 30 to this:

30 CIRCLE (127, 95), 20, 4, 2

RUN it and BREAK to end.
The new parameter you have added is the RATIO – which controls the 'roundness' of the circle.

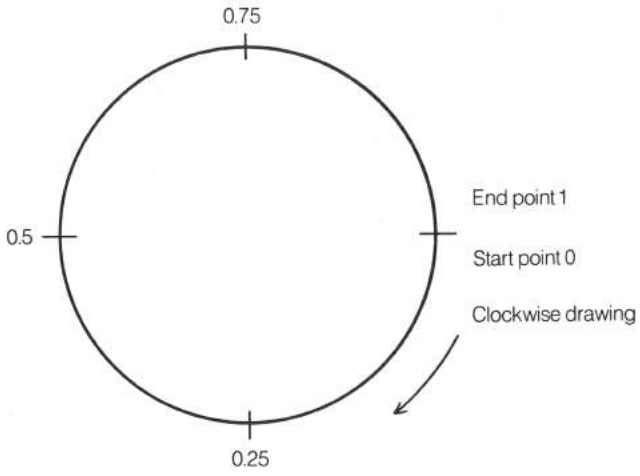
Here's an example of Ratio 0.5 

Here's an example of Ratio 2 – 

Now change line 30 to this:

30 CIRCLE (127, 95), 20, 4, 0.2, 0.8

RUN it and then BREAK.
The two new parameters you have added are the start point and the end point – as shown in the diagram.



You can also add B or BF to the end of the CIRCLE command, with similar effects as the LINE command.
Just B alone, creates an enclosed area.

So try:

30 CIRCLE (127, 95), 20, 4, 2, 0.2, 0.8, B

By using BF you would be coloring in the enclosed area.
You can use (you guessed it!) BCIRCLE to erase graphics created with the CIRCLE command. The format is the same, but like before, the color assignment is ignored.

Coloring in

There's one more graphics command your John Sands Sega knows about – and that's PAINT. Which is used to color an enclosed area.

Let's do it. Enter this:

```
10 SCREEN 2, 2
20 CLS
30 LINE (50, 50) – (100, 75), 1
40 LINE (100, 75) – (80, 120), 13
50 LINE (80, 120) – (50, 50), 8
60 PAINT (80, 80), 7
70 GOTO 70
```

RUN, then BREAK.

Here's how the PAINT command works –

The numbers in brackets specify the point to begin painting from.
The last number specifies the painting color.

Just one more thing before we finish this chapter. If you want to see the graphics screen in direct mode, simply press SHIFT-BREAK.
To revert to the text screen, do the same.

Chapter 16 Summary

Now that you've reached the last chapter in this book, you can justifiably feel extremely proud of yourself.
As we hope you have realised, your John Sands Sega is totally under your command at all times. A precisely logical friend, able to help you do literally millions of things. And a friend for life too.

In Chapter 16 we covered the following BASIC words:

SCREEN	LINE
COLOR	BLINE
PSET	CIRCLE
PRESET	BCIRCLE
	PAINT

So there we are!
You're now a fully fledged programmer, able to control your Sega as you wish. The more you use it, the more often you will probably discover other things for yourself.

Notes:

Phew!

Well, there we are!

At the end of the final chapter.

And it wasn't all that hard was it?

Once you have mastered and understood everything we have gone through, you will find yourself able to create and produce a really wide variety of programmes.

We wish you all the best with your John Sands Sega Personal Computer. As we mentioned before, you have chosen a product which is both very special and which allows you to do things which you would normally only be able to do on very expensive and highly powerful equipment.

If you have any questions you still need answered, or if there is anything else you would like to know about the capabilities of your John Sands Sega, please contact John Sands Electronics and we will provide you with all the information we can.

Appendix A

Color table

Color No.	Color
0	Transparent
1	Black
2	Green
3	Light green
4	Dark blue
5	Light blue
6	Dark red
7	Cyan
8	Red

Color No.	Color
9	Light red
10	Deep yellow
11	Light yellow
12	Dark green
13	Magenta
14	Gray
15	White

Appendix B

Frequency table

Scales:	f1	f2	f3	f4	f5	f6
C		131	262	523	1047	2094
$C^{\#}$. D^{b}		139	277	554	1109	2218
D		147	294	587	1175	2350
$D^{\#}$. E^{b}		156	311	622	1245	2490
E		165	330	659	1319	2638
F		175	349	698	1397	2794
$F^{\#}$. G^{b}		185	370	740	1480	2960
G		196	392	784	1568	3136
$G^{\#}$. A^{b}		208	415	831	1661	3322
A	110	220	440	880	1760	3520
$A^{\#}$. B^{b}	117	233	466	932	1864	
B	123	247	494	988	1976	

Appendix C

Control codes

Key operation	PRINT CHR$ (Value)	Functions
CTRL + A	PRINT CHR$ (1);	NULL No character
C	——	BREAK Stops programme run
E	5	Clears Characters after CURSOR
G	7	BELL Makes "beep" sound
H	8	DEL Deletes characters
I	9	HT Horizontal TAB
J	10	LF Line feed
K	11	HM Returns CURSOR to home position
L	12	CL Clears screen
M	13	CR Carriage return
N	14	Dieresis↔Alphanumeric shift
O	15	↑↓ Screen shift, text↔graphic

Appendix C

Control Codes (continued)

Key operation	PRINT CHR$ (Value)	Functions
P	16	Standard character size
Q	17	Character size, horizontally 2 times as large (graphic) (Screen 2)
R	18	INS (Insert)
S	19	Key input (A ~ Z) no shift, capital letter
T	20	Key input (a ~ z) no shift, small letter
U	21	Clears lines and returns CURSOR to left head
V	22	Normal mode
W	23	GRAPH key input graphic mode<=>alphabetic character shift
X	24	Click sound ON<=>OFF shift
-	28	➡ CURSOR movement
-	29	⬅ CURSOR movement
-	30	⬆ CURSOR movement
-	31	⬇ CURSOR movement

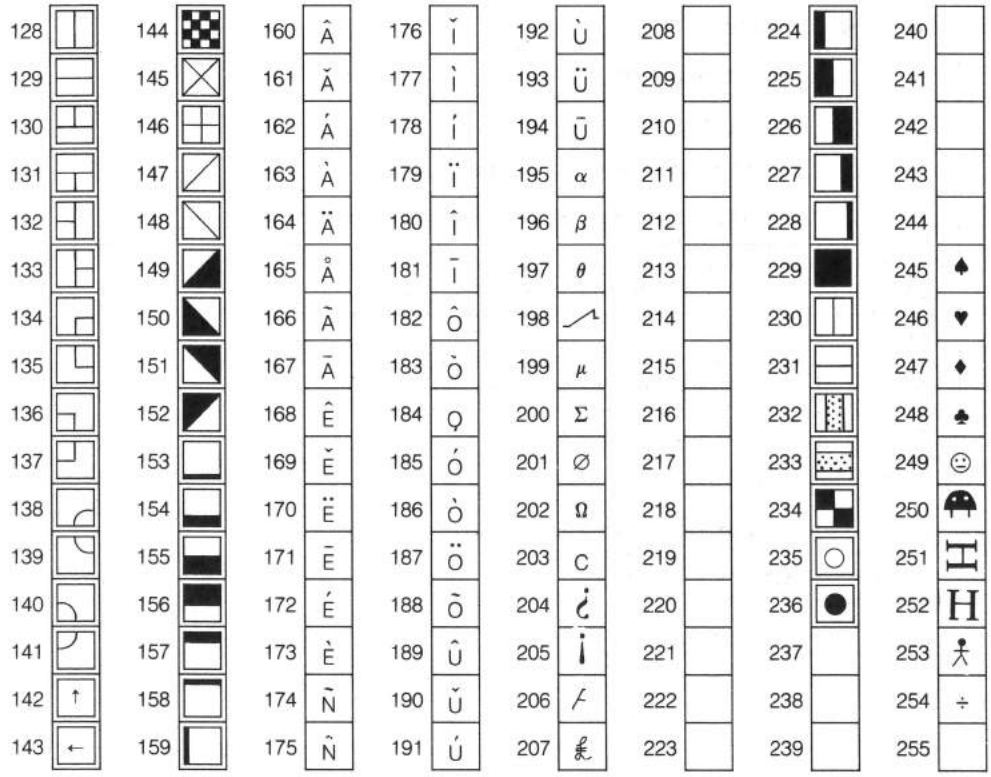
Appendix D

Character codes

32	SP	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	▼	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	¥	108	l	124	:
45	–	61	=	77	M	93	]	109	m	125	}
46	•	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	π	111	o	127	

Appendix D

Character codes (Continued)



Appendix E

Commands

No.	Command	Functions
1	LIST	Displays programmes on screen.
2	LLIST	Prints programmes on PRINTER.
3	SAVE	Records programmes on cassette tapes.
4	VERIFY	Compares programmes in MEMORY and those recorded on cassette tapes.
5	LOAD	Loads cassette tape programmes on MEMORY.
6	RUN	Runs programmes.
7	CONT	Continues discontinued programmes.
8	NEW	Clears variables and programmes.
9	DELETE	Clears programmes partially.
10	AUTO	Generates line numbers automatically.
11	RENUM	Renumbers line numbers.

Appendix F

Statements

No.	Statement	Functions
1	REM	Comment.
2	STOP	Stops programmes. Continuable by CONT.
3	END	Completes programmes run.
4	LET	Input substitution. LET omittable.
5	PRINT or ?	Displays on display.
6	LPRINT or L?	Prints on Printer.
7	INPUT	Input from key.
8	READ	Reads data from 'DATA' statements.
9	DATA	Shows data to be read from 'READ' statements.
10	RESTORE	Assigns positions of 'DATA' statement to be read from 'READ' statements.
11	DIM	Declares arrays.
12	ERASE	Clears declared array.
13	DEF FN	Defines user function.
14	GOTO	Branches to assigned address.
15	GOSUB	Go to subroutine.
16	RETURN	Returns from subroutine.
17	ON-GOTO	Selects line numbers to be branched.
	ON-GOSUB	Selects line numbers to be branched.
18	FOR-TO-STEP-	Repeats statements between FOR and NEXT for a set number of times. STEP omissible.
19	NEXT	Assigns positions of repeating by 'FOR' statement.
20	IF-THEN	Conditional branch.
21	CONSOLE	Assigns the ranges of screen scroll, click sound and character set.
22	CLS	Clears screen.

Appendix F

Statements (Continued)

No.	Statement	Functions
23	SCREEN	Shifts the screen.
24	COLOR	Color assignment.
25	PATTERN	Changes character sprite PATTERN.
26	CURSOR	Assigns display positions.
27	POSITION	Assigns coordinates.
28	PSET	Displays dots.
29	PRESET	Erases by dots.
30	LINE	Draws lines.
31	BLINE	Erases by lines.
32	CIRCLE	Draws circles.
33	BCIRCLE	Erases by circles.
34	PAINT	Paints enclosed extent.
35	SPRITE	Assigns sprite position, color and pattern.
36	MAG	Assigns sprite magnitude.
37	SOUND	Produces effective sound.
38	BEEP	Produces beep sound.
39	HCOPY	Prints text on screen on to printer.
40	CALL	Branches to machine language subroutine.
41	POKE	Writes in memory.
42	OUT	Outputs to output port.
43	VPOKE	Writes data in video RAM.

Appendix G

Functions

No.	Statement	Functions
1	ABS(*x*)	Finds the absolute value of *x*.
2	RND(*x*)	Generates random numbers.
3	SIN(*x*)	Finds the sine of *x*.
4	COS(*x*)	Finds the cosine of *x*.
5	TAN(*x*)	Finds the tangent of *x*.
6	ASN(*x*)	Finds the arc sine of *x*.
7	ACS(*x*)	Finds the arc cosine of *x*.
8	ATN(*x*)	Finds the arc tangent of *x*.
9	LOG(*x*)	Finds the natural logarithm of *x*.
10	LGT(*x*)	Finds the common logarithm of *x*.
11	LTW(*x*)	Finds the logarithm of *x* with 2 as a base.
12	EXP(*x*)	Finds *e*.
13	RAD(*x*)	Converts degrees into radians.
14	DEG(*x*)	Converts radians into degrees.
15	PI	Specifies the ratio of the circumference of a circle to its diameter.
16	SQR(*x*)	Finds the square root of *x*.
17	INT(*x*)	Finds the greatest integer not exceeding *x*.
18	SGN(*x*)	Specifies the positive and negative codes of *x*.
19	ASC(*s*)	Specifies the first code of character-string *s* by numeric values.
20	LEN(*s*)	Specifies the number of character-string *s*.
21	VAL(*s*)	Converts character-string *s* into numeric values.
22	CHR$(*x*)	Specifies the corresponding character and functions of *x*.
23	HEX$(*x*)	Specifies the hexadecimal number character-string.

Appendix G

Functions (Continued)

No.	Statement	Functions
24	INKEY$(*x*)	Checks whether or not key was pressed. When key is pressed, the character is given. (Null) if it is not pressed.
25	LEFT$(*s*,*x*)	Substitutes the character-string covering from the left of the character-string *s* to *x* places.
26	RIGHT$(*s*,*x*)	Substitutes the character-string covering from the right of the character-string *s* to *x* places.
27	MID$(*s*,*x*,*y*)	Substitutes the character-string of length *y* from the *x* places of the left of the character-string. *y* is omissible and in this case, substitutes from *x* place character to the end character.
28	STR$(*x*)	Converts *x* into the character-string which indicates *x*.
29	TIME$	Determines the time of the inside clock.
30	PEEK(*x*)	Specifies the content of the *x* address of memory.
31	INP(*x*)	Specifies the input content of input port.
32	FRE	Specifies memory area space for users.
33	SPC(*x*)	Used by print statement. Provides space.
34	TAB(*x*)	Used by print statement. Assigns display positions.
35	STICK(*n*)	Shows the *n* direction of joysticks.
36	STRIG(*n*)	Shows the trigger button condition of joystick *n*.
37	VPEEK(*x*)	Specifies the content of VRAM *x* address.

Appendix H

Limitations

Contents	Limitation
Characters taken into the inside from the screen.	256 characters
Character numbers usable for actual text image by reserved words converted from line buffer.	256 characters
Character numbers which can be handled as character-string.	255 characters
Level number such as operator priority, etc.	32 levels
Area for string operation.	300 characters
FOR – NEXT nesting level number.	16 levels
GOSUB, RETURN nesting level number.	8 levels

Appendix I

Error messages

Message	Description
System	System error due to Basic Interpreter Programme. Generally this occurrence is impossible.
N – formula too Complex	Numeric values are too complicated.
S – formula too Complex	Character-string is too complicated.
Overflow	Values and operation results exceed permissible range.
Division by Zero	The denominator in division is 0.
Function Parameter	Function parameter is unusual.
String too long	The length of character-string exceeds 255.
Stack overflow	Excessive use of parentheses ( ). Patterns to PAINT are too complicated. User define function calls itself.
Out of memory	Memory is insufficient. Text. Variable. Array.
Number of Subscripts	Number of subscripts is unusual.
Value of Subscript	Value of subscript is improper.
Syntax	Syntax Error.
Command Parameter	Command Parameter is unusual.
Line number over	In AUTO or RENUM, line No. exceeds 65535.
Illegal line number	Line No. is improper.
Line image too long	Line image is too long, (RENUM, etc.).
Undefined line number	Line No. is undefined. (RENUM, GOTO, GOSUB, IF – THEN, RESTORE, RUN.)
Type mismatch	The type of substituting side and that of substituted side do not match. (Values, strings.)
Out of DATA	Reading by READ Statement was attempted but DATA of DATA statement is unavailable.
RETURN without GOSUB	RETURN statement was executed without GOSUB.
GOSUB nesting	GOSUB nesting exceeded 4 levels.
NEXT without FOR	For statement corresponding to NEXT is not available.

Appendix I

Error messages (Continued)

Message	Description
FOR nesting	FOR ~ NEXT nesting exceeded 4 levels.
Statement Parameter	Statement parameter is unusual.
Can't continue	Can't continue by CONT statement.
FOR variable name	FOR statement loop variable is not numeric variable. (Character-string or array.)
Array name	DIM statement parameter is not in array.
Redimensioned array	DUAL array definition was attempted.
Undefined array	Erase of undefined array was attempted.
No Programme	SAVE was attempted despite unavailability of programme in text.
Memory writing	Memory writing error (at the time of LOAD).
Device not ready	Printer is not connected or in trouble.
Undefined Function	Undefined user function was called.
Verifying	Errors in comparison with tape programmes.
Illegal direct	Direct statement run is impossible.
Redo from start	INPUT DATA of input statement is unusual. Redo input from the start.
Extra ignored	INPUT DATA of input statement is unusual. Extra data was entered. Extra data was ignored.
Unprintable	Errors other than the above.

INDEX