BEGINNER'S STEP-BY-STEP THEC64 COMPUTING CODING COURSE

Rich Stals

ISBN: 979-869666737-9

First printing edition 2020.

Rich Stals
www.stals.com

CONTENTS

INTRODUCTION

You have gone and bought yourself your THEC64 Maxi and played a bunch of games while reliving the glory days of 8-bit home computing in the 80s. If you are now asking yourself, "What's next?" This book is for you.

Today I work in a global Educational Technology company. My background in Computer Science means that I have been able to create computer programs for a wide range of industries. From web-based information tracking systems for mining companies to immersive first-person educational simulations for Paramedics and Nurses, being able to code has opened a wide world of creative expression for me.

I started programming when I was 10 years old. My parents bought me a brand-new Commodore 64 for my birthday. I spent hours playing Boulder Dash, Pitstop II and Ace of Aces. However, it was when I found a copy of a step-by-step programming guide in my local library that my love of the Commodore 64 was cemented. I was no longer limited to interacting with my computer in the way that someone else had decided. I was now able to make my computer do what I wanted. It now displayed the text and images I constructed. It played the sounds and music I created. Suddenly, a whole new world had opened up before me, and I was its creator.

This step-by-step coding course for THEC64 is based on the way that I first learned to code my Commodore 64. You will learn to code using BASIC (Beginner's All-purpose Symbolic Instruction Code), growing your skills and knowledge until you are able to create a fully-fledged program complete with user input, animated graphics, music and more.

This coding course is written especially for THEC64 Maxi. However, it will work for the original Commodore 64 too, if you have one.

This course is full of straightforward information given in easy to digest bite-size pieces. Each part builds on the ones before it. There is computer jargon, but it is jargon you will understand as you make your way through it. Is learning to code THEC64 essential to enjoying it? No. Will it help you understand and engage with it more? I hope so. Could this lead to a new and amazing career direction? Definitely, if that's what you want.

THE KEYBOARD

RUN/STOP Stops a program that has been running and shows where it stopped. Use it with the SHIFT key to load a program from a cassette file.

CLR/HOME Will move the cursor to the top left corner of the screen. Used with the SHIFT key, it will clear the screen.

RESTORE When used with the RUN/STOP key it will reset the computer back to the READY prompt.

CTRL Use it with the NUMBER keys 1-8 to access the eight colours marked on the keys. Used with keys 9 and 0 to produce reversed characters.

INST/DEL Used for editing your program. On its own, it backspaces and deletes a character. Used with the SHIFT key, it inserts a space.

SHIFT & SHIFT LOCK These keys allow you to access the top characters on double-character keys. They access the graphic characters on the right side of the key.

FUNCTION These four keys can be used with the SHIFT key to access the 8 function key combinations.

CURSOR These two keys, used with the SHIFT key, move the cursor around the screen.

THEC64 Use it with the SHIFT key to switch between uppercase and lowercase display modes. Use it with the NUMBER keys 1 to 8 to access the second set of colours. Use it to access the graphic characters on the left side of the key.

RETURN This tells the computer to process what you have just typed in. It also moves the cursor to the next line down.

THEC64 keyboard is similar to any modern computer keyboard, but it has some keys and functions specific to running in BASIC mode. Let's take a closer look at these keys and functions.

RETURN

The RETURN key acts a little like the return or enter key on your regular computer keyboard by moving the cursor to a new line.

When entering a program, pressing the RETURN key tells the computer to look at what you have just typed in, check it and then enter it into the computer's memory.

SHIFT AND SHIFT LOCK

The SHIFT key works similar to that of a regular computer keyboard. It lets you type onto the screen the top characters on the double-character keys.

When you want to type in the graphics characters on the front of the keys, press and hold the SHIFT key and then tap the key to use the graphic character on the **right** side of the key.

For example **SHIFT + W** will produce .

Press the SHIFT LOCK key to 'lock' the shift function on the keyboard. An arrow appears in the top-right corner of the screen to let you know that SHIFT LOCK is ON.

With this on, any key you press will act as if you are also holding down the SHIFT key. Press it again to turn SHIFT LOCK to OFF.

CRSR (CURSOR KEYS)

The two cursor control keys let you move the cursor around the screen. Use the ⇑ CRSR ⇓ key to move the cursor down, use it with the SHIFT key to move the cursor up. Use the ⇐CRSR⇒ key to move the cursor right, use it with the SHIFT key to move the cursor left.

You don't need to repeatedly tap the cursor key. Instead, you can hold it down to move the cursor to where you want it to go.

INST/DEL

The INST/DEL key is used for editing your program and other text on the screen. When you press the INST/DEL key by itself, it backspaces and deletes one character to the left of the cursor. When you press it with the SHIFT key it will insert a blank space to the right of the cursor.

The Insert functionality of the key is important when editing your programs. This is because whatever you type will overwrite what is already on the screen.

For example, I have a command that will PRINT out the word WORLD on screen. I want to actually PRINT out the words HELLO WORLD. So, I use the CRSR keys to move my cursor over the letter W.

```
    PRINT "WORLD"
```

I then press SHIFT+INST/DEL six times to insert enough spaces for my missing word.

```
    PRINT "     WORLD"
```

Now I can type in the missing text.

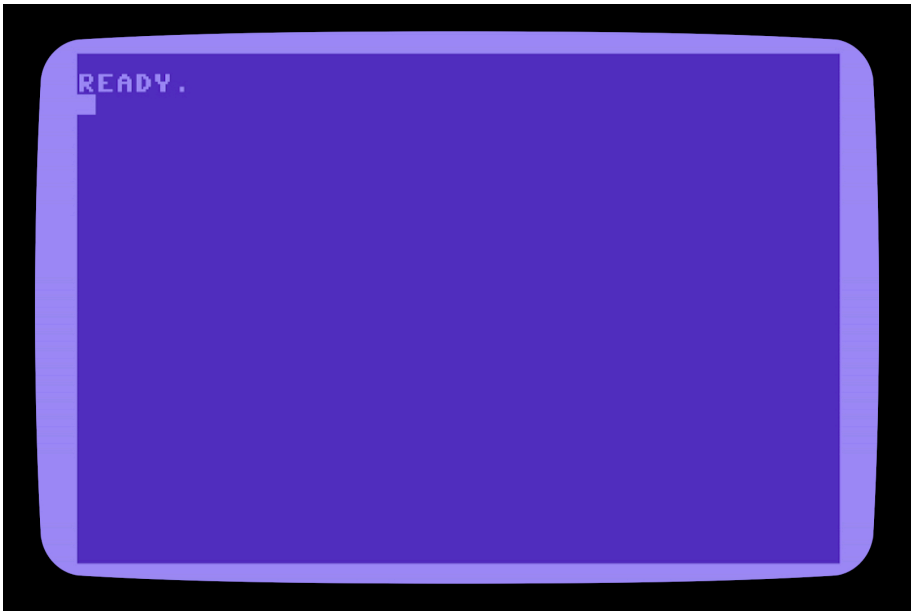
```
    PRINT "HELLO WORLD"
```

CLR/HOME

Pressed on its own, the CLR/HOME key will move the cursor back to its home position, the top-right corner of the screen. If you hold the SHIFT key and press the CLR/HOME key, it will clear the screen and move the cursor to its home position.

RESTORE

Hold down the RUN/STOP key and press the RESTORE key to restore your computer to its normal state. This will clear the screen, return it to its original colour and reset the video and sound chips.



CTRL (CONTROL)

The Control key lets you perform special control functions on the computer. Use it with the number keys 1 to 8 to change the text colour to the colour written on that key. Press the CTRL key and the numbers 9 and 0 to produce reversed characters.

THEC64

Use this key with the SHIFT key to switch between the uppercase and lowercase display modes.

Use it with the numbers 1 to 8 to access the additional 8 colours.

When you want to type in the graphics characters on the front of the keys, press and hold the THEC64 key and then tap the key to use the graphic character on the **left** side of the key.

For example **THEC64 + W** will produce ▌.

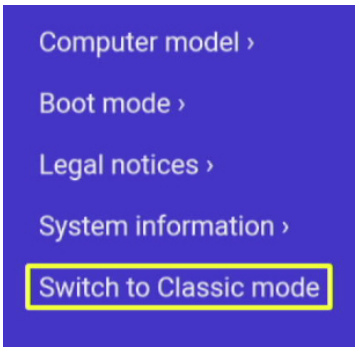
LESSON 1
GETTING STARTED

Connect your THEC64 and turn it on. By default, it will boot up into the Carousel mode, giving you a Graphical User Interface to a collection of pre-installed games. To begin your programming journey, you will need to access the Classic mode.

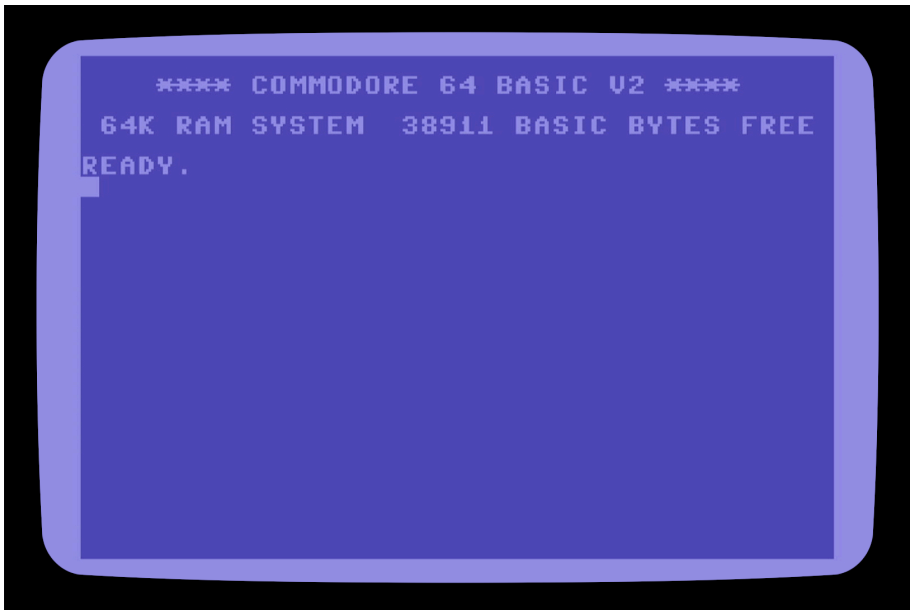
CLASSIC MODE

To access the Classic mode, use your joystick to open the ***Device Settings*** menu – click on the spanner icon.

Then select Switch to Classic Mode.



You will now be in THEC64's Classic Commodore 64 BASIC mode.



SET THE DEFAULT BOOT MODE

You can also set your THEC64 to automatically boot into Classic mode by default. To do this, access the ***Device Settings*** menu. Click the menu button on your joystick.

The menu appears at the bottom of the screen.

Select the **Options** item.

Select Device Settings > Boot Mode. Then select the Classic option.

Click the menu button on the joystick to return to the Commodore 64 BASIC mode. Every time you start your THEC64, you will automatically boot into this mode.

BLANK VIRTUAL DISK

When you launch your THEC64 it automatically searches for an attached USB stick. If it finds one, it then searches for a compatible virtual disk file named THEC64-drive8.d64exists in the root directory of your USB stick. This file acts as if it is a floppy disk drive attached to your computer. You will need this up and running in order to save and load the programs you create during this course.

STEP 1: PREPARE YOUR USB

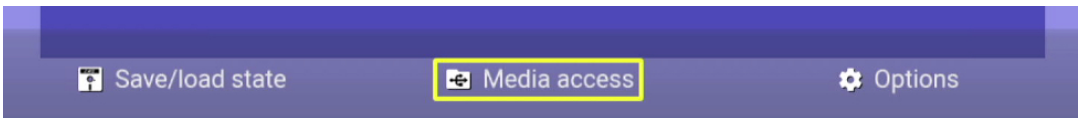
You will need a USB drive that you are happy to completely erase of any existing files. Choose a USB stick that is 32GB or less in size. You will need to use an existing computer to format the USB drive using the FAT32 (or FAT) format with Master Boot Record (MBR). The steps for doing this vary depending on the type and version of your computer's operating system. I suggest you follow the steps outlined in Appendix B of THEC64 User Manual, which can be downloaded for free at:

https://retrogames.biz/thec64/support/manuals-thec64

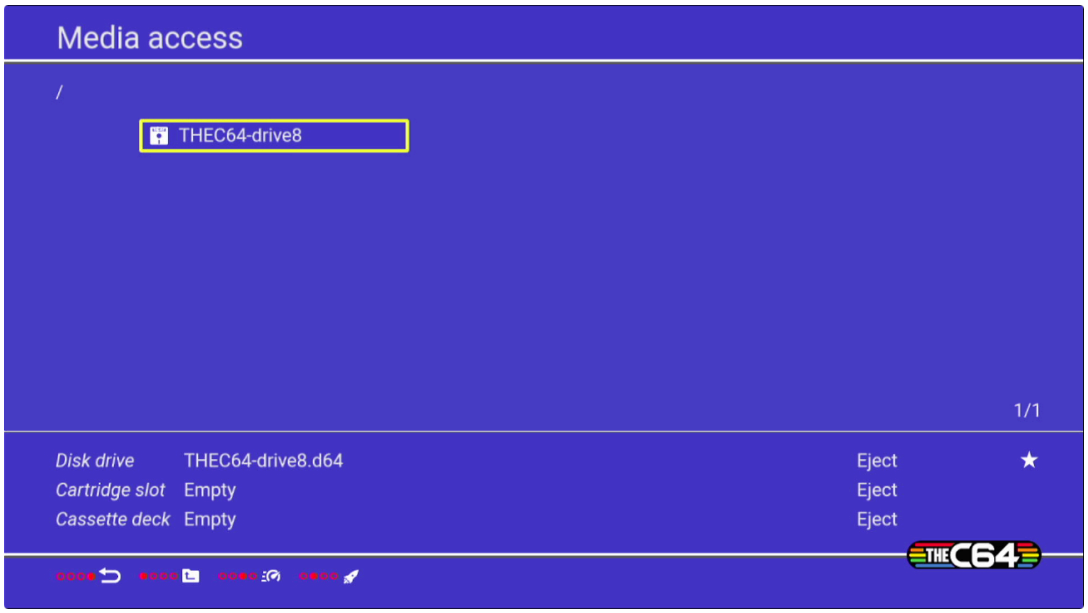
STEP 2: INITIALISE THE USB

Make sure your THEC64 is turned off. Insert your formatted USB stick and turn it on.

Check your USB now has a blank virtual disk. Click the **Menu** button on your joystick and select the **Media access** option.



You will see that your blank virtual disk is listed. Notice that the virtual disk is also listed as being ‘inserted’ into the computers virtual disk drive.



Exit the menu. You are ready to begin your Commodore 64 coding journey on your THEC64.

LESSON 2
YOUR FIRST PROGRAM

We are going to write your very first Commodore 64 BASIC program, save it to your virtual disk and then list the contents of your virtual disk to see if everything worked as expected.

ENTER YOUR FIRST PROGRAM

Make sure you are in Classic mode.

Type in the following BASIC computer code exactly as it is written below. Be sure to press the RETURN key after every line.

```
10 PRINT "HELLO WORLD"
20 GOTO 10
```

Every time you type a line of BASIC code and hit the RETURN key; the computer places it into the short-term memory of the computer.

Let's examine what each part of this program does.

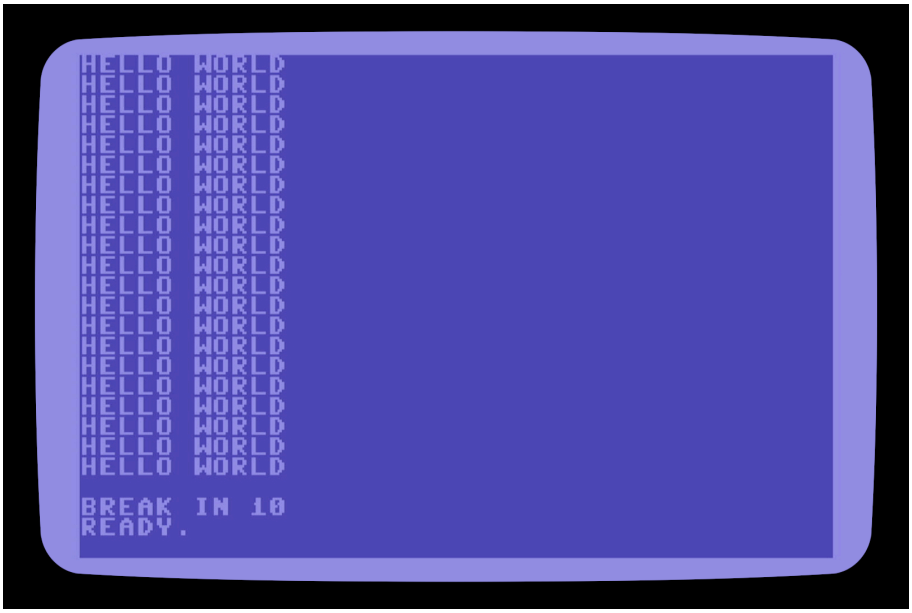
Firstly, you will notice that each line begins with a line number. This tells the computer in what order to run each line of code. Traditionally, we increment the line numbers by 10. This means that if we find we need to insert a few extra lines of code, we have another 9 number to use to add these extra commands.

The program is very simple. The first line tells the computer to PRINT the words HELLO WORLD to the screen. The second line tells the computer to go to line 10. The computer will loop back to the first command to PRINT the text to screen.

Now you are ready to run your first program. To run a program that is loaded in the computer's memory we use the BASIC command **RUN**. Type the word RUN and hit RETURN.

You will see a column of text run down the screen as your program repeatedly PRINTs the words HELLO WORLD on the screen. We wrote our program to cause what is known as an ***infinite loop***. The computer will simply keep on looping through our code with no way to know how to stop.

Press the RUN/STOP key to force your program to stop.



Notice how the computer tells you which line you in your code that the computer cancelled the programming running.

LIST YOUR PROGRAM

The program is sitting in the computer's short-term memory. You can list the code of any program that is in memory by using the LIST command. Type in LIST and press the RETURN key.

Your program will display on the screen.

EDIT YOUR PROGRAM

Let's try a couple of ways that we can edit a program listing. Firstly, let's add a new line of code between lines 10 and 20. Type in a new line 15 and tell the computer to PRINT some text of your choice. E.g.

```
15 PRINT "HELLO PERTH!!!"
```

Now, type the LIST command and press the RETURN key to list the edited computer program. [Note, from now on I will not remind you to press the RETURN key when you want to run a command.]

Notice that even though we typed line 15 in last, when we list the full program the computer lists it in the order it will execute the code ('execute' is just the way computer programmers refer to it when a computer runs a program).

Next, we will edit an existing line of code to change it. I want to change line 10 to say hello to me. Theoretically we could just type in a new line of code and if we give it the line number 10, it will replace what was already in line number 10.

Instead, we can use the LIST command to PRINT out our program on the screen and then use our cursor keys to move our cursor to the line and change it there.

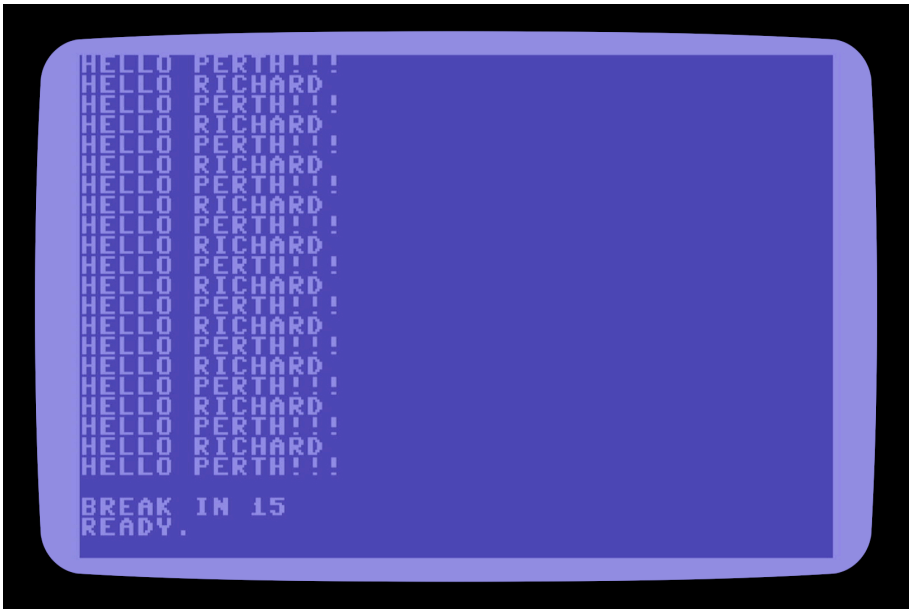
Let's do it. Use your cursor keys to change line number 10 to say HELLO [YOUR NAME]. E.g.

```
10 PRINT "HELLO RICHARD"
```

When you press the RETURN key after editing the line of code, it will replace the code currently in memory for line 10 with the edited version you just typed in.

```
10 PRINT "HELLO RICHARD"
15 PRINT "HELLO PERTH!!!"
20 GOTO 10
READY.
```

Use the cursor keys to move back down the page past the READY prompt and use the RUN command to execute your newly edited program.



SAVE YOUR PROGRAM TO DISK

Your program currently sits in the computer's short-term memory, known as Random Access Memory or RAM. If you turn off your computer, you will lose everything in the RAM. To keep a copy of your program long term, you will need to save it to the Virtual Disk.

To save your computer program to disk we use the SAVE command in the following format: SAVE "[FILE-NAME]",8. This will save your program as the filename you type between the quotation marks. The ,8 part of the command tells the computer to save it to the disk drive. Type in a SAVE command to store your program.

```
SAVE "HELLO-WORLD",8
```

LOAD YOUR PROGRAM FROM DISK

Turn your computer off (hold down the power button for 2 seconds) and then turn it back on again to wipe out program from RAM.

You can check that your program is no longer in memory by typing the LIST command once you restart your computer.

If you know the file name of the program you want to load, you can load it straight away. However, let's look at how you can view the list of files on your virtual disk. We will use the LOAD command to load something into the computer's RAM. We use the $ symbol to specify we want to load the contents of the disk directory into memory.

Type in the following commands:

```
LOAD "$",8
LIST
```

You will see all the files that have been saved on this disk.

To load your program, type in the load command using the file name you used to save it with.

```
LOAD "HELLO-WORLD",8
```

You can now LIST and RUN your code again.

The LOAD command replaces whatever is currently within the computer's RAM. This means that if you now re-run the directory load command, it will replace your code with the directory listing. Try it.

```
LOAD "$",8
RUN
```

The computer will display a Syntax Error. This means that whatever is currently in memory cannot be run as a BASIC language computer program. Go ahead and re-load your program and check that it does indeed now run correctly.

LESSON 3
BUILDING BLOCKS

In this chapter we will look at the basic building blocks of coding the Commodore 64 in BASIC. Don't be tempted to skip this section. It lays a solid foundation for the rest of the course.

DISPLAY NUMBERS AND TEXT

Remember how we used the SHIFT + CLR/HOME key combination to clear the screen and send the cursor to the home position? You can do the exact same thing by PRINTing the matching character code to the screen. (Don't worry about character codes yet, just try out the following exercise).

The character code for to clear the screen is 157. We can use the PRINT CHR$(147) command to PRINT this character code to the screen. However, computers need you to be exactly correct when you type in commands and if you get it even just slightly wrong, it will display some kind of error message.

For example, if you type in PRINT CHRS(147) or PRNT CHR$(147) the computer will display an error message. Try it and see what happens.

DISPLAY A NUMBER

Clear the screen and type the following command to display a number.

```
PRINT 7
```

Try this with other numbers.

DISPLAY SOME TEXT

When you want to display text, called a ***String*** in programming languages, you use the PRINT command and type the text you want to appear inside quotation marks. For example:

```
PRINT "COMMODORE 64"
```

Try it with other strings. You can use any characters on the keyboard to PRINT.

USING VARIABLES

So far, we have displayed numbers and strings using static values. Let's have a look at what is called ***Variables***.

Clear the screen and type in the following command.

```
PRINT AGE
```

The result might surprise you. Instead of displaying the word AGE, the computer displayed the number 0. Now type in this command:

```
PRINT "AGE"
```

This time the computer responded as you might have originally expected.

When you use a word without the surrounding quotation marks, the computer thinks you are referring to a variable called AGE. A variable is a label for a particular slot in its memory. When you typed in the command without quotation marks, the computer searched its memory for a slot labelled AGE and when it couldn't find it, it created it and assigned an initial value of 0.

We use the command LET to assign a value to a variable. Let's assign a value to our variable AGE.

```
LET AGE=43
```

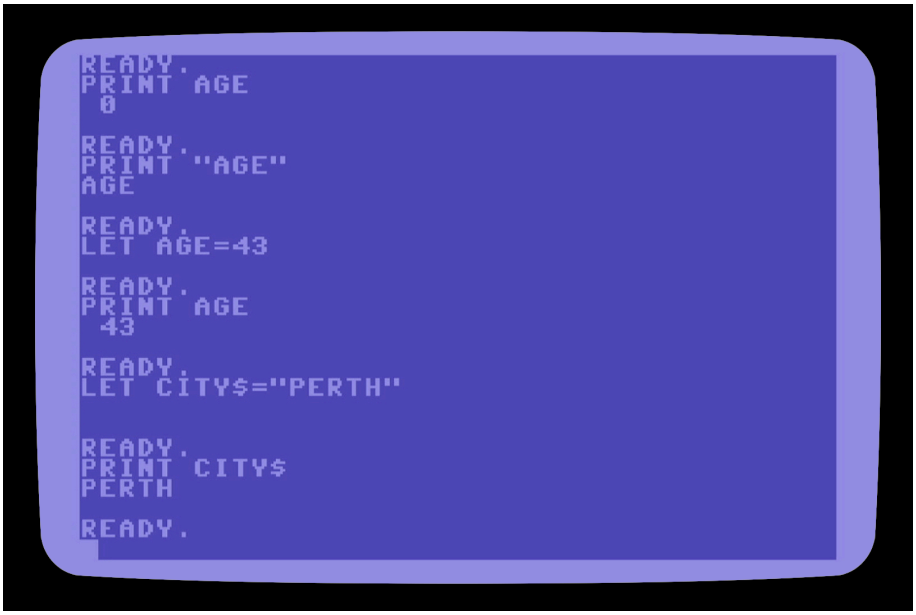
Now type the command to display the variable AGE to screen. When you use a word without quotation marks it always refers to a variable that points to a numeric value.

What if we want to have a variable that points to a string value? In that case we use a word that ends with the $ symbol. For example, we can use the variable CITY$ to hold the string value of a city.

Assign a string value to the variable CITY$.

```
LET CITY$="PERTH"
```

Now type the command to display the variable CITY to screen.



You can change the value of a variable as many times as you like. Let's change the value of CITY$.

```
LET CITY$="LOS ANGELES"
```

Experiment with assigning and displaying some numeric and string variables.

COLOURS

The Commodore 64 can display 16 colours, all of which can be accessed from the keyboard. In this section you will learn some ways of displaying colour on the screen.

By default, the Commodore 64's display is blue with a light blue border. Let's go ahead and change the display to black instead, type in:

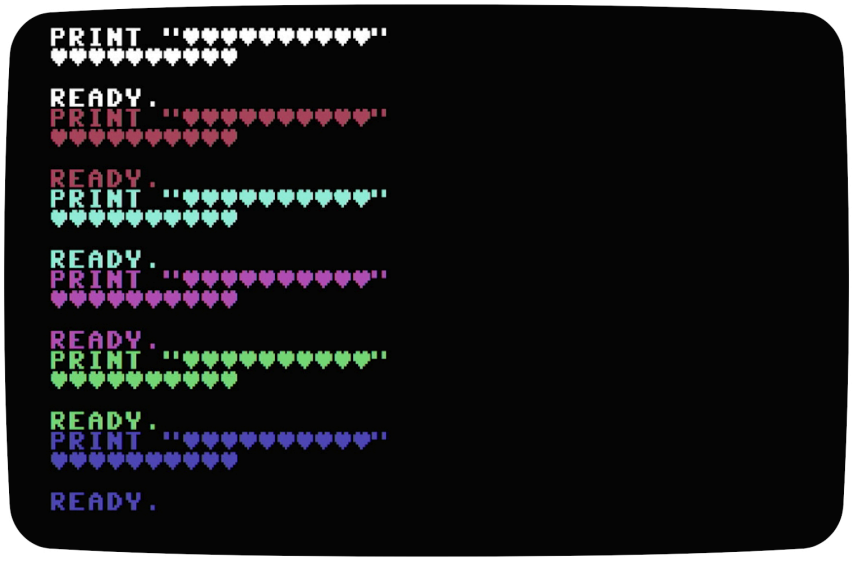
```
POKE 53280,0:POKE 53281,0
```

(You will find out how these commands work in a later lesson.)

CHANGE COLOURS USING THE KEYBOARD

If you look at the Keyboard you will see 8 colours PRINTed on the front of the number keys 1 to 8. You can change the current text colour by holding the CTRL key and tapping one of these number keys. Another 8 colours can be accessed by holding the THEC64 key and tapping one of the number keys.

Go ahead and experiment with changing text colours using the keyboard.



(You can access the heart character using SHIFT + S.)

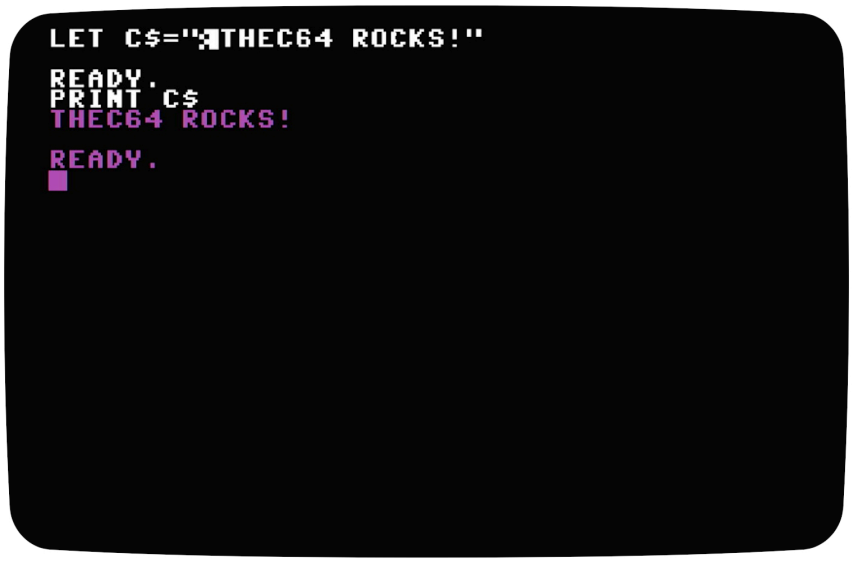
CHANGE COLOURS USING CONTROL SYMBOLS

Instead of using the keyboard, you can use control symbols in a string variable to change the colours. This is useful because the colour doesn't change until the actual string is instructed to PRINT, which can be useful when writing your own programs.

Typing in colour control symbols is easy, you do it exactly the same way that you used to change the colours using the keyboard. However, because we are declaring a string variable using the LET command, a graphic symbol is inserted into the string.

Let's declare a string called C$ and make it PRINT out a purple string.

```
LET C$="[CTRL+5]THEC64 ROCKS!"
```



Although using colour control symbols is easy to type in using the keyboard, it is much harder to do so if we are trying to copy in the code from a computer program listing. Instead, we can use a third way.

CHANGE COLOURS USING CHARACTER CODES

The Commodore 64 uses a set of codes called American Standard Code for Information Interchange or ASCII to represent all of the characters that can be PRINTed to the screen. For example, the capital letter A is represented by the ASCII code 65, the capital letter B by 66 and so on. There are ASCII codes also for operations such as moving the cursor, creating a new line and changing the text colour.

(The full set of ASCII codes used in the Commodore 64 are included in the Appendices at the back of this book.)

In Commodore BASIC you use the CHR$ command to produce the character or operation of an ASCII code.

ASCII Colour Codes

Colour	ASCII Code	Colour	ASCII Code
Black	144	Orange	129
White	5	Brown	149
Red	28	Light Red	150
Cyan	159	Dark Gray	151
Purple	156	Medium Gray	152
Green	30	Light Green	153
Blue	31	Light Blue	154
Yellow	158	Light Gray	155
Reverse On	18	Reverse Off	146

You can also use the CHR$ command and ASCII codes when defining string variables. To do this you can use what is called 'String Concatenation', that is, joining more than one string or ASCII code command together into a single string variable.

For example, we want to declare a string variable A$, set the colour to White, display some text, change the colour to Green and then display some more text.

```
LET A$ = CHR$(5)+"ONE " +CHR$(30)+"TWO"
```

Then PRINT the A$ variable to display the text and colour codes.

CALCULATIONS

You can use BASIC code to perform a wide variety of mathematical calculations.

BASIC CALCULATIONS

The four basic mathematics calculations have matching symbols in much the same way that symbols are used on a calculator.

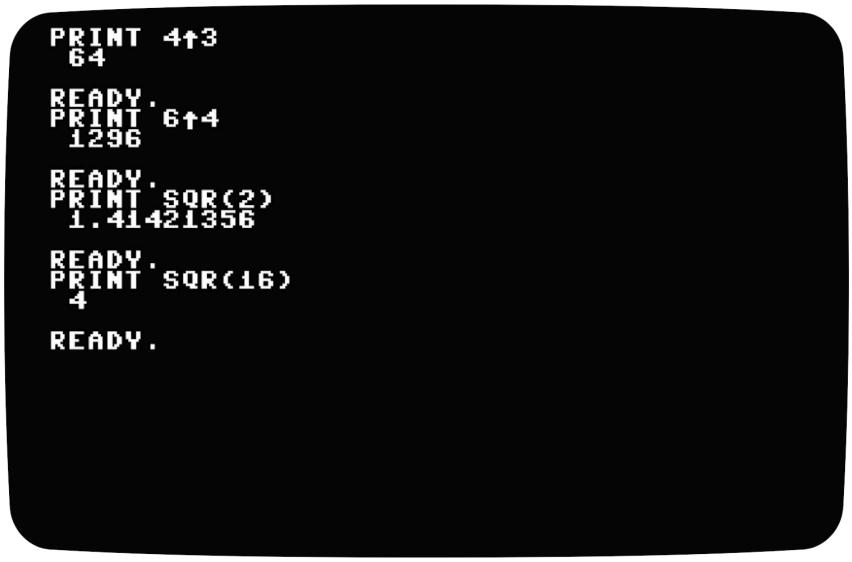
```
+   Add
-   Subtract
*   Multiply
/   Divide
```

Use the PRINT command to perform some basic maths calculations and display the result on the screen.

ADVANCED CALCULATIONS

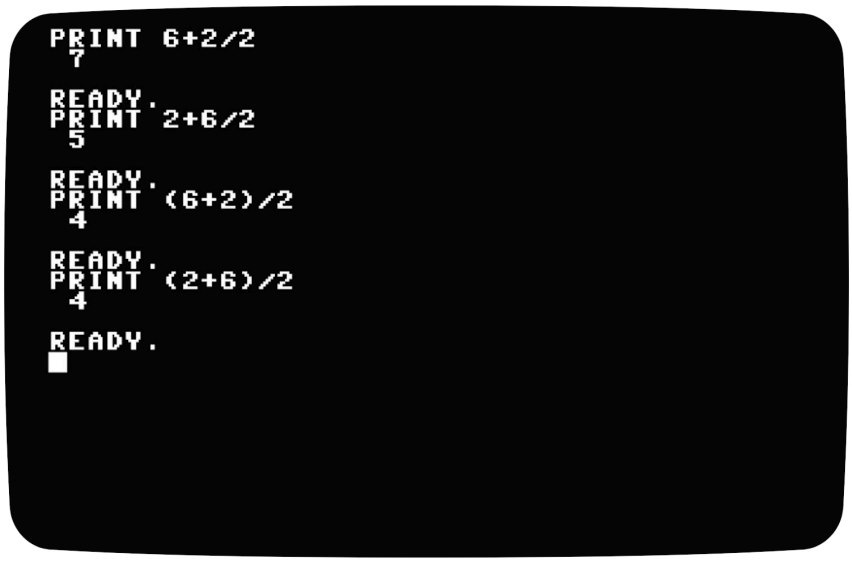
You can use in-built mathematical functions to calculate the exponents and square roots of numbers. Exponential numbers are a number multiplied by itself a specified number of times. For example, 2^3 is the same as doing 2x2x2 which equals 8. To do an exponential calculation on the Commodore 64, you use the up-arrow character ↑. So, 2^3 is entered as 2↑3.

There isn't a character used to represent the square root calculation, instead we use the SQR command, for example SQR(2) to calculate the square root of 2.



SEQUENCING CALCULATIONS

You can chain together a number of calculation steps into a single calculation. Say you want to add to numbers together and divide the result by 2. You might think the order you type the numbers in should not make a difference because 6+2 is the same as 2+6. However, you will see something you might not expect when you try this on your computer.



Your computer will not necessarily process your calculation in the order you type it in. Computers use what is called the BIMDAS order of mathematical operations. BIMDAS represents the following order that is used:

1. **B**rackets
2. **I**ndices
3. **M**ultiplication and **D**ivision
4. **A**ddition and **S**ubtraction

In our example above, we wanted the computer to add the first two numbers together first before dividing that result by the third number. However, by following the rules of BIMDAS the computer performs the division first and then performs the addition second. You can see that neither of the first two calculations produced the result we were looking for. To change the order, we needed to use brackets to define which part of the calculation to perform first.

LIMITS FOR NUMBERS

The Commodore 64 has two kinds of limits when it comes to its numbers, size and accuracy.

In terms of size, numbers that have a decimal point you can have any number between $1x10^{38}$ (1 followed by 38 zeros) to $-1x10^{38}$ (-1 followed by 38 zeros). Whole numbers (also called integers) can be within the range of 32,767 to -32,768.

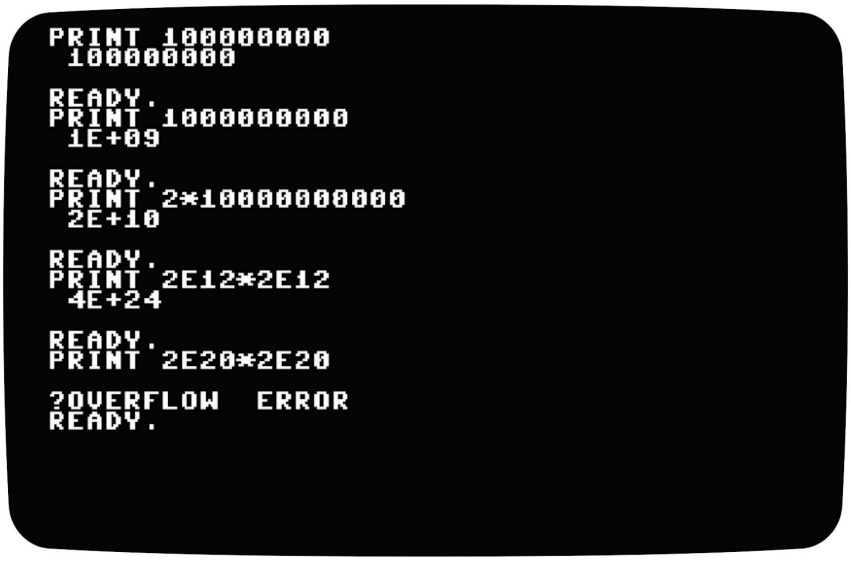
In terms of accuracy, although you can store a number with a decimal point in a very large range (size), the computer only stores the first nine digits, the rest it sets to zero. For whole numbers, the computer stores these with complete accuracy.

For very large numbers, you will notice that the computer will display these in a strange format that uses the letter E. Try the following command:

```
PRINT 1000000000000
```

It will PRINT 1E+12 on the screen. The E stands for 'exponent' and is a shorthand way of displaying $1x10^{12}$ or 1 followed by 12 zeros.

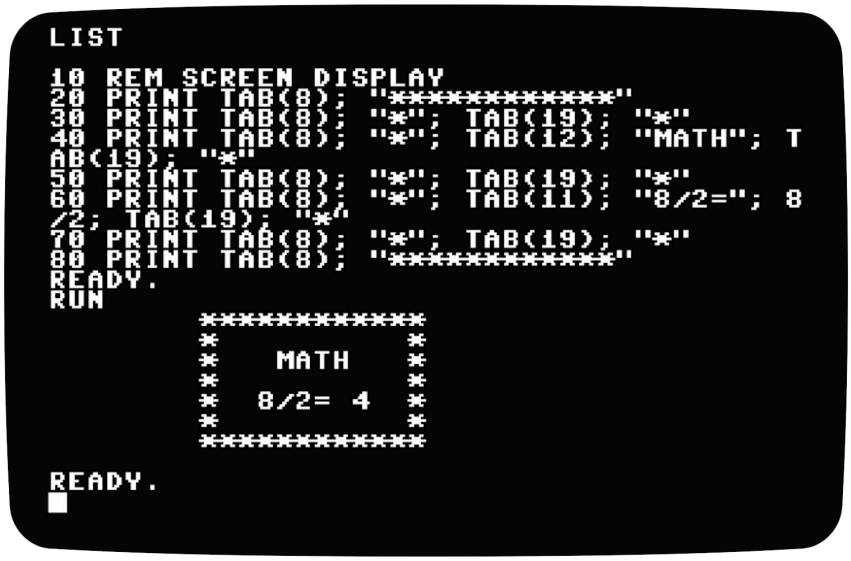
Try displaying and executing some large number calculations of your own. What do you think the ?OVERFLOW ERROR means?



DISPLAYING LISTINGS

So that we have a program to list, type the following code into your Commodore 64. Then RUN the program.

```
10 REM SCREEN DISPLAY
20 PRINT TAB(8); "**************"
30 PRINT TAB(8); "*"; TAB(19); "*"
40 PRINT TAB(8); "*"; TAB(12); "MATH";
TAB(19); "*"
50 PRINT TAB(8); "*"; TAB(19); "*"
60 PRINT TAB(8); "*"; TAB(11); "8/2="; 8/2;
TAB(19); "*"
70 PRINT TAB(8); "*"; TAB(19); "*"
80 PRINT TAB(8); "**************"
```



Let's look at what the above code does.

In line 10, the code REM stands for REMark, the computer doesn't do anything with this line. These are designed to let you add comments into your programs to help you label your code to help you identify and understand sections of your program.

Lines 20 to 80 PRINT * symbols in a frame around some text and a calculation. These lines use the TAB command to position the symbols and text around the screen at the correct positions. PRINT TAB(8) means that the computer will start the display at column 8 on the screen instead of the left-hand edge. The semi-colon symbol is used to chain together a group of items to display when you want to show them all on the same line.

Go ahead and save your program to disk.

```
SAVE "MATH",8
```

Let's explore the LIST command. We already know that typing the command by itself will list out the entire program. However, there are some neat little tricks that will become very useful when you begin entering very large program listings.

You can list a single line by tying the LIST command followed by the line number, for example, LIST 10.

You can look at a range of lines, say lines 40 to 50 by using LIST 40-50. Look at the first 30 lines with LIST -30 or all the lines from 70 until the end of the program with LIST 70-.

If you have a very long program and you want to scroll through the listing while looking for a particular section of code, just use the LIST command on its own. You will notice that the code scrolls very fast. However, you can slow down the list scrolling by holding down the CTRL key. When you see the section you want, hit the STOP key to abort, or BREAK, the listing. You can then LIST the specific line or range of lines you were looking for.

CORRECTING MISTAKES

Mistakes are an unavoidable part of computer programming. Sometimes you need to correct a simple mistyped line, maybe you want to change something already in the program or you want to add in a line you forgot to when you first keyed in your code.

Say we want to change our Math screen display program from earlier. Instead of calculating 8/2 we want to calculate 8*2. We could simply retype line 60 in and replace it with our new line, but because we only want to change to characters in the line, we can use our cursor control keys to edit the line instead.

You could LIST the entire program code listing, but because we know which line we want to edit, we will just list line 60. Then we use the cursor control keys, along with the SHIFT key to move our cursor over the first / symbol. Then we can type over that with the * symbol instead. (If you ever need to edit a line that needs more room for your changes you will need to use the INSRT key to add in the required number of spaces before typing in your change.)

Once you have made the changes, press the RETURN key to save your changes.

```
LIST 60

60 PRINT TAB(8); "*"; TAB(11); "8*2="; 8
2; TAB(19); "*
READY.
```

Often you will want to add in a line into your code. For example, we want to add in a line to clear the screen before anything is PRINTed to the screen. You don't need to edit any line numbers to do this. This is why we use multiples of 10 for our line numbers, it makes adding in new lines much easier. Try:

```
5 PRINT CHR$(147)
```

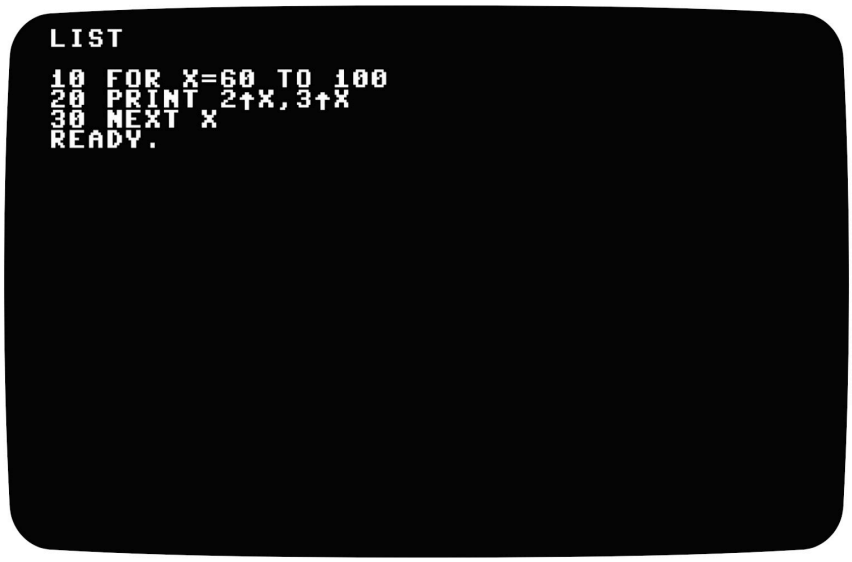
```
5 PRINT CHR$(147)
LIST

5 PRINT CHR$(147)
10 REM SCREEN DISPLAY
20 PRINT TAB(8); "**********"
30 PRINT TAB(8); "*"; TAB(19); "*"
40 PRINT TAB(8); "*"; TAB(12); "MATH"; T
AB(19); "*"
50 PRINT TAB(8); "*"; TAB(19); "*"
60 PRINT TAB(8); "*"; TAB(11); "8*2="; 8
*2; TAB(19); "*
70 PRINT TAB(8); "*"; TAB(19); "*"
80 PRINT TAB(8); "**********"
READY.
```

ERROR MESSAGES

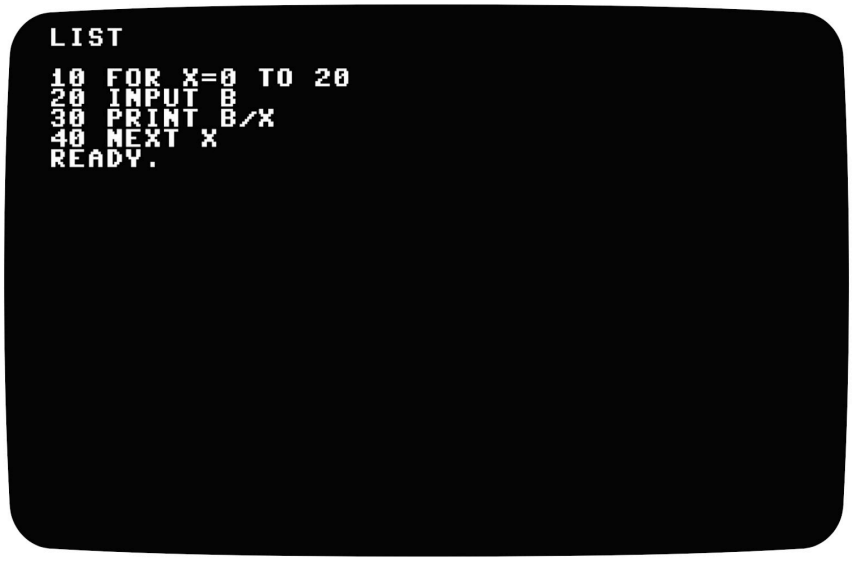
Mistakes or errors in code are called bugs and the processes of troubleshooting and fixing these errors is called debugging. The Commodore 64 has a range of helpful error messages to help you track down and fix these bugs in your code.

Try the following typing in and RUNning the following small programs that each contain an error. You should be able to see which line the error happens on.

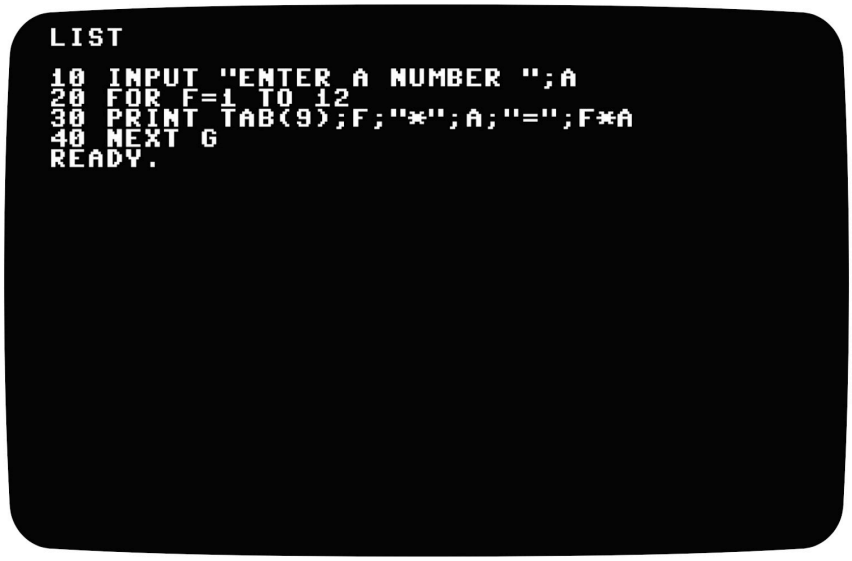
```
10 FOR X=60 TO 100
20 PRINT 2↑X,3↑X
30 NEXT X
```



```
10 FOR X=0 TO 20
20 INPUT B
30 PRINT B/X
40 NEXT X
```



```
10 INPUT "ENTER A NUMBER ";A
20 FOR F=1 TO 12
30 PRINT TAB(9);F;"*";A;"=";F*A
40 NEXT G
```



APPENDIX A
ASCII CHARACTER SET

ASCII Code	Character	ASCII Code	Character	ASCII Code	Character	ASCII Code	Character
0		64	@	128		192	─
1		65	A	129	Orange	193	♠
2		66	B	130		194	│
3	STOP	67	C	131	RUN	195	─
4		68	D	132		196	─
5	White	69	E	133	F1	197	─
6		70	F	134	F3	198	─
7		71	G	135	F5	199	│
8		72	H	136	F7	200	│
9		73	I	137	F2	201	╮
10		74	J	138	F4	202	╯
11		75	K	139	F6	203	╭
12		76	L	140	F8	204	╰
13	RETURN	77	M	141	SHIFT+RETURN	205	╲
14	Lower Case	78	N	142	Upper Case	206	╱
15		79	O	143		207	╭
16		80	P	144	Black	208	╮
17	Cursor Down	81	Q	145	Cursor Up	209	●
18	Reverse On	82	R	146	Reverse Off	210	─

ASCII Code	Character	ASCII Code	Character	ASCII Code	Character	ASCII Code	Character
19	HOME	83		147	CLR	211	
20	DEL	84		148	INS	212	
21		85		149	Brown	213	
22		86		150	Light Red	214	
23		87		151	Dark Grey	215	
24		88		152	Grey	216	
25		89		153	Light Green	217	
26		90		154	Light Blue	218	
27		91		155	Light Grey	219	
28	Red	92		156	Purple	220	
29	Cursor Right	93		157	Cursor Left	221	
30	Green	94		158	Yellow	222	
31	Blue	95		159	Cyan	223	
32	SPACE	96		160	SHIFT+SPACE	224	
33		97		161		225	
34		98		162		226	
35		99		163		227	
36		100		164		228	
37		101		165		229	
38		102		166		230	
39		103		167		231	
40		104		168		232	
41		105		169		233	

ASCII Code	Character	ASCII Code	Character	ASCII Code	Character	ASCII Code	Character
42		106		170		234	
43		107		171		235	
44		108		172		236	
45		109		173		237	
46		110		174		238	
47		111		175		239	
48		112		176		240	
49		113		177		241	
40		114		178		242	
51		115		179		243	
52		116		180		244	
53		117		181		245	
54		118		182		246	
55		119		183		247	
56		120		184		248	
57		121		185		249	
58		122		186		250	
59		123		187		251	
60		124		188		252	
61		125		189		253	
62		126		190		254	
63		127		191		255	

APPENDIX B
SCREEN MEMORY CODES

Screen Code (Normal / Reverse)		Character (Upper / Lower)		Screen Code (Normal / Reverse)		Character (Upper / Lower)	
0	128	@		64	192		
1	129	A	a	65	193		A
2	130	B	b	66	194		B
3	131	C	c	67	195		C
4	132	D	d	68	196		D
5	133	E	e	69	197		E
6	134	F	f	70	198		F
7	135	G	g	71	199		G
8	136	H	h	72	200		H
9	137	I	i	73	201		I
10	138	J	j	74	202		J
11	139	K	k	75	203		K
12	140	L	l	76	204		L
13	141	M	m	77	205		M
14	142	N	n	78	206		N
15	143	O	o	79	207		O
16	144	P	p	80	208		P
17	145	Q	q	81	209		Q
18	146	R	r	82	210		R
19	147	S	s	83	211		S

Screen Code (Normal / Reverse)		Character (Upper / Lower)		Screen Code (Normal / Reverse)		Character (Upper / Lower)	
20	148	T	t	84	212		T
21	149	U	u	85	213		U
22	150	V	v	86	214		V
23	151	W	w	87	215		W
24	152	X	x	88	216		X
25	153	Y	y	89	217		Y
26	154	Z	z	90	218		Z
27	155	[		91	219	+	
28	156	£		92	220		
29	157	]		94	221		
30	158	↑		94	222		
31	159	←		95	223		
32	160	SPACE		96	224	SPACE	
33	161	!		97	225		
34	162	"		98	226		
35	163	#		99	227		
36	164	$		100	228		
37	165	%		101	229		
38	166	&		102	230		
39	167	'		103	231		
40	168	(		104	232		
41	169	)		105	233		
42	170	*		106	234		

Screen Code (Normal / Reverse)		Character (Upper / Lower)	Screen Code (Normal / Reverse)		Character (Upper / Lower)
43	171	+	107	235	
44	172	,	108	236	
45	173	-	109	237	
46	174	.	110	238	
47	175	/	111	239	
48	176	0	112	240	
49	177	1	113	241	
50	178	2	114	242	
51	179	3	115	243	
52	180	4	116	244	
53	181	5	117	245	
54	182	6	118	246	
55	183	7	119	247	
56	184	8	120	248	
57	185	9	121	249	
58	186	:	122	250	
59	187	;	123	251	
60	188	<	124	252	
61	189	=	125	253	
62	190	>	126	254	
63	191	?	127	255	

APPENDIX C
ERROR MESSAGES

BAD DATA
String data was received from an open file, but the program was expecting numeric data.

BAD SUBSCRIPT
The program was trying to reference an element of an array whose number is outside of the range specified in the DIM statement.

BREAK
Program execution was stopped because you hit the <STOP> key.

CAN'T CONTINUE
The CONT command will not work, either because the program was never RUN, there has been an error, or a line has been edited.

DEVICE NOT PRESENT
The required I/O device was not available for an OPEN, CLOSE, CMD, PRINT#, INPUT#, or GET#.

DIVISION BY ZERO
Division by zero is a mathematical oddity and not allowed.

EXTRA IGNORED
Too many items of data were typed in response to an INPUT statement. Only the first few items were accepted.

FILE NOT FOUND
If you were looking for a file on tape, and END-OF-TAPE marker was found. If you were looking on disk, no file with that name exists.

FILE NOT OPEN
The file specified in a CLOSE, CMD, PRINT#, INPUT#, or GET#, must first be OPENed.

FILE OPEN
An attempt was made to open a file using the number of an already open file.

FORMULA TOO COMPLEX
The string expression being evaluated should be split into at least two parts for the system to work with, or a formula has too many parentheses.

ILLEGAL DIRECT
The INPUT statement can only be used within a program, and not in direct mode.

ILLEGAL QUANTITY
A number used as the argument of a function or statement is out of the allowable range.

LOAD
There is a problem with the program on tape.

NEXT WITHOUT FOR
This is caused by either incorrectly nesting loops or having a variable name in a NEXT statement that doesn't correspond with one in a FOR statement.

NOT INPUT FILE
An attempt was made to INPUT or GET data from a file which was specified to be for output only.

NOT OUTPUT FILE
An attempt was made to PRINT data to a file which was specified as input only.

OUT OF DATA
A READ statement was executed but there is no data left unREAD in a DATA statement.

OUT OF MEMORY
There is no more RAM available for program or variables. This may also occur when too many FOR loops have been nested, or when there are too many GOSUBs in effect.

OVERFLOW
The result of a computation is larger than the largest number allowed, which is 1.70141884E+38.

REDIM'D ARRAY
An array may only be DIMensioned once. If an array variable is used before that array is DIM'D, an automatic DIM operation is performed on that array setting the number of elements to ten, and any subsequent DIMs will cause this error.

REDO FROM START
Character data was typed in during an INPUT statement when numeric data was expected. Just re-type the entry so that it is correct, and the program will continue by itself.

RETURN WITHOUT GOSUBA
RETURN statement was encountered, and no GOSUB command has been issued.

STRING TOO LONG
A string can contain up to 255 characters.

?SYNTAX ERROR
A statement is unrecognizable by the Commodore 64. A missing or extra parenthesis, misspelled keywords, etc.

TYPE MISMATCH
This error occurs when a number is used in place of a string, or vice-versa.

UNDEF'D FUNCTION
A user defined function was referenced, but it has never been defined using the DEF FN statement.

UNDEF'D STATEMENT
An attempt was made to GOTO or GOSUB or RUN a line number that doesn't exist.

VERIFY
The program on tape or disk does not match the program currently in memory.