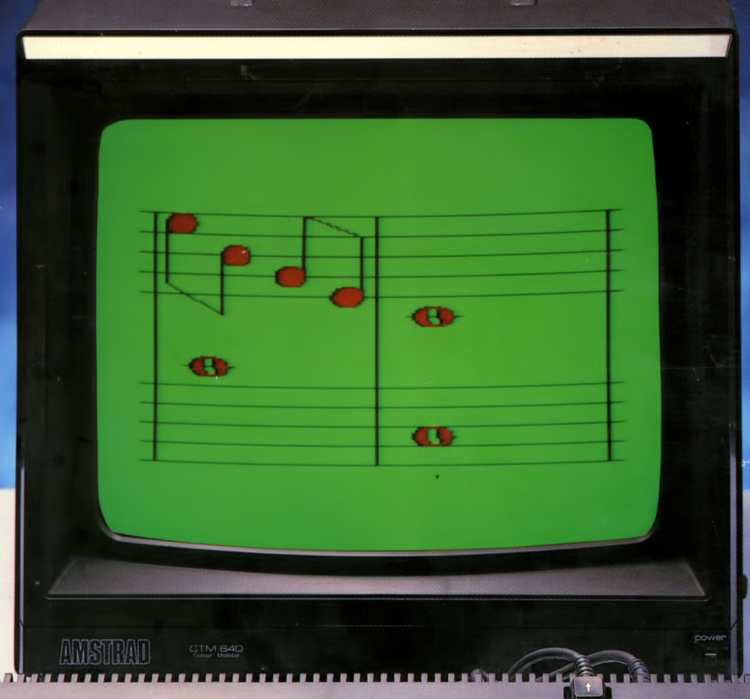
Making Music on the Amstrad CPC 464 & 664

Ian Waugh



Making Music on the Amstrad CPC 464 & 664

Ian Waugh

First published 1985 by:
Sunshine Books (an imprint of Scot Books Ltd.)
12–13 Little Newport Street
London WC2H 7PP

Copyright © Ian Waugh, 1985

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording and/or otherwise, without the prior permission of the Publishers.

British Library Cataloguing in Publication Data
Waugh, Ian
Making music on the Amstrad CPC 464 & 664
Microcomputer
I. Title
780'.28'5404 MT723

ISBN 0–946408–82–3

Cover photograph by Ian Batchelor
Typeset and printed in England by Nene Litho,
Irthlingborough, Northants

Contents

		Page
Introduction		
1	**What is Sound?**	**3**
	The nature of sound	3
	Looking at sound waves	4
	The sound of the Amstrad	6
	Pitch	7
	Volume	8
	Duration	10
	Motility	*10*
	Timbre	12
2	**What is Music?**	**15**
	The language of music	15
	The pitch of a note	16
	Scales	18
	Enharmonics	21
	Accidentals	21
	The length of a note	21
	Beats in the bar	23
	Triplets, ties and slurs	23
	Harmony and chords	24
3	**The Sound Command**	**27**
	Channel status	28
	Rendezvous	*29*
	Hold and RELEASE	*31*
	Flush	*32*
	Pitch	33

Duration 33
Volume 34
Volume envelope 34
Tone envelope 35
Noise 35
Putting them all together 35
The state of the sound queue 36
Stereo sound and external speakers 38

4 Programming Scales and Pitches 41
Calculating the pitch number 41
The scale of equal temperament 42
The scale of just intonation 42
The out-of-tune scale 43
Microtonal scales 45
Calculating notes in the equal-tempered scale 45
Alternative octave and note numbers 46
Constructing microtonal scales 47
The shape of music to come 48

5 ENV and ENT — the Volume and Tone Envelopes 49
The volume envelope (ENV) 49
The attack phase 50
The decay phase 51
The sustain phase 51
The release phase 51
A complete ADSR envelope 52
The ENV parameters 52
Special P, Q, R values 54
An envelope graph generator 55
Designing envelopes 68
The hardware ENV 68
The tone envelope (ENT) 70
Special S, T, V and W values 71
Designing tone envelopes with the graph program 71
The hardware ENT 72

Multi-section envelopes 72
The envelope storage area 72
Hardware envelope storage 75
Increasing the number of envelope sections 75
Experimenting with the programs 77
Instrument characteristics 78

6 Musical Miscellanea 81
Vibrato and tremolo — pitch and amplitude modulation 81
Creating vibrato with the tone envelope 82
Producing tremolo effects 83
Trills 84
Producing echoes on the Amstrad 85
Chorus, phasing, flanging and other special effects 86
Delay effects on the Amstrad 87
Ring modulation for producing bells 88

7 Zaps and Zings and other things 93
White noise 93
Simple sound effects 94
Designing a rhythm unit 96
Soundscapes 99
Further experiments in soundscapes 101

8 Playing the Amstrad 103
Monophonic and polyphonic instruments 103
The Amstrad as a monophonic instrument 103
The Amstrad synthesiser 104
Adding a bass sequencer 109
Altering the bass riff 110
Developing the sequencer 110

9 Making Micro Music 113
Talking music 113
Note durations 115
Note to number conversions 115

Using the alternative octave numbering system *117*
Error routines 117
Playing two-part or three-part tunes 118
Stereo positioning 118
Modifications and suggestions 124
Debugging the data *125*
Saving the tune 125
Experimenting with the programs 128

10 Computer Compositions **137**
Human compositions — algorithms and heuristics 138
Aspects of a composition *138*
Note analysis in composition 144
A total tune analysis program 154

11 Your Amstrad Composes **157**
The harmonic structure of popular songs 157
Producing acceptable results 158
Random harmonic compositions 158
Using chords as a compositional base 159
Adding rhythm variations 166
Further extensions and modifications 167
Applying further control to random note selections 168
Improving the melody 169
Designing and developing programs 170
The Upside-Down Composer program 171
Retrogressive progression 173
Sing-along-a-matic 174

12 Appendices

Introduction

This book is for everyone with an Amstrad computer. Whatever your reasons for buying a computer, your decision to purchase an Amstrad has been rewarded with a very versatile and powerful machine. If all its features were investigated thoroughly the resulting books would occupy several shelves. This book is about one of its most exciting features — the sound generator. The inclusion of a sound generating chip in personal computers is a fairly recent development and it is no doubt responsible, along with advanced graphics capabilities, for the growing interest in computers.

Over 75% of all information we receive from the outside world comes through our sense of sight so it is hardly surprising that graphics tend to dominate computer advertisements. But what is a missile exploding without a bang and what is a flight simulator or powerboat race without the whine of the engines? Music and sound have a greater effect upon us that you may realise. The next time you are watching a horror movie or car chase on TV, turn the sound off for a moment and you will see how much the music and effects contribute to the excitement.

As well as sound effects, the Amstrad can be programmed to produce music. Armed with the ability to produce ordered sequences of notes we have a complete music system which can not only play tunes in three-part harmony but which has the speed and decision-making attributes of a computer.

The aim of this book is to act as a springboard for further experiments and programs which, I hope, you will write and develop. The accent is on sound and music and how you can get the best from the powerful sound generating system incorporated in the Amstrad.

The programs are written in a structured manner and are documented so you can understand the workings behind them. Generally, they will not have many frills, in order to minimise the time required to enter them and to cut down on mistakes. Suggestions for further experiments, alterations and developments are also made, usually in such cases where a subject has too many aspects and is too complex to tackle completely — without writing another book. It is hoped that you will explore further those areas of special interest to you.

This book is intended to be read from Chapter 1 onwards but you can dip into it at whatever chapter takes you fancy. For those who decide to read it so, I make no apologies for the odd repetition of information and constant

reference to other chapters. Those well versed in music and computing will forgive me, those who are not so accomplished will, I hope, thank me.
Before you begin, perhaps I can refer you to Appendix 2 which contains hints and tips about entering and merging programs and other information you may find of interest. Please read it before entering programs.
If you are reluctant to enter some of the longer programs in this book because of the time it will take, a cassette (£5.95 including postage and packing) of all the programs is available from the author. Please send a cheque/PO payable to Ian Waugh to Sunshine Books, 12–13 Little Newport Street, London, WC2H 7PP. Available for readers in the UK only.
I would like to thank David Lawrence once again for his advice and suggestions, to Bill Poel of Amsoft for his cooperation, to Amstrad for producing such a wonderful computer and, finally, to my Mother and Father who continue to support and encourage me in all my ventures.

CHAPTER 1
What is Sound?

This chapter looks at sound from the computer's point of view. Amstrad BASIC is a fast and powerful language. It includes several commands to help the user program the sound generator. Even a few casual experiments with these commands will reveal how complex and difficult they can sometimes be to control. They have been well designed, however, so that we do not always need to specify every single parameter; those we omit default to sensible values. This means we can learn how to use the sound generator quite painlessly, adding parameters as required.
The User Guide devotes only a few pages to the sound facilities and a lot more information is required to get the best from the system. Unfortunately, Amstrad's Concise BASIC Specification has little to add. The problem here is exactly the same as the one we faced when we started to learn BASIC. The computer has a set way of operating and in order to control it we must give it instructions in its own terms. This means we need to know something about the properties of sound and how to convert this information into a program the computer can understand.

The nature of sound

Sound is sometimes difficult to understand because it is invisible. A sound is produced when an object is struck or rubbed or, in scientific terms, otherwise vibrated. This in turn causes the air to vibrate. These vibrations are sensed by the ear and we perceive them as sound.
Musical instruments which produce sound by being struck include the drums, piano, gong and xylophone. Stringed instruments such as the violin produce sound by being rubbed with a bow. Brass instruments such as the trumpet and trombone are played by blowing and vibrating the lips. This excites the air inside the instrument which vibrates at a pitch inversely proportional to the length of the brass tubing. The flute is played by blowing across the mouthpiece to excite the air column inside it. The same principle is at work when you blow across the mouth of a bottle. Instruments such as the oboe, clarinet and saxophone contain a reed which vibrates in response to vibrations from the lips.

Just as sound is caused by vibrations in the air, so the sound chip generates its sounds with electrical vibrations. Basically, it sets up a series of oscillations, the higher the pitch, the faster the oscillations. These oscillations are sent to the loudspeaker which vibrates at the same frequency and produces a sound.

Looking at sound waves

Sound vibrations travel in a series of waves and different sounds produce different waveforms. If we play a sound through a microphone and feed it into an oscilloscope we can see what its waveform looks like.

A sine wave is a pure tone of a single pitch or frequency which is usually only produced by a tuning fork or by electronic means. It is possible to produce many sounds by combining sine waves in the right proportions. This process is known as additive synthesis because waves are added together and it is used in some commercial synthesisers. Because of the large number of sine waves it is often necessary to add, it is a costly and time-consuming process.

The following program will plot sine waves according to the amplitude (loudness) and frequency (pitch) you input. It demonstrates how frequency and amplitude affect the waveform and we will use it later in this chapter to plot the results of additive synthesis.

```
100 REM PROGRAM 1.1
110 REM Sine Wave Plotter
120 :
130 MODE 1
140 WINDOW #0,1,40,1,5
150 ORIGIN 0,160,0,639,319,0
160 REM Black Graph Screen, White Graph
170 INK 2,0:INK 3,13:CLG 2
180 DEG
190 :
200 WHILE k$<>"F"
210 INPUT "Frequency (1-20)";freq
220 INPUT "Amplitude (5-150)";amp
230 GOSUB 290:REM Draw Waveform
240 PRINT"Press ANY KEY to enter another
 wave","'C' to clear screen, 'F' to fini
sh"
250 k$="":WHILE k$="":k$=UPPER$(INKEY$):
WEND
260 IF k$="C" THEN CLG 2
270 WEND
```

```
280 END
290 :
300 REM Draw Waveform
310 MOVE 0,0
320 FOR stime=0 TO 639 STEP 2
330 DRAW stime,amp*SIN(freq*stime),3
340 NEXT stime
350 RETURN
```

Commentary

The program houses a simple graph plotting procedure between lines 320 and 340. Line 330, which we will extend later, performs the calculations.
Try inputting 1 for frequency and 100 for amplitude to begin with. If you increase the frequency you will see how it bunches the waves closer together. Frequency is normally measured in 'cycles per second' and the higher the frequency, the more cycles occur every second. That is, there will be more waves across the X or time axis. An increase in amplitude will make the wave taller without affecting the frequency. In other words the volume will increase but the pitch will remain the same. If you replace line 330 with:

```
330 DRAW stime,amp*SIN(freq*stime)+amp*SIN(freq*2*stime),3
```

you will be adding two waves together and can see the result of additive synthesis. Notice how the waveform changes. You can add more sine waves in line 330 by modifying the variable, freq, in the expression:

```
amp*SIN(freq*stime)
```

and adding it to the line with a + as in the above example. If you modify the value of amp such as:

```
amp*.5*SIN(freq*stime)
```

you are reducing the amplitude so its effect on the resulting wave will be reduced. These additions are known as harmonics and they are what makes each sound distinctive. Most sounds we hear in everyday life have quite complex waveforms and would be made up from many sine waves of varying frequencies and amplitudes. More examples are given in the section about timbre later in this chapter.

The sound of the Amstrad

Another form of synthesis, known as subtractive synthesis, takes a waveform and filters out certain harmonics. This method is more common than additive synthesis and is in general use in most synthesiser systems. A tone control is a simple filter and blocks out the higher frequencies as you increase its effect. Before you can start filtering, you need something to filter. A sine wave, consisting of only one frequency would be of little use. The best waveforms are those which contain a lot of harmonics which give you plenty of body to chip away from. Most synthesisers offer triangular, square and sawtooth waveforms. The triangular wave is very like a sine wave but contains higher harmonics, the square wave sounds a little like a clarinet and the sawtooth wave produces a sound with reed-like qualities. Try this short program:

```
10 FOR pitch=1600 to 20 STEP -10
20 SOUND 1,pitch
30 NEXT pitch
```

This plays some of the pitches the sound chip can produce, albeit in a very uneven fashion, but it serves to illustrate the waveform produced by the sound chip. Can you tell which type of waveform it is? It's not a terribly easy task because the low pitches sound like a rasp and the high pitches are 'plinky'. The middle range, however, is quite mellow and if you alter line 10 to:

```
10 FOR pitch=500 to 100 STEP -10
```

you may sense a clarinet-like quality about the tone. The sound generator produces a square wave. It's not quite a perfect square wave, however, and you can hear how the tone varies as the pitch changes.
In order for a sound to exist at all it must have four parameters:

(i) Pitch.
(ii) Volume.
(iii) Duration.
(iv) Timbre.

The sounds produced by acoustic instruments are actually very complex and change throughout their duration. We will look at these four aspects of sound and see how they relate to musical instruments.

Pitch

The pitch of a note means how high or low it is on the musical scale. The word frequency is often used in the same way, but frequency is more properly an attribute of the waveform, namely how many times it vibrates or oscillates per second. The human ear can sense sounds with a frequency of from 20 to 20,000 cycles per second. A cycle per second is called a Hertz, and is abbreviated to Hz.

A tune consists of a series of pitches which have a definite relation to each other. In western music, tunes are based on the notes we can play on a piano, where each note is a semitone away from its neighbours. The notes are grouped into sections, which we will look at later, to form scales such as C major and B minor.

On a piano the pitch of the notes is fixed. You cannot, unless the piano is out of tune, play in the cracks. Even if you are tone deaf, as long as you hit the right keys you will produce pleasant music. Other instruments such as those of the string and brass family require more control over pitch and notes can be 'slurred' from one to the other. If this takes place over several notes it is known as a portamento, and is easily produced on most synthesisers. The same thing on a piano, harp or xylophone would result in a series of discrete tones or semitones known as a glissando. Both effects are much used by jazz musicians and can add a human touch to synthesised music. The next program demonstrates these effects.

```
100 REM PROGRAM 1.2
110 REM Portamento & Glissando Demo
120 :
130 FOR pitch=478 TO 239 STEP -1
140 SOUND 1,pitch,1
150 NEXT pitch
160 :
170 FOR d=1 TO 1000:NEXT d
180 :
190 FOR note=0 TO 12
200 freq=261.626*(2^(note/12))
210 pitch=ROUND(125000/freq)
220 SOUND 1,pitch,10
230 NEXT note
```

The portamento is produced by passing very quickly through pitch values 478 to 239. Each pitch sounds for 1/100th of a second, set by the last parameter of the SOUND command in line 140. If you remove this last parameter

completely the duration will default to 1/5th of a second and you will hear the pitch creep upwards very slowly. If you have a good sense of pitch you may be able to hear individual tones proving that the Amstrad is playing discrete pitches after all. The portamento effect is produced because the pitch is moving so rapidly.
The glissando example produces a series of semitones played over the same pitch range. Lines 200 and 210 calculate the pitch values. A full description of the SOUND parameters is given in Chapter 3 and the pitch formula is explained in Chapter 4. You can skip ahead if you want to know how they work: these examples are simply to illustrate the effects.
The Amstrad can produce a wide range of frequencies. Oriental music uses pitches which are less than a semitone apart, which is why it often seems out of tune to westerners.

Volume

This is how loud or quiet a sound is. At first, loudness as a quality of sound may seem rather simple and unimportant. It is not quite as straightforward as that, however, because many factors affect the perceived volume of a sound. Reverberation, echo, vibrato and duration all tend to increase volume as does the addition of harmonics. For example, a sound lasting 1/100th or even 1/10th of a second will not seem as loud as a sound lasting one second. As you will have heard from some of the previous program examples, the volume tends to alter with pitch. If you are writing a tune in two or three parts you may find that, at certain points in the tune, some lines get lost behind others. This is a result both of the properties of sound and the sound chip and can only be overcome by altering the characteristics of individual lines where required. You will find that generally this is not a serious problem.
The loudness of a sound will vary during its production. For example, a piano, xylophone, or any other percussive instrument produces a note which achieves its maximum volume immediately upon playing and then dies away. A violin takes just a fraction of a second for its note to reach full volume. Brass instruments sound with a sharp attack, even when played quietly, as an initial gust of breath is required to start the air in the tube vibrating. This variation in volume is called the loudness contour, or envelope, of a sound, and plays an important part in determining instrument characteristics. Try this:

```
100 REM PROGRAM 1.3
110 REM Volume Demo
120 :
130 FOR vol=1 TO 7
```

```
140 SOUND 1,478,5,vol,1
150 NEXT vol
160 :
170 FOR d=1 TO 1000:NEXT d
180 :
190 FOR vol=7 TO 1 STEP -1
200 SOUND 1,478,5,vol
210 NEXT vol
```

The first example sounds like a recording of an instrument being played backwards. It sounds unnatural, and it is, because most sounds don't happen that way: they don't work up to a crescendo and stop. The second example sounds like a percussive instrument being tapped smartly. If you run both sequences together by removing line 170 you will see how the sound has become more natural. The ability to produce backward sounds is useful in synthesis and we can make use of it on the Amstrad to create lots of interesting effects.
Rather than control the volume with a FOR/NEXT loop, we can use the ENV command to create a predetermined set of volume characteristics like this:

```
100 REM PROGRAM 1.4
110 REM ENV Demo
120 :
130 ENV 1,16,15,10
140 :
150 FOR note=0 TO 12
160 freq=261.626*(2^(note/12))
170 pitch=ROUND(125000/freq)
180 SOUND 1,pitch,100,0,1
190 NEXT note
```

This creates a percussive envelope and produces a piano-like sound. Alter line 130 to:

130 ENV 1,16,17,4

This gives us our backward sound and if it did not cut off so sharply it could form the start of a violin-type envelope. ENV is explained in Chapter 5 and one of the programs in that chapter shows graphically exactly how the ENV and ENT commands affect the amplitude and pitch.

Duration

Again, the complexity of a note's duration can be deceptive. In order for a sound to exist at all it needs *some* duration. As far as the Amstrad is concerned this will not normally be below 1/100th of a second. This is the minimum duration you are able to assign a note in the SOUND command, and the time parameters in the ENV and ENT commands are given in 1/100ths of a second, too. This gives us very fine control over a sound, so we can produce finely-tuned volume (ENV) and pitch (ENT) envelopes to suit most requirements.

From a psychological point of view, it is interesting to note the difference in time perception between individuals. There seems to be little evidence to show that a good musical appreciation of pitch, volume and timbre will endow a person with a good sense of time, because timing sense is not normally dependant upon the ear.

A reliable sense of timing, and hence rhythm, plays a great part in the creative production of music. The only attributes a piano player has control over are volume and time. The timbre and pitch are determined by the instrument and composer.

Motility: speed and accuracy

Coordination is often regarded as being of prime importance to a musician. The ability to perform accurately and at speed is at the root of a competent musical performance. A person may be quick and accurate, quick an inaccurate, slow and accurate or slow and inaccurate, all in varying degrees. There is a natural limit to the speed at which a musician can play but this does not determine how good a musician is. Rather, the way a musician makes fine alterations in the timing of a piece will affect the performance.

Such movements and timing can be measured but are well beyond the scope of this book. We can arrange a simple motility test which will be of use not only to musicians but to anyone wanting to develop quick reactions. It may give you an insight into your performance with arcade games. Motility is a measurement of speed and accuracy in movement and can be measured by tapping a key or a pencil and recording the average number of taps made each second. The following program does this and records the number of taps made in a five second period.

```
100 REM PROGRAM 1.5
110 REM Motility Tester
120 :
130 KEY DEF 18,0:REM ENTER Repeat Off
140 ENT -1,=478,10,=379,10,=358,10,=319,
```

```
10
150 score=0
160 CLS
170 LOCATE 5,10:PRINT"Tap the ENTER key
repeatedly","   as quikly as possible an
d with","      the minimum of movement."
180 WHILE k$="":k$=INKEY$:WEND
190 tim=TIME+1500
200 :
210 WHILE TIME<tim
220 k$=INKEY$:IF k$=CHR$(13) THEN score=
score+1
230 WEND
240 :
250 SOUND 1,0,90,7,0,1
260 LOCATE 18,14:PRINT"STOP":PRINT
270 PRINT TAB(9)"Your MOTILITY rating is
"
280 PRINT TAB(10) score/5;"taps per seco
nd."
290 PRINT:PRINT TAB(11)"Another Try (Y/N
)?"
300 k$=UPPER$(INKEY$):IF k$="Y" THEN RUN
310 IF k$<>"N" GOTO 300
320 END
```

Commentary

The important part of the program lies between lines 210 and 230 which increment the variable, score, when the user presses ENTER. The ENTER key does not normally repeat when held down but line 130 confirms this. Replace it with:

130 KEY DEF 18,1

and you'll see the difference. The ENT definition in line 140 and the sound command at line 250 produce a little fanfare without using lots of data statements. For the curious and the impatient, this programming method is examined in detail in Chapter 5.

Practice will generally increase your motility rating only slightly. An average for normal adults will be around 8.5 taps per second rising to 9.3 after two or

three weeks practice. Men tend to average one half tap per second faster than women. For such an exercise to be valid as a test of musical ability, the test should be made on a movement similar to the one used during performance. A pianist, therefore, should be tested on a piano key and a violinist on a violin string. On simple tapping tests such as provided by the program, the highest speed recorded is about 12 taps per second although rates as high as 15 have been reported.

Timbre: the quality of sound

Timbre (pronounced 'tam-burr') or tone colour is that quality of a sound which enables us to distinguish between two sound sources producing sounds at the same pitch. It is usually very much affected by pitch and the sound envelope. For example, we know that the low notes of a clarinet have a sound quality different to that of the high notes. This is evident in the Amstrad's sound generator, too, as we have already heard.
Tone colour is a result of the combination of harmonics in a sound. We saw the effects of adding sine waves in the Sine Wave Plotter Program. Load the program again and REM out line 330. Add line 334 from the following program and run it. Then REM out that line and add line 336.

```
100 REM PROGRAM 1.6
110 REM Additive Synthesis Demo
120 :
333 REM Sawtooth
334 DRAW stime,amp*SIN(freq*stime)+amp*1
/2*SIN(freq*2*stime)+amp*1/3*SIN(freq*3*
stime)+amp*1/4*SIN(freq*4*stime)+amp*1/5
*SIN(freq*5*stime),3
335 REM Square
336 DRAW stime,amp*SIN(freq*stime)+amp*1
/3*SIN(freq*3*stime)+amp*1/5*SIN(freq*5*
stime)+amp*1/7*SIN(freq*7*stime)+amp*1/9
*SIN(freq*9*stime),3
```

The first example, in line 334, adds the second, third, fourth and fifth harmonics to the fundamental or main tone. The fundamental is usually the strongest, that is loudest, frequency and gives the note its pitch. Adding harmonics in this proportion produces a waveform like a sawtooth from which it takes its name. If you add more harmonics you will iron out the bumps and produce a better-looking sawtooth.

Line 336 adds only the odd harmonics to produce a square wave. Again, if you add more harmonics in the same proportion, you will produce a better-looking wave.

Experiment by adding various other harmonics, even by altering the SIN function to COS, and you will produce some quite complex waveforms. If you obtain a more detailed book about sound synthesis which contains harmonic analyses of instrument waveforms you will be able to work out which sine waves are required to produce the sound.

On the Amstrad, we have no control over the waveform but by clever use of the SOUND, ENV and ENT commands we can trick the ear into thinking that what it hears is something other than a dressed-up square wave. This is because the ear takes as much notice of the envelope of a sound as the timbre. There are limits, however, as to what we can do, and we will be testing and exploring these throughout the book.

CHAPTER 2

What is Music?

Music has been called a compromise between chaos and monotony and we can easily find examples of both. Many musicians are experimenting with computers, (with or without sound chips), and many computer enthusiasts are exploring the sound capabilities of their micros. Unfortunately, not all computer users are musicians and some may feel that the benefits of studying music do not outweigh the effort required to learn it. If their aim is simply to get more from their micro, they may have a point, although music brings its own pleasure and rewards.

As this is a book about music, not written solely for the experienced musician, it would be incomplete without some attempt to explain the rudiments of music. Complete books have been written on the subject and it would be foolish to try to duplicate their contents in a few pages. This chapter, however, is intended as a reference section, and lays down at least a few rules to aid those with little prior musical knowledge.

Since you are reading this book, you probably have some interest in music. This chapter aims to provide sufficient information for you to take a piece of music and program it into the computer — and to know what you're doing and why you're doing it.

The language of music

Learning music is like learning any other language, only easier. If you want to be a concert pianist and can't yet read music you have probably left it too late, but it is never too late to learn music for its own sake. It will bring many hours of pleasure and enjoyment.

One of the problems facing the newcomer to music is the sight of hundreds of black dots on a page full of lines — they all look the same. If you aim is to take a sheet of music, sit down and play it then you need to study. But, for the purposes of this book, you need only read about the ideas and principles behind the dots, and refer back to this chapter when necessary. It is intended to be a potted reference section rather than an intensive teaching course.

We all know what music looks like, even if we can't read it. In our programs we will not always be referring to conventional music notation, but it helps to know what it is and what it represents.

Music is written the way it is because of convention. It is simply the way that has developed over the years. The two most important items of information we get from a piece of music are the pitch of the note and how long it lasts. We will look first at how pitch is represented.

The pitch of a note

In conventional notation, notes are arranged on a set of five lines called a staff or stave. Pitch is indicated by the vertical position of the note: the higher the note, then higher the pitch. The notes are given letter names from A to G. When you reach G you start again with A, as shown in *Figure 2.1*.

Figure 2.1: Notes on a Stave.

Notes can be placed above and below the lines to extend the range. These notes are written on and between short lines called leger lines which are really just an extention of the stave, as shown in *Figure 2.2*. The stave could consist

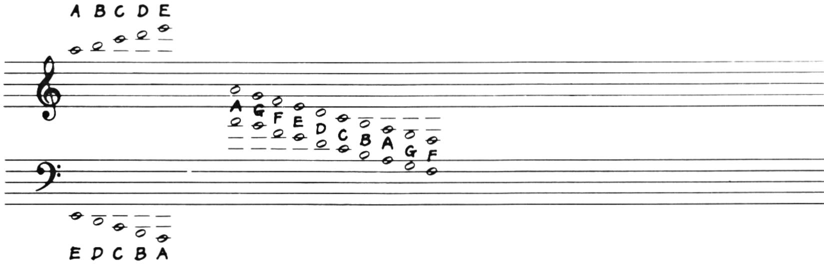


Figure 2.2: Leger Lines.

of a set of ten or more lines but that would be very confusing and difficult to read. The number of leger lines can be extended as far as you wish but, again, too many make the music difficult to read.

To increase our range of notes still further and maintain readability, we can add another stave below the first one. To distinguish one stave from the other, each is given a clef sign which shows the position of the notes in relation to the stave. The two most common clefs, and the only ones we will concern ourselves with, are the treble or G clef and the bass or F clef, shown in Figures 2.1 and 2.2. The treble clef loops about the line which represents G, and the bass clef has two dots which sit either side of the F line.

You will see that as notes on leger lines in the treble clef move down, they correspond to notes on the stave in the bass clef and vice versa. Piano music is normally written on both treble and bass clefs but you will sometimes see two treble clefs, one above the other, or two bass clefs in a similar manner.
If these two clefs are still not enough, you can add a small '8va' with a dotted line above notes to be played an octave higher and below notes to be played an octave lower, as shown in *Figure 2.3*. This should cover all contingencies.

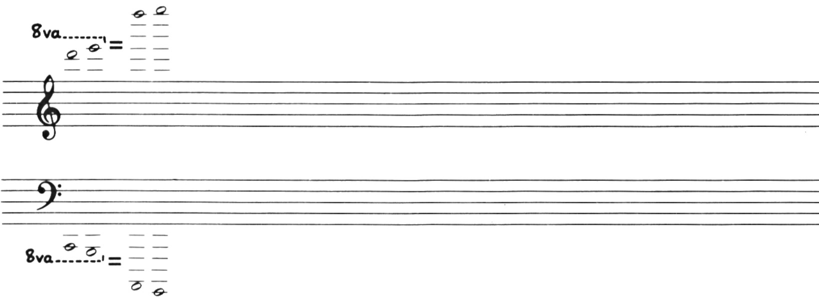


Figure 2.3: Playing an Octave Higher or Lower.

The interval in pitch between two similar letters is known as an octave and represents a doubling in pitch or frequency. The interval between a note on a line and a note in a space is either a tone or a semitone. The Glissando Demo in Program 1.2 and the ENV Demo in Program 1.4 played a series of semitones. Even if you can't yet read music you could probably tell that this was not a 'proper' scale. The following program will play the scale of C starting on middle C.

```
10 FOR scale=1 TO 8
20 READ note
30 SOUND 1,note
40 NEXT scale
50 END
60 DATA 478,426,379,358,319,284,253,239
```

This sounds complete and more musically satisfying than a sequence of semitones. The notes are read from a DATA statement and the progression of the intervals runs: tone, tone, semitone, tone, tone, tone, semitone. This is the sequence of intervals that all major scales follow and is what you would get if you started on middle C on a piano and moved upwards playing the white notes. There are scales other than major scales which we will look at later.

If all this is new to you, don't try too hard to take it all in at once. Just read through the chapter and refer back whenever you wish. *Figure 2.4* should help. It displays all this information in relation to a piano keyboard. The note names are shown along with the notes as they would appear on the stave. Also shown are the pitch numbers or tone periods required by the SOUND command to play a particular pitch. The numbers used by the Amstrad have a special relationship to the pitches they produce (although this may not be obvious from the diagram) and we will have a lot more to say about this later. The Amstrad sound generator has a range of over eight octaves. Figure 2.4 covers only the middle five, mainly to keep it within manageable proportions but musically, you will find the sounds in this range to be the most useful and other pitches are easy to program if you need them.

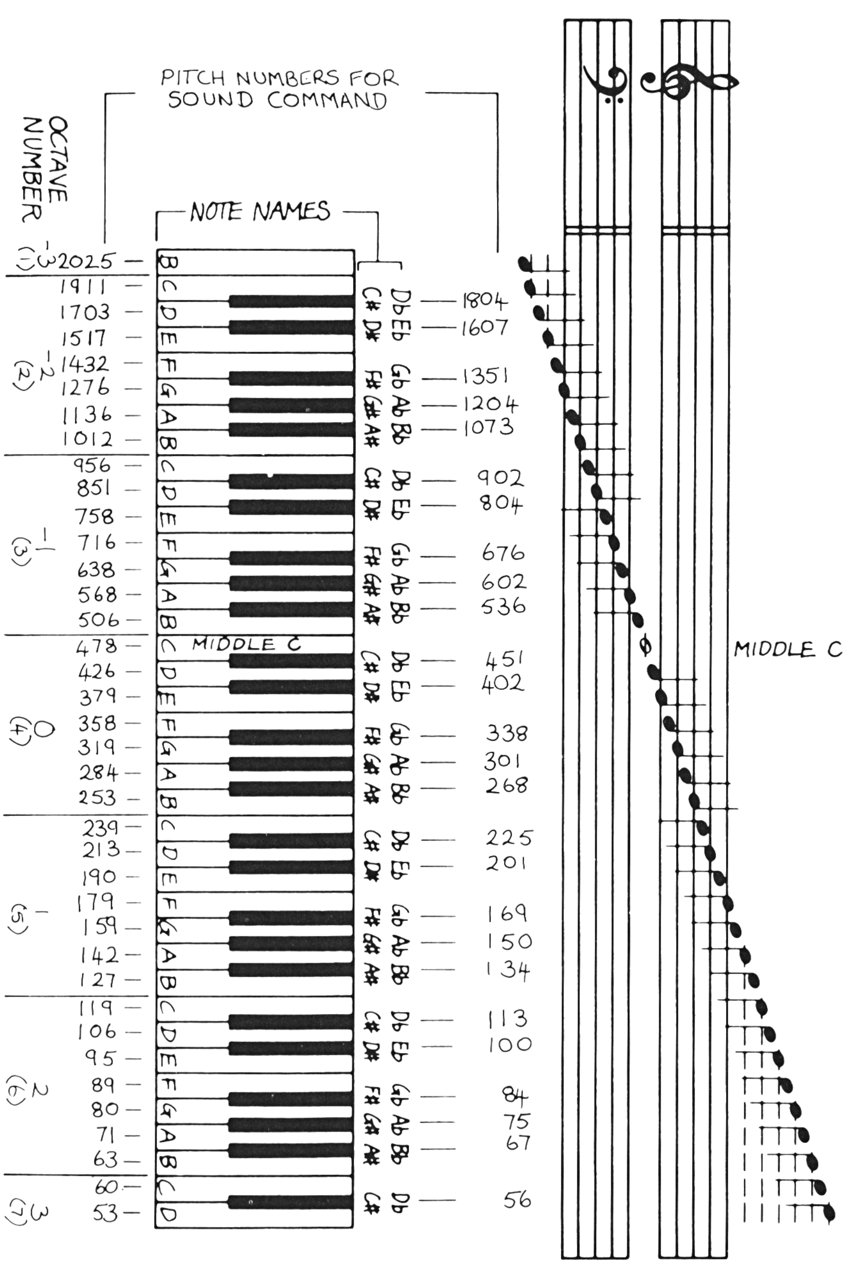
Scales

An important property of music, not always obvious at first sight, is that there are really only 12 separate notes in the whole musical spectrum. When you reach the 13th note, the sequence is simply repeated and the notes sound an octave higher. We can test the musical usefulness of each octave with the following program:

```
10 pitch=3822
20 FOR oct=-3 TO 4
30 PRINT "Octave =";oct
40 SOUND 1,pitch,100
50 pitch=pitch/2
60 NEXT
```

Notes below octave –2 are generally more of a buzz or rumble and above octave 3 the waveform has little character. We said earlier that an octave increase represents a doubling in pitch or frequency and we can produce octaves by doubling and halving the pitch number as above. Figure 2.4 shows two sets of octave numbers: the upper ones conform to the User Guide but I have included the lower set for reasons which will be explained in Chapter 4. We will use the octave numbers in Figure 2.4 — one set or the other — in our musical notation when entering tunes.
From Figure 2.4, where each note on the keyboard is a semitone away from its neighbours, we can see that the intervals in a scale, moving up, have the relation: tone, tone, semitone, tone, tone, tone, semitone. This means we can play any scale at all by selecting a start note and adding those intervals to it. Try it on Figure 2.4. You will realise that every scale other than C contains at least one black piano key. If you study the keyboard, you will realise this is a

Figure 2.4: The Keyboard, Note Names and SID 16-bit Numbers.



consequence of its construction. This means we need a method of telling the player that the music is not based on the scale of C but on some other scale. This is done by including a number of sharps (#) or flats (*b*) at the beginning of the music to form a key signature. They are arranged on the stave in a certain order as shown in *Figure 2.5*. They tell the musician that each note with the same name as the one upon whose line or space the sharp or flat rests is to be played either a semitone higher (sharp) or a semitone lower (flat) throughout the piece. The absence of sharps or flats indicates that the piece is in the key of C.

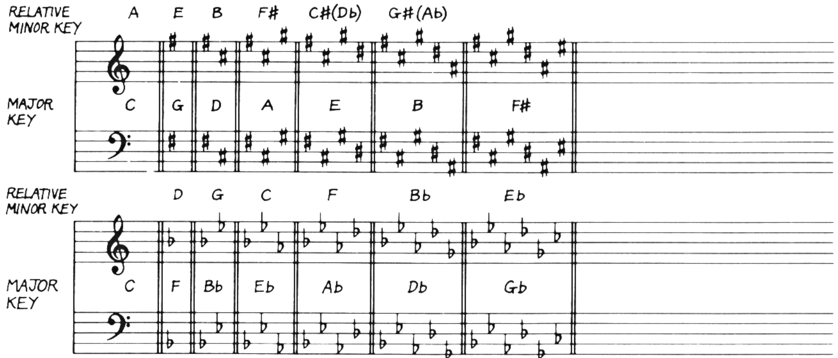


Figure 2.5: Key Signatures.

For example, if we look at the key of D with two sharps, we can see that it tells us to play every F and every C a semitone up. By referring to Figure 2.4 we can see that this produces F# and C#, and if we play a scale starting on D using these two notes we will move through the intervals required to produce a major scale. In a similar way, the key of F indicates that the B note is to be flattened before playing and this produces a scale of F.

Scales provide the basic building blocks from which a tune is constructed and give the music a sense of tonality or affinity with a certain group of pitches. We can play scales on the keyboard other than major scales. If we play only on the black notes of a piano we are using five notes which form a pentatonic (meaning five) scale. It sounds very oriental — or what westerners consider to be oriental.

In more common use is the minor scale. Just to complicate matters, there are technically two forms of minor scale — the melodic and the harmonic. Both forms have the same key signature, shown in Figure 2.5, but vary in the way the actual scales are played. An upward harmonic minor scale moves through the following intervals: tone, semitone, tone, tone, semitone, three semitones, semitone. When playing the scale downwards, the same notes are

used as you might expect.
The melodic minor scale is different. When moving upwards the sequence is: tone, semitone, tone, tone, tone, tone, semitone, and downwards: tone, tone, semitone, tone, tone, semitone, tone. You really need not be too concerned with this if you are just learning music — just remember it exists, and that the notes used in a composition can include any combination of any of the above scales.
When a major and minor key share the same key signature they are known as relative keys, eg A minor is the relative minor of C major and F# minor is the relative minor of A major. The scales of C major and C minor are illustrated later in this chapter in Figure 2.11 so you can compare the notes in the scales with the notes used to construct various chords.
Other scales exist. These contain various numbers of notes and various intervals but most of them are written using the standard notation we are discussing and will probably only come to light, if at all, during an academic discussion of musical theory.

Enharmonics

For the sake of completeness, it is necessary to add that the same note can have two names, eg A flat is the same note as G# because a flattened A produces the same pitch as a sharpened G. Likewise, E flat is the same as D#. These notes are known as enharmonics. This simply means that they sound the same. Musically, if you are playing in a key with flats in the key signature, you will normally refer to and write notes as flats, and similarly for sharps.

Accidentals

It may have occurred to you that while playing in one key you may want to play a note which is not a part of that key. This is done by placing a sharp or flat sign immediately before the note to be altered. The change in pitch refers only to that particular note, not notes an octave up or down, and the change lasts only for the remainder of the bar.
If a note has been sharpened or flattened by the key signature, or a sharp or flat as just described, and you want to naturalise it, you use a natural sign (♮) which, again, applies only to that note and for the duration of that bar. Used in this way, these signs are known as accidentals.

The length of a note

This section is concerned with the timing of the music. There are two aspects involved which should not be confused: the first is the duration of individual

notes and the second is the tempo or speed of a piece of music.
The duration of an individual note is relative only to the other notes in a piece and in no way does it determine the speed or tempo of the music. The duration of notes in standard music notation is shown in *Figure 2.6* along with their English and American names.
The duration value shows how long each note sounds in relation to the others. If a note has a dot placed after it, this lengthens its duration by one half. The tempo of a piece is determined by an instruction given at the beginning.
Rests play an important part in music, too, and rest values are shown in *Figure 2.7*. They have the same names as their note equivalents.

NOTATION	ENGLISH NAME	AMERICAN NAME	DURATION VALUE
[semibreve]	SEMIBREVE	WHOLE NOTE	32
[dotted minim]	DOTTED MINIM	DOTTED HALF NOTE	24
[minim]	MINIM	HALF NOTE	16
[dotted crotchet]	DOTTED CROTCHET	DOTTED QUARTER NOTE	12
[crotchet]	CROTCHET	QUARTER NOTE	8
[dotted quaver]	DOTTED QUAVER	DOTTED EIGHTH NOTE	6
[quaver]	QUAVER	EIGHTH NOTE	4
[dotted semiquaver]	DOTTED SEMIQUAVER	DOTTED SIXTEENTH NOTE	3
[semiquaver]	SEMIQUAVER	SIXTEENTH NOTE	2
[demi-semiquaver]	DEMI-SEMIQUAVER	THIRTYSECOND NOTE	1

Figure 2.6: British and American Note Names.

NOTATION	DURATION VALUE
[semibreve rest]	32
[minim rest]	16
[crotchet rest]	8
[quaver rest]	4
[semiquaver rest]	2
[demi-semiquaver rest]	1

Figure 2.7: Rest Values.

Beats in the bar

The time signature of a piece of music is indicated at the beginning of the stave by two figures, one over the other. The upper figure denotes the number of 'beats in a bar' and the lower figure denotes the length of each beat. For example, a time signature of 2/4 tells us that there are 2 beats to the bar, each made up of a quarter note or crotchet. 3/4 is three beats to the bar, each a crochet, and is the time signature in which most waltzes are written. 4/4 is sometimes written just as a large C. It is referred to as *Common Time* and is by far the most common signature.

Time signatures can be altered at any point in the music and, indeed, can consist of any combination of notes the composer wishes to use. A selection of time signatures is shown in *Figure 2.8* along with various note values which could be used to fill a bar. In practice, you will rarely come across anything more exotic except perhaps in jazz or *avant garde* music.

Figure 2.8: Time Signatures.

Triplets, ties, slurs and staccato

A triplet is a grouping of three notes as shown in *Figure 2.9*, and they are played in the same time as two notes of the same value. If you want to try playing the rhythm, tap two with your left hand and six with your right. That's easy. Now try tapping four with your left and six with your right. Not so easy. We can play triplets on the computer where at least it's a question of programming, not co-ordination.

Figure 2.9: Triplets.

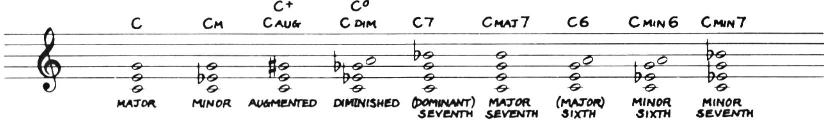
The tie is a curved line which 'ties' two notes together as shown in *Figure 2.10*. It means that the time value of the second note, and any further tied notes, are added to the first, and all of them are played as one long note. This is most often used when a composer wants a note to sound for more than one bar, but

Figure 2.10: Ties.

it can also be found within a bar to join notes of odd time values.
Note that the tie can *only* join notes of the same pitch. If you see similar looking lines which seem to join notes of different pitches, these are slurs and used to indicate that the two notes should be played as smoothly as possible.
The opposite of slur is staccato. This is a dot placed above a note and indicates that it is to be played in a quick, sharp manner and does not have to sound for its full written duration value.

Harmony and chords

Harmony refers to tones sounding simultaneously. Even if we restrict ourselves to a single octave we have 13 notes which can be combined in various ways to create thousands of harmonies, most of which would be quite unmusical. Restricting ourselves to playing only three or four notes at once still produces a lot of combinations. Over the years, certain combinations of intervals have proven useful in composition and for providing a background to a melody. These combinations are known as chords and provide a fairly easy way of adding harmony to a melody. A chord is generally accepted to be a combination of three or more notes which means we will not be able to produce very complicated harmonies on the Amstrad.
A chord is built up by a sequence of intervals much like a scale, except that the notes in a chord can be played together without sounding too unmusical. The most common chord is the major chord which is built up from a root note from which the chord takes its name.
To construct a C major chord we add an interval of a third and an interval of a fifth to the C note. These intervals are reckoned in note names, counted from the root note and in the key of the root note. To arrive at an interval of a third in the key of C, begin on C, count that as one and move up the scale until you reach three. This is E. The interval of a fifth is found in the same way, again counting from the root note. This will bring you to G. Work this out on the keyboard in Figure 2.4.
To produce a minor chord, the third interval is flattened, i.e. taken down a semitone. This produces a 'sad' sound as opposed to the quite bright effect of a major chord.
Many chords are named according to their construction and names are given in terms of flattened, augmented (sharpened) and added intervals. Some examples are given in *Figure 2.11*.



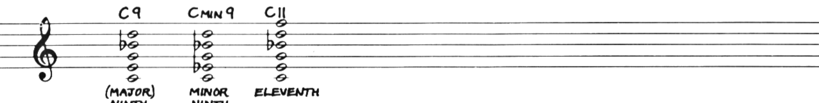


Figure 2.11:Chords and Scales.

Chords are useful for all manner of musical things and we will be experimenting with them later in the book. Meanwhile, if you want to hear what different chords sound like, try the following program:

```
100 REM PROGRAM 2.1
110 REM Chord Sound Demo
120 :
130 MODE 2:PRINT:PRINT:PRINT
140 :
150 FOR chord=1 TO 4
160 READ chord$,notes$,pitch1,pitch2
170 PRINT"C ";chord$;" = C + ";notes$
180 SOUND 1,478,300
190 SOUND 2,pitch1,300
200 SOUND 4,pitch2,300
210 FOR d=1 TO 3000:NEXT d
220 NEXT chord
230 :
240 REM Chords with 4 Notes
250 FOR arp=1 TO 5
```

```
260 READ chord$,notes$
270 PRINT"C ";chord$;" (Arpeggio) = C +
";notes$
280 FOR note=1 TO 17
290 READ pitch
300 SOUND 1,pitch,10
310 NEXT note
320 FOR d=1 TO 1000:NEXT d
330 NEXT arp
340 END
350 :
360 DATA Major,E + G,379,319
370 DATA Minor,Eb + G,402,319
380 DATA Augmented,E + G#,379,301
390 DATA Suspended 4th,F + G,358,319
400 DATA Diminished, Eb + Gb + A,478,402
,338,284,239,201,169,142,119,142,169,201
,239,284,338,402,478
410 DATA 7th,E + G + Bb,478,379,319,268,
239,190,159,134,119,134,159,190,239,268,
319,379,478
420 DATA Major 7th,E + G + B,478,379,319
,253,239,190,159,127,119,127,159,190,239
,253,319,379,478
430 DATA Major 6th,E + G + A,478,379,319
,284,239,190,159,142,119,142,159,190,239
,284,319,379,478
440 DATA Minor 6th,Eb + G + A,478,402,31
9,284,239,201,159,142,119,142,159,201,23
9,284,319,402,478
```

As we cannot sound more than three channels together, chords containing more than three notes are played as arpeggios and carry with them an implied harmony. An arpeggio is when the notes of a chord are played in rapid succession as opposed to all at once.

There are many other types of chords but the basic construction principles remain the same. Now, armed with the knowledge of these last two chapters, we will see how it relates to the Amstrad computer and the SOUND command.

CHAPTER 3

The SOUND Command

When BASIC comes across a SOUND command, the note information it contains is sent to the sound chip. The ENV and ENT commands merely alter the volume and pitch characteristics of the note and have no part in its immediate production.
In its simplest form, SOUND need only be followed by the first two parameters but it can take up to seven. We can experiment with the SOUND command, adding extra parameters only when required. You need to try to absorb all the information in this chapter at once. Read it through and refer back to it when necessary. Demonstration programs are included and we'll be using the commands and parameters discussed here in programs throughout the book.
The User Guide lists the SOUND parameters using a series of brackets to indicate which parameters are optional. I have omitted the brackets for clarity.

SOUND channel status,tone period,duration,volume,volume envelope,tone envelope,noise period

The parameters are also given arbitrary single-letter names in the User Guide:

SOUND G,H,I,J,K,L,M

As these are not mnemonic I would suggest that the following is easier to remember:

SOUND C,P,D,V,VE,TE,N

where the letters stand for channel, pitch, duration, volume, volume envelope, tone envelope and noise.
I feel that 'pitch' is easier to understand than 'tone period' which suggests, to me as a musician, a note duration. Hence the use of P for the second parameter. Likewise, 'tone envelope' suggests that it might affect the

waveform in some way when in reality it modulates the pitch. In an attempt to minimise any possible confusion I will normally refer to the second SOUND parameter as pitch and most variables holding pitch information will be called pitch or note. The tone envelope, however, I will continue to refer to as the tone envelope. Its use is far less likely to cause confusion and it can be expressed in a variable as TE or ET.
When a SOUND command is issued, it places the note in a queue in a sound buffer and immediately passes control back to BASIC. Each channel has its own buffer which can hold four notes while a fifth note is playing. If a buffer is full the SOUND command will be unable to process its instruction and the program will halt until a space appears in the queue.
For reference, *Figure 3.1* lists the SOUND parameters along with the range of values they can take. Some of the values set special conditions so we'll look at each one in turn.

Figure 3.1: SOUND parameters and value ranges

Parameter	Name	Range	Default
C	Channel	1 to 255	none
P	Pitch	0 to 4095	none
D	Duration	−32768 to 32767	20
V	Volume	0 to 15 (see text)	4 with no ENV
VE	Volume Envelope	0 to 15	0
TE	Tone Envelope	0 to 15	0
N	Noise	0 to 31	0

Channel Status (C)

This takes a value in the range 1 to 255. There is no default, a value must be included. The sound chip contains three sound channels and this parameter selects which channel or channels are to be used to produce the sound. To specify a channel we don't use 1, 2 and 3 as you might expect but rather 1, 2 and 4. (The Guide refers to the channels as A, B and C which avoids the numbering problem.) This is because this parameter is bit significant which means the computer looks at the value as a binary number and the instructions it processes depend upon whether the bits in the number are set (on) or reset (off). The overall decimal value has little meaning. A brief look at the binary system should make this easier to understand.
As the binary system is at the root of all computer operations it is worthwhile becoming familiar with its use and Appendix II in the User Guide delves into this side of computing. As Amstrad BASIC is generally quite friendly, however, you will only rarely need to get down to programming at bit level.

Some sound-associated commands, however, such as SQ, use bit significant values.
Returning to the Channel Status parameter, the meaning of the overall value depends upon the bit settings. These are shown in *Figure 3.2*. To send a sound

Figure 3.2: Channel status — the bit setting explained

Bit	Decimal	Command
0 LSB	1	send sound to channel A
1	2	send sound to channel B
2	4	send sound to channel C
3	8	rendezvous with channel A
4	16	rendezvous with channel B
5	32	rendezvous with channel C
6	64	hold
7 MSB	128	flush

to channels A and C, therefore, we would set bits 0 and 2 which is 00000101 in binary and 5 in decimal. You can use Figure 3.2 to help determine the correct decimal values until you become more familiar with the system. Alternatively, you can specify the numbers in binary by prefixing them with &X. To send middle C to channels A and C you could write:

SOUND &X101,478

Rendezvous

The rendezvous facility permits notes on different channels to synchronise or sound at exactly the same time. For example, to synchronise channels A and B a rendezvous must be set on A telling it to rendezvous with B and a rendezvous must be set on B telling it to rendezvous with A. If no rendezvous is set, notes on each channel will sound as soon as they reach the front of their respective queues. Rendezvous are set using bits 3, 4 and 5, for channels A, B and C. Look at the next program, and try to see how the system works.

```
100 REM PROGRAM 3.1
110 REM Rendezvous Demo
120 :
130 REM Un-Rendezvoused
140 SOUND 1,478,200
150 FOR d=1 TO 1000:NEXT d
160 SOUND 2,379,200
```

```
170 SOUND 4,319,200
180 :
190 GOSUB 410
200 :
210 REM Rendezvoused
220 REM A with B & C
230 REM B with A & C
240 REM C with A & B
250 SOUND 49,239,200
260 FOR d=1 TO 1000:NEXT d
270 SOUND 42,190,200
280 SOUND 28,159,200
290 :
300 GOSUB 410
310 :
320 REM Rendezvoused - A & B & C
330 rend=56
340 SOUND 1+rend,119,200
350 FOR d=1 TO 1000:NEXT d
360 SOUND 2+rend,95,200
370 SOUND 4+rend,80,200
380 END
390 :
400 REM Wait for a key
410 PRINT"Press a key"
420 k$="":WHILE k$="":k$=INKEY$:WEND
430 PRINT"Okay"
440 RETURN
```

If BASIC is processing data then a small delay will be introduced between sound commands. Line 150 introduces a delay and illustrates how channels can be thrown out of sync. The second example uses rendezvous to keep the channels together. Each channel has been correctly synchronised with the others but this results in different channel status values for each channel which could be confusing to program. Fortunately, if we want to synchronise all three channels we can issue a blanket rendezvous command, 56 or &X111000, which means synchronise all three channels.
If more than one channel is specified in C then rendezvous is automatically implemented, e.g. if C was set to 3.
When issuing a string of sound commands we don't usually want every note to synchronise, because one channel will probably be playing several notes in the same time as another channel plays one or two. In any event, it's not

always necessary to rendezvous all the notes. If we rendezvous the first notes then the following notes should play in time, providing BASIC is not interrupted for too long. We'll look at this more closely in Chapter 9.

Hold and the RELEASE command

A hold is set with the sixth bit. It does just as it implies and holds the sound in the queue, halting the playing of any commands. The hold can be released by flushing the channel(s) or by issuing a RELEASE command.
Hold can be used to start all the channels playing together as an alternative to an initial rendezvous setting. To see how it works we'll consider an example on just one channel. Enter this program exactly as it is.

```
100 REM PROGRAM 3.2
110 REM Hold & RELEASE Demo
120 :
130 REM ON BREAK GOSUB 230
140 :
150 FOR scale=1 TO 8
160 IF scale=1 THEN status=65 ELSE statu
s=1
170 READ note
180 SOUND status,note
190 NEXT scale
200 END
210 DATA 478,426,379,358,319,284,253,239
220 :
230 ON BREAK STOP
240 PRINT scale
250 PRINT"Press a key to RELEASE"
260 k$=INKEY$:IF k$="" GOTO 260
270 RELEASE 1
280 RETURN
```

Commentary

The program plays, or attempts to play, a simple scale. Line 160 sets the hold parameter only on the first note of the scale. When run, nothing will appear to have happened. Press ESC twice and you will see

Break in 180

If you enter

PRINT scale

5 will be printed showing that the program successfully issued four sound commands before coming to a halt at line 180 while trying to issue a fifth.
Delete the REM line 130. Run it again and press ESC twice. This will take the program to line 230. Line 240 prints the value of scale which will be 5 as before. Press a key and the program will issue the RELEASE 1 command at line 270 which releases the hold on channel 1. This frees the four frozen notes in the sound queue which immediately play. The program returns via line 280 to line 180 and plays out the remainder of the scale.
The RELEASE command is bit significant: 1 for channel A, 2 for channel B, 4 for channel C and 7 for all three. The release of a channel not held has no effect.
In practical terms, to set all channels playing together, a hold would be issued for all channels by:

SOUND 71,0,1,0

and three sound commands would be issued for each channel. RELEASE 7 would free all channels at once so they would all start playing at the same time. Control would then pass to the main program which would keep the sound channels filled.

Flush

If bit 7 is set to 1 when the SOUND statement in which it occurs is executed, it flushes the sound buffer of any notes waiting in the queue and stops execution of whatever note may be sounding at that time. The SOUND command then executes its note. This can be thought of as jumping the queue and has several useful applications.
In a game which plays a background tune or which creates sound effects as objects move, at any point in the program the music can be interrupted and a different tune played. Explosions can be made to occur exactly at the time an object is hit.
Using Program 3.2, we can illustrate the flush command by altering line 270 to:

270 SOUND 129,0,1,0

With the line 130 unREMed (delete the REM statement), run the program

again. This time when the program enters the ON BREAK routine the channel will be flushed throwing out the four notes in the buffer. Control passes back to line 180 and the last four notes will play.
In a musical context, for example, when priming the sound channels with a hold command as above you could also include a flush command in case any unplayed sounds are in the buffer:

```
SOUND 199,0,1,0
```

The intricate nature of the first parameter of the SOUND command demonstrates the versatility of the Amstrad's sound production system. The other parameters are generally less complicated.

Pitch (P)

This must take a value in the range 0 to 4095 — there is no default. It determines the pitch or frequency of the sound. *Figure 2.4* and Appendix 1 list the pitch numbers required to produce notes in the conventional western scale. You will realise, of course, that the range of values allows us to produce pitches outside the western scale. We'll see exactly how to calculate pitch numbers in the next chapter.
The User Guide says if P is set to 0 then no frequency is set. In fact, it produces a slight click. Try this:

```
10 FOR pitch=10 TO 0 STEP -1
20 SOUND 1,pitch
30 NEXT pitch
```

It is suggested that a value of 0 is used when only noise is required and in this case the noise is likely to cover up the click.

Duration (D)

This takes a value in the range -32768 to 32767. If omitted it defaults to 20. If greater than 0 it sets the length of time the note is to sound in 1/100ths of a second. If equal to 0, the duration is governed by the length of the volume envelope specified. If no volume envelope is specified when D=0:

```
SOUND 1,478,0
```

then the duration is 200, the duration of the default envelope. When D is less than 0, the positive or absolute (ABS) value of D stipulates the number of

times the volume envelope will repeat. For example, these two sounds have the same duration:

```
SOUND 1,478,400
SOUND 1,478,-2
```

The repeating enveloping is the default envelope with a built-in duration of 200. This can be put to good use in special effects, especially when more complicated envelopes are used.

Volume (V)

This takes a value in the range 0 to 15. If no ENV is specified then the value range effectively runs from 0 to 7; the values 8 to 15 in this case being a duplication of 0 to 7, so that 8 is the same as 0, 9 is equal to 1, and so on. If an ENV is specified then the volume can be set to any one of the 16 values. 0 is off. V defaults to 4 when no ENV is specified. The Guide states that it also defaults to 12 if an ENV is specified but if an ENV is specified then V must be given a value as ENV is set after V in the parameter string. What it probably intended to say was a volume of 4 with no ENV is the same as a volume of 12 with an ENV. V is the initial volume and can be altered by an ENV.
You can test these values by running this:

```
10 FOR vol=0 to 15
20 SOUND 1,478,50,vol
30 NEXT vol
```

and then this:

```
 5 ENV 1,1,15,1
10 FOR vol=0 to 15
20 SOUND 1,478,50,vol,1
30 NEXT vol
```

You will see that using an envelope gives us twice the number of volume levels although the maximum volume is the same.

Volume Envelope (VE)

This takes a value in the range 0 to 15. 0 is the default value. VE specifies the number of the volume envelope (ENV) which will control the volume of the sound. The ENV must previously have been defined. If an ENV is specified

which has not been defined then the default of 0 is used. ENV 0 can not be redefined by the user and plays the note at the volume set by the V parameter.
The volume of a sound is affected by many factors as we saw in Chapter 1. If the amplitude of a waveform is doubled, however, the sound does not appear to be twice as loud. This is because we perceive sound in a logarithmic fashion. If vibrato or tremolo is applied to a note it will seem louder, and volume varies with pitch, too, so that low notes need more power to sound as loud as higher notes.

Tone Envelope (TE)

This takes a value from 0 to 15. 0 is the default value. TE specifies the number of the tone envelope (ENT) which controls the pitch of the sound. The ENT must previously have been defined. If an ENT is specified which has not been defined then the default of 0 is used. ENT 0 can not be redefined by the user. It is set to play an unmodulated pitch as specified by P in the SOUND command, in other words, it has no effect.

Noise (N)

This takes a value from 0 to 31. In Chapter 6 Page 8 the User Guide erroneously states the range as being from 0 to 15. The correct range is given on Page F3.20. The default is 0 which is no noise at all.
You can hear the range of sounds available from this parameter by running this:

```
10 FOR n+1 to 31
20 SOUND 1,0,200,7,0,0,n
30 NEXT n
```

It can be used to produce all manner of sounds from explosions to rhythm patterns.
The sound generation hardware has only one noise setting. If different values of N are set on more than one channel then each new value of N overrides the previous one. All channels with an active N parameter will produce noise at the same frequency.
Noise deserves a section to itself and we look at it more closely in Chapter 7.

Putting them all together

All SOUND parameters are expected in integers. If they are given as real

numbers they are rounded in a similar way to the ROUND function:

SOUND 1,16.4999

produces the same note as:

SOUND 1,16

and

SOUND 1,16.5

is the same as

SOUND 1,17

The state of the sound queue (SQ(C))

After issuing a number of complex sound commands it can be difficult to ascertain exactly what each channel is doing at a particular time. The SQ function allows us to interrogate a channel to find out how many free entries are in the queue, whether the channel is active (playing) and if not, why the entry at the head of the queue is waiting.
The c parameter in the SQ(c) function is bit significant: 1 specifies channel A, 2 is channel B and 4 is channel C. Other values will not be accepted. It returns an integer which, again, is bit significant. To extract the information we must look at the value as a binary number. We can do this by using BIN$ which prints a string of binary digits representing the form of a number (see Chapter 8 Page 4 of the User Guide), e.g.:

PRINT BIN$(SQ(1))

The bits have the following significance:

Bits 0 to 2: number of free spaces in the queue (0 to 4)
Bit 3: rendezvous with channel A
Bit 4: rendezvous with channel B
Bit 5: rendezvous with channel C
Bit 6: hold is set
Bit 7: channel currently playing

The SQ interrogation only takes place on the sound at the head of the queue.

Interestingly enough, if you try to rendezvous a channel with itself, e.g. SOUND 9 or SOUND 56 for channel A then this rendezvous doesn't show. If a hold and a rendezvous are both present, then only the hold bit, bit 6, will show as set. The rendezvous bits and bit 6 and bit 7 are mutually exclusive. The next program illustrates some possible results.

```
100 REM PROGRAM 3.3
110 REM SQ Demo
120 :
130 rend=56
140 SOUND rend+1,478,200
150 GOSUB 210
160 SOUND rend+2,379,200
170 GOSUB 210
180 SOUND rend+4,319,200
190 GOSUB 210
200 END
210 PRINT "A...";BIN$(SQ(1),8)
220 PRINT "B...";BIN$(SQ(2),8)
230 PRINT "C...";BIN$(SQ(4),8)
240 PRINT
250 RETURN
```

When run, the states of the sound channels as revealed by the SQ function are shown. Notice that the last rendezvous command in channel C does not show because as soon as it is issued all channels become active and play. To see the effect of a hold alter 180 and add line 195:

```
180 SOUND rend+4+64,319,200
195 RELEASE 4:GOSUB 210
```

When an SQ function is issued it disables any ON SQ GOSUB interrupt set for that channel. We look at this now.
Amstrad BASIC helpfully provides the programmer with a system of interrupts. These are explained in the User Guide Chapter 10 Page 1. They enable us to perform a number of operations simultaneously, a facility often referred to as multitasking. You may already be familiar with the AFTER and EVERY commands which activate normal programming interrupts. The ON SQ GOSUB is an interrupt designed specially for sound production.
The c parameter in ON SQ(c) GOSUB is bit significant as usual and takes a value of 1, 2 or 4. When the command is issued it will automatically GOSUB to the specified line number if there is a space in the sound queue of that

particular channel. It ensures that the channel is kept supplied with notes.
There are four interrupt timers — 0 to 3 — with different interrupt priorities. 3 has the highest priority and 0 the lowest. The sound queues have independent interrupts all of equal priority which is the same as timer 2. Once a sound interrupt routine has started, therefore, it will not be interrupted by other sound routines.
The action of interrupting disables the event and to rearm the interrupt the ON SQ GOSUB call must be issued again from within the subroutine. As mentioned above using the SQ function will disable the interrupt as will issuing a SOUND command for that channel.
Interrupts, and the SQ interrupts in particular, are powerful programming aids. Once set in motion they automatically take care of the sound production housekeeping so BASIC can process other instructions at the same time. They simplify multi-channel programming and we will be making use of them later in the book.

Stereo sound and external speakers

You can give an added boost to your sound and music programs by using the stereo capabilities of the Amstrad. To do this, you must connect the I/O port at the back of the computer to a hi-fi system or pair of stereo headphones. Channel A plays from one side, channel C plays from the other and channel B plays from both and will appear to come from the centre.
To plug into the I/O socket you need a mini-stereo jack plug. Headphones which are intended for use with personal walkman cassette players often have this type of plug. Alternatively, you can buy a variety of adapters which will allow you to plug into most music systems, e.g. a mini stereo jack plug on one end and a normal stereo jack socket on the other. These are usually available from electronic component suppliers.
Playing the output through external speakers has two advantages: stereo sound and an increase in sound quality. The Amstrad's internal speaker is really too small to do justice to the sound output and if the volume is turned up too high or if all channels are playing low or loud notes then it will rattle and the sound will distort. Even if you cannot route the sound through a stereo system, at least try to listen to it through an external speaker so you can compare the quality.
Most of the programs in this book which produce multi-channel music have been arranged to make use of the stereo effect by playing different sections through opposite channels. Limited though the stereo positioning may be, it really does add another dimension to the sound.
Panning is the act of moving and positioning a sound within the stereo field. A very impressive effect is to pan a note from one speaker to the other. We can

pan notes on the Amstrad but we need two channels to do it. Here's a simple demonstration.

```
10 ENV 1,1,15,1,10
20 ENV 2,1,15,-1,10
30 SOUND 1,956,0,0,1
40 SOUND 4,956,0,15,2
```

Alter the last parameter, the pause time, in the envelope commands (envelopes are fully explained in Chapter 5). Experiment with the volume levels and envelopes themselves, too, and see what stereo effects you can produce. Panning could be useful for sending a bomb or missile whizzing from one side of the screen to the other.
Versatile though the SOUND command is on its own, the range of sounds it can produce can be expanded enormously with ENV and ENT which are examined in detail in Chapter 5.

CHAPTER 4

Programming Scales and Pitches

The sound chip in the Amstrad can produce frequencies ranging from 31 to well over 10000 Hertz. This covers a wide spectrum of sounds from a series of clicks at the lower end of the range up to notes even a piccolo can't produce. We can use this range to good effect and we'll be exploring the potential throughout the book.

We can program the sound chip to produce scales other than the one we normally associate with western music. Some scales can only be produced by a computer or a very expensive synthesiser and we'll take a look at these, too. First, we'll see where the pitch numbers come from.

Calculating the pitch number

For convenience, a list of pitch numbers is given in Appendix 1. These cover the notes of our western scale over the eight octave range of the sound chip. We can simply use these values without more ado but for the curious and the more adventurous, we'll delve a little deeper into where these numbers come from.

If we know the frequency of the sound we require in Hertz, we can calculate the pitch number using the following formula:

pitch=ROUND(125000/Hz)

There's nothing special about this formula from the music point of view, it's just the way the Amstrad's sound system works. It expresses the relationship between the number we put in the SOUND command (pitch) and the frequency (Hertz) which it plays. The next program gives an audible demonstration of this pitch formula. It requests an input in Hertz, prints out the pitch number and plays the sound.

```
100 REM PROGRAM 4.1
110 REM Hertz To Pitch Conversion
120 :
130 CLS
```

```
140 WHILE -1
150 INPUT "Hertz";hz
160 pitch=ROUND(125000/hz)
170 PRINT"Pitch =";pitch:PRINT
180 SOUND 1,pitch,100
190 WEND
```

Check the pitch numbers against those given in the appendix. To use these frequencies in a melodic context we must be able to produce specific pitches which have a musical relationship with each other. The most obvious example is our western scale. There are scales other than this and it is interesting to see how the western scale we now use came into being.

The scale of equal temperament

The notes we normally use in western music make up what is known as the equal tempered scale. This consists of octaves which are split into 12 equal intervals called tempered half tones — these are our normal semitones. The reason why the scale is called equal tempered is because, believe it or not, the notes have been squeezed out of their more natural harmonic relationship to each other. Temperament is the process of reducing the number of tones per octave by altering the frequency of the tones from their more natural frequencies.

This talk of natural frequencies may sound a bit strange and what's this about 'reducing the number of tones per octave'? Surely there are 13 tones in an octave. Well, there are now but once there were more, 17 to be exact. Tones, that is, not notes.

The scale of just intonation

The most pleasing, that is harmonious, combination of tones is produced when frequencies have a simple mathematical relationship to each other. By simple I mean ratios such as 8:5 and 5:4, not 1.189207:1.

We can construct a scale based upon simple frequency intervals — ratios known as the harmonic series — and this will produce a scale of just intonation. Intonation is the process of selecting the tones of a scale with respect to frequency. A scale based on the harmonic series will produce notes which are more pleasing to the ear than those produced by a scale of equal temperament. It would seem natural, therefore, to use the most harmonious tones — the scale of just intonation.

Using frequency intervals based on the harmonic series produces three types of intervals — major tones, minor tones and semitones. (The names just

distinguish one interval from another. In the scale of equal temperament a semitone *is* a minor tone.)
To help clarify this a little, *Figure 4.1* shows the ratios between the intervals in the scale of just intonation. It also shows the pitches which would be used in a scale of C major. The last column shows the ratio between adjacent notes of the scale: a major tone has a ratio of 9:8,a minor tone a ratio of 10:9 and a semitone a ratio of 16:15. So, for example, the ratio between C and D is 9:8 and between D and E, 10:9.

INTERVAL	RATIO FROM UNISON	NOTE	RATIO BETWEEN ADJACENT NOTES
Unison	1:1	C	9:8
Semitone	16:15		
Minor tone	10:9		
Major tone	9:8	D	10:9
Minor third	6:5		
Major third	5:4	E	16:15
Perfect fourth	4:3	F	9:8
Augmented fourth	45:32		
Diminished fifth	64:45		
Perfect fifth	3:2	G	10:9
Minor sixth	8:5		
Major sixth	5:3	A	9:8
Harmonic minor seventh	7:4		
Grave minor seventh	16:9		
Minor seventh	9:5		
Major seventh	15:8	B	16:15
Octave	2:1	C	

Figure 4.1: Ratios between Intervals in Scale of Just Intonation

The out-of-tune scale

With a little study I hope you can see that using these ratios we cannot construct any scale other than that of C. Try the scale of D. The first interval we want is a major tone (D to E) which is expressed as a ratio of 9:8. We can see from the figure that the difference between D and E is 10:9, a minor tone. (This also leads to the situation where C# is a different pitch to D flat.)
Once you grasp that, you'll realise that in order to stay in tune instruments would be restricted to playing in one key. To play in another key a separate set of frequencies, based on the above ratios, would have to be used.

Although very harmonious, a more flexible method was required and so, quite simply, the octave was split into 12 equal intervals — the scale of equal temperament which we use today. J. S. Bach wrote a set of pieces called 'The Well Tempered Clavier' which used the scale of equal temperament — he clearly saw a future for it. For comparison, *Figure 4.2* shows the ratios between the intervals in the scale of equal temperament.

INTERVAL	RATIO FROM UNISON	NOTE
Unison	1:1	C
Semitone/minor second	1.059463:1	
Whole tone/major second	1.122462:1	D
Minor third	1.189207:1	
Major third	1.259921:1	E
Perfect fourth	1.334840:1	F
Augmented fourth / Diminished fifth	1.414214:1	
Perfect fifth	1.498307:1	G
Minor sixth	1.587401:1	
Major sixth	1.681793:1	A
Minor seventh	1.781797:1	
Major seventh	1.887749:1	B
Octave	2:1	C

Figure 4.2: Ratios between Intervals in Scale of Equal Temperament.

If you want to experiment with the scale of just intonation here are the frequencies in Hertz of the scale of C in octave 0:

Note	Hertz
C0	264
D0	297
E0	330
F0	352
G0	396
A0	440
B0	495
C1	528

You can calculate other keys by using the ratios in Figure 4.1. For example, in the key of D, to arrive at the frequency of the second note, E, a major tone is

required so divide by 8 and multiply by 9.

297/8*9 = 334.125

You can see how this differs from the frequency of E in the scale of C. For completeness, here are the base frequencies of the other notes.

Key	Base FQ
C#0	278.4
D*b*0	278.1
Eb0	312.9
F#0	371.2
G*b*0	370.8
A*b*0	417.2
B*b*0	469.3
C*b*0	494.4

If all this seems a little confusing, don't worry. Throughout the book we will be producing music based on the equal tempered scale — the normal one — but as your knowledge of music increases you might like to come back to the concept of scales with different intervals. It's fascinating topic with plenty scope for exploration.

Microtonal scales

As the equal tempered scale divides the octave into 12 equal parts, we could, for example divide it into 13 parts, or 14 parts, or any other number of parts we wish. Tunings that divide an octave into more than 12 parts are known as microtonal. As the Amstrad can produce a wide range of frequencies this is easy to do. A more challenging task is to write music to use these tunings. We'll come back to microtonal scales in a moment.

Calculating notes in the equal-tempered scale

We can calculate the notes in the western scale using the following formula:

```
freq=440*(2^(oct+(note-10)/12))
pitch=ROUND(125000/freq)
```

where freq is the frequency in Hertz, oct is the octave number and note is the note number, e.g. 1=C, 2=C#, d=D etc. This is how the figures in Appendix 1 were arrived at. Note that the FREQENCY formula on Page 3 of Appendix

VII in the User Guide uses the expression (10–N) instead of (N–10).
You can see from the formulae that the notes are first converted into Hertz and the frequency is calculated from International A. Even if you are not a musician in the strictest sense of the word you may have heard of this international tuning standard which stipulates a frequency of 440 Hz for A above middle C. This ensures that all instruments wherever they may be will sound the same pitch when they play the same note.
The formula is based on the fact that the frequency of each note in an octave is twice that of the note in the octave below it. You will also have noticed that as the frequency doubles, the pitch number halves. It seems rather contrary but as long as we remember this it shouldn't cause any problems. The next program lets us experiment with the formula.

```
100 REM PROGRAM 4.2
110 REM Octave and Note
120 REM To Pitch Conversion
130 :
140 WHILE -1
150 INPUT "Octave";oct
160 INPUT "Note";note
170 freq=440*(2^(oct+(note-10)/12))
180 pitch=ROUND(125000/freq)
190 PRINT"Freq =";freq,"Pitch =";pitch
200 PRINT
210 SOUND 1,pitch,100
220 WEND
```

Alternative octave and note numbers

The User Guide calls the octave beginning with middle C octave 0. Lower octaves, naturally, take minus numbers. To arrive at the frqeuency, we could base our calculations on a frequency other than International A. Programs 1.2 and 1.4 use middle C.
For the sake of conformity, in the larger programs in the book I I have adhered to the User Guide's notation but I'll list here some formulae so you can use an alternative system if you prefer. Use them in place of the formula in line 170 in the last program.
If you want to use positive octave numbers, with the lowest octave numbered 1, as listed at the bottom of Figure 2.4, you could use:

freq=15.43375*(2^(oct+note/12))

Using the above base frequency, B five octaves below middle C, we don't need to subtract an offset from note. A note value of 1 still represents C. If you want to use the note numbers given in Appendix 1 then use this:

freq=30.8675*(2^(note/12))

As the note name (or number) doesn't need an octave we can omit oct from the formula. Although this is shorter, note data written as a list of numbers may be more difficult to understand. We could, of course, enter notes as note name plus octave number and instead of reducing this to a note and octave value, reduce it to a single note value. Musically, however, it's preferable to know the note name and the octave it belongs to so, again, I haven't used this method in any programs but please experiment.

Constructing microtonal scales

By varying the above formulae only slightly we can produce microtonal scales. Replace the 12 by the number of parts you want to split the octave into. It's as simple as that on the Amstrad, as the next program demonstrates.

```
100 REM PROGRAM 4.3
110 REM Microtonal Scales
120 :
130 oct=0
140 WHILE -1
150 INPUT"How many intervals";intv
160 PRINT
170 FOR note=0 TO intv
180 freq=261.262*2^(oct+note/intv)
190 pitch=ROUND(125000/freq)
200 PRINT"Freq =";freq,"Pitch =";pitch
210 SOUND 1,pitch
220 NEXT
230 WEND
```

You can enter values less than 12, too, which will produce macrotonal scales. Javanese, Siamese and Arabian cultures use octaves with five or seven intervals.

Note that the base frequency is that of middle C. Using 440 would produce a scale starting on International A. It's more usual to begin a scale on C but the relationship between the intervals would be exactly the same. We can still use oct to change octaves if required.

Try the program. With intervals of between 12 and 20 the scale will probably just sound like a sequence of semitones — unless you have an excellent sense of pitch. The real test begins when you try combining intervals to produce chords (see Chapter 2) and harmonies. That would really take another book so, hopefully having whetted your appetite, I'll leave you to experiment.

The shape of music to come

When we speak of music we usually mean a sequence of notes and harmonies from the equal tempered scale. We have seen that other scales exist and how easy it is to produce them on the Amstrad.

Throughout the book we will concentrate on 'normal' music and conventional notation. This will produce sounds which most people will identify as music. After all, it has proved to be the most successful and common method of musical expression yet invented.

Many people, musicians included, are not aware that other scales exist. Details of these scales have been given because, as music continues to develop and the boundaries of acceptability are pushed back, other scales, microtonal ones in particular, will become more popular.

The Amstrad provides us with a unique opportunity to experiment with these scales and to produce music which surpasses the limits of accepted tonality. The rest is up to you, your imagination and ingenuity.

CHAPTER 5

ENV and ENT — the Volume and Tone Envelopes

The ENV and ENT commands may appear, at first sight, quite complex. The fact that they can take up to 16 parameters and must be used in conjunction with the SOUND command may seem to support that view, but they can be broken down into manageable sections which are easy to understand. They can also take two forms: a software envelope and a hardware envelope. We'll examine ENV first.

The volume envelope

The purpose of ENV is to alter the volume of a note during its production. We saw how this can affect a sound in Chapter 1 and Program 1.4 gave a brief ENV demonstration. The initial volume on which the envelope works is determined by the value of V in the SOUND command.

The volume of a sound can vary enormously during its production. There could be six, seven, eight or more different stages and volume levels. Most commercial synthesisers, however, offer control over only four stages called Attack, Decay, Sustain and Release, usually abbreviated to ADSR. You may have heard this term. The fact that ADSR only give us four parameters or volume changes does not impose great restrictions because the volume envelope of most sounds and instruments can be accurately described using just these parameters.

The ENV command does not have ADSR settings as such: its control over amplitude is far wider but, especially when attempting to create instrument sounds, you will find it convenient to think in ADSR terms.

Although ADSR is most commonly used to describe instrument characteristics, the favourite example used to explain it is that of a car approaching us along a straight road. We hear it very quietly at first and it gradually becomes louder until it draws level with us at which point it is as loud as it is going to get. The volume then immediately begins to decrease. If it stops a little further on for the driver to ask directions, the engine volume will remain constant. When it drives off again the volume will gradually fade to nothing.

If we plot the volume against time, the resulting graph might well look like *Figure 5.1*.

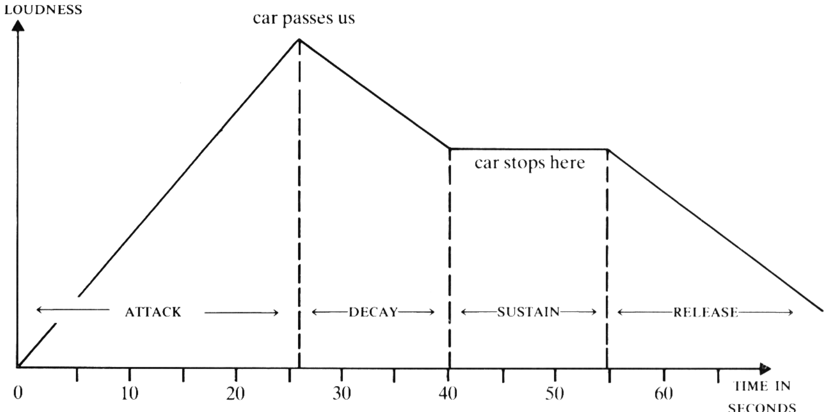


Figure 5.1: The ADSR Principle.

This is obviously a very coarse and long (in terms of time) example, but the principle behind the volume variations involved are exactly the same as those which occur when an instrument produces a note. An instrument envelope, however, usually lasts for one or two seconds, often less.
For convenience, the loudness contour is divided into phases called attack, decay, sustain and release. Some sounds have more phases and we can use ENV to produce some very complex envelopes. First, we'll see what the individual ADSR phases encompass.

The attack phase

This is when the sound first begins and refers to the length of time required for the sound to reach a particular, usually maximum, volume. In the case of the car, the attack phase is long and builds up slowly. String instruments have a slow attack phase which might be around a tenth of a second. In contrast, percussion instruments such as the piano and drums, and plucked string instruments like the guitar, have a fast attack phase and their sound reaches maximum volume immediately upon playing. Their attack phase might be as short as one hundredth of a second. Brass and woodwind have an attack rate somewhere between these extremes.
If you have ever played a synthesiser, the attack is the sound you hear when you first hit a key.

The decay phase

This is what happens immediately after the attack phase and is the length of time taken for the volume to reach a second, usually lower level — the sustain level. As its name implies, the sound usually decays or drops in volume during this phase. Notes from musical instruments tend to reduce in volume: some will stay the same, but this will only happen if the sustain level is the same as the volume reached at the end of the attack phase. In this case there will be no decay phase and the envelope will pass straight into the sustain phase.

The decay phase is often longer than the attack phase because whatever is producing the sound, be it a trumpet or a tin can, will be resonating or oscillating and the vibrations cannot usually be stopped dead. A useful analogy, again, is the car which can not pull to a halt without first slowing down — unless it hits a brick wall which takes us into another area of sound effects altogether. Instrumental sounds generally need at least a few hundredths of a second or so to dampen down. This helps to explain why a backwards recording of an instrument sounds so strange — it dies away too rapidly. Backwards recording is easily duplicated by electronic means and the ENV command makes it easy to produce on the Amstrad (see Program 1.4).

The sustain phase

After the decay phase, the sound enters the sustain phase. As its name implies, the volume is sustained at a constant level. The name is confusing because this is not really a phase or time period, but a volume level. It may help to think of the sustain phase as the volume level after the decay phase. For synthesiser players, the sustain phase is the volume at which the note continues to sound until you take your finger off the key.

The release phase

When the sound has gone through all the above phases it reaches the stage where it must eventually terminate. This is the release phase.

The vibraphone and many percussion instruments have long release times and fade away slowly. String instruments don't take quite so long but you can still hear the note hanging there a little before it dies completely. This phase is a measurement of time, like the attack and decay phases, and determines how quickly the sound finally fades away.

The previous remarks describing why decay times are usually longer than attack times apply here, too. The release phase, in fact, is what many people would refer to as the decay of a note. ADSR is just a convenient way of subdividing the volume envelope into manageable sections and it is unlikely that confusion will arise over terminology as the meaning will usually be clear

from the context.
On a synthesiser the release phase begins when you remove your finger from the key. If you immediately hit another key, the release will not sound and the attack phase of the next note will occur.

A complete ADSR envelope

The loudness contour of most instruments and many sounds can be defined quite accurately with an ADSR envelope.
You will probably have realised that not all sounds require all four phases. An electric organ for example has an attack, sustain and release phase but no decay. A woodblock has an attack and a quick release and a plucked string on a guitar or banjo has an attack and a somewhat slower release. Some sample envelopes are shown in *Figure 5.2*.
You may have realised, or at least you will when you run the next program, that it is not always possible to determine where one phase ends and another begins. Sometimes the decay phase will lead straight into the release phase and the resulting sound will be just one long decay — or release.
You can program sounds without a decay phase or a sustain phase if you wish. Some synthesisers have a simple envelope generator known as an AR generator which only allows the creation of an attack and a release phase. We will now see how the ENV command can be used to create ADSR and other envelopes.

The ENV parameters

We concentrate on the software definition first. The User Guide lists this as follows:

```
ENV N,P1,Q1,R1,P2,Q2,R2,P3,Q3,R3,P4,Q4,R4,P5,Q5,R5
```

where N is the envelope number. The P, Q and R names are not very helpeful, mnemonically, but as each group of P, Q and R parameters is the same, we only have three names to remember. We can refer to a P, Q and R group as an envelope section. The parameters and their ranges are summarised in *Figure 5.3*.
After defining an ENV, the computer will remember it until it is reset or until the ENV is redefined. An ENV can be cleared by entering the ENV number without any parameters.
Each P, Q and R section is complete in itself and up to five such sections can be defined. They each operate in exactly the same way. A complete section is mandatory but the number of sections is optional.

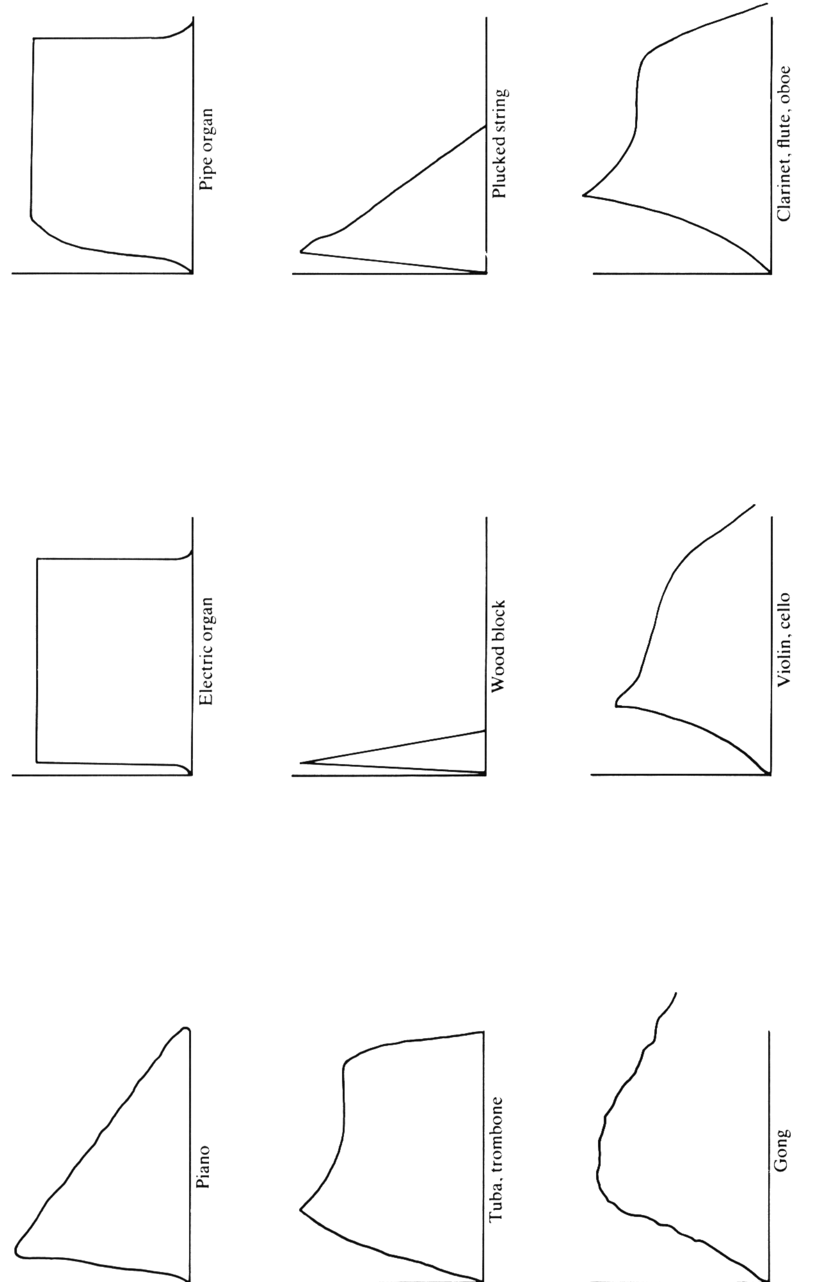


Figure 5.2: Sample Envelopes.

Figure 5.3: ENV parameters and value ranges

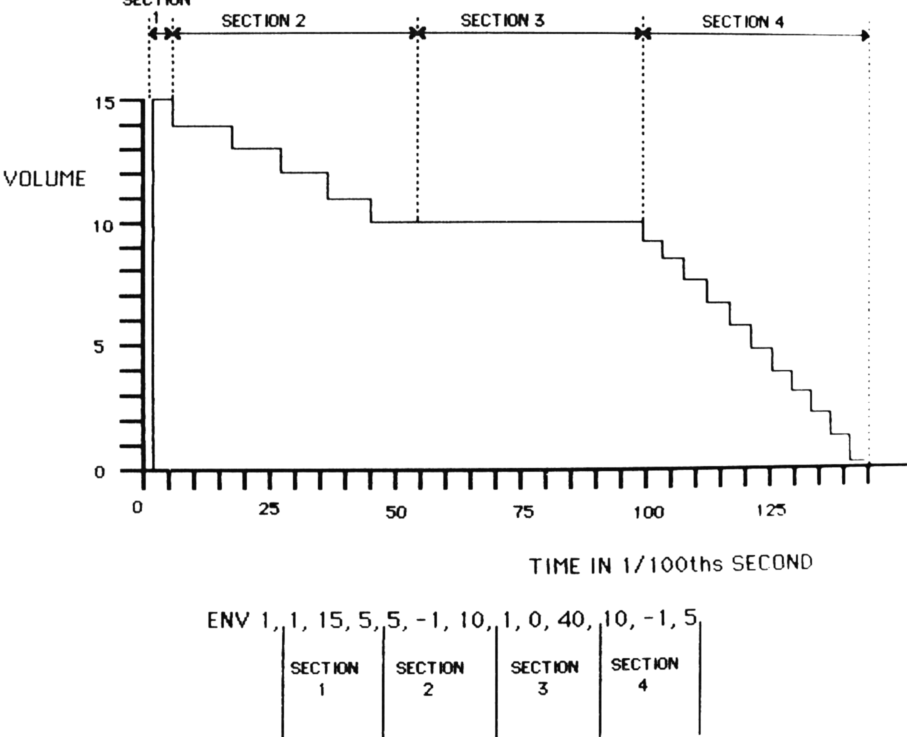
Parameter	Name	Range
N	Envelope Number	1 to 15
Software	ENV N,P,Q,R	
P	Step Count	0 to 127
Q	Step Size	−128 to 127
R	Pause Time	0 to 255
Hardware	ENV N,=HE,EP	
=HE	Hardware Envelope	0 to 15
EP	Envelope Period	−32768 to 65535

P specifies the number of steps the section will have, Q specifies the amount by which the volume will vary over each step and R specifies how long each step will take in one hundredth of a second.
When using a volume envelope with the SOUND command, the volume can be varied over 16 levels — 0 to 15. *Figure 5.4* shows how an ENV could be defined to produce a typical ADSR envelope. The V parameter of the SOUND command sets the initial volume level. When using ENV it is usually advisable to set V equal to 0. This puts the envelope in total control of the volume and if unexpected sounds occur they can't be attributed to spurious V values. You can see from the figure how the parameters affect the volume.

Special P, Q and R values

As with the SOUND command, certain values set special conditions. If P equals 0 then the resulting amplitude is the absolute value of Q which will last for the time specified by R.
The volume level will 'fold over' if Q takes it outside its range, i.e. a value of 16 produces a level of 0, 17 a level of 1, and so on. This can be used to create one-section envelopes as in Program 1.4.
If Q=0 then the volume level will stay at its present level for a duration equal to P×R as in section 3 in Figure 5.4.
If P and Q both equal 0 then the volume level will be 0 for R duration.
An R value of 0 is treated as 256.
If a positive value of D is given in the SOUND command and this is longer

Figure 5.4: Volume Envelope Producing a typical ADSR envelope



than the duration of the envelope, the remaining duration will sound at whatever volume the sound has reached. If D is less than the envelope duration, the excess envelope will be ignored.

An envelope graph generator

The best way to explore the envelope commands is to be able to hear and see what is happening as you read about it. The next program allows you to alter the envelope parameters and see and hear the resulting ENV. It also allows you to do the same with the tone envelope but that will lie dormant until activated so we can concentrate on the volume envelope before trying them both together. Enter it now and run it and we'll take a look at some ENV settings.

```
1000 REM PROGRAM 5.1
1010 REM ENV and ENT Graph Generator
1020 :
1030 MODE 1
1040 GOSUB 1270:REM Set Up
1050 GOSUB 1600:REM Draw Axis
1060 GOSUB 1750:REM Print ENV,ENT
1070 :
1080 WHILE -1
1090 com$=INKEY$:IF com$="" GOTO 1090
1100 com=ASC(LOWER$(com$))
1110 IF com=13 OR com=32 THEN GOSUB 2550
:GOTO 1090:REM Sound and Graph
1120 IF com=100 THEN GOSUB 1940:GOTO 109
0:REM Alter Note Duration
1130 IF com=112 THEN GOSUB 1990:GOTO 109
0:REM Alter Pitch
1140 IF com=121 THEN GOSUB 2040:GOTO 109
0:REM Alter Yscale of ENT Graph
1150 IF com=119 THEN GOSUB 1600:GOTO 109
0:REM CLG & Draw Axis
1160 IF com=105 THEN ON vort GOSUB 1520,
1550:REM Initial Envelope
1170 IF com=99 THEN GOSUB 1880:REM Clear
 ENV, ENT
1180 IF com=114 THEN LOCATE 4,3:etr=etr*
-1:IF etr=-1 THEN PRINT "-";:GOTO 1090 E
LSE PRINT " ";:GOTO 1090:REM Toggle ENT
Repeat on/off
1190 IF com=242 OR com=243 THEN GOSUB 21
00:GOTO 1220:REM Select Parameter
1200 IF com>239 AND com<246 THEN GOSUB 2
170:REM Alter Parameter
1210 IF com=9 THEN pn=1:GOSUB 1750:pn=2:
vort=vort XOR 3:REM Swap bewteen ENV and
 ENT:pn=black PEN
1220 GOSUB 1750:REM Print ENV, ENT
1230 WEND
1240 END
1250 :
1260 REM Set Up
1270 INK 0,12:INK 1,0:INK 2,26:INK 3,10
```

```
1280 BORDER 10
1290 DEFINT a-f,h-s
1300 DIM ev(15),et(15)
1310 etr=1:REM ENT Repeat
1320 cpos=1:REM Cursor Position
1330 dur=25
1340 pitch=956
1350 vort=1:REM enVORenT:1=ENV 2=ENT
1360 pn=2:REM PEN = Black
1370 yscale=256/4095:ysca=1
1380 :
1390 WINDOW #1,1,40,1,7
1400 WINDOW SWAP 0,1
1410 PAPER 3:CLS
1420 PRINT "ENV 1";
1430 LOCATE 1,3:PRINT "ENT";etr;
1440 PEN 2
1450 LOCATE 10,6:PRINT"Y";SPC(15)"P";
1460 LOCATE 1,7:PRINT "W";SPC(4)"C";SPC(
5)"I";SPC(7)"R";SPC(6)"D";
1470 PEN 1
1480 LOCATE 11,6:PRINT CHR$(22)+CHR$(1);
"scale 1:";ysca;"      itch:";pitch;
1490 LOCATE 2,7:PRINT "ipe  lear  nitial
  epeat  ur :";dur;CHR$(22)+CHR$(0);
1500 ORIGIN 32,32,0,639,287,0
1510 :
1520 ev(1)=1:ev(2)=15:ev(3)=5:ev(4)=5:ev
(5)=-1:ev(6)=10:ev(7)=1:ev(8)=0:ev(9)=50
:ev(10)=10:ev(11)=-1:ev(12)=5:ev(13)=0:e
v(14)=0:ev(15)=0
1530 REM 16 sec ENV:ev(1)=1:ev(2)=15:ev(
3)=200:ev(4)=14:ev(5)=0:ev(6)=100:ev(7)=
0:ev(8)=0:ev(9)=0:ev(10)=0:ev(11)=0:ev(1
2)=0:ev(13)=0:ev(14)=0:ev(15)=0
1540 RETURN
1550 et(1)=10:et(2)=32:et(3)=10:et(4)=40
:et(5)=-16:et(6)=5:et(7)=0:et(8)=0:et(9)
=0:et(10)=0:et(11)=0:et(12)=0:et(13)=0:e
t(14)=0:et(15)=0
1560 REM Vibrato:et(1)=1:et(2)=1:et(3)=5
:et(4)=2:et(5)=-1:et(6)=5:et(7)=1:et(8)=
```

```
1:et(9)=5:et(10)=0:et(11)=0:et(12)=0:et(
13)=0:et(14)=0:et(15)=0
1570 RETURN
1580 :
1590 REM Draw Axis
1600 CLG 0
1610 MOVE 0,0:DRAW 592,0,1
1620 MOVE 0,-16:DRAW 0,240
1630 TAG
1640 FOR n=15 TO 255 STEP 16
1650 MOVE -16,n:PRINT "_";
1660 NEXT n
1670 TAGOFF
1680 WINDOW SWAP 1,0
1690 LOCATE 3,25
1700 PAPER 0:PRINT "ENV Time:
ENT Time:";
1710 WINDOW SWAP 0,1
1720 RETURN
1730 :
1740 REM Print ENV, ENT
1750 LOCATE 6,vort OR 1:REM 1 or 3
1760 FOR n=1 TO 15
1770 IF vort=1 THEN e$=STR$(ev(n)) ELSE
e$=STR$(et(n))
1780 IF VAL(e$)>=0 THEN e$=RIGHT$(e$,LEN
(e$)-1)
1790 PRINT ",";
1800 IF cpos=n THEN PEN pn
1810 PRINT USING "&";e$;
1820 PEN 1
1830 NEXT n
1840 IF com=99 THEN PRINT SPC(35) ELSE P
RINT SPC(4)
1850 RETURN
1860 :
1870 REM Clear ENV or ENT
1880 cpos=1
1890 IF vort=2 THEN 1910
1900 FOR n=1 TO 15:ev(n)=0:NEXT:RETURN
1910 FOR n=1 TO 15:et(n)=0:NEXT:RETURN
1920 :
```

```
1930 REM Alter Note Duration
1940 LOCATE 32,7:PRINT SPC(4):LOCATE 32,
7:INPUT dur
1950 LOCATE 32,7:PRINT dur;
1960 RETURN
1970 :
1980 REM Alter Pitch
1990 LOCATE 32,6:PRINT SPC(5):LOCATE 32,
6:INPUT pitch
2000 LOCATE 32,6:PRINT pitch;
2010 RETURN
2020 :
2030 REM Alter Yscale of ENT Graph
2040 LOCATE 19,6:PRINT SPC(4):LOCATE 19,
6:INPUT ysca
2050 LOCATE 19,6:PRINT ysca;"  ";
2060 yscale=256/4095*ysca
2070 RETURN
2080 :
2090 REM Select Parameter
2100 IF com=242 THEN cpos=cpos-1
2110 IF com=243 THEN cpos=cpos+1
2120 IF cpos>15 THEN cpos=1
2130 IF cpos<1 THEN cpos=15
2140 RETURN
2150 :
2160 REM Alter p,q,r and t,v,w
2170 IF com=240 THEN inc=1
2180 IF com=241 THEN inc=-1
2190 IF com=244 THEN inc=10
2200 IF com=245 THEN inc=-10
2210 :
2220 IF vort=2 THEN 2250
2230 ev(cpos)=ev(cpos)+inc
2240 ON ((cpos+2) MOD 3)+1 GOTO 2290,233
0,2370
2250 et(cpos)=et(cpos)+inc
2260 ON ((cpos+2) MOD 3)+1 GOTO 2420,246
0,2500
2270 :
2280 REM p
2290 IF ev(cpos)<0 THEN ev(cpos)=127
```

```
2300 IF ev(cpos)>127 THEN ev(cpos)=0
2310 RETURN
2320 REM q
2330 IF ev(cpos)<-128 THEN ev(cpos)=127
2340 IF ev(cpos)>127 THEN ev(cpos)=-128
2350 RETURN
2360 REM r
2370 IF ev(cpos)<0 THEN ev(cpos)=255
2380 IF ev(cpos)>255 THEN ev(cpos)=0
2390 RETURN
2400 :
2410 REM t
2420 IF et(cpos)<0 THEN et(cpos)=239
2430 IF et(cpos)>239 THEN et(cpos)=0
2440 RETURN
2450 REM v
2460 IF et(cpos)<-128 THEN et(cpos)=127
2470 IF et(cpos)>127 THEN et(cpos)=-128
2480 RETURN
2490 REM w
2500 IF et(cpos)<0 THEN et(cpos)=255
2510 IF et(cpos)>255 THEN et(cpos)=0
2520 RETURN
2530 :
2540 REM Sound and Graph Routine
2550 GOSUB 2720:REM Get Last ENV Section
2560 IF vlast=0 THEN LOCATE 1,5:PRINT "T
here are no sections in this ENV.":FOR d
=1 TO 3000:NEXT:LOCATE 1,5:PRINT SPC(34)
:RETURN
2570 ENV 1,ev(1),ev(2),ev(3),ev(4),ev(5)
,ev(6),ev(7),ev(8),ev(9),ev(10),ev(11),e
v(12),ev(13),ev(14),ev(15)
2580 POKE(&B61A),vlast\3+1:REM \ is abov
e CTRL
2590 :
2600 GOSUB 2790:REM Get Last ENT Section
2610 IF tlast=0 THEN POKE(&B70A),0:GOTO
2660:REM Skip ENT Definition
2620 :
2630 ENT etr,et(1),et(2),et(3),et(4),et(
5),et(6),et(7),et(8),et(9),et(10),et(11)
```

```
,et(12),et(13),et(14),et(15)
2640 IF etr=1 THEN POKE(&B70A),tlast\3+1
 ELSE POKE(&B70A),tlast\3+1+128:REM \ is
 above CTRL
2650 :
2660 GOSUB 3560:REM Play Note or Tune
2670 GOSUB 2860:REM ENV & ENT Times
2680 ON vort GOSUB 3100,3290:REM Print
ENV or ENT Graph
2690 RETURN
2700 :
2710 REM Find Last ENV Section
2720 vlast=0
2730 FOR j=13 TO 1 STEP -3
2740 IF NOT(ev(j)=0 AND ev(j+1)=0 AND ev
(j+2)=0) THEN vlast=j:j=1
2750 NEXT j
2760 RETURN
2770 :
2780 REM Find last ENT Section
2790 tlast=0
2800 FOR j=13 TO 1 STEP -3
2810 IF NOT(et(j)=0 AND et(j+1)=0 AND et
(j+2)=0) THEN tlast=j:j=1
2820 NEXT j
2830 RETURN
2840 :
2850 REM Find ENV Time
2860 vtim=0
2870 FOR n=1 TO vlast STEP 3
2880 pt=ev(n+2):IF pt=0 THEN pt=256:REM
 If Pause Time=0 it is treated like 256
2890 IF ev(n)=0 THEN vtim=vtim+pt:GOTO 2
910:REM Step Count=0
2900 vtim=vtim+ev(n)*pt:REM Step Count*P
ause Time
2910 NEXT n
2920 :
2930 REM Find ENT Time
2940 ttim=0
2950 FOR n=1 TO tlast STEP 3
2960 pt=et(n+2):IF pt=0 THEN pt=256:REM
```

```
 If Pause Time=0 it is treated like 256
2970 IF et(n)=0 THEN ttim=ttim+pt:GOTO 2
990:REM A Step Count of 0 acts like 1
2980 ttim=ttim+et(n)*pt:REM Step Count*P
ause Time
2990 NEXT
3000 :
3010 REM Print Times
3020 WINDOW SWAP 1,0
3030 LOCATE 12,25:PRINT SPC(8)
3040 PAPER 0:LOCATE 12,25:PRINT vtim/100
;
3050 LOCATE 32,25:PRINT SPC(8):LOCATE 32
,25:PRINT ttim/100;
3060 WINDOW SWAP 0,1
3070 RETURN
3080 :
3090 REM Draw ENV
3100 xscale=ROUND(592/vtim,2)
3110 MOVE 0,0
3120 FOR n=1 TO vlast STEP 3
3130 gxpos=ev(n+2):IF gxpos=0 THEN gxpos
=256:REM a Pause Time of 0 acts like 256
3140 gxpos=gxpos*xscale
3150 gypos=(ev(n+1) MOD 16)*16:REM Each
 step = 16 Graphics Units
3160 IF gypos<0 THEN gypos=gypos+256:REM
 Fold over effect
3170 IF ev(n)=0 THEN DRAW XPOS,gypos,2:D
RAWR gxpos,0:GOTO 3250:REM If Step Count
=0 THEN Amp=Absolute Step Size
3180 :
3190 REM Number of Step Counts
3200 FOR g=1 TO ev(n)
3210 IF YPOS+gypos>=256 THEN DRAW XPOS,Y
POS+gypos-256,2:DRAWR gxpos,0:GOTO 3230:
REM If repeated, steps will take graph
off scale
3220 DRAWR 0,gypos,2:DRAWR gxpos,0
3230 NEXT g
3240 :
3250 NEXT n
3260 RETURN
```

```
3270 :
3280 REM Draw ENT
3290 IF tlast=0 THEN LOCATE 1,5:PRINT "T
his ENT is empty.":FOR d=1 TO 3000:NEXT:
LOCATE 1,5:PRINT SPC(18):RETURN
3300 :
3310 MOVE 0,INT(pitch/4095*256)
3320 DRAWR 592,0,1:REM pitch's X axis
3330 MOVE 0,INT(pitch/4095*256)
3340 xscale=ROUND(592/ttim,2)
3350 gypos=pitch
3360 :
3370 FOR n=1 TO tlast STEP 3
3380 gxpos=et(n+2):IF gxpos=0 THEN gxpos
=256:REM a Pause Time of 0 acts like 256
3390 gxpos=gxpos*xscale
3400 gystep=et(n+1)
3410 :
3420 REM Number of Step Counts
3430 IF et(n)=0 THEN scount=1 ELSE scoun
t=et(n):REM Step Count of 0 acts like 1
3440 FOR g=1 TO scount
3450 IF gypos+gystep>4095 THEN gypos=gyp
os+gystep-4096:DRAW XPOS,gypos*yscale,2:
GOTO 3490
3460 IF gypos+gystep<0 THEN gypos=4096+g
ypos+gystep:DRAW XPOS,gypos*yscale,2:GOT
O 3490
3470 gypos=gypos+gystep
3480 DRAWR 0,gystep*yscale,2
3490 DRAWR gxpos,0
3500 NEXT g
3510 :
3520 NEXT n
3530 RETURN
3540 :
3550 REM Play Note or Tune
3560 IF com=13 THEN SOUND 129,pitch,0,0,
1,1:RETURN:REM Play Note
3570 :
3580 REM Play Tune
3590 SOUND 129,0,1,0:REM Flush channel
```

```
3600 RESTORE
3610 FOR n=1 TO 7:READ note
3620 SOUND 1,note,dur,0,1,1
3630 NEXT n
3640 DATA 478,478,536,478,402,478,478
3650 RETURN
```

Commentary

A description of the routines affecting the tone envelope will be given as they occur. You may like to skip these explanations now and refer back to them when you reach the tone envelope section.

The *Set Up* routine at line 1270 sets the screen and border colours and some initial variables. The array, ev, holds the volume envelope parameters and et holds the tone envelope parameters. yscale and ysca are used to alter the Y axis of the ENT graph. The options are printed in the text window. CHR$(22) in lines 1480 and 1490 turns transparency on and off so the black letters don't overwrite the white initials (see the User Guide Chapter 5 Page 2). A graphics window is defined in line 1500 with a new graphs origin corresponding to the intersection of the X and Y axis. Volume and tone envelopes are defined: an alternative pair have been included. The routine at 1600 draws the graph axis.

The routine at line 1750 prints the envelope definition to the screen. 'vort OR 1' is a short way of returning a 1 to 3 value — try it. Amstrad BASIC prints positive numbers with a leading space and all numbers with a trailing space. This separates a list of numbers and has many advantages — until you have an application such as this which doesn't require it. We solve the problem by turning the parameters into strings in line 1770 and then in line 1780 stripping of a leading space if the number is positive. If the cursor position is the same as the parameter number, the parameter is printed in white. Line 1840 prints spaces to blank out any possible remains of a previous definition.

All this sets up the program. Control then passes to line 1090 which waits for a key to be pressed, a command, which is converted to a number and stored in the variable, com. Lines 1110 to 1220 call the subroutines and are well REMed. We examine the subroutines in the order in which they appear in the program.

The routine at line 1880 places the cursor at the start of the envelope definition and clears the parameters to 0.

The next three routines alter the duration, the pitch and yscale. They are all similar and straightforward.

Lines 2100 to 2130 alter the cursor position to indicate a move from parameter to parameter. It is set to wrap around, so that if it goes of the end it appears at the front.

Lines 2170 to 2200 determine the amount by which the parameter is to be altered. Line 2220 skips to the ENT section if ENT has been selected. Lines 2230 and 2250 add the increment and a jump is made to check that the parameter has not gone out of bounds. These, too, wrap around.
The *Sound and Graph* routine at line 2550 does most of the hard work. To understand what it does and why it does it, we must know how envelopes are recognised by the computer.
The act of entering an envelope definition into the computer determines how many sections the envelope has. For example:

ENV 1,1,15,10

automatically defines one section, and

ENV 1,1,15,10,1,–5,20

defines two sections. If we were to enter

ENV 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

into the computer we would not, in fact, have a blank or default envelope but a five-section envelope. The R parameters of 0 acting as 256 would give the envelope a duration of 5 × 256 although the volume level would be 0.
Technically, in order to program corectly an envelope with less than five sections we must enter the short form of the envelope: entering unwanted sections as 0,0,0 will not do. If you look at lines 2570 and 2630, however, you will see that we do just that. As with all information, the computer must store it somewhere and what we do is to tap into its storage area and alter the memory location which tells it how many sections the envelope has. For ENV 1 this is memory location &B61A and for ENT 1 it's &B70A. (We'll take another look at this storage area towards the end of the chapter.)
In order to do this we need to know how many sections have been defined. The program lists all five envelope sections for convenience but it assumes that if the parameters in the last section are all 0 then this section is not required. The sound and graph routine, therefore, makes a call to the routine at line 2720 to find the last ENV section. This routine, and the one at line 2790 for the tone envelope, steps backwards through the parameters in steps of three looking for a section which *doesn't* consist of three 0's. If it finds one then that's the last section. If it doesn't then there are no sections at all. Line 2740 may look complicated but it simply sets the conditions which it is NOT looking for. So, to use an envelope you've defined, if it appears on the screen with trailing 0's, omit final sections consisting only of 0's. If a blank ENV has

been defined, a message is printed and the routine returns. Otherwise, the envelope is defined and the number of sections POKEd into the relevant memory address.

Lines 2600 to 2640 do the same for the tone envelope. Notice line 2640. If a repeat has been set, the computer stores this by adding a further 128 to the number of envelope sections. You may recognise this as a bit significant figure. In this case bit 7 controls whether nor not an ENT will repeat.

Notice the reversed backslash character in lines 2580 and 2640. This provides a DIV function which returns only the whole number part of the result after a division.

Line 2660 calls a routine to play a note or tune. This should fill the sound buffer and return quickly, enabling the program to draw the graph at the same time as the sound is playing, saving a little time.

Line 2670 calls a routine which calculates the length of the envelopes. These take into account the special pause time and step count settings. The times are printed at the bottom of the graph screen.

Finally, line 2680 calls the relevant graph-drawing routine. These are quite intricate and must take into account all the ifs and buts of the envelope definitions and the scaling factors.

The ENV graph first calculates the xscale in line 3100 according to the total length of the envelope. Line 3120 steps through each envelope section. gxpos holds the position of the graphics cursor on the X axis. gypos holds the Y position and is adjusted in line 3160 to take into account the fold over effect. Line 3170 draws the volume level if the step count is 0.

The loop between lines 3200 and 3230 runs through the number of steps counts drawing the volume levels as it goes. Most lines are drawn relative to the last point with DRAWR but if the level is going to fold over, line 3210 draws a line to the correct volume level.

If ENT has been selected, the routine at line 3290 draws the tone envelope. If the envelope is empty, the first line lets us know and returns. Line 3310 draws a line across the X axis proportional to the pitch value. xscale is determined as it is in the ENV graph. gypos is made equal to pitch. The program then runs through each envelope section. gxpos is set equal to the pause time and gystep is set according to the step size. The program loops through the number of step counts and increments gypos by gystep. Lines 3450 and 3460 take care of the fold over effect. They use yscale which was set in line 1370 and can be altered by pressing Y.

The variable gystep is used to keep track of the exact pitch value. We have only 256 graphics units to work with and as we are trying to display a resolution of 4096 units we cannot work in terms of graphic units as we did with the ENV graph. If no fold over occurs then lines 3470 to 3480 can draw the pitch changes normally.

The last routine plays a note or the tune. The tune consists of seven notes. If the note duration is very long, the buffer will fill and the program will be held up. If you want to experiment with very long notes then it may be advisable to reduce the number of notes in the tune to five. Note that line 3590 flushes the buffer first, so once a couple of notes have played you can trigger the routine again without having to wait until the tune has played right through. If you use only five notes try this:

```
3640 DATA 213,190,239,478,319
```

Using the program

When run, the program will print all five sections of ENV 1 along the top of the screen. Just below, ENT 1 is waiting to be activated which we will do later in the chapter. The Y axis below is divided into the 16 volume levels, the X axis represents time. As an envelope can last from 1/100th of a second to several minutes the graph is scaled to fill as much of the screen as possible. The times of the envelopes, therefore, are printed along the bottom and are in seconds.

If you press ENTER, the current envelope will sound and its graph will be displayed on the screen. The note will last for the duration of ENV 1 which will appear at the bottom of the screen. Pressing P allows you to alter the pitch of the note which must be between 0 and 4095. Type in the new value and press ENTER.

If you press the space bar, a short tune will play so you can gauge the effect of the envelope on several notes played in succession. Pressing D allows you to alter the duration of the notes.

You can leave each graph on the screen for comparisons. Pressing W will wipe the display if it becomes too cluttered.

C will clear all envelope parameters to 0 and return the cursor to the first parameter so you can begin a new envelope definition.

I re-establishes the initial envelope parameters which are defined at the end of the *Set Up* routine.

Y and R are used in conjunction with the tone envelope and will be explained later.

The envelope parameters are altered with the cursor keys. Left and right moves from one parameter to another with the current position highlighted in white. The up and down keys alter the value of the parameter in increments of one. If you hold SHIFT down at the same time, they will alter in increments of ten.

The only other control is the TAB key which toggles between ENV and ENT.

Designing envelopes

A straightforward approach to envelope design might entail using one ENV section to produce one of the ADSR sections. The initial envelope does this and you can see the result on the screen. This is a good way to familiarise yourself with the parameters and how they work.

You will notice how the volume 'folds over' when values try to take it out of range and the graph shows this quite clearly. This folding over can be used to produce very complex envelopes indeed and it can also be used to create useful one section envelopes. Program 1.4 used a one section percussive envelope:

```
ENV 1,16,15,14
```

Enter this in the program and alter the D parameter from 1 through to 16. This is only one of five sections so you can see how complex envelopes can be built up. Because of the fold over effect, notice how a Q setting of 1 is the same as a setting of 17. Also notice that the duration is determined by the P and R parameters.

From the sample envelopes in Figure 5.2 see if you can produce equivalent ENV definitions.

The hardware ENV

The User Guide has little to say about hardware envelopes. You could ignore them altogether with no loss to your sound and music programs, but they can produce effects not readily available from software envelopes so let's take a look at them.

A hardware envelope section provides an alternative to the P, Q and R of a software section and consists of only two parameters instead of three. It takes the form:

```
=HE,EP
```

where HE is the hardware envelope number and EP is the envelope period. HE can take values from 0 to 15. EP takes values from –32768 to 65535 and determines the length of the steps within the hardware envelope in units of 128 microseconds.

Hardware envelopes have predefined characteristics and you can mix hardware and software sections in the same envelope definition. Hardware sections, however, do not have an associated pause time and in order to be heard they must be followed by a software section which defines a pause. For example:

ENV 1,=HE,EP,5,0,100

will provide a five second pause. If two or more consecutive hardware sections are defined control will pass straight through to the last section. If an ENV definition ends with a hardware envelope, however, it will be given a duration of two seconds.
The main envelopes are from 8 to 15. Envelopes 0 to 3 are the same as number 9, and 4 to 7 are the same as number 15. This short program gives a demonstration:

```
100 REM PROGRAM 5.2
110 REM Hardware Amplitude Envelopes
120 :
130 ep=3200
140 FOR he=8 TO 15
150 PRINT he
160 ENV 1,=he,ep,5,0,100
170 SOUND 1,478,0,0,1
180 a$=INKEY$:IF a$="" THEN 180
190 NEXT he
```

The last envelope section in the ENV definition produces a pause time of 5 seconds, so wait 5 seconds before pressing a key to move on to the next envelope. Alter ep in line 130 and hear the difference. As ep determines the 'speed' of the envelopes we can only talk about their characteristics in relative terms. Here is a summary of their properties.

8: Repeating fast attack, slow decay.
9: Fast attack, slow decay then held at zero volume.
10: Fast attack followed by repeating slow decay, slow attack.
11: Fast attack, slow decay, fast attack, sustained at maximum volume.
12: Repeating slow attack, fast decay.
13: Slow attack, sustained at maximum volume.
14: Repeating slow attack, slow decay.
15: Slow attack, fast decay then held at zero volume.

The value of V in the SOUND command is ignored by a hardware envelope. With an ep value of 300, envelope 8 produces a banjo effect. Try with smaller values, too, less than 20. This tends to thicken the sound. Envelope 14 can produce tremolo effects (see Chapter 6 for an explanation of tremolo): try ep values of 300 and more. For many applications, you may well find a hardware envelope to suit.

The tone envelope (ENT)

You may be relieved to know that this is slightly easier to understand than the volume envelope. It is used to create many weird and wonderful effects as well as more subtle musical sounds.

As ENV varies the volume of a note over a period of time, so ENT varies the pitch of the note. The initial frequency on which the tone envelope operates is determined by the value of P in the SOUND command. The User Guide lists the tone envelope as follows:

ENT S,T1,V1,W1,T2,V2,W2,T3,V3,W3,T4,V4,W4,T5,V5,W5

where S is the ENT number. Again T, V and W are not very meaningful but they have the same names as the P, Q and R parameters although their ranges differ. They are listed in *Figure 5.5*. After defining an ENT, the computer will remember it until it is reset or until the ENT is redefined. An ENT can be cleared by entering the ENT number without any parameters.

Figure 5.5: ENT parameters and value ranges

Parameter	Name	Range
S	Envelope Number	1 to 15 (−ve for repeat)
Software	ENT S,T,V,W	
T	Step Count	0 to 239
VC	Step Size	−128 to 127
W	Pause Time	0 to 255
Hardware	ENT S,=TP,PT	
=TP	Tone Period	0 to 4095
PT	Pause Time	0 to 255

Each T, V and W section is complete in itself and up to five such sections can be defined. They each operate in exactly the same way. A complete section is mandatory but the number of sections is optional.

T specifies the number of steps the section will have, V specifies the amount by which the pitch will vary over each step and W specifies how long each step will take in one hundredths of a second.

Special S, T, V and W values

If S is negative it causes the tone envelope to repeat. It is not called up with a negative parameter in the SOUND command like a repeating volume envelope.
If T=0 it acts like a value of 1.
If V tries to take the pitch outside its range (0 to 4095), the pitch will fold over as the volume did in the volume envelope.
If V=0 then the section will hold the pitch for a duration equal to T=W.
A W value of 0 is treated as 256.
The duration of a tone envelope has no effect upon the duration of a note. Remember, the lower the pitch number, the higher the pitch, so a tone envelope section set to increase the pitch will produce lower notes — and vice versa.

Designing tone envelopes with the graph program

Load and run Program 5.1. If you haven't done so before, press TAB. This toggles between ENV and ENT. ENT 1 will be defined as a string of 0s. Press I to call up the initial envelope. ENTER and the space bar play a note and the tune as before but now you will see the pitch envelope on the screen. As there are 4096 pitch values, the Y axis is divided into 4096 sections. Obviously, given the screen resolution, very small changes in pitch will appear as a straight line. Pressing Y allows you to alter the Y scale. If you scale it up too far, it will go off the screen. This facility was included to enable you to see small changes in the pitch value.
An axis is drawn across the screen at a level proportional to the pitch of the note. If you press P and set pitch near to its upper or lower limt you will see and hear how it folds over when it reaches these limits.
R toggles the tone envelope repeat on and off. A repeat is set by making the envelope number negative. The graph will only draw one envelope cycle even with repeat set but the effects of a repeating envelope will be heard. The graph reflects the changes in the pitch number, not the pitch produced. An upward line, therefore, represents a downward pitch.
The ENT time is now printed on the screen. This is the time of one cycle and will, of course, not apply if the envelope repeats. If it is longer than the ENV time its full effect will not be heard. If it is shorter and does not repeat, the note will conclude at a constant pitch. The ENV and ENT graphs are not drawn to the same scale. I haven't included any sample envelope parameters — discovering your own is far better and far more interesting — but the instrument characteristics table near the end of the chapter may give you some ideas to experiment with.

The hardware ENT

This, too, is easier to understand than the hardware volume envelope. It takes the form:

=TP,PT

where TP is the tone period or pitch number and PT is the pause time or duration. TP sets the absolute pitch of the note and PT effectively sets the duration of the note: it is the same as W in the software envelope. Hardware and software sections can be mixed.
Hardware envelopes can be defined to play short tunes as in Program 1.5. You can experiment along similar lines quite easily.

Multi-section envelopes

If you read the program commentary and realised that we could alter the number of sections in an envelope by POKEing a memory location, perhaps you wondered what would happen if we POKEd that location with a value greater than five. Well, as you might expect, the computer believes that envelope has more sections. We can use this to create ENVs and ENTs with a hundred or more sections — a facility not found on even the most comprehensive of synthesisers. If you are just coming to terms with five-section envelopes, you may pale at the thought of defining more. I did say, too, that most sounds can be defined using only four sections — ADSR — and this is true. However, once you become familiar with five sections you may be inclined to experiment further.

The envelope storage area

There are three memory areas involved in storing envelope definitions and we will use hex notation when referring to them. Hex notation is explained in the User Guide in Appendix II. It is particularly suited to describing the envelope storage area because each envelope occupies 16 bytes or &10 in hex.
When an envelope is first defined, the definition goes into a buffer which occupies the area from &ADBB to &ADCA. The volume envelope definitions are held in memory from &B61A to &B709 and the tone envelopes are held in locations &B70A to &B7F9. The ENT storage area follows on from the ENV area.
When an envelope is defined, the total contents of the buffer are transferred to the storage area regardless of how many sections the definition contains. The first location of each envelope's storage area holds the number of sections it contains and this can be altered by POKEing as we have seen. The next 15

bytes define the envelope sections and are read in threes. This does not mean that if ENV 1 was POKEd with 10 sections, ENV 2 would follow on from ENV 1. The first byte after the five ENV 1 section definitions holds the number of sections in ENV 2. To program more than five sections, therefore, we must POKE consecutive bytes with the required values.
The memory locations are easy to calculate in hex:

ENV 1 &B61A to &B629
ENV 2 &B62A to &B639
ENV 3 &B63A to &B649 . . .
ENV 15 &B6FA to &B709
ENT 1 &B70A to &B719
ENT 2 &B71A to &B729
ENT 3 &B72A to &B739
ENT 15 &B7EA to &B7F9

The next program displays these memory locations and shows how definitions are stored. Switch the computer off then on before entering or loading and running the program.

```
100 REM PROGRAM 5.3
110 REM Envelope Storage Area
120 :
130 PRINT"BUFF   ";
140 FOR n=&ADBB TO &ADCA
150 IF n=&ADBB THEN PEN 3 ELSE PEN 1
160 PRINT PEEK(n);CHR$(8);
170 NEXT n
180 PRINT
190 :
200 FOR n=1 TO 3:REM First 3 ENVs Only
210 PRINT:PRINT"ENV";n;
220 FOR m=0 TO 15
230 IF m=0 THEN PEN 3 ELSE PEN 1
240 PRINT PEEK(&B61A+m+(n-1)*16);CHR$(8)
;
250 NEXT m
260 NEXT n
270 :
280 PRINT:PRINT
290 FOR n=1 TO 3:REM First 3 ENTs Only
300 PRINT:PRINT"ENT";n;
```

```
310 FOR m=0 TO 15
320 IF m=0 THEN PEN 3 ELSE PEN 1
330 PRINT PEEK(&B70A+m+(n-1)*16);CHR$(8)
;
340 NEXT m
350 NEXT n
360 SOUND 1,478,0,0,1,1
370 PRINT:PRINT
380 LIST 420-440
390 END
400 :
410 REM Envelopes
420 ENV 1,1,15,10,2,-2,20,1,0,200,11,-1,
10
430 ENV 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0:
ENV 1
440 ENT 1,1,1,1
```

Commentary

Line 160 prints the contents of the buffer. Lines 240 and 330, which may look more complicated, just calculate the offset from the start of the storage areas. By altering lines 200 and 290 you can select which envelopes will be printed out. The number of sections in each envelope is highlighted in red. If you switched the computer off and on as suggested, when run, the buffer will show random figures. Pressing CTRL/SHIFT and ESC leaves it unaltered — if you try this you 'll have to load the program in again. The storage areas will all show 0. To clear the buffer enter:

ENV 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0:ENV 1

This is one of the lines listed at the bottom of the screen. As they are outside the program loop they will not be executed. Use the cursor and edit keys to enter and/or alter them. Copy in the first ENV 1 definition then run the program. You will see that the buffer and ENV 1 now hold the same information. Note how negative parameters are stored. Now type in:

ENV 2

and run again. ENV 2 will hold the same information as the buffer. Notice also that the number of sections is now zero. Copy in ENT 1. No surprises. Now enter:

ENT -1,1,1,1

and the number of sections will be shown as 129. In bit terms, this represents setting bit 7 and is how the computer stores a repeat. Enter a blank repeating envelope to prove it:

ENT -1

and the number of sections will show as 128.

Hardware envelope storage

The first parameter of a hardware envelope is stored as a 16 bit number in two consecutive bytes. We'll examine the tone envelope. Enter:

ENT 1,=478,20,=319,40

The pause time is the same but the pitch number has been split into two bytes, replacing the T (step count) and V (step size) parameters. Only values up to 239 are accepted for step count (T) and now you will see that T1 and T2 have values of 241.
The pitch number is stored in the following way: for values over 240 in the T column, add 2^8 or 256 to the pitch value. 478 is stored as:

2141,222

which means

(241–240)*2^8 + 222

which equals 478. This, generally, is how the computer stores numbers greater than 255, i.e. in more than one byte. It may seem complicated to us but we don't have to worry about it in normal programming.
See if you can discover how the volume hardware envelopes are stored and why they have no associated pause time. You can start by comparing these:

ENV 1,=8,255
ENV 1,=8,256

Increasing the number of envelope sections

After experimenting with the last program you may like to see what results

can be produced by defining a multi-section envelope. The next program prompts for each P, Q and R parameter and automatically POKEs the values into ENV 1's storage area. Add the routine to the end of Program 5.3 and run it by entering:

RUN 1030

After completion, you will be able to see how the values have been stored by running the original program.

```
1000 REM PROGRAM 5.4
1010 REM Multi-Section Envelopes
1020 :
1030 count=0
1040 mem=&B61A
1050 WHILE k$<>"N"
1060 count=count+1
1070 PRINT "section";count
1080 INPUT "P (step count)";ev(1)
1090 INPUT "Q  (step size)";ev(2)
1100 ev(2)=(256+ev(2)) MOD 256
1110 INPUT "R (pause time)";ev(3)
1120 FOR n=1 TO 3
1130 POKE(mem+n+(count-1)*3),ev(n)
1140 NEXT
1150 PRINT
1160 PRINT "Another section (Y/N)?"
1170 k$=UPPER$(INKEY$):IF k$="" THEN 117
0
1180 PRINT
1190 WEND
1200 POKE(mem),count
```

Commentary

Set mem at line 1040 to the beginning of the memory storage area of the envelope you wish to define. It is currently set to define ENV 1. Lines 1120 to 1140 POKE the memory with the section values just entered. The routine is very short and no error traps have been included. Line 1100 adjusts negative values of Q because a byte cannot intrinsically hold a negative value. When all your sections have been entered, press N and the number of sections will be POKEd to the storage area.

We have seen how values outside the range of the software envelopes are used to store hardware envelopes. Using this program, you can enter such values directly but the results may be unpredictable.
An interesting and practical use for a multi-section envelope is to define a tone hardware envelope with a series of pitches which will play a tune. You need to convert the pitch number into two bytes, high and low, as follows:

dif=pitch/256
high=240+dif
low=pitch−dif*256

Add these lines to Program 5.4:

```
1015 REM PROGRAM 5.5
1016 REM Multi-Section Hardware
1017 REM Tone Envelopes
1018 REM Insert These Lines
1019 REM Into PROGRAM 5.4
1020 :
1040 mem=&B70A
1080 INPUT "Pitch";pitch
1090 dif=pitch\256
1100 ev(1)=240+dif
1105 ev(2)=pitch-dif*256
1195 PRINT "Repeat (Y/N)?"
1196 r$=UPPER$(INKEY$):IF r$="" THEN 119
6
1200 IF r$="Y" THEN POKE(mem),count+128
ELSE POKE(mem),count
```

Don't forget to define a suitable volume envelope so that your tune can be heard.

Experimenting with the programs

There is a wealth of music potential in the envelope commands, using them either to define normal software and hardware envelopes or programming them directly as we have been doing in the last two programs. A few experiments, especially with Program 5.1, will quickly help you to become familiar with the parameters and their effects.
Multi-section envelopes can be used economically to save memory and programming effort. How much easier it is to issue a SOUND command than

to run through a read-data-play-note routine.

Instrument characteristics

To aid further experiments, *Figure 5.6* describes the characteristics of some common instruments which you may like to convert into envelope parameters. Bearing in mind the limitations of the sound chip, you are unlikely to produce exact imitations but remember that one man's clarinet is another man's oboe so do program the sound that *you* like. The envelope shapes are described in terms of attack and decay. In this context, the decay refers to the release phase. The ADSR decay phase will run along similar lines to the sample instrument envelopes in Figure 5.2. The Amstrad's five phase envelopes give you another string to your bow, so to speak, and multi-section envelopes release you from the confines four-section ADSR envelopes sometimes impose. Don't forget, too, you can create an enormous range of completely new sounds. This is one of the most interesting and creative aspects of music-making. With the addition of a few frills, courtesy of the tone envelope, you can create a most acceptable micro orchestra. The next chapter looks at how to create and use such frills which we refer to as musical ornaments.

Figure 5.6:

Instrument	Octave Range	Attack	Decay	Applications and Special Effects
Violin	0 to 2	Slow	Medium/Slow	Portamento, Vibrato, Scales
Viola	−1 to 1	Slow	Medium/Slow	
Cello	−2 to 0	Slow	Medium/Slow	
Bass Guitar	−2 to 0	Fast	Slow	Rhythmic Figures
Trumpet	0 to 2	Fast	Medium/Fast	Vibrato
Trombone	=2 to 0	Medium/Fast	Medium/Fast	Portamento
Tuba	−3 to −1	Slow	Medium/Fast	Use for staccato bass line
Alto Saxophone	1 to 1	Medium/Fast	Medium/Fast	Slides up to note
Tenor Saxophone	−2 to 0	Medium/Fast	Medium/Fast	Bends note
Flute	0 to 2	Medium/Fast	Medium/Fast	Trills, Arpeggios
Clarinet	−1 to 1	Medium/Fast	Medium/Fast	Slow Vibrato
Oboe	0 to 2	Medium/Fast	Medium/Fast	
Bassoon	−2 to 0	Fast	Medium/Fast	Slow, mournful
Guitar	−2 to 1	Fast	Slow	Bend the note
Accordion	− to 2	Medium/Slow	Medium/Fast	Slightly out of tune
Harp	−3 to 3	Fast	Slow	Arpeggios, Glissando
Xylophone	0 to 33	Fast	Medium/Fast	Glissando
Snare Drum	Medium	Fast	Medium	Drum Rolls
Bass Drum	Low	Fast	Medium	Short beats
Organ	−3 to 3	Fast	Fast	Vibrato, full chords

CHAPTER 6
Musical Miscellanea

A synthesiser has many modules or sections which allow the musician to enhance and embellish the basic sounds he or she creates. Some of the resulting effects are quite intricate, like sustain, echo, reverberation, chorusing, phasing, flanging . . . the list goes on. In this chapter we will see what effects we can produce on the Amstrad.

Vibrato and tremolo — pitch and amplitude modulation

These are usually the first effects one learns to incorporate into synthesiser patches. They are so much a part of natural sounds and music that they really do add an extra dimension to the relatively lifeless sine or square wave. There exists a certain amount of confusion over these two terms, even among musicians, so we'll clear it up now.

Vibrato is a frequency modulation, tremolo is an amplitude modulation. The modulation is usually regular and consists of an increase and decrease in pitch or volume above and below the note's pitch or volume level. A graph of this variation would be similar to the sine waves we saw in Chapter 1.

Not many instruments produce tremolo. Electronic organs produce it mechanically and electronically and singers often use tremolo (and vibrato) to enhance their tone. Vibrato is far more common and often used by strings, woodwind and many brass instrumentalists. The 'tremolo' arm on a guitar, popular in the 1960s, actually produces vibrato. By stretching and relaxing the strings, the pitch rises and falls. If the arm is worked rapidly it produces a regular vibrato; if it is used slowly it stretches the note producing a portamento.

The most pleasant rate of modulation in both vibrato and tremolo is around seven cycles per second, and the amount of modulation can vary from the subtle to the ridiculous. You have probably discovered examples of both in arcade-type games and you may have invented a few yourself with the program in the last chapter. Vibrato is the more common of the two so we will look at that first.

Creating vibrato with the tone envelope

The pitch variation in vibrato is not usually as great as a semitone. In a musical vibrato the pitch varies regularly, rising and falling in a sine wave pattern moving above and below the pitch of the tone. Extreme examples where the pitch varies more rapidly and over larger intervals are still technically vibrato, but can only be produced by electronic means for special effects. Here is an example:

```
10 ENT -1,20,10,1,40,-10,1,20,10,1
20 SOUND 1,478,500,7,0,1
```

Alteration of the speed and degree of pitch variation will dramatically alter the sound. Although we can create innumerable vibrato effects, only a few will be of use in a strictly musical context. To produce vibrato with the tone envelope, we can use the following formula as a rule of thumb:

```
ENT -1,X,D,S,2*X,-D,S,X,D,S
```

where D represents the depth of the vibrato and S represents the speed. X will have an effect on both these factors which is why the formula is only offered as a base from which to work. To produce a sine wave type of vibrato, the middle section should last twice as long as the first and third sections. You could also experiment with this arrangement:

```
ENT -1,X,D,S,X,-D*2,S,X,D,S
```

The degree of vibrato also depends upon the pitch of the note as the difference between pitch numbers increases as the pitch gets lower. Here are some vibrato envelopes for you to experiment with. You can use Program 5.1 to compare them and see how effective they are over a large pitch range. Set Yscale to 1:100.

```
ENT -1,3,1,2,6,-1,2,3,1,2
ENT -1,3,1,2,3,-2,2,3,1,2
ENT -1,1,1,1,1,-2,1,1,1,1
ENT -1,1,1,6,1,-2,6,1,1,6
```

You will soon find which vibrato effects are musically useful.

Producing tremolo effects

To produce tremolo, we need ideally to vary the volume in a sine wave pattern just as we varied the pitch to produce vibrato. We can do this by making the D parameter in the SOUND command negative. When D is negative, the absolute value determines the number of times the volume envelope will repeat. This, unfortunately, means losing some control over note durations, because they become dependent on the length of the volume envelope. To produce a sine wave tremolo modulation, set the initial volume in the SOUND command to 7. If you're using Program 5.1, alter these lines:

```
3560 IF com=13 THEN SOUND 129,pitch,-10,7,1,1
3620 SOUND 1,note,-dur,7,1,1
3110 MOVE 0,112
```

Line 3110 will offset the graph so we see how the volume varies with the initial volume level set to 7. Run the program and set this envelope:

```
ENV 1,8,1,3,14,-1,3,6,1,3
```

Pressing ENTER will produce an acceptable tremolo. If you try to play the tune, however, each note will repeat 25 times and will last 25 × 0.56 seconds. Alter the envelope to:

```
ENV 1,8,1,1,14,-1,1,6,1,1,
```

and set the duration to 1 and then 2. These are better but you can see the problems inherent in producing tremolo this way. You may like to try to define a non-repeating envelope which produces a tremolo effect. I haven't been able to produce a perfect tremolo envelope in this way although envelopes which work their way through a series of volume levels may suffice for certain applications. The above tremolo varies the volume from maximum to minimum level. For more subtle effects, restrict the range so it only varies over four or six volume levels. Hardware envelope number 14 with an EP of 500 or more will produce a tremolo without any side effects although you can not restrict the range. Add tone envelopes to these experiments and you will discover many interesting sounds, with pitch figures fading in and out with the tremolo effect.

In practice, you are unlikely to want tremolo — or vibrato — throughout a piece.

Trills — a special kind of vibrato

To produce vibrato on an instrument you need control over the pitches lying between individual notes. The notes on a piano, for example, are fixed and you cannot produce vibrato on a piano. The best you can do is to alternate rapidly between two adjacent notes and this is called a trill. Run the unmodified version of Program 5.1 and define this envelope:

ENT –1,1,–105,8,1,105,8

If you press ENTER it will produce a trill between the C below middle C and the D above. The pitch does not follow a sine wave pattern as with vibrato and tremolo but rather a square wave pattern as it jumps up and down from note to note. A trill is an oscillation between discrete notes and as such can be played by most instruments. If we want to include a trill in a piece of music we can instruct the computer to play the two notes just as we would instruct it to play any others but it is often more convenient to switch control to an envelope which would then take the place of, perhaps, up to 16 pitch commands.

A trill can be played over any interval. You will realise, however, that as the difference between pitches varies from note to note, the pitch produced by a trill envelope will only produce a true note when used with the initial pitch it was designed for. Luckily, our ears will forgive this on many occasions and a trill envelope may produce an acceptable trill over a one or two octave range. A flute playing an occasional trill sounds very effective, especially in military or brass band music, and a trill on a sustained note above a melody line is quite common.

```
100 REM PROGRAM 6.1
110 REM "Military Music" Introduction
120 :
130 ENT -1,1,-7,8,1,7,8
140 SOUND 1,60,270,6,0,1
150 FOR n=1 TO 9
160 READ pitch,dur
170 SOUND 4,pitch,dur,7
180 NEXT n
190 END
200 DATA 478,60,506,20,536,60,568,20,638
,20,716,20,758,20,851,20,956,30
```

Echo and reverberation

These are probably two of the most overused effects of a synthesiser, but in the right hands they are capable of creative and beautiful effects.
Echoes are produced when sound waves are reflected from a smooth hard surface such as a cliff. If you stand before a cliff and shout, the sound waves produced by your voice will hit the cliff and bounce back. The time lag will depend upon your distance from the cliff. In order to hear the reflected waves as a separate echo they must be separated from the original sound by at least one tenth of a second which means you must stand at least 54 feet away from the cliff.
A sound emitted in a room will bounce around the walls, floor and ceiling. As the reflections bounce back and forth, the result is a most complex series of multiple reflections. The net result is a reinforcement of the sound and it will seem to continue after the original sound source stops. This is a form of echo called reverberation, where the individual echoes are not discernible.
Reverberation time is the length of time required for the sound reflections in a room to fall to a certain level. Rooms with hard, reflective surfaces have longer reverberation times than rooms with padded walls.
Most commercial reverberation units produce their effect by means of a spring or metal plate. Because of the enormous number of vibrations involved, reverberation is not strictly possible on the Amstrad. The best we can do is to program a long release in the envelope. We can, however, produce echoes.

Producing echoes on the Amstrad

There must be a discernible gap between echoes, so the volume must drop significantly and rise again. We can program a volume envelope to produce this effect. Using Program 5.1, define this envelope:

```
ENV 1,18,5,10
```

You will see that we have the beginnings of a good echo but the last step reaches zero volume. If we increase the first parameter to 19 you will see the cycle starts over again sending the volume level up to 15 again. This is due to the fold over nature of the envelope we have defined. We can add a slow release to the end of the envelope to produce a pseudo reverberation effect like this:

```
ENV 1,18,5,10,10,-1,15
```

This echo envelope may suffice for many applications. You can alter the echo

speed by varying the R parameters. We can utilise all the sections:

ENV 1,1,15,17,8,7,17,1,–7,17,1,5,17,2,–2,17

This is quite a good echo. Again, the R parameters control the echo speed. Increase the very last parameter to lengthen the reverb effect. If you want to define the ultimate echo envelope you can do so by creating a multi-section envelope as described in the last chapter. You would then have control over the time lag between each individual echo as well as absolute control over the volume level of each echo.

Chorus, phasing, flanging and other spatial effects

These have been grouped under one heading because, although they each have their own character, their effects are produced by delaying a sound and/or varying its pitch or frequency. They are called 'spatial' effects because they alter our perception of the environment we think produced the sound — just as reverberation can make us think a sound originated in a big theatre.
When a group of musicians play in unison, they each play a slightly different pitch. This difference is very small but it tells the ear that there is more than one instrument. This is known as a chorus effect and is responsible for the beautiful sound of a string orchestra. Chorus units produce their effect by slightly altering the pitch and recombining it with the original sound.
Phasing, flanging and audio delay are very difficult to describe in words. They all produce a sort of wooshing sound and such effects can be heard on many electronic music albums. The sound of a jet plane taking off creates a phasing effect.
Flanging is rumoured to have been first produced by applying slight pressure to the flange of a spool of tape as it was playing, causing a small delay. If you can imagine two tapes being played in this manner, drifting in and out of sync with each other, you will have a good idea what flanging sounds like (and an excellent imagination!).
All these effects are produced by various forms of delay. The delays are reckoned in thousandths and hundredths of a second. When a sound is delayed and mixed back with the original sound certain frequencies are cancelled. If the delay is varied, the range of cancelled frequencies will vary and produce a shifting effect.
We'll leave the description there because there is not a lot we can do to recreate these effects exactly and you need to hear the sounds to appreciate them. Perhaps your local music shop will demonstrate their range of effects pedals.

Delay effects on the Amstrad

If we program the same pitch into two or more different channels the pitches should be exactly the same. If we alter the pitch of one channel by a small amount this will produce a chorus effect. The greater the pitch difference, the more pronounced and less subtle the effect. We can really go to town and use three chanels with a small pitch difference between each. As an example of how you can use this effect, the following program imitates an accordion.

```
100 REM PROGRAM 6.2
110 REM French Accordion Music
120 REM Using Chorus Effect
130 :
140 FOR tune=1 TO 14
150 READ note,dur
160 SOUND 1,note,dur,7
170 SOUND 2,note-1,dur,6
180 NEXT tune
190 END
200 DATA 426,48,338,48,201,48,213,432
210 DATA 426,48,319,48,201,48,213,240,26
8,48
220 DATA 239,72,402,24,426,24,478,24,426
,24
```

Alter the amount subtracted from note in line 170. You may prefer the effect produced by subtracting 3 or 4.
As you listen to two pitches which are almost in tune you will hear a pulse in the frequencies. This is called a beat, and is produced whenever two notes sound at not quite the same frequency. The beat frequency is the difference between the two pitches. For example, if 440Hz and 441Hz were sounded together we would hear a beat frequency of 1Hz. Try this:

```
10 SOUND 1,284,1000,7
20 SOUND 2,283,1000,7
```

Beat frequencies can be heard when playing intervals, too, if the interval is not quite in tune.
We can also experiment with the hardware envelopes. As the envelope period is in multiples of 128 microseconds, some strange effects can be produced by using small EP values. In Program 6.2, REM out line 170 and alter line 160:

```
160 SOUND 1,note,dur,0,1
```

Define a repeating hardware envelope such as:

ENV 1,.=14,1

and listen to the sound. Alter the EP parameter. The effect varies with the pitch of the note. At times it's not unlike an overdriven guitar. When it reaches 50 and over it starts to sound more like a volume envelope.
The other spatial effects are really beyond the capability of the Amstrad's sound chip.

Ring modulation for producing bells

The last synthesiser module we will examine and try to duplicate is the ring modulator. This produces bell-like and metallic sounds and works in a way unlike any of the modules or effects we have covered so far. A standard ring modular requires an input of two frequencies and it produces an output which is a compound of the sum of the two frequencies and the difference between them. To take an example, if it was fed with a frequency of 440Hz and 1220Hz, the output would be a compound of 1660Hz (the sum: 440 + 1220) and 760Hz (the difference: 1200 − 440). The result of such an output is a waveform whose harmonics are not related to each other as they would be in the harmonic series. It can be used to simulate bells and gongs and other such metallic noises.
The Amstrad doesn't have a ring modulator but we can still produce convincing sounds by playing the two calculated frequencies together. The next program will allow you to experiment with various combinations of frequencies which are derived from the ring modulation formula.

```
100 REM PROGRAM 6.3
110 REM Ring Modulation Experiments
120 :
130 ENV 1,1,15,6,15,-1,8
140 :
150 INPUT "1st Frequency";freq1
160 INPUT "2nd Frequency";freq2
170 CLS
180 LOCATE 6,8:PRINT"Freq 1:";freq1
190 LOCATE 25,8:PRINT"Freq 2:";freq2
200 LOCATE 1,10:PRINT"Ring Freq 1:"
210 LOCATE 20,10:PRINT"Ring Freq 2:"
220 GOSUB 370
230 :
```

```
240 WHILE -1
250 com$=INKEY$:IF com$="" THEN 250
260 com=ASC(com$)
270 IF com=59 THEN freq2=freq2+1
280 IF com=47 THEN freq2=freq2-1
290 IF com=97 THEN freq1=freq1+1
300 IF com=122 THEN freq1=freq1-1
310 LOCATE 13,8:PRINT freq1
320 LOCATE 32,8:PRINT freq2
330 GOSUB 370
340 WEND
350 :
360 REM Calculate Frequencies
370 ring1=freq1+freq2
380 ring2=ABS(freq1-freq2)
390 pitch1=ROUND(125000/ring1)
400 pitch2=ROUND(125000/ring2)
410 LOCATE 13,10:PRINT ring1
420 LOCATE 32,10:PRINT ring2
430 :
440 SOUND 129,pitch1,1000,0,1
450 SOUND 130,pitch2,1000,0,1
460 RETURN
```

Commentary

Line 130 defines a suitably bell-like envelope and lines 150 and 160 request two initial frequencies. Try 440 and 1220 to begin with. The sum and difference of the frequencies and the pitch numbers of the resulting frequencies are calculated in lines 370 to 400. Lines 440 and 450 flush channels A and B and play the sounds. You can raise and lower the frequencies using the A and Z keys for Freq 1 and the ; and / keys for Freq 2. You will find some combinations result in pitch numbers outside the sound chip's range and these will produce an improper argument error. You could include lines to print out pitch1 and pitch2.

This isn't true ring modulation but I think you'll agree the results are very effective. The next program shows how this ring modulation effect can be used in a tune.

```
100 REM PROGRAM 6.4
110 REM Ring Modulating a Tune
120 :
```

```
130 ENV 1,1,14,6,14,-1,8
140 :
150 FOR stone=1 TO 25
160 PRINT:PRINT stone;"semitones up"
170 RESTORE
180 FOR tune=1 TO 13
190 READ note,dur
200 :
210 freq1=125000/note
220 freq2=2^(stone/12)*freq1
230 REM freq2=freq1/2^(stone/12):REM Mov
es downwards
240 ring1=freq1+freq2
250 ring2=ABS(freq1-freq2)
260 note1=ROUND(125000/ring1)
270 note2=ROUND(125000/ring2)
280 :
290 PRINT"note1=";note1;" note2=";note2
300 SOUND 1,note1,dur,1,1
310 SOUND 2,note2,dur,0,1
320 NEXT tune
330 NEXT stone
340 END
350 DATA 190,48,213,48,159,48
360 DATA 142,24,106,24,127,48,142,24,106
,24,127,48
370 DATA 159,48,142,48,190,48,213,48
```

Commentary

The program plays the tune 25 times. Each time, the frequency that a note is modulated with increases by a semitone. If you are able to sit through all 25 renditions without a grimace you are indeed dedicated. It does, however, enable you to hear which combinations of frequencies produce the best ring modulation effect. You can then base future experiments on your preferred frequency combination.

The program reads the notes as pitch numbers and converts them back into frequencies at line 210. These will not always be exact but they are near enough for our purpose. Line 220 calculates the second frequency according to a formula discussed in Chapter 4. If line 230 is unREMed and substituted for line 220, the semitones will move downwards. The calculations are similar to those in Program 6.3. The envelope takes the volume up only to level 14.

The SOUND command in line 310 will be slightly quieter than the one in line 300. Normally, the original frequencies do not appear at the output of the modulator. Add the original pitch and listen to the result:

```
315 SOUND 4,note,dur,0,1
```

Bell effects are not used very often on the Amstrad or in computer music so perhaps you will find it worthwile to experiment with them further.

CHAPTER 7
Zaps and Zings and Other Things

The SOUND command's seventh parameter, N, which we have barely mentioned so far, controls the noise output. It is responsible for the unpitched sounds which emanate from the computer. Without it, games would have no bangs or crashes, and it has far more subtle uses in other types of program as we shall see.

Noise is . . . well, noise but there are different kinds. The noise produced by the Amstrad's sound chip is pseudo-random noise which sounds very like white noise.

White Noise

All electronic circuits generate a certain amount of noise and this is generally undesirable. In synthesis this can be used in numerous ways; as a source of unpitched sounds or as an unpitched part of a pitched sound.

White noise is a combination of equal amplitudes of all audio frequencies in the same way that white light is a combination of all colours. If we move up the scale, say one octave from middle C (C0) to the C above (C1), the actual frequency of the note doubles. The frequency doubles every octave we go up so there are more frequencies (not counting fractions) in the higher octaves than in the lower ones. White noise, therefore, tends to contain a lot of high frequencies which is responsible for its characteristic hissing sound.

There are other forms of noise. The second most common form is known as pink noise which contains equal amounts of frequencies from all octaves and is similar to white noise with some of the higher frequencies filtered out. This is useful for producing surf and sea sounds. You can make 'red' noise by filtering out even more high frequencies and various other shades by filtering out other parts of the sound spectrum. The pitch of the noise produced by the sound chip varies with the value of the N parameter as we heard in Chapter 3.

There are two ways noise can be used. It can be used by itself with control over pitch, or in conjunction with some other sound.

Simple sound effects

Program 7.1 contains five examples of sound effects. You can type in the whole program or one section at a time. Each section has been given its own envelopes.

```
100 REM PROGRAM 7.1
110 REM Sound Effects Using Noise
120 :
130 REM Machine Gun
140 FOR burst=1 TO 3
150 FOR bullet=1 TO 12
160 SOUND 1,0,5,7,0,0,13
170 SOUND 1,0,5,7,0,0,18
180 NEXT bullet
190 FOR d=1 TO 1500:NEXT
200 NEXT burst
210 :
220 REM Ricochet
230 ENV 1,1,15,5,5,-1,10,10,-1,5
240 ENT 1,1,-120,1,40,1,3
250 FOR shot=1 TO 3
260 SOUND 1,150,100,0,1,1,15
270 NEXT shot
280 FOR d=1 TO 3500:NEXT
290 :
300 REM Cymbal or Anvil
310 ENV 2,1,15,5,5,-1,10,10,-1,5
320 FOR clang=1 TO 8
330 SOUND 1,16,0,0,2,0,15
340 NEXT clang
350 FOR d=1 TO 6000:NEXT
360 :
370 REM Sleeping Creature
380 ENV 3,7,2,10,5,-1,12,9,-1,6
390 ENV 4,6,2,1,12,-1,20,1,0,100
400 FOR snore=1 TO 5
410 SOUND 1,0,200,0,3,0,28
420 SOUND 1,0,0,0,4,0,31
430 NEXT snore
440 :
450 REM Flying Saucer Taking Off
460 ENV 5,1,15,2,15,-1,50
```

```
470 ENT -2,1,2,2,1,-4,2,1,2,2
480 whine=46
490 FOR engines=31 TO 2 STEP -1
500 whine=whine-1.25
510 SOUND 1,whine,(engines/2)^2,15,0,2,e
ngines
520 NEXT engines
530 SOUND 1,whine,0,0,5,2,2
```

Commentary

The Machine Gun just alternates between two pitches of white noise. This principle can be used to produce a number of effects. REM out line 190 and alter the N parameter in line 160 to 8 and in 170 to 11. Add another delay:

```
175 FOR d=1 TO 90:NEXT
```

to produce the sound of a helicopter's blades. UnREM line 190 and add:

```
174 SOUND 1,0,5,7,0,0,18
```

to produce a car that doesn't want to start. Increase the D parameter in the SOUND commands to make it even more reluctant.

The Ricochet is produced by combining noise with a sound undergoing a pitch drop produced by a tone envelope. The sound produced by the pitch alone might be acceptable as a passing seagull, albeit an electronic one. More of this later.

Cymbal is similar to the Ricochet in that it combines noise with a pitch. Synthesisers usually produce a cymbal sound with a little chuff of white noise with a pine added as here. The result is usually a very electronic-sounding cymbal. If you reduce the duration of the effect, you will get metallic-like clicks. Redefine ENV 2 in line 310:

```
310 ENV 2,1,15,1,5,–1,2,10,–1,2
```

The Sleeping Creature was an attempt to produce an organic sound, i.e. one emanating from a living creature. Perhaps, with a little stretch of the imagination, it might be accepted as a snoring giant.

The Flying Saucer combines sound and noise again, cranking itself up to escape velocity and then vanishing with a shimmer. A little experimentation was necessary to produce the values for whine and engines and to arrive at the duration formula in line 510.

Designing a rhythm unit

After experimenting with the sounds of various guns and vehicles, it is not difficult to produce a rhythm unit. The main problem, and the one requiring the most individual attention, is to produce good drum sounds. The best drum noises need careful setting of the noise pitch and the volume envelope. Try ring modulation for steel drum effects and experiment with variations on the cymbal setting in the last program.
You will find that the duration of the sound plays a very important part in determining the drum characteristics and if you use more than one envelope you can produce quite a reasonable rhythm unit. The next program is one way to approach the design of a rhythm generating program.

```
1000 REM PROGRAM 7.2
1010 REM Rhythm Unit
1020 :
1030 GOSUB 1260:REM Title Page
1040 GOSUB 1400:REM Set Up
1050 :
1060 WHILE -1
1070 READ drum,dur
1080 WHILE drum<>0
1090 com$=INKEY$:IF com$>"" THEN beat$=c
om$
1100 SOUND 1,pitch(drum),dur*tempo,0,ev(
drum),et(drum),noise(drum)
1110 READ drum,dur
1120 WEND
1130 IF beat$="1" THEN RESTORE 1820
1140 IF beat$="2" THEN RESTORE 1840
1150 IF beat$="3" THEN RESTORE 1870
1160 IF beat$="4" THEN RESTORE 1890
1170 IF beat$="5" THEN RESTORE 1920
1180 IF beat$="6" THEN RESTORE 1940
1190 IF beat$="7" THEN RESTORE 1960
1200 IF beat$="8" THEN RESTORE 1980
1210 IF beat$="9" THEN RESTORE 2010
1220 WEND
1230 END
1240 :
1250 REM Title Page
1260 MODE 1
1270 LOCATE 9,3:PRINT "R H Y T H M   U N
```

```
 I T":PRINT
1280 PRINT " 1) Rock 1"
1290 PRINT " 2) Rock 2"
1300 PRINT " 3) Rock 3"
1310 PRINT " 4) Mexican Rock"
1320 PRINT " 5) Fill 1"
1330 PRINT " 6) Fill 2"
1340 PRINT " 7) Fill 3"
1350 PRINT " 8) Swing"
1360 PRINT " 9) Cha Cha"
1370 RETURN
1380 :
1390 REM Set Up
1400 ENV 1,1,15,2,5,-3,4
1410 ENV 2,1,15,1,15,-1,2
1420 ENV 3,1,15,1,4,-2,2,7,-1,4
1430 ENV 4,7,2,1,14,-1,4
1440 ENV 5,1,15,1,15,-1,8
1450 ENT 1,1,-100,1,5,25,3
1460 ENT -2,1,-75,1,4,25,3
1470 ENT -3,1,4,2,1,-8,2,1,4,2
1480 :
1490 DIM pitch(11),ev(11),et(11),noise(1
1)
1500 REM Store Drum Sounds
1510 FOR n=1 TO 11
1520 READ pitch(n),ev(n),et(n),noise(n)
1530 NEXT n
1540 :
1550 REM 1 Tom1
1560 DATA 270,1,2,2
1570 REM 2 Tom2
1580 DATA 340,1,2,8
1590 REM 3 Tom3
1600 DATA 390,1,2,15
1610 REM 4 Tom4
1620 DATA 440,1,2,20
1630 REM 5 Bass Drum
1640 DATA 900,1,1,31
1650 REM 6 Snare
1660 DATA 180,3,1,13
1670 REM 7 Hi Hat
```

```
1680 DATA 0,4,0,1
1690 REM 8 Cymbal
1700 DATA 16,2,0,1
1710 REM 9 Wood Block
1720 DATA 0,1,0,15
1730 REM 10 Guiro
1740 DATA 20,5,3,15
1750 REM 11 Cow Bell
1760 DATA 80,1,0,10
1770 :
1780 tempo=4:beat$="1"
1790 RETURN
1800 :
1810 REM Rock 1
1820 DATA 1,3,1,3,6,3,6,3,5,6,5,6,8,6,6,
3,8,3,5,6,5,3,8,3,0,0
1830 REM Rock 2
1840 DATA 5,6,5,6,6,6,6,3,5,6,5,6,5,3,6,
3,6,3,6,6
1850 DATA 5,6,5,6,6,6,6,3,5,6,5,6,5,3,1,
3,1,3,2,3,2,3,0,0
1860 REM Rock 3
1870 DATA 5,6,8,6,6,3,8,6,6,3,5,6,6,3,8,
3,5,3,6,3,8,3,5,3,0,0
1880 REM Mexican Rock
1890 DATA 10,18,6,12,6,6,5,6,5,6
1900 DATA 10,18,6,12,1,3,1,3,2,3,2,3,4,3
,4,3,0,0
1910 REM Fill 1
1920 DATA 8,6,8,6,6,6,8,6,8,6,6,3,6,3,1,
3,1,3,4,3,4,3,0,0
1930 REM Fill 2
1940 DATA 1,4,1,4,1,4,2,4,2,4,2,4,3,4,3,
4,3,4,4,4,4,4,4,4,0,0
1950 REM Fill 3
1960 DATA 1,3,1,3,1,3,1,3,2,3,2,3,2,3,2,
3,3,3,3,3,3,3,3,3,4,3,4,3,4,3,4,3,0,0
1970 REM Swing
1980 DATA 5,12,7,8,7,4,7,12,7,8,7,4
1990 DATA 7,12,7,8,7,4,7,12,7,8,4,4,0,0
2000 REM Cha Cha
2010 DATA 11,12,11,6,5,6,11,12,11,12
```

```
2020 DATA 11,12,11,6,5,6,11,6,11,6,11,6,
5,6,0,0
```

When run, the program will play a rock beat and pressing the indicated keys will alter the rhythm.

Commentary

The program is fairly self-explanatory. Each drum is produced by a combination of parameters — pitch, volume envelope, tone envelope and noise setting — which are read into four arrays in line 1520. The rhythms are listed in data statements at the end of the program. They are stored as a drum number followed by a duration. The number of each drum is given in the REM statements between lines 1550 and 1760. The rhythm patterns are terminated by two 0s.

The central WHILE/WEND loop between lines 1080 and 1120 reads the drum number and its duration and plays the sound accordingly. If a new key has been pressed, line 1090 stores this in beat$ which is used in lines 1130 to 1210 to restore to the correct set of rhythm data. At the end of a rhythm pattern, drum will be 0 and the program will fall through line 1120 to these lines. Amstrad BASIC does not support

RESTORE line

where line is a variable, hence the list of RESTORE commands. Otherwise we would have been able to restore to a calculated line number and save a little space.

The drum sounds and rhythms would not be out of place in an electro-music composition. Not all the drum sounds have been used in the rhythm patterns. Experiment with the original envelopes to see if you can improve on the sound and construct other drum sounds, too. The effect will be very different if played through an external speaker.

The rhythms only play one or two bars before repeating. You can add more variations by adding to the data and, of course, create more rhythms. You could add a facility to change tempo, too, set in line 1780.

Soundscapes

With such a versatile computer and sound chip there are lots of sound effect collages you could build up: trains, ships, cars, a factory, the countryside, a laboratory or the jungle. The most interesting are ones which will not repeat for a long time or which do not repeat exactly. The next program uses some of

the ideas discussed in this and previous chapters to form a sea soundscape.

```
100 REM PROGRAM 7.3
110 REM Sea, Surf & Seagulls
120 :
130 ENV 1,1,15,3,15,-1,8:REM Seagull
140 ENT 1,20,1,2:REM Seagull
150 ENV 2,5,3,1,2,0,175,15,-1,3:REM Fogh
orn
160 :
170 EVERY 750 GOSUB 410:REM Foghorn
180 ON SQ(1) GOSUB 270:REM Waves
190 :
200 WHILE -1
210 REM Seagulls
220 IF INT(RND*300+1)=1 THEN SOUND 2,INT
(RND*15+1)+15,42,0,1,1
230 IF INT(RND*300+1)=1 THEN SOUND 4,INT
(RND*30+1)+30,42,0,1,1
240 WEND
250 :
260 REM Waves
270 flow=INT(RND*18+1)+2
280 peak=INT(RND*200+1)
290 ebb=INT(RND*90+1)+10
300 wave=INT(RND*25+1)+6
310 lull1=INT(RND*9+1)
320 lull2=INT(RND*4+1)*50
330 ENV 3,15,1,flow
340 ENV 4,1,0,peak,13,-1,ebb,lull1,0,lul
l2
350 SOUND 1,0,0,0,3,0,wave
360 SOUND 1,0,0,15,4,0,wave-6
370 ON SQ(1) GOSUB 270
380 RETURN
390 :
400 REM Foghorn
410 SOUND 162,1073,400,0,2
420 SOUND 148,1074,400,0,2
430 RETURN
```

As it stands, you just run the program, sit back and listen.

Commentary

Most of the program is devoted to generating random numbers in a range which will produce interesting results. Lines 130 to 150 set up envelopes for the seagulls and foghorn. Line 170 uses an interrupt to sound a foghorn every 15 seconds. It means that every 750 fiftieths of a second control will GOSUB to line 410. The foghorn routine uses channel B and C. Line 410 flushes the channel and sends a sound to channel B to rendezvous with channel C. Line 420 flushes the channel and sends a sound to channel C to rendezvous with channel B. The flushes are necessary because, while not making foghorn noises, both channels are making seagull noises. Vary the pitches of the notes to produce different foghorn sounds.

Line 180 sets the sound interrupt, SQ(1) to call the wave routine at line 270. SQ was discussed in Chapter 3. Notice how the interrupt must be re-primed at line 370. This routine repeatedly redefines volume envelopes 3 and 4 and produces different values, wave, to be sent to the noise parameter of the SOUND command. Each wave uses two pitches of noise and the effects produced by the volume envelopes will vary from the quiet to the stormy.

The main loop of the program, between lines 200 and 240 has nothing to do but trigger the seagulls. These are produced by a variation on our Ricochet envelope in Program 7.1. Some of the gulls are a little electronic and you may like to restrict the pitch ranges. Also, change the channel allocations to alter the stereo image.

Further experiments in soundscapes

Although we do not have the facilities of a full-blown synthesiser, we can still produce background effects which can be played throughout a program. An extension to a soundscape could be a graphics design program controlled, possibly, by the random values produced by the soundscape. That should not be too difficult: the Amstrad can produce an enormous range of sound and graphic effects.

CHAPTER 8
Playing the Amstrad

Most musical instruments are designed to be easy, ergonomically, to play. A piano keyboard is an example.
Computers are not normally supplied with a musical keyboard. If we want to 'play' the computer we must make do with what we have, i.e. the QWERTY typewriter keyboard. Depending on your musical upbringing, you may find this easy or difficult to adapt to. We can still, however, have a lot of fun using the computer in this way.

Monophonic and polyphonic instruments

A monophonic instrument is one which can only play one note at a time. Most instruments fall into this category, and although it is technically possible to play more than one note on some of them, they are generally classed as monophonic.
A polyphonic instrument is one which can sound many notes at once, and usually all of them if required, such as the piano, organ or harp.
You will often see synthesisers described as monophonic or polyphonic. Sometimes the polyphonic category is qualified by a number such as 6-note or 8-note polyphonic. Some monophonic synthesisers have a duophonic mode which means they can sound two notes at once. With the ever-decreasing cost of electronics and silicon chips, the trend is towards producing instruments with ever greater polyphonic capabilities.

The Amstrad as a monophonic instrument

There is more than one way of writing a program which allows us to play music from the keyboard. The next program illustrates just one way in which it can be approached and it turns the computer into a monophonic keyboard.

```
100 REM PROGRAM 8.1
110 REM Monophonic Keyboard
120 REM From F#-1 to D#1
130 :
```

```
140 ON BREAK GOSUB 270
150 SPEED KEY 255,255
160 keyboard$="1q2w3er5t6yu8i9o0p@^["+CH
R$(16):REM Last note=CLR key
170 ENV 1,1,15,1,2,-1,2,13,-1,10
180 :
190 WHILE -1
200 play$=INKEY$:IF play$="" THEN 200
210 note=INSTR(keyboard$,play$)-6
220 freq=440*(2^(0+(note-10)/12))
230 pitch=ROUND(125000/freq)
240 IF pitch<677 THEN SOUND 129,pitch,0,
0,1
250 WEND
260 :
270 SPEED KEY 20,3
280 STOP
```

Commentary

The program is so simple it could probably be condensed into a couple of lines. Line 150 slows down the repeat rate on the keys. Line 140 sees that they return to normal when you press ESC.

The variable keyboard$ contains the keys we use and each key from left to right increases the pitch by a semitone. If you refer to Figure 2.4 they correspond to the notes from F#–1 to D#1. Input is detected by INKEY$ and checked with the INSTR function to produce the note number, note. 6 is deducted from note in line 210 so the "1" key will play F#–1. Other offsets could be used to make the keyboard begin on another note. From note, freq is calculated and then the pitch number. If the pitch is within range a sound is made at line 240. If a key not included in keyboard$ is pressed, note will have a value of –1, pitch will have a value of 716 and no sound will be made. The SOUND command contains a flush instruction so each new note sounds immediately upon being played.

The Amstrad synthesiser

From the humble beginnings of our last program, we can add a few subroutines to produce a range of synth effects.

```
1000 REM PROGRAM 8.2
1010 REM Amstrad Synth
```

```
1020 REM From F#-1 to D#1
1030 :
1040 ON BREAK GOSUB 1900
1050 SPEED KEY 255,255
1060 ENV 1,1,15,1,2,-1,2,13,-1,10
1070 ENV 2,16,15,2
1080 ENV 3,15,1,1,4,-1,10,11,-1,25
1090 ENV 4,1,15,17,8,7,17,1,-7,17,1,5,17
,2,-2,17
1100 ENT -1,1,1,2,1,-2,2,1,1,2
1110 ENT -2,2,1,2,4,-1,2,2,1,2
1120 ENT 3,1,100,1,10,-10,1
1130 keyboard$="1q2w3er5t6yu8i9o0p@^["+C
HR$(16):REM Last note=CLR key
1140 oct=0:ev=1:et=0
1150 GOSUB 1630:REM Draw Keyboard
1160 GOSUB 1560:REM Print Envelope Arrow
s
1170 :
1180 WHILE -1
1190 play$=INKEY$:IF play$="" THEN 1190
1200 IF play$=CHR$(240) OR play$=CHR$(24
1) THEN GOSUB 1360:GOTO 1190:REM Octave
1210 REM note-6 makes "1" sound F#-1
1220 note=INSTR(keyboard$,play$)-6
1230 IF note>-6 THEN GOSUB 1290:GOTO 119
0:REM Play Note
1240 IF play$="	" THEN GOSUB 1430:GOTO 11
90:REM TAB Key Toggles Chorus
1250 GOSUB 1480:REM Alter Envelopes
1260 WEND
1270 :
1280 REM Play Note
1290 freq=440*(2^(oct+(note-10)/12))
1300 pitch=ROUND(125000/freq)
1310 SOUND 129,pitch,0,0,ev,et
1320 IF chorus THEN SOUND 130,pitch-1,0,
0,ev,et
1330 RETURN
1340 :
1350 REM Change Octave
1360 IF play$=CHR$(240) THEN oct=oct+1 E
```

```
LSE oct=oct-1
1370 IF oct<-2 THEN oct=3
1380 IF oct>3 THEN oct=-2
1390 LOCATE 38,16:PRINT oct;
1400 RETURN
1410 :
1420 REM Chorus
1430 chorus=NOT chorus
1440 LOCATE 29,16:PRINT ABS(chorus);
1450 RETURN
1460 :
1470 REM Alter Envelopes
1480 IF play$="z" THEN ev=1
1490 IF play$="x" THEN ev=2
1500 IF play$="c" THEN ev=3
1510 IF play$="v" THEN ev=4
1520 IF play$="a" THEN et=0
1530 IF play$="s" THEN et=1
1540 IF play$="d" THEN et=2
1550 IF play$="f" THEN et=3
1560 LOCATE 1,21:PRINT SPC(19)
1570 LOCATE 4+5*et,21:PRINT CHR$(240)
1580 LOCATE 1,22:PRINT SPC(19)
1590 LOCATE 4+5*(ev-1),22:PRINT CHR$(241
)
1600 RETURN
1610 :
1620 REM Draw Keyboard
1630 MODE 1
1640 INK 0,13:INK 1,0
1650 k1$=" "+CHR$(143)+CHR$(143)
1660 k2$=k1$+k1$
1670 k3$=k2$+k1$+"  "+CHR$(211)
1680 k4$=k3$+k2$+"  "+CHR$(211)+k3$+k2$
1690 k5$="  "+CHR$(211):FOR n=1 TO 13:k6
$=k6$+k5$:NEXT
1700 PRINT:PRINT " 1  2  3     5  6
8  9  0     ^"
1710 FOR n=1 TO 5:PRINT k4$:NEXT
1720 FOR n=1 TO 6:PRINT k6$:NEXT
1730 PRINT CHR$(11)+CHR$(22)+CHR$(1);"
";
```

```
1740 FOR n=1 TO 36:PRINT CHR$(210);:NEXT
1750 PRINT CHR$(11);CHR$(11);"   Q  W  E
  R  T  Y  U  I  O  P  @  ["
1760 PRINT CHR$(22)+CHR$(0)
1770 :
1780 PRINT:PRINT TAB(35);CHR$(240)
1790 PRINT TAB(34);"oct:";oct;
1800 PRINT TAB(35);CHR$(241)
1810 :
1820 LOCATE 18,16:PRINT "TAB chorus: 0"
1830 :
1840 LOCATE 1,19:PRINT "   A    S    D
  F"
1850 PRINT " ENT0 ENT1 ENT2 ENT3"
1860 PRINT:PRINT:PRINT "   Z    X    C
  V"
1870 PRINT " ENV1 ENV2 ENV3 ENV4"
1880 RETURN
1890 :
1900 ON BREAK STOP
1910 SPEED KEY 20,3
1920 STOP
```

When run, the program draws a piano keyboard on the screen and shows the corresponding keys on the computer. The octave is altered with the up and down cursor keys, TAB toggles a chorus effect on and off, A, S, D and F select tone envelopes and Z, X, C and V select volume envelopes.

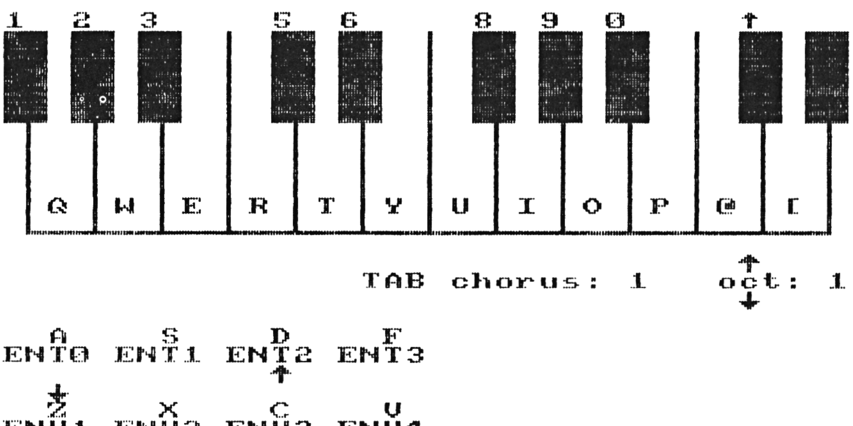
Commentary

The program begins by slowing down the repeat speed of the keys and defining the envelopes we are going to use. Keyboard$ is the same as in the last program but now we can change octaves and envelopes so line 1140 sets default values.

Line 1150 calls the routine at 1630 which draws the keyboard. It is made up from six strings which are constructed in lines 1650 to 1690 and drawn in lines 1700 to 1760. The following lines print the other 'controls' on the screen. Transparency is turned on in line 1730 and off in line 1760 to allow the letters to be printed along the bottom of the keyboard.

The routine at line 1560 which is called by line 1160 is actually the end of the routine which handles the envelope selection. It is called to print the arrow pointers in the correct place.

Figure 8.1: Screen dump from Program 8.2



The main loop lies between lines 1180 and 1260. Line 1190 waits for a key to be pressed. If it is a cursor key it calls the octave change routine. If it is a key in keyboard$, note will have a value greater than –6 and line 1230 will call the play note routine. If TAB has been pressed line 1240 calls the chorus routine. The other control keys alter the envelopes and these are checked for within the envelope alteration routine at line 1480 which is called if the program reaches line 1250.

The subroutines should need little explanation. Lines 1290 to 1300 calculate frequency and pitch as usual. Line 1320 brings in a second voice, slightly out of tune, if chorus is on. Lines 1360 to 1390 take care of octave changes. The octave will wrap around if you try to take it out of range. The chorus routine switches the variable, chorus, between 0 (off) and –1 (on). We print 1 for on in line 1440 as opposed to –1 because it looks neater. Lines 1480 to 1550 alter the envelopes. Lines 1570 and 1590 calculate the position of the arrows which point to the selected envelopes.

When playing a strange instrument it is often helpful to have a diagram of the keyboard with the relevant buttons or keys marked on. You may also find it useful to stick small pieces of paper over the unused keys to make the QWERTY layout look a little more like a piano keyboard.

The effects are all quite short and most of the ideas and principles have been discussed in previous chapters. The control keys were chosen because most people are righthanded. They can be easily altered if desired. Add more effects. More envelopes can be added — even all 15 — and you could include the option to add noise, too. You could add a sub-octave switch. These can be found on many synthesisers and they sound a note an octave lower than the

one you're playing to thicken the sound.
If you want to try some dazzling fingerwork, you may find the response a little slow. This tiny delay is not likely to be a problem but you can cut the response time by compressing the coding, using multi-statement lines and single letter integer variables.

Adding a bass sequencer

A sequencer is a device which can be programmed with a set of notes and used to control a synthesiser module to produce a repeating riff or sequence of notes. Sequencers vary in their sophistication — some can only store a dozen notes, others can store several thousand.
Most synthesisers control pitch by voltage — the higher the voltage the higher the pitch — so many sequencers actually store a list of voltages. These can be used to control any voltage-controllable part of the synthesiser. Most synthesiser modules are designed to be governed by voltages and are called such things as VCO (Voltage Controlled Oscillator), VCA (Voltage Controlled Amplifier) and VCF (Voltage Controlled Filter). Every time we use a DATA statement to read in a string of notes we are, in effect, using a sequencer.
We can add a Bass Sequencer to the last program, which will play a riff and allow us to improvise over the top of it. This is common in electronic music and the use of sequencers has been popularised by such musicians as Tangerine Dream, Kraftwerk and Jean-Michel Jarre. The next listing shows the lines we must add to Program 8.2.

```
10 REM PROGRAM 8.3
20 REM Bass Sequencer
30 REM Add these lines to PROGRAM 8.2
40 :
50 tempo=4
60 ENV 15,1,15,1,1,-2,2,13,-1,5
70 ENT -15,2,3,1,2,-6,1,2,3,1
80 ON SQ(4) GOSUB 2010
90 :
2000 REM Sequencer Routine
2010 READ pit,dur
2020 IF pit=0 THEN RESTORE 2070:GOTO 201
0
2030 SOUND 4,pit,dur*tempo,0,15,15
2040 ON SQ(4) GOSUB 2010
2050 RETURN
```

```
2060 :
2070 DATA 804,6,758,6,956,6,1073,12,1073
,12,1136,6,1073,12,1073,6,1136,6,1073,6,
1136,6,1073,6,956,6
2080 DATA 804,6,758,6,956,6,1073,12,1073
,12,1136,6,1073,12,1073,6,1136,6,1073,24
,0,0
2090 DATA 638,6,536,6,402,6,426,6,0,0
2100 DATA 638,24,536,24,568,24,602,24,0,
0
2110 DATA 638,6,638,6,536,6,47
```

Commentary

The sequencer uses ENV 15 and ENT 15 which are defined in lines 60 and 70. The main routine, from lines 2010 to 2050 is as simple as can be — read note and play it. It uses ON SQ GOSUB which is primed in line 80 and rearmed at line 2040. The program will run by itself if you add:

85 WHILE –1:WEND

Sequence data is terminated by two 0s.

Altering the bass riff

Extra sequences have been included in lines 2090 to 2110 for you to experiment with and new ones can easily be added. Try the sequence in line 2100 with a longer envelope such as:

ENV 15,1,15,1,1,–2,10,13,–1,15

Alter the tone envelope to suit. The riffs in the program are quite simple but could be extended to any length by adding more data. You could even program the sequencer to play the complete bass line of a tune.

Developing the sequencer

You could add a rhythm to the sequencer based upon the ideas in Program 7.2 but you would lose the chorus facility. This would be quite interesting.
Apart from adding more envelopes, rhythms, sequences and effects, you might like to put a memory capability into the program so it remembers what you have played and can play it back. This could be very useful. If you are

improvising and play a good sequence of notes it is not always easy to remember later what notes you played.
To accomplish this it would be necessary to include an incrementing variable in the program and to record the variable's value whenever a key is pressed or released. The notes would be stored in an array, the variable providing the subscript. Taking the program even a stage further, if you added editing facilities, allowed envelope changes during playback and had facilities for altering pitch and tempo you would be on the way to a miniature recording console. It is certainly worthwhile as an exercise. If you want to control and alter music to such a fine degree, the difficulty of playing the QWERTY keyboard does not make your job easy. Perhaps a better and more suitable solution is to preprogram the entire piece and let the computer manage the difficult passages. This is what we look at in the next chapter.

CHAPTER 9
Making Micro Music

After experimenting with the programs in Chapter 8 you will probably have come to the conclusion that the QWERTY keyboard is not the easiest musical instrument to play. It's fine for improvising and playing along with a rhythm or bass pattern but playing tunes accurately in real time is extremely difficult. Even with a program which would allow us to play two or three notes simultaneously, we would have to consider the physical difficulties of actually playing the right ones.

We can overcome these problems by providing the computer with all necessary information regarding pitch, envelopes and durations, and let it carry out the hard work of putting them together. It can play sequences and harmonies we would never be able to manage and it will play them right every time.

The simplest and most obvious method is to read the information into the program through DATA statements, but it soon becomes quite complicated when you have more than say 30 or 40 items of data. We need a method of entering notes in a form which is easier to use than a list of numbers. Let us look at some options.

Talking music

Standard music notation allows us to communicate our musical ideas to anyone who speaks the same language. The Amstrad has not yet been schooled in music, so we have to devise our own method of communicating musical notation to it. There are several methods of converting from a notation which we find easy to understand, to one which the computer can accept.

The SOUND command, for example, uses simply a list of numbers each of which corresponds to a particular pitch. Other methods, not using numbers, are all variations on a theme. The idea is that we input notes in a form we find easy to understand and let the computer convert them to numbers it can understand. The variations lie in the way we initially code the notes.

There already exist several forms of notation for specifying notes without using a stave. Most consist of the note name followed by a number to

represent the octave such as C1, B5, F#3 etc. Other systems employ a set of apostrophe marks (') to indicate the octave such as C' or A'. Some systems place lines horizontally, directly over the note, some use roman numerals to indicate the octave and some count in semitones so that C1 would be followed by C#2. I think the method which is easiest to understand, to enter and to edit is the octave number version.

Appendix VII of the User Guide contains an octave number system which puts middle C in octave 0. Other numbering systems use a superscript and subscript system which designates middle C simply as C and the octaves above as C^1, C^2 etc. and the octaves below as C_1 and C_2.

In Figure 2.4 and in Appendix 1 of this book you will also see another set of figures. My own preference would be to use these figures to number the octaves, for two reasons. Firstly, the maximum length of a note name would be three characters. With negative octave numbers, the maximum is four characters, e.g. C#–1. Secondly, because of this, the note analysis procedure which converts a note name into a pitch number would be slightly shorter. The advantages would be, therefore, less memory would be required and note data would be shorter and easier to enter. Many people will now be familiar with the User Guide's system of octave numbering, however, so I felt

Figure 9.1: The rests in brackets are implied and refer to Channel B.

it best to conform to this system. If other writers and programmers do likewise, there may be a certain amount of compatibility between Amstrad music programs and ideas.

The system described below can be adapted to any other notation system you feel more comfortable with. Chapter 4 lists some formulae to use with alternative systems. Data is entered as a note name plus octave number and a duration. This method of note storage gives us immediate visual and physical access to the information which is saved along with the program.

Note durations

The duration of a note will normally be a multiple of the length of the shortest note in the piece. In the example in *Figure 9.1* the shortest note is a sixteenth. This could have been given a duration of 1, but has, in fact, been given a value of 3. This gives us a time base which allows us to program triplet timings, for example. If sixteenths have a duration of 3, a crochet or quarter note will have a duration of 12. To fit three notes (triplets) into a crotchet time, each will have a duration of 4. Look at Program 7.2 and the data for Fill 2 and Fill 3 for examples of this. Quaver triplets would have durations of 2.

Note to number conversion program

This is based upon the User Guide's notation illustrated in Figure 2.4 and in Appendix 1.

```
100 REM PROGRAM 9.1
110 REM Mozart's Rondo Alla Turca
120 REM Using 1 Channel
130 :
140 scale$=" C C#D D#E F F#G G#A A#B"
150 tempo=4
160 ev1=1
170 ENV 1,1,15,1,3,-1,4,12,-1,8
180 :
190 FOR n=1 TO 46
200 READ note$,dur
210 GOSUB 270:REM Analyse Note
220 SOUND 1,pitch,dur*tempo,0,ev
230 NEXT n
240 END
250 :
260 REM Analyse Note
```

```
270 IF note$="R" THEN ev=0:RETURN ELSE e
v=ev1
280 length=LEN(note$):IF length<2 OR len
gth>4 THEN PRINT "Error in Note";n;note$
:STOP
290 IF length=2 THEN name$=LEFT$(note$,1
):oct=VAL(RIGHT$(note$,1)):GOTO 320
300 IF length=4 THEN name$=LEFT$(note$,2
):oct=VAL(RIGHT$(note$,2)):GOTO 320
310 IF MID$(note$,2,1)="-" THEN name$=LE
FT$(note$,1):oct=VAL(RIGHT$(note$,2)) EL
SE name$=LEFT$(note$,2):oct=VAL(RIGHT$(n
ote$,1))
320 note=INSTR(scale$,name$)/2
330 freq=440*(2^(oct+(note-10)/12))
340 pitch=ROUND(125000/freq)
350 RETURN
360 :
370 DATA B0,3,A0,3,G#0,3,A0,3
380 DATA C1,6,R,6,D1,3,C1,3,B0,3,C1,3
390 DATA E1,6,R,6,F1,3,E1,3,D#1,3,E1,3
400 DATA B1,3,A1,3,G#1,3,A1,3,B1,3,A1,3,
G#1,3,A1,3
410 DATA C2,12,A1,6,C2,4,G1,1,A1,1
420 DATA B1,6,A1,6,G1,6,A1,4,G1,1,A1,1
430 DATA B1,6,A1,6,G1,6,A1,4,G1,1,A1,1
440 DATA B1,6,A1,6,G1,6,F#1,6
450 DATA E1,12
```

The program plays the first eight bars of Mozart's Rondo Alla Turca. Figure 9.1 shows a complete three channel version which we will program later.

Commentary

Line 140 defines scale$ which contains the notes of the scale. The spaces are important. It is worth pointing out that the black notes are all written as sharps. It is generally not a problem for us to enter notes in this way although if the music contains flats you will have to do a mental conversion to sharps. Figure 2.4 will help. Also, the keyboard has a ready-made sharp sign (#), the use of which avoids any possible confusion between the flat sign and the letter b.

The main program loop between lines 190 and 230 reads the note and plays it, but we make a call to the routine at line 270 to analyse the note. You will have noticed the two small notes at the beginning of bars five and six. Without getting bogged down in more musical theory, the small notes are to be played very quickly before the first 'real' note of the bar. They are called acciaccatura (from the Italian meaning 'crushed in'). Using the time base outlined above we can give them a duration of 1.

If note$ is an R, this signifies a rest and ev is set to 0. Otherwise, the note is analysed as follows: note$ can be two, three or four characters long. No check is made on whether a character is a sharp or flat and the program could be adapted to cater for the input of both sharps and flats if this suits you better. However, the method used above will be a little shorter as it only makes a check on the length of note$. If note$ is two charcters long, the first will be the note name and the second will be the octave. Line 290 takes care of this. If note$ is four characters long, the first two will be the note name and the second two will be the octave. Line 300 takes care of this. If note$ is three characters long there are two possibilities. It could be a note with an accidental in a positive octave or a note without an accidental in a negative octave. Line 310 looks at the middle character and note$ is split up accordingly. The note number is extracted from scale$ in line 320 and then the frequency and pitch are calculated. The routine returns and the pitch is played.

Using the alternative octave numbering system

With only positive octaves, the analysis routine could be reduced to a couple of lines:

```
IF LEN(note$)=2 THEN name$=LEFT$(note$,1) ELSE
name$ =LEFT$(note$,2)
oct=VAL(RIGHTnote$,1))
```

We could then rewrite the freq formula in line 330 or adjust oct (by subtracting 4) in the above example.

Error routines

You may find it helpful to include error routines which will report instances of incorrect data. Line 280 incorporates one such check. The data is arranged in lines of one bar to aid entering, reading and debugging.

Playing two-part or three-part tunes

If the notes of all the channels have the same duration, a simple variation on the last program will read and play two-part and three-part tunes. Problems arise, however, when each channel contains notes of different durations. From Figure 9.1 you can see that by the time we reach the end of bar 3, 24 notes will have passed through channel A and only 13 through the other channels. If we tried to queue these note for note, the program would be held up waiting for the longer notes to clear. The solution is to store the notes in arrays from which we can read and play a note when it's required. The ON SQ and rendezvous commands make our task quite easy.

Stereo positioning

I normally arrange the channels as follows:

Channel A: Melody line.
Channel C: Bass line.
Channel B: Anything between, often reinforcing the bass/accompaniment.

If the piece has a prominent bass line I sometimes program that first into channel A.
This method tries to take into account the stereo positions of the voices. As some channels, for example B, may be used to play more than one part the stereo image may sometimes sound strange. The use of stereo is very effective and if you can connect the output to stereo speakers it is certainly worthwhile, not least for the increase in sound quality which larger speakers will bring. Even plugging in a pair of stereo headphones will enable you to appreciate the stereo effect. Generally, the melody will come from one side, the bass from the other and the middle parts from the centre.
The three voices are stored as independent sets of data and you can program any channel to play any voice. To experiment with stereo placings, read the data into different channels. The melody and bass lines are the most important and set the character of the piece. I use channel B to fill in the harmony where required and it can switch from bass to melody as necessary. With this sort of arrangement, channels A and C will be dealt with before channel B, which may seem strange when you look through the listing.
The next program gives us control over five elements of each note. In practice, it is possible to employ some time-saving procedures — which I have done in certain cases — and you will find that you rarely have to enter five data items for each note.

```
1000 REM PROGRAM 9.2
1010 REM Mozart's Rondo Alla Turca
1020 REM Using 3 Channels
1030 :
1040 va=46:vb=29:vc=30
1050 :
1060 REM 1st Subscript refers to:
1070 REM 1=status
1080 REM 2=pitch
1090 REM 3=dur
1100 REM 4=ENV
1110 REM 5=ENT
1120 :
1130 DIM chana(5,va)
1140 DIM chanb(5,vb)
1150 DIM chanc(5,vc)
1160 :
1170 scale$=" C C#D D#E F F#G G#A A#B"
1180 tempo=4
1190 ENV 1,1,15,1,3,-1,4,12,-1,8
1200 ENT 1
1210 ENV 2,3,5,1,1,0,8,1,-2,4,13,-1,8
1220 ENT -2,1,1,3,1,-2,3,1,1,3
1230 ENV 3,1,15,1,3,-1,2,12,-1,6
1240 ENT 3
1250 :
1260 CLS:PRINT "Reading in note data..."
1270 FOR n=1 TO va
1280 GOSUB 1740:REM Check Channel Status
1290 chana(1,n)=status+1
1300 GOSUB 1780:REM Analyse Note
1310 IF note$="R" THEN ev=0 ELSE IF n=5
OR n=11 OR n=25 OR n=30 OR n=36 OR n=42
THEN ev=2 ELSE ev=1
1320 et=ev
1330 chana(2,n)=pitch
1340 chana(3,n)=dur
1350 chana(4,n)=ev
1360 chana(5,n)=et
1370 NEXT n
1380 PRINT "Channel A Complete"
1390 :
```

```
1400 FOR n=1 TO vc
1410 GOSUB 1740:REM Check Channel Status
1420 chanc(1,n)=status+4
1430 GOSUB 1780:REM Analyse Note
1440 IF note$="R" THEN ev=0 ELSE ev=3
1450 et=ev
1460 chanc(2,n)=pitch
1470 chanc(3,n)=dur
1480 chanc(4,n)=ev
1490 chanc(5,n)=et
1500 NEXT n
1510 PRINT "Channel C Complete"
1520 :
1530 FOR n=1 TO vb
1540 GOSUB 1740:REM Check Channel Status
1550 chanb(1,n)=status+2
1560 GOSUB 1780:REM Analyse Note
1570 IF note$="R" THEN ev=0 ELSE ev=1
1580 et=ev
1590 chanb(2,n)=pitch
1600 chanb(3,n)=dur
1610 chanb(4,n)=ev
1620 chanb(5,n)=et
1630 NEXT n
1640 PRINT "Channel B Complete"
1650 :
1660 voca=0:vocb=0:vocc=0
1670 ON SQ(1) GOSUB 1890
1680 ON SQ(2) GOSUB 1950
1690 ON SQ(4) GOSUB 2010
1700 WHILE voca<va OR vocb<vb OR vocc<vc
:WEND
1710 END
1720 :
1730 REM Check Channel Status
1740 READ note$:IF LEFT$(note$,1)="&" TH
EN status=VAL(note$):READ note$,dur ELSE
 status=0:READ dur
1750 RETURN
1760 :
1770 REM Analyse Note
1780 IF note$="R" THEN pitch=0:RETURN
```

```
1790 length=LEN(note$):IF length<2 OR le
ngth>4 THEN PRINT "Error in Note";n;note
$:STOP
1800 IF length=2 THEN name$=LEFT$(note$,
1):oct=VAL(RIGHT$(note$,1)):GOTO 1830
1810 IF length=4 THEN name$=LEFT$(note$,
2):oct=VAL(RIGHT$(note$,2)):GOTO 1830
1820 IF MID$(note$,2,1)="-" THEN name$=L
EFT$(note$,1):oct=VAL(RIGHT$(note$,2)) E
LSE name$=LEFT$(note$,2):oct=VAL(RIGHT$(
note$,1))
1830 note=INSTR(scale$,name$)/2
1840 freq=440*(2^(oct+(note-10)/12))
1850 pitch=ROUND(125000/freq)
1860 RETURN
1870 :
1880 REM Channel A
1890 voca=voca+1:IF voca>va THEN RETURN
1900 SOUND chana(1,voca),chana(2,voca),c
hana(3,voca)*tempo,0,chana(4,voca),chana
(5,voca)
1910 ON SQ(1) GOSUB 1890
1920 RETURN
1930 :
1940 REM Channal B
1950 vocb=vocb+1:IF vocb>vb THEN RETURN
1960 SOUND chanb(1,vocb),chanb(2,vocb),c
hanb(3,vocb)*tempo,0,chanb(4,vocb),chanb
(5,vocb)
1970 ON SQ(2) GOSUB 1950
1980 RETURN
1990 :
2000 REM Channel C
2010 vocc=vocc+1:IF vocc>vc THEN RETURN
2020 SOUND chanc(1,vocc),chanc(2,vocc),c
hanc(3,vocc)*tempo,0,chanc(4,vocc),chanc
(5,vocc)
2030 ON SQ(4) GOSUB 2010
2040 RETURN
2050 :
2060 REM Voice 1
2070 DATA &38,B0,3,A0,3,G#0,3,A0,3
```

```
2080 DATA C1,6,R,6,D1,3,C1,3,B0,3,C1,3
2090 DATA E1,6,R,6,F1,3,E1,3,D#1,3,E1,3
2100 DATA B1,3,A1,3,G#1,3,A1,3,B1,3,A1,3
,G#1,3,A1,3
2110 DATA C2,12,A1,6,C2,4,G1,1,A1,1
2120 DATA B1,6,A1,6,G1,6,A1,4,G1,1,A1,1
2130 DATA B1,6,A1,6,G1,6,A1,4,G1,1,A1,1
2140 DATA B1,6,A1,6,G1,6,F#1,6
2150 DATA E1,12
2160 :
2170 REM Voice 2
2180 DATA &38,R,12
2190 DATA A-1,6,C0,6,C0,6,C0,6
2200 DATA A-1,6,C0,6,C0,6,C0,6
2210 DATA A-1,6,C0,6,A-1,6,C0,6
2220 DATA A-1,6,C0,6,C0,6,C0,6
2230 DATA E-1,6,B-1,6,B-1,6,B-1,6
2240 DATA E-1,6,B-1,6,B-1,6,B-1,6
2250 DATA E-1,6,B-1,6,B-2,6,B-1,6
2260 DATA E-1,12
2270 :
2280 REM Voice 3
2290 DATA &38,R,12
2300 DATA R,6,E0,6,E0,6,E0,6
2310 DATA R,6,E0,6,E0,6,E0,6
2320 DATA R,6,E0,6,R,6,E0,6
2330 DATA R,6,E0,6,E0,6,E0,6
2340 DATA R,6,E0,6,E0,6,E0,6
2350 DATA R,6,E0,6,E0,6,E0,6
2360 DATA R,6,E0,6,R,12
2370 DATA R,12
```

Commentary

The variables pertaining to each channel are suffixed with their channel letter. I usually refer to the music parts as voices 1, 2 and 3 or A, B and C. The former seems more natural but the latter tells us which channel the voice will be played on. The variables va, vb and vc hold the number of notes in each voice. They refer to the number of notes per channel, not the number of data items. Three arrays are dimensioned at lines 1130 to 1150 to hold information about each note. The first subscript refers to the following aspects of the note:

Subscript = 1: Status, e.g. channel number, rendezvous, and so on.
Subscript = 2: Pitch number.
Subscript = 3: Duration.
Subscript = 4: ENV number.
Subscript = 5: ENT number.

The tempo is set and envelopes are defined. The program links the volume and tone envelopes together, so if a tone envelope is not defined it is good practice to reset it, in case a definition has been left over from a previous program.
The next section sets the data into relevant arrays. The process is repeated once for each channel. This way is easier to understand but you might like to substitute a single array such as chan(3,5,n) to avoid repetition. N would be the maximum number of notes in any channel, in this case equal to va. As the process is exactly the same for each channel we will only look at channel A in detail.
The screen is cleared and a message printed to let you know the computer is doing something. A FOR . . . NEXT loop runs through the data, once for each note. The first step is a call to the routine at line 1740. This examines the first item of data. If it begins with an ampersand (&) it knows it is a channel instruction and evaluates it to produce the channel status. It then reads note$ and dur. I have used hex notation to represent channel status because it is relatively easy to use and the ampersand shows at a glance that an item is a status value. You could use decimal notation and ignore the ampersand. See Appendix II in the User Guide for further details of hex notation.
If the data item does not begin with an ampersand it is taken to be a note and a second time, dur, is read. As these have no specific status value, status is set at 0. This method of assigning status values saves us having to enter a value for each note. The channel status is assigned in line 1290 where the channel number is added, in this case, 1. A call is then made to line 1780 to analyse the note. This is the same routine as the one in Program 9.1, with one small change. If note$ is a rest, pitch is given the arbitrary value of 0. This is not strictly necessary but it may help in debugging. Next, note$ is examined to see if it is an 'R', which I have used to represent a rest. If it is, the volume envelope, ev, is set to 0. This produces a sound at a volume determined by the V parameter of the SOUND command which, under envelope control, we set to 0. Otherwise, ev is set to the required envelope number. You can see this more clearly in lines 1440 and 1570 where only one other envelope is used. Line 1310 is arranged to give certain notes certain envelopes. This avoids having to include an envelope number with the note data. The tone envelope, et, is set to equal the volume envelope in line 1320. A similar routine to the one in line 1310 could be used to vary the tone envelope.

The lines following copy all this information into the relevant arrays. Line 1380 lets us know the channel data has been stored.
At the end of these three sections, the arrays are filled with figures which the SOUND command can work on directly.
Line 1660 sets three variables which are used to check the number of notes sent to each channel. The next three lines prime the ON SQ commands. As the first notes are all set to rendezvous, the tune will begin in sync. By the time the sound chip has played the first notes, the computer should have processed a few more, keeping the buffers full. An alternative way to start the tune would be to hold the channels, fill them with notes and then issue a RELEASE command.
The routines at lines 1890, 1950 and 2010 are similar. Each time a routine is called, it adds 1 to the value of the voc variable. If this is greater than the number of notes in the channel, the routine returns and, not having reprimed itself, is not called again. The next line simply issues the SOUND command and the following line reprimes the interrupt. Even at this stage, we control the speed of the piece with tempo.
The WHILE/WEND loop at line 1770 does nothing. It just keeps the program running while the interrupts do the work. You can alter the tempo (if you wish) and play the tune again by entering:

```
GOTO 1660
```

Modification and suggestions

Once you have this program you will be able to play almost any three-part tune by inserting new data and altering a few variables. If you arrange the data in lines of one bar it will help when editing and debugging. When programming other tunes, if all the channels do not start together be sure to include rests in the channels which do not begin immediately.
You will notice that only the very first notes have been rendezvoused. Normally, this will be enough to keep most tunes together but you could add more rendezvous commands if required. The ON SQ interrupts allow BASIC to process other commands while sounds are being played. In this program, the looping at line 1700 is wasted.
Envelope changes are often tedious to enter. If your program requires a lot of envelopes you could arrange a separate routine to take care of this.
When calculating the note durations, it is important to get the relationship between the notes correct. Adjustments to the variable, tempo, can be effective, too.

Debugging the data

You will be lucky if you enter a set of data statements for a tune and get no errors. The note analysis routine will detect a simple note length error. If the program tries to read a string into dur, it will report an error. You will know which channel the error is in because a message is printed when a channel has been successfully filled with notes.
An easy mistake to make is to insert wrong va, vb or vc values, causing one channel to fill with another channel's notes. You could provide further checks by inserting a termination character on the end of each channel's data. You could use this principle to read through the data to find the number of notes in each channel and use this information to dimension the arrays. Remembner, though, that upon exit from a FOR/NEXT loop, the loop counter is incremented so for example, after line 1370, n will equal 47. Once the data is correct, you can remove the error routines.

Saving the tune

You can use the program to check the note data and then save the information to tape or disc. This means the data does not have to be stored twice — in the data statements and in the arrays — and there will be no note analysis routines.
Tape files will take longer to load than the original program takes to analyse the tune, so you do not actually save any time. The main program must hold the ENV and ENT settings and the play routine — which you could condense — so you can save quite a lot of memory. You could build a library of tunes saved as files and load and play any one of them. The main loading program sets the envelope parameters but you could add a routine to save these along with the note data. A file would then load its own 'orchestra' and play the tune.
The file can be saved quite simply by LOADing Program 9.2 and adding these lines.

```
10 REM PROGRAM 9.3
20 REM Saving Tune Data to a File
30 REM Insert These Lines
40 REM In PROGRAM 9.2
1711 :
1712 OPENOUT "RONDO"
1713 WRITE #9,tempo,va,vb,vc
1714 FOR n=1 TO va:FOR m=1 TO 5
1715 WRITE #9,chana(m,n)
1716 NEXT m:NEXT n
```

```
1717 FOR n=1 TO vb:FOR m=1 TO 5
1718 WRITE #9,chanb(m,n)
1719 NEXT m:NEXT n
1720 FOR n=1 TO vc:FOR m=1 TO 5
1721 WRITE #9,chanc(m,n)
1722 NEXT m:NEXT n
1723 CLOSEOUT
1724 END
1725 :
```

Commentary

When you are satisfied with the arrangement of a tune, run the program. When the tune has finished enter GOTO 1172. Line 1712 opens a file to tape for writing called RONDO. The tempo and number of notes in each channel are saved followed by the contents of the arrays. The file is closed at line 1723. The program to load and play back the information is derived from Program 9.2.

```
1000 REM PROGRAM 9.4
1010 REM Load Tune Data from a File
1020 REM and Play it
1030 :
1040 REM Define Envelopes
1050 ENV 1,1,15,1,3,-1,4,12,-1,8
1060 ENT 1
1070 ENV 2,3,5,1,1,0,8,1,-2,4,13,-1,8
1080 ENT -2,1,1,3,1,-2,3,1,1,3
1090 ENV 3,1,15,1,3,-1,2,12,-1,6
1100 ENT 3
1110 :
1120 CLS:PRINT "Reading in note data..."
1130 OPENIN "RONDO"
1140 INPUT #9,tempo,va,vb,vc
1150 :
1160 DIM chana(5,va)
1170 DIM chanb(5,vb)
1180 DIM chanc(5,vc)
1190 :
1200 FOR n=1 TO va:FOR m=1 TO 5
1210 INPUT #9,chana(m,n)
1220 NEXT m:NEXT n
```

```
1230 FOR n=1 TO vb:FOR m=1 TO 5
1240 INPUT #9,chanb(m,n)
1250 NEXT m:NEXT n
1260 FOR n=1 TO vc:FOR m=1 TO 5
1270 INPUT #9,chanc(m,n)
1280 NEXT m:NEXT n
1290 CLOSEIN
1300 PRINT:PRINT "Here's the Tune..."
1310 :
1320 voca=0:vocb=0:vocc=0
1330 ON SQ(1) GOSUB 1400
1340 ON SQ(2) GOSUB 1460
1350 ON SQ(4) GOSUB 1520
1360 WHILE voca<va OR vocb<vb OR vocc<vc
:WEND
1370 END
1380 :
1390 REM Channel A
1400 voca=voca+1:IF voca>va THEN RETURN
1410 SOUND chana(1,voca),chana(2,voca),c
hana(3,voca)*tempo,0,chana(4,voca),chana
(5,voca)
1420 ON SQ(1) GOSUB 1400
1430 RETURN
1440 :
1450 REM Channal B
1460 vocb=vocb+1:IF vocb>vb THEN RETURN
1470 SOUND chanb(1,vocb),chanb(2,vocb),c
hanb(3,vocb)*tempo,0,chanb(4,vocb),chanb
(5,vocb)
1480 ON SQ(2) GOSUB 1460
1490 RETURN
1500 :
1510 REM Channel C
1520 vocc=vocc+1:IF vocc>vc THEN RETURN
1530 SOUND chanc(1,vocc),chanc(2,vocc),c
hanc(3,vocc)*tempo,0,chanc(4,vocc),chanc
(5,vocc)
1540 ON SQ(4) GOSUB 1520
1550 RETURN
```

Envelopes must be defined, otherwise the program will use those which may

be left over from a previous program. Line 1130 opens a channel to tape for reading and the following lines read in the data which was saved from the last program.

Experimenting with the programs

Program 9.2 is only the beginning. There are several other enhancements you can add. If 15 envelopes are not enough you can redefine envelopes in mid-program. Insert calls to such routines in the WHILE/WEND loop.
You could add another element to the channel arrays to introduce noise to a voice. You could even program one channel to provide a rhythm track.
You can produce your own arrangements of your favourite tunes as well as programming your own compositions.
These three programs include the data and information required to play other tunes. Load Program 9.2 and enter the following lines exactly as they are. Some lines will be blank (followed by a colon) which means they are deleted from the original program.
Program 9.5 plays the next 16 bars of Ronda Alla Turca and includes a repeat of the first eight.

```
10 REM PROGRAM 9.5
20 REM Another 24 Bars of Mozart's
30 REM Rondo Alla Turca
40 REM Insert These Lines
50 REM In PROGRAM 9.2
60 :
1040 va=168:vb=114:vc=114
1245 ENV 4,1,13,1,1,0,30,13,-1,6
1246 ENT -4,1,-14,6,1,14,6
1275 IF n=47 OR n=129 THEN RESTORE 2070
1276 IF n=153 THEN RESTORE 2159
1310 IF note$="R" THEN ev=0 ELSE IF n=51
 OR n=57 OR n=71 OR n=76 OR n=82 OR n=88
 THEN ev=2 ELSE ev=1
1315 IF n=153 OR n=165 THEN ev=4
1405 IF n=31 OR n=85 THEN RESTORE 2180
1406 IF n=98 THEN RESTORE 2269
1535 IF n=30 OR n=85 THEN RESTORE 2290
1536 IF n=98 THEN RESTORE 2379
2060 REM Voice 1
2150 DATA E1,12,E1,6,F1,6
2151 DATA G1,6,G1,6,A1,3,G1,3,F1,3,E1,3
```

```
2152 DATA D1,12,E1,6,F1,6
2153 DATA G1,6,G1,6,A1,3,G1,3,F1,3,E1,3
2154 DATA D1,12,C1,6,D1,6
2155 DATA E1,6,E1,6,F1,3,E1,3,D1,3,C1,3
2156 DATA B0,12,C1,6,D1,6
2157 DATA E1,6,E1,6,F1,3,E1,3,D1,3,C1,3
2158 DATA B0,12
2159 DATA C2,12,A1,6,B1,6
2160 DATA C2,6,B1,6,A1,6,G#1,6
2161 DATA A1,6,E1,6,F1,6,D1,6
2162 DATA C1,12,B0,10,A0,1,B0,1
2163 DATA A0,12
2170 REM Voice 2
2260 DATA E-1,12,R,12
2261 DATA C-1,6,C0,6,E-1,6,E0,6
2262 DATA G-1,12,R,12
2263 DATA C-1,6,C0,6,E-1,6,E0,6
2264 DATA G-1,12,R,12
2265 DATA C-1,6,A-1,6,C-1,6,C0,6
2266 DATA E-1,12,R,12
2267 DATA C-1,6,A-1,6,C-1,6,C0,6
2268 DATA E-1,12
2269 DATA F-1,6,A-1,6,A-1,6,A-1,6
2270 DATA E-1,6,A-1,6,D-1,6,F-1,6
2271 DATA C-1,6,E-1,6,D-1,6,F-1,6
2272 DATA E-1,6,E-1,6,E-1,6,E-1,6
2273 DATA A-1,12
2280 REM Voice 3
2370 DATA R,12,C1,6,D1,6
2371 DATA E1,6,E1,6,R,12
2372 DATA B0,6,G0,6,C1,6,D1,6
2373 DATA E1,6,E1,6,R,12
2374 DATA B0,12,A0,6,B0,6
2375 DATA C1,6,C1,6,R,12
2376 DATA G#0,6,E0,6,A0,6,B0,6
2377 DATA C1,6,C1,6,R,12
2378 DATA R,12
2379 DATA R,6,D#0,6,D#0,6,D#0,6
2380 DATA R,6,E0,6,R,6,B-1,6
2381 DATA R,6,A-1,6,R,6,B-1,6
2382 DATA A-1,6,A-1,6,G#-1,6,G#-1,6
2383 DATA A-2,12
```

Notice how the repeats have easily been reprogrammed by the RESTORE command. The only other addition is ENV 4 and ENT 4, used to produce a trill. The first trilll, on C2, is 1 pitch number out: the other trill, on B0, is exactly right.

```
10 REM PROGRAM 9.6
20 REM John Philip Sousa's
30 REM The Liberty Bell March
40 REM Insert These Lines
50 REM Into PROGRAM 9.2
60 :
1040 va=174:vb=179:vc=209
1180 tempo=5
1190 ENV 1,7,2,1,14,-1,10
1210 ENV 2,7,2,2,4,-1,4,10,-1,6
1230 ENV 3,1,13,4,13,-1,30
1240 ENT -3,1,0,4,1,-9,4,1,9,4
1245 ENV 4,5,3,1,4,-1,6,11,-1,8
1246 ENT -4,1,4,2,1,-8,2,1,4,2
1275 IF n=65 THEN RESTORE 2110
1310 IF note$="R" THEN ev=0 ELSE IF (n=1
 OR n=78 OR n=93 OR n=108 OR n=162) THEN
 ev=3 ELSE ev=1
1405 IF n=80 THEN RESTORE 2500
1440 IF note$="R" THEN ev=0 ELSE IF n=20
8 THEN ev=1 ELSE IF (n>79 AND n<144) THE
N ev=4 ELSE ev=2
1535 IF n=74 THEN RESTORE 2890
1570 IF note$="R" THEN ev=0 ELSE IF n=13
3 THEN ev=3 ELSE IF n=178 THEN ev=1 ELSE
 ev=2
2060 REM Voice 1
2070 DATA &38,F2,63,R,6
2080 :
2090 :
2100 DATA C1,3
2110 DATA &38,A0,6,A0,3,A0,3,G#0,3,A0,3
2120 DATA F1,6,C1,3,C1,6,A0,3
2130 DATA A#0,6,A#0,3,A#0,6,C1,3
2140 DATA D1,15,A#0,3
2150 DATA G0,6,G0,3,G0,3,F#0,3,G0,3
2160 DATA E1,6,D1,3,D1,6,A#0,3
```

```
2170 DATA A0,6,A0,3,A0,6,A#0,3
2180 DATA C1,15,C1,3
2190 DATA A0,6,A0,3,A0,3,G#0,3,A0,3
2200 DATA A1,6,F1,3,F1,6,C1,3
2210 DATA B0,6,G1,3,G1,6,G1,3
2220 DATA G1,15,F1,3
2230 DATA E1,6,G1,3,G1,3,F#1,3,G1,3
2240 DATA D1,6,G1,3,G1,3,F#1,3,G1,3
2250 DATA C1,6,B0,3,C1,6,B0,3
2260 DATA C1,9,C1,9
2270 REM 2nd Part
2280 DATA A0,3,G#0,3,A0,3,D1,6,C1,3
2290 DATA A0,9,F0,9
2300 DATA D0,9,G0,9
2310 DATA F0,15,F0,3
2320 DATA G0,3,A0,3,A#0,3,E1,6,D1,3
2330 DATA C1,9,F1,9
2340 DATA E1,9,D1,9
2350 DATA C1,15,C1,3
2360 DATA D1,6,D1,2,E1,1,D1,3,C"1,3,D1,3
2370 DATA E1,9,E1,9
2380 DATA F1,6,F1,2,A1,1,G1,3,F1,3,G1,3
2390 DATA A1,15,A1,2,A1,1
2400 DATA G1,6,F1,3,D1,6,A#0,3
2410 DATA A0,9,F0,9
2420 DATA G0,9,E0,9
2430 DATA F0,12,R,6
2440 :
2450 REM Voice 2
2460 DATA &38,F0,6,E0,3,D#0,6,D0,3
2470 DATA C0,6,B-1,3,A#-1,6,A-1,3
2480 DATA G-1,3,A-1,3,A#-1,3,A-1,6,G-1,3
2490 DATA C-1,6,R,12
2500 DATA &38,F-1,6,F0,3,F0,6,F0,3
2510 DATA F-1,6,F0,3,F0,6,F0,3
2520 DATA E-1,6,F-1,3,G-1,6,F-1,3
2530 DATA E-1,6,D-1,3,C-1,6,R,3
2540 DATA C-1,6,A#-1,3,A#-1,6,A#-1,3
2550 DATA C-1,6,A#-1,3,A#-1,6,A#-1,3
2560 DATA F-1,6,G-1,3,A-1,6,G-1,3
2570 DATA F-1,6,E-1,3,D-1,6,C-1,3
2580 DATA F-1,6,F0,3,F0,6,F0,3
```

```
2590 DATA F-1,6,F0,3,F0,6,F0,3
2600 DATA D-1,6,E-1,3,F-1,6,E-1,3
2610 DATA D-1,6,C-1,3,B-2,6,A-1,3
2620 DATA G-1,6,E0,3,E0,6,E0,3
2630 DATA G-1,6,F0,3,F0,6,F0,3
2640 DATA C0,6,G-1,3,C0,6,G-1,3
2650 DATA C-1,6,R,3,C0,6,R,3
2660 REM 2nd Part
2670 DATA F-1,6,A-1,3,C-1,6,A-1,3
2680 DATA F-1,6,A-1,3,C-1,6,A-1,3
2690 DATA A#-1,6,F-1,3,C-1,6,E-1,3
2700 DATA F-1,6,A-1,3,A-1,6,A-1,3
2710 DATA E-1,6,A#-1,3,C-1,6,A#-1,3
2720 DATA F-1,6,A-1,3,D-1,6,G#-1,3
2730 DATA C-1,4,F-1,2,G-1,3,D-1,4,F-1,2,
G-1,3
2740 DATA C-1,6,G-1,3,G-1,6,G-1,3
2750 DATA B-2,6,B-1,3,B-1,6,B-1,3
2760 DATA C#-1,6,A-1,3,A-1,6,A-1,3
2770 DATA D-1,6,A-1,3,D-1,6,A#-1,3
2780 DATA C-1,6,A-1,3,A-1,6,A-1,3
2790 DATA G-1,6,A#-1,3,A#-1,6,A#-1,3
2800 DATA C-1,6,A-1,3,A-1,6,A-1,3
2810 DATA E-1,6,A#-1,3,C-1,6,A#-1,3
2820 DATA F-1,6,R,3,F-1,6,R,3
2830 :
2840 REM Voice 3
2850 DATA &38,F1,6,E1,3,D#1,6,D1,3
2860 DATA C1,6,B0,3,A#0,6,A0,3
2870 DATA G0,3,A0,3,A#0,3,A0,6,G0,3
2880 DATA C0,6,R,12
2890 DATA &38,F0,6,C0,3,C0,6,C0,3
2900 DATA F0,6,C0,3,C0,6,C0,3
2910 DATA G0,6,G0,3,G0,6,G0,3
2920 DATA G0,15,R,3
2930 DATA G-1,6,C0,3,C0,6,C0,3
2940 DATA G-1,6,C0,3,C0,6,C0,3
2950 DATA F0,6,F0,3,F0,6,G0,3
2960 DATA A0,15,A0,3
2970 DATA F0,6,C0,3,C0,6,C0,3
2980 DATA F0,6,C0,3,C0,6,C0,3
2990 DATA D0,9,B-1,9
```

```
3000 DATA B-1,6,C0,3,D0,6,R,3
3010 DATA G0,6,C0,3,C0,6,C0,3
3020 DATA G0,6,B-1,3,B-1,6,B-1,3
3030 DATA E0,6,R,3,G-1,6,R,3
3040 DATA E0,6,R,3,E0,6,R,3
3050 REM 2nd Part
3060 DATA R,18
3070 DATA F2,54
3080 :
3090 :
3100 DATA R,6,C0,3,R,6,C0,3
3110 DATA R,6,A-1,3,R,6,G#-1,3
3120 DATA G0,9,F0,9
3130 DATA R,6,C0,3,C0,6,C0,3
3140 DATA F0,6,D0,3,D0,6,D0,3
3150 DATA R,6,C#0,3,C#0,6,C#0,3
3160 DATA R,6,D0,3,R,6,D0,3
3170 DATA R,6,E0,3,E0,6,E0,3
3180 DATA R,6,G0,3,G0,6,G0,3
3190 DATA R,6,C0,3,C0,6,C0,3
3200 DATA R,6,C0,3,C0,6,C0,3
3210 DATA F0,6,R,3,F0,6,R,3
```

This makes use of RESTORE for repeats and several envelopes for tonal variations. The slow attack rate of the envelopes help give it a brass band feel but try it with different envelopes. You may like to define individual tone envelopes for each trill.

```
10 REM PROGRAM 9.7
20 REM Tschaikowsky's
30 REM Dance of the Sugar-plum Fairy
40 REM Insert These Lines
50 REM Into PROGRAM 9.2
60 :
1040 va=104:vb=86:vc=95
1180 tempo=10
1190 ENV 1,2,7,1,14,-1,8
1210 ENV 2,1,12,1,12,-1,8
1220 ENT -2,1,1,3,2,-1,3,1,1,3
1230 ENV 3,1,10,1,10,-1,10
1245 ENV 4,1,12,1,12,-1,5
1246 ENT -4,1,1,5,2,-1,5,1,1,5
```

```
1247 ENV 5,1,12,1,12,-1,8
1248 ENT -5,1,2,2,1,-4,2,1,2,2
1310 IF note$="R" THEN ev=0 ELSE IF n=va
 THEN ev=5 ELSE IF n<17 THEN ev=2 ELSE e
v=1
1440 IF note$="R" THEN ev=0 ELSE IF n=vc
 THEN ev=5 ELSE ev=2
1570 IF note$="R" THEN ev=0 ELSE IF n=vb
 THEN ev=5 ELSE IF (n>36 AND n<48) THEN
ev=3 ELSE ev=2
2060 REM Voice 1
2070 DATA &38,R,4,E1,4,R,4,F#1,4
2080 DATA R,4,G1,4,R,4,D#1,4
2090 DATA R,4,E1,4,R,4,F#1,4
2100 DATA R,4,G1,4,R,4,D#1,4
2110 DATA R,4,G3,2,E3,2,G3,4,F#3,4
2120 DATA D#3,4,E3,4,D3,2,D3,2,D3,4
2130 DATA C#3,2,C#3,2,C#3,4,C3,2,C3,2,C3
,4
2140 DATA B2,2,E3,2,C3,2,E3,2,B2,4,R,4
2150 DATA R,4,G2,2,E2,2,G2,4,F#2,4
2160 DATA C3,4,B2,4,G3,2,G3,2,G3,4
2170 DATA F#3,2,F#3,2,F#3,4,E3,2,E3,2,E3
,4
2180 DATA D#3,2,F#3,2,E3,2,F#3,2,D#3,4,R
,4
2190 DATA R,4,G3,2,E3,2,G3,4,F#3,4
2200 DATA D#3,4,E3,4,D3,2,D3,2,D3,4
2210 DATA C#3,2,C#3,2,C#3,4,C3,2,C3,2,C3
,4
2220 DATA B2,2,E3,2,C3,2,E3,2,B2,4,R,4
2230 DATA R,4,E2,2,C#2,2,E2,4,D#2,4
2240 DATA R,4,D2,2,B1,2,D2,4,C#2,4
2250 DATA R,4,C2,2,A1,2,C2,4,B1,4
2260 DATA R,4,B1,1,D#2,1,F#2,1,B2,1,E2,4
,B0,4
2270 :
2280 REM Voice 2
2290 DATA &38,E0,4,G0,4,E0,4,A0,4
2300 DATA E0,4,A#0,4,E0,4,A0,4
2310 DATA E0,4,G0,4,E0,4,A0,4
2320 DATA E0,4,A#0,4,E0,4,A0,4
```

```
2330 DATA E0,4,B0,4,E0,4,C1,4
2340 DATA E0,4,C#1,4,E0,4,D1,4
2350 DATA E0,4,E1,4,E0,4,F#1,4
2360 DATA E1,4,E1,4,E1,4,E0,1,D0,1,C0,1,
B-1,1
2370 DATA A#-1,4,C1,4,A-1,4,C1,4
2380 DATA G-1,4,B0,4,F#-1,4,A#0,4
2390 DATA F#0,4,B0,4,F#0,4,C#1,4
2400 DATA B-1,4,C0,4,B-1,4,B-1,1,A-1,1,G
-1,1,F#-1,1
2410 DATA E-1,4,B0,4,E0,4,C1,4
2420 DATA E0,4,C#1,4,E0,4,D1,4
2430 DATA E0,4,E1,4,E0,4,F#1,4
2440 DATA E1,4,E1,4,E1,4,G1,1,F#1,1,E1,1
,D1,1
2450 DATA C#1,4,F#0,8,F#1,1,E1,1,D#1,1,C
#1.1
2460 DATA B0,4,E0,8,E1,1,D1,1,C#1,1,B0,1
2470 DATA A0,4,D0,8,D1,1,C1,1,B0,1,A0,1
2480 DATA G0,4,F#0,4,E0,4,B-2,4
2490 :
2500 REM Voice 3
2510 DATA &38,R,4,B0,4,R,4,C1,4
2520 DATA R,4,C#1,4,R,4,C1,4
2530 DATA R,4,B0,4,R,4,C1,4
2540 DATA R,4,C#1,4,R,4,C1,4
2550 DATA R,4,G0,4,R,4,A0,4
2560 DATA R,4,A#0,4,R,4,B0,4
2570 DATA R,4,C#1,4,R,4,D#1,4
2580 DATA G1,4,A1,4,G1,4,R,4
2590 DATA R,4,E0,4,R,4,D#0,4
2600 DATA F#2,4,E2,4,A#2,2,A#2,2,A#2,4
2610 DATA G#2,2,G#2,2,G#2,4,F#2,2,F#2,2,
F#2,4
2620 DATA B0,4,A#0,4,F#0,4,R,4
2630 DATA R,4,G0,4,R,4,A0,4
2640 DATA R,4,A#0,4,R,4,B0,4
2650 DATA R,4,C#1,4,R,4,D#1,4
2660 DATA G1,4,A1,4,G1,4,R,4
2670 DATA R,4,A#1,2,F#1,2,A#1,4,A1,4
2680 DATA R,4,G#1,2,E1,2,G#1,4,G1,4
2690 DATA R,4,F#1,2,D1,2,F#1,4,G1,4
```

```
2700 DATA R,4,A0,4,E0,4,B-1,4
```

The envelopes have been defined to produce percussive, shimmering sounds which are in keeping with the music-box nature of the music. The stereo effect in this piece is quite good, too.

CHAPTER 10

Computer Compositions

Musical compositions produce the chaos and monotony we referred to in Chapter 2. Composing is very much an art although there are several methods and ideas around which produce compositions through scientific and mathematical means. This is one area which is relatively unexplored and there is great scope for you to discover new ideas — and compositions — with your Amstrad computer.

Art must follow certain rules, unless it is totally anarchistic and music is no exception. Anarchy is easily expressed by this short program.

```
100 REM PROGRAM 10.1
110 REM Random Music - Chaos
120 :
130 WHILE -1
140 chan=INT(RND*3+1)
150 IF chan=3 THEN chan=4
160 oct=INT(RND*3+1)-1
170 note=INT(RND*12+1)
180 freq=440*(2^(oct+(note-10)/12))
190 pitch=ROUND(125000/freq)
200 PRINT chan,pitch
210 SOUND chan+128,pitch,dur
220 WEND
```

You can build on this idea to produce more interesting compositions — actually, it's not unlike some *avant-garde* music. Try inserting random note durations. If you remove line 200, the piece will speed up.

Of course, this is not what we normally mean when we talk about composing music. Our main objection is likely to be that it is a completely random series of notes with no relation to each other at all. To this chaos we must bring some order, and order is easily demonstrated by playing the same note over and over again or by playing scales, up and down. The point of all this is to show the two extremes, chaos versus monotony, and to illustrate the necessity of a compromise between the two. The computer is unable to exercise any

artistic judgement over the notes it produces and we must tell it, by careful programming, what will and what will not produce acceptable music. The more rules we lay down, the nearer we will get to a particular style and the more rigid will be the composition. Inspiration is provided by the RND function — and clever programming.

Human composition — algorithms and heuristics

In computing, algorithms are frequently used to solve problems. An algorithm is simply a method of solving a problem — providing a solution exists. If there is no solution, the algorithm should determine this. If it does not either solve the problem or determine that no solution exists it is not an algorithm. For example, mathematical addition and subtraction can be solved by algorithms because we know and can describe exactly how to get to the solution.
An heuristic is frequently described as a rule of thumb. It is used in instances where there is no readily available algorithm, or where such an algorithm would take too long to solve the problem. It involves a commonsense approach. Unlike an algorithm, an heuristic is not guaranteed to produce the best solution. It is not even guaranteed to produce a solution, but if it does it can often find it quicker than an algorithm.
Heuristic procedures are used to determine computer strategy in games such as draughts and chess. An algorithm for these games would involve working out every possible move and deducing the best play from the results. Although this is theoretically possible, it is not practical because of the very large number of possible moves.
The composition process is a mixture of algorithm and heuristic. There are certain chord sequences and series of notes which a composer knows sound good. A common chord sequence such as A minor, G major, F major, E major creates a harmonic framework which has been used as the basis for many hit tunes. The composer still needs, however, to apply rules of thumb to create a melody over the top of the sequence. These rules are the result of experience and inspiration, but *we* shall resort to the RND function, and to some heuristics.

Aspects of a composition

There are three aspects of composition of direct relevance to us, melody, rhythm and harmony. The first two are closely related and the third, harmony, is complex. In this chapter we will examine melody and its associated rhythm, and delve into harmony in Chapter 11.
First steps at computer composition involve trying to create a pleasing melody or sequence of notes. We can devise a set or rules to do this, making them as

simple or as complex as we like. This is not difficult, but the notes in a melody also form a rhythmic pattern, so we must take note lengths into consideration too. This is certainly the more difficult task, because note pitch and note length are generally so tightly interwoven that one often effectively determines the other.
We will begin our experiments with the next program. It uses a set of rules to guide it towards a more musically meaningful output. If we restrict ourselves to the key of C, the rules which we set might look like this:

1) The first note must be a member of a C major chord, i.e. C, E or G.
2) The interval between any two consecutive notes must not be more than four notes.
3) A note B must lead directly to the C above it.
4) The last bar must end on a C and the note must last for the length of the bar.

These simple rules will give us better results than random notes as you will hear.

```
1000 REM PROGRAM 10.2
1010 REM Computer Composition
1020 REM Based on Rules
1030 :
1040 GOSUB 1670:REM Set Up
1050 :
1060 bars=4
1070 count=0
1080 :
1090 FOR b=1 TO bars
1100 PRINT "Composing Bar";b
1110 GOSUB 1260:REM Compose a Bar
1120 NEXT b
1130 :
1140 FOR p=1 TO count
1150 PRINT tune$(p),tune(2,p)
1160 SOUND 1,tune(1,p),tune(2,p)*tempo,0
,1
1170 NEXT p
1180 :
1190 PRINT "Press SPACE for Another Tune
"
1200 PRINT "Press 'R' for a Replay"
```

```
1210 com$=INKEY$:IF com$=" " THEN 1070
1220 IF UPPER$(com$)="R" THEN 1140 ELSE
GOTO 1210
1230 END
1240 :
1250 REM Compose a Bar
1260 duration=0
1270 WHILE duration<>16
1280 count=count+1
1290 noteok=0
1300 WHILE noteok=0
1310 notenum=INT(RND*15+1)
1320 note$=notes$(notenum)
1330 GOSUB 1460:REM Analyse Note
1340 GOSUB 1800:REM Rules
1350 WEND
1360 lastnote=notenum
1370 GOSUB 1530:REM Calculate Pitch
1380 tune(1,count)=pitch:tune$(count)=no
te$
1390 GOSUB 1590:REM Calculate Duration
1400 tune(2,count)=dur
1410 duration=duration+dur
1420 WEND
1430 RETURN
1440 :
1450 REM Analyse Note
1460 length=LEN(note$)
1470 IF length=2 THEN name$=LEFT$(note$,
1):oct=VAL(RIGHT$(note$,1)):RETURN
1480 IF length=4 THEN name$=LEFT$(note$,
2):oct=VAL(RIGHT$(note$,2)):RETURN
1490 IF MID$(note$,2,1)="-" THEN name$=L
EFT$(note$,1):oct=VAL(RIGHT$(note$,2)) E
LSE name$=LEFT$(note$,2):oct=VAL(RIGHT$(
note$,1))
1500 RETURN
1510 :
1520 REM Calculate Pitch
1530 position=INSTR(scale$,name$)/2
1540 freq=440*(2^(oct+(position-10)/12))
1550 pitch=ROUND(125000/freq)
```

```
1560 RETURN
1570 :
1580 REM Calculate Duration
1590 dur=2^INT(RND*2+1)
1600 :
1610 REM Set Last Note to Semibreve
1620 IF b=bars THEN dur=16
1630 :
1640 IF duration+dur<=16 THEN RETURN ELS
E 1590
1650 :
1660 REM Set Up
1670 scale$=" C C#D D#E F F#G G#A A#B"
1680 tempo=10
1690 ENV 1,16,15,6
1700 :
1710 DIM tune(2,150),tune$(250)
1720 DIM notes$(15)
1730 FOR n=1 TO 15
1740 READ note$
1750 notes$(n)=note$
1760 NEXT n
1770 RETURN
1780 :
1790 DATA G-1,A-1,B-1,C0,D0,E0,F0,G0,A0,
B0,C1,D1,E1,F1,G1
1800 :
1810 REM Rules
1820 noteok=0
1830 :
1840 REM Set First Note
1850 IF count=1 AND NOT(name$="C" OR nam
e$="G" OR name$="E") THEN RETURN
1860 :
1870 REM Make a "B" move up to a "C"
1880 IF count>1 AND LEFT$(tune$(count-1)
,1)="B" THEN notenum=lastnote+1:note$=no
tes$(notenum):GOSUB 1460:REM Analyse
1890 :
1900 REM Restrict Jumps to 4 Notes
1910 IF count>1 THEN IF ABS(lastnote-not
enum)>4 THEN RETURN
```

```
1920 :
1930 REM Set Last Bar
1940 IF b=bars AND name$<>"C" THEN RETUR
N
1950 noteok=-1
1960 RETURN
```

Commentary

The first call is to the Set Up routine at line 1670 which defines scale$, tempo and an envelope. scale$ will be familiar from other programs. At line 1710 the array, tune, stores the pitch and duration values of the notes and tune$ stores the note name and octave number. As you experiment with the program and add more rules you will find it helpful either to refer to the name of a note or to its pitch value or position in note$ array.

The array note$ holds the notes from which the program can choose. The notes are read from the DATA statement in line 1790 and base the composition in the key of C. You can alter the choice of notes if you wish. In order to create some sort of order, the program composes in bars, the number of which is determined by line 1060. If you increase this by too much, you may have to redimension the tune and tune$ arrays. The loop between lines 1090 and 1120 composes each bar. The work is done by the routine at line 1260. The variable duration keeps track of the cumulative length of the notes in each bar and each bar will contain the equivalent of 16 quavers. The program therefore produces music in 4/4 time. The loop between lines 1270 and 1420 cycles until the duration is equal to 16. The variable count stores the actual number of notes composed.

The inner loop between lines 1300 and 1350 cycles until the note has run the gauntlet of the rules. Each time round the loop, a new note is chosen at random from the note$ array. The analysis routine at line 1460 should be familiar by now. When the program gets to line 1360, the note has passed the rules. The pitch is calculated by the routine at line 1530 and the pitch and note are stored in tune and tune$ arrays.

The duration routine at line 1590 gives the note a duration of 2 or 4 and ensures that the last note has a value of 16.

After the bars have been composed, lines 1140 to 1170 play them and print out the notes. Lines 1190 to 1220 give you the opportunity to hear the tune again or compose another one.

The note analysis and the pitch routines have been kept separate so you can refer to one aspect of the note without referring to the other. It also enables you to force a note into either of the routines, as may be necessary, for example, in rule 3 above.

The rules routine is at the end of the program, so you can add to it if you wish. It assumes that the note is *not* going to be OK (i.e. FALSE) in line 1820 and it must run the gauntlet of the rules until it comes out OK (i.e. TRUE) at line 1950. The first rule checks that the first note is C, E or G. The second rule creates a C to follow a B. The variables notenum and lastnote refer to the position of the note in the data stream. If you want each G to move to an A, adding 1 to G1 (note = 15) would take it off the scale. The next rule checks the steps between the last note and the current one and rejects the new note if the distance between is too far. Finally, the last rule ensures a C fills the last bar.

Experimenting with the program

Although this is a great improvement on random music it's probably fair to say it produces music that only a programmer could love. We need more melodic rules, and the phrasing produced by the duration routine needs more attention. You can easily add to and adapt the melodic rules described above. Here are some further suggestions:

1) Do not have more than five notes rising or descending without a complementary movement.
2) A rising B moves to a C, a descending B moves to an A.
3) A rising E moves to an F.
4) A B will not move to an F and vice versa. (This is quite a harsh interval unless harmonically controlled.)
5) Permit the inclusion of accidentals, possibly only F# or A#. Further rules would be needed to cope with these.

Music consists of a series of phrases, much like phrases in English grammar, which make sense but which are not complete. The duration routine makes no attempt to regulate the rhythmic content. The phrasing could be controlled by passing the durations through a set of rules in a similar way to the melody notes. Such rules could include:

1) If quavers occur, there must be at least two of them consecutively.
2) A bar cannot start with a set of three quavers.
3) Give the duration values of bar 1 to bar 2, bar 3 to bar 4, for example.

An alternative is to use a preset series of durations. This is the easiest option but, of course, results in a repetitive rhythm pattern. It can be helpful if you wish to concentrate on the melodic aspect and it is probably an improvement on the random method.

```
10 REM PROGRAM 10.3
20 REM Computer Compositions with
30 REM Fixed Rhythm Pattern
40 REM Insert Into PROGRAM 10.2
50 :
1075 RESTORE 1610
1590 READ dur
1600 RETURN
1610 DATA 2,2,2,2,4,4
1620 DATA 2,2,4,8
1630 DATA 2,2,4,2,2,4
1640 DATA 4,4,8
1695 RESTORE 1790
1940 IF count=19 AND name$<>"C" THEN RET
URN
```

Commentary

The value that count is checked against in line 1940 refers to the number of notes in the DATA statements. You can add variation by allowing the option of switching between sets of preset durations. This will give you the best of both worlds. As you add more rules you affect the style of the composition — if you add enough you will create a style unique to yourself (and the computer). There are other ways of programming style into a composition program and we will look at one such method next.

Note analysis in composition

If we analyse a musical composition note by note and construct a table of how often each note occurs we will have a first order note analysis of that tune. If we then write a program to play the notes according to the frequency of their appearance in the table we will have a composition which tends towards the style of the music we analysed.

This idea is not new and experiments along these lines were carried out over 20 years ago on Stephen Foster compositions. (He wrote such songs as 'Camptown Races', 'Oh Susannah' and 'Old Folks at Home'.)

The success of such experiments depends upon the composisions under analysis. If the notes of the scale appear in roughly equal proportions, the result is not going to sound unlike random music. In fact, using only first order note analysis, the result will tend to sound a little like random notes anyway. There is a need, too, to take into account the note durations.

We can increase the accuracy of our analysis by recording how often a note

follows each other one. This is known as second order analysis and we can take it even further and do a third and fourth order analysis. This produces considerably better results but as we perform higher and higher order analysis on the music, we end up with a composition which sounds increasingly like the original. The next program performs a first, second and third order analysis upon a tune entered in DATA statements. It will then compose a tune based upon one of these levels of analysis: the level can be changed as the tune is playing.

```
1000 REM PROGRAM 10.4
1010 REM Computer Compositions
1020 REM Based Upon Note Analysis
1030 :
1040 MODE 1
1050 DEFINT a-z
1060 DIM tune$(238),dur(238),f1(36)
1070 scale$="   C-1 C#-1D-1 D#-1E-1 F-1
F#-1G-1 G#-1A-1 A#-1B-1 C0  C#0 D0  D#0
E0  F0  F#0 G0  G#0 A0  A#0 B0  C1  C#1
D1  D#1 E1  F1  F#1 G1  G#1 A1  A#1 B1
"
1080 WINDOW #0,1,40,2,40
1090 WINDOW #1,20,35,1,1
1100 ENV 1,1,15,1,1,0,6,1,-2,2,13,-1,8
1110 ENT -1,1,1,3,2,-1,3,1,1,3
1120 tempo=6
1130 :
1140 GOSUB 1690:REM Get Tune
1150 GOSUB 1840:REM New Scale
1160 GOSUB 2080:REM Analyse Tune
1170 GOSUB 2220:REM Calc Percentages
1180 GOSUB 2530:REM Printout
1190 :
1200 penult=INT(RND*scalelength+1)
1210 lastnote=INT(RND*scalelength+1)
1220 PRINT "Enter Search Depth (1/2/3) -
 This can","be altered as the program is
 running"
1230 play=VAL(INKEY$):IF play<1 OR play>
3 THEN 1230
1240 CLS:PRINT#1,"Depth :";play
1250 :
```

```
1260 d=1
1270 WHILE -1
1280 GOSUB 1360:REM Get Note
1290 GOSUB 1600:REM Play Note
1300 pl=VAL(INKEY$):IF pl>0 AND pl<4 THE
N play=pl:PRINT#1,"Depth :";play
1310 WEND
1320 :
1330 END
1340 :
1350 REM Get Note
1360 dice=INT(RND*100+1)
1370 note=0:sum=0
1380 WHILE sum<dice
1390 note=note+1
1400 REM In Case a Note has never
1410 REM Followed a Particular
1420 REM Sequence of Notes
1430 IF note>scalelength THEN note=INT(R
ND*scalelength+1):sum=dice:GOTO 1450
1440 ON play GOSUB 1530,1550,1570
1450 WEND
1460 :
1470 note$=MID$(scale2$,note*4,4)
1480 penult=lastnote:lastnote=note
1490 REM PRINT note$,dur(d)
1500 RETURN
1510 :
1520 REM First Order
1530 sum=sum+f1(note):RETURN
1540 REM Second Order
1550 sum=sum+f2(lastnote,note):RETURN
1560 REM Third Order
1570 sum=sum+f3(penult,lastnote,note):RE
TURN
1580 :
1590 REM Play Note
1600 position=INSTR(scale$,note$)/4-12
1610 freq=440*(2^(oct+(position-10)/12))
1620 pitch=ROUND(125000/freq)
1630 SOUND 1,pitch,dur(d)*tempo,0,1,1
1640 d=d+1
```

```
1650 IF d>tunelength THEN d=1
1660 RETURN
1670 :
1680 REM Get Tune
1690 CLS:PRINT "Reading in Tune for Anal
ysis..."
1700 REM RESTORE to Required Tune
1710 RESTORE 2780:REM 2780,2850,2920
1720 :
1730 count=0
1740 WHILE note$<>"X"
1750 READ note$,dur:IF note$="X" THEN 17
90
1760 count=count+1
1770 tune$(count)=note$
1780 dur(count)=dur
1790 WEND
1800 tunelength=count
1810 RETURN
1820 :
1830 REM New Scale
1840 PRINT "Calculating New Scale..."
1850 :
1860 FOR n=1 TO tunelength
1870 npos=INSTR(scale$,tune$(n))/4
1880 f1(npos)=f1(npos)+1
1890 NEXT n
1900 :
1910 scale2$="    "
1920 :
1930 FOR n=1 TO 36
1940 IF f1(n)>0 THEN scale2$=scale2$+MID
$(scale$,n*4,4)
1950 NEXT n
1960 :
1970 PRINT "New Scale:":PRINT:PRINT scal
e2$:PRINT
1980 scalelength=(LEN(scale2$)-3)/4
1990 PRINT "Scale Length =";scalelength:
PRINT
2000 :
2010 DIM f2(scalelength,scalelength),f3(
```

```
scalelength,scalelength,scalelength)
2020 :
2030 REM Reset f1 array
2040 FOR n=1 TO 36:f1(n)=0:NEXT n
2050 RETURN
2060 :
2070 REM Analyse Tune
2080 PRINT "Analysing Tune..."
2090 FOR n=1 TO tunelength
2100 pos1=INSTR(scale2$,tune$(n))/4
2110 f1(pos1)=f1(pos1)+1
2120 IF n>tunelength-1 THEN 2150
2130 pos2=INSTR(scale2$,tune$(n+1))/4
2140 f2(pos1,pos2)=f2(pos1,pos2)+1
2150 IF n>tunelength-2 THEN 2180
2160 pos3=INSTR(scale2$,tune$(n+2))/4
2170 f3(pos1,pos2,pos3)=f3(pos1,pos2,pos
3)+1
2180 NEXT n
2190 RETURN
2200 :
2210 REM Calculate Percentages
2220 PRINT "Calculating First Order Freq
uency..."
2230 sum1=0
2240 FOR n1=1 TO scalelength
2250 sum1=sum1+f1(n1)
2260 NEXT n1
2270 FOR n1=1 TO scalelength
2280 f1(n1)=f1(n1)*100/sum1
2290 NEXT n1
2300 :
2310 PRINT "Calculating Second Order Fre
quency..."
2320 FOR n1=1 TO scalelength
2330 sum2=0
2340 FOR n2=1 TO scalelength
2350 sum2=sum2+f2(n1,n2)
2360 NEXT n2
2370 IF sum2>0 THEN FOR n2=1 TO scalelen
gth:f2(n1,n2)=f2(n1,n2)*100/sum2:NEXT n2
2380 NEXT n1
```

```
2390 :
2400 PRINT "Calculating Third Order Freq
uency..."
2410 FOR n1=1 TO scalelength
2420 FOR n2=1 TO scalelength
2430 sum3=0
2440 FOR n3=1 TO scalelength
2450 sum3=sum3+f3(n1,n2,n3)
2460 NEXT n3
2470 IF sum3>0 THEN FOR n3=1 TO scalelen
gth:f3(n1,n2,n3)=f3(n1,n2,n3)*100/sum3:N
EXT n3
2480 NEXT n2
2490 NEXT n1
2500 RETURN
2510 :
2520 REM Printout
2530 PRINT:PRINT "Do you want a Printout
 (Y/N)?"
2540 com$=INKEY$:IF UPPER$(com$)="N" THE
N PRINT:RETURN ELSE IF UPPER$(com$)<>"Y"
 THEN 2540
2550 :
2560 FOR n1=1 TO scalelength
2570 IF f1(n1)>0 THEN PRINT MID$(scale2$
,n1*4,4);"...";f1(n1)
2580 NEXT n1
2590 :
2600 FOR n1=1 TO scalelength
2610 FOR n2=1 TO scalelength
2620 IF f2(n1,n2)>0 THEN PRINT MID$(scal
e2$,n1*4,4);"- ";MID$(scale2$,n2*4,4);".
..";f2(n1,n2)
2630 NEXT n2
2640 NEXT n1
2650 :
2660 FOR n1=1 TO scalelength
2670 FOR n2=1 TO scalelength
2680 FOR n3=1 TO scalelength
2690 IF f3(n1,n2,n3)>0 THEN PRINT MID$(s
cale2$,n1*4,4);"- ";MID$(scale2$,n2*4,4)
;"- ";MID$(scale2$,n3*4,4);"...";f3(n1,n
```

```
2,n3)
2700 NEXT n3
2710 NEXT n2
2720 NEXT n1
2730 PRINT
2740 :
2750 RETURN
2760 :
2770 REM Ode To Joy - Beethoven
2780 DATA E0,8,E0,8,F0,8,G0,8,G0,8,F0,8,
E0,8,D0,8,C0,8,C0,8,D0,8,E0,8,E0,12
2790 DATA D0,4,D0,16,E0,8,E0,8,F0,8,G0,8
,G0,8,F0,8,E0,8,D0,8,C0,8,C0,8,D0,8
2800 DATA E0,8,D0,12,C0,4,C0,16,D0,8,D0,
8,E0,8,C0,8,D0,8,E0,4,F0,4,E0,8,C0,8
2810 DATA D0,8,E0,4,F0,4,E0,8,D0,8,C0,8,
D0,8,G-1,16,E0,8,E0,8,F0,8,G0,8,G0,8
2820 DATA F0,8,E0,8,D0,8,C0,8,C0,8,D0,8,
E0,8,D0,12,C0,4,C0,16,X,0
2830 :
2840 REM Jesu, Joy - Bach
2850 DATA G0,6,A0,6,B0,6,D1,6,C1,6,C1,6,
E1,6,D1,6,D1,6,G1,6,F#1,6,G1,6,D1,6
2860 DATA B0,6,G0,6,A0,6,B0,6,C1,6,D1,6,
E1,6,D1,6,C1,6,B0,6,A0,6,B0,6,G0,6
2870 DATA F#0,6,G0,6,A0,6,D0,6,F#0,6,A0,
6,C1,6,B0,6,A0,6,B0,6,G0,6,A0,6,B0,6
2880 DATA D1,6,C1,6,C1,6,E1,6,D1,6,D1,6,
G1,6,F#1,6,G1,6,D1,6,B0,6,G0,6,A0,6
2890 DATA B0,6,E0,6,D1,6,C1,6,B0,6,A0,6,
G0,6,D0,6,G0,6,F#0,6,G0,18,X,0
2900 :
2910 REM Irish Washer Woman - Jig
2920 DATA D0,4,B-1,4,G-1,4,G-1,4,D-1,4,G
-1,4,G-1,4,B-1,4,G-1,4,B-1,4,D0,4,C0,4,B
-1,4
2930 DATA C0,4,A-1,4,A-1,4,E-1,4,A-1,4,A
-1,4,C0,4,A-1,4,C0,4,E0,4,D0,4,C0,4
2940 DATA B-1,4,G-1,4,G-1,4,D-1,4,G-1,4,
G-1,4,B-1,4,G-1,4,B-1,4,D0,4,C0,4,B-1,4
2950 DATA C0,4,B-1,4,C0,4,A-1,4,D0,4,C0,
4,B-1,4,G-1,4,G-1,4,G-1,8,G0,4
```

```
2960 DATA G0,4,D0,4,G0,4,G0,4,D0,4,G0,4,
G0,4,D0,4,G0,4,B0,4,A0,4,G0,4
2970 DATA F#0,4,D0,4,F#0,4,F#0,4,D0,4,F#
0,4,F#0,4,D0,4,F#0,4,A0,4,G0,4,F#0,4
2980 DATA E0,4,G0,4,G0,4,D0,4,G0,4,G0,4,
C0,4,G0,4,G0,4,B-1,4,G0,4,G0,4
2990 DATA C0,4,B-1,4,C0,4,A-1,4,D0,4,C0,
4,B-1,4,G-1,4,G-1,4,G-1,8,X,0
3000 d=1:WHILE -1:note$=tune$(d):GOSUB 1
600:WEND
```

The program prompts you for input where required and will produce a continuous composition.

Commentary

Note analysis is ideally suited to a computer. First and second order analyses are quick but in third order analysis we need to keep track of a list of three consecutive elements. For a three-octave range, this could mean dimensioning an array like

DIM f3(36,36,36)

or larger, which uses up more memory than we can afford. Usually, however, you will find that not all of the notes between the lowest and highest are used, and so reserving space for them is a waste of memory. With this in mind and with the aim of maintaining readability, I have developed a way around this problem. It consists of calculating a new scale based upon the notes used in the tune and dimensioning the arrays after we see how large they need to be. The array tune$ is dimensioned to hold the notes and dur is dimensioned to hold the durations. f1, and later f2 and f3 hold the first, second and third order sequences.

The range of available notes is put into scale$ in line 1070. If you enter other tunes check that they do not contain notes outside this range. Line 1140 calls a routine at line 1690 which reads in the tune from DATA statements. If you have more than one tune in the program, set RESTORE to point to the required line. The tune data is terminated with an "X". The number of notes read is stored in tunelength.

The next routine, called at line 1840, calculates the new scale. It compares the notes of the tune with scale$ and forms a new scale, scale2$, consisting only of the notes used in the tune. A new scale will typically contain about 12 notes — see how much memory we are saving. The arrays for storing the second and

third order sequences are then dimensioned in line 2010. The f1 array is cleared for further use.

The routine at line 2080 analyses the tune. It runs through the tune note by note and counts the number of times each sequence of notes occurs. The results are stored in the arrays f1, f2 and f3. f1 just counts the notes, f2 counts each sequence of two notes and f3 counts each sequence of three notes. The f1 array is used hereafter to count the notes in the new scale routine.

The routine at line 2220 calculates each order analysis. First, the number of times each sequence occurs in the tune is calculated. The loop runs through them again and assigns new values to the arrays on a percentage basis. This is most easily seen in the first example between lines 2230 and 2290. The second and third order sequences use a series of nested loops to check every combination. This method reuses all the arrays, saving memory. Because we are using integer variables and arrays, defined in line 1050, the 'percentages' will not always add up to 100, because of rounding errors. This has a negligible affect on the composition and results in a faster program and a memory saving.

You can use the same principle to calculate higher order analysis. The results should be very interesting but you will see how close we are to the orignal tune even with third order analysis.

The print routine at line 2530 uses nested loops in a way similar to the percentage calculation routine except it prints out the first, second and third order sequences along with their occurrence expressed as a percentage. If you see three notes followed by the figure, 100, you know that after the first two notes, the third note *always* occurs. If the figure was only 50 it would indicate that the third note follows the other two 50% of the time. Before it can start composing, the program needs two seed notes on which to base its first calculation. These are determined randomly in lines 1200 and 1210. Line 1220 asks if you want a first, second or third order composition.

The loop between lines 1270 and 1310 plays the tune. The routine at line 1360 decides which note should be played. The variable dice represents a random percentage.

The depth of analysis depends upon sum.

This may require further explanation. The variables note, lastnote and penult have values which are used to access a certain note or sequence of notes in the arrays, f1, f2 and f3. These arrays contain percentages ranging from 0 to 100 which represent the frequency occurrence of the notes represented by note, lastnote and penult. As an example, if the f2 array held the following:

f2(lastnote,1) = 20
f2(lastnote,2) = 50
f2(lastnote,3) = 10

f2(lastnote,4) = 8
f2(lastnote,5) = 12

The only figure we are concerned with is note which is increased by one each time round the loop. Each time round, sum has added to it the figure (percentage) held in the array.
If dice equals 72 the routine would work as follows: sum begins with a value of 0 and note with a value of 1. We add f2(lastnote,note) to sum. If it equals or is more than dice we leave the loop. In this case, with note equal to 1, it will equal 20 which is not enough so one is added to note and we try again. This time 50 is added to sum which is still not enough so we repeat the process until note equals 3 which will give sum a value of 80, greater than dice, so the loop is exited.
We calculated note$ from the value of note. Then penult and lastnote are adjusted and we move to the play routine at line 1600.
Line 1490 has been REMed out because the time taken to print this information can slow the program up considerably. This will be most noticeable with fast tunes but you can unREM it if you want to see which notes are being produced by the program. The Get Note routine at line 1360 can also take a while to produce a note.
The play routine works out the pitch number in a manner which should be familiar by now. Notice that position will return a value from 1 to 36 — the number of notes in scale$ — and we deduct 12 from it effectively to lower the octave. Because oct has not been defined anywhere it will default to 0. We can thus calculate freq without needing an octave number. As all numeric variables were defined as integers, the values returned by freq and pitch will not always be exact but they will be close enough to make little difference.
In line 1260 d is used to access the duration values held in the array, dur, and when it is greater than the length of the tune it is set again to 1 at line 1650.
The program won't play the original tune exactly (it's theoretically possible but very unlikely). To play it, run the program, break out of it after the tune has been read in and enter:

```
GOTO 3000
```

You can use this to check that new tune data has been entered correctly, too.

Experimenting with the program

The tunes supplied in the DATA statements were selected because the durations of each note are roughly equal. This makes it easier for us to tell how close the compositions are to the original. It is interesting to play the

compositions to other people and ask them if they can tell from which tunes they are derived. Start with the first order frequency and move on to second and third if they find it difficult.
The compositions play with exactly the same note durations as the original tune. This does not encourage originality but as the durations are stored in a separate array it would be easy to program your own note lengths into the compositions. You could also run the durations through a routine similar to the one used on the melody notes. Combining the two should prove interesting.
The program only analyses one tune at a time and the result is, predictably, a variation on the tune. If you use several tunes by the same composer you should get a composition in that composer's style but which is new.
You could also try several rock and roll tunes. These use the same basic chords and harmonic structure but have different melodies. Remember that when you enter large amounts of data it is important to save memory. If the tunes and data are very long you may have to increase the size of the arrays at line 1060. As you already have the data for some tunes from Chapter 9, Rondo Alla Turca etc., you could use this for analysis. You may have to alter scales$ in some cases.

A total tune analysis program

From the principles we have discussed and demonstrated so far it is possible to envisage a program which would analyse every part of a tune. There are at least two ways to approach this. One has been suggested and involves running the durations through the same sort of procedures as the melody notes. This would result in a rhythm pattern based on the tune but not related to the melody notes. A second method would be to analyse the notes and their respective durations together so that C1 with a duration of 2 would be treated as one case and C1 with a duration of 4 would be treated as another. You can see that this would consume memory quickly, but it would tie the melody and rhythm together in a more realistic way. Taking the idea a step further, we could also include an analysis of the harmony indicated by the chord structure. Many modern songs change chords every bar or every four bars and this could easily be analysed. If you decided to analyse rock and roll tunes there would be no need to analyse the chords as most songs use the same pattern. In the key of C the bars would contain the following chords:

C/C/C/C/F/F/C/C/G/F/C/C

This is the famous 12 Bar Blues, known in the music business simply as a 12 bar. If another chorus is to be played, the last bar of C is often replaced with a

bar of G (or G7).
In an ideal program we could analyse chords, notes and durations all together. It would be very interesting and not difficult to program, but very greedy of memory. I leave it for the virtuoso.

CHAPTER 11
Your Amstrad Composes

In the previous chapter, we investigated some of the difficulties we need to overcome when writing programs to produce a melody. The programs produce melodies of various qualities but there is one thing they all lack — harmony.

Harmony is produced when more than one note sounds at the same time. A harmonic framework or background to a piece of music can be provided by strumming a guitar or playing chords on a piano or organ. (See Chapter 2: Program 2.1 gives an aural indication of the harmonies produced by various chords.) Given a chord progression, any number of melody lines can be written to fit over the top. Conversely, backing chords can be supplied for melody notes.

The harmonic structure of popular songs

Most popular tunes rely heavily upon their chord progressions for their appeal. The type of chord used to harmonise a tune is indicative of the level of harmonic appreciation we, the public, have reached. The wandering minstrel of a few hundred years ago would use a far less complicated set of chord progressions, because the level of harmonic appreciation (or tolerance) the public had reached was lower. We are now musically more tolerant, so more complicated and dissonant harmonies are finding increasing acceptance.

Jazz musicians specialise in taking a melody or a chord progression and improvising new melodies or harmonies for that piece. Often, these will be deliberately removed from the original tune. This is why many people find it difficult to listen to and understand jazz. Classical music had its own harmonic and compositional rules (most of the great musicians broke them) and you can hear how we have progressed, harmonically, if you listen to a piece of music by Purcell or Arne.

Within any particular genre of music, our ears expect to hear a certain type of chord sequence (or harmonic structure) and a melody ordered in a certain way. A program to produce a pop tune, for example, would necessarily be quite complex and intricate. It is perhaps slightly easier to produce music in a classical style which may be, apparently, less harmonically (and structurally) demanding.

Producing acceptable results

In the field of computer compositions we can safely assume that anything which sounds vaguely pleasant and does not make the listener squirm in his seat is a success. This chapter goes a little further, I hope, and enables us to program the Amstrad to produce compositions in up to three voices which will be musically acceptable.

You might imagine that the production of a two-voiced or three-voiced composition would be two or three times as difficult as that for one voice. If we were to try to implement some standard, academic rules of harmony, that would indeed be the case. Our main aim, however, will be to produce a series of two or three notes sounding together which are not dissonant and which form, in total, a reasonably pleasant harmonic (and melodic) progression.

Random harmonic compositions

Music combines both order and disorder in various proportions. Any composition program needs a random element otherwise it would not be at all original. Our problem is to find a way of controlling the amount of randomness.

The quickest and most effective way is to use the random function to select a note or series of notes which you know will harmonise with each other. This, obviously, will produce pleasing but not very original results.

Mozart is reputed to have devised a compositional system based upon throwing a die. His method could easily be converted to a computer program and would produce spectacular results. The dice were used to select one of a number of bars of music which had previously been composed. For example, to compose a tune eight bars long a table would be drawn up of eight columns and six rows each containing a bar of music. The first throw of the dice would select a bar from column 1, the second throw from column 2, and so on until you had eight bars.

The initial problem is in composing 48 bars of music in such a way that any bar from one column can follow any bar from the previous column. No problem for Mozart but perhaps more daunting for us not-so-great composers. If you do attempt it, and it may not be as difficult as it sounds, the results would certainly be fun. In this case though, you are doing all the composing, not the computer.

You can apply other rules and modifications to a composition program so that, for example, it periodically inserts a previously written series of notes into its otherwise random output. To help the computer produce good results, we can specify a list of permissible notes for it to choose from.

Using chords as a compositional base

One way to broaden the note selection is to relate the choice of notes to particular chords and their relative constituent notes. The computer will then choose notes from within a predefined harmonic framework. Program 11.1 does this and allows you to program the computer with any chord sequence and any chord type that you wish.

```
1000 REM PROGRAM 11.1
1010 REM Computer Compositions
1020 REM Based Upon Chord Sequences
1030 :
1040 MODE 1
1050 DEFINT a-e,g-z
1060 scale$=" C C#D D#E F F#G G#A A#B"
1070 chordrange$="   M    7    9    min min
6min7min9maj6maj7aug dim"
1080 :
1090 DIM notes$(11,6)
1100 :
1110 REM RESTORE to Required Chord DATA
1120 RESTORE 2390:REM 2390,2420,2450,252
0
1130 READ noofchords
1140 DIM melody$(noofchords)
1150 :
1160 FOR n=1 TO noofchords
1170 READ melody$(n)
1180 NEXT n
1190 :
1200 RESTORE 2160
1210 FOR n=1 TO 11
1220 FOR c=1 TO 6
1230 READ notes$(n,c)
1240 NEXT c
1250 NEXT n
1260 :
1270 ENV 1,1,14,10,4,-1,6,1,0,10,10,-1,1
2
1280 ENT 1
1290 ENV 2,1,15,20,1,0,20,4,-1,10,11,-1,
12
1300 ENT -2,1,2,3,2,-2,3,1,2,3
```

```
1310 ev1=1:ev2=1:ev3=2
1320 :
1330 PRINT:PRINT:PRINT
1340 INPUT "Enter Number of Beats in Bar
";noofbeats
1350 INPUT "Enter Tempo  (25 or greater)
";tempo1
1360 tempo=tempo1
1370 :
1380 PRINT "Dou you want Rhythm Variatio
ns (Y/N)? ";
1390 sync$=UPPER$(INKEY$):IF sync$="Y" T
HEN sync=-1 ELSE IF sync$="N" THEN sync=
0 ELSE GOTO 1390
1400 PRINT sync$
1410 :
1420 opus=0
1430 WHILE -1
1440 opus=opus+1
1450 in0=INT(RND*26):in1=INT(RND*26):IF
in0=in1 THEN 1450
1460 INK 0,in0:INK 1,in1:BORDER INT(RND*
26)
1470 LOCATE 12,10:PRINT "Composing Opus"
;opus
1480 :
1490 FOR n=1 TO noofchords
1500 syncpoint=INT(RND*4+1)
1510 FOR beat=1 TO noofbeats*2
1520 IF sync THEN GOSUB 1970:REM Rhythm
1530 GOSUB 1610:REM Play
1540 NEXT beat
1550 NEXT n
1560 WEND
1570 :
1580 END
1590 :
1600 REM Play
1610 GOSUB 1740:REM Analyse Chord
1620 n$=note1$:GOSUB 1880:REM Get Note
1630 SOUND &38+1,pitch,tempo,0,ev1,ev1
1640 n$=note2$:GOSUB 1880:REM Get Note
```

```
1650 SOUND &38+4,pitch,tempo,0,ev2,ev2
1660 REM Bass
1670 position=INSTR(scale$,ke$)/2
1680 freq=440*(2^(-1+(position-10)/12))
1690 pitch=ROUND(125000/freq)
1700 SOUND &38+2,pitch,tempo,0,ev3,ev3
1710 RETURN
1720 :
1730 REM Analyse Chord
1740 chord$=melody$(n)
1750 IF MID$(chord$,2,1)="#" THEN ke$=LE
FT$(chord$,2):chordtype$=MID$(chord$,3)
ELSE ke$=LEFT$(chord$,1):chordtype$=MID$
(chord$,2)
1760 ke=INSTR(scale$,ke$)/2
1770 chordnum=INSTR(chordrange$,chordtyp
e$)/4
1780 :
1790 choice1=INT(RND*6+1)
1800 choice2=INT(RND*6+1)
1810 IF choice2=choice1 THEN 1800
1820 :
1830 note1$=notes$(chordnum,choice1)
1840 note2$=notes$(chordnum,choice2)
1850 RETURN
1860 :
1870 REM Get Note
1880 IF LEN(n$)=2 THEN name$=LEFT$(n$,1)
 ELSE name$=LEFT$(n$,2)
1890 oct=VAL(RIGHT$(n$,1))
1900 position=INSTR(scale$,name$)/2+ke-1
1910 freq=440*(2^(oct+(position-10)/12))
1920 pitch=ROUND(125000/freq)
1930 RETURN
1940 :
1950 REM Rhythm Variations
1960 REM GOTO Required Rhythm
1970 ON syncpoint GOTO 2000,1990,2020,20
50
1980 :
1990 IF beat=1 THEN tempo=tempo1*1.5 ELS
E IF beat=2 THEN tempo=tempo1*0.5 ELSE t
```

```
empo=tempo1
2000 RETURN
2010 :
2020 IF beat=noofbeats OR beat=noofbeats
+1 THEN tempo=tempo1*0.5 ELSE IF beat=1
THEN tempo=tempo1*2 ELSE tempo=tempo1
2030 RETURN
2040 :
2050 IF beat=noofbeats OR beat=noofbeats
+1 OR beat=noofbeats+2 OR beat=noofbeats
+3 THEN tempo=tempo1*0.5:GOSUB 1610 ELSE
 tempo=tempo1
2060 RETURN
2070 :
2080 IF beat MOD 2=1 THEN tempo=tempo1*0
.75 ELSE IF beat MOD 2=0 THEN tempo=temp
o1*0.25
2090 RETURN
2100 :
2110 REM Out of Sync - 9/8 time
2120 IF beat=1 THEN tempo=tempo1*2 ELSE
IF beat=2 THEN beat=3 ELSE tempo=tempo1
2130 RETURN
2140 :
2150 REM Major
2160 DATA G0,C1,E1,G1,C2,E2
2170 REM Seventh
2180 DATA A#0,C1,E1,G1,A#1,C2
2190 REM Major Ninth
2200 DATA D1,E1,G1,A#1,C2,D2
2210 REM Minor
2220 DATA G0,C1,D#1,G1,C2,D#2
2230 REM Minor Sixth
2240 DATA A0,C1,D#1,G1,A1,C2
2250 REM Minor Seventh
2260 DATA A#0,C1,D#1,G1,A#1,C2
2270 REM Minor Ninth
2280 DATA D1,D#1,G1,A#1,C2,D2
2290 REM Major Sixth
2300 DATA A0,C1,E1,G1,A1,C2
2310 REM Major Seventh
2320 DATA B0,C1,E1,G1,B1,C2
```

```
2330 REM Augmented
2340 DATA G#0,C1,E1,G#1,C2,E2
2350 REM Diminished
2360 DATA D#1,F#1,A1,C2,D#2,F#2
2370 :
2380 REM Tune DATA Starts Here
2390 DATA 12
2400 DATA C7,C7,C7,C7,F7,F7,C7,C7,G7,F7,
C7,G7
2410 :
2420 DATA 12
2430 DATA F7,F9,F7,F9,A#7,A#9,F7,F9,C9,A
#9,F9,C7
2440 :
2450 DATA 32
2460 DATA Cmin,Cmin,G7,G7,G7,G7,Cmin
2470 DATA Cmin,Cmin,Cmin,G7,G7,G7,G7
2480 DATA Cmin,Cmin,Fmin,Fmin,Cmin,Cmin
2490 DATA G7,G7,Cmin,Cmin,Fmin,Fmin
2500 DATA Cmin,Cmin,G7,G7,Cmin,Cmin
2510 :
2520 DATA 16
2530 DATA Amin7,D7,Gmaj6,Emin6,Gmin9,C7
2540 DATA Fmaj7,Dmin6,Fmin7,G#min6
2550 DATA Gmaj6,D#M,Cmin6,D7,Bmin,E7
```

The program will prompt for the number of beats in a bar, tempo, and rhythmic variations. The compositions are based upon a chord sequence resident in DATA statements and the rhythm variations can be altered and adjusted within the program.
This program produces quite a full sound, and you may need to turn the volume down to prevent distortion.

Commentary

In line 1070 chordrange$ holds the available chords in much the same way that scale$ holds the available notes. The program is supplied with details of the following chords:

M:	Major
7:	Seventh (dominant seventh)
9:	Major Ninth

min: Minor
min6: Minor Sixth
min7: Minor Seventh
min9: Minor Ninth
maj6: Major Sixth
maj7: Major Seventh
aug: Augmented
dim: Diminished

The chord information is listed in DATA statements from line 2160 and relates to C chords. This information is adjusted for chords of other keys as we shall see. Chapter 2 gives a brief description of chords and Figure 2.11 illustrates some of the more common chords.

The chord information is a list of notes which are included in the chord: it does not show how to construct a chord. The number has been arbitrarily restricted to six notes from octaves 0 to 2. I have arranged the notes used to emphasise the dominant feature of the chord. For example, the pertinent feature of a minor ninth chord is the ninth and the notes in line 2280 have two ninths (D notes in this case for the key of C). The more complex a chord, the more notes of the scale it uses. The notes of the chords are read into the notes$ array in lines 1200 to 1250.

The melody is derived from the chord sequences held in DATA statements beginning at line 2390. The first figure is the number of chords. This is read into the variable, noofchords, at line 1130. The remainder of the data are chords which are read into melody$ at line 1170.

The noofbeats asked for at line 1340 represents the number of beats per bar. We use twice this figure in line 1510 to produce a composition based around 8th notes or quavers. For example, an entry of 4 will produce bars containing 8 quavers.

The tempo is requested at line 1350 and stored in tempo1. The rhythm variations are produced by altering this input value so it is also stored in a second variable, tempo.

Line 1380 asks if you want rhythm variations. The variable sync (for syncopation) is set to TRUE (–1) or FALSE (0) depending upon your answer.

The REPEAT loop running from 1430 to 15460 controls the tune production. A tune will play through each chord in melody$ once. This is controlled by line 1490. Each chord lasts for the length of a bar. A bar consists of quavers equal to the number of beats in the bar (as assigned to noofbeats) multiplied by 2.

If rhythm variations have been selected, line 1520 calls the variations routine. Line 1500 randomly selects one of four variations for the routine prior to each

bar so that each variation will only last for one bar — unless it is selected again.
The play routine at line 1610 is called by line 1530 and plays three notes. It has quite a lot of work to do and tempo values of less than 25 will try to get hold of notes quicker than the BASIC program can supply them, causing uneven and hesitant results.
The play routine calls another routine at line 1740 to analyse the chord. It performs a similar analysis upon the chord as our note analysis routine does on notes. The first thing it does is to look at the first two or three letters to find the key. The other letters are taken as the chord type. Again, using only the sharp (#) sign keeps the routine fairly short. The key is calculated as we would calculate a note at line 1760, and line 1770 gives us a chord number which is simply how far a long chordrange$ it is. We then pick two different notes from the chord in lines 1790 to 1840 and end the routine.
The Get Note routine at line 1880 is like the note analysis routines in the book but it is called twice, once for each of the two notes we picked from the chord. Lines 1620 to 1650, therefore, send two notes to sound channels A and C. The final act of the play routine is to produce a bass note from the key of the chord. This just sustains the root note of the chord, e.g. a C chord will produce a C bass note. All three notes are programmed to rendezvous so they sound at the same time. In stereo, this produces two melody notes from either side and a sustained bass in the centre.
The last routine at line 1970 produces the rhythm variations. It is only called if variations have been asked for, when it is called before every note to determine its duration or tempo as it is termed in this program. The variation in rhythm is produced by altering the value of tempo according to the beat number. Five examples are included and line 1970, along with line 1500, selects the particular syncopation or variation for that bar.

Experimenting with the program

There are two other ways you can alter the output. The first is through the selection of chords used, and the second involves the complexity of the rhythmic variations you introduce.
The data in lines 2390 and 2400 will play a 12 bar blues, the second set of data includes some ninth chords which produce a more jazzy feel. The data at line 2450 produces a chord sequence similar to that found on many electronic music albums and you may prefer the output here without any rhythm variations. The last set of data at line 2520 is very jazzy and bluesy and works best with a slowish tempo and with variations. The use of minor ninth, minor sixth and major seventh chords produces some good jazz style harmonies.
As the output is determined by the chords you put into the DATA statements

you have complete control over the chord progressions and you can experiment with whatever sequence of chords you wish. You can extend the range of chords by including new chord data. Insert the chord name in chordrange$ at line 1070 and increase the notes$ array at line 1090 and the n loop at line 1210 to suit. Be careful just exactly where in chordrange$ you put the new chord name, e.g. if min6 came before min the program would return an incorrect chord number for min at line 1770 as it would find the min in min6 before arriving at the min we want.

For convenience, a major chord is represented by M. In standard notation a major chord will usually stand by itself such as C or G#. In our notation these would be CM and G#M. Giving every chord a symbol in this way simplifies the chord analysis routine.

For more variation, increase the number of notes allocated to each chord. This can simply be used to increase the range so notes occur over more octaves. More notes will also allow you to use more complex chords containing more than six notes. You can try substituting a complete scale for each chord. For example, C major would consist of a normal C scale. C7 would consist of an F major scale as it contains one flat and that is the key of F. C minor would contain the notes of the D# (E flat) major scale. (Minor scales have been discussed in Chapter 2 and you can mix harmonic and melodic scales as you wish.)

Using complete scales may produce dissonant results, because the program picks notes at random. The principle of picking notes from the relevant scale used by the chords is another basis for tune production and we will look at some suggestions as to how this might be accomplished later in this chapter.

A value of 4 beats in the bar will produce music in 4/4 time and each chord will last for one bar. With a value of 2 it will seem as if each chord lasts for half a bar. Actually, all we are doing is reducing the number of quavers in a bar, but by careful selection of this value and the chord data we can arrange an apparent change of chord at any point in the bar. You could arrange the envelopes so the first beat of the bar sounds louder than the others.

Adding rhythm variations

The addition of rhythmic variations is a major part of the program and helps abolish the monotony of a sequence of quavers. The method used can be adapted to produce many varied rhythmic effects. A look at the first example at line 1990 will show how they work.

According to certain criteria which you set up — in this case the note number contained in beat — the duration of a note given by tempo is altered. Unless you want the bar to run out of sync you must ensure that any extra time you give to one note is taken away from another. The example at line 2120

demonstrates what can happen when you don't. It still produces a good variation and keeps the listener on his toes because it is not obvious where the beat has gone. A number of out-of-sync rhythms may be too disorderly but one or two certainly add interest to a piece.
If you look at the other examples in the routine you will see how they work. Generally, time is taken from one note and given to another so each bar contains eight notes. The principle can be adapted to produce many different rhythm configurations.
The routine at line 2050 is slightly different in that it adds notes to the bar. It does this by calling the chord analysis routine for more notes, which it adds with a duration value of half that of tempo.
Instead of altering tempo, the variations could be stored in DATA statements or in arrays. The above method, however, will adapt to whatever noofbeats happens to be.
The variations at lines 2080 and 2120 can be included by adding the line numbers to line 1970. By altering this and the sync pointer in line 1500 you can produce quite a varied output. You will find that for tempos of less than 25 the computer cannot always respond to tempo variations, because of the time taken to calculate note values.

Further extensions and modifications

As you will have realised, the program produces a duophonic melody with a third note in the bass. The bass was introduced to reinforce the tonality of the chord, to provide a foundation for the tune and to add a little depth. The two melody notes always have the same duration but could be altered to have independent durations. This would create a more truly polyphonic effect. The bass could also be given its own duration values and be made to play say the root and fifth intervals of the chord. This would maintain the chord's tonality while providing yet more variation and interest. To top it all, you could turn one channel into a drum track.
The composition routines are quite complex and take time to calculate which is why the program can't play very fast compositions. A worthwhile exercise would involve setting all the relevant data into arrays and reading directly from these. The program would then only have to perform the calculations once. Alternatively, instead of a continuous composition, the pitch numbers and durations could be stored in arrays and you would be able to play this back at any tempo. I prefer continuous compositions but it is worth experimenting yourself.
There are other ways we can choose notes from a scale which relate to a particular chord. In fact, there are probably many other ways to produce computerised music and several more chapters, if not books, could be written

on the subject. I do not want computer composition to dominate this book, however, so I shall only discuss the principles of another method and leave you to work out the details. The programming should be only a little more complicated than the programs already listed.

Applying further control to random note selections

This chapter and Chapter 10 both look at ways of controlling the random element in computer compositions. There must be a random element otherwise there would be no original music but we must be able to control it and shape it in order to produce the results we want.

If you put complete scales into the DATA statements of Program 11.1 you will see that the results are quite different to those produced when we limited the choice to six notes. This is because the program is choosing the notes with equal probability. If you include eight notes in a scale, each note has a one in eight chance of being selected. This has two effects which detract from the melodic and harmonic effect.

First of all, the notes chosen are too random within any given harmonic framework. In human compositions, certain notes occur more often than others — the melody over a C minor chord will probably contain more C, G and E flat (D#) notes than any others. In the program, all notes have an equal chance of being selected. Our first step, therefore, would be to arrange a probability table which gave more weight to notes which were in the chord or which were more likely to be heard with that chord.

Initially, for simplicity, we could ignore all accidentals. A probability table for the chord of C minor might then look like this:

C0	20%
D0	5%
D#0	15%
F0	10%
G0	15%
G#0	10%
A#0	5%
C1	20%

The figures are based on rule of thumb and could be extended to cover at least one more octave. You could also include a B natural. Implementation of this set of rules will show a marked improvement in the melodic output but if the choice of notes ranges over more than an octave the melody will have a tendency to skip large intervals between notes which is not common in normal compositions. We can overcome this by applying another set of probability restrictions.

Improving the melody

If you examine almost any piece of music you will see that the melody line moves up and down the scale, usually a note or two at a time, and very seldom does it jump more than an interval of a fifth. Our program can jump the full range of available notes and will often cover intervals much larger than a fifth. This is the second way in which its melody production is not quite satisfactory. To this, we can apply a set of probabilities in a similar way to the probabilities we gave to the notes of the scale. The most common interval jump is a scale step, followed in probability by two scale steps and three scale steps. If we measure melodic movement in scale steps we will avoid accidentals. A probability table for melodic intervals might then look like this:

+1	30%
−1	30%
+2	15%
−2	15%
+3	5%
−3	5%

Again, these figures should only be used as a starting point and altered as results demand. When these two sets of rules are combined you should see a tremendous improvement in the melodic output.

So far, we have not mentioned note durations. This is probably one of the most difficult areas of computer composition to program successfully. In our program the best results will probably come from a pre-programmed set of note durations or simply a string of quavers. This will probably produce a Bach-like piece.

The output from such a program would obviously be much improved if it played all three channels. Because the two melody channels will play their own parts, and because each output will be independent of the others (apart from their reliance on the common chord), you may well find that notes which may have tended to clash before seem to pass over one another as, for example, one channel plays a falling melody and the other a rising one.

You can try restricting the secondary channel to playing the actual notes of the chord to add support to the harmonic base.

The bass notes can be controlled in two ways. First, the durations can be fixed to a certain number of beats in a bar, and secondly you can restrict the choice of note. As the purpose of a bass line is to support a harmonic framework, if it is restricted to the root and fifth and possibly the third, the harmony will be reinforced.

Designing and developing programs

Within the framework of rules set out above, you have scope to apply your own ideas. One thing to be aware of is not to ask the computer to get a note through the stepping procedure that it has been forbidden to get through the scale table. For example, if it was currently playing a G1 and it was ordered to move down a step, it may generate a percentage on the scale table which said F1 is out of bounds. The overall note selection must take both these procedures into consideration. One way to do this would be to present the computer with only valid notes to choose from.

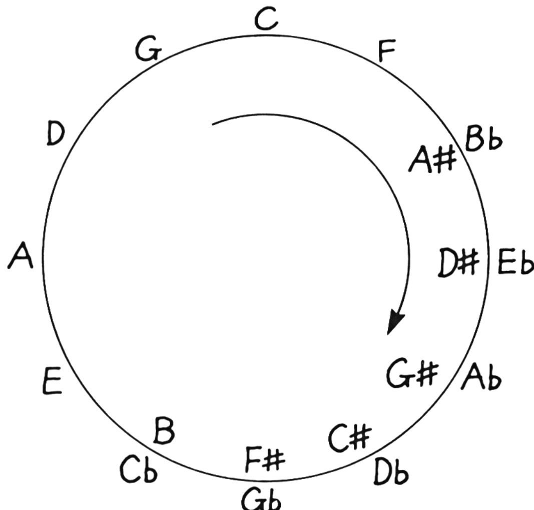
Like most computer programs and ideas, these compositional programs can be added to and developed. Here are some more ideas for further investigation. They are mentioned briefly as food for thought and as suggestions for further experiments. Detailed instructions and listings could quite easily consume a full book.

It would be interesting and useful to allow the operator or programmer to alter various parameters as the program is running. This could be accomplished by input from the keyboard or you could employ a more subjective form of control by allowing various parameters to be controlled by a joystick. This has great possibilities, allowing you to affect the way the music is composed.

The User Guide provides details about joysticks and how to read values into the computer from them. The parameters you could control are key, tempo, rhythm variations and even the chords used. I shall describe one possible method of altering the chords. There is a cycle of chord progressions known as the Circle of Fifths which is illustrated in *Figure 11.1*. Seventh chords, like C7 have a tendency to want to move to the chord a fifth down the scale. So a C7 will want to move to an F, and an F7 will want to move to an A# (B flat). The movement sounds satisfying, final and complete and so we say that a C7 chord resolves to F.

If you do not have access to a musical instrument to prove this, you should easily be able to program your Amstrad to play through the cycle to confirm this harmonic behaviour. The Circle of Fifths is very useful in composition and shows how chord progressions tend to move.

If movement of the joystick accessed the chords in a similar order to the circle, i.e. moving it right would move through C, F, A# etc., and if the joystick position was not exact, the change of key would not sound out of place as might be the case if the chords were arranged in chromatic (semitone) order. The purpose of joystick control is not necessarily to specify the exact chord, but rather to suggest a harmonic progression. Once the computer starts to produce good results, you may want to save them. All the programs listed so far, apart from Program 10.2, are designed to play a continuous composition which is composed instantly, as the program runs.



You could add a facility which would permit the computer to play a bar or a phrase (say four bars), then stop and ask if you wanted to save it. In this way you could build up a catalogue of the best of the Amstrad. Individual bars or phrases could be played again in any order. The tune parameters would initially be stored in an array and later saved to tape or disc. (See the User Guide for details of file handling.)

The Upside-Down Composer Program

The Note Analysis Program in Chapter 10 produced an output which varied according to the note frequency of an existing tune. We can use existing tunes as an information base to produce a different output. One such method is to perform a mathematical operation upon the pitches representing the notes. Notes and scales have a decidedly mathematical relationship with one another and the computer provides an excellent means of rearranging notes according to mathematical rules. You can apply all sorts of mathematical functions to a set of notes to produce a wide range of results but for simplicity only one example will be given here. This is the Upside-Down Composer Program which reverses the pitch of the notes so that high notes will be played low and low notes will be played high. The results are interesting and often quite humorous. Other mathematical permutations will produce quite different results. We already have complete programs to play Mozart's Rondo, the Liberty Bell and the Dance of the Sugar-plum Fairy (from Chapter 9), so we will try out our method on these. The first step is to find the highest and

lowest notes in a piece and then find a central note around which these revolve. For example, if the highest and lowest notes were C2 and G#−1, they would revolve around A#0 which is half way between the two. Add the following lines to any program produced from Program 9.2.

```
100 REM PROGRAM 11.2
110 REM Upside-Down Routine
120 REM Add To Program 9.2
130 :
1025 hipit=16:lopit=3822
1026 pp=349.228:REM See Text
1843 GOTO 1850
1844 IF freq<pp+2 AND freq>pp-2 THEN 185
0
1845 semi=1
1846 IF freq>pp THEN GOSUB 1862:GOTO 185
0
1847 IF freq<pp THEN GOSUB 1866:GOTO 185
0
1851 IF pitch>hipit THEN hipit=pitch
1852 IF pitch<lopit THEN lopit=pitch
1861 :
1862 freq=freq/2^(1/12)
1863 IF freq<pp+2 AND freq>pp-2 THEN fre
q=freq/2^(semi/12):RETURN
1864 semi=semi+1:GOTO 1862
1865 :
1866 freq=2^(1/12)*freq
1867 IF freq<pp+2 AND freq>pp-2 THEN fre
q=2^(semi/12)*freq:RETURN
1868 semi=semi+1:GOTO 1866
```

Commentary

The first step is to find the highest and lowest notes. The computer can do this for you. Run the modified program and then print out hipit and lopit in command mode. Look up these notes in the Appendix and find the middle note between these two. The central point does not have to be exact and you can offset it but, especially in multi-part tunes, you will probably find that an exact central pitch will play in tune better than offset values. If the notes produced by the new routine stray too far, there is always the danger they may try to produce out of range pitch numbers which will result in errors.

When you find the central note, assign its frequency value to the pivot point, pp, in line 1026. Then delete or REM out line 1843. The value for the eight-bar version of Rondo is 349.228 or 369.994. The full version is 329.628 or 342.229. For the Liberty Bell try 415.305, and 622.254 for the Sugar-Plum Fairy.

There are two routines you can use, depending upon whether the candidate note is above or below the pivot point. The first step is to see how far above or below the pivot point the note is. The two routines (line 1862 to 1864 and 1866 to 1868) work in similar ways. The formulae in lines 1862 and 1866 will increase or decrease the frequency in steps of one semitone. They are variations on the frequency formula we explored in Chapter 4. Lines 1863 and 1867 check the new frequency against the pivot point. We allow a tolerance of plus or minus 2 because the formulae will not produce the exact frequency but it usually won't be out by more than 1 so 2 should cover it. If a tune gives trouble, check this range. The variable semi keeps count of the number of semitones the note is above or below the pivot point, and is then used to calculate the new frequency (in lines 1863 and 1867) using the same formula. The addition of these calculations will increase the time it takes the computer to store the necessary data in the arrays.

The duration values are the same which is what makes the tunes half recognisable but you could apply a similar function to the duration values just to confuse your listeners, i.e. make long notes short and vice versa.

The above example is very simple. Try modifying the notes with SIN and COS functions or an algebraic expression. You could alter the pitch values so the music plays in steps other than the semitone intervals of the western scale (see Chapter 4 for details of microtonal scales).

Retrogressive progression

As the notes are stored in arrays, we can make the program play them in a different order, even backwards. The alterations are quite simple. Replace the following lines in any program derived from Program 9.2.

```
10 REM PROGRAM 11.3
20 REM Playing a Tune Backwards
30 REM Alter These Lines In PROGRAM 9.2
40 REM Also Rendezvous the
50 REM Last Notes - See Text
60 :
1660 voca=va+1:vocb=vb+1:vocc=vc+1
1700 WHILE voca>0 OR vocb>0 OR vocc>0:WE
ND
```

```
1890 voca=voca-1:IF voca<1 THEN RETURN
1950 vocb=vocb-1:IF vocb<1 THEN RETURN
2010 vocc=vocc-1:IF vocc<1 THEN RETURN
```

Commentary

This should require little comment. Instead of reading the arrays forwards, we read them backwards. You should ensure that the very last notes of each channel rendezvous so that all channels begin in sync. You can either add a spurious last note, e.g. a rest, to each channel and add 1 to va, vb and vc, or you can physically add a rendezvous to the note data. The last notes in each channel may not all sound at the same time so make sure you rendezvous the correct notes. In Program 9.2 — the unaltered version — this would mean these additions:

```
2150 DATA &38,E1,12
2260 DATA &38,E-1,12
2370 DATA &38,R,12
```

Use backward envelopes. Combine the effects of the last two programs and alter the tempo of the pieces. Try playing the notes in random order.

Sing-a-long-a-matic

There are now speech packs available which can be programmed with a pitch to enable the production of a singing voice. If we could link a music composition program to a poetry generation program we would have the latest in singer-songwriters. Even if this technology is not available for the Amstrad — yet — we can still link a music program to a poetry generation program. At its simplest we could count the syllables in the verse and compose music with the same number of notes. Anyone lucky enough to have a speech system could let the computer talk the words over a musical background in the style of Leonard Cohen. One such system is the dk'tronics speech system, which retails for under £40. The package includes two external speakers which enhance the sound and provide stereo output. The system plugs snugly into the back of the Amstrad and has an adjustable volume control.

These suggestions are just the beginning. Computer compositions are one area in which relatively few experiments have been done. There is plenty of scope for new ideas.

APPENDIX 1

Notes, Frequencies and Pitch Numbers

In the table which follows, the octave numbers used in the note names follow the User Guide's specification. The numbers in parentheses offer an alternative octave numbering system. See Chapter 4 for further details.
The frequencies listed are those produced by notes in the western scale. The pitch numbers do not produce these frequencies exactly but the differences are so small as to be virtually unnoticable. For those interested, the relative errors are listed in Appendix VII of User Guide.
Middle C lies in octave 0, i.e. C0.

NOTE NUMBER	NOTE	FREQUENCY IN HERTZ	PITCH NUMBER
	OCTAVE -3 (1)		
1	C-3	32.703	3822
2	C#-3	34.648	3608
3	D-3	36.708	3405
4	D#-3	38.891	3214
5	E-3	41.203	3034
6	F-3	43.654	2863
7	F#-3	46.249	2703
8	G-3	48.999	2551
9	G#-3	51.913	2408
10	A-3	55.000	2273
11	A#-3	58.270	2145
12	B-3	61.735	2025
	OCTAVE -2 (2)		
13	C-2	65.406	1911
14	C#-2	69.296	1804
15	D-2	73.416	1703
16	D#-2	77.782	1607

NOTE NUMBER	NOTE	FREQUENCY IN HERTZ	PITCH NUMBER
17	E-2	82.407	1517
18	F-2	87.307	1432
19	F#-2	92.499	1351
20	G-2	97.999	1276
21	G#-2	103.826	1204
22	A-2	110.000	1136
23	A#-2	116.541	1073
24	B-2	123.471	1012
	OCTAVE -1 (3)		
25	C-1	130.813	956
26	C#-1	138.591	902
27	D-1	146.832	851
28	D#-1	155.564	804
29	E-1	164.814	758
30	F-1	174.614	716
31	F#-1	184.997	676
32	G-1	195.998	638
33	G#-1	207.652	602
34	A-1	220.000	568
35	A#-1	233.082	536
36	B-1	246.942	506
	OCTAVE 0 (4)		
37	C0	261.626	478
38	C#0	277.183	451
39	D0	293.665	426
40	D#0	311.127	402
41	E0	329.628	379
42	F0	349.228	358
43	F#0	369.994	338
44	G0	391.995	319
45	G#0	415.305	301
46	A0	440.000	284
47	A#0	466.164	268
48	B0	493.883	253

NOTE NUMBER	NOTE	FREQUENCY IN HERTZ	PITCH NUMBER
	OCTAVE 1 (5)		
49	C1	523.251	239
50	C#1	554.365	225
51	D1	587.330	213
52	D#1	622.254	201
53	E1	659.255	190
54	F1	698.457	179
55	F#1	739.989	169
56	G1	783.991	159
57	G#1	830.609	150
58	A1	880.000	142
59	A#1	932.328	134
60	B1	987.767	127
	OCTAVE 2 (6)		
61	C2	1046.502	119
62	C#2	1108.731	113
63	D2	1174.659	106
64	D#2	1244.508	100
65	E2	1318.510	95
66	F2	1396.913	89
67	F#2	1479.978	84
68	G2	1567.982	80
69	G#2	1661.219	75
70	A2	1760.000	71
71	A#2	1864.655	67
72	B2	1975.533	63
	OCTAVE 3 (7)		
73	C3	2093.004	60
74	C#3	2217.461	56
75	D3	2349.318	53
76	D#3	2489.016	50
77	E3	2637.021	47
78	F3	2793.826	45
79	F#3	2959.955	42

NOTE NUMBER	NOTE	FREQUENCY IN HERTZ	PITCH NUMBER
80	G3	3135.963	40
81	G#3	3322.438	38
82	A3	3520.000	36
83	A#3	3729.310	34
84	B3	3951.066	32
	OCTAVE 4 (8)		
85	C4	4186.009	30
86	C#4	4434.922	28
87	D4	4698.636	27
88	D#4	4978.032	25
89	E4	5274.041	24
90	F4	5587.652	22
91	F#4	5919.911	21
92	G4	6271.927	20
93	G#4	6644.875	19
94	A4	7040.000	18
95	A#4	7458.621	17
96	B4	7902.133	16

APPENDIX 2

Entering, Protecting and Working with the Programs

The most annoying thing that can happen to a programmer is to lose a program he or she has spent hours typing in. This happens to every programmer at least once in their life. It should happen *only* once because you vow never to let it happen again. It can happen for a variety of reasons:

1) You simply switch off and forget to save the program.
2) You save the program, but only once and the copy won't load again.
3) Someone, possibly yourself, trips over a wire and unplugs the equipment.
4) You run a program before saving it and it crashes.

Most of these problems are easily avoided with a little thought. The following suggestions may help:

1) See that all wires are safely out of the way and cannot be caught by moving hands or feet.
2) Make it a rule *always* to save a program before running it. CAT it, too, if you have the patience. If you are fortunate enough to have a disc drive, this is no problem but it can be laborious with a cassette recorder.

Using cassettes

When using cassettes, I find the best thing to do to ensure trouble-free loading and saving is to clean the heads regularly. Cassettes, cheap ones especially, shed their coating onto the heads which impairs the signal and makes secure saving a problem. It pays, therefore, to purchase good quality tapes.
I would advise against using a cassette tape with built-in cleaner as these sweep the rubbish from the heads into themselves and sometimes have a tendency to sweep it out again. Far better is a head cleaning fluid, available from most chemists and hi-fi stores, which can be applied with cotton buds.

Entering programs

The programs have been listed directly from the computer at a print width of 40 characters per line — the number of characters per line on a Mode 1 screen. This will help when checking your programs against the listings.

You will probably be aware that function keys can be programmed to speed up program entry and development. Details are given in the User Guide Chapter 1 Page 13 and it is worth developing your own initialisation program to set up keys with commonly used commands.

I have tried to use meaningful variable names throughout the programs and all variables are in lower case. You may have noticed that the Amstrad automatically converts all keywords to upper case. This should help you to spot entry errors and assist during debugging.

I have also refrained from using too many 'clever' programming techniques which may make the logic and programming difficult to follow or understand. My aim was always that the reader should understand how the programs work and beginners may well be confused by such material. Those more expert will, I hope, let loose all the tricks of our trade as they experiment with the programs.

Many programs could be shortened and speeded up by the use of tighter programming techniques and multi-statement lines.

Merging programs

Some programs listed in this book use the same or similar routines and to save time and effort many have been designed to be inserted directly into other programs. These program will not run by themselves. For example, Program 5.5 must be merged with Program 5.4.

The original programs are all numbered in steps of 10 but, for obvious reasons, this was not always possible with the other programs. Most of these are fairly short and can be added to the other programs by loading the first one and typing in the second. It is essential that every single line be entered, including the blank ones (containing only a colon) which are sometimes there to delete unwanted lines from the original program.

Alternatively, programs can be merged using the MERGE command (see the User Guide Chapter 8 Page 27). If a single program is saved separately, it can be MERGEd into several others. This is also the method to use if you take advantage of the cassette offer, details of which are given in the Introduction.

Re-setting the random number generator

When experimenting with some of the programs you may want to repeat a set of random numbers. This can be done by first setting or seeding the RND

function with RANDOMIZE such as:

RANDOMIZE 1

Details are included in the User Guide in Chapter 8 Page 37. This can be useful if you want to check a program's output.
Above all, experiment and have fun.

Other titles from Sunshine

SPECTRUM BOOKS

ZX Spectrum Astronomy
Maurice Gavin — **£6.95**
ISBN 0 946408 24 6

Spectrum Adventures
A guide to playing and writing adventures
Tony Bridge & Roy Carnell — **£5.95**
ISBN 0 946408 07 6

Spectrum Machine Code Applications
David Laine — **£6.95**
ISBN 0 946408 17 3

The Working Spectrum
David Lawrence — **£5.95**
ISBN 0 946408 00 9

Master your ZX Microdrive
Andrew Pennell — **£6.95**
ISBN 0 946408 19 X

COMMODORE 64 BOOKS

Mathematics for the Commodore 64
Czes Kosniowski — **£5.95**
ISBN 0 946408 14 9

Advanced Programming Techniques on the Commodore 64
David Lawrence — **£5.95**
ISBN 0 946408 23 8

Graphic Art for the Commodore 64
Boris Allan — **£5.95**
ISBN 0 946408 15 7

Commodore 64 Adventures
Mike Grace — **£5.95**
ISBN 0 946408 11 4

Business Applications for the Commodore 64
James Hall — **£5.95**
ISBN 0 946408 12 2

The Working Commodore 64
David Lawrence **£5.95**
ISBN 0 946408 02 5

Commodore 64 Machine Code Master
David Lawrence & Mark England **£6.95**
ISBN 0 946408 05 X

ELECTRON BOOKS

Graphic Art for the Electron
Boris Allan **£5.95**
ISBN 0 946408 20 3

Programming for Education on the Electron Computer
John Scriven & Patrick Hall **£5.95**
ISBN 0 946408 21 1

BBC COMPUTER BOOKS

Functional Forth for the BBC computer
Boris Allan **£5.95**
ISBN 0 946408 04 1

Graphic Art for the BBC computer
Boris Allan **£5.95**
ISBN 0 946408 08 4

DIY Robotics and Sensors for the BBC computer
John Billingsley **£6.95**
ISBN 0 946408 13 0

Programming for Education on the BBC computer
John Scriven & Patrick Hall **£5.95**
ISBN 0 946408 10 6

Ian Waugh has written **'Making Music on the Amstrad CPC 464 and 664'** for the many owners of these micros who would like to take advantage of their sophisticated sound facilities, to write musical programs.

Ian Waugh is a professional musician and keen micro enthusiast, who shows you how you can make the Amstrad play music of all kinds, including effects like vibrato, echo, trills, polyphonic music and even sound effects like seagulls and ricochets. Did you know, for example, that you can turn your Amstrad into a drum synthesiser or rhythm unit using a BASIC program?

All the programs are written in BASIC, and their operation is fully documented, line by line, so that newcomers to computing can understand them, while for those new to the concepts of music there is an introduction to the basic theory.

Some of the later chapters cover computer compositions, harmony and transposition.

Here's what **Educational Computing** had to say about **'Making Music on the BBC Computer'**, one of the author's previous books:

> 'This book is a real springboard to the many musical applications of the BBC Micro. It's not quite instant Mozart, but it is the best book on making music on the BBC yet'.

And **Your Commodore** described **Commodore 64 Music** by Ian Waugh as:

> 'A most instructive and entertaining book . . . highly commendable and excellent value for the price'.

The Author

Ian Waugh is a professional musician, whose interests cover most forms of music. He is the author of **Making Music on the BBC Computer,** and **Commodore 64 Music,** both of which have been well reviewed in the computer and music press.

£6.95 net

ISBN 0 946408 82 3

MAKING MUSIC ON THE AMSTRAD CPC 464 AND 664

Ian Waugh

SUNSHINE

Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE

https://acpc.me/