making music on the BBC computer

a musician's guide to programming

ian waugh

First published 1983 by:
Sunshine Books (an imprint of Scot Press Ltd.)
12-13 Little Newport Street,
London WC2R 3LD

ISBN 0 946408 26 2

Cover design by Graphic Design Ltd.
Illustration by Stuart Hughes.
Typeset and printed in England by Commercial Colour Press, London E7.

DIGITALLY REMASTERED ON RISC OS COMPUTERS,
DECEMBER 2011.

CONTENTS

Contents in detail

number of steps — the Pitch Graph Generator — apparent peculiarities of the pitch envelope — experimenting with the programs — instrument characteristics — producing other waveforms.

CHAPTER 6
Musical Miscellanea

Vibrato and tremolo: pitch and amplitude modulation — creating vibrato with the pitch envelope Envelope Comparisons — producing tremolo effects — Tremolo Demonstration — trills: a special kind of vibrato — Military Music Introduction — echo and reverberation — commercial echo units — producing echoes on the BBC micro — Echo Production — Echo Using a Procedure — Pseudo Echoes Using Single Envelopes — using the pitch envelope to play tunes — chorus, phasing, flanging and other spatial effects — Chorus Effects — beat frequencies: the weaving in and out — French Accordion Music — the ring modulator: producing bells and other ringing noises — the frequencies produced by the sound chip — the out of tune chip — bells and the BBC micro — Bells and Chimes.

CHAPTER 7
Zaps and Zings and Other Things

White noise — simple sound effects — Examples of Channel 0 —Machine Gun — Ricochet — Cymbal — Creature — Mad Factory —Space Ship — exploring the sound channel — Sound Effects Generator — using channel 0 to produce otherwise unobtainable low notes — use of the lower octave — Rhythm Unit — the CAPS LOCK and SHIFT LOCK fights and the ADVAL function — using sound effects in utility programs — Sea, Surf & Seagulls.

CHAPTER 8
Playing the BBC Micro

Using the BBC micro as a musical keyboard — monophonic and polyphonic instruments — the BBC micro as a Monophonic Keyboard — Keyboagd Display — alternative methods of note production — 3—Note Polyphonic Keyboard — Bass Sequencer with Duophonic Keyboard — altering the bass riff — developing the sequencer.

CHAPTER 9
Making Micro Music

Playing two— or three—part tunes — selecting the notes and octave range — 1 Channel Version of Mozart's Rondo Alla Turca — the tracking method — the negative ADVAL method — 3 Channel Rondo Alla Turca — debugging the data — *SPOOL Routine — more tunes to play: Dance

To my Mother and Father

. . . with love and gratitude for their care, encouragement and understanding not only during the production of this book but throughout my entire life.

Introduction

This book is for everyone with a BBC micro. Whatever your reasons for buying a computer, your decision to purchase a BBC micro 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 per cent of all information we receive from the outside world comes through our sense of sight, so it is hardly surprising that computer 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 than you may realise. The next time you're 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. It is a case of the sum of the parts being greater than the whole and the addition of sound to any computer program, even a utility program, can more than double its enjoyment and effectiveness.

As well as sound effects, the BBC micro can be programmed to produce music. This opens up a totally new area for exploration, an area without precedence in the world of personal computing. 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 is imbibed with the speed and decision-making attributes of a computer.

The Operating System

As the BBC micro underwent development, various additions and modifications were made to the Operating System, which resulted in the issue of at least three separate OS chips. The first was called OS 0.1: it contained an infamous number of bugs and was way below specification. Subsequent issues cured the bugs and included many 'enhancements' and

new functions which made the BBC micro even more powerful. The main problem with OS 0.1 was the notorious 'cassette bug' which sometimes refused to save the first block of a program. This and other bugs relating to plotting routines and data handling with PUT and GET have been fixed.

Enhancements include the use of the function keys with the SHIFT key and CTRL key to produce teletext effect codes in mode 7. These allow colour codes and graphics characters to be printed directly on to the screen. Additions have been made to the PLOT command to include fill routines and there are a host of new *FX commands and OSBYTE calls which give direct access to the Operating System through BASIC commands. Some of these are fisted on page 418 of the User Guide, but many more are undocumented and are only coming to light through exploration.

The new Operating Systems should have been upward compatible but, for a variety of reasons, some programs which worked on OS 0.1 will not work on OS 1.2. Any programs using the new features of OS 1 .2 will not, of course, work on OS 0.1. If you do not have OS 1.2, it is really worthwhile having it fitted because, soon, most new software will not run on the old system as programmers will use more of the new commands.

The programs in this book were written with OS 1.2, although they may run on earlier versions. To check what version is in your machine type:

```
*FX0
```

This will print the OS number. If you need a new Operating System, contact your dealer.

About this book

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 BBC micro.

The programs are written in a fairly structured manner and are documented so that you can understand the workings behind them. Generally, they will not have many frills, which should minimise the time required to enter them and should help to cut down on mistakes. The overall appearance of a program is as important as the performance, and suggestions regarding the finishing touches are made where relevant. These can be added later to suit your own taste and style. 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.

This book was written to be read from Chapter 1 onwards, but you can dip into it at whichever chapter takes your fancy. For those who decide to read it so, I make no apologies for the odd repetition of information and the constant referral to other chapters. Those well versed in music and computing will forgive me; those who are not so accomplished will, I hope, thank me.

Whether you are looking for a new laser sound for your latest arcade game, whether you want to write a new tune for your musical doorbell, whether you want to use the computer to help you learn about music, whether you want to add something to the business utility program you've written or whether you simply want to see what you can do with the SOUND and ENVELOPE commands, I hope this book contains something for you and that it will encourage you to carry on experimenting from where I leave off.

Before you begin, perhaps I can refer you to the appendices which contain hints and tips about entering and merging programs and other information you may find of interest. Please read them before entering programs.

Programs tape

Some of you might be unwilling to enter some of the longer programs in this book, because of the time this will take. For anyone who does find this a problem, a tape of these programs is available from me, Ian Waugh, do Sunshine Books, 12-13 Little Newport Street, London WC2R 3LD, for £5.95, including p&p. Available for readers in the UK only.

Thanks and acknowledgments

I would like to thank John Paisley for his help in measuring and listing the output frequencies of the sound chip and Leslie Thwaites for checking my early programs.

Thanks too, to David Lawrence for his encouragement and to Jenny Ireland for her international persistence.

Warning: When entering programs from this book, please be careful that you key them in correctly. Copy from a computer printer can be rather faded - commas, in particular, may look like full stops at first glance.

CHAPTER 1
What is Sound?

This chapter looks at sound from the computer's point of view. In order to assess the sound commands on some computers, it is necessary to POKE numeric values into various registers. This is clearly a very user-unfriendly way of creating sounds and music and, thanks to the ingenious programming of the BBC micro's OS, we have a much easier and more versatile method of controlling the sound output - through the BASIC SOUND and ENVELOPE commands.

Even a few casual experiments with these commands will reveal how complex and difficult they can sometimes be to control. The User Guide, excellent though it is, devotes only a total of 20 pages to the sound facilities: more information is required to get the best from the system. 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 we are dealing with something we cannot see. A sound is produced when an object is struck or rubbed or, in scientific terms, otherwise excited. This causes the object to vibrate which in turn causes the air to vibrate. These vibrations are sensed by the ear and we perceive them as sound.

Sound does not only travel through air, it travels through gases, liquids and solids. You can see sound waves passing .through water by tapping the side of a rain barrel. However, in a vacuum, such as on the moon or in space, there is nothing for sound vibrations to travel through and such environments are totally silent. When you next watch a space movie and see a spaceship - or planet - blow up with an ear-shattering explosion, you know that the vibrations produced by such an explosion would have nothing to travel through and the spectacle would, in reality, be

accompanied by silence. Don't let this deter you from including suitable sound effects in your games. Films are made for entertainment.

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 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. Clearly, unless we hook up the computer to some outboard equipment, we cannot create sound in this way.

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 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 fight 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 you often need 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.

```
 10 REM PROGRAM 1.1
 20 REM Sine Wave Plotter
 30
 40 MODE 4
 50 REM Define Windows
 60 VDU28,0,4,39,0
 70 VDU24,0,0;1279;850;
 80
 90 REPEAT
100 INPUT"Frequency (1-10)",Freq
110 INPUT"Amplitute (50-400)",Amp
120 PROCSine
```

```
  130 PRINT"Press SPACE to enter another
 wave"'"'C' to clear screen, 'F' to fini
sh"
  140 REPEAT
  150 Key=GET
  160 UNTIL Key=32 OR Key=67 OR Key=70
  170 IF Key=67 THEN CLG
  180 UNTIL Key=70
  190 MODE7
  200 END
  210
  220 DEF PROCSine
  230 VDU29,0;450;
  240 MOVE0,0
  250 FOR Time=0 TO 1279 STEP 10
  260 DRAWTime,Amp*SIN(RAD(Freq*Time))
  270 NEXT Time
  280 ENDPROC
```

Program notes

The program houses a simple graph-plotting procedure between lines 250 and 270. Line 260, which we will extend later, performs the calculations. All the VDU calls are very well explained in the User Guide.

At line 150, GET returns the ASCII value of the input character (see the User Guide page 263).

Try inputting 1 for frequency and 50 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. Our frequency figures are scaled down for the program and you can assume an input of 1 represents a frequency of around 100 cycles per second. 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 260 with:

```
260 DRAWTime,Amp *SIN(RAD(Freq*Time))+Amp*SIN
(RAD(Freq *2*Time))
```

you are adding two sine waves together, and can see the effects of additive synthesis. Notice how the waveform changes. You can add more sine waves in line 260 by modifying the variable Freq in the expression:

```
Amp*SIN(RAD(Freq*Time))
```

and tagging it on to the line with a + as in the above example. If you modify the value of Amp such as

```
Amp*.5*SIN(RAD(Freq*Time))
```

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

The sound of the BBC micro

Another form of synthesis, known as subtractive synthesis, takes a waveform and filters out certain harmonics. A tone control is a simple filter and blocks out the higher frequencies as you increase its effect. This method is more common than additive synthesis and is in general use in most synthesiser systems.

Before you 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 a few harmonics, the square wave sounds a little like a clarinet and the sawtooth wave produces a sound with reed-like qualities. Type and run this program:

```
10 FOR Pitch=1 TO 253 STEP 4
20 SOUND1,-15,Pitch,10
30 NEXT Pitch
```

This will play 64 notes, each a semitone apart: it covers the five octave range of the sound chip. Can you tell which type of waveform the sound chip is using to make the sounds? As you listen, you will notice that the lower notes sound quite rich and full but, as the notes rise, they seem more percussive and lose their warmth. The lower notes almost have a clarinet-like quality about them and you would be right in thinking that the sound chip produces a square wave. In reality, it is a distorted square wave and its waveform alters as the pitch alters. This is why you hear a change in tonal quality as the notes get higher.

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: it is not necessary to know exactly how it does this, but the result is, perhaps obviously , an electronic sound. What restrictions this places on our programming, we will see during the course of this book.

In order for a sound to exist at all, it must have four parameters:

1) pitch
2) volume
3) duration
4) 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: high and low notes

The pitch of a note means how high or low it is on the musical scale and the word frequency is often used synonymously with it. (Frequency is more properly an attribute of the waveform, in terms of 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 (scientifically referred to as hertz and abbreviated to Hz.). The upper frequency limit will drop as a person gets older but no one should have any trouble hearing the range of the BBC sound chip.)

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

On a piano, the pitch of the notes is fixed. You can't, 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 sounds like this:

```
10 FOR Pitch=53 TO 149
20 SOUND1,-15,Pitch,1
30 NEXT Pitch
```

This is a favourite sound easily created on most synthesisers. The same thing on a piano, harp xylophone would sound something like this:

```
10 FOR Pitch=53 TO 149 STEP 4
20 SOUND1,-15,Pitch,1
30 NEXT Pitch
```

Here, we are playing in semitones and you can hear the discrete pitches: this is known as a glissando. Both effects are much used by jazz musicians and can add a human touch to synthesised music.

The BBC micro divides each semitone into four and, when we get down to this minute level of note division, the notes tend to blur into one another as this shows:

```
10 FOR Pitch=101 TO 149
20 SOUND1, -15,Pitch,10
30 NEXT Pitch
```

You can probably still hear the separate pitches, but they are generally too close together for normal western ears to appreciate musically. If you replace line 10 by:

```
10 FOR Pitch=1 TO 255
```

you will hear that the scale is uneven in parts, indicating that the pitches produced by the sound chip are not equally spaced. 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 and, at first, loudness as a quality of sound may seem rather simple and not as important as the others. It is not quite as straightforward as that

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. For most purposes this will make little difference to our experiments and we can simply set volume levels as we require them but, 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 and set all the parts at the same volume level, 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 of the sound chip, and can only be overcome by altering the volume of individual lines and notes where required. You will find that generally it 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 sounds immediately upon playing and then dies away. A violin takes just a fraction of a second before its note reaches 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:

```
40 FOR Volume=-l5 TO -1
50 SOUND1,Vo1ume,53,1
60 NEXT Volume
```

It sounds like a percussive instrument being tapped smartly. Now try this:

```
10 FOR Volume=l TO -15 STEP -1
20 SOUND1,Volume,53,1
30 NEXT Volume
```

This 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 then stop. If you run both programs together, 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 BBC micro to create lots of interesting effects.

Rather than control the SOUND command with a FOR. . . NEXT loop, we can use the ENVELOPE command to create a predetermined set of volume characteristics like this:

```
10 ENVELOPE1,1,0,0,0,0,0,0,127,-1,-1,-1,126,1
20 FOR Pitch=53 TO 101
30 SOUND1,1,Pitch,10
40 NEXT Pitch
```

This creates a percussive envelope and produces a piano-like sound. Alter line 20 to:

```
20 FOR Pitch=149 TO 245 STEP 5
```

and notice how, because of the change in pitch, it sounds more like a xylophone. Change line 120 back, and alter line 10 to:

```
10 ENVELOPEl,1,0,0,0,0,0,0,3,-100,0,0,126,0
```

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. If you alter the Pitch values,

notice how it loses its violin-like quality.

All these effects use the same sound generator and demonstrate how the ear can be deceived by clever control of envelope parameters. We will look at this more closely later on.

Duration: the length of a note

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 BBC micro is concerned, this will not normally be below 1/20th of a second. This is the minimum time you are able to allot a note in the SOUND command, although the step intervals in the ENVELOPE command increase in multiples of 1/100ths of a second.

From a psychological point of view, it is interesting to note the difference in time perception between individuals. Time seems to pass more quickly or slowly according to the events surrounding the individual. A boring after-dinner speaker may think that he has had the floor for fifteen minutes when he has been talking for half an hour: his listeners may think that three quarters of an hour have passed. 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, as timing sense is not normally dependent upon the ear.

Having a sense of timing plays a great part in the creative production of music. Consider, the only attributes of music a piano player has control over are volume and time. The timbre and pitch are determined by the instrument and composer. A performance is judged, however subconsciously, upon accent, rhythm and phrasing - and of such things are great musicians made.

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 and inaccurate, slow and accurate or slow and inaccurate, all in varying degress. 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 timings can be measured, but are well beyond the scope of this book. However, we can arrange a simple motility test which will be of use not only to musicians but to anyone wanting to develop quick reactions. 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.

```
   10 REM PROGRAM 1.2
   20 REM Motility Tester
   30
   40 ON ERROR GOTO 290
   50
   60 ENVELOPE1,11,16,4,8,2,1,1,100,0,0,
-100,100,100
   70 REM Turn off Auto Repeat
   80 *FX11,0
   90
  100 REPEAT
  110 Score=0
  120 CLS
  130 PRINTTAB(4,6)"Tap the RETURN key r
epeatedly"'"   as quickly as possible an
d with"'"      the minimum of movement."
  140 Begin=GET
  150 TIME=0
  160
  170 REPEAT
  180 Tap=INKEY(0):IF Tap=13 Score=Score
+1
  190 UNTIL TIME>=500
  200
  210 PRINTTAB(16,10)"STOP"
  220 SOUND1,1,100,20
  230 PRINTTAB(6,12)"Your MOTILITY ratin
g is"'Score/5;" taps per second"
  240 PRINTTAB(8,15)"Another try (Y/N)?"
  250 REPEAT:Key=GET AND &DF:UNTIL Key=8
9 OR Key=78
  260 UNTIL Key=78
  270
  280 REM Turn On Auto Repeat
  290 *FX12,0
  300 END
```

Program notes

The important part of the program lies between lines 170 and 190 which increment the variable, Score, when the user presses RETURN. The rest of the program is well REMed. For the curious and the impatient, the envelope at line 60 is examined in detail in Chapter 6.

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 the one 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 'tarn-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 BBC micro'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 alter line 260 to:

```
260 DRAWTime,Amp*SIN(RAD(Freq*Time))+Amp*1/2*SIN
(RAD(Freq*2*Time))+Amp*1/3*SIN(RAD(Freq*3*Time))+
Amp*1/4*SIN(RAD(Freq*4*Time))+Amp*1/5*SIN
(RAD(Freq*5*Time))
```

This adds the second, third, fourth and fifth harmonics to the fundamental, or main, tone, which is why it's so long. The fundamental is usually the strongest, that is loudest, frequency and gives the note its pitch. If you run the program, you will see that it produces a waveform like a sawtooth, from which it gets its name. If you add more harmonics in the same proportion, you will iron out the bumps and produce a better-looking sawtooth.

Alter line 960 again.

```
260 DRAWTime,Amp*SIN(RAD(Freq*Time))+Amp*1/3*SIN
(RAD(Freq*3*Time))+Amp*1/5*SIN(RAD(Freq*5*Time))+
Amp*1/7*SIN(RAD(Freq*7*Time))
```

This draws a square waveform, similar to the one produced by the BBC micro's sound chip. This time we are adding the odd harmonics and if you add more you will get a squarer wave.

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

As far as the BBC micro is concerned, we have no control over the waveform but, by clever use of the SOUND and ENVELOPE commands, we can trick the ear into thinking that what it hears is something other than a dressed-up distorted square wave. This is because the ear tends to take more notice of the envelope of a sound than the actual timbre. There are limits, however: 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, not necessarily computers with 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 in learning 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 at explaining 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. However, the desirability of having a reference section built in to the book, and the necessity to lay down at least a few rules to aid those with little prior musical knowledge, prompted this section.

As 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 another 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 masses of black dots on a page full of lines. They all look the same. If your 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 only need to read about the ideas and principles behind the dots and lines and refer back to this chapter when necessary: it is intended to be a potted reference section rather than an intensive teaching course. If you want to go further and study music, there are dozens of good books available. As with most things, you will find that

repeated study leads to a natural memorisation.

We all know what music looks like, even if we can't read it, and you may think that there must be an easier way to represent the ideas behind the notes. 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. You will find musical ideas much easier to follow and it will increase your appreciation of all kinds of musical events. You will also be able to convert written music to program form.

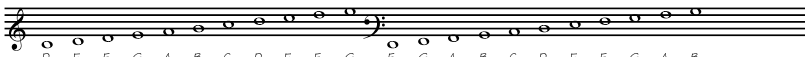
Music is written the way it is because of convention. It is simply the way it developed over the years and the advantage of the system, some say the only advantage, is that it is recognised world-wide. If you have no intention at all of communicating your ideas to others through anything other than a finished performance, there is nothing to prevent you developing your own musical system. There are, in fact, several alternative systems of music notation in use, all supposedly easier to learn: these have not yet achieved widespread use and are therefore limited in their range of effective communication, which is what music is all about.

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 at how pitch is represented first.

The pitch of a note

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

Figure 2.1

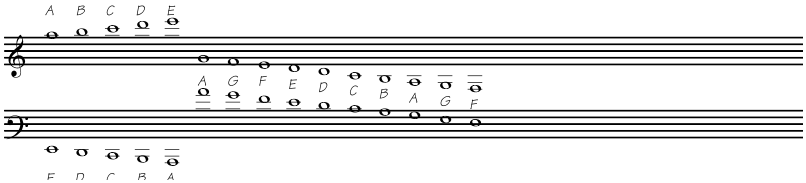


Notes can be placed above and below the fines to extend the range. These notes are written on and between short fines called leger lines which are really just an extension of the stave, as shown in Figure 2.2. The stave could consist of a set of ten or more fines but that would be very confusing and difficult to read. The number of leger fines can be extended as far as you wish but, again, too many make the music difficult to read.

To increase the range of notes still further and maintain readability, 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 here, are the treble or G clef and the bass or F clef, as shown in Figures 2.1 and 2.2. The treble clef loops about

the fine which represents G, and the bass clef has two dots which sit either side of the F line.

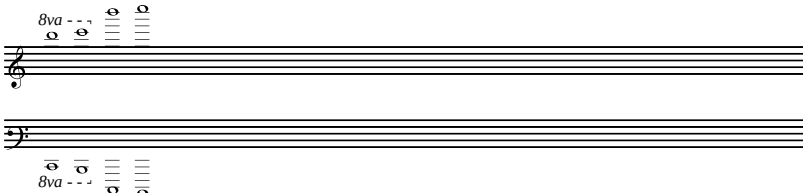
Figure 2.2



You will see that, as notes on leger fines 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



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 first program example in Chapter 1 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 Pitch
30 SOUND1 ,-15,Pitch,10
```

```
40 NEXT Scale
50 END
60 DATA 53,61,69,73,81,89,97 ,101
```

It sounds complete and is more musically satisfying than a sequence of semitones. The notes are read from a DATA statement and you will notice the progression of the intervals: 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 numbers required by the SOUND command to play a particular pitch. The numbers used by the BBC micro are completely arbitrary and bear no relation to the actual frequency of the notes.

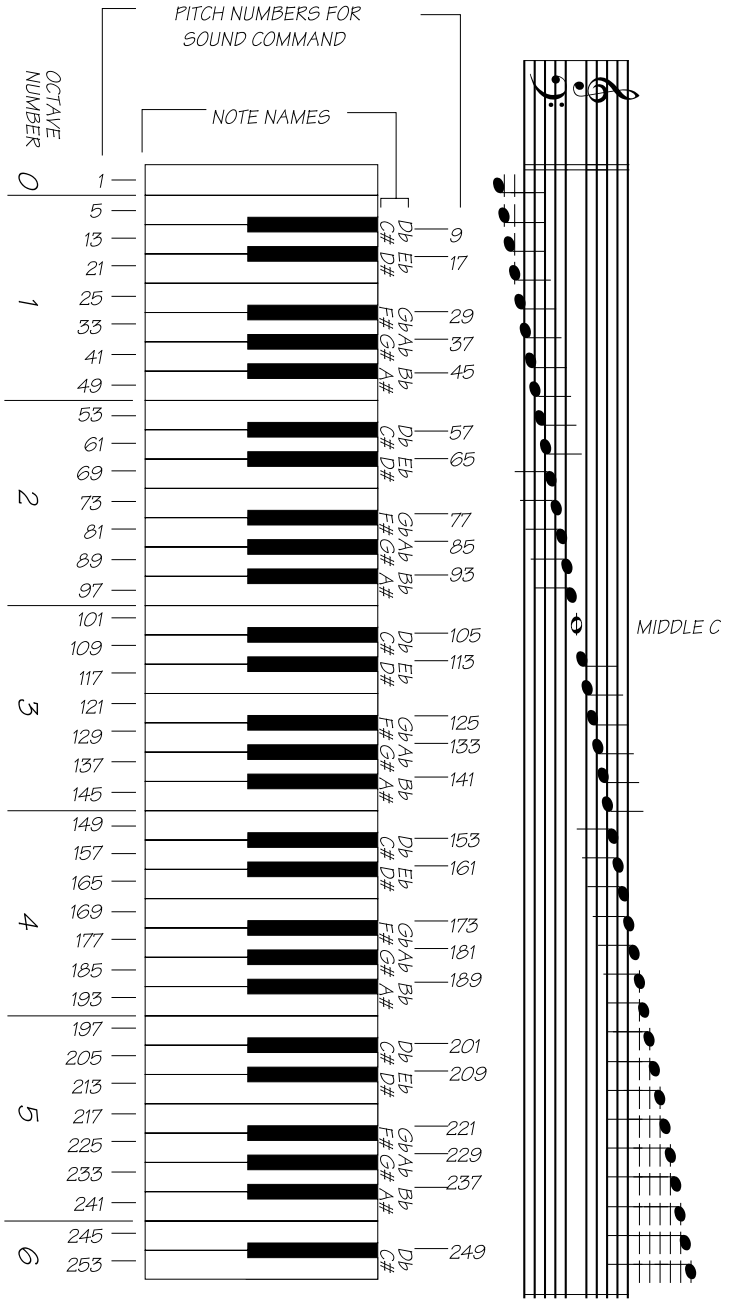
You will see that the notes on the stave have been transposed an octave, so that middle C on the keyboard appears an octave lower than middle C on the stave. The pitches still maintain their relation to each other and we will discuss why we have done this in the next chapter. We will use the octave numbers in Figure 2.4 in our musical notation when entering tunes and they will be of help in transposing music up or down an octave.

Scales

Most musicians can remember practising scales when they began to learn their instrument - most good musicians still practise - but the study of scales seems to carry with it a sense of boredom. Fortunately, we do not have to practise five finger exercises all day, and we really do not need to wade through mounds of musical theory in order to understand scales.

An important property of music, not always obvious at first sight, is the fact 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 will sound an octave higher. This shows what octaves sound like and plays the six C notes available from the sound chip:

Figure 2.4


PITCH NUMBERS FOR SOUND COMMAND
NOTE NAMES
OCTAVE NUMBER
0
1
2
3
4
5
6
1
5
13
21
25
33
41
49
53
61
69
73
81
89
97
101
109
117
121
129
137
145
149
157
165
169
177
185
193
197
205
213
217
225
233
241
245
253
Db C#
Eb D#
Gb F#
Ab G#
Bb A#
9
17
29
37
45
57
65
77
85
93
105
113
125
133
141
153
161
173
181
189
201
209
221
229
237
249
MIDDLE C

```
10 FOR Pitch=5 TO 245 STEP 48
20 SOUND1,-15,Pitch,10
30 NEXT Pitch
```

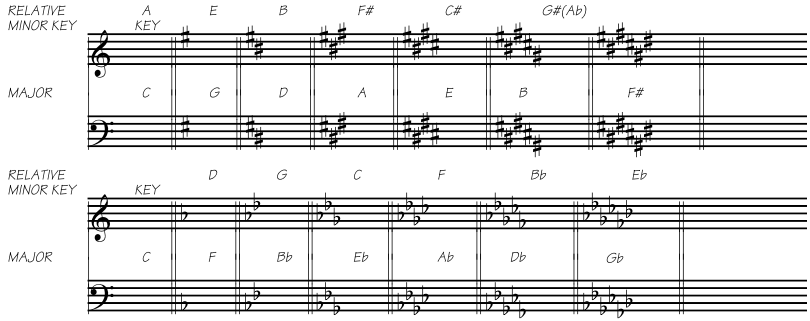
From the User Guide, page 180, we can see that the intervals in a scale, moving upwards, have the relation: tone, tone, semitone, tone, tone, tone, semitone. This means that we can play any scale at all by selecting a start note and adding those intervals to it. The next program allows you to do exactly that by altering the variable, Note, in line 10. A jump of a semitone is represented on the BBC micro by an increase of four in the pitch parameter of the SOUND statement and an increase in eight gives a jump of a tone.

```
10 Note=53
20 FOR Pitch=1 TO 8
30 READ Interval
40 Note=Note+Interval
50 SOUND1,- 15,Note,10
60 NEXT Pitch
70 END
80 DATA 0,8,8,4,8,8,8,4
```

If you alter the variable, Note, to any value other than 53 (C), and calculate the notes resulting from the addition of the intervals, you will realise that every other scale contains at least one black piano key. If you study the piano keyboard, you will realise that this is a consequence of its construction. This means that 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. No sharps or flats indicates that the piece is in the key of C.

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#. 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.



Minor scales

The notes on the keyboard can be arranged into scales other than major scales. 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. 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 that the actual scales are played.

An upward harmonic 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 this: tone, semitone, tone, tone, tone, tone, semitone. When moving downwards the sequence is: tone, tone, semitone, tone, tone, semitone, tone. You really need not be too concerned with this if you are just learning music. You only need to know that it exists and to know that the notes used in a composition can include any combination of any of the above scales.

We will stop before we get too enmeshed in the subject. Other books will supply more detailed explanations; my purpose here is only to make you aware of some of the basics.

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. In Chapter 12, Program 12.3 prints key signatures along with their relative major and minor kevs.

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 fight, 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 Ab is the same note as G# because a flattened A produces the same pitch as a sharpened G. Likewise, Eb is the same note 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. Conversely 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 by 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 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 musical notation is shown in Figure 2.6 along with their British and American names. The American names are easier to understand immediately and seem to be attracting numerous converts from the British system.

Figure 2.6

NOTATON	ENGLISH NAME	AMERICAN NAME	DURATION VALUE
	SEMIBREVE	WHOLE NOTE	32
	DOTTED MINIM	DOTTED HALF NOTE	24
	MINIM	HALF NOTE	16
	DOTTED CROTCHET	DOTTED QUARTER NOTE	12
	CROTCHET	QUARTER NOTE	8
	DOTTED QUAVER	DOTTED EIGHTH NOTE	6
	QUAVER	EIGHTH NOTE	4
	SEMIQUAVER	SIXTEENTH NOTE	2
	DEMI-SEMIQUAVER	THIRTYSECOND NOTE	1

The duration value shows how long each note sounds in relation to any other. 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 of the music and, although fast pieces will often contain semiquavers and demi-semiquavers, you cannot absolutely determine the speed of a piece by just looking at the notation.

Rests play an important part in music too, and rest values are shown in Figure 2.7. They go by the same name as their note equivalent with 'rest' tagged on the end, eg quaver rest. These can be increased in length by one half by the addition of a dot but it is more usual to see a rest of the equivalent half value placed after the other.

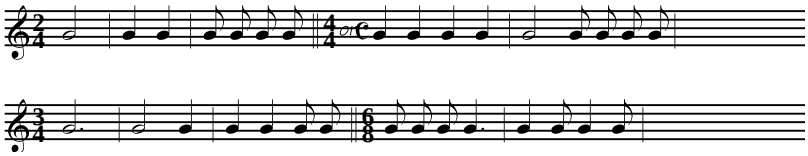
Figure 2.7

NOTATION	DURATION VALUE
	32
	16
	8
	4
	2
	1

Beats in the bar

The time signature of a piece of music is indicated at the beginning of the stave by two numbers, 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 two beats to the bar, each made up of a quarter note or crotchet. 3/4 is three beats to the bar, each a crotchet, and is the time signature in which most waltzes are written. 4/4 is sometimes written just as a large C and referred to as Common Time, and is by far the most common time signature of all encompassing quicksteps, foxtrots, rock 'n' roll, ballads and most classical music. The upper figure indicates the pulse or rhythm which runs through the music.

Figure 2.8



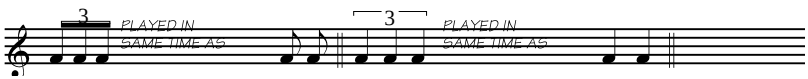
Time signatures can be altered at any point in the music and, indeed, can consist of any combination of notes the composer wishes to use. A selection of time signatures is shown in Figure 2.8 along with various note values which could be used to fill a bar. In practice, you will rarely come.: across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

Triplets, ties, slurs and staccato

These are aspects of music you may well see and they are worth explaining.

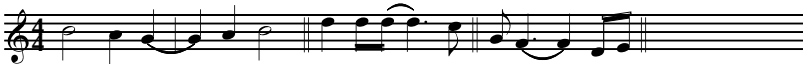
A triplet is a grouping of three notes as shown in Figure 2.9 and they are played in the same time as two notes of the same value. If you want to try playing the rhythm, tap two with your left hand and six with your right. That's easy. Now try tapping four with your left hand and six with your fight.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

Figure 2.9



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, is 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 it can 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 fines which seem to join notes of different pitches, these are slurs and are used to indicated 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 (as opposed to tones sounding consecutively, which would be called a melody).

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 sounding 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 that we will not be able to produce very complicated harmonies on the BBC micro.

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.

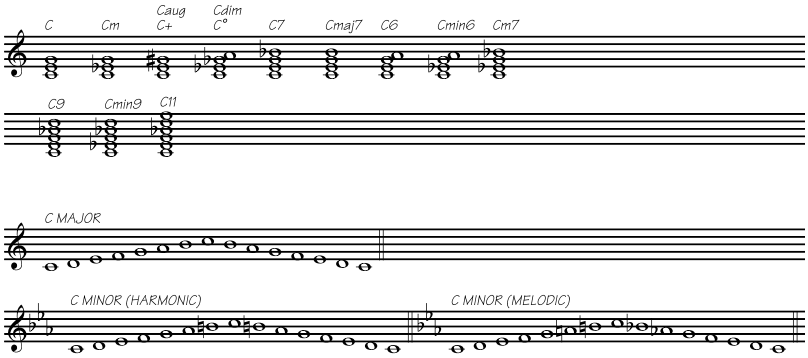
To construct a C major chord, we add an interval of a third and an interval of a fifth to the C note. These intervals are reckoned in note names, counted from the root note and in the key of the root note. To arrive at an interval of a third in the key of C, begin on C, count that as one and move up 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, ie 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. That .is exactly how they are constructed, eg. by flattening the fifth or by adding a ninth. Some examples are given in Figure 2.11.

Figure 2.11



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:

```
   10 REM PROGRAM 2.1
   20 REM Chord Sound Demonstration
   30
   40 CLS
   50 PRINT'''
   60
   70 FOR Chord=1 TO 4
   80 READ Chord$,Notes$,Pitch1,Pitch2
   90 PRINT"C ";Chord$;"=C + "Notes$'
  100 SOUND&201,-12,53,60
  110 SOUND&202,-12,Pitch1,60
  120 SOUND&203,-12,Pitch2,60
  130 PROCDelay(300)
  140 NEXT Chord
  150
  160 REM Chords with 4 Notes
  170 FOR Arp=1 TO 5
  180 READ Chord$,Notes$
  190 PRINT"C ";Chord$;" (Arpeggio)=C +
"Notes$'
```

```
  200 FOR Note=1 TO 17
  210 READ Pitch
  220 SOUND1,-15,Pitch,3
  230 NEXT Note
  240 PROCDelay(200)
  250 NEXT Arp
  260 END
  270
  280 DEF PROCDelay(Time)
  290 TIME=0:REPEAT UNTIL TIME>Time
  300 ENDPROC
  310
  320 DATA Major,E + G,69,81
  330 DATA Minor,Eb + G,65,81
  340 DATA Augmented,E + G#,69,85
  350 DATA Suspended 4th,F + G,73,81
  360 DATA Diminished,Eb + Gb + A,53,65,
77,89,101,113,125,137,149,137,125,113,10
1,89,77,65,53
  370 DATA 7th,E + G + Bb,53,69,81,93,10
1,117,129,141,149,141,129,117,101,93,81,
69,53
  380 DATA Major 7th,E + G + B,53,69,81,
97,101,117,129,145,149,145,129,117,101,9
7,81,69,53
  390 DATA Major Sixth,E + G + A,53,69,8
1,89,101,117,129,137,149,137,129,117,101
,89,81,69,53
  400 DATA Minor Sixth,Eb + G + A,53,65,
81,89,101,113,129,137,149,137,129,113,10
1,89,81,65,53
```

As we cannot sound more than three channels together (not including the noise channel), 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 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 BBC micro.

CHAPTER 3
The BBC Micro and Sound and Music

As we have seen, the nature of sound is quite an intricate and complex thing. Synthesisers do exist which can recreate and duplicate instrumental and natural sounds perfectly, but these cost several thousands of pounds. All of them, as you would expect, are computer based, but produce their sounds in a slightly different way to the BBC sound chip. Nevertheless, the BBC micro offers exceptional control over its sound system and we shall see in this chapter just exactly what can be done with it.

The waveform and timbre of the sound generator's output is fixed and set: there is nothing we can do to alter it under normal conditions. We can, however, alter the other three attributes of sound, namely pitch, volume and duration, and it is by controlling these parameters that we create our effects.

Pitch

The range of the sound generator extends over five octaves as we saw in the last chapter. We can not merely program the normal notes of the western scale but we can, should we feel inclined, experiment with quarter tone scales and other tonal systems.

The western scale divides the octave into 12 equal pitches which we call semitones. Experiments have been carried out on other octave divisions and music has been written for scales consisting of 13, 14, 15, l6 parts, etc. Such tunings, which divide an octave into other than 12 parts, are known as 'microtonal'. If handled correctly, they can produce fascinating results and provide a relatively undeveloped area of musical research for the experimenter.

Such tunings need even greater control over the sound source than we have with the BBC micro, but you can experiment with 16 and 24 part tunings and even devise your own scales. I'll leave these ideas with the more capable adventurers.

The subdivision of a semitone into four enables us to program portamento effects as we have already seen. This is not possible on all micros. It also allows us to 'bend' notes in a way similar to a guitar player, as this program demonstrates.

```
10 REM Hawaiian Guitar
20 ENVELOPE1,132,0,0,0,0,0,0,126,-10,0,-4,126,100
30 ENVELOPE2,132,-8,0,1,1,0,8,126,-10,0,-4,126,100
40 FOR Note=1 TO 8
50 READ Env,Pitch,Dur
60 SOUND 1,Env,Pitch,Dur
70 NEXT Note
80 END
90 DATA1,53 ,5,1,41,10,1,53,10,2,73,15
100 DATA1,41,5,1,33,10, 1,53,10,2,69,20
```

The principle involved is simply one of switching envelopes: when you consider that you can create 16 separate envelopes, you begin to see the versatility of this idea. If your imagination really runs riot you can create even more by redefining envelopes while a program is running. For most purposes, 16 envelopes will probably be sufficient.

In previous chapters we have mentioned envelopes only in relation to volume and loudness contours but, as you can see and hear from the program, the ENVELOPE command can also alter the pitch of a note. This principle will be familiar to most synthesiser owners and players, who are used to controlling pitch and timbre with the envelope generator. The ENVELOPE command performs similar functions but it doesn't operate in quite the same way.

Fitting the notes to the music

The lowest note produced by the sound generator is the B 13 semitones below middle C. The highest note is the D just over four octaves above middle C. This arrangement is a little one-sided, catering more for the piccolo player than the tuba player, and, in any event, most music tends to be written around the centre of the musical spectrum. A lot of music has a wide range, some even more than the five octaves the sound chip can produce, and a lot will have a good balance of high and low notes. Before programming such tunes, they will have to be arranged or transposed.

You will find the arrangement in Figure 2.4 more convenient when tackling such programs, as it provides an instant transposition which tends towards the range of notes you are most likely to find. If a piece of music contains notes outside this range, or which are predominantly high or low, it is an easy matter to transpose the notes another octave. For this purpose, if you wish, you may photocopy the diagram. If you cut between the keyboard and the stave, you can position the keyboard, which_ covers the range of the sound chip, beneath the stave in such a Chapter 3 The BBC Micro and Sound and Music chip you want to use. It is always a good idea to check the range of notes in the music before commencing programming.

There is, however, an easier and more convenient way. If a tune is

entered exactly in accord with Figure 2.4 and you want to play it an octave lower, you can deduct 48 from each pitch value, either during calculation of the pitch or when the pitch is sent to the SOUND command. Most of the programs in this book use a variable called Key, which can be set at the start of the program to make such alterations.

This is such an easy and simple way around the problem it is almost like cheating, but the facility is there for us to use, and use it we should.

Volume and duration

For the purpose of programming, these can be considered together. This is the real domain of the ENVELOPE command and, for synthesiser users, the ENVELOPE control of the ADSR functions should be easy to grasp.

ADSR stands for Attack, Decay, Sustain and Release and refers to the change in volume of a note during its production. The ADSR section of the ENVELOPE command is very comprehensive and allows us to imitate the volume characteristics of many musical instruments.

The ENVELOPE command is at the heart of most of the effects we create on the BBC micro and deserves a section to itself. In Chapter 5, we will investigate the natural envelopes produced by various instruments and look at ways of making easy the creation of the envelope parameters. We will also look at ways of relating the numbers to the sounds and effects they produce.

Improving the sound output

Early models of the BBC micro emit a continuous buzz from the speaker which changes pitch as a program is being executed. Acorn have said that this is due to the sound being generated in a digital fashion, but it seems more likely that it is caused by interference reaching the audio circuits. Whatever the reason, it can be annoying.

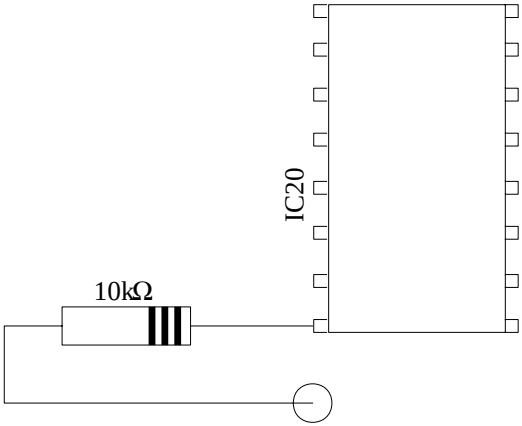
Acorn recommend a modification which involves connecting a }OK resistor between pin 8 on IC20 and ground, which is a small hole just below pin 8. IC20 can be found on the main board near the front of the computer underneath the keyboard. Figure 3.1 illustrates the modification.

This should only be attempted by someone competent in electronics, and it will probably be taken as invalidating your guarantee. Your dealer will be able to fit it for you, probably without invalidating your guarantee, at suitable cost.

An alternative solution is to wire the resistor between pins 15 and 16 an the 1 MHz expansion bus The easiest way to do this is to solder it to a socket and fit the socket to the plug under the computer. This will not invalidate your guarantee. You need an IDC (Insulation Displacement Connector) Speedblock type, female header socket 2*17 pin.

The modification is not a 100 per cent cure, but it is certainly worth performing.

Figure 3.1



Adding an external speaker

Although the two-inch speaker produces a better sound than that of most micros, it can seem quiet at times and, because it is a small speaker, some of the range and depth of the sound quality will be lost.

As a first step, there is a small preset volume control close to the loudspeaker connector under the keyboard and this can often be turned up a little. This will help in situations where it is not possible to connect another speaker but, in order to appreciate the full range of sounds available from the BBC micro, we really do need to feed the output to a larger speaker.

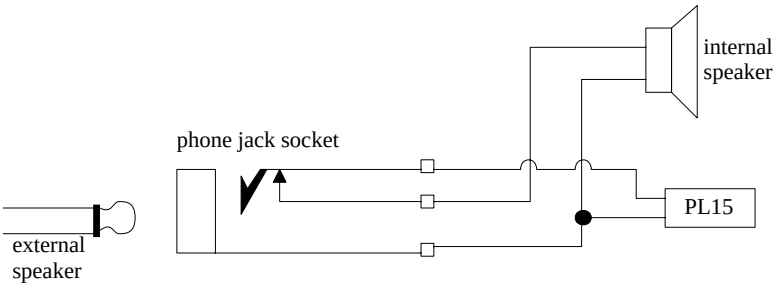
The audio amplifier in the BBC micro is an LM386 and it is quite capable of putting out half a watt through an 8-ohm speaker. The following modification, again, will probably be taken as invalidating your guarantee and should be carried out with care. The procedure is a very simple and straightforward one, but should not be attempted by anyone who cannot handle a soldering iron.

The simplest method is to disconnect the wires running to the internal speaker and connect them to an external speaker. If we are going to delve inside the machine, it makes more sense to adapt the system so that we can select either the internal or the external speaker. We can do this by using a socket which disconnects the internal speaker when a plug is inserted (and

connects it when no plug is inserted). This socket can be mounted in the back of the computer where the reset button used to be. The reset function was replaced by the BREAK key but on early models the hole will still be there. Newer models have no reset hole: you can either use the econet opening or else put a hole in the case or rear panel at another point. Some owners have mounted the socket hole in the case at another point. Some owners have mounted the socket on the side of the case beside the speaker. Wherever you mount the plug, take special care or get a qualified engineer to do the job.

The speaker is connected to the main board with two wires which run into plug PL15. You need to remove the case and the keyboard PCB to get to it; the plug can be removed, the adaptions made and the plug then refitted. The connections required are shown in Figure 3.2. The wires will normally be red and black and, if the external output seems rather quiet, reverse the leads.

Figure 3.2



This is certainly a modification worth doing. The increase in volume and quality will more than repay the small effort required to perform it.

Another solution

There is yet another alternative, which consists of routing the sound through the UHF modulator into your television set. Unless you are an accomplished electronics constructor, it will be simpler and safer to buy a ready-made unit from a manufacturer. They cost in the region of £10 and some can be fitted without soldering. The sound is undoubtedly an improvement, but is subject to the constraints of the modulator and TV set.

If you have a monitor, or use a TV set which has been adapted for RGE input, then this will not work. Many monitors and adapted TVs are fitted with a separate audio input and, if you are considering the purchase of such a unit, it may be a good idea to check its sound potential. It is, after all, a major part of your computer.

Further improvements and considerations

Hopefully, the newer models of the BBC micro will no longer have the speaker buzz, although it is unlikely that they will include an extension speaker socket.

Dirty mains and mains fluctuations are problems which can affect the whole computer. A severe glitch in the mains power supply can cause the computer to crash completely. The severity of such fluctuations varies with location and time of use, etc. The BBC micro is very tolerant of such things and it is extremely unlikely to succumb and crash completely but fluctuations can produce a hiss from the speaker. If this worries you enough you can consider the purchase of a filter plug which 'cleans up the mains'. These retail at around £15.

The command:

```
*FX210,1
```

will turn off the sound generator output and:

```
*FX210,0
```

will turn it on again. Unfortunately, this does not turn off the buzz.

Talking music

Standard music notation allows us to communicate our musical ideas to anyone who speaks the same language. As this is known and accepted world-wide, we rarely have any problems apart from matters of artistic interpretation. The micro has not yet been schooled in music which leaves us to devise our own methods of communication. This can be quite an exciting and challenging task but, as there are no industry standards, our methods are only likely to be of use to ourselves. Such considerations need not worry us unduly so we will consider some ways of talking music to the BBC micro.

An interesting place to start is to see how other micro manufacturers have approached the problem. Some require figures to be POKEd into certain memory locations. Others use ideas similar to those used in the BBC micro with variations of the SOUND command and some even allow the user to specify a particular note by name and octave number. Not all can synchronize four channels of sound; some of those which can, require the use of machine code.

The lesson to be learned here is that no personal micro manufacturer has yet devised a good BASIC music programming language. BASIC is

just not a very suitable language for the programming of music. The more expensive, dedicated music micro manufacturers tend to develop their own MCL (Music Composition Language).

So where does that leave us? Really, we're free to design a system to suit our own ideas, unhampered by precedence. Needs may vary and we can design different systems to suit different applications. Throughout the book, though, we will normally assume that we want to enter notes from conventional notation and play them through the computer. There are several methods of turning the notes into numbers.

The numbers method

This is the method used by the SOUND command: it is simply a list of numbers which correspond to a particular pitch, eg 53=middle C. This is fine, but if you want to enter a lot of notes it is easy to lose your place; and if you subsequently want to edit the tune, you have few clues as to what sound a particular sequence of numbers makes. Computers like numbers, but most humans prefer something a little less abstract.

It is worth bearing in mind that, if you only require a fanfare or other short tune, this is the method to choose. A few dozen notes will not usually cause severe problems in entering or editing and this is the method used in the previous examples in this book.

The other methods

These 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 ways in which we initially code the notes. Preference will vary from person to person.

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', A'', D#''', etc or D'''#, G''#. Some systems place the lines horizontally, directly over the note, some use roman numerals to indicate the octave and some count in semitones so that C4 would be followed by C#5, D6, D#7, etc.

I think the easiest to understand, to enter and to edit, and which produces the least number of mistakes en route, is the octave number version. You may disagree, so feel free to adapt the system to your own needs. Although the principle behind the octave number systems are the same, they may vary according to which octave the numbers represent.

On page 181 of the User Guide is an octave number system which puts middle C, pitch value 53, in octave 3. The only note in octave 1 is B - not counting the A# for reasons we shall discuss later - which seems like a

waste of an octave. Other numbering systems count middle C as C4 and work up and down from there, some use a superscript and subscript system which designates middle C simply as C, the octaves above C as C1, C2, etc and the octaves below as C1, C2 etc. The system described below can be adapted to any other notation system you feel more comfortable with.

Figure 3.3

	OCTAVE NUMBER						
	0	1	2	3	4	5	6
C		5	53	101	149	197	245
C#		9	57	105	153	201	249
D		13	61	109	157	205	253
D#		17	65	113	161	209	
E		21	69	117	165	213	
F		25	73	121	169	217	
F#		29	77	125	173	221	
G		33	81	129	177	225	
G#		37	85	133	181	229	
A		41	89	137	185	233	
A#		45	93	141	189	237	
B	1	49	97	145	193	241	

Note to Number Conversion Program

This is based upon the notation illustrated in Figure 3.3 and Figure 2.4. Octave 1 begins with the lowest available C which has a pitch value of 5. It is usually musically convenient to begin an octave with a C. The lower tones of the sound chip are richer than the upper ones and are generally more pleasing: I tend to find myself working more with these lower tones, especially for 'orchestral' pieces. Using the method described above in relation to Figure 2.4 you can, of course, use any range you wish.

```
  10 REM PROGRAM 3.1
  20 REM Basic Program to Convert
  30 REM Note Names & Octave Numbers
  40 REM Into PITCH values
  50
  60 PROCSetup
  70 FOR Count=1 TO 16
  80 PROCChooseNote
  90 PROCPlayNote
 100 NEXT Count
```

```
  110 END
  120
  130 DEF PROCSetup
  140 Scale$="  C  C# D  D# E  F  F# G
G# A  A# B"
  150
  160 REM Key Sets Value for Transposing
  170 REM Key=1 Will Play as You Would
  180 REM Expect.  Key=5 Will Play
  190 REM 1 Semitone Up. Key=9 will
  200 REM Play 1 Tone Up etc.
  210
  220 Key=1
  230
  240 DIM NotesToChooseFrom$(15)
  250 FOR S%=1 TO 15
  260 READ Note$
  270 NotesToChooseFrom$(S%)=Note$
  280 NEXT S%
  290 ENDPROC
  300
  310 DATAG1,A0,B0,C2,D2,E2,F#2,G2,A2,B2
,C3,D3,E3,F#3,G3
  320
  330 DEF PROCChooseNote
  340 Note=RND(15)
  350 Note$=NotesToChooseFrom$(Note)
  360 ENDPROC
  370
  380 DEF PROCPlayNote
  390 PROCAnalyseNote
  400 PROCCalculatePitch
  410 PRINT Note$,Pitch,Octave
  420 SOUND1,-15,Pitch,5:SOUND1,0,0,0
  430 ENDPROC
  440
  450 DEF PROCAnalyseNote
  460 IF LEN(Note$)<2 OR LEN(Note$)>3 TH
EN PRINT"ERROR IN DATA ";Note$:STOP
  470 IF LEN(Note$)=2 THEN NoteName$=LEF
T$(Note$,1) ELSE NoteName$=LEFT$(Note$,2
```

```
)
  480 PositionInScale=INSTR(Scale$,NoteN
ame$)/3
  490 Octave=VAL(RIGHT$(Note$,1))
  500 ENDPROC
  510
  520 DEF PROCCalculatePitch
  530 Pitch=Key+PositionInScale*4+(Octav
e-1)*48
  540 IF Pitch<0 OR Pitch>255 THEN PRINT
"ERROR IN DATA ";Note$:STOP
  550 ENDPROC
```

Program notes

As a demonstration, the program plays a series of 16 notes chosen at random from the 15 notes in the DATA statement at line 310. The notes are allocated to the array, NotesToChooseFrom$.

Line 140 sets up Scale$ which contains the notes of the scale. The spaces are important. It is worth pointing out that the black notes are all written in 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 program can be adapted to cater for the input of both sharps and flats if this suits you better.

The use of the variable, Key, at fine 220 is optional but makes any transposition easy. As mentioned earlier, setting Key to -47 (ie 1-48) will play the tune an octave lower.

The note name is analysed at fine 450. As each note will be either two or three letters long, fine 460 checks for errors in the DATA.

Line 470 separates the note name from the octave number and 480 determines its position in the scale. The higher its position, the higher the pitch.

Line 490 determines the octave and converts it into a numeral, and 530 calculates the pitch.

Line 540 is there in case something has gone wrong.

The task of calculating the pitch has been divided into two procedures, PROCAnalyseNote and PROCCalculatePitch. These could be combined into one procedure or turned into a function so that FNPitch(Note$) would return the pitch value to be used in place of the pitch variable in line 420. This will be the optimum solution in many cases but, by splitting it, it allows us to calculate note names and octaves separately. We may want to do this if the computer is following a set of rules, for example during

composition.

The second SOUND command in line 420 produces a silence of one two-hundredths of a second which is used to separate the notes. This is produced by a duration value of 0 and is useful for separating notes in simple arrangements.

The octave numbering system is one lower than that in the User Guide. It seems to make more sense to call the lowest octave, and one Chapter 3 The BBC Micro and Sound and Music writing Cl, D1, etc to C0, D0 and it still allows a pitch value of 1 which is B0. This means we need to deduct 1 from the octave number to produce the correct pitch, which we do in line 530. As the computer is doing the calculating work, once we have the conversion program all we have to do is enter the DATA.

There is an error in the DATA statement in line 310 which shows how one of the error routines operates. The DATA also illustrates how the system caters for B0.

We will use this method of note conversion, and variations on it, throughout the book.

The lowest A#

The User Guide states that a pitch value of 0 produces an A note. This is not true as this program shows.

```
10 FOR Pitch=13 TO 1 STEP -4
20 SOUND1,-15,Pitch,8
30 NEXT Pitch
40 SOUND,-15,0,8
```

You will hear that the difference between pitch values I and 0 is not as great as that between pitch values separated by a factor of four. It produces the same interval as that between any two adjacent pitch values, ie a quarter of a semitone, which is what you would expect.

From the above, it would therefore appear more sensible to use a pitch value of 0 as the lowest note. This would also be more in keeping with the true pitches produced by the sound chip: Chapter 6 gives a detailed listing of its actual output frequencies. As the official pitch values and note names are fairly well known, it would probably only serve to confuse if we were to start using other pitch names and values. In practice this will make little difference and enough information is given throughout the book for you to make any adjustments necessary for your own purposes.

CHAPTER 4
The SOUND Command

It is when BASIC comes across a SOUND command that information is sent to the sound chip. The ENVELOPE command merely alters the volume and pitch characteristics of the note the SOUND command has been programmed to produce. In its simplest form, it must be followed by four parameters:

SOUND C,A,P ,D

These stand for the following and take the indicated values:

Channel:	0 to 3
Amplitude:	normally - 15 to 0 but also 1 to 16 when used with ENVELOPE
Pitch:	0 to 255
Duration:	0 to 255

Channel and its extensions

The sound chip contains four sound channels numbered 0 to 3 and this parameter selects which channel is to be used to produce the sound. Channel 0 is the noise channel and channels 1, 2 and 3 produce the square wave tones described previously. As channel 0 produces a noise as opposed to a tone, the pitch parameters have a different effect, which we will look at later. All four channels can be programmed to sound together to produce quite sophisticated harmonies and effects.

This first sound parameter, the channel number, can be extended to four figures like this:

SOUND &HSFC, A, P, D

The ampersand (&) means the following numbers are hexadecimal digits (see page 71 of the User Guide) but, as the values used in the SOUND command are all in the range 0 to 3, we do not need to perform any decimal to hex conversions and can forget about it for practical purposes. We must not forget, however, to include the ampersand whenever we use more than one figure in the first SOUND parameter; otherwise the figure

as a whole will be taken as a decimal number and will not produce the effect we want. The additional parameters and their ranges are as follows:

Hold:	0 or 1
Synchronization:	0 to 3
Flush:	0 or 1
Channel:	0 to 3

When we use the shortened form of the SOUND command as in:

SOUND 1,-15,53,20

the channel is specified - in this case I - and the other parameters default to 0. So, if we do not want to use them, we can ignore them.

Hold

Hold (or Continuation) is described in the User Guide on pages 187, 350 and 352. This is probably the least understood of the SOUND parameters - which isn't surprising because it is described inaccurately on pages 187 and 350 and all descriptions tell you what can be done with the command, not what the command actually does.

If H is 0, the default value, the SOUND command operates as normal. If H is set to 1, the amplitude and pitch parameters in the SOUND command are ignored, but the duration is obeyed. Normally this would just create a silence or a rest but, if this command follows a note on the same channel which is under envelope control, then the previous note may be extended. To understand this further we need to be aware of the ADSR functions of the ENVELOPE command.

If the amplitude A in a SOUND command is set to a value between 1 and 16, the volume of the sound is controlled by the envelope with that particular number. Without an envelope, each SOUND command will last for the duration set by its D parameter - no longer, no shorter. Once an envelope is used, D determines the value of the attack, decay and sustain periods only. If no other commands follow on the sarrye channel, it will be allowed to continue into the release phase of the envelope instead of terminating. Normally the next note in the sound queue will occur as soon as the SOUND command has completed its allotted duration and the release phase will not occur. The ENVELOPE command is described in detail in the next chapter.

A SOUND command with H set to 1 produces a 'dummy' note, as it is called in the User Guide, for the duration of the D parameter. This will allow the previous note to continue for this extra time and, if the previous

note goes into a release phase, it will not be automatically terminated. Setting H to I does not automatically allow the release phase to complete but simply allows it to continue. If there is nothing left in the release phase, there will be nothing to sound and there will be silence for the duration of the D parameter. Conversely, if the additional time given by the D value of the dummy note is too short, the release phase will not reach its end.

In effect, setting H to 1 in a SOUND statement creates a time gap equal to the D parameter through which a previous note on that channel can play - assuming that there is something there to play.

Misunderstandings may have arisen because the User Guide states, incorrectly, on page 187 that the amplitude, pitch and duration are ignored, leading one to believe that this command calculates the release time on the previous note and allows it to play out. The description on page 350 is ambiguous and other writers have followed the wrong description.

In practice, this command is not often used. Its main function is to allow the release section of a sound to occur as the User Guide says. When you are experimenting with the ENVELOPE command and the parameters which alter the pitch values, you will find a lot of good noises, sounds and effects occur during the release part of the ADSR phase. If you want the release phases to occur in a series of notes, the easiest way to do this is with the Hold option. This short program illustrates its effect:

```
   10 REM PROGRAM 4.1
   20 REM HOLD PARAMETER DEMONSTRATION
   30
   40 ENVELOPE1,4,0,0,0,0,0,0,126,-1,0,-
1,126,60
   50 TIME=0
   60
   70 SOUND1,1,53,40
   80 PROCTime
   90 SOUND1,1,69,20
  100 PROCTime
  110 END
  120
  130 DEF PROCTime
  140 REPEAT
  150 IF TIME/100=INT(TIME/100) PRINT TI
ME/100
  160 UNTIL TIME>800
  170 STOP
  180 ENDPROC
```

Program notes

Trying to time SOUND commands is difficult, because the commands are stored in a queue and a BASIC program simply moves through them - unless the buffer is full, in which case the BASIC program is held up. PROCTime begins after the commands have been issued and prints to the screen approximately every second.

When you run the program you will notice that the sound produced by line 70 lasts four seconds. Its D value is 40 which means that the attack, decay and sustain phases last two seconds, the other two seconds being the release phase. If you remove line 80 you will see that the new note produced by fine 90 occurs after two seconds, indicating that the release phase of the first note was cut short by the appearance of another note in the sound queue.

Enter another line 80 as follows:

```
80 SOUND&1001,0,0,40
```

The first note will now last for four seconds - its own two-second duration plus the extra two seconds afforded it by the dummy note in line 80. If you increase the duration parameter in line 80, you will hear the first note complete its release phase and then there will be silence, while the dummy note runs its time, before the note on line 90 sounds.

Synchronization

This allows two or more notes to sound at exactly the same time. If S is 0, the default value, notes are queued as usual and notes on each channel sound as soon as they reach the front of their respective queues. If S is 1, 2 or 3, then the note does not sound until there is a corresponding note or notes with the same S value on another channel or channels. If S is set to I the computer waits for one more note. If S equals 2 it waits for two more notes and if S equals 3 it waits for notes on all four channels.

The use of this command is quite well documented in the User Guide but it is not always so obvious how to use it. As a means of sounding notes at the same instant, it does not seem to be so different from the usual queuing method. Is this:

```
10 SOUND1,-15,73,30
20 SOUND2,-15,89,30
30 SOUND3,-15,101,30
```

so different from this:

```
10 SOUND&201,-15,73,30
```

```
20 SOUND&202,-15,89,30
30 SOUND&203,-15,101,30
```

The value of this command, however, lies in the fact that it allows us to execute other statements between sounds without throwing our sound out of synchronization, or 'sync' as it is often referred to. Insert this line in the two previous examples and observe the result:

```
15 FOR Delay=1 TO 400:NEXT Delay
```

This feature of the SOUND command can be used to ensure that programmed tunes play in sync and also to prevent individual channels wandering off due to the relative slowness of BASIC. Short sequences of notes may not need this to ensure synchronization. The whole topic is covered more fully in Chapter 9.

Flush

This, too, is well documented in the User Guide. When F is set to I and the SOUND statement in which it occurs is executed, it flushes the sound channel of any notes waiting in the sound queue and stops execution of whatever note may be sounding at that time. The sound channel with F set to 1 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. It does not have to wait until its present meanderings are completed. By using the F parameter, it flushes the relevant sound channels and plays something new. Explosions occur in the same way, exactly at the time an object is hit.

In a musical context, by issuing a command with F set to 1 we can stop notes sounding on a particular channel. The next program demonstrates the flushing facility.

```
 10 REM PROGRAM 4.2
 20 REM CHANNEL FLUSHING DEMONSTRATION
 30 SOUND3,-10,5,245
 40 SOUND1,-15,53,60:SOUND2,-12,5,60
 50 SOUND1,-15,81,60:SOUND2,-12,53,60
 60 SOUND1,-15,101,60:SOUND2,-12,53,60
 70 SOUND1,-15,117,5:SOUND2,-12,69,5
 80 SOUND1,-15,113,60:SOUND2,-12,65,60
 90 PRINT"PRESS ANY KEY TO STOP"
100 IF INKEY(400)<>-1 THEN SOUND&11,0,
```

```
0,0:SOUND&12,0,0,0:SOUND&13,0,0,0 ELSE P
RINT"TOO LATE"
  110 SOUND1,-15,149,2:SOUND1,-15,137,2
  120 SOUND1,-15,129,2:SOUND1,-15,137,2
  130 SOUND1,-15,145,2:SOUND1,-15,137,2
  140 SOUND1,-15,145,2:SOUND1,-15,149,4
```

Program notes

The INKEY delay in fine 100 gives you four seconds to hit the key. If you do so, it will flush all channels and proceed with the routine beginning at fine 110. If you do not press a key within the time limit, the first tune plays out and the second routine is queued in the normal way and will be heard after the first tune.

In practice, this program would be better written using DATA statements and the commands could even be synchronized, but the purpose is to illustrate the flushing procedure. You can experiment by flushing individual channels and listening to the effect.

Amplitude

This is well documented and generally well understood. Normal values vary from - 15 which is very loud (comparatively speaking) to 0 which is off. (-1 is quite quiet.)

If a positive value in the range 1 to 4 is substituted, this puts the SOUND command under envelope control, which permits far more sophisticated variations over volume. If you do not use the RS423 and cassette output buffers, ie the BPUT# command, then up to 16 envelopes can be used and A can take a value up to 16.

The volume of a sound is affected by many factors, as we saw in Chapter 1. If the amplitude of a waveform is doubled 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.

The study of loudness and volume is a science in itself. For our purposes we only need remember that, because of the vagaries of the sound chip, the loudspeaker (whichever one we are using) and its enclosure, and the properties of sound itself, some notes may overpower others.

Pitch

This can take values from 0 to 255 and selects tones in quarter of a semitone intervals. If we want to use the conventional western scale of music, we must work in increments of four. Figures 2.4 and 3.3 show how the values of P correspond to printed notes and the keyboard.

Channel 0, the noise channel, operates in a different way and the P parameter produces the following effects:

0 High frequency periodic noise.
1 Medium frequency periodic noise.
2 Low frequency periodic noise.
3 Periodic noise of a frequency determined by the pitch of channel 1.
4 High frequency white noise.
5 Medium frequency white noise.
6 Low frequency white noise.
7 White noise of a frequency determined by the pitch of channel 1.

Try the sounds to hear what they are like. The following demonstrates the effects of setting P to 3:

```
10 SOUNDO,-15,3,200
20 FOR Pitch=0 TO 200
30 SOUND1,0,Pitch,1
40 NEXT Pitch
```

Alter the P parameter in line 10 to 7. Experiment by putting channel 1 under envelope control.

The noise channel deserves a section to itself and is discussed in detail in Chapter 7.

Duration

This sets the length of time the note is to sound in twentieths of a second. If D is set to 255, the sound will continue indefinitely until stopped by pressing ESCAPE or by flushing the channel. With D set to 0, the note is given a duration of one two-hundredth of a second. This is very useful for separating notes of the same pitch. For example:

```
10 SOUND1,-15,53,20
20 SOUND1,-15,54,20
```

will sound as one note with a total duration of two seconds. If you insert:

```
15 SOUND1,0,0,0
```

this will cause a slight separation between the notes.

We can time the length of this duration by the following:

```
10 TIME=0
20 FOR X%=0 TO 500
30 NEXT
40 PRINT TIME/100
```

Run it a few tunes to get an average time for the completion of the loop. Insert line 25:

```
25 SOUNDl,0,0,0
```

This will hold up the loop for the length of the duration. If you subtract the first time from the second time and divide by 500, you will get a figure just over one two-hundredth of a second. The extra time is accounted for by the time, it takes the OS to sort out its instructions, and it is not very long.

If a tune is programmed in more than one part and this technique is used a lot, the tune may eventually run out of sync. There are other ways of separating notes and maintaining synchronization. One answer is to use envelope control.

Referring to Figure 2.6, if you assign the duration values there to the actual D parameter, you have a workable set of note durations which will be about right for tunes of moderate tempo. As tempos can vary widely you may find you have to increase or decrease these values but the relationship between note durations must be maintained. If you reduce the values to their lowest common denominator, you can add a 'tempo' variable for easy adjustment of the speed of the piece.

If an envelope has been selected, D determines the total of the attack, decay and sustain periods but not the release phase. This was mentioned in relation to the Hold parameter and is discussed more fully in the next chapter.

Out of range values

The User Guide gives the range of duration values as being between 0 and 255. On page 348 it says that setting D to -1 will make the note last indefinitely, and a duration of between 0 and 254 will give a note a duration of that number of twentieths of a second. The statement about 0 is not true as we have seen and, in fact, -1 has the same effect as 255. This is because the values are brought within the range the sound chip expects. If you try to go off the top of the range the values wrap around and you find yourself coming up from the bottom - and vice versa.

The reasons for this are a little technical and it is not necessary to know why this happens, but the curious may gain further insight from examination of the MOD and DIV commands in the User Guide.

All the SOUND command parameters are treated in a similar way and reasons need not concern us-so long as we keep our values within their allotted range. There is nothing to be gained by going outside the range, as the computer will simply reduce them to an acceptable value. Such instructions will not generate an error and the only time, apart from incorrect input of data, when we need to be aware of the limitations is if we are creating values through calculation. For example, trying to play a scale with pitches going beyond 255 will bring the pitch back down to 0 to start again from there.

All SOUND commands are expected in integers. If they are not given, the non-integer part is ignored, eg:

```
SOUND 1,-15,53,20
```

produces the same note as:

```
SOUND1,- 15,53.99999 ,20
```

This can be useful if we are calculating values.

Versatile though the SOUND command is on its own, the range of sounds it can produce can be expanded enormously by the use of the ENVELOPE command - which is what we look at in the next chapter.

CHAPTER 5
The ENVELOPE Command

The ENVELOPE command is arguably one of the most difficult commands to master in BBC BASIC. Part of the problem lies in the fact that it must be followed by 14 parameters and used in conjunction with the SOUND command. This alone gives us a myriad of possibilities to choose from and the chances of getting things wrong are considerable.

The advantages of knowing what to do when searching for an effect, as opposed to resorting to a trial and effort method, cannot be overemphasised - unless you have a lot of time on your hands; and when does time pass more quickly than when you're programming your computer?

Chapter 7 explores the trial and effort method and how to get the most out of it - with the minimum of effort. This chapter explores the systematic method, one you will find infinitely rewarding once you are able to think of a sound and know immediately how to produce it.

The ENVELOPE command has two separate functions. The first is to control the amplitude of the sound and the second is to modulate the pitch. When a SOUND command is controlled by an envelope, amplitude control is automatically passed to the envelope. In order to produce a sound, the envelope must be configured to do so. Control over pitch is optional and can safely be ignored when experimenting with the amplitude parameters. The Hawaiian Guitar program in Chapter 3 demonstrates pitch control.

The complete ENVELOPE command

Using the notation on page 245 of the User Guide, the ENVELOPE command is followed by 14 parameters and described as follows:

ENVELOPE N,T,PI1,PI2, PI3,PN1,PN2,PN3,AA,AD,AS,
AR,ALA,ALD

The parameter names could have been slightly better chosen but, as these are in common use and many people will be used to thinking in these terms, there is little point in adding further complexities to the situation by introducing new ones. I think of the parameters in these terms:

*N*umber of envelope
*T*ime of each step

*PI*tch 1
*PI*tch 2
*PI*tch 3

*P*itch *N*umber of steps *1*
*P*itch *N*umber of steps *2*
*P*itch Number of steps *3*

*A*mplitude change during *A*ttack
*A*mplitude change during *D*ecay
*A*mplitude change during *S*ustain
*A*mplitude change during *R*elease

*A*mplitude *L*evel for *A*ttack phase
*A*mplitude *L*evel for *D*ecay phase

They may help, or you may have your own mnemonics. The parameters, their ranges and functions are fisted in **Figure 5.1** for easy reference.

Exploration of the two aspects of envelope control will be much easier if they are considered separately and, if the six pitch parameters are set to 0, we can observe the effects of altering the amplitude section.

First, we will see how the loudness contour of a sound can be broken down into sections.

ADSR: the amplitude envelope

ADSR or Attack, Decay, Sustain and Release, has been mentioned in previous chapters in relation to the way in which the volume of a note varies during production. Although the ADSR principle 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 5.2.**

This example is obviously very coarse and long (in terms of time), but the principle behind the volume variations involved are exactly the same as those which occur when an instrument produces a note. The note envelope, however, will usually be over in one or two seconds, often less.

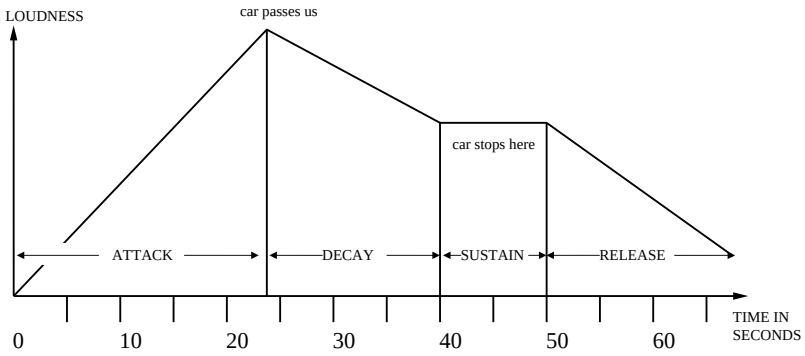
Figure 5.1

AMPLITUDE ENVELOPE PARAMETERS

PITCH ENVELOPE PARAMETERS

PARA-METER	RANGE	FUNCTION
N	1 to 16	Envelope Number
T		Length of each step in hundredths of a second.
	1 to 127	Pitch envelope auto repeats.
	129 to 255	Pitch envelope does not repeat. T assumes a value mod 128.
PI1	-128 to 127	Change of pitch per step in firs section.
PI2	-128 to 127	Change of pitch per step in second section.
PI3	-128 to 127	Change of pitch per step in third section.
PN1	0 to 255	Number of steps in first sectior
PN2	0 to 255	Number of steps in second section.
PN3	0 to 255	Number of steps in third section.
AA	-127 to 127	Change in amplitude per step during attack phase (heading towards ALA).
AD	-127 to 127	Change in amplitude per step during decay phase (heading towards ALD).
AS	-127 to 0	Change in amplitude per step during sustain phase (heading towards 0).
AR	-127 to 0	Change in amplitude per step during release phase (heading towards 0).
ALA	0 to 126	Target amplitude level AA is aiming for.
ALD	0 to 126	Target amplitude level AD is aiming for.

Figure 5.2



For convenience, the loudness contour is divided into phases called attack, decay, sustain and release. In practice, some sounds have more phases but details of this nature will be of more interest to synthesists and are generally beyond the scope of the BBC micro - although such effects can be obtained by using two envelopes in succession.

The attack phase

This is when the sound first begins. It 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. Most string instruments have a slow attack phase which might be around a quarter 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 in between.

If you have ever played a synthesiser, this 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, specific level. As its name implies, the sound usually decays or drops in volume

during this phase, but it can remain the same or even increase. Notes from musical instruments tend to reduce in volume although some will stay the same.

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 can't usually be stopped dead. A useful analogy, again, is the car which cannot 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 hundredth of a second or so to dampen down. This helps to explain why a backwards recording of ar} instrument sounds so strange - it dies away too rapidly. In fact, it doesn't die away, it's cut off. This is easily duplicated by electronic means and the ENVELOPE command makes it easy to produce on the BBC micro.

We have now touched upon the final stage of a sound - the release phase - and you will often find that the decay phase slides into the release phase. There is a difference, though, which will soon become apparent.

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 and, because the sustain phase is not a measurement of time and the attack and decay phases are, it sometimes causes confusion. It may help to think of it as the volume level the sound is at after the decay phase and it could more accurately be termed a 'volume level'.

The BBC micro adds a further complication by allowing the volume to reduce during this phase. This is not in keeping with the technical definition of sustain phase and a reduction in volume is not always possible even on synthesisers.

On the BBC micro, the sustain parameter is 0 or a negative number. The length of the phase is determined by the duration parameter, D, of the SOUND command and the note will continue for that length of time unless the sustain parameter is a negative number and the volume reaches 0 before the duration time is up. When given a negative number, the sustain phase behaves like a second decay phase with a target level of 0.

For synthesiser players, the sustain phase is the volume at which the note continues to sound until you take your finger off the key.

If these descriptions seem to be getting a little involved and complicated, it is because of the flexibility of the amplitude envelope. The next program and accompanying explanations allow you to see and hear what we are talking about, so don't give up.

The release phase

When the sound has gone through all the above phases, it reaches the stage where it will either continue forever or terminate. If it terminates, it can die away slowly or it can be cut off suddenly. 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 amplitude 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. Like a synthesiser, the ENVELOPE command on the BBC micro allows the note to continue indefinitely or to fade at a predetermined rate. If another note is waiting in the sound queue, the release phase will not occur and the new note will sound immediately after the sustain phase.

The complete ADSR envelope: putting them all together

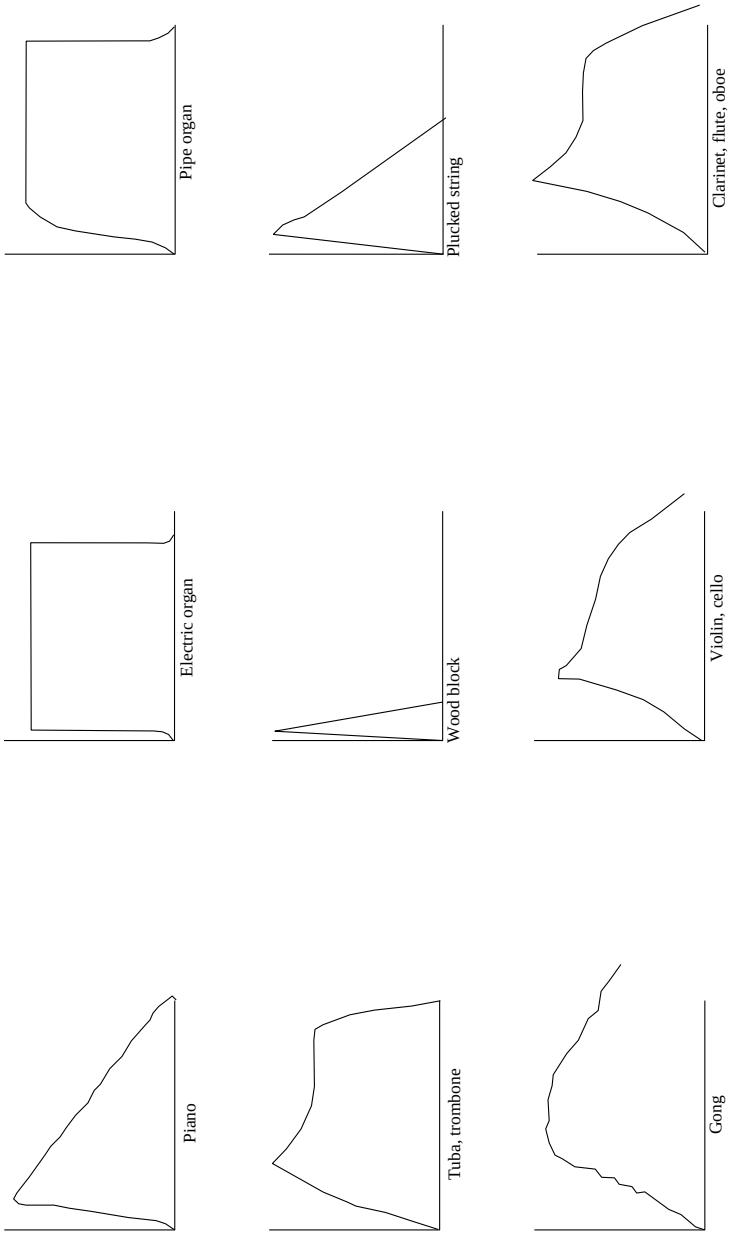
The loudness contour of most instruments and many other sounds can be duplicated quite accurately by controlling the ADSR phases of the envelope.

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 wood block 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 5.3**.

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. For a single note, the difference is not important but, if a note is followed by another one and its decay effect is produced by the release section of the envelope, it will be cut short.

You can program sounds without a decay phase or a sustain phase if you wish. Some synthesisers have a simplified envelope generator known as an AR generator which only allows the creation of an attack and a release phase.

Figure 5.3



We will now see how the ENVELOPE command tackles the problem of creating an ADSR envelope.

ADSR and the ENVELOPE command

The best way to explore the ENVELOPE command is to be able to hear and see what is happening as you read about it. The following program allows you to alter the ADSR parameters and see and hear the resulting envelope. The program fines increment by 10. Even if you do not enter all the REM statements, put the fine numbers in so that you can easily add the Pitch Graph Generator described later in the chapter. To create a blank fine, enter the fine number, hit the space bar and RETURN.

```
   10 REM PROGRAM 5.1
   20 REM ADSR Graph Generator
   30
   40 *TV255,1
   50 MODE1
   60 PROCSetUp
   70 PROCAxis
   80 Input$="":PROCReset:PROCPrintEnv
   90
  100 REPEAT
  110 *FX15,1
  120 Input$=GET$
  130 IF Input$=" " PROCSound
  140 IF Input$>="1" AND Input$<="8" PRO
CEnv
  150 IF Input$="C" PROCAxis
  160 UNTIL Input$="Q"
  170 END
  180
  190 DEF PROCSetUp
  200 REM Set Text Window
  210 VDU28,0,4,39,0
  220 REM Set Graphics Window
  230 VDU24,0;0;1279;860;
  240 REM Background=Yellow
  250 GCOL0,130:CLG
  260 REM Foreground=Black
  270 GCOL0,0
  280 REM Set COLOUR 1 to Flashing
  290 VDU19,1,9,0,0,0
  300
  310 REM Set Initial Parameters
```

```
  320 PI1=0:PI2=0:PI3=0
  330 PN1=0:PN2=0:PN3=0
  340 T=20:T1=T:Pitch=10:Pit1=Pitch
  350 AA=126:AD=-4:AS=-1:AR=-6
  360 ALA=126:ALD=80:Duration=80
  370 REM Marker for X Axis
  380 VDU23,224,128,128,128,128,128,128,
128,128
  390 ENDPROC
  400
  410 DEF PROCAxis
  420 YScale=6
  430 CLG
  440 VDU29,0;0;
  450 MOVE50,0:DRAW50,860
  460 MOVE0,50:DRAW1279,50
  470 VDU5
  480 FOR Mark%=0 TO 780 STEP YScale*10
  490 MOVE20,Mark%+YScale*10:PRINT"-"
  500 NEXT Mark%
  510 FOR Mark%=50 TO 1250 STEP 100
  520 MOVEMark%,50:PRINTCHR$224
  530 NEXT Mark%
  540 VDU 4
  550 REM Set Graphics Origin
  560 VDU29,50;50;
  570 ENDPROC
  580
  590 DEF PROCEnv
  600 PROCReset
  610 IF Input$="1" t=1
  620 IF Input$="2" aa=1
  630 IF Input$="3" ad=1
  640 IF Input$="4" as=1
  650 IF Input$="5" ar=1
  660 IF Input$="6" ala=1
  670 IF Input$="7" ald=1
  680 IF Input$="8" dur=1
  690 PROCPrintEnv
  700 PROCAlter
  710 PROCPrintEnv
```

```
  720 ENDPROC
  730
  740 DEF PROCReset
  750 b=3:t=3:aa=3:ad=3:as=3:ar=3
  760 ala=3:ald=3:dur=3
  770 ENDPROC
  780
  790 DEF PROCPrintEnv
  800 COLOURb:PRINTTAB(0,0)"ENV1,";:COLO
URt:PRINT;T;
  810 COLOURb:PRINT;",";PI1;",";PI2;",";
PI3;",";PN1;",";PN2;",";PN3;",";
  820 COLOURaa:PRINT;AA;:COLOURb:PRINT;"
,";:COLOURad:PRINT;AD;:COLOURb:PRINT;","
;
  830 COLOURas:PRINT;AS;:COLOURb:PRINT;"
,";:COLOURar:PRINT;AR;:COLOURb:PRINT",";
  840 COLOURala:PRINT;ALA;:COLOURb:PRINT
;",";:COLOURald:PRINT;ALD:COLOURb
  850 COLOURdur:PRINTTAB(14,1)SPC(4)TAB(
10,1)"Dur=";Duration:COLOURb
  860 ENDPROC
  870
  880 DEF PROCAlter
  890 INPUT NewVal$:PRINTTAB(0,2)SPC(6)
  900 IF NewVal$="" PROCReset:PROCPrintE
nv:ENDPROC
  910 NewVal=EVAL(NewVal$)
  920 PRINTTAB(30,0)SPC(20)
  930 ON EVAL(Input$) GOTO940,950,960,97
0,980,990,1000,1010
  940 T=NewVal:T1=T:ENDPROC
  950 AA=NewVal:ENDPROC
  960 AD=NewVal:ENDPROC
  970 AS=NewVal:ENDPROC
  980 AR=NewVal:ENDPROC
  990 ALA=NewVal:ENDPROC
 1000 ALD=NewVal:ENDPROC
 1010 Duration=NewVal:ENDPROC
 1020
 1030 DEF PROCSound
```

```
 1040 ENVELOPE1,T,PI1,PI2,PI3,PN1,PN2,PN
3,AA,AD,AS,AR,ALA,ALD
 1050 SOUND1,1,Pit1,Duration
 1060 REM Clear Last Timing Results
 1070 PRINTTAB(0,4)SPC(39);:PRINTTAB(0,4
);
 1080 IF T=0 T1=1
 1090 IF T=128MT1=129
 1100 Time=0:Amp=0
 1110 MOVE0,0
 1120 YScale=6
 1130 PROCAttack
 1140 PROCPrint("A",8,32)
 1150 IF OverTime GOTO 1210
 1160 PROCDecay
 1170 PROCPrint("D",8,32)
 1180 IF OverTime GOTO 1210
 1190 PROCSustain
 1200 PROCPrint("S",8,32)
 1210 PROCPrint("r",40,16)
 1220 PROCRelease
 1230 PROCPrint("R",0,0)
 1240 PRINT;"Secs";
 1250 ENDPROC
 1260
 1270 DEF PROCAttack
 1280 REPEAT
 1290 Amp=Amp+AA
 1300 IF Amp>ALA Amp=ALA
 1310 DRAW Time,Amp*YScale
 1320 Time=Time+T1 MOD128
 1330 PROCTimeCheck:IF OverTime GOTO1350
 1340 DRAW Time,Amp*YScale
 1350 UNTIL OverTime OR Amp=ALA
 1360 ENDPROC
 1370
 1380 DEF PROCDecay
 1390 REPEAT
 1400 Amp=Amp+AD
 1410 IF ALD<ALA:IF Amp<ALD Amp=ALD
 1420 IF ALD>ALA:IF Amp>ALD Amp=ALD
```

```
 1430 IF ALD=ALA Amp=ALD
 1440 DRAW Time,Amp*YScale
 1450 Time=Time+T1 MOD128
 1460 PROCTimeCheck:IF OverTime GOTO1480
 1470 DRAW Time,Amp*YScale
 1480 UNTIL OverTime OR Amp=ALD
 1490 ENDPROC
 1500
 1510 DEF PROCSustain
 1520 REPEAT
 1530 Amp=Amp+AS
 1540 IF Amp<0 Amp=0
 1550 DRAW Time,Amp*YScale
 1560 Time=Time+T1 MOD128
 1570 PROCTimeCheck:IF OverTime GOTO1590
 1580 DRAW Time,Amp*YScale
 1590 UNTIL OverTime OR Amp=0
 1600 ENDPROC
 1610
 1620 DEF PROCRelease
 1630 REPEAT
 1640 Amp=Amp+AR
 1650 IF Amp<0 Amp=0
 1660 DRAW Time,Amp*YScale
 1670 IF Amp=0 GOTO1700
 1680 Time=Time+T1 MOD128
 1690 DRAW Time,Amp*YScale
 1700 UNTIL Amp=0
 1710 ENDPROC
 1720
 1730 DEF PROCTimeCheck
 1740 OverTime=FALSE
 1750 IF Time>Duration*5 Time=Duration*5
:OverTime=TRUE
 1760 ENDPROC
 1770
 1780 DEF PROCPrint(Phase$,Oset1,Oset2)
 1790 IF Phase$="R" GOTO1860
 1800 VDU5
 1810 MOVE Time+Oset1,Amp*YScale+Oset2
 1820 PRINTPhase$
```

```
1830 MOVE Time,Amp*YScale
1840 VDU4
1850 IF Phase$="r" ENDPROC
1860 PRINTPhase$;"=";Time/100;"  ";
1870 ENDPROC
```

When run, the program will print envelope parameters along the top of the screen and a duration value just below. The pitch values have been set to 0 but can be altered in PROCSetUp. They can't be altered during the course of the program and will not effect the graph which shows only the ADSR envelope.

How to use the program

If you press the space bar, the current envelope will sound and its ADSR graph will be displayed on the screen. The Y (vertical) axis is scaled in amplitude units of 10. The X (horizontal) axis is in seconds. The keys 1 to 8 allow you to alter the amplitude and duration parameters. 1 will alter T, 2 will alter AA, 3 will alter AD, etc. 8 alters the duration. Selection of one of these will cause the present value to flash on and off in red and cyan and a '?' prompt will appear under the envelope. Input the new value and press RETURN and the new envelope will be displayed. If you press RETURN without entering a figure the option will be cancelled and the envelope will remain the same.

You can alter parameters and draw one envelope over another. Pressing 'C' will clear the screen for a new graph. Press 'Q' to quit the program.

If you wish, you can define the function keys to produce the required input and label them accordingly. This will be useful particularly if you add the Pitch Graph Generator program later in the chapter.

```
 10 REM PROGRAM 5.1A
 20 REM Function Key SetUp for
 30 REM ADSR Graph Generator
 40
 50 REM CLEAR
 60 *KEY0 C
 70 REM T
 80 *KEY1 1
 90 REM AA
100 *KEY2 2
110 REM AD
120 *KEY3 3
130 REM AS
140 *KEY4 4
150 REM AR
```

```
160 *KEY5 5
170 REM ALA
180 *KEY6 6
190 REM ALD
200 *KEY7 7
210 REM Duration
220 *KEY8 8
230 REM SOUND
240 *KEY9 " "
```

Run this before loading the ADSR Graph Program, or renumber and add it to the end of the main program as a separate procedure. When the graph is displayed, the letters A, D and S will appear on the screen at a point immediately following completion of that part of the envelope phase. The letter r will appear when the release phase begins. A release phase will always end at 0 (except when it is infinite) unless it is prematurely terminated by another SOUND command or unless the channel is flushed. This will not happen in this program as we are dealing with only one note at a time.

A running total of the time taken so far, in seconds, is printed after each phase along the bottom of the text screen, the time after R is the total time taken for the execution of the envelope. This includes the release phase. Very fast attack times will be printed in exponent format (see the User Guide pages 225 and 236 for details of print formats).

Program notes

PROCSetUp and PROCAxis are self-explanatory. The main program works through the REPEAT loop in lines 100 to 160.

PROCSound produces the sound you have set up in the envelope and then initiates the graph-drawing procedures.

The Pitch and T variables are given assistants in Pitl and Tl. This is so that we can modify these parameters to make allowances for the way these values work in the ENVELOPE command, as in lines 1080 and 1090, without disturbing the original values.

YScale increases the Y values of the graph so it fills the screen. The X axis represents seconds and one X displacement represents one hundredth of a second.

PROCAttack, PROCDecay, PROCSustain and PROCRelease are similar and a description of PROCAttack will cover the principles involved. The amplitude is first incremented by AA, checked to see it is not greater than the ALA level and then drawn. Time is incremented by Tl - MOD 128 so that it stays within amplitude boundaries - and checked to see if it is greater than the D parameter. If not, it is drawn along the 3£ axis. These two DRAW commands produce the stepped effect which you will hear with anything but the smallest values of T.

If the input is a number it calls PROCEnv which controls the text display and the alteration of parameters. The parameters flash because they are given their own colour value which is used with the COLOUR command just before printing in PROCPrintEnv. PROCReset sets all colours to white and PROCEnv sets the selected parameter's colour to flash. PROCAlter accepts a new input and allocates it to the selected parameter. PROCPrintEnv prints the new envelope and returns to the main loop.

PROCPrint prints A, D, S and r at the appropriate time and place upon completion of the appropriate phases.

In order to keep the program to a reasonable length there is no scaling, so sounds which last more than 12 seconds will run off the graph, and there is no error-checking. You can input almost any value and get some sort of sound from the program but, unless the values follow the parameter ranges set out in Figure 5.1, the graph will not always follow the sound accurately

The volume range: hardware and software differences

It is as well to point out now that the volume level can only vary through 16 states which range from - 15 (maximum volume) to 0 (off). The parameters we feed into the ENVELOPE command would have us believe otherwise but you can hear the 16 discrete intervals quite easily with large values of T. Rather than try to show volume variations in terms of these 16 steps, the program produces steps according to the ENVELOPE parameters.

The important areas of the graph are the start and end points of the various phases. The steps will indicate the amount of software movement produced by the envelope, not by the sound chip.

Now, having mentioned it, it will probably be better to ignore it. You will find it easier to construct envelopes as if the volume varied through the whole envelope range and, in practice, you will find that the difference will have little effect upon your envelopes. This information will be of use if you want to write direct to the sound chip (apart from which it explains why you can only hear l6 pitch variations)

A detailed look at the amplitude commands

If you follow the parameter ranges listed in Figure 5.1, you will not go out of bounds but there are other things you need to be aware of:

1) The T parameter determines the length of each step in hundredths of a second. This affects both the amplitude and the pitch envelopes and is critical to both. It determines how often the sound is updated by the OS. With T set to 1, the sound is checked and updated every hundredth of a second. If T is 5, it is only updated every twentieth of a second. The update is in the form of a new pitch or volume command. This supersedes the old value and, when T is greater than I, you can hear the new value taking over from the old one. This results in the stepped effect which we have

discussed.

When T equals 0 it produces the same effect as when it is set to 1 and the resulting sounds will be the same.

Values of T over 127 determine whether or not the pitch envelope repeats. A value of 129 produces the same amplitude envelope as a value of 1. A value of 128 is the same as a value of 0 (which is the same as a value of 1). Large values of T will run off the screen.

2) AA can have a positive or a negative value. A negative value is only of use if it can decrement a positive value. IF a sound follows a sound with a positive amplitude (eg a one whose volume has not yet fallen to zero) then AA can be negative but ALA must be set below the existing amplitude level. This can't be clearly shown in this program as it only deals with one SOUND and ENVELOPE command at a time and all new sounds begin with an amplitude of 0. If, however, you attempt to reach a positive ALA level with a negative AA value, the amplitude level will simply jump to the ALA value. The program, of course, will try to draw it - unsuccessfully.

3) Likewise, AD can take a positive or negative value. Usually it is negative but, if ALA is below the 126 maximum, it can have a positive value and the amplitude will continue to increase. If AD is set to 0 the sound will continue at the ALA level until the sustain phase is reached, regardless of the ALD value.

It is worth noting that, to move from an ALA of 126 to an ALD of, say, 100, setting AD to - 126 will get you there no quicker than setting it to -26 as it only needs to be reduced by 26 to reach the ALE) and it will only take one T step to do so.

4) AS must be 0 or negative. This controls how quickly the sound decays over the remainder of the duration period. The target amplitude for the sustain and release phases is always 0.

5) AR must also be 0 or negative. If it is 0 and the amplitude has not reached 0, the sound will continue indefinitely and the graph will try to draw it. AR comes into play as soon as the duration period has expired and you will notice that, if any phase fails to complete before the duration has expired, the sound immediately passes to the release phase.

6) A duration of 255 will also make the sound last indefinitely.

Using the program and further suggestions

The best way to discover how the ADSR envelope works is to experiment with the program. You can add extra error-checking routines if you wish. This will be easy if they simply restrict inputs to the ranges in Figure 5.1,

but, for complete error-checking, this is not enough. It would be necessary to ensure, for example, that a positive AD value leads to an ALD value higher than ALA. You could also add a scaling feature so that envelopes over 12 seconds would fit the screen. It is possible to produce envelopes lasting several minutes although they may be monotonous to listen to. The program will draw most envelopes.

The program is also useful for calculating the exact time taken by particular phases of the envelope and it is interesting to alter just one parameter and compare the resulting effects. Synthesists may find it easier to relate to a specific attack time rather than an attack increment. The times taken by the attack and decay phases are not explicitly stated in the ENVELOPE command. Both phases end when the amplitude has reached its preset value or when the duration has expired. Apart from using the program, the attack time can be calculated by the following, assuming the duration is long enough to allow it to complete:

Attack Time=ALA/AA*T

To calculate the decay time we need to take into account the amplitude at the start of the decay phase:

Decay Time=ABS(ALA-ALD)/ABS(AD)*T

The ABS function (see User Guide page 200) ensures all values are positive.

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 this chapter may give you some ideas to experiment with.

The pitch envelope

You may be relieved to know that this is slightly easier to understand than the amplitude envelope. This is the part of the envelope which allows us to create many weird and wonderful effects as well as more subtle musical sounds.

As the amplitude envelope varies the volume of the note over a period of time, so the pitch envelope varies the pitch of the note. The initial frequency on which the pitch envelope works is determined by the value of P in the SOUND statement.

PI and PN: the pitch change and the number of steps

The six pitch parameters are paired up, PI1 with PN1, etc, and each set of two acts in the same way PI determines the change of pitch per step much as AA and AD, etc determine the change of amplitude per step. The PN

parameter determines the number of steps in its corresponding PI section and has no direct amplitude envelope equivalent. If it did, it would be calculated from the formulae given above, but omitting T from the calculation.

The change of pitch referred to by PI is in quarter of a semitone increments, the same as the SOUND statement, so a value of 4 will produce a pitch change of a semitone.

Figure 5.4 shows how a single PI/PN pair might affect the pitch. **Figure 5.5** illustrates two complete pitch envelopes. Note that the value of T does not affect the actual pitch changes, only the speed with which they alter and whether or not the envelope repeats. Large values of T will make the individual changes more obvious.

Figure 5.4

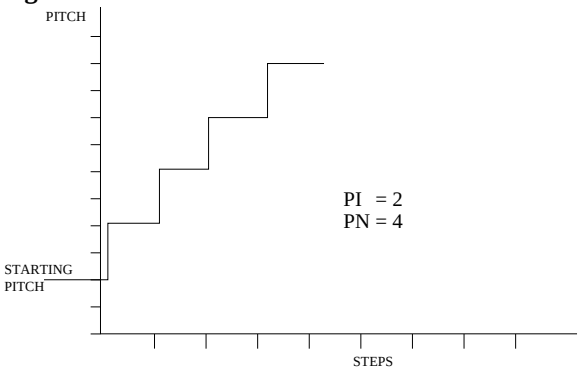


Figure 5.5a

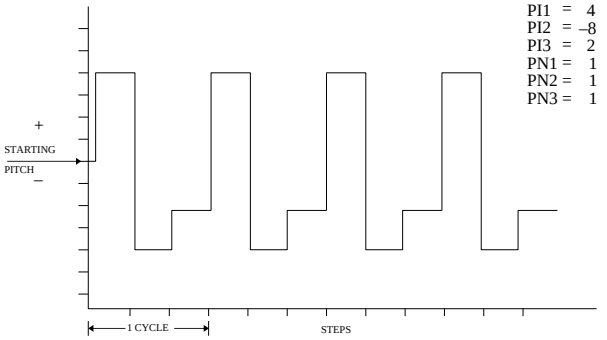
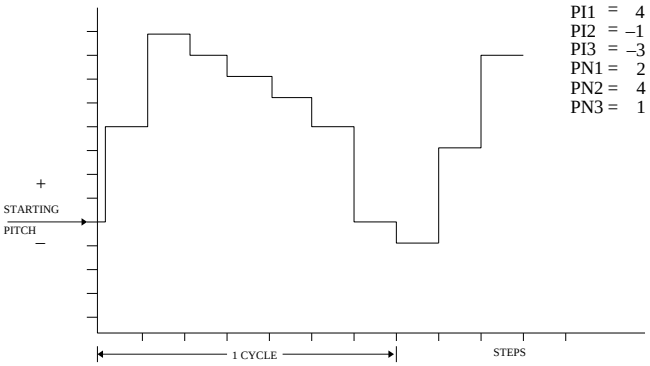


Figure 5.5b



When you write your own programs, be careful not to include PI as a variable or it will be taken as the constant PI with a value of 3.14159265. The use of PH and PI2, etc is quite acceptable.

The Pitch Graph Generator program

After running the previous program, you will now want to see what the pitch envelope looks like. The following additions can be made to the last program and will allow you to alter the pitch and pitch envelope parameters in a similar way to the ADSR parameters in the last program. Seven fines need to be added to the body of the program, two need to be altered and the rest is added to the end.

When entering line 810, enter PRINT in its abbreviated form (ie P.) otherwise it will not fit into one line. (See the User Guide page 484.)

```
    9 REM PROGRAM 5.2 (Combined)
   10 REM PROGRAM 5.1
   19 REM ADSR & PITCH Graph Generator
   20 REM ADSR Graph Generator
   24 REM Added Line Numbers are not
   25 REM Multiples of 10 and are+
   26 REM 75,145,762,764,855,1242,1244
   27 REM Plus 2000 to 2670
   29 REM Altered Lines=150,810
   35
   75 PROCAxis2
  145 IF Input$>="!" AND Input$<="'" Inp
ut=ASC(Input$):PROCPitEnv
  762 pi1=3:pi2=3:pi3=3
  764 pn1=3:pn2=3:pn3=3:p=3
```

```
  810 COLOURb:PRINT;",";:COLOURpi1:PRINT
;PI1;:COLOURb:PRINT;",";:COLOURpi2:PRINT
;PI2;:COLOURb:PRINT;",";:COLOURpi3:PRINT
;PI3;:COLOURb:PRINT;",";:COLOURpn1:PRINT
;PN1;:COLOURb:PRINT;",";:COLOURpn2:PRINT
;PN2;:COLOURb:PRINT;",";:COLOURpn3:PRINT
;PN3;:COLOURb:PRINT;",";
  855 COLOURp:PRINTTAB(23,1)SPC(6)TAB(18
,1)"Pitch=";Pit1:COLOURb
 1242 FinalTime=Time
 1244 PROCPitchEnv
 2000 DEF PROCAxis2
 2010 YScale=3
 2020 VDU29,0;0;
 2030 VDU5
 2040 FOR Mark%=60 TO 780 STEP YScale*10
 2050 MOVE40,Mark%+YScale*10:PRINT"-"
 2060 NEXT Mark%
 2070 VDU4
 2080 VDU29,50;50;
 2090 ENDPROC
 2100
 2110 DEF PROCP(pi,pn)
 2120 FOR P%=1 TO pn
 2130 Pitch=Pitch+pi
 2140 IF Pitch>255 Pitch=Pitch MOD256
 2150 IF Pitch<0 Pitch=Pitch+256
 2160 DRAW Time,Pitch*YScale
 2170 Time=Time+T1 MOD128
 2180 PROCFinalTimeCheck:IF FTime P%=pn
 2190 DRAW Time,Pitch*YScale
 2200 NEXT P%
 2210 ENDPROC
 2220
 2230 DEF PROCPitEnv
 2240 PROCReset
 2250 IF Input=33 pi1=1
 2260 IF Input=34 pi2=1
 2270 IF Input=35 pi3=1
 2280 IF Input=36 pn1=1
 2290 IF Input=37 pn2=1
```

```
2300 IF Input=38 pn3=1
2310 IF Input=39 pn3=1
2320 PROCPrintEnv
2330 PROCAlterPit
2340 PROCPrintEnv
2350 ENDPROC
```

The combined programs are quite long and disk users may get a No Room error when running, especially if the function key program has been added. The extra memory required is not a lot and the problem is most quickly solved by lowering PAGE before loading and running. Enter:

PAGE=&1100

before loading. The program will sit in memory quite easily; it is only when MODE 1 is selected that it discovers it has not enough room. Do not press BREAK otherwise PAGE will be reset and you will probably lose the program.

The removal of REM statements is another alternative but beware of coming back to the program a few weeks later and wondering what the various sections do. Cassette users will have no memory problems.

Using the program

You now have control over pitch and the six pitch envelope parameters. These are accessed by pressing SHIFT and the keys 1 to 7, ie the characters !, ", #, $, %, & and '. You may find it more convenient to use the function keys to control the ADSR parameters, set SHIFT LOCK and use the number keys for the pitch parameters. In practice, once you are familiar with the ADSR envelope, you will probably leave it set (alter the initial parameters in lines 340 to 360) and concentrate on the pitch envelope

A second set of points along the Y axis indicates the pitch values and is scaled in units of 10. The X axis is still the same and is in seconds. The ADSR graph is drawn first, followed by the pitch graph. Some envelopes can take quite a while to draw and when superimposing one graph on another it is not always possible to tell when a graph is finished. So, when complete, a double '??' prompt indicates that the program is ready for further instructions.

Program notes

The main additions are PROCAxis2 to draw the second Y axis and the

inclusion of p, pi and pn colour values in PROCReset in fines 762 and 764. Line 810 puts them into PROCPrintEnv. Note the change of YScale in line 2010 and 2520 during the drawing of the pitch graph.

The input at line 145 allows us to input a string containing the double quotes character (") but this is changed to an ASCII value to allow easy handling of the input.

PROCPitEnv serves the same purpose as PROCEnv, and PROCAlterPit is similar to PROCAlter.

Like T, Pitch has an assistant in Pit1, so whatever value the pitch actually reaches during the envelope execution, the program knows the original pitch value. See lines 2490, 2530, 2580 and the original 340.

The drawing is done by PROCP which is the same for all three PI/PN pairs and is called with the three sets of parameters by PROCPitchEnv. In line 2590, if T is greater than 127, ie the envelope is set not to repeat, the pitch envelope is only drawn once.

The total duration of the sound after the ADSR graph has been drawn is put into FinalTime as the pitch envelope will continue until the release phase completes. PROCFinalTimeCheck performs a similar function to PROCOverTime.

There is still no error-checking and out of range parameters will produce incorrect graphs, but we do not have to worry about whether or not we are moving up or down to ALA or ALD levels.

Apparent peculiarities of the pitch envelope

It is worth demonstrating that, although the three PI and PN parameters perform the same functions, they do not always behave as you might expect - and hope. For example try this:

```
10 ENVELOPE1,20,4,0 ,0,1,0,0,126,-1,0,-6,126,80
20 SOUND1,1,101,80
```

It is the same as this:

```
10 ENVELOPE1,20,4,0,0,1,0,1,126,-1,0,-6,126,80
20 SOUND1,1,101,80
```

The value of PI1, 4, takes the pitch up a semitone and it stays there. One might expect the same result to follow from this:

```
10 ENVELOPE1,20,0,0,4,0,0,1,126,- 1,0,-6,126,80
20 SOUND1,l,101,80
```

All we've done is to transfer the working parameters to another of the three PI/PN sets. Even more peculiar is this:

```
10 ENVELOPE1,20 ,0,0,4,1,0,1,126,- 1,0, -6,126,80
20 SOUNDl.1.10l.80
```

This envelope produces an oscillating effect while the second example doesn't because the T parameter is set to auto repeat and, when the pitch phase completes, the pitch is set back to its original value before beginning again. In the second example, PI1 acts immediately upon the pitch to raise it a semitone. This is one factor which can cause a few problems when trying to work out a precise pitch effect and on such occasions it is always worth trying out the envelope with no repeat.

You may wonder why you would want to set a PN parameter if its corresponding PI value is 0. Even with a PI value of 0 the PN parameter creates a gap or time span with no new pitch change. This can be used to create a delay before a pitch modulation.

Notice also that, if a pitch envelope tries to take the pitch over 255 or under 0, it will add or subtract 256 to bring it within range. This can produce some fascinating effects which can be seen on the graph.

The value of T and bit 7

The User Guide tells us that if bit 7 of T is set to 0 the pitch envelope will auto repeat, and if it is set to 1 it will only sound once. This was mentioned in the ADSR section but now we will examine it more closely.

To understand what a bit is, it is necessary to know a little about the binary system. A complete discussion is beyond the scope of this book but the following explanation may help a little.

Bit stands for BInary digiT, which is the term given to the individual digits which go to make up a binary number. For a variety of reasons it is convenient to work with binary numbers containing eight digits. Computers count, more often than not, beginning at 0, unlike humans who tend to start counting at 1. These eight digits or bits are numbered 0 to 7 from right to left. Bit 7, therefore, is the leftmost digit which will have a value of 128 (2^7) in the binary system. If this bit is set to 1, ie 1*2^7, then the value will be increased by 128.

On page 245 the User Guide gives the range of T as being from 0 to 127, and on page 182 from 1 to 127. If T equals 0, the envelope operates as though T equals 1 and I suggest 1 to 127 as the correct range. If you add 128 to the value of T you are setting the seventh bit to 1. As far as the ADSR envelope is concerned, it reduces any value over 128 by the MOD command (see User Guide paige 299) so 128 MOD 128=0 or, as I would have it, the upper range should run from 129 so we would say 129 MOD 128=1. By this reckoning, 130 has the same effect as 2, 131 as 3, etc. If you put larger values in, they are still reduced so, for example, 12345 MOD 128=57.

The pitch envelope does the same thing only it takes note of whether or

not the seventh bit has been set. If it has not, it repeats its envelope.

Experimenting with the programs

Apart from idiosyncracies you may come across in the SOUND and ENVELOPE commands, you will find that being able to relate the numbers to a visual representation of the sound is a great help in mastering the commands. If you are searching for a specific effect and can see a visual result of your efforts, you will find it easier to make adjustments and find out why a sound may not be behaving as you expect.

Further experiments can be done by including the noise channel and controlling its frequency from channel I by setting the P parameter to 3 or 7 as follows:

```
1045 SOUND 0,-15,3,254
```

If you set the ADSR parameters to 0 you will hear the results of the pitch envelope on channel 1 working through channel 0. The noise channel is examined more thoroughly in Chapter 7.

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 sounds which you like. With the addition of a few frills, courtesy of the pitch 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.

The octave range in Figure 5.6 refers to the notes on the staves in figure 2.4. For example, the octave range 2 to 4 extends from pitch number 53 to 193. As we have offset the keyboard in Figure 2.4 to compensate for the predominantly high notes of the sound chip, remember that the actual instruments are pitched an octave lower.

The envelope shapes are described in terms of attack and decay. In this context, the decay refers to the release phase. Our ADSR decay phase will run along similar lines to the sample instrument envelopes in Figure 5.3. The sustain phase will vary. An organ note can have an infinite sustain, a guitar string has none and a trumpet can last as long as the player can keep blowing.

Figure 5.6

INSTRUMENT	OCTAVE RANGE (See Figure 3.3)	ATTACK	DECAY	APPLICATIONS AND SPECIAL EFFECTS
Violin	3 to 5	slow	medium/slow	portamento, vibrato, scales
Viola	2 to 4	slow	medium/slow	
Cello	1 to 3	slow	medium/slow	
Bass guitar	1 to 3	fast	slow	rhythmic figures
Trumpet	3 to 5	fast	medium/fast	vibrato
Trombone	1 to 3	medium/fast	medium/fast	portamento
Tuba	1 to 3	slow	medium/fast	use for staccato bass fine
Alto saxophone	2 to 4	medium	fast medium/fast	slides up to note
Tenor saxophone	1 to 3	medium/fast	medium/fast	bends the note
Flute	3 to 5	medium/fast	medium/fast	trills,arpeggios
Clarinet	2 to 4	medium/fast	medium/fast	slow vibrato
Oboe	3 to 5	medium/slow	medium/fast	
Bassoon	1 to 3	fast medium	fast slow	mournful
Guitar	2 to 4	fast	slow	bend the note
Accordion	1 to 6	medium/slow	medium/fast	slightly out of tune
Harp	1 to 6	fast	slow arpeggios	glissando
Xylophone	3 to 6	fast	medium/fast	glissando
Snare drum	medium	fast	medium	drum rolls
Bass drum	slow	fast	medium	short beats
Organ	1 to 5	fast	fast	vibrtao, full chords

You do not have to try to faithfully imitate an instrument - create your

own. The characteristics are there for information only, feel free to use them or ignore them completely, according to your musical tastee 5.6

Producing other waveforms

If you want to take your interest in music and the BBC micro even further, there are ways to produce waveforms other than the square wave of the sound chip. These involve connecting a digital to analogue converter to your system, and setting up the computer to produce cycles of a particular wave in a similar way to the Sine Wave Plotter program in the first chapter. The cycles are generated by rapidly altering the contents of a series of memory locations and you need to program in assembly language to produce the cycles fast enough to be of use.

The information required to implement such a process would fill several more chapters, if not a book, and is mentioned here only to illustrate the possibilities open to the experimenter.

The addition of a proper musical keyboard and separate synthesiser voice modules is another possibility but we'll leave all that to the more experienced and adventurous explorer.

CHAPTER 6
Musical Miscellanea

A synthesiser has many modules or sections which allow the synthesist to enhance and embellish the basic sounds he or she creates. Some of the resulting effects are quite intricate; many are subtle, some are not. The musician can buy a mountain of effects pedals and boxes to produce sustain, echo, reverberation, chorusing, phasing, flanging .. .. The list goes on. In this chapter we will see what effects we can produce on the BBC micro.

Vibrato and tremolo: pitch and amplitude modulation

These are usually the first effects a synthesist learns to incorporate into his or her 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 if any exists 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 vibrato or tremolo to enhance their tone. Vibrato is far more common and is 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 probably the easier of the two to duplicate so we will look at that first.

The pitch variation in vibrato is not usually as great as a semitone and the quarter semitone increments of the sound chip are very useful in duplicating this effect. In a musical vibrato, the pitch varies regularly, rising and falling in a sine wave pattern moving above and below the pitch of the note. Extreme examples, where the pitch varies more rapidly and over larger intervals, are still technically vibrato but are really only produced by electronic means for special effects such as this:

```
10 ENVELOPE1,1,0,3,-3,0,20,20,63,-1,0, -4,126,100
20 SOUND1,1,101,l60
```

Alteration of the speed and degree of pitch variation will drastically alter the sound and, although we can create innumerable vibrato effects, only a few will be of any use in a strictly musical context.

Comparing envelopes

Because of the rather subtle nature of vibrato, it is not always easy to tell the difference between one degree of vibrato and another unless they are considerably pronounced. The nature of the sound will change, too, as the note changes from one octave to another. What we need is a program which allows us to compare one envelope with another, alter a parameter here and there, and test out the new sound. The following program was designed as such a utility. It has been kept purposely short and is easily modified and adapted to cater for different requirements. Some suggestions for modifications are made after the listing.

```
 10 REM PROGRAM 6.1
 20 REM Envelope Comparisons
 30
 40 VDU15
 50 ON ERROR GOTO 230
 60 REM Reset Delay on Keys
 70 *FX12,0
 80 *KEY0 RUN|M
 90 *KEY10 OLD|MLIST|M
100 PROCEnvelope
110
120 REPEAT
130 PRINT"Which Envelope?";
140 REPEAT
150 Env=GET-48
160 UNTIL Env>-1 AND Env<10
170 PRINT;TAB(16)Env
180 IF Env=0 SOUND1,-12,Pitch,40 ELSE
```

```
SOUND1,Env,Pitch,40
  190 UNTIL FALSE
  200
  210 REM Escape Routine
  220 REM Speed Up Key Repeat
  230 *FX12,6
  240 *FX11,20
  250 ON ERROR WHEN 280,
  260 ERROR
  270
  280 DEF PROCEnvelope
  290 Pitch=149
  300 ENVELOPE1,4,0,0,1,1,0,1,4,-1,0,-3,
126,80
  310 ENVELOPE2,2,0,0,1,2,0,2,4,-1,0,-3,
126,80
  320 ENVELOPE3,3,0,0,1,2,0,2,4,-1,0,-3,
126,80
  330 ENVELOPE4,4,-2,1,1,1,1,1,4,-1,0,-3
,126,80
  340 ENVELOPE5,6,1,-2,1,1,1,1,4,-1,0,-3
,126,80
  350 ENVELOPE6,4,1,1,1,1,1,1,4,-1,0,-3,
126,80
  360 ENVELOPE7,4,0,8,-8,0,1,1,16,-1,0,-
3,126,80
  370 ENVELOPE8,4,0,28,-28,0,1,1,16,-1,0
,-3,126,80
  380 ENVELOPE9,8,0,8,-12,1,1,1,16,-1,0,
-3,126,80
  390 ENDPROC
```

Upon running, you will be asked to input an envelope number. This envelope will be used to produce a sound. If you input 0, an unmodified note will be heard.

To alter an envelope, press ESCAPE and PROCEnvelope will list to the screen. The key repeat period will be speeded up, too. I prefer these faster keys for editing but you can remove fines 70, 230 and 240 if you wish, or alter the repeat period to suit. The necessary envelopes can be edited and the program run again by pressing £0. In this way you will be able to compare two or more envelopes. If one immediately follows

another, the release phase will not occur, which is how most sounds are heard in music.

Program notes

When typing in the program, preface fines 50, 250 and 260 with REMs in case you enter some of the lines incorrectly. Otherwise, the program's error routines will take over and you will find debugging difficult.

Line 40 ensures that the screen is not in page mode. *KEYI0, the BREAK key, is programmed to LIST and pressing BREAK will, of course, set the key repeat periods to their default values. We do not want fast keys when we are entering envelope numbers, so they are set to default in line 70. When ESCAPE is pressed, control passes to fine 230 which speeds up the keys. The ON ERROR command at line 250 is immediately followed by an error in line 260. The word ERROR cannot stand alone and creates a syntax error which causes the program to list. You could just as easily substitute any other letters the computer does not recognise.

After altering the envelopes and pressing £0 to run, you will still be able to see the envelope parameters before they scroll off the screen.

As it stands, you can program up to nine envelopes. If this is not enough, you can expand the input parameters in lines 150 and 160 to take up to 16 envelopes.

By putting the envelope parameters into DATA statements, you could cause the envelopes to be printed to the screen as they sound. Also, by incorporating RESTORE xxx in the escape routine following line 210, where xxx is the line number containing the envelope parameters, you could cause only the last envelope called to be listed. That may seem to be taking a utility program just a step too far but you may find it helpful and it would be an interesting exercise.

Function keys 1 to 6 can be programmed to alter Pitch to produce a note from each octave. The fact that the sound does vary so much throughout the sound chip's range can be used to advantage both in music and sound effects.

Using the program

You will soon find that there are few vibrato effects suitable for use in a purely musical context as extremes are simply not musical. However, the program will be useful for comparing some of the other musical ornaments detailed in this chapter. You may also want to use the program to compare more severe forms of vibrato which add so much to game programs. Alter the Pitch parameter, too, during your editing as this makes a tremendous difference to the note.

Only ENVELOPEs 1 to 6 produce a vibrato effect and you can see by comparing I and 2 that an alteration of the T parameter needs to be compensated for in the FN parameter. You will also notice that

ENVELOPE 4 sounds lower than the pitch of the note. This is because P11 immediately sets the pitch down two steps for reasons that were discussed in the last chapter. The input 0 option lets you compare the actual pitch with the pitch after modification. This will tell you that ENVELOPE 4 is not quite right.

ENVELOPEs 7, 8 and 9 and their effects will be discussed under trills and echoes.

Producing tremolo effects

The ENVELOPE command provides a repeat option on the pitch envelope but not the amplitude envelope. To produce' tremolo, we ideally need the latter. In its absence we must use a loop. This in itself causes complications, especially if we want to use tremolo during the production of a tune. The next program demonstrates two tremolo production techniques.

```
   10 REM PROGRAM 6.2
   20 REM Tremolo Demonstration
   30
   40 PROCTremolo(&1,1,-1)
   50 PROCTremolo(&1,1,-.25)
   60 END
   70
   80 DEF PROCTremolo(Chan,Dur,Step)
   90 PRINT"Chan=";Chan;" Dur=";Dur;" St
ep=";Step
  100 REPEAT
  110 FOR Amp=-2 TO -15 STEP Step
  120 SOUND Chan,Amp,53,Dur
  130 NEXT Amp
  140 FOR Amp=-14 TO -3 STEP ABS(Step)
  150 SOUND Chan,Amp,53,Dur
  160 NEXT Amp
  170 UNTIL INKEY$(2)=" "
  180 ENDPROC
  190
  200 REM Tremolo Using Env Control
  210
  220 ENVELOPE1,5,0,0,0,0,0,0,8,-8,0,-8,
120,16
  230 FOR Trem=1 TO 8
  240 SOUND1,1,53,28
```

```
  250 NEXT Trem
  260 END
  270
  280 REM Faster Tremolo
  290
  300 ENVELOPE1,2,0,0,0,0,0,0,8,-8,0,-8,
120,104
  310 FOR  Trem=1 TO 30
  320 SOUND1,1,53,2
  330 NEXT Trem
```

PROCTremolo does it the hard way and alters the amplitude of the SOUND commands one at a time. Pressing the space bar will call the next tremolo effect - only two have been included.

The timing of the loop is actually controlled by the Dur parameter which, in the program, has been set to 1, its smallest value. This might seem to indicate that we can't get a faster tremolo; but if we flush the sound channel we can, and we are then only limited by the speed of the BASIC loop.

Alter line 40 to include a flush command:

```
40 PROCTremolo(&11,1,-1)
```

and notice the effect. It's not very musical because the tremolo takes the note through its complete amplitude range. The Step parameter in line 50 can be used to slow down the BASIC loop. The SOUND command will take no notice of non-integer values and making the loop step through more values than the SOUND command recognises is a good way to waste time.

To produce something more musical reduce the Amp range in fines 110 and 140.

The two short routines tacked on to the end of the program at fines 220 and 300 demonstrate how to produce tremolo with envelope control. The first example produces the same effect as fine 40.

To produce an even tremolo, it is necessary to calculate the attack and decay times and use that as the duration parameter in the SOUND command. If it is too short the sound will cut off during the decay phase and another attack phase will begin. If it is too long there will be a delay before the next cycle.

The tremolo at fine 300 is perhaps more useful and you should now be able to produce a more subtle example. If you do, you will probably notice how similar it is to vibrato. Try the last two examples again but include a vibrato in- the pitch envelope such as:

```
220 ENVELOPE1,5,1,-2,1 ,1,1,1,8,-8,0,-8,120,16
```

Experiments will produce many interesting sounds, with pitch figures produced by the pitch envelope fading in and out with the tremolo effect.

You can see the problem with loops - they take control away from the SOUND command and hold up the BASIC program. In order to put a constant tremolo into a piece of music, the controlling loop has to last for the length of each individual note. In this way, we are constantly interfering with the timing of the tune and we may find it necessary to make synchronization adjustments. This is dealt with in Chapter 9.

In practice, you are unlikely to want tremolo - or vibrato - all the way through a piece; as with all embellishments, too much ;and it ceases to be pleasant.

While vibrato is more musically useful and easier to apply, the full potential of tremolo has not been realised and its use as a source of sound effects can lead to something just that little bit different.

Trills: a special kind of vibrato

To produce vibrato on an instrument you need control over the pitches in between individual notes. You can't produce vibrato on a piano, for example, as the notes are fixed. The best you can do is to alternate rapidly between two adjacent notes and this is called a trill. ENVELOPE 7 in Program 6.1 produces a trill and you can hear the pitch vary between C (Pitch = 149) and D (Pitch = 157). 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 between notes.

You may have realised that the vibratos produced in Program 6.1 jump up and down in a similar way except they jump one pitch interval at a time, ie quarter of a semitone. If T is set fairly low, we can't hear the individual pitches and the ear assumes the pitch is varying in a smooth and continuous manner.

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. With a duration value of I it will sound about fight, 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 and ENVELOPE 8 in Program 6.1 plays a trill over an interval of a fifth (see Chapter 2 for further information about intervals).

A flute with the 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.

```
   10 REM PROGRAM 6.3
   20 REM "Military Music" Introduction
   30
   40 ENVELOPE1,4,0,8,-8,0,1,1,63,0,0,-1
2,126,126
   50 ENVELOPE2,3,0,0,1,2,0,2,126,-8,0,-
8,126,30
   55 ENVELOPE3,4,0,0,1,1,0,1,32,-1,0,-8
,96,60
   60 SOUND1,1,149,54
   70 FOR Note=1 TO 9
   80 READ Pitch,Dur
   90 SOUND2,3,Pitch,Dur
  100 SOUND3,2,Pitch+48,Dur
  110 NEXT Note
  120 END
  130 DATA 53,12,49,4,45,12,41,4,33,4,25
,4,21,4,13,4,5,6
```

ECHO ECHo ECho Echo echo and reverberationnnn

These are probably two of the most overused effects in the synthesist's armoury. They are great fun to play with, which is why they are overused, but in the right hands they are also 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 that you must stand at least 54 feet away from the cliff. If you are closer than this, the echo will not seem distinct but will seem to be a continuation of your original shout.

A sound emitted in a room will bounce around the wall, 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 or reverb time is the length of time required for the sound reflections in a room to fall to a certain level. Rooms with reflective surfaces have longer reverb times than rooms with insulated walls, which is why more people prefer to sing in the bathroom than in a padded cell.

Most commercial reverb units produce their effects by means of a spring or metal plate. Because of the enormous number of vibrations involved, reverb is not possible on the BBC micro (although you can feed the output to a reverb unit) but we can imitate an echo.

Commercial echo units

An echo is a single repeat of a note. Echo units are capable of producing from one to, say, five or six echoes. Many can produce even more and some are capable of an infinite number. A quality of natural echo, which you will be aware of, is that each echo is quieter than the preceding one. If an echo is repeated many times, there is a limit to how low the volume can go without becoming inaudible altogether and most units ,allow you to control the repeat volume so that the echoes fade quickly or slowly.

One type of unit popular with singers consists of a series of tape recorder heads. The first one records the sound and the following ones play it back.

Solid state echo units are available which use either analogue or digital techniques. The analogue units contain circuits which hold the signal for a short while to cause a delay before passing it to other circuits which do likewise. This form of echo production is known as a bucket brigade method for obvious reasons. As the signal passes from one circuit to another, the quality soon deteriorates and there are severe limits on how many times an echo produced in this way can repeat.

Digital units work on a similar principle, but as they pass around a series of numbers which do not deteriorate they can produce echoes which go on for ever.

Producing echoes on the BBC micro

There are several methods we can use to produce echo, depending upon the particular echo effect we require.

As a starting point, assume we want to repeat a single note. Ideally, we should be able to create this with a single ENVELOPE command - but we can't. To produce an echo, there must be a discernible gap between notes which means the volume has to drop to zero and rise again. We cannot do this with one envelope unless we play it twice, and that alone would not produce a drop in volume.

One method is to use a series of envelopes and play the sound using each in turn like this:

```
10 REM PROGRAM 6.4
20 REM Echo Production
30 REM Using Multiple Envelopes
40
50 Dur=3
60 ENVELOPE1,1,0,0,0,0,0,0,126,-4,-4,
```

```
-6,126,102
   70 ENVELOPE2,1,0,0,0,0,0,0,102,-4,-4,
-6,102,78
   80 ENVELOPE3,1,0,0,0,0,0,0,78,-4,-4,-
6,78,54
   90 ENVELOPE4,1,0,0,0,0,0,0,54,-4,-4,-
6,54,30
  100 ENVELOPE5,1,0,0,0,0,0,0,30,-4,-4,-
6,30,6
  110 SOUND1,1,101,Dur
  120 SOUND1,2,101,Dur
  130 SOUND1,3,101,Dur
  140 SOUND1,4,101,Dur
  150 SOUND1,5,101,Dur
```

This produces a rather good echo but is wasteful of envelopes. Lines 110 to 150 could be replaced with a FOR . . . NEXT loop for neatness. Try doubling the SOUND commands like this:

```
110 SOUND1,1,101 ,Dur:SOUND1,1,101,Dur
```

The duration of the note determines the echo repeat time and long notes will not produce a very good effect.

The next program tries to reduce the envelope waste by using only one and repeatedly redefining it within a procedure. In doing this, we must be careful not to redefine an envelope before it's ready to be used. Try the program.

```
   10 REM PROGRAM 6.5
   20 REM Echo Using a Procedure
   30
   40 EchoSpeed=10
   50 RateOfDecay=3
   60
   70 FOR Note=1 TO 5
   80 READ Chan,Pitch,Dur
   90 PROCEcho
  100 NEXT
  110 END
  120
  130 DATA 1,5,16,2,33,16,3,53,28,1,69,2
,2,65,32
```

```
  140
  150 DEF PROCEcho
  160 AA=126
  170 FOR Count=1 TO Dur
  180 ALD=AA-RateOfDecay
  190 ENVELOPE1,1,0,0,0,0,0,0,AA,-4,-4,-
1,AA,ALD
  200 SOUNDChan,1,Pitch,1
  210 TIME=0:REPEAT UNTIL TIME>EchoSpeed
  220 AA=ALD
  230 NEXT Count
  240 ENDPROC
```

Line 210 holds up the whole program while the echo runs its course. If we try to build the delay into the SOUND command by, increasing the duration, it does not work. This is because the sound chip stores the commands and, while it is waiting to execute them, the rest of the program has already redefined the envelope the required number of times and we get no echo at all. Remove fine 210 and see what happens.

Note also that the durations given to the SOUND command from the DATA statement at fine 130 no longer determine the length of the note. This is determined by the variable, EchoSpeed. The duration values still maintain the note-length relationship between notes. Alter Echospeed and RateOfDecay and listen to the effect. It may be necessary to alter AD, AS and AR values in the envelope if your values become extreme.

The above examples assume that you want a drop in volume with the repeat. If we do away with the amplitude reduction it detracts from the echo effect which results in only a sequence of notes.

Pseudo echo

You can see that the implementation of a single note echo is not particularly easy, although the results can be very good and well worth the effort. Another alternative is to create a pseudo echo using the pitch envelope. This is often the easiest way and can lead to many unexpected results. The following fines contain sample envelopes for inclusion in the Envelope Comparison program:

```
   10 REM PROGRAM 6.6
   20 REM Pseudo Echoes Using
   30 REM Single Envelopes
   40 REM Insert in PROGRAM 6.1
   50
   60 ENVELOPE1,20,12,-12,0,1,1,0,126,-6
,-6,-6,126,0
```

```
   70
   80 ENVELOPE2,4,4,0,32,4,1,0,126,-1,-1
,-1,126,0
   90
  100 ENVELOPE3,4,4,16,12,1,1,1,126,-1,-
1,-1,126,0
  110
  120 ENVELOPE4,6,8,-16,8,8,4,2,126,-1,-
1,-1,126,0
  130 ENVELOPE5,4,32,-64,4,1,1,16,126,-1
,-1,-1,126,0
```

ENVELOPE 9 in Program 6.1 also contains a pseudo echo.

If you want an echo of two or more notes this is the way to do it. Even if you don't, you can often get away with using two notes where you really only wanted one. In a particularly complicated program, this method will save you a lot of time and effort.

Program 6.6 contains a few ideas to start you off. The temptation is to forget your original purpose and design some complicated envelopes - which is not necessarily a bad thing provided you have the time.

Using the pitch envelope to play tunes

Only one step away from the echo examples, we can devise envelopes that will play a tuneful sequence of notes. The little fanfare in the Motility Tester program in Chapter I could easily have been the result of a sequence of notes read in from a DATA statement. Instead, it was produced by one envelope which is repeated here for analysis:

```
10 ENVELOPE1,11,16,4,8,2,1,1 ,100,0,0,-100,100
20 SOUND1,1,101,20
```

The pitch changes in an envelope can be described as three variations, each variation being a number of rises or falls in pitch, each rise or fall being over the same interval.

There are five main points to consider when inventing envelope tunes and these are:

1) To stay within the western scale, all PI intervals should be in steps of multiples of four.

2) Remember that PI1 affects the pitch immediately so, in the example above, the starting note is E, not C as you might expect.

3) Consider the length of the sequence carefully. If AR is altered from -100 to -1, the release phase will occur and it will sound like another echo

envelope and lose its impact. As it is, the envelope will last as long as the SOUND% duration parameter.

You can calculate the time, in hundredths of a second, of a single pitch envelope as follows:

Time = (PN1+PN2+PN3)*T

If you want to terminate the sound in the middle of a pitch envelope, as above, you can calculate the amount of time required and apply it to the SOUND command by dividing by five. In this case the envelope repeats once and has an extra note on the end. This is a total of 9 xT which is 99. Dividing by five gives us our duration of 19 or 20.

4) T controls the speed of the notes. It is sometimes easier io work with a large value to T so that you can hear what's happening, and then reduce it to the required level when your sequence is correct.

5) If the pitch in the SOUN.D statement is too high or too low, the envelope might take the pitch to the other end of its range. This can be used purposely to produce some good effects.

Correct planning, as they say in textbooks about structured programming, is very important. I will just repeat that, if you are looking for a specific effect, it helps to know first if it's possible and, if it is, how to set about producing it. If you know how the SOUND and ENVELOPE commands work you will not waste your time trying to produce something they can't do.

As a basis for further experiments, here are a few envelopes which play a musical sequence of notes. The first plays an arpeggio of a C diminished chord (see Chapter 2 for further information about chords) up four octaves, comes down in semitones to an octave lower than the start note and rises in semitones until it reaches C. The envelope does not repeat.

```
10 ENVELOPE1,136,12,-4,4,17,60,12,100,0,0,-100,100,100
20 SOUND1,1,41,144
```

Arpeggios of diminished and augmented chords are easy to program because they increase the start note by 3 and 4 semitones respectively.

Playing a scale other than in semitones is difficult because the western scale is composed of:

tone + tone + semitone + tone + tone + tone + semitone

which will not fit into the pitch envelope. We can fiddle our way around

this by simply playing a scale in tones, known appropriately enough as a 'whole tone scale', but this may not harmonise exactly if it is used with other notes. See if you can tell the difference, though:

```
10 ENVELOPEl,10,8,-8,0,18,18,6,100,0,0,-100,100,100
20 SOUND1,1,45,84
```

Notice the use of a value in the PN3 parameter without a corresponding PI3 value to sustain the last note.

There are only two whole tone scales, one contains C and the other one doesn't! The other one contains C#. Check this out on the diagram in Figure 2.4. The whole tone scale is used in the introduction to Stevie Wonder's song, 'You Are the Sunshine of My Life'.

If the tune or effect is to stand alone, you need not restrict yourself to the western scale or pitch intervals. Try PI values in increments of 2, 3, 5 or 6, etc. I leave it to you to experiment further on these lines.

As an example of a sequence going off the top of the scale and coming up from the bottom, try this:

```
10 ENVELOPE1,9,20,20,0,10,10,5 ,100,0,0,-100,100,100
20 SOUND1,1,149,135
```

Finally, you can use two or more envelopes one after the other to produce even more complicated tunes. Having worked through this chapter, you should not find that too difficult.

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 which 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 whooshing sound and such effects can be heard on many electronic music albums. (The sound of a jet plane taking off creates a phasing effect.)

There is more than one story floating around the music world of how

Hanging was invented. One version recalls how flanging was first produced by applying slight pressure to the flange of a spool of tape as it was playing, to cause a small delay. If you can imagine two such tapes being played in this manner and 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. These 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 log 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 BBC micro

Because of the way the BBC micro's sound chip works, if we produce the same pitch on two or more different channels, the pitches will not be exactly the same. This is a chorus effect. The pitch difference is not as great as a pitch interval and, if we add 1 to one of the pitch values, the effect will be more pronounced and less subtle. We can really go to town and use three channels with a pitch difference of 1 between each. You may wish to set the volume of the main pitch SOUND command slightly higher than the others. This short program runs through some of the possibilities:

```
 10 REM PROGRAM 6.7
 20 REM Chorus Effects
 30
 40 REM Alter Pitch
 50 REM To Other Values
 60 Pitch=227
 70
 80 REM 2 Channels
 90 SOUND1,-12,Pitch,120
100 SOUND2,-12,Pitch,120
110 Next=GET
120
130 REM 3 Channels
140 SOUND1,-12,Pitch,120
150 SOUND2,-12,Pitch,120
160 SOUND3,-12,Pitch,120
170 Next=GET
180
```

```
  190 REM 2 Channels with Increased
  200 REM Pitch Value
  210 SOUND1,-12,Pitch,120
  220 SOUND2,-12,Pitch+1,120
  230 Next=GET
  240
  250 REM 3 Channels with Increased
  260 REM Pitch Value
  270 SOUND1,-12,Pitch,120
  280 SOUND2,-12,Pitch+1,120
  290 SOUND3,-12,Pitch-1,120
  300 Next=GET
  310
  320 REM Pseudo Chorus
  330 REM Using Fast Vibrato
  340 ENVELOPE1,1,0,0,1,1,0,1,100,0,0,-1
00,100,100
  350 SOUND1,1,Pitch,120
```

With Pitch set to 227, the first two effects are probably as near as you'll get to a scream - shorten the duration before use! More pleasing and musical effects occur in the lower octaves.

This example tries to produce its effect with the pitch envelope. How well it succeeds I leave to you to judge. At best, it's only half the effect because we only hear one pitch at a time and chorus relies on the interaction between two or more frequencies. You may find it useful, however, when you only have one spare channel.

The other spatial effects are really beyond the capability of the BBC micro's sound chip but we can content ourselves with the thought (even if it's not strictly true) that they just sound like a more sophisticated chorus effect.

Beat frequencies: the weaving in and out

Especially with lower notes, you will hear the frequencies weaving in and out and, at times, the note or notes almost cease. This pulse 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. If we listen to this:

```
10 SOUND1,-12,52,120
20 SOUND2,-12,53,120
```

we should hear a beat frequency of 3 to 4Hz (cycles per second). A pitch value of 52 has a frequency of about 263.1 and 53 has a frequency of about

266.3. Beat frequencies can be heard when playing intervals, too, if the interval is not in tune.

The beat frequencies you will hear, as you run the above program and alter the pitch, will vary and you may find some pitches which produce no beats at all, indicating that they are exactly in tune.

As an example of how you can use this effect, the following program imitates an accordion.

```
   10 REM PROGRAM 6.8
   20 REM French Accordion Music
   30 REM Using Chorus Effect
   40
   50 ENVELOPE1,3,0,0,0,0,0,0,8,-1,0,-6,
124,60
   60
   70 FOR Tune=1 TO 16
   80 READ Note,Dur
   90 IF Note=0 Amp=0 ELSE Amp=1
  100 SOUND1,Amp,Note,Dur
  110 SOUND2,Amp,Note,Dur
  120 NEXT Tune
  130 END
  140
  150 DATA 61,8,77,8,113,8,109,72
  160 DATA 61,8,81,8,113,8,109,32
  170 DATA 0,8,93,8,101,8,0,4,65,4,61,4,
53,4,61,4
```

This uses the natural beat frequencies of the sound chip. You can increase the effect by adding a whole pitch and, less subtle though it may be, I like this better:

```
110 SOUND2,Amp,Pitch+1,Dur
```

Notice also the way we create a gap in the music - switching from envelope control to a SOUND command with an amplitude of 0.

The ring modulator: producing bells and other ringing noises

The last synthesiser module we will examine and try to duplicate is the ring modulator. This is responsible for producing 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 modulator requires an input of two frequencies: it produces an output which is a compound of the sum of these two frequencies and the difference between them. To take an example, if the modulator was fed with a frequency of 440Hz and 1220 Hz, the output would be a compound of 1660Hz (the sum: 440 + 1220) and 760Hz (the difference: 1200-440).

That's fairly straightforward, but the BBC micro is not calibrated to work in hertz and so our choice of frequencies is limited. Also, we do not know what frequency is produced by each pitch command. The only way to discover this is to measure it. Figure 6.1 displays a table of the frequencies in hertz produced by the range of P values. The measurements were taken from a single BBC micro and it is quite likely that measurements taken from other machines will show slightly different figures, but for our purposes these will be accurate enough.

Before we see how to produce bell-like sounds, let us have a closer look at the results shown in Figure 6.1.

The frequencies produced by the sound chip

If you look at the table, you will notice that the sound chip is not terribly accurate and that some adjacent values of P produce very similar frequencies. Unless you want to compose music in quarter of a semitone intervals, this should make no difference. Although the ear is capable of distinguishing between differences in pitch of as little as 10 Hz, musically, it makes many allowances for such discrepancies. Only if you have perfect pitch (also called absolute pitch - the ability to identify the correct pitch of a note) or the tuning is out by about half a semitone will your ear be offended. Notice, in fact, how pleasing the difference of 1 in the P parameter can be, as illustrated by the last program.

Through our experiments with scales you will have realised that, when the range is stepped through in semitone intervals, the sound is quite acceptable. Only when you reach the very upper range of the sound chip may you find tuning problems.

The out of tune chip

Even if you are not a musician in the strictest sense of the word, you may have heard of the international tuning standard which stipulates a value of 440 Hz for A above middle C. A glance at our table reveals that this would be more accurately produced by a P value of 88, not 89. In fact, the whole of the scale seems to be one pitch value out Unless the sound chips vary throughout production (which they do not appear to do), it would appear we could get a more accurate musical tuning if we gave the lowest B a value of 0 and worked upwards from there in steps of 4. It would tend to make more sense, too, instead of starting with 1. In the fight of this, it is a bit of a mystery why the User Guide stipulates the scale as it does.

This whole topic, however, I leave to the purists. You can adjust the values if you wish but there is little to be gained. I have used the values given in the User Guide to avoid confusion and only the most sensitive ears are likely to know the difference. If you want to use your computer to play along with other instruments you may have to make adjustments to the basic pitch values.

Figure 6.1

P Param	Freq Hz	Note
0	124.0	
1	125.7	B
2	127.5	
3	129.4	
4	131.4	
5	133.3	C
6	135.8	
7	136.9	
8	139.2	
9	141.8	C#
10	143.8	
11	144.9	
12	147.5	
13	149.7	D
14	151.8	
15	154.8	
16	156.2	
17	158.4	D#
18	160.6	
19	162.9	
20	165.5	
21	167.7	E
22	178.8	
23	172.3	
24	175.5	
25	178.8	F
26	188.5	
27	183.2	
28	185.9	
29	188.4	F#
30	191.8	
31	193.7	
32	196.8	
33	199.6	G

34	202.5	
35	205.5	
36	208.5	
37	211.4	G#
38	214.4	
39	217.3	
40	221.1	
41	224.3	A
42	227.5	
43	230.9	
44	234.8	
45	237.1	A#
46	240.3	
47	234.6	
48	247.9	
49	251.3	B
50	255.8	
51	258.7	
52	263.1	
53	266.3	C
54	278.4	
55	274.8	
56	278.4	
57	282.2	C#
58	285.9	
59	0.0	
60	295.3	
61	299.9	D
62	303.9	
63	308.4	
64	312.3	
65	317.2	D#
66	321.3	
67	326.3	
68	331.8	
69	335.8	E
70	348.4	
71	345.1	
72	358.9	
73	355.9	F
74	361.1	
75	366.3	
76	371.8	
77	377.6	F#

78	382.8	
79	388.8	
80	394.2	
81	399.8	G
82	485.8	
83	411.1	
84	418.3	
85	423.7	G#
86	429.4	
87	435.4	
88	443.1	
89	449.3	A
90	456.8	
91	462.7	
92	467.9	
93	475.8	A#
94	488.4	
95	488.8	
96	495.8	
97	583.7	B
98	589.9	
99	518.5	
100	524.0	
101	533.9	C
102	548.8	
103	547.9	
104	557.8	
105	566.8	C#
106	573.1	
107	581.1	
108	592.3	
109	600.9	D
110	609.6	
111	618.5	
112	625.3	
113	634.3	D#
114	644.4	
115	654.4	
116	664.5	
117	671.7	E
118	682.6	
119	690.2	
120	781.9	
121	714.1	F

122	722.1	
123	734.8	
124	743.6	
125	757.1	F#
126	766.4	
127	776.0	
128	790.8	
129	800.8	G
130	811.1	
131	821.9	
132	838.4	
133	849.8	G#
134	861.5	
135	873.5	
136	886.8	
137	898.7	A
138	911.8	
139	925.4	
140	939.3	
141	953.7	A#
142	961.1	
143	976.1	
144	991.4	
145	1007.9	B
146	1023.8	
147	1040.9	
148	1058.5	
149	1068.3	C
150	1087.0	
151	1095.8	
152	1115.3	
153	1135.9	C#
154	1146.3	
155	1167.4	
156	1198.3	
157	1281.7	D
158	1224.8	
159	1236.9	
160	1249.9	
161	1274.6	D#
162	1288.1	
163	1315.2	
164	1329.7	
165	1343.7	E

166	1372.8	
167	1388.3	
168	1484.1	
169	1436.8	F
170	1452.6	
171	1469.6	
172	1487.0	
173	1523.4	F#
174	1542.1	
175	1561.5	
176	1581.3	
177	1601.5	G
178	1622.3	
179	1643.8	
180	1688.3	
181	1711.2	G#
182	1735.0	
183	1759.7	
184	1784.5	
185	1818.5	A
186	1837.3	
187	1864.4	
188	1892.8	
189	1921.8	A#
190	1921.7	
191	1951.8	
192	1982.7	
193	2814.9	B
194	2847.8	
195	2681.8	
196	2117.3	
197	2153.8	C
198	2191.6	
199	2192.1	
200	2238.5	
201	2271.4	C#
202	2313.1	
203	2357.8	
204	2482.1	
205	2428.1	D
206	2451.5	
207	2588.5	
208	2581.3	
209	2549.2	D#

210	2686.3	
211	2661.3	
212	2663.8	
213	2715.6	E
214	2776.2	
215	2775.8	
216	2839.8	
217	2912.8	F
218	2914.6	
219	2979.8	
220	2980.9	
221	3851.8	F#
222	3123.8	
223	3123.2	
224	3211.1	
225	3213.3	G
226	3298.7	
227	3298.9	
228	3377.6	
229	3470.7	G#
230	3470.4	
231	3569.7	
232	3578.4	
233	3678.2	A
234	3681.4	
235	3785.6	
236	3786.0	
237	3983.6	A#
238	3984.8	
239	3983.9	
240	4029.4	
241	4038.8	B
242	4164.8	
243	4164.5	
244	4389.1	
245	4310.3	C
246	4482.9	
247	4484.9	
248	4489.1	
249	4638.3	C#
250	4636.2	
251	4811.5	
252	4811.6	
253	4808.5	D

254	4996.5
255	4996.4

Bells and the BBC micro

A ring modulator works best with sine waves and produces a fairly accurate output according to the formula we set out above. As the BBC micro does not produce sine waves and, more importantly, as we cannot specify pitch values in hertz, our results will be less than perfect. But, we can still experiment and produce some convincing sounds.

The way to calculate the output frequencies is as follows:

1) Decide upon the input frequencies, say C3 and F3.
2) Look up their respective frequencies in Figure 6.1. In this case they would be 533.9 (P = 101) and 714.1 (P = 121). 3)
3) Find the sum and the difference:

   714.1 + 533.9 = 1248
   714.1 - 533.9 = 180.2
4) Find the nearest P value to these two frequencies. The nearest to 1248 is 160 and the nearest to 180.2 is 26.
5) Try them:

```
10 SOUND1,-15,160,60
20 SOUND2,- 15,26,60
```

It sounds quite good on my micro.

Of course, it is quite time-consuming to go through this process whenever you want a chime; especially if you're writing a piece of music for tubular bells. Sometimes your calculations will produce frequencies which are higher or lower than those available from the sound chip, in which case you will have to try other input values.

The lower frequency will give you the pitch of the note and the upper one will seem like a strong harmonic. You may like to try reducing the volume of the harmonic. You can also try adding one of the original pitches, in this case C3 or E3. Normally, the original frequencies do not appear at the output of the modulator.

You can base your calculations on any two frequencies at all. Intervals over a minor third tend to give the best results. The obvious thing to do is to write a program which allows you to input two pitches and which would output the two nearest P parameters. The programming would be simple but quite long as you would need to include all 256 frequencies. Bell effects are not used very often on the BBC micro, so perhaps you will find the exercise worthwhile.

Fortunately for the more eager among us, an unscientific but workable shortcut is possible. This consists of adding a fixed pitch difference to the melody notes. This will usually produce good results but be careful if the tune spreads over a large range. The next program illustrates this principle.

```
   10 REM PROGRAM 6.9
   20 REM Bells and Chimes
   30
   40 ENVELOPE1,4,0,0,0,0,0,0,126,-1,-1,
-1,126,0
   50
   60 FOR Pitch=1 TO 8
   70 READ Note,Dur
   80 SOUND1,1,Note,Dur
   90 SOUND2,1,Note+120,Dur
  100 NEXT Pitch
  110
  120 SOUND1,1,117,64
  130 SOUND2,1,6,64
  140 SOUND1,1,129,16
  150 SOUND2,1,53,16
  160 END
  170
  180 DATA 117,16,101,16,109,16,81,32
  190 DATA 81,16,109,16,117,16,101,64
```

Try substituting different values for the pitch difference in line 90, and add a pitch difference to other tunes to hear them played by bells or chimes.

Finally, for pure cacophony, experiment along these lines:

```
10 FOR Bell=1 TO 20
20 SOUND1,-15,RND(101),1030 SOUND2,-15,RND(153)+102,10
40 NEXT Bell
```

CHAPTER 7
Zaps and Zings and Other Things

The fourth sound channel, channel 0, which we have barely mentioned so far, is the noise channel. 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 either types of program as we shall see.

It is also capable of producing pitched sounds lower than the pitch of the lowest B (P=1) but we will begin by looking at the parameters and values relevant to channel 0.

The parameters for channel 0 are the same as for the other channels, but the P or pitch parameter has a range of only 0 to 7 and produces a different effect. This was described in Chapter 4, and is repeated in Figure 7.1 for easy reference.

Figure 7 .1

P PARAMETER	EFFECT
0	High frequency periodic noise.
1	Medium frequency periodic noise.
2	Low frequency periodic noise.
3	Periodic noise. Frequency is determined by the P parameter on channel 1.
4	High frequency white noise.
5	Medium frequency white noise.
6	Low frequency white noise.
7	White noise. Frequency is determined by the P parameter on channel 1

The periodic noise is a sort of rasping and the white noise produces a hissing sound. The periodic noise does not sound too dissimilar to that of an ordinary tone but the white noise is clearly not at all musical. Both have their uses in music and in sound effects.

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 amounts of audio frequencies in the same way that white fight is a combination of all colours. If we move up the scale, say one octave from middle C (P=53) to the C above (P= 101), 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 are 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 a bit here and there but we'fl leave that to the synthesists.

There are three ways channel 0 can be used:

1) By itself, with P equal to 0, 1, 2, 4, 5 or 6, which produces a fixed-pitch sound. These can be played one after the other to produce rhythmic effects.
2) With P set to 3 or 7 and in conjunction with channel 1. In these cases, the pitch is dependent upon the P parameter of channel 1. It is varied during the production of a sound, the pitch of channel 0 will vary too.
3) As 1) and 2), but in conjunction with some other sound.

Simple sound effects

Program 7.1 contains six examples of sound effects. You can type in the whole program at once or one section at a time. In some cases, PROCDeIay is a part of the program, in others it is used to separate two effects.

```
10 REM PROGRAM 7.1
20 REM Examples of Channel 0
30 REM Sound Effects
40
```

```
   50 REM Machine Gun
   60 FOR Burst=1 TO 3
   70 FOR Bullet=1 TO 12
   80 SOUND0,-15,5,1
   90 SOUND0,-15,6,1
  100 NEXT Bullet
  110 PROCDelay(100)
  120 NEXT Burst
  130 PROCDelay(100)
  140
  150 REM Ricochet
  160 ENVELOPE1,132,28,-1,0,1,28,0,126,-
8,-3,-6,126,80
  170 ENVELOPE2,6,0,0,0,0,0,0,126,-8,-3,
-6,126,80
  180 FOR Shot=1 TO 3
  190 SOUND2,1,149,20
  200 SOUND0,2,6,20
  210 NEXT Shot
  220 PROCDelay(400)
  230
  240 REM Cymbal or Anvil
  250 ENVELOPE1,3,0,0,0,0,0,0,126,-4,-2,
-4,126,100
  260 ENVELOPE2,3,0,0,0,0,0,0,80,-2,-2,-
2,80,40
  270 FOR Clang=1 TO 8
  280 SOUND&101,1,197,1
  290 SOUND&100,2,4,4
  300 PROCDelay(100)
  310 NEXT Clang
  320 PROCDelay(200)
  330
  340 REM Creature
  350 ENVELOPE1,4,0,0,0,0,0,0,32,-1,0,-4
,126,0
  360 SOUND1,0,220,0
  370 FOR Step=1 TO 8
  380 FOR Splodge=5 TO 7
  390 SOUND0,1,Splodge,Splodge-3
  400 NEXT Splodge
```

```
  410 PROCDelay(200)
  420 NEXT Step
  430
  440 REM Mad Factory
  450 SOUND1,0,100,0
  460 FOR Work=1 TO 16
  470 FOR Noise=1 TO 7
  480 SOUND0,-12,Noise,2
  490 NEXT Noise
  500 NEXT Work
  510 PROCDelay(200)
  520
  530 REM Space Ship Taking Off
  540 ENVELOPE1,40,0,0,0,0,0,0,126,0,0,-
2,126,126
  550 SOUND0,-15,7,150
  560 FOR Engines=180 TO 255 STEP.5
  570 SOUND1,0,Engines,1
  580 NEXT Engines
  590 SOUND0,1,7,100
  600 END
  610
  620 DEF PROCDelay(Time)
  630 TIME=0:REPEAT UNTIL TIME>Time
  640 ENDPROC
```

The Machine Gun

The Machine Gun just alternates between two pitches of white noise. This principle can be used to produce a number of effects. Alter lines 80 and 90 to produce the sound of a helicopter's blade:

80 SOUND0,-15,4,1
90 SOUND0,-15,5,2

Add this line:

95 SOUND0,-15,6,1

to produce a car that doesn't want to start. Increase the D parameter in line 95 to make it even more reluctant.

The Ricochet

The Ricochet is produced by combining white noise from channel 0 with a

sound from another channel undergoing a pitch drop produced by a pitch envelope.

Two envelopes are required for this effect, as we cannot use channel 0 with the pitch drop envelope for the following reason. If you run channel 0 from an envelope containing pitch variations, you can probably guess what happens - the pitch changes run through channel 0 as they would through any other channel, and the output changes in accordance with Figure 7.1. This can be used as yet another source of effects although it is not so easy to use methodically, .especially if channel I is undergoing pitch variations too.

Returning to the Ricochet, two envelopes are also needed to ensure that the volume levels of the respective sounds are correct.

The sound produced by channel 2 alone might be acceptable as a passing seagull.

The Cymbal or Anvil

This is similar to the Ricochet in that it combines white noise with a pitch. The two have been synchronized to ensure a clean percussive start, although in this context the difference would be minute. Again, two envelopes are used, this time purely to control relative volumes.

Synthesisers usually produce a cymbal sound with a little chuff of white noise, sometimes with a ping added as we have done here. The result is usually a very electronic-sounding cymbal. If you reduce the duration of the effect, you will get metallic-like clicks.

The Creature

This is an attempt to produce an organic sound, ie one emanating from a living creature. It is intended to portray a creature with rasping breath walking through wet mud. Obviously, you need some imagination to accept all that but you may find it a useful start for further experiments. Try a backwards envelope to produce a sharp intake of breath and a variation on the Ricochet to produce a wheeze to follow.

The Mad Factory

This was inspired by comedy scenes from old films in which all mechanical devices bang, clang or hiss - often used for a car as it falls to pieces.

The idea is very simple. The loop calls each of the P parameters in turn. Note that channel 1 has been set in line 360 to fix the pitch when P is 3 and 7.

As mentioned earlier, you could replace the loop with a single envelope; I leave it to you to work out the details. Also, alter the pitch of channel 1 and see how it affects the program as a whole.

Sounds are not generally thought of as humorous. Perhaps you can develop this into something more.

Space Ship Taking Off

This illustrates an extreme example of the pitch on channel 1 controlling the pitch on channel 0. The STEP value in line 560 makes the build-up last longer and the, possibly excessive, release stretches the final sound out to around half a minute. This could easily be increased . . .

Exploring the sound channel

As some of the best effects from channel 0 are produced in association with other channels, and with channel 1 in particular, it is not easy to plan a methodical search to discover all its possibilities. Once you know what each P parameter does, however, it will be easier to imagine them in different contexts. You may find the Envelope Comparison program in Chapter 6 useful. If you add an extra line:

185 SOUND0,-15,3,40

you can put channel 0 under envelope control. If you do not want to hear channel 1, you must make sure the volume sections of the envelopes are silent.

Sound Effects Generator program

Although the accent of this book is on understanding the principles involved in creating sounds, many of the effects produced with the pitch envelope and channel 0 (not necessarily together) are difficult to predict. If we go one step further, we can give the computer the task of creating new effects: the results will then be totally unpredictable.

The Sound Effects Generator program produces random sets of envelopes while still giving you control over various aspects of the sound.

```
 10 REM PROGRAM 7.2
 20 REM Sound Effects Generator
 30
 40 *TV255,1
 50 MODE7
 60
 70 PROCSetUp
 80 PROCOptions
 90
100 REPEAT
110 PROCInput
120 REM Cursor Off
130 VDU23;11,0;0;0;0
140 PROCAction
150 PROCPrintEnv
```

```
  160 PROCOptions
  170 REM Cursor On
  180 VDU23;11,255;0;0;0
  190 UNTIL Input$="Q"
  200 END
  210
  220 DEF PROCSetUp
  230 SN$="Both "
  240 Rep$="On "
  250 P=3
  260 Pitch=101
  270 Inp$="EPRSNTAQ"
  280 PROCSetSN
  290 ENDPROC
  300
  310 DEF PROCSetSN
  320 SAA=126:SAD=-1:SAS=0:SAR=-4
  330 SALA=126:SALD=80
  340 NAA=126:NAD=-1:NAS=0:NAR=-4
  350 NALA=126:NALD=80
  360 ENDPROC
  370
  380 DEF PROCOptions
  390
  400 FOR Title=0 TO 1
  410 PRINTTAB(6,Title)CHR$134;CHR$141"S
OUND EFFECTS GENERATOR"
  420 NEXT Title
  430
  440 PRINTTAB(12,3)CHR$133"O P T I O N
S"
  450 PRINTTAB(0,5)CHR$129"E"CHR$130"nve
lope"
  460 PRINTTAB(0,6)CHR$129"P"CHR$130"lay
"
  470 PRINTTAB(0,7)CHR$129"R"CHR$130"epe
at Pitch Envelope"TAB(25)CHR$131;Rep$
  480 PRINTTAB(0,8)CHR$129"S"CHR$130"oun
d or Noise"TAB(25)CHR$131;SN$
  490 PRINTTAB(0,9)CHR$129"N"CHR$130"ois
e Parameter"TAB(25)CHR$131;"P = ";P
```

```
  500 PRINTTAB(0,10)CHR$129"T"CHR$130"Pa
rameter - Alter    "
  510 PRINTTAB(0,11)CHR$129"A"CHR$130"lt
er Pitch"TAB(25)CHR$131;Pitch
  520 PRINTTAB(0,12)CHR$129"Q"CHR$130"ui
t"
  530 ENDPROC
  540
  550 DEF PROCInput
  560 PRINTTAB(26,13)CHR$131"?? ";
  570 REPEAT
  580 Input$=GET$
  590 Action=INSTR(Inp$,Input$)
  600 UNTIL Action>0
  610 PRINTInput$
  620 ENDPROC
  630
  640 DEF PROCAction
  650 ON Action GOTO 660,670,680,690,700
,710,720,730
  660 PROCEnvelope:ENDPROC
  670 PROCPlay:ENDPROC
  680 PROCRepeat:ENDPROC
  690 PROCSoundNoise:ENDPROC
  700 PROCNoiseParam:ENDPROC
  710 PROCTParam:ENDPROC
  720 PROCAlterPitch:ENDPROC
  730 ENDPROC
  740
  750 DEF PROCEnvelope
  760 T1=RND(20):T=T1
  770 PI1=-129+RND(256)
  780 PI2=-129+RND(256)
  790 PI3=-129+RND(256)
  800 PN1=RND(256)-1
  810 PN2=RND(256)-1
  820 PN3=RND(256)-1
  830 PROCPlay
  840 ENDPROC
  850
  860 DEF PROCPlay
```

```
  870 ENVELOPE1,T,PI1,PI2,PI3,PN1,PN2,PN
3,SAA,SAD,SAS,SAR,SALA,SALD
  880 ENVELOPE2,T,0,0,0,0,0,0,NAA,NAD,NA
S,NAR,NALA,NALD
  890 SOUND&11,1,Pitch,254
  900 SOUND&10,2,P,254
  910 ENDPROC
  920
  930 DEF PROCRepeat
  940 IF Rep$="Off" Rep$="On ":T=T1:ENDP
ROC
  950 IF Rep$="On " Rep$="Off":T=T1+128
  960 ENDPROC
  970
  980 DEF PROCSoundNoise
  990 PROCSetSN
 1000 IF SN$="Sound" SN$="Noise":SAA=-12
6:SAD=0:SAS=0:SAR=-126:SALA=0:SALD=0:END
PROC
 1010 IF SN$="Noise" SN$="Both ":ENDPROC
 1020 IF SN$="Both " SN$="Sound":NAA=-12
6:NAD=0:NAS=0:NAR=-126:NALA=0:NALD=0
 1030 ENDPROC
 1040
 1050 DEF PROCNoiseParam
 1060 IF P=3 P=7 ELSE IF P=7 P=3
 1070 ENDPROC
 1080
 1090 DEF PROCTParam
 1100 PRINTTAB(2,10)CHR$133;"Enter T Val
ue 1 - 127"
 1110 REPEAT
 1120 PRINTTAB(24,10)CHR$133;CHR$136;
 1130 INPUT T1
 1140 PRINTTAB(26,10)SPC(8)
 1150 UNTIL T1>0 AND T1<128
 1160 IF Rep$="On ":T=T1 ELSE T=T1+128
 1170 ENDPROC
 1180
 1190 DEF PROCAlterPitch
 1200 PRINTTAB(2,11)CHR$133;"  Enter Pit
```

```
ch 0 - 255     "
 1210 REPEAT
 1220 PRINTTAB(24,11)CHR$133;CHR$136;
 1230 INPUT Pitch
 1240 PRINTTAB(26,11)SPC8
 1250 UNTIL Pitch>-1 AND Pitch<256
 1260 ENDPROC
 1270
 1280 DEF PROCPrintEnv
 1290 PRINTTAB(0,15)CHR$134"Channel 1"
 1300 PRINTTAB(0,17)SPC(17)
 1310 PRINTTAB(0,16)"ENVLOPE1,";T1;",";P
I1;",";PI2;",";PI3;",";PN1;",";PN2;",";P
N3;",";SAA;",";SAD;",";SAS;",";SAR;",";S
ALA;",";SALD
 1320 PRINTTAB(0,19)CHR$134"Channel 0"
 1330 PRINTTAB(0,21)SPC(17)
 1340 PRINTTAB(0,20)"ENVELOPE2,";T;",0,0
,0,0,0,";NAA;",";NAD;",";NAS;",";NAR;"
,";NALA;",";NALD
 1350 ENDPROC
```

How to use the program

When run, the first thing to do is press 'E'. This generates an envelope which will be printed on the screen and sounded at the same time. Every time you press 'E' a new envelope will be generated and the old one will be lost.

'P' enables you to play or repeat the current envelope.

'R' switches the repeat on the pitch envelope on and off.

'S' switches between hearing sound only, noise only or both.

'N' lets you switch the P parameter in the noise channel from 3 (periodic noise) to 7 (white noise).

'T' lets you alter the T parameter, You will be asked for a figure between 1 and 127. Enter the figure and press RETURN. The 'R' option will take care of whether or not the envelope repeats.

'A' lets you alter the pitch. It works in a similar way to the T parameter, only over a 0 to 255 range.

'Q' quits the program.

Program notes

The procedures and variable names should help you understand the program. Imp$ at line 270 contains the only valid letters accepted by

PROCInput. These are tested for at line 590.

PROCsetSN sets the initial amplitude levels for the sound and noise channels. This is called each time a change is made in the Sound/Noise. Both output and the relevant variables are altered in the procedure at fine 990.

PROCEnvelope sets up a random envelope every time it is called. You will notice that T is restricted to a maximum value of 20, but you can alter that if you wish. You will probably find that even values over 10 produce envelopes which are too slow. T1 is used for the initial value of the T parameter and T is finally set according to whether the envelope is to repeat or not.

PROCPlay forms the envelopes. The pitch parameters in the noise envelope are set to 0. If you alter them, alter PROCPrintEny, too. The sounds are programmed with the flush command so that you can play the sound at any time without waiting for the previous sound to terminate .

PROCTParam and PROCAlterPitch are similar and self-explanatory. PROCPrintEnv does just that. If you alter or add anything, be sure to see that this prints the correct envelope.

Suggestions and modifications

You may find it useful to restrict the PI and FN parameters. If you want to produce tuneful sequences you can arrange for the steps to be in multiples of four like this:

```
PI = StartPitch + RND(64)*4
```

You could also arrange to set the PI values in a certain pattern such as up/down/up.

Taking it a step further, you may want to alter individual PI or FN parameters while the program is running, in a similar way to the ADSR and Pitch Graph program. That, however, is putting the task back in your hands, not the computer's, but please do so if you wish. You could also add a procedure to adjust the relative volumes of the two channels.

Instead of having to write down the envelopes, you could arrange for the program to save the current envelope parameters in an array. A separate recall procedure could transfer the parameters of a previously saved envelope back to the current working envelope. In this way you could save more than l6 envelopes. Personally, I find it just as easy to jot them down.

Sometimes you will get a sound you like but which is not quite what you're after. You could include a less severe procedure to alter the current envelope only slightly, say by four points per parameter. Of course, you're just as likely to move away from what ou want as towards it, but at least it's not likely to be something completely different.

Using channel 0 to produce otherwise unobtainable low notes

During your experiments, you will have noticed that the periodic noise on channel 0 sounds very like a tone from one of the other channels. If the P parameter is set to 3, we can alter the pitch of the noise by controlling the pitch of channel 1 like this:

```
10 FOR Pitch=200 TO 1 STEP-4
20 SOUND1 ,0,Pitch,10
30 SOUND0,- 15,3,10
40 NEXT Pitch
```

That *should* sound like a reasonable semitone scale but research indicates that the tie-in between the pitch on channel 0 and the pitch on channel 1 may not be as accurate as we would like. Figure 7 .2 indicates the pitch values of channel 1 which are necessary to produce the indicated pitch values on channel 0. These carry on downwards from the octaves fisted in Figure 3.3.

Figure 7.2

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	144	148	152	156	160	164	168	172	176	180	183	188
-1	96	100	104	108	112	116	120	124	128	132	136	140

The relationship between these two pitch values on your computer may not be the same but should only deviate by one figure. The only anomaly in the table appears to be the A#1 but check this on your own computer. You can do an ear test by playing the relative pitch along with the same note an octave higher on channel 2. The notes produced by channel 0 may not be exact octaves of those of the other channels but they will be near enough to be of use in a composition.

You will realise that when the pitch drops' to octave -1, it starts to sound more like a buzz although it is arguably useful right down to octave C-1.

Use of the lower octave

All music is written in a specific pitch and key and this method allows us to use one or two octaves of sound not normally available. Many tunes were written in and sound better in the lower octaves and these lower notes can be especially useful if the music range exceeds the five octave range of the sound chip. The only loss is any possible use of channel 0 as a rhythm accompaniment which brings us to that very topic.

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, lies in producing acceptable drum sounds. The best drum noises seem to come from channel 0 with P set to 4, 5 or 6 - the white noise settings - but do try other settings and use it in conjunction with channel 1 as described above.

You will find that duration of the sound plays a very important part in determining the drum characteristics and, if you use more than one envelope, you can vary the sound and produce quite a reasonable rhythm unit.

The next program is one way of approaching the design of a rhythm generating program. Be wary of renumbering the program because the variable, Rhythm, holds the starting line of the DATA statements containing the rhythms.

```
   10 REM PROGRAM 7.3
   20 REM Rhythm Unit
   30
   40 *TV255,1
   50 MODE7
   70 PROCSetUp
   80 PROCTitlePage
   90 PROCPlay
  100 END
  110
  120 DEF PROCSetUp
  130 Tempo=2
  140 Rhythm=360
  150 ENVELOPE1,1,0,0,0,0,0,0,126,-3,-3,
-6,126,100
  160 ENVELOPE2,1,0,0,0,0,0,0,32,-4,-4,-
8,110,60
  170
  180 DEF PROCTitlePage
  190 FOR Title=1 TO 2
  200 PRINTTAB(6,Title)CHR$141;CHR$134"R
 H Y T H M   U N I T"
  210 NEXT Title
  220 PRINTTAB(1,4)CHR$129;"1";CHR$130;"
Bossa Nova"
  230 PRINTTAB(1,5)CHR$129;"2";CHR$130;"
```

```
Rock"
  240 PRINTTAB(1,6)CHR$129;"3";CHR$130;"
12/8 Rock"
  250 PRINTTAB(1,7)CHR$129;"4";CHR$130;"
Swing"
  260 PRINTTAB(1,8)CHR$129;"5";CHR$130;"
Waltz"
  270 PRINTTAB(1,9)CHR$129;"Q";CHR$130;"
Quit"
  280 ENDPROC
  290
  300 DEF PROCPlay
  310 REPEAT
  320 *FX15,1
  330 IF INKEY(-49) Rhythm=450
  340 IF INKEY(-50) Rhythm=480
  350 IF INKEY(-18) Rhythm=520
  360 IF INKEY(-19) Rhythm=550
  370 IF INKEY(-20) Rhythm=580
  380 READ Env,Pit,Dur
  390 SOUND0,Env,Pit,Dur*Tempo
  400 IF Env=0 RESTORE Rhythm
  410 UNTIL INKEY(-17)
  420 ENDPROC
  430
  440 REM Bossa Nova
  450 DATA 1,4,2,2,5,2,2,5,2,1,4,2,2,5,2
,2,5,2,1,4,2,2,5,2,0,0,0
  460
  470 REM Rock
  480 DATA 1,5,2,2,5,2,1,4,2,1,4,1,1,5,2
,2,5,2,2,5,1,1,4,1,2,4,1,2,4,2
  490 DATA 1,5,2,2,5,2,1,4,2,1,4,1,1,5,2
,2,5,2,2,5,1,1,4,1,2,5,1,1,6,1,1,6,1,0,0
,0
  500
  510 REM 12/8 Rock
  520 DATA 1,5,2,2,4,1,2,4,1,2,4,2,1,4,2
,2,4,2,2,4,2,1,5,2,2,4,1,2,4,1,2,4,2,1,4
,2,2,4,2,1,6,2,0,0,0
  530
```

```
  540 REM Swing
  550 DATA 1,5,4,2,4,3,2,4,1,1,5,4,2,4,3
,2,4,1,1,5,4,2,4,3,2,4,1,1,6,2,2,4,2,2,4
,1,2,4,1,2,4,2,0,0,0
  560
  570 REM Waltz
  580 DATA 1,5,4,2,4,4,2,4,4,1,6,4,2,4,4
,2,4,4,1,5,4,2,4,4,2,4,4,1,6,4,2,4,2,2,4
,2,2,5,2,1,5,2,0,0,0
```

When run, the program will play a Bossa Nova: pressing the indicated keys will alter the rhythm.

Program notes

The program should be self-explanatory. As we are only using one channel, the rhythms are produced by alternating between various shades of white noise. Lines 380 to 400 do this by reading information from the DATA statements.

The envelopes in PROCSetUp play a large part in determining the drum sound. Only two have been used but you can add more to produce different instruments. For example, a very short duration will sound like a wood block and a medium-pitched noise with just a little decay will sound like a hand clap. You may wish to bring another channel in, toq. Experiment with the original envelopes to see if you can improve on the sound. 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. Alter the tempo in line 130, too.

The INKEY function with a negative number in brackets reads a particular key on the keyboard (see the User Guide page 275) to see if it is being pressed. From this, the program sets the next RESTORE operation to the required line.

The CAPS LOCK and SHIFT LOCK lights and the ADVAL function

You will notice, if you have a 1.0 or later operating system, that the CAPS LOCK and SHIFT LOCK lights go on when the program is running. These light whenever the buffers fill up and indicate that the program is being held up waiting for the sound queue to clear. In this program, it does not really matter but if we wanted to do something else, such as play along with the rhythm, we may have a problem or two. One way around it is to replace lines 380, 390 and 400 by this single line:

```
380 IF ADVAL(-5)>0 READ Env,Pit,Dur:
SOUND0,ENV,Pit,Dur*Tempo:IF Env=0 RESTORE Rhythm
```

This ADVAL function with a negative parameter in brackets returns the number of free spaces in the channel 0 buffer. If there is a free space, it reads the next data item and processes the next sound command. ADVAL is explained on page 202 of the User Guide and we will look at it in more detail in Chapters 8 and 9.

You will notice that the lights no longer stay on because the program does not come to a halt at line 390 when the buffer is full. Instead, it continually cycles through the REPEAT loop. This means we could put other commands here, such as information for more sounds, without disturbing the rhythm - provided the new information does not hold up the program either.

Using sound effects in utility programs

While the zap of a laser blast might be considered disturbing in a utility program, sound effects can be used to enhance such programs, making them more interesting to work with and generally assisting the user.

Many people, myself included, do not like being beeped at by a computer when we make a wrong input. The most common sound on the BBC micro used for such a purpose can be made by holding down the CTRL key and pressing 'G'. It can be written into a program simply as:

```
VDU 7
```

which has the same effect.

The presentation of a program, its ease of operation and its error-trapping techniques all determine how user-friendly it is. Detailed discussions on this topic, however interesting, are beyond the scope of this book but we can look at one or two ways of improving a program through the use of sound.

If using a program is a matter of hitting a certain number of keys, eg selecting items from a menu, you could incorporate a procedure to make a small sound whenever an item is correctly selected. Something like this:

```
1000 DEF PROCSound
1010 SOUNDO,-12,4,2
1020 ENDPROC
```

This tends to give the keys a chunky feel; something a bit more solid than the tap or rattle of the keyboard. The user also knows that he or she has hit a correct key and can expect some action from it.

In place of the beep, you could use a different sound to indicate an

incorrect selection - may I suggest something a little more quiet and less presumptuous.

Mistakes are generally made because the user does not understand the instructions and nothing is more frustrating than to press a key you think is right and receive only *a beep in response. Action should be directed to another part of the program to give the required information or back to the instruction sheet.

If the inputs are restricted and the program is correctly error-trapped (as it should be) and will not respond to an invalid input there may not be a need for any sound at all on an incorrect key press. This is the opposite of the normal method of operation - silence on a correct entry - and preferable in many instances.

If a program steps through a number of sections which cumulate in a result, eg a character generating program, you could incorporate a small fanfare, such as in the Motility Tester program in Chapter 1, to indicate that a section of the program has been completed and the user is about to move on to the next step. Such fanfares can be constructed from pitch envelopes.

The above suggestions can make utility programs more interesting to work with, as long as they work towards the aim of the program and are not there simply for decoration.

In such programs, it is always advisable to give the user the option of switching the sound off so that they do not have to listen to your creations, however marvellous they may be, if they do not want to.

Soundscapes: a total sound effects program

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, the jungle, etc. 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 chapter to form a sea soundscape.

```
   10 REM PROGRAM 7.4
   20 REM Sea, Surf & Seagulls
   30
   40 ENVELOPE1,130,28,-1,0,1,28,0,63,-8
,-4,-6,126,80
   50 ENVELOPE2,2,0,0,0,0,0,0,63,-1,0,-4
,126,116
   60 Foghorn=0
   70
   80 REPEAT
```

```
   90 Foghorn=Foghorn+1
  100 Pitch=RND(64)+190
  110 IF ADVAL(-6)>0 SOUND1,0,Pitch,0
  120
  130 Pulse=RND(20)+15
  140 Lull1=RND(10)+25
  150 Lull2=RND(10)+25
  160 Flow=RND(6)+8
  170 Ebb1=RND(2)+3
  180 Force=RND(26)+100
  190
  200 Wave=RND(3)+3
  210 Ebb2=RND(5)^3+129
  220
  230 REM Envelope for Waves
  240 ENVELOPE3,Pulse,0,1,0,Lull1,1,Lull
2,Flow,-Ebb1,-1,-1,Force,90
  250
  260 IF ADVAL(-5)>3 SOUND0,3,Wave,Ebb2:
SOUND&1000,0,0,Ebb2
  270
  280 REM Seagull
  290 IF ADVAL(-7)>0:IF RND(80)=1 SOUND2
,1,RND(76)+101,20
  300
  310 REM Foghorn
  320 IF ADVAL(-8)>0:IF Foghorn>140 SOUN
D&113,2,0,80:SOUND&112,2,0,80:IF Foghorn
>140 Foghorn=0
  330 UNTIL FALSE
```

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

Program notes

The variables in lines 130 to 180 repeatedly redefine ENVELOPE 3 in fine 240.

Wave in line 200 selects white noise with a P value of from 4 to 6. The PI2 parameter of the envelope increases the pitch parameter on channel 0

by one, so a complete wave will use two white noise types. These will be 4 and 5, 5 and 6, 6 and 7. As 7 is based on the pitch of channel 1 this is selected randomly in fine 100. Ebb2 determines how quickly one wave will finish and another begin. Notice the use of a dummy parameter in the second sound command in line 260 which lets the release phase occur and the wave ebb.

The Foghorn variable is increased automatically so the foghorn sounds regularly. Inclusion of the flush parameter will cut off a seagull in mid-call.

The seagull is produced by a variation on our Ricochet envelope in Program 7.1.

The use of ADVAL functions ensure that the program never sticks and, by using similar methods, you can add even more to it. If you add:

```
315 PRINT Foghorn
```

you will see that the program is not held up by sound queues. If you then remove the ADVAL statement in line 260:

```
260 SOUND0,3,Wave,Ebb2:SOUND&1000,0,0,Ebb2
```

you will see how the program grinds to a halt.

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. See what you can do.

CHAPTER 8
Playing the BBC Micro

The introduction of sound chips into personal computers brought about the birth of a totally new musical instrument - the computer itself. How easy and effective it is to play depends both upon the hardware and the software used to drive it. The BBC micro excels in both departments and we can use it to perform some quite complex musical feats.

Using the BBC micro as a musical keyboard

Most musical instruments are designed to be ergonomically easy to play - within the confines of the shape required by the instrument to produce whatever sound it is supposed to produce. A piano-type keyboard is probably one of the best examples, although designs exist for other keyboards which are intended to be easier to play.

Computers, unless they are specifically designed to operate as a musical instrument, are not normally supplied with a musical keyboard and if we want to play the computer we must make do with what we have, ie the QWERTY typewriter keyboard.

Depending upon your musical upbringing, you may find this easy or difficult to adapt to. The QWERTY keys are not laid out like a piano keyboard and are not labelled to correspond to musical notes. It may well be that here the non-musician has a distinct advantage over the keyboard player.

If you can play a piano keyboard your playing will tend to be partly automatic and, after a little practice, your fingers know how to move in order to play a certain sequence of notes. Much the same applies to the typist who is used to the QWERTY keyboard but in this case the fingers are responding to different patterns, ie word patterns, not musical ones.

Musicians and non-musicians alike will find that a little practice greatly improves their skill in using the QWERTY keyboard as a musical instrument but it will still be difficult to play anything of any technical difficulty. We can, however, still 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, eg flute, trumpet, violin: 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 should this be required) such as the piano, organ, harp, etc.

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 BBC as a monophonic keyboard

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: it turns the computer into a monophonic keyboard.

```
   10 REM PROGRAM 8.1
   20 REM Monophonic Keyboard
   30 REM From G (Pitch=33)
   40 REM  To  E (Pitch=117)
   50
   60 ON ERROR GOTO150
   70
   80 Keyboard$="Q2W3ER5T6YU8I9O0P^[\_"
   90 *FX11,1
  100 REPEAT
  110 Key$=INKEY$(0)
  120 Pitch=29+4*INSTR(Keyboard$,Key$)
  130 IF Key$<>"":IF Pitch>29 SOUND&11,-
15,Pitch,2
  140 UNTIL FALSE
  150 *FX12,0
```

The program is so simple it could probably be condensed into a couple of fines. 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 G (P=33) to E (P=117).

Program notes

Input is detected by INKEY$ and checked with the INSTR function to see if it corresponds to a key in Keyboard$. The basis for pitch calculation is 29 and the pitch is increased in multiples of four, according to the position of the pressed key in Keyboard$ as determined by the INSTR function. This is done in line 120. If no key is pressed the INSTR function returns a value of 0, and if Pitch is no more than 29 the sound will not occur.

The SOUND command contains a flush instruction so each new note sounds immediately upon being received. *FX11,1 removes the auto delay.

The program continually cycles through the REPEAT loop and sends a continuous series of instructions to the SOUND command. The duration value of 2 is necessary to prolong the note for the length of time the BASIC program takes to work through the loop. Reduce it to 1 and hear what happens .

If you try to use envelope control, you will find that the cycling will sometimes cause the note to continually repeat. A case in point would be this envelope:

ENVELOPE1,1 ,0,0,0 ,0,0,0,126,-4,0,-4,126,80

You may find the idea useful for mandolin or banjo effects.

Keyboard display program

When playing a strange instrument it is often helpful to have a diagram of the keyboard with the relevant buttons or keys marked on. The next program draws such a display in mode 7 graphics and can be appended to any of these programs to provide a visual display and aid as you play the keyboard. You may also find it helpful to stick small pieces of paper over the keys you don't play to make the QWERTY layout look a little more like a piano keyboard.

```
 1000 REM PROGRAM 8.2
 1010 REM Keyboard Display
 1020
 1030 DEF PROCKeyBoard
 1040 CLS
 1050 FOR Title=1 TO 2
 1060 PRINTTAB(9,1+Title)CHR$141;CHR$132
;"K E Y B O A R D"
 1070 NEXT Title
 1080
 1090 PRINT'" f1   f2   f3   f4"
```

```
 1100 PRINT"ENV1 ENV2 ENV3 ENV4"
 1110 PRINT''"  2  3     5  6     8  9
0     ^ |"
 1120
 1130 K$=CHR$156+CHR$151+CHR$157
 1140 J$=CHR$148+CHR$181+" "
 1150
 1160 FOR Key=1 TO 4
 1170 PRINTCHR$151;CHR$255;
 1180 PRINTK$;K$;J$;K$;K$;J$;
 1190 PRINTK$;K$;K$;J$;K$;K$;CHR$156
 1200 NEXT Key
 1210
 1220 FOR Key=1 TO 5
 1230 PRINTCHR$151;CHR$157;CHR$148;
 1240 PRINTSTRING$(12,CHR$181+" "+" ")
 1250 NEXT Key
 1260
 1270 PRINT" Q  W  E  R  T  Y  U  I  O
P  @  [  _"
 1280 ENDPROC
```

The procedure assumes the computer is in mode 7 - do not change mode inside the procedure. Insert a line containing:

PROCKeyBoard

near the beginning of the program. See Appendix 2 and the User Guide page 402 for tips on merging programs.

The display is made up from teletext characters and is relatively straightforward although it may not be easy at first to see what the various strings produce. See Chapter 28, page 150, of the User Guide for more information on the use of teletext graphics.

Lines 1090 and 1100 are there for the benefit of the next program.

Alternative methods of note production

If we want to use envelope control or if we want to play more than one note at once, we need to employ a slightly more sophisticated method of key detection. One such method involves the use of the negative INKEY function which we used in Program 7 .3, the Rhythm Unit Program.The advantage of the negative INKEY function is that it only tests for one

particular key and any number can be used together so we can test for three simultaneous key depressions. (See page 273 of the User Guide for further information.)

The next program allows us to play up to three notes at once to produce chords (see Chapter 2).

```
 10 REM PROGRAM 8.3
 20 REM 3-Note Polyphonic
 30 REM Keyboard (Q - _)
 40 REM From G (Pitch=81)
 50 REM To   E (Pitch=165)
 60
 70 DIM KBoard%(26)
 80 DIM KFlag%(26)
 90 DIM CH%(3)
100
110 FOR Channel=1 TO 3
120 CH%(Channel)=0
130 NEXT Channel
140
150 FOR Keys=1 TO 26
160 READ Data
170 KBoard%(Keys)=Data
180 KFlag%(Keys)=-1
190 NEXT Keys
200
210 DATA 17,50,34,18,35,52,20,36,53
220 DATA 69,54,22,38,39,55,40,56,72
230 DATA 25,57,121,41,114,115,116,21
240
250 REM f1=ENVELOPE1:f2=ENVELOPE2
260 REM f3=ENVELOPE3:f4=ENVELOPE4
270
280 ENVELOPE1,1,0,0,0,0,0,0,126,-4,-4,
-4,126,100
290 ENVELOPE2,129,12,0,-4,1,0,3,126,-1
,0,-4,126,100
300 ENVELOPE3,1,0,1,-1,0,1,1,126,-1,0,
-4,126,100
310 ENVELOPE4,8,0,0,0,0,0,0,63,10,0,-6
3,63,126
```

```
  320
  330 Pitch%=77
  340 E%=2
  350
  360 KPressed%=0
  370 REPEAT
  380 FOR N%=1 TO 26
  390 IF INKEY(-(KBoard%(N%)))=KFlag%(N%
) PROCP
  400 NEXT N%
  410 UNTIL FALSE
  420 END
  430
  440 DEF PROCP
  450 IF N%>22 PROCE:ENDPROC
  460 IF KPressed%=3 AND KFlag%(N%) ENDP
ROC
  470 Chan%=0
  480 IF KFlag%(N%) REPEAT Chan%=Chan%+1
:UNTIL CH%(Chan%)=0:CH%(Chan%)=N%:SOUND&
10+Chan%,E%,Pitch%+N%*4,255:KPressed%=KP
ressed%+1
  490 IF NOT KFlag%(N%) REPEAT Chan%=Cha
n%+1:UNTIL CH%(Chan%)=N%:CH%(Chan%)=0:SO
UND&1010+Chan%,0,0,0:KPressed%=KPressed%
-1
  500 KFlag%(N%)=NOT KFlag%(N%)
  510 ENDPROC
  520
  530 DEF PROCE
  540 E%=N%-22
  550 ENDPROC
```

The keys used are the same as in the monophonic program and pressing function keys f1 to f4 will put the keyboard under control of that envelope number. Program 8.2 produces a suitable display to go with this program.

Program notes

There is more than one way in which this program could have been written. One way, the brute force method, would be to include 26 fines such as:

```
390 IF KeysPressed>3 ENDPROC ELSE IF INKEY (-17)
Note=33:PROCSound:KeysPressed:KeysPressed+1
```

At least such a method would be quite easy to understand. A more sophisticated method is also doubtless possible but at the expense of comprehension. This program tries to tread a middle path. Once you understand the principles involved you can experiment and write your own - as simple or as sophisticated as you wish. To minimise the time taken by the program to interpret the BASIC code, integer variables and short variable names have been used.

The way the program works is described first, followed by individual sections and aspects which may need clarification.

We have substituted the 26 possible lines mentioned above by an array, KBoard%, which contains the negative INKEY values of the keys we want the program to respond to. A second array, KFlag%, keeps track of whether a key is currently pressed or not. The array, CH%,, keeps track of which channel is being used to produce which note. Pitch% sets the basic root pitch and E% is the envelope number.

The REPEAT loop between lines 370 and 410 cycles through the 26 negative INKEY values in the KBoard% array. The KFlag% array checks to see if there has been any change in keys pressed since the last loop and if there has the program is diverted to PROCP.

N% refers to how far up the scale we are. If N% equals 23, 24, 25 or 26, one of the function keys is being pressed and the program is diverted to PROCE which simply sets E% to a new envelope number. If N% is less than 23 it means a note is required.

As we can't sound more than three notes at once, the program checks, in line 460, to see how many keys are currently pressed. If there are already three keys down and another key has been pressed, control is immediately passed back to the REPEAT loop. (See the User Guide pages 89 and 100 for more information about the use of TRUE and FALSE.)

If control gets to fine 480, there is an empty channel and a key has been pressed telling the program to make a sound. Chan% is incremented by 1 until it finds an empty channel. This is given the value of N%o which tells the program which key enabled that particular channel and it plays a sound at the required pitch. KPressed%, is also incremented to keep track of how many keys are down. Lastly, the KFlag% variable is changed in line 500. If KFlag%(N%) was TRUE, ie pressed down, it is set to FALSE. The next time the loop looks at this value of N% in line 390 it will be looking to see if the key has been lifted.

If a key has been lifted, control passes to line 490 instead of 480. Chan'% is incremented until the program finds which channel was responsible for the sound produced by the key which has just been lifted. When the channel has been found, it is flushed. Notice the use of the dummy note parameter to allow the release phase to occur.

KPressed% is decremented to show that a sound channel has been freed. KFlag%(N%) is changed again by line 500 to TRUE.

How the program works and making modifications

The use of KFlag% ensures that a channel is not given a sound request until the key responsible for the present sound on that channel is lifted. This prevents a stream of continuous information going to the sound channels as in the monophonic program, and it permits envelope controL.

The sound command in line 480 which produces the sound has been given an infinite duration. If the AS phase of the envelope is 0, the note will continue until you take your finger off the key. The channel will then be flushed the next time around the loop by line 490.

You will notice that we have not needed to use an ADVAL function to check if the sound channels are full because, if a channel is being used, the program does not allow any further commands to get through.

The use of Kpressed% in line 460 to terminate the procedure if too many keys are down can be altered. As it is, if three keys are down and you press another, control just passes back to the REPEAT loop. Some synthesisers have a high, a low or a last note priority which means that the three highest, lowest or the three last notes take precedence over any others. You can achieve this by altering the '3 keys down' criteria in this fine and you will probably also have to alter the channel allocation, according to your aims.

If you want to try some dazzling fingerwork, you may find the response a little slow; and you may notice a very small time lag between the notes of a chord if you press three keys exactly together. You will see that the program does not attempt to synchronize any notes. This is a result of the program design and the fact that each note has to run through a lot of BASIC programming before it is heard. This tiny delay is not likely to be a problem but you can cut the response time by compressing the coding, using multi-statement fines and single letter integer variables.

As a modification, if you wanted to synchronize the notes, instead of calling PROCP with every note pressed you could call a PROCGetNotes to count the keys pressed and work out the pitch values. At the end of the FOR . . . NEXT loop you could call a PROCPlay which would carry out the information gathered by PROCGetNotes and synchronize the notes if necessary. The same effect could probably be put into PROCP as it stands with a little clever coding.

You can add more commands via the negative INKEY function. These can be used to increase the range of the keyboard and to access more envelopes. Detection of, say, the SHIFT key could increase the pitch by an octave. Include the relevant key numbers (User Guide page 275) in the DATA statements following line 230. The arrays, initiating loops and the repeating FOR . . . NEXT loop will need to be altered, too.

Further additions to the musical keyboard: 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.

As most synthesisers control pitch by voltage (the higher the voltage the higher the pitch) 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 additions and alterations necessary to Program 8.3.

```
   10 REM PROGRAM 8.4
   20 REM Bass Sequencer with
   30 REM Duophonic Keyboard
   40 REM Alter 460: Add 345, 375 &
   50 REM 560 plus in PROGRAM 8.3
   60
  345 Env%=1
  375 PROCBass
  560 DEF PROCBass
  570 IF ADVAL(-8)<1 ENDPROC
  580 READ P%,D%:IF P%=-1 RESTORE 620:RE
AD P%,D%
  590 SOUND3,Env%,P%,D%
  600 ENDPROC
  610
  620 DATA 33,4,33,4,45,4,53,4,-1,-1
```

Program notes

Env% at line 345 selects the envelope the sequencer will use. PROCBass is inserted in the REPEAT loop of the main program. As one channel is taken up with the sequence, we can only use two channels now, so fine 460 is altered to take care of that.

The sequencer uses ADVAL(-8) to keep it supplied with notes and to ensure that the program does not grind to a halt. It also allows us to interrupt the main loop with calls to different envelopes and instructions to channels 1 and 2 without disturbing the sequence.

Altering the bass riff

The bass riff in the program is very simple but it can be extended to any length by adding more data. If you alter the pitch data, P%, to intervals instead of notes and add a BassPitch% variable such as:

```
346 BassPitch%=33
590 SOUND3,Env%,BassPitch%+P%,D%
620 DATA 0,4,0,4,12,4,20,4,-1,-1
```

you can alter the pitch of the sequence by changing BassPitch% during the program. Remember to alter all the loop lengths to accommodate the extra keys.

Due to the queuing of the commands on channel 3, the stored notes at the old Basspitch% will play before changing. To overcome this you could reduce the number of queued notes by raising the number of free spaces in the buffer as detected by the ADVAL statement at line 570, only changing to the new BassPitch%, when the data is RESTOREd.

You could program the sequencer to play the complete bass fine of a tune and play along with it and you could also add some rhythm to the sequencer like this:

```
565 IF ADVAL(5)>0 SOUND0,Env%,4,2
```

If the bass sequence is now louder than the keyboard, give the drums a separate, quieter envelope

It should not be too difficult to combine this program with the Rhythm Unit program in Chapter 7 and to incorporate also the envelopes and effects illustrated in the last two or three chapters. Here are some more bass lines to try:

```
REM Key G Minor
620 DATA 33,4,81,4,49,4,53,4,57,4,61,4,13,4,61,4,-1,1

REM Key G Minor
620 DATA 33,4,45,4,65,4,61,4,-1,-1

REM Key G Major
620 DATA 33,8,49 ,2,61,2,81,4,73,4,61,4,73,4,89,4,-1,-1
```

```
REM Key A Minor/G Major
620 DATA 41,4,49 ,4,53,4,33,4,49,4,61,4,-1,-1

REM Key C Minorish!: Set Env%=4 or Similar
620 DATA 5,16,17,16,13,16,9,16,-1,-1
```

Developing the sequencer

Apart from adding more envelopes, rhythms, sequences and effects, you might like to put a memory capability into the program so that it remembers what you have played and plays 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 which 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 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 a worthwhile exercise.

If you want to control and alter music to such a fine degree, the difficulty in 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. If you have tried to play a three-part or even a two-part tune with any degree of accuracy, you will be aware of the restrictions the keyboard places upon us. It's fine for improvising and playing along with a rhythm or bass pattern, but playing true multi-part tunes is extremely difficult.

We can overcome these problems by providing the computer with all necessary information regarding pitch, envelopes, durations, etc, and letting it get on with the hard work of putting them together. It can play sequences and harmonies we would never be able to imagine 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: this method is used in most of the programs in this book. For short musical examples this is fine, but it soon becomes quite complicated when you have more than 20 or 30 items of data. This is where Program 3.1 can help by converting note names and octave numbers into pitch values. This permits us to enter data in a way more comprehensible to us and it makes subsequent editing considerably easier. If you include error routines it will also report instances of incorrect data.

Another problem arises if we want more than one channel to sound at once, namely that of synchronization. If we just want to play a single melody line there are no problems. This short program will do exactly that:

```
10 FOR Note=1 TO NumberOfNotes
20 READ Env,Pitch,Dur
30 SOUND1,Env,Pitch,Dur
40 NEXT Note
50 DATA E1,P1,D2,E2,P2,D2,E3,P3 etc
```

If we want to play more than one channel at a time we need to use a slightly different method. We mentioned this in earlier chapters and now we will examine it in detail.

Playing two- or three-part tunes

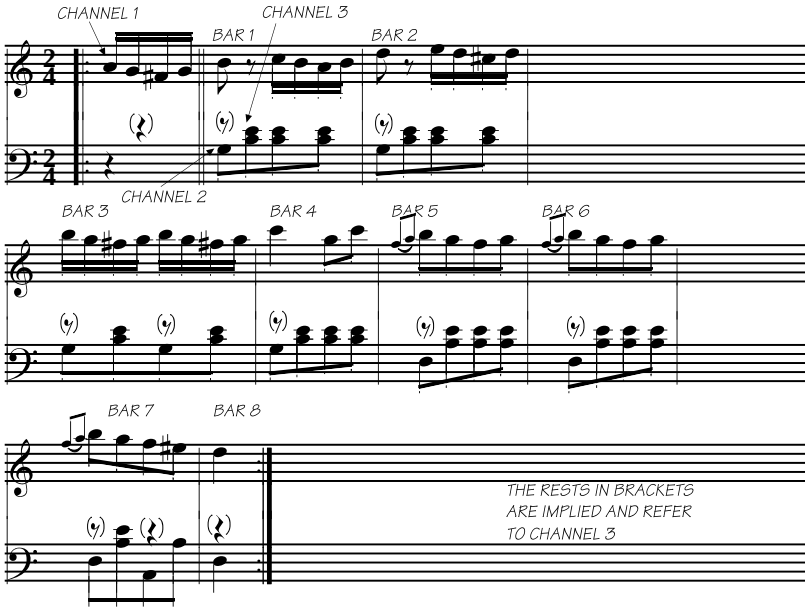
Because of the sound queues used by the BBC micro, the sound channels seem to race ahead of the rest of the program. We found this in Program 6.5 where we had to hold up the program with a TIME loop before redefining the envelope.

The optional synchronization parameter of the SOUND command can be used to delay a sound channel until another channel appears with the same synchronization value. However, the organisation of information to each channel must be carefully controlled if the program is not to seize up.

Problems arise when two or three voices are to be played together and each voice contains notes of different durations. If we try to read data for two channels from one DATA statement, each time the data is read it must supply information for both channels. This means we would have to organise the data so that the two sets of notes end up with the same number of data items. This is a time-consuming and inconvenient process. With three-part tunes the problem is even worse as one channel will probably only be playing one note to the others' four or more.

The most obvious answer is to fill arrays, one for each channel, with the relevant data, and only read from a particular array either when a note is required or when the sound channel is capable of taking it without holding up the program.

Figure 9.1



As an example, Figure 9.1 shows the first eight bars of Mozart's Rondo Alla Turca. You can see that, by the time we reach the end of bar 3, 24 notes will have passed through channel 1 and only eight through channel 3. If these were queued together, the program would be held up waiting for the notes on channel 3 to execute. We can solve this in two ways:

(1) By keeping track of the relative durations of each channel and releasing data accordingly.

Or

(2) By only releasing data when the channels can take it at a time determined by the negative ADVAL function.

Before beginning our experiments, let us see how we can best organise the data.

Selecting the notes and octave range

The programs presented here read note information from DATA statements. This gives us immediate visual and physical access to the information which is saved along with the program.

When programming a musical score, the first thing we must look at is the range of notes it uses. This example ranges from B an octave below middle C to C two octaves above middle C. This is fortunate in that we can use the correct notes and octaves produced by the sound chip and illustrated on the keyboard in Figure 2.4. It does mean that we need to transpose or mentally shift the keyboard up an octave to make middle C correspond to the middle C on the stave. This is not as complicated as it might seem. If you enter the notes as names and octave numbers, read down from the stave to the note name and octave number and, if you want it lower, subtract 1 from the octave number.

An alternative, as we mentioned in Chapter 3, is to enter the notes exactly as they read, eg middle C as 101, and use the variable, Key, to take the piece down an octave. Using this method, of course, you can play the tune in any key at all. I decided on the first method.

The second eight bars of Rondo Alla Turca (not illustrated) uses A an octave below middle C. In this case you would use the keyboard and octave numbers exactly as set out in Figure 2.4, an octave higher than written.

Another solution is to 'cheat' by substituting a note within your range for the A. If only one or two notes are outside the range this will usually produce acceptable results and I prefer to do this rather than take the whole tune up an octave. Musically (and, perhaps, sardonically) this is known as doing an arrangement.

I have stated before my preference for the lower octaves. I tend to program as low as possible. These notes are richer and more full than the

high ones, but use whichever range suits you and the music best. Not all pieces will benefit by being so low.

I normally arrange the channels as follows:

Channel 1: Melody line.
Channel 2: Bass line.
Channel 3: Anything in between, often reinforcing the bass/ accompaniment

The melody and bass fines are the most important and set the character of the piece. I use channel 3 to fill in the harmony where required, and it can switch from bass to melody as necessary. If the piece has a prominent bass fine, I may sometimes program that first into channel 1.

The next program uses only one channel to play a melody. You may prefer not to type it in and use the listing only as a reference as we will be moving on to a more sophisticated program. However, as we will be using the data and PROCAnalyseNote in the other program, if you type it in, it will not be wasted.

This raises another point worthy of consideration, which is - the programs need only be as complicated and sophisticated as you require. If you only want a monophonic tune, this program will do fine.

```
   10 REM PROGRAM 9.1
   20 REM 1 Channel Version of Mozart's
   30 REM Rondo Alla Turca
   40
   50 Scale$="  C  C# D  D# E  F  F# G
G# A  A# B"
   60 Key=1
   70
   80 ENVELOPE1,1,0,0,0,0,0,0,126,-2,0,-
10,126,100
   90 CurrentEnv=1
  100
  110 FOR N=1 TO 46
  120 READ Note$,Dur
  130 PROCPlayNote
  140 NEXT N
  150
  160 END
  170
  180 DATA B2,2,A2,2,G#2,2,A2,2
  190 DATA C3,4,R,4,D3,2,C3,2,B2,2,C3,2
  200 DATA E3,4,R,4,F3,2,E3,2,D#3,2,E3,2
```

```
  210 DATA B3,2,A3,2,G#3,2,A3,2,B3,2,A3,
2,G#3,2,A3,2
  220 DATA C4,8,A3,4,C4,2,B3,1,A3,1
  230 DATA B3,4,A3,4,G3,4,A3,2,G3,1,A3,1
  240 DATA B3,4,A3,4,G3,4,A3,2,G3,1,A3,1
  250 DATA B3,4,A3,4,G3,4,F#3,4
  260 DATA E3,8
  270
  280 DEF PROCPlayNote
  290 PROCAnalyseNote
  300 PRINT Note$,Pitch,Octave
  310 SOUND1,Env,Pitch,Dur
  320 ENDPROC
  330
  340 DEF PROCAnalyseNote
  350 IF Note$="R" Env=0:ENDPROC ELSE En
v=CurrentEnv
  360 IF LEN(Note$)<2 OR LEN(Note$)>3 TH
EN PRINT"ERROR IN DATA ";Note$:PRINT"Not
e Number ";N:STOP
  370 IF LEN(Note$)=2 THEN NoteName$=LEF
T$(Note$,1) ELSE NoteName$=LEFT$(Note$,2
)
  380 Octave=VAL(RIGHT$(Note$,1))
  390 Pitch=Key+INSTR(Scale$,NoteName$)/
3*4+(Octave-1)*48
  400 IF Pitch<0 OR Pitch>255 THEN PRINT
"ERROR IN PITCH DATA ";Note$;" Pitch = "
;Pitch:PRINT"Note Number ";N:STOP
  410 ENDPROC
```

Program notes

This is very similar to Program 3.1. PROCCa1culatePitch has been incorporated into PROCAnalyseNote and the tune is under envelope control. Line 350 sets the volume to 0 if a rest is required. Extra information has been included in lines 360 and 400 to print the note number in case of an error. The data has been organised into a bar per fine to aid editing and debugging.

The main points of the program were discussed in Chapter 3 and, from here, we will move straight on to a three channel version and discuss some

options open to us.

The tracking method

This entails keeping track of the elapsed durations of the three channels. When a SOUND command is executed the duration value is added to its 'track'. The program is arranged to execute the SOUND commands as follows :

1) If all three channels show the same amount of elapsed duration then sync them.
2) If two channels are behind the other with the same elapsed duration then sync them.
3) If neither 1 nor 2 above apply, send a command to the channel which is lagging behind the most.
4) Repeat the above until the tune is finished.

You may like to work out a program which follows the above guidelines. In operation, you will find that the buffers are usually full and consequently the program will be unable to do anything other than play the tune. Why should that matter? Well, if that's all you want to do, fine, but the BBC micro is capable of (apparently) doing more than one thing at a time. If the programming is not too long or complex it is no hardship to program for that possibility. By using, once more, the negative ADVAL function we can do just that.

The negative ADVAL method

ADVAL with a negative argument is described quite clearly in the User Guide on page 203. If we fill two or three arrays with the required notes and use ADVAL to send SOUND commands only when a channel has space to take them, this will be enough to synchronize the music.

To control every aspect of a note we need to specify not only its pitch and duration but also which envelope it is to use and whether or not the sound channel is given a flush, synchronization or hold command. It takes no computer to calculate that each note would require four items of data. This in itself presents no problem providing you do not object to entering it.

The next program gives control over the four elements of each note. In practice, it is possible to devise some time-saving procedures, which I have done in certain cases - and you will find that you rarely have to enter four data items for each note.

```
10 REM PROGRAM 9.2
20 REM 3 Channel Version of Mozart's
30 REM Rondo Alla Turca
40 REM Using a Single Array for
50 REM Each Channel
60
```

```
   70 REM C1=Number of Notes for
   80 REM Channel 1 etc
   90 C1=46:C2=30:C3=29
  100
  110 REM 1st Subscript Refers to:
  120 REM 1 - Channel Number/Attributes
  130 REM 2 - Envelope Number
  140 REM 3 - Pitch Value
  150 REM 4 - Duration
  160
  170 DIM Chan1(4,C1)
  180 DIM Chan2(4,C2)
  190 DIM Chan3(4,C3)
  200
  210 Scale$="  C  C# D  D# E  F  F# G
G# A  A# B"
  220 Key=1
  230 Tempo=1
  240
  250 ENVELOPE1,1,0,0,0,0,0,0,126,-2,0,-
5,126,100
  260 ENVELOPE2,4,0,0,1,1,0,1,63,-1,0,-1
0,126,100
  270 ENVELOPE3,1,0,0,0,0,0,0,126,-4,-1,
-4,126,100
  280
  290 REM Channel 1
  300 FOR N=1 TO C1
  310 PROCChan
  320 Chan1(1,N)=Chan
  330 IF Note$="R" Env=0 ELSE IF N=5 OR
N=11 OR N=25 OR N=30 OR N=36 OR N=42 Env
=2 ELSE Env=1
  340 Chan1(2,N)=Env
  350 PROCAnalyseNote
  360 Chan1(3,N)=Pitch
  370 Chan1(4,N)=Duration
  380 NEXT N
  390 PRINT"Channel 1 Complete"
  400
  410 REM Channel 2
```

```
  420 FOR N=1 TO C2
  430 PROCChan
  440 Chan2(1,N)=Chan
  450 IF Note$="R" Env=0 ELSE Env=3
  460 Chan2(2,N)=Env
  470 PROCAnalyseNote
  480 Chan2(3,N)=Pitch
  490 Chan2(4,N)=Duration
  500 NEXT N
  510 PRINT"Channel 2 Complete"
  520
  530 REM Channel 3
  540 FOR N=1 TO C3
  550 PROCChan
  560 Chan3(1,N)=Chan
  570 IF Note$="R" Env=0 ELSE Env=1
  580 Chan3(2,N)=Env
  590 PROCAnalyseNote
  600 Chan3(3,N)=Pitch
  610 Chan3(4,N)=Duration
  620 NEXT N
  630 PRINT"Channel 3 Complete"
  640
  650 Ch1=0:Ch2=0:Ch3=0
  660
  670 REPEAT
  680 IF ADVAL(-6)>0 AND Ch1<C1 Ch1=Ch1+
1:SOUNDChan1(1,Ch1)+1,Chan1(2,Ch1),Chan1
(3,Ch1),Chan1(4,Ch1)*Tempo
  690 IF ADVAL(-7)>0 AND Ch2<C2 Ch2=Ch2+
1:SOUNDChan2(1,Ch2)+2,Chan2(2,Ch2),Chan2
(3,Ch2),Chan2(4,Ch2)*Tempo
  700 IF ADVAL(-8)>0 AND Ch3<C3 Ch3=Ch3+
1:SOUNDChan3(1,Ch3)+3,Chan3(2,Ch3),Chan3
(3,Ch3),Chan3(4,Ch3)*Tempo
  710 UNTIL Ch1=C1 AND Ch2=C2 AND Ch3=C3
  720
  730 END
  740
  750 DEF PROCChan
  760 READ Note$:IF LEFT$(Note$,1)="&" C
```

```
han=EVAL(Note$):READ Note$,Duration ELSE
 Chan=0:READ Duration
  770 ENDPROC
  780
  790 DEF PROCAnalyseNote
  800 IF Note$="R" Pitch=255:ENDPROC
  810 IF LEN(Note$)<2 OR LEN(Note$)>3 TH
EN PRINT"ERROR IN DATA ";Note$:PRINT"Not
e Number ";N:STOP
  820 IF LEN(Note$)=2 THEN NoteName$=LEF
T$(Note$,1) ELSE NoteName$=LEFT$(Note$,2
)
  830 Octave=VAL(RIGHT$(Note$,1))
  840 Pitch=Key+INSTR(Scale$,NoteName$)/
3*4+(Octave-1)*48
  850 IF Pitch<0 OR Pitch>255 THEN PRINT
"ERROR IN PITCH DATA ";Note$;" Pitch = "
;Pitch:PRINT"Note Number ";N:STOP
  860 ENDPROC
  870
  880 REM Channel 1
  890 DATA &200,B2,2,A2,2,G#2,2,A2,2
  900 DATA &200,C3,4,R,4,D3,2,C3,2,B2,2,
C3,2
  910 DATA &200,E3,4,R,4,F3,2,E3,2,D#3,2
,E3,2
  920 DATA &200,B3,2,A3,2,G#3,2,A3,2,B3,
2,A3,2,G#3,2,A3,2
  930 DATA &200,C4,8,A3,4,C4,2,G3,1,A3,1
  940 DATA &200,B3,4,A3,4,G3,4,A3,2,G3,1
,A3,1
  950 DATA &200,B3,4,A3,4,G3,4,A3,2,G3,1
,A3,1
  960 DATA &200,B3,4,A3,4,G3,4,F#3,4
  970 DATA &200,E3,8
  980
  990 REM Channel 2
 1000 DATA &200,R,8
 1010 DATA &200,A1,4,C2,4,C2,4,C2,4
 1020 DATA &200,A1,4,C2,4,C2,4,C2,4
 1030 DATA &200,A1,4,C2,4,C2,4,C2,4
```

```
1040 DATA &200,A1,4,C2,4,C2,4,C2,4
1050 DATA &200,E1,4,B1,4,B1,4,B1,4
1060 DATA &200,E1,4,B1,4,B1,4,B1,4
1070 DATA &200,E1,4,B1,4,B1,4,B1,4
1080 DATA &200,E1,8
1090
1100 REM Channel 3
1110 DATA &200,R,8
1120 DATA &200,R,4,E2,4,E2,4,E2,4
1130 DATA &200,R,4,E2,4,E2,4,E2,4
1140 DATA &200,R,4,E2,4,R,4,E2,4
1150 DATA &200,R,4,E2,4,E2,4,E2,4
1160 DATA &200,R,4,E2,4,E2,4,E2,4
1170 DATA &200,R,4,E2,4,E2,4,E2,4
1180 DATA &200,R,4,E2,4,R,8
1190 DATA &200,R,8
```

Like Program 9.1, the data is arranged in lines of one bar: you can use the data from Program 9.1 and add the extra data items to it. PROCAnalyseNote has one change in it at line 800 as you will see.

Program notes

The numbers of notes for each part are assigned to variables, C1, C2, and C3 in fine 90. These numbers need to be accessed several times during the program so that, if you want or need to alter data, you only need change these values once. They refer to the number of notes allocated to each channel, not to the number of data items.

Three arrays are dimensioned at lines 170 and 190 to hold information about each note. The first subscript refers to the following aspects of the note:

Subscript-l: Channel attributes, ie sync, flush, etc.
Subscript=2: Envelope number.
Subscript=3: Pitch value.
Subscript=4: Duration value.

The next section sets the data into its relevant arrays and the process is repeated once for each channel. I have kept it this way to aid understanding, although you might like to substitute a single procedure to avoid the repetition. As the process is exactly the same for each channel we will only look at channel 1 in detail.

A FOR . . . NEXT loop runs through the data, once for each note. The first step is a call to PROCChan at line 750. This 'cautiously' examines the first item of data. If the data begins with an ampersand (&) it knows it is a

channel instruction, and evaluates the string with EV AL to produce the channel attribute, Chan. It then proceeds to read Note$ and Duration. I have used hexadecimal notation to represent the sound channel parameters because it is easy to use and the ampersand tells us at a glance that this data item is a special channel command. (See Chapter 4 for further details.)

If the string does not begin with an ampersand it is taken to be a note and a second item of data, Duration, is read. As the procedure has no specific channel information, Chan is set to 0. This method of assigning the channel attributes saves us having to enter a channel parameter for every note. We only enter a parameter if we require a sync, flush or hold. The channel attribute is assigned to Chan1(1,N) in line 320.

Next, Note$ is examined to see if it is an 'R', which I have used to represent a rest. If it is an 'R', Env is set to 0 to produce a sound at zero volume: otherwise, it is set to the required envelope number. You can see this more clearly in lines 450 and 570, where only one other envelope is used.

Line 330 is arranged to give certain notes different envelopes. This saves us including a whole set of new data for the envelope number, which can be time-consuming if the envelope only changes once or twice throughout the piece. Env is assigned to Chan1(2,N) in line 340.

The program then calls PROCAnalyseNote. Unlike Program 9.1, which sent each note through PROCAnalyseNote before playing it, this program sends it through to calculate the correct pitch value of the note, and to check for any incorrect data entries. The first thing it does is check if Note$ equals 'R', ie a rest. If it does, Pitch is given an arbitrary value of 255 and the procedure ends. This is the only way this procedure differs from that in Program 9.1. Having turned the note name into a pitch value, assuming no errors, Chan(3,N) is set equal to Pitch at line 360.

Line 370 copies Duration directly into Chan1(4,N).

At the end of these three sections, the arrays are filled with figures which the SOUND command can work on directly; although they can be modified if required as we shall see.

Line 650 sets three variables, Chl, Ch2 and Ch3, to 0 and these are used to check the number of data items sent to each channel.

The REPEAT loop at lines 670 and 710 does all the work. The principle is the same for each sound channel, so only channel 1 at fine 680 will be described.

The sound buffer is checked for free spaces and the checker variable. Ch1, is compared with C1 to see if channel 1 has had all its notes. If there is free space in the buffer and there are more notes to come, Ch1 is incremented and used to provide information for the sound channel from the Chan1 array. It is at this point that the channel number is assigned, and this; is added to the channel attribute derived from PROCChan. The duration is adjusted according to the variable, Tempo.

Line 710 ensures that the loop repeats enough times to play all the notes, and then terminates it.

Modifications, alterations and suggestions

Once you have this program, you will be able to play any three-part tune by inserting new data and altering the variables C1, C2 and C3 in line 90.

If you arrange the data in fines of one bar it will aid editing and debugging.

You will notice that the channel attributes are arranged to synchronize at the beginning of each bar. This may not always be possible, especially if the music is heavily syncopated (where notes are tied or held over from one bar to the next) but you should be able to fit some sync commands in somewhere.

Earlier I said that simply using ADVAL statements would provide synchronization. Basically it will, but we must be aware of one or two points. If none of the notes are synchronized, the channels will not be exactly in sync because of the time taken by BASIC to move from the first sound command to the second, etc, especially if it has to look up values in arrays and do some calculating as in this program. If you synchronize even only the first notes, the program should keep good time provided that it is not interrupted. This can be done by altering the READ command to read only Note$ and Duration and by inserting a line such as:

```
635 Chan1(1,1)=&200:Chan2(1,1)=&200:Chan 3(1,1)=&200
```

The other elements in this array will be set to 0 upon dimensioning.

If you were to go to the trouble of removing the sync commands, you would find that the program stayed in time reasonably well, although it might drift if programmed with a longer piece. Another reason for synchronizing the channels is to allow the computer to perform other functions as the music is playing. We will cover this later in the book, but for the time being assume that we want a print-out of the notes as the music is playing. We could insert a line similar to fine 300 in Program 9 .1 such as:

```
685 PRINT;Chan1(1,Ch1)+1TAB(4)Chan1(2,Ch1)TAB(9)Chan1
(3,Ch1)TAB(14)Chan1(4,Ch1)*Tempo
```

If the channels are not synchronized, the delay between channel 1 and channel 2 (caused by BASIC taking time out to process the command) will throw the program out of sync.

If you feel that you do not require four parameters per note, it is easy to alter the arrays.

The envelope changes are often tedious {a enter. If your program requires a lot of envelopes you could arrange a separate routine to set the envelope numbers such as:

```
100 FOR E=1 TO 100
1010 IF E>0  Chan1(2,E)=1
```

```
1020 IF E>12 Chan1(2,E)=2
1030 IF E>28 Chan1(2,E)=3
1040 IF E>40 Chan1(2,E)=1
. . . .
. . . .
1100 NEXT E
```

You may find this more convenient than entering 100 or more separate figures. Assuming you do want 100 or so envelopes, you could enter them in a separate DATA statement.
The method of allocating envelopes in Program 9.2 by a series of IF . . . THEN . . . ELSE statements (the THEN commands are implied) will prove satisfactory for all but the most demanding of programs,
When programming other tunes, if all channels {lo not start together, ie if each does not sound a note at the start of the piece, be sure to include rests in the channels which are not used to keep them in sync. This was necessary in Program 9.2.

When calculating the duration of each note, I base the values on those in the chart in Figure 2.6. The important thing is to get the relationship between the notes correct. Adjustments can be made with the variable Tempo in fine 230. This is most effective when given integer values. Insert values less than 1 and observe the results.

Debugging the data

It will be a rare occurrence if you enter a set of data statements for a tune and get no errors. PROCAnalyseNote will detect errors in pitch data entry and tell you which data item it does not understand and which note number this is associated with. You will know which channel the item belongs to because a message is printed when a channel has been successfully filled with data.

An easy mistake to make is to insert wrong C1, C2 or C3 values, causing a channel to be filled with another channel's notes. You could provide further checks by inserting a termination character on the end of each channel's data: so, for example, if Note$ read a 'F and N did not equal C1 you would know that you had set C1 to the wrong value. You could use this principle to read through the data before assigning any information and use the value to dimension the arrays. You could also incorporate checks on the other data.

Once the data is correct, you could remove the error routines.

Saving the tune

If you want to play the music in a separate program, for example as

background to a graphics display, you can use the program to check out the data values and then *SPOOL the figures for the sound commands to tape or disk. The next program shows the additions necessary to do this.

```
    1 REM PROGRAM 9.3
    2 REM *SPOOL Routine To Put Sound
    3 REM Data Onto TAPE or DISC
    4 REM Include These Lines In
    5 REM PROGRAM 9.2
    6 REM Tempo=2 For DISC, 5 For TAPE
    7
  230 Tempo=2
  664 Line=5000
  665 PRINT"INSERT DISC OR TAPE then RET
URN"
  666 REPEAT:A=GET:UNTIL A=13
  667 *SPOOL"TUNE"
  680 IF ADVAL(-6)>0 AND Ch1<C1 Ch1=Ch1+
1:SOUNDChan1(1,Ch1)+1,Chan1(2,Ch1),Chan1
(3,Ch1),Chan1(4,Ch1)*Tempo:PROCSpool1
  690 IF ADVAL(-7)>0 AND Ch2<C2 Ch2=Ch2+
1:SOUNDChan2(1,Ch2)+2,Chan2(2,Ch2),Chan2
(3,Ch2),Chan2(4,Ch2)*Tempo:PROCSpool2
  700 IF ADVAL(-8)>0 AND Ch3<C3 Ch3=Ch3+
1:SOUNDChan3(1,Ch3)+3,Chan3(2,Ch3),Chan3
(3,Ch3),Chan3(4,Ch3)*Tempo:PROCSpool3
  715 *SPOOL
  861
  862 DEF PROCSpool1
  863 PRINT;Line;" DATA ";Chan1(1,Ch1)+1
;",";Chan1(2,Ch1);",";Chan1(3,Ch1);",";C
han1(4,Ch1)*Tempo
  864 Line=Line+10
  865 ENDPROC
  866
  867 DEF PROCSpool2
  868 PRINT;Line;" DATA ";Chan2(1,Ch2)+1
;",";Chan2(2,Ch2);",";Chan2(3,Ch2);",";C
han2(4,Ch2)*Tempo
  869 Line=Line+10
  870 ENDPROC
```

```
  871
  872 DEF PROCSpool3
  873 PRINT;Line;" DATA ";Chan3(1,Ch3)+1
;",";Chan3(2,Ch3);",";Chan3(3,Ch3);",";C
han3(4,Ch3)*Tempo
  874 Line=Line+10
  875 ENDPROC
  876
10000 DATA -1,-1,-1,-1
10010
10020 REM These Lines Play the DATA
10030 RESTORE5000
10040 REPEAT
10050 READ Chan,Env,Pitch,Dur
10060 REM Set Divisor Equal to Tempo
10070 SOUNDChan,Env,Pitch,Dur/2
10080 UNTIL Chan=-1
```

Program notes

The three Spool procedures are identical for each channel and the data items are printed one note per line. You may like to modify the program to print more data per line and reduce the Spool procedures into one. Be sure to REM out these lines or insert a GOTO to jump over them until your data has been checked and verified.

The data is retrieved by executing:

```
*EXEC"TUNE"
```

See page 402 of the User Guide for more information about *SPOOL and *EXEC.

As they stand, these procedures may require some modification. The principle is to let the ADVAL function calculate the correct spacing of the data items. However, because the filing systems themselves require attention from the operating system, the ADVAL function may sometimes be waiting for notes when the operating system is not ready to give them. This will happen if a channel is supplied with a lot of short notes.

One way around this is to increase the variable Tempo in line 230, so that the sound channels do not empty while waiting for the filing system to finish with the OS. Tape will require longer values than disk, but experiments have shown that setting Tempo to 5 for Tape and to 2 for disk will work for Rondo Alla Turca. On playback, divide the duration by the increase in tempo value or, in the PROCSpool procedures, divide Tempo by whatever you needed to multiply it by in line 230. Other tunes may require different values.

The routine beginning at line 10000 will play the data. If you substitute 'SOUND' for the string 'DATA' in the procedures at fines 863, 868 and 873, the *SPOOLed program will play the tune when run.

The program depends so much upon the buffers not emptying before another SOUND command comes along, that the whole program may be better served by a routine which produces SOUND commands according to the tracking method described earlier. This would remove it from the mercy of the *SPOOLing filing system: although if you are using disks this will not be a serious problem.

Experimenting with the program

Program 9.2 is only the beginning. You could add an extra array and include channel 0 for drum effects. You could also use channel 0 instead of channel 1 to produce the lower notes we mentioned in Chapter 7.

If 16 envelopes are not enough, you can redefine envelopes in mid-program. You can do this by calling a procedure from a fine which is inserted in a similar way to the PRINT statement in fine 685 described earlier.

If you include the variable Key inside the REPEAT loop instead of in PROCAna1yseNote, you can produce a key change by altering the value of Key as explained in Chapter 3.

A separate array can be used to trigger calls to any number of procedures, to create any of the effects we've covered in the previous chapters; you can produce your own arrangements of your favourite tunes as well as programming your own compositions - which leads us straight into the next chapter. But first . . .

More tunes to play

These three programs include the data and information required to play other tunes. If the programs are entered as written and saved as files, using *SPOOL, you can load Program 9.2 and *EXEC these programs into the computer: they will be ready to run. See Appendix 2 and the User Guide page 402 for futher information.

Program 9.4 plays the next 16 bars of Ronda Aha Turca and includes a repeat of the first eight.

```
    1 REM PROGRAM 9.4
    2 REM Anothr 24 bars of Mozart's
    3 REM Rondo Alla Turca
    4 REM Insert These Lines into
    5 REM PROGRAM 9.2
    6
   90 C1=168:C2=114:C3=114
  275 ENVELOPE4,4,0,8,-8,0,1,1,126,-1,0,
-2,126,100
```

```
  305 IF N=47 OR N=129 RESTORE 890
  306 IF N=153 RESTORE 979
  330 IF Note$="R" Env=0 ELSE IF N=52 OR
 N=58 OR N=72 OR N=77 OR N=83 OR N=89 En
v=2 ELSE Env=1
  335 IF N=153 OR N=165 Env=4
  345 IF N=153 OR N=165 Env=4
  425 IF N=31 OR N=85 RESTORE 1000
  426 IF N=98 RESTORE 1089
  545 IF N=30 OR N=85 RESTORE 1110
  546 IF N=98 RESTORE 1199
  880 REM Channel 1
  970 DATA &200,E3,8,E3,4,F3,4
  971 DATA &200,G3,4,G3,4,A3,2,G3,2,F3,2
,E3,2
  972 DATA &200,D3,8,E3,4,F3,4
  973 DATA &200,G3,4,G3,4,A3,2,G3,2,F3,2
,E3,2
  974 DATA &200,D3,8,C3,4,D3,4
  975 DATA &200,E3,4,E3,4,F3,2,E3,2,D3,2
,C3,2
  976 DATA &200,B2,8,C3,4,D3,4
  977 DATA &200,E3,4,E3,4,F3,2,E3,2,D3,2
,C3,2
  978 DATA &200,B2,8
  979 DATA &200,C4,8,A3,4,B3,4
  980 DATA &200,C4,4,B3,4,A3,4,G#3,4
  981 DATA &200,A3,4,E3,4,F3,4,D3,4
  982 DATA &200,C3,8,B2,6,A2,1,B2,1
  983 DATA &200,A2,8
  990 REM Channel 2
 1080 DATA &200,E1,8,R,8
 1081 DATA &200,C1,4,C2,4,E1,4,E2,4
 1082 DATA &200,G1,8,R,8
 1083 DATA &200,C1,4,C2,4,E1,4,E2,4
 1084 DATA &200,G1,8,R,8
 1085 DATA &200,C1,4,A1,4,C1,4,C2,4
 1086 DATA &200,E1,8,R,8
 1087 DATA &200,C1,4,A1,4,C1,4,C2,4
 1088 DATA &200,E1,8,R,8
 1089 DATA &200,F1,4,A1,4,A1,4,A1,4
```

```
 1090 DATA &200,E1,4,A1,4,D1,4,F1,4
 1091 DATA &200,C1,4,E1,4,D1,4,F1,4
 1092 DATA &200,E1,4,E1,4,E1,4,E1,4
 1093 DATA &200,A1,8
 1100 REM Channel 3
 1190 DATA &200,R,8,C3,4,D3,4
 1191 DATA &200,E3,4,E3,4,R,8
 1192 DATA &200,B2,4,G2,4,C3,4,D3,4
 1193 DATA &200,E3,4,E3,4,R,8
 1194 DATA &200,B2,8,A2,4,B2,4
 1195 DATA &200,C3,4,C3,4,R,8
 1196 DATA &200,G#2,4,E2,4,A2,4,B2,4
 1197 DATA &200,C3,4,C3,4,R,8
 1198 DATA &200,R,8
 1199 DATA &200,R,4,D#2,4,D#2,4,D#2,4
 1200 DATA &200,R,4,E2,4,R,4,B1,4
 1201 DATA &200,R,4,A1,4,R,4,B1,4
 1202 DATA &200,A1,4,A1,4,G#1,4,G#1,4
 1203 DATA &200,A1,8
```

Notice how repeats have easily been programmed by the RESTORE function at lines 305, 306, 425, 426, 545 and 546. The only other programming has been an additional envelope.

```
   10 REM PROGRAM 9.5
   20 REM Tschaikowsky's
   30 REM Dance of the Sugar-plum Fairy
   40 REM Insert These Lines Into
   50 REM PROGRAM 9.2
   60
   90 C1=104:C2=95:C3=86
  230 Tempo=2
  250 ENVELOPE1,6,0,0,0,0,0,0,126,-8,0,-
8,126,94
  260 ENVELOPE2,4,0,0,1,1,0,1,100,-10,-1
0,-10,100,70
  270 ENVELOPE3,3,0,0,0,0,0,0,116,-8,0,-
8,116,84
  275 ENVELOPE4,6,0,0,1,1,0,1,100,-5,-5,
-5,100,80
  276 ENVELOPE5,1,0,0,1,1,0,1,100,-5,-1,
-10,100,50
  330 IF Note$="R" Env=0 ELSE IF N=C1 En
```

```
v=5 ELSE IF N<17 Env=2 ELSE Env=1
  450 IFNote$="R" Env=0 ELSE IF N=C2 Env
=5 ELSE IF N>73 Env=4 ELSE Env=2
  570 IF Note$="R" Env=0 ELSE IF N=C3 En
v=5 ELSE IF N<68 Env=2 ELSE Env=3
  870
  880 REM Channel 1
  890 DATA &200,R,4,E3,4,R,4,F#3,4
  900 DATA &200,R,4,G3,4,R,4,D#3,4
  910 DATA &200,R,4,E3,4,R,4,F#3,4
  920 DATA &200,R,4,G5,2,E5,2,G5,4,F#5,4
  930 DATA &200,D#5,4,E5,4,D5,2,D5,2,D5,
4
  940 DATA &200,B4,2,E5,2,C5,2,E5,2,B4,4
,R,4
  950 DATA &200,R,4,G4,2,E4,2,G4,4,F#4,4
  960 DATA &200,B4,2,E5,2,C5,2,E5,2,B4,4
,R,4
  970 DATA &200,R,4,G4,2,E4,2,G4,4,F#4,4
  980 DATA &200,C5,4,B4,4,G5,2,G5,2,G5,4
  990 DATA &200,F#5,2,F#5,2,F#5,4,E5,2,E
5,2,E5,4
 1000 DATA &200,D#5,2,F#5,2,E5,2,F#5,2,D
#5,4,R,4
 1010 DATA &200,R,4,G5,2,E5,2,G5,4,F#5,4
 1020 DATA &200,D#5,4,E5,4,D5,2,D5,2,D5,
4
 1030 DATA &200,C#5,2,C#5,2,C#5,4,C5,2,C
5,2,C5,4
 1040 DATA &200,B4,2,E5,2,C5,2,E5,2,B4,4
,R,4
 1050 DATA &200,R,4,E4,2,C#4,2,E4,4,D#4,
4
 1060 DATA &200,R,4,D4,2,B3,2,D4,4,C#4,4
 1070 DATA &200,R,4,C4,2,A3,2,C4,4,B3,4
 1080 DATA &200,R,4,B3,1,D#4,1,F#4,1,B4,
1,E4,4,B2,4
 1090
 1100 REM Channel 2
 1110 DATA &200,E2,4,G2,4,E2,4,A2,4
 1120 DATA &200,E2,4,A#2,4,E2,4,A2,4
 1130 DATA &200,E2,4,G2,4,E2,4,A2,4
 1140 DATA &200,E2,4,A#2,4,E2,4,A2,4
 1150 DATA &200,E2,4,B2,4,E2,4,C3,4
```

```
 1160 DATA &200,E2,4,C#3,4,E2,4,D3,4
 1170 DATA &200,E2,4,E3,4,E2,4,F#3,4
 1180 DATA &200,E3,4,E3,4,E3,4,E2,1,D2,1
,C2,1,B1,1
 1190 DATA &200,A#1,4,C3,4,A1,4,C3,4
 1200 DATA &200,G1,4,B2,4,F#1,4,A#2,4
 1210 DATA &200,F#2,4,B2,4,F#2,4,C#3,4
 1220 DATA &200,B1,4,C2,4,B1,4,B1,1,A1,1
,G1,1,F#1,1
 1230 DATA &200,E1,4,B2,4,E2,4,C3,4
 1240 DATA &200,E2,4,C#3,4,E2,4,D3,4
 1250 DATA &200,E2,4,E3,4,E2,4,F#3,4
 1260 DATA &200,E3,4,E3,4,E3,4,G3,1,F#3,
1,E3,1,D3,1
 1270 DATA &200,C#3,4,F#2,8,F#3,1,E3,1,D
#3,1,C#3,1
 1280 DATA &200,B2,4,E2,8,E3,1,D3,1,C#3,
1,B2,1
 1290 DATA &200,A2,4,D2,8,D3,1,C3,1,B2,1
,A2,1
 1300 DATA &200,G2,4,F#2,4,E2,4,B0,4
 1310
 1320 REM Channel 3
 1330 DATA &200,R,4,B2,4,R,4,C3,4
 1340 DATA &200,R,4,C#4,4,R,4,C4,4
 1350 DATA &200,R,4,B2,4,R,4,C3,4
 1360 DATA &200,R,4,C#4,4,R,4,C4,4
 1370 DATA &200,R,4,G2,4,R,4,A2,4
 1380 DATA &200,R,4,C#4,4,R,4,C4,4
 1390 DATA &200,R,4,G2,4,R,4,A2,4
 1400 DATA &200,G3,4,A3,4,G3,4,R,4
 1410 DATA &200,R,4,E2,4,R,4,D#2,4
 1420 DATA &200,F#4,4,E4,4,A#4,2,A#4,2,A
#4,2
 1430 DATA &200,G#4,2,G#4,2,G#4,4,F#4,2,
F#4,2,F#4,4
 1440 DATA &200,B2,4,A#2,4,F#2,4,R,4
 1450 DATA &200,R,4,G2,4,R,4,A2,4
 1460 DATA &200,R,4,A#2,4,R,4,B2,4
 1470 DATA &200,R,4,C#3,4,R,4,D#4,4
 1480 DATA &200,G3,4,A3,4,G3,4,R,4
 1490 DATA &200,R,4,A#3,2,F#3,2,A#3,4,A3
,4
 1500 DATA &200,R,4,G#3,2,E3,2,G#3,4,G3,
```

```
4
 1510 DATA &200,R,4,F#3,2,D3,2,F#3,4,G3,
4
 1520 DATA &200,R,4,A2,4,E2,4,B1,4
```

The envelopes have been redefined and re-allocated, but otherwise this just supplies new data to Program 9.2

```
   10 REM PROGRAM 9.6
   20 REM John Philip Sousa's
   30 REM The Liberty Bell March
   40 REM Insert These Lines Into
   50 REM PROGRAM 9.2
   90 C1=174:C2=207:C3=173
  250 ENVELOPE1,1,0,0,0,0,0,0,50,50,-2,-
2,50,126
  260 ENVELOPE2,1,0,0,0,0,0,0,32,-8,-2,-
4,126,90
  270 ENVELOPE3,1,0,0,0,0,0,0,16,-8,-2,-
4,126,90
  275 ENVELOPE4,4,0,8,-8,0,1,1,110,0,0,-
10,110,110
  276 ENVELOPE5,2,1,0,1,0,1,1,16,-8,-2,-
4,126,90
  277 ENVELOPE6,1,1,0,1,0,1,1,32,-8,-2,-
4,126,90
  305 IF N=65 RESTORE 930
  330 IF Note$="R" Env=0 ELSE IF (N=1 OR
 N=78 OR N=93 OR N=108 OR N=162) Env=4 E
LSE Env=1
  425 IF N=79 RESTORE 2560
  450 IF Note$="R" Env=0 ELSE IF N=C2-1
Env=1 ELSE IF N>78 Env=5 ELSE Env=3
  545 IF N=71 RESTORE 3050
  570 IF Note$="R" Env=0 ELSE IF N=130 E
nv=4 ELSE IF N=C3-1 Env=1 ELSE Env=2
  880 REM Channel 1
  890 DATA &200,F4,60,&1000,R,9
  900
  910
  920 DATA C3,3
  930 DATA &200,A2,6,A2,3,A2,3,G#2,3,A2,
3
  940 DATA &200,F3,6,C3,3,C3,6,A2,3
  950 DATA &200,A#2,6,A#2,3,A#2,6,C3,3
```

```
  960 DATA &200,D3,15,A#2,3
  970 DATA &200,G2,6,G2,3,G2,3,F#2,3,G2,
3
  980 DATA &200,E3,6,D3,3,D3,6,A#2,3
  990 DATA &200,A2,6,A2,3,A2,6,A#2,3
 1000 DATA &200,C3,15,C3,3
 1010 DATA &200,A2,6,A2,3,A2,3,G#2,3,A2,
3
 1020 DATA &200,A3,6,F3,3,F3,6,C3,3
 1030 DATA &200,B2,6,G3,3,G3,6,G3,3
 1040 DATA &200,G3,15,F3,3
 1050 DATA &200,E3,6,G3,3,G3,3,F#3,3,G3,
3
 1060 DATA &200,D3,6,G3,3,G3,3,F#3,3,G3,
3
 1070 DATA &200,C3,6,B2,3,C3,6,B2,3
 1080 DATA &200,C3,9,C3,9
 1090 REM 2nd Part
 1100 DATA &200,A2,3,G#2,3,A2,3,D3,6,C3,
3
 1110 DATA &200,A2,9,F2,9
 1120 DATA &100,D2,9,G2,9
 1130 DATA &100,F2,15,F2,3
 1140 DATA &200,G2,3,A2,3,A#2,3,E3,6,D3,
3
 1150 DATA &200,C3,9,F3,9
 1160 DATA &200,E3,9,D3,9
 1170 DATA &200,C3,15,C3,3
 1180 DATA &200,D3,6,D3,2,E3,1,D3,3,C#2,
3,D3,3
 1190 DATA &200,E3,9,E3,9
 1200 DATA &200,F3,6,F3,2,A3,1,G3,3,F3,3
,G3,3
 1210 DATA &200,A3,15,A3,2,A3,1
 1220 DATA &200,G3,6,F3,3,D3,6,A#2,3
 1230 DATA &200,A2,9,F2,9
 1240 DATA &200,G2,9,E2,9
 1250 DATA &200,F2,12,R,6
 2500
 2510 REM Channel 2
 2520 DATA &200,F2,6,E2,3,D#2,6,D2,3
 2530 DATA &100,C2,6,B1,3,A#1,6,A1,3
 2540 DATA &100,G1,3,A1,3,A#1,3,A1,6,G1,
3
```

```
 2550 DATA &100,C1,6,R,12
 2560 DATA &200,F1,6,F2,3,F2,6,F2,3
 2570 DATA &200,F1,6,F2,3,F2,6,F2,3
 2580 DATA &200,E1,6,F1,3,G1,6,F1,3
 2590 DATA &200,E1,6,D1,3,C1,6
 2600 DATA &200,C1,6,A#1,3,A#1,6,A#1,3
 2610 DATA &200,C1,6,A#1,3,A#1,6,A#1,3
 2620 DATA &200,F1,6,G1,3,A1,6,G1,3
 2630 DATA &200,F1,6,E1,3,D1,6,C1,3
 2640 DATA &200,F1,6,F2,3,F2,6,F2,3
 2650 DATA &200,F1,6,F2,3,F2,6,F2,3
 2660 DATA &200,D1,6,E1,3,F1,6,E1,3
 2670 DATA &200,D1,6,C1,3,B0,6,A1,3
 2680 DATA &200,G1,6,E2,3,E2,6,E2,3
 2690 DATA &200,G1,6,F2,3,F2,6,F2,3
 2700 DATA &200,C2,6,T1,3,C2,6,G1,3
 2710 DATA &200,C1,6,R,3,C2,6,R,3
 2720 REM 2nd Part
 2730 DATA &200,F1,6,A1,3,C1,6,A1,3
 2740 DATA &200,F1,6,A1,3,C1,6,A1,3
 2750 DATA &100,A#1,6,F1,3,C1,6,E1,3
 2760 DATA &100,F1,6,A1,3,A1,6,A1,3
 2770 DATA &200,E1,6,A#1,3,C1,6,A#1,3
 2780 DATA &200,F1,6,A1,3,D1,6,G#1,3
 2790 DATA &200,C1,4,F1,2,G1,3,D1,4,F1,2
,G1,3
 2800 DATA &200,C1,6,G1,3,G1,6,81,3
 2810 DATA &200,B0,6,B1,3,B1,6,B1,3
 2820 DATA &200,C#1,6,A1,3,A1,6,A1,3
 2830 DATA &200,D1,6,A1,3,D1,6,A#1,3
 2840 DATA &200,C1,6,A1,3,A1,6,A1,3
 2850 DATA &200,G1,6,A#1,3,A#1,6,A#1,3
 2860 DATA &200,C1,6,A1,3,A1,6,A1,3
 2870 DATA &200,E1,6,A#1,3,C1,6,A#1,3
 2880 DATA &200,F1,6,R,3,F1,6,R,3
 2890
 3000 REM Channel 3
 3010 DATA &200,F3,6,E3,3,D#3,6,D3,3
 3020 DATA &100,C3,6,B2,3,A#2,6,A2,3
 3030 DATA &100,G2,3,A2,3,A#2,3,A2,6,G2,
3
 3040 DATA &100,C2,6,R,12
 3050 DATA &200,F2,6,C2,3,C2,6,C2,3
 3060 DATA &200,F2,6,C2,3,C2,6,C2,3
```

```
3070 DATA &200,G2,6,G2,3,G2,6,G2,3
3080 DATA &200,G2,15,R,3
3090 DATA &200,G1,6,C2,3,C2,6,C2,3
3100 DATA &200,G1,6,C2,3,C2,6,C2,3
3110 DATA &200,F2,6,F2,3,F2,6,G2,3
3120 DATA &200,A2,15,A2,3
3130 DATA &200,F2,6,C2,3,C2,6,C2,3
3140 DATA &200,F2,6,C2,3,C2,6,C2,3
3150 DATA &200,D2,9,B1,9
3160 DATA &200,B1,6,C2,3,D2,6,R,3
3170 DATA &200,G2,6,C2,3,C2,6,C2,3
3180 DATA &200,G2,6,B1,3,B1,6,B1,3
3190 DATA &200,E2,6,R,3,G1,6,R,3
3200 DATA &200,E2,6,R,3,E2,6,R,3
3210 REM 2nd Part
3220 DATA &200,R,18
3230 DATA &200,F4,54
3240
3250
3260 DATA &200,R,6,C2,3,R,6,C2,3
3270 DATA &200,R,6,A1,3,R,6,G#1,3
3280 DATA &200,G2,9,F2,9
3290 DATA &200,R,6,C2,3,C2,6,C2,3
3300 DATA &200,F2,6,D2,3,D2,6,D2,3
3310 DATA &200,R,6,C#2,3,C#2,6,C#2,3
3320 DATA &200,R,6,D2,3,R,6,D2,3
3330 DATA &200,R,6,E2,3,E2,6,E2,3
3340 DATA &200,R,6,G2,3,G2,6,G2,3
3350 DATA &200,R,6,C2,3,C2,6,C2,3
3360 DATA &200,R,6,C2,3,C2,6,C2,3
3370 DATA &200,F2,6,R,3,F2,6,R,3
```

This makes uses of the RESTORE function for repeats and several envelopes for tone variation. The trill envelope is also used to good effect.

Notice the empty data lines, where a note lasts for more than one bar, and the necessary change in sync parameters in the other channels. The empty fines are there purely to aid readability.

Notice also the use of the dummy hold parameter to allow the initial trill to fade rather than cease abruptly.

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 we produce compositions through scientific and mathematical means. This is one area which is relatively unexplored and there is great scope for us to discover new ideas - and compositions - with our BBC micro.

Even art must follow certain rules unless it is to be totally anarchistic. Music is no exception. Anarchy is easily expressed by this short program.

```
10 REPEAT
20 SOUND RND(3),RND(15)-16,RND(255),RND(10)
30 UNTIL FALSE
```

Interesting is perhaps the most apt description, and you can build upon this idea to produce even more interesting compositions.

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: 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 v. 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.

The human compositional process: 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.

A 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 to point the way to, hopefully, the correct answer or best solution. Unlike an algorithm, a heuristic is not guaranteed to produce the best solution. It's not even guaranteed to produce any 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. This is theoretically possible because the number of moves is not infinite but it is large enough to render such an algorithmic approach impossible.

A heuristic approach might work on the principle of controlling the centre of the board. This is likely to produce a good game but is not guaranteed to win. Algorithms and heuristics are often combined to produce doubly effective results.

The human compositional process is a mixture of algorithm and heuristic (and inspiration, but whether or not this would be classed as a part of the heuristic operation is debatable).

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 knows this but still needs to apply rules of thumb to create a melody over the top of the sequence. These rules are the result of experience and inspiration - and this is where we resort to the RND function and to devising some heuristics and rules for the computer.

Aspects of a composition

There are three aspects of composition of direct relevance to us:

1) Melody
2) Rhythm
3) Harmony.

The first two are very closely related and the third, harmony, is complex enough to have had many volumes written about it. In this chapter, we wm examine melody anti Its associated rhythm and delve into harmony in Chapter 11.

The first steps in computer compositions, once we've got past the totally random note stage, usually involve trying to create a pleasing melody or sequence of notes. We can devise a set of rules to do this, making them as simple or as complex as we like. This in itself is not so 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 as note pitch and note length are generally so tightly interwoven that one can often determine the other - from an artistic viewpoint.

We will begin our experiments with the next program. It is an expanded version of Program 3.1, which played a series of completely unrelated notes. This program 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 ourselves might look like this:

1) The first note must be a member of a C major chord, ie C, E or G.
2) The interval between any two consecutive notes must not be more than four notes.
3) A B note 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 Program 3.1, as you will hear.

```
   10 REM PROGRAM 10.1
   20 REM Computer Composition
   30 REM Based on Rules
   40
   50 PROCSetup
   60
   70 NoOfBars=4
   80 Count=0
   90
  100 FOR B=1 TO NoOfBars
  110 PRINT"Composing Bar ";B
  120 PROCBar
  130 NEXT B
  140
  150 PROCPlay
  160
  170 PRINT"Press SPACE For Another Tune
"
  180 PRINT"Press ""R"" For A Replay"
```

```
  190
  200 REPEAT
  210 *FX15,1
  220 Key$=GET$
  230 UNTIL (Key$=" " OR Key$="R")
  240 IF Key$="R" GOTO 150 ELSE GOTO 80
  250
  260 END
  270
  280 DEF PROCSetup
  290 Scale$="  C  C# D  D# E  F  F# G
G# A  A# B"
  300
  310 ENVELOPE1,4,0,1,0,1,1,0,126,-8,0,-
8,126,80
  320 Key=1
  330 Tempo=2
  340
  350 DIM Tune(2,129), Tune$(129)
  360
  370 DIM NotesToChooseFrom$(15)
  380 FOR S%=1 TO 15
  390 READ Note$
  400 NotesToChooseFrom$(S%)=Note$
  410 NEXT S%
  420 ENDPROC
  430
  440 DATAG1,A1,B1,C2,D2,E2,F2,G2,A2,B2,
C3,D3,E3,F3,G3
  450
  460 DEF PROCChooseNote
  470 Note=RND(15)
  480 Note$=NotesToChooseFrom$(Note)
  490 ENDPROC
  500
  510 DEF PROCPlay
  520 FOR P=1 TO Count
  530 PRINTTune$(P),Tune(2,P)*Tempo
  540 SOUND1,1,Tune(1,P),Tune(2,P)*Tempo
  550 NEXT P
  560 ENDPROC
```

```
  570
  580 DEF PROCAnalyseNote
  590 IF LEN(Note$)<2 OR LEN(Note$)>3 TH
EN PRINT"ERROR IN DATA ";Note$:STOP
  600 IF LEN(Note$)=2 THEN NoteName$=LEF
T$(Note$,1) ELSE NoteName$=LEFT$(Note$,2
)
  610 PositionInScale=INSTR(Scale$,NoteN
ame$)/3
  620 Octave=VAL(RIGHT$(Note$,1))
  630 ENDPROC
  640
  650 DEF PROCCalculatePitch
  660 Pitch=Key+PositionInScale*4+(Octav
e-1)*48
  670 IF Pitch<0 OR Pitch>255 THEN PRINT
"ERROR IN PITCH DATA ";Note$;" Pitch = "
;Pitch:STOP
  680 ENDPROC
  690
  700 DEF PROCBar
  710 DurationSoFar=0
  720 REPEAT
  730 Count=Count+1
  740 REPEAT
  750 PROCChooseNote
  760 PROCAnalyseNote
  770 PROCRules
  780 UNTIL NoteOK
  790 LastNote=Note
  800 PROCCalculatePitch
  810 Tune(1,Count)=Pitch:Tune$(Count)=N
ote$
  820 PROCDur
  830 Tune(2,Count)=Dur
  840 DurationSoFar=DurationSoFar+Dur
  850 UNTIL DurationSoFar=16
  860 ENDPROC
  870
  880 DEF PROCDur
  890 REPEAT
```

```
  900 Dur=2^RND(2)
  910
  920 REM Set Last Note To Semibreve
  930 IF B=NoOfBars Dur=16
  940
  950 UNTIL DurationSoFar+Dur<=16
  960 ENDPROC
  970
  980 DEF PROCRules
  990 NoteOK=FALSE
 1000
 1010 REM Set First Note
 1020 IF Count=1 AND NOT(NoteName$="C" O
R NoteName$="G" OR NoteName$="E") ENDPRO
C
 1030
 1040 REM Make a "B" Move up to a "C"
 1050 IF Count>1 AND LEFT$(Tune$(Count-1
),1)="B" Note=LastNote+1:Note$=NotesToCh
ooseFrom$(Note):PROCAnalyseNote
 1060
 1070 REM Restrict Jumps To 4 Notes
 1080 IF Count>1:IF ABS(LastNote-Note)>4
 ENDPROC
 1090
 1100 REM Set Last Bar
 1110 IF B=NoOfBars AND NoteName$<>"C" E
NDPROC
 1120
 1130 NoteOK=TRUE
 1140 ENDPROC
```

Program notes

At line 350 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 sometimes to refer to the name of a note and sometimes to its pitch value or position in NotesToChooseFrom$.

NotesToChooseFrom$ stores the available notes. This has been altered slightly from Program 3.1 to base our experiments in the key of C.

In order to create some sort of order, the program composes in bars: the number of bars is determined by line 70. If you alter this to more than eight

bars, you may have to redimension the Tune and Tune$ arrays.

After the bars have been composed, PROCPlay at line 510 plays them and prints out the notes. Lines 170 to 240 give you the opportunity to hear the tune again or to compose another one.

PROCAnalyseNote and PROCCa1cuIatePitch remain the same and have been kept separate so that 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 procedures, as may be necessary, for example, in rule 3 above.

DurationSoFar in PROCBar keeps track of the cumulative length of the notes in each bar so that each bar contains the equivalent of 16 quavers. The program therefore produces music in 4/4 time.

The loop at line 720 repeats and calls PROCRules until NoteOK is TRUE, which means it's passed the rules. The pitch is then calculated and, along with the note name, assigned to the relevant array.

PROCDur gives the note a duration of 2 or 4 and ensures that the last note has a value of 16. This is assigned in line 830.

The outside loop in lines 720 to 850 repeat until the bar is full, when DurationSoFar is equal to 16.

PROCRules assumes that the note is not going to be OK in line 990 and it must run the gauntlet of the rules until it comes out OK at line 1130.

The first rule checks that the first note is C, E or G.

The second rule creates a C to follow a B. This principle could be used to ensure a particular note was always followed by another certain note but be careful how you arrange the note data. Note 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 G3 (Note= 15) would take it off the scale.

The next rule checks the steps between the last note and this 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 Program 3.1, it's probably fair to say that it produces music only a programmer could love. We still need more melodic rules and the phrasing produced by PROCDur needs more attention. You can easily add and adapt the melodic rules described above. Here are some further suggestions:

1) Not more than five notes rising or descending without 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 or vice versa. (This is quite a harsh interval unless harmonically controlled.)
5) Permit the inclusion of accidentals, possibly only F# and/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. PROCDur makes no attempt to regulate the rhythmic content.

The phrasing can 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, etc. Or bar 1 to bar 3 and bar 2 to bar 4.

Alter the RND parameter in fine 900 to produce other note durations.

Another 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.

```
    1 REM PROGRAM 10.2
    2 REM Computer Compositions with
    3 REM Fixed Rhythm Pattern
    4 REM Insert in PROGRAM 10.1
   85 RESTORE 920
  890 READ Dur
  900 ENDPROC
  910
  920 DATA 2,2,2,2,4,4
  930 DATA 2,2,4,8
  940 DATA 2,2,4,2,2,4
  950 DATA 4,4,8
  960
  970
  980 DEF PROCRules
  990 NoteOK=FALSE
 1110 IF Count=19 AND NoteName$<>"C" END
```

Program notes

The value that Count is checked against in line 1110 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 and 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 arrange 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' J)

The success of such experiments depends upon the composition(s) 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 and perform a similar analysis upon them.

We can increase the accuracy of our analysis by recording how often any note follows every 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 more 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.

```
   10 REM PROGRAM 10.3
   20 REM Computer Compositions
   30 REM Based Upon Note Analysis
   40
   50 MODE 7
   60 REM Page Mode Off
   70 VDU15
   80 ENVELOPE1,4,0,1,0,1,1,0,126,-8,0,-
8,120,90
   90
  100 DIM Tune$(238),Dur%(238),F1%(36)
  110 Key%=37
  120 Scale$="  A1 A#1B1 C1 C#1D1 D#1E1
F1 F#1G1 G#1A2 A#2B2 C2 C#2D2 D#2E2 F2 F
#2G2 G#2A3 A#3B3 C3 D#3D3 D#3E3 F3 F#3G3
 G#3"
  130
```

```
  140 PROCGetTune
  150 PROCNewScale
  160 PROCAnalyseTune
  170 PROCCalcPercentages
  180 PROCPrint
  190
  200 INPUT"Enter 'LAST BUT ONE' and 'LA
ST NOTE'    in terms of note number in N
ew Scale",Penult%,LastNote%
  210 PRINT'"Press 'S' to STOP"'"Enter s
earch depth (1/2/3) - This may bealtered
 as the program is running?":Play%=GET
  220
  230 D%=0
  240 REPEAT
  250 PROCGetNote
  260 PROCPlay
  270 PL%=INKEY(0):IF PL%>48 AND PL%<52
THEN Play%=PL%
  280 UNTIL PL%=83
  290
  300 END
  310
  320 DEFPROCGetTune
  330 PRINT"Reading in Tune For Analysis
"'
  340 REM RESTORE To Required Tune
  350 RESTORE 1810
  360
  370 Count%=0
  380 REPEAT
  390 Count%=Count%+1
  400 READ Note$,Dur:IF Note$="X" GOTO 4
30
  410 Tune$(Count%)=Note$
  420 Dur%(Count%)=Dur
  430 UNTIL Note$="X"
  440
  450 TuneLength%=Count%
  460 ENDPROC
  470
```

```
  480 DEFPROCNewScale
  490 PRINT"Calculating New Scale"'
  500
  510 FOR Note%=1 TO TuneLength%
  520 Pos%=INSTR(Scale$,Tune$(Note%))/3
  530 F1%(Pos%)=F1%(Pos%)+1
  540 NEXT Note%
  550
  560 Scale2$="  "
  570
  580 FOR Note%=1 TO 36
  590 IF F1%(Note%)>0 THEN Scale2$=Scale
2$+MID$(Scale$,Note%*3,3)
  600 NEXT Note%
  610
  620 PRINT"New Scale = ";Scale2$'
  630 ScaleLength%=(LEN(Scale2$)-2)/3
  640 PRINT"Scale length = ";ScaleLength
%'
  650
  660 DIM F2%(ScaleLength%,ScaleLength%)
,F3%(ScaleLength%,ScaleLength%,ScaleLeng
th%)
  670
  680 REM Reset F1% Array
  690 FOR C%=1 TO 36
  700 F1%(C%)=0
  710 NEXT C%
  720 ENDPROC
  730
  740 DEFPROCAnalyseTune
  750 PRINT"Analysing Tune..."'
  760 FOR Note%=1 TO TuneLength%
  770 Pos1%=INSTR(Scale2$,Tune$(Note%))/
3
  780 F1%(Pos1%)=F1%(Pos1%)+1
  790 IF Note%>TuneLength%-1 THEN GOTO 8
20
  800 Pos2%=INSTR(Scale2$,Tune$(Note%+1)
)/3
  810 F2%(Pos1%,Pos2%)=F2%(Pos1%,Pos2%)+
```

```
1
  820 IF Note%>TuneLength%-2 THEN GOTO 8
50
  830 Pos3%=INSTR(Scale2$,Tune$(Note%+2)
)/3
  840 F3%(Pos1%,Pos2%,Pos3%)=F3%(Pos1%,P
os2%,Pos3%)+1
  850 NEXT Note%
  860 ENDPROC
  870
  880 DEFPROCCalcPercentages
  890
  900 PRINT"Calculating First Order Freq
uency..."
  910 Sum1%=0
  920 FOR n1%=1 TO ScaleLength%
  930 Sum1%=Sum1%+F1%(n1%)
  940 NEXT n1%
  950 FOR n1%=1 TO ScaleLength%
  960 F1%(n1%)=F1%(n1%)*100/Sum1%
  970 NEXT n1%
  980
  990 PRINT"Calculating Second Order Fre
quency..."
 1000 FOR n1%=1 TO ScaleLength%
 1010 Sum2%=0
 1020 FOR n2%=1 TO ScaleLength%
 1030 Sum2%=Sum2%+F2%(n1%,n2%)
 1040 NEXT n2%
 1050 IF Sum2%>0 THEN FOR n2=1 TO ScaleL
ength%:F2%(n1%,n2)=F2%(n1%,n2)*100/Sum2%
:NEXT n2
 1060 NEXT n1%
 1070
 1080 PRINT"Calculating Third Order Freq
uency..."
 1090 FOR n1%=1 TO ScaleLength%
 1100 FOR n2%=1 TO ScaleLength%
 1110 Sum3%=0
 1120 FOR n3%=1 TO ScaleLength%
 1130 Sum3%=Sum3%+F3%(n1%,n2%,n3%)
```

```
 1140 NEXT n3%
 1150 IF Sum3%>0 THEN FOR n3=1 TO ScaleL
ength%:F3%(n1%,n2%,n3)=F3%(n1%,n2%,n3)*1
00/Sum3%:NEXT n3
 1160 NEXT n2%
 1170 NEXT n1%
 1180 ENDPROC
 1190
 1200 DEF PROCPrint
 1210 PRINT'"Do You Want a Printout (Y/N
)?"
 1220 Ans$=GET$:IF Ans$="N" THEN PRINT:E
NDPROC ELSE IF Ans$<>"Y" THEN 1220
 1230
 1240 FOR n1%=1 TO ScaleLength%
 1250 IF F1%(n1%)>0 THEN PRINTMID$(Scale
2$,n1%*3,2);"...";F1%(n1%)
 1260 NEXT n1%
 1270
 1280 FOR n1%=1 TO ScaleLength%
 1290 FOR n2%=1 TO ScaleLength%
 1300 IF F2%(n1%,n2%)>0 THEN PRINTMID$(S
cale2$,n1%*3,2);"-";MID$(Scale2$,n2%*3,2
);"...";F2%(n1%,n2%)
 1310 NEXT n2%
 1320 NEXT n1%
 1330
 1340 FOR n1%=1 TO ScaleLength%
 1350 FOR n2%=1 TO ScaleLength%
 1360 FOR n3%=1 TO ScaleLength%
 1370 IF F3%(n1%,n2%,n3%)>0 THEN PRINTMI
D$(Scale2$,n1%*3,2);"-";MID$(Scale2$,n2%
*3,2);"-";MID$(Scale2$,n3%*3,2);"...";F3
%(n1%,n2%,n3%)
 1380 NEXT n3%
 1390 NEXT n2%
 1400 NEXT n1%
 1410 PRINT
 1420
 1430 ENDPROC
 1440
```

```
 1450 DEFPROCGetNote
 1460 Dice%=RND(100)
 1470 Note%=0:Sum%=0
 1480
 1490 REPEAT
 1500 Note%=Note%+1
 1510
 1520 REM In Case a Note Has Never
 1530 REM Followed a Particular
 1540 REM Sequence of Notes
 1550 IF Note%>ScaleLength% THEN Note%=R
ND(ScaleLength%):Sum%=Dice%
 1560
 1570 REM Third Order
 1580 IF Play%=51 THEN Sum%=Sum%+F3%(Pen
ult%,LastNote%,Note%)
 1590
 1600 REM Second Order
 1610 IF Play%=50 THEN Sum%=Sum%+F2%(Las
tNote%,Note%)
 1620
 1630 REM First Order
 1640 IF Play%=49 THEN Sum%=Sum%+F1%(Not
e%)
 1650 UNTIL Sum%>=Dice%
 1660
 1670 Note$=MID$(Scale2$,Note%*3,3)
 1680 Penult%=LastNote%:LastNote%=Note%
 1690 PRINTNote$;TAB(12)"Depth = ";CHR$P
lay%
 1700 ENDPROC
 1710
 1720 DEFPROCPlay
 1730 Position%=(INSTR(Scale$,Note$))/3
 1740 Note%=Key%+(Position%*4)
 1750 SOUND1,1,Note%,Dur%(D%)
 1760 D%=D%+1
 1770 IF D%>TuneLength% D%=0
 1780 ENDPROC
 1790
 1800 REM Ode To Joy - Beethoven
```

```
 1810 DATA E2,8,E2,8,F2,8,G2,8,G2,8,F2,8
,E2,8,D2,8,C2,8,C2,8,D2,8,E2,8,E2,12
 1820 DATA D2,4,D2,16,E2,8,E2,8,F2,8,G2,
8,G2,8,F2,8,E2,8,D2,8,C2,8,C2,8,D2,8
 1830 DATA E2,8,D2,12,C2,4,C2,16,D2,8,D2
,8,E2,8,C2,8,D2,8,E2,4,F2,4,E2,8,C2,8
 1840 DATA D2,8,E2,4,F2,4,E2,8,D2,8,C2,8
,D2,8,G1,16,E2,8,E2,8,F2,8,G2,8,G2,8
 1850 DATA F2,8,E2,8,D2,8,C2,8,C2,8,D2,8
,E2,8,D2,12,C2,4,C2,8,X,0
 1860
 1870 REM Jesu, Joy - Bach
 1880 DATA G1,6,A2,6,B2,6,D2,6,C2,6,C2,6
,E2,6,D2,6,D2,6,G2,6,F#2,6,G2,6,D2,6
 1890 DATA B2,6,G1,6,A2,6,B2,6,C2,6,D2,6
,E2,6,D2,6,C2,6,B2,6,A2,6,B2,G1,6
 1900 DATA F#1,6,G1,6,A2,6,D1,6,F#1,6,A2
,6,C2,6,B2,6,A2,6,B2,6,G1,6,A2,6,B2,6
 1910 DATA D2,6,C2,6,C2,6,E2,6,D2,6,D2,6
,82,6,F#2,6,G2,6,D2,6,B2,6,G1,6,A2,6
 1920 DATA B2,6,E1,6,D2,6,C2,6,B2,6,A2,6
,G1,6,D1,6,G1,6,F#1,6,G1,18,X,0
 1930
 1940 REM Irish Jig
 1950 DATA D2,3,B2,3,G1,3,G1,3,D1,3,G1,3
,G1,3,B2,3,G1,3,B2,3,D2,3,C2,3,B2,3
 1960 DATA C2,3,A2,3,A2,3,E1,3,A2,3,A2,3
,C2,3,A2,3,C2,3,E2,3,D2,3,C2,3,B2,3
 1970 DATA G1,3,G1,3,D1,3,G1,3,G1,3,B2,3
,G1,3,B2,3,D2,3,C2,3,B1,3,C2,3,B1,3
 1980 DATA C2,3,A2,3,D2,3,C2,3,B2,3,G1,3
,D2,3,G2,3,B3,3,A3,3,G2,3,F#2,3,D2,3
 1990 DATA F#2,3,F#2,3,D2,3,F#2,3,F#2,3,
D2,3,F#2,3,A3,3,G2,3,F#2,3,E2,3,G2,3
 2000 DATA G2,3,D2,3,G2,3,G2,3,C2,3,G2,3
,G2,3,B2,3,G2,3,G2,3,C2,3,B2,3,C2,3
 2010 DATA A2,3,D2,3,C2,3,B2,3,G1,3,G1,3
,G1,6,X,0
```

The program prompts you for input where required and will merrily trundle out a continuous composition.

Program notes

Note analysis is the type of operation ideally suited to a computer. First and second order analyses do not take very long and consume little memory, but when we move up to third order analysis we need to keep track of a fist of three consecutive elements. If we work with arrays this could mean, working with a three-octave range, dimensioning an array such as:

```
DIM F3%(36,36,36)
```

or larger, which uses up more memory than we can afford. One solution is to use byte arrays, as described in the User Guide page 237, along with indirection operators, as described in Chapter 39, which will save some memory.

With the aim of maintaining readability, I have developed another method which can still be used with byte arrays. It consists of calculating a new scale based upon the notes used in the tune and dimensioning the arrays often; we see how large they need to be, ie calculate their minimum size.

The range of available notes is put into Scale$ in line 120. If you enter other tunes, check that they do not contain notes outside this range or else alter Scale$ to suit.

Tune$ is dimensioned to hold the notes and Dur95 is dimensioned to hold the durations. F1% and, later, F2% and E3% hold the first, second and third order sequences.

PROCGetTune at line 320 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. Tunetength% is set to the number of notes read.

PROCNewScale at line 480 runs through the notes of the tune, comparing them with the ones in Scale$, and forms a new scale, Scale2$, consisting of the notes used in the tune. A new scale will typically contain about 12 notes which shows how much memory we are saving. The arrays for storing the second and third order sequences are then dimensioned in fine 660. The F1% array is cleared for further use.

PROCAnalyseTune at line 740 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 for a different purpose here, after being used to count the notes in PROCNewScale.

PROCCalcPercentges at line 880 calculates each order analysis one at a time. First, the number of times each sequence occurs in the tune, now resident in the F1%, F2% and F3'% arrays after PROCAnalyseTune, are added to find how many there are in total. 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 910 and 970. The

second and third order sequences use a series of nested loops to check every combination, but a percentage is only calculated if there is something there to calculate as in lines 1050 and 1150. This method reuses all the arrays, saving memory.

As we are using integer variables and arrays, you may wonder at the values which are going into the arrays to represent the percentages. Clearly such fines as 960 will not always produce an integer. The result is a set of figures which are not always 100% accurate. The integer variables and arrays reduce any figure to the next lowest integer. The overall inaccuracy will be very small and the sums of the contents of the arrays may not always add up to 100. This has a negligible effect on the composition and results in a faster program and a saving in memory.

You can alter these lines to produce true percentages or correctly rounded integers if you so wish. In the following formula:

B=INT (A 10^D)+0.5)/10^D

B will have the value of A to D decimal places.

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 original tune even with third order analysis.

PROCPrint at line 1200 is similar to PROCCalcPercentges except that 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 occurs after the other two 50% of the time. The following figures will show how the remaining 50% is split up.

Before it can start composing, the program needs two seed notes on which to base its first calculation. These are asked for by line 200, and line 210 asks if you want a first, second or third order composition.

PROCGetNote at line 1450 decides which note should be played according to their percentage chance of occurrence. Dice% represents a random percentage. Sum% is incremented in line 1580, 1610 or 1640, depending upon the depth of analysis required.

This procedure may require further explanation. 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, assume that the F2?£ array held the following:

F2%4LastNote%,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. Also each time round, Sum% has added to it the figure (percentage) held in the array.

If Dice% equals 72 the procedure 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 Note% is incremented by one 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%, and so the loop is exited.

Note$ is calculated from the value of Note%. Penult% and LastNote% are adjusted, the note name is printed and we move to PROCPlay.

PROCPlay at line 1720 works out the correct pitch in a similar manner to PROCAnalyseNote and PROCCalculatePitch in Program 10.1. You can include these procedures as data checks if you wish; none has been included in this program, although the conversion principles are the same.

D% in line 230 is used to keep track of the duration values held in Dur% array: once the program runs through the values it is set to 0 at line 1770 and starts again.

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.

The duration values are used to play back the compositions with exactly the same note lengths. 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 data included only analyses one tune 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 a new tune.

You could also try several rock 'n' roll tunes. These use the same basic chords and harmonic structure but have different melodies.

The program will produce the most interesting results when used in this way. When you are entering large amounts of data you will realise how important it is to save memory. If the tunes and data are very long you may have to increase the size of the arrays at fine 100.

As you already have the data for some tunes from Chapter 9, Rondo Alla Turca, etc, you could extract the data, remove the ampersand commands, renumber them, save as an ASCII file by *SPOOL and then * EXEC them into the program for analysis. You may have to alter Scale$ 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's rhythm pattern but not related to the melody notes.

A second method would be to analyse the notes and their respective durations togeth&, 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, etc. You can see how this would consume memory very quickly ag it is quite possible that each note may have four or more different durations in a piece. This, however, would fie the melody and rhythm together in a much more realistic way.

Taking the idea a step further, we could also include an analysis of the harmony indicated by the chord structure. Most modern songs change chords perhaps every bar or every four bars and this could easily be analysed.

If you decided to analyse rock 'n' roll tunes there would be no need to analyse the chords as all songs (for pedants - most of them) 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. All the chords are major chords although they are often played as sevenths (ie dominant seventh - C7 is formed from the notes C, E, G and A# and many of the melody notes will play around the seventh (ie in the case of C7, A#. 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, while we are analysing the notes and their durations, I suppose that we could also see what chord was playing underneath and analyse these three combinations as one item. That would be very interesting indeed, not terribly difficult to program but a little greedy of memory. I leave it to the virtuoso.

CHAPTER 11
More Programs that Compose

In the last 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 and it is easily achieved. To produce a harmony pleasing to our ears, however, is another matter altogether.

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 produced to fit over the top. Conversely, given a set of melody notes, any number of backing chords can be used to harmonise it. Just to illustrate the possible variety, each melody note could be given a different chord or a single note could be given two or more chords. The chords could change every bar, twice per bar or once every eight bars. Some of these combinations will sound distinctly unpleasant and others will be boring. The point is, anything goes, as long as it pleases someone, even if it's only the composer. Most music, however, follows a more agreeable (some might say predictable) harmonic structure.

The harmonic structure of popular songs

Most popular tunes rely heavily upon their chord progressions for their appeal, and some musicians can take any melody line and work out a chordal accompaniment for it. It may not be the same as the original, but it will be close enough for most people to recognise the tune. It is unlikely that two musicians will harmonise a tune in exactly the same way and no way different to that determined by the composer would be regarded as more right or wrong.

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 that would be the level of harmonic appreciation (or tolerance) the public had reached at the time. We are now musically more tolerant, and more complicated and dissonant harmonies are finding increasing acceptance. (When two or more tones are played together and produce a sound unpleasant to the ear, it is said to be dissonant. If the sound is pleasing it is said to be consonant. Dissonant harmonies generally refer to sounds such as those produced by playing two notes only a semitone apart.)

Jazz musicians specialise in taking a melody and/or a chord progression and improvising or spontaneously inventing new melodies or harmonies for that piece. Often, these will deliberately be obscure and quite removed from the original tune which 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 (who wrote 'Rule Britannia'.)

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. The programs in this chapter go a little further, I hope, and produce compositions in up to three voices which will be musically acceptable to all but the most critical pedant.

Judging by the problems we needed to overcome to generate even a single series of notes, you might imagine that the production of a two- or three-voiced composition would be two or three times as difficult. If we were to try to implement some standard, academic rules of harmony, that would indeed be the case. If we were to attempt the composition of a classical three-part canon, it would be even more difficult although the rules could be found and formulated much more rigidly.

Our first aim 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

After all this talk about compositions and rules, it is still difficult (if not impossible) to explain exactly what makes a piece of music good or bad, pleasant or unpleasant. It combines both order and disorder in various

proportions. Any compositional program needs a random element, otherwise it would simply be playing pre-programmed music which would not be at all original. Our problem is to find a way of controlling the amount of randomness applied to the choice of notes.

The quickest, easiest 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 very good results but not very original compositions.

Instant Mozart

Mozart is reputed to have devised a compositional system based upon throwing a dice (I refuse to use the word die). His method could easily be converted to a computer program and would produce spectacular results, but the catch is, you need to do a certain amount of composing yourself.

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, etc, until you had eight bars.

The initial problem is in composing 48 bars of music in such a way that any bar from column 2 can follow any bar from column 1, etc. No problem for Mozart but perhaps a little 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 very good. In this case though, you are doing all the composing, not the computer.

You can apply other rules and modifications to a compositional program so that, for example, it periodically inserts a previously-written series of notes into its otherwise random output but, again, you are usually the one composing the most interesting parts.

When we turn the compositional process over to the computer we need a way to control the amount of randomness it uses to make its selection of notes without imposing our own compositions upon it.

A three-part computer composition

As the computer does not know which combination of notes produces good results, our first step is to specify a list of permissible notes for it to choose from. Playing notes from this selection at random will still produce an unordered sequence of notes which could jump from one extreme of the range to the other. We need a way of softening the random factor.

Such an algorithm was devised some years ago by Richard Voss of IBM. For our purposes, we can liken the procedure to rolling a set of five

four-sided dice. We add the numbers shown to get our random number. If we always roll the five dice we will still have a series of totally random numbers, but if we usually roll one or two dice and only sometimes three, four or five we will get a series of numbers which vary from each other only slightly but which are still capable of producing a large change. This idea is implemented in Program 11.1.

```
   10 REM PROGRAM 11.1
   20 REM Computer Composition
   30 REM In 3-Part Harmony
   40
   50 MODE7
   60
   70 FOR X%=0 TO 1:PRINTTAB(0,X%)CHR$14
1;CHR$133;TAB(6)"3-PART HARMONY COMPOSIT
ION":NEXT X%
   80 PRINTTAB(0,2)CHR$131;"Channel 1";T
AB(13);"Channel 2";TAB(26);"Channel 3"
   90
  100 FOR Col=3 TO 24
  110 PRINTTAB(0,Col);CHR$(129+Col MOD 6
);
  120 NEXT Col
  130
  140 REM Set text Window
  150 VDU28,1,24,39,4
  160
  170 DIM Dice%(4),Key$(15)
  180 Key=1
  190 Tempo%=1
  200 Scale$="  C  C# D  D# E  F  F# G
G# A  A# B"
  210
  220 REM Chinese Sticks
  230 ENVELOPE1,1,0,0,0,0,0,0,126,-10,-5
,-2,126,100
  240
  250 REM Slow Attack
  260 ENVELOPE2,1,0,0,0,0,0,0,10,-1,0,-1
,126,100
  270
```

```
  280 REM Vibrato
  290 ENVELOPE3,4,0,1,0,2,1,0,126,-10,-4
,-4,126,100
  300 Env1=3:Env2=3:Env3=3
  310
  320 PROCSetScale
  330 PROCDuration
  340
  350 REPEAT
  360 IF ADVAL(-6)>0 PROCPlay1
  370 IF ADVAL(-7)>0 PROCPlay2
  380 IF ADVAL(-8)>0 PROCPlay3
  390 UNTIL FALSE
  400 END
  410
  420 DEF PROCSetScale
  430
  440 REM RESTORE To Required Scale
  450 RESTORE 530
  460
  470 FOR I%=0 TO 15
  480 READ Key$(I%)
  490 NEXT I%
  500 ENDPROC
  510
  520 REM Hornpipe
  530 DATA C2,E2,G2,C3,E3,G3,C4,G3,A3,F3
,D3,A2,F2,D2,F2,A2
  540
  550 REM Chinese
  560 DATA F#2,G#2,A#2,C#2,D#2,F#3,G#3,A
#3,C#3,D#3,F#2,G#2,A#2,C#2,D#2,F#2
  570
  580 REM Minor Key
  590 DATA A#3,D3,G2,C2,D#2,G2,C3,G#3,G3
,C4,D#3,G#3,C4,D3,D#3,G#3
  600
  610 REM Minstrel
  620 DATA B2,G2,D3,G2,A3,E3,C4,A2,C3,A2
,E3,C3,G3,D3,G3,B2
  630
```

```
  640 REM Minstrel 2
  650 DATA F3,A3,G3,C2,E3,G2,C2,G2,E2,C3
,A2,G3,C3,A#2,G2,F2
  660
  670 DEF PROCDuration
  680
  690 REM GOTO Required Durations
  700 REM See Text For Explanation
  710 GOTO 770
  720
  730 REM Fast (Chinese)
  740 X1=3:Y1=0:Dotted1=1:X2=1:Y2=2:Dott
ed2=9:X3=1:Y3=1:Dotted3=9:GOTO 820
  750
  760 REM Mad Hornpipe
  770 X1=1:Y1=1:Dotted1=9:X2=1:Y2=1:Dott
ed2=9:X3=1:Y3=1:Dotted3=9:GOTO 820
  780
  790 REM Minstrel
  800 X1=1:Y1=3:Dotted1=1:X2=2:Y2=3:Dott
ed2=1:X3=1:Y3=5:Dotted3=1:GOTO 820
  810
  820 ENDPROC
  830
  840 DEF PROCPlay1
  850 PROCRollDice
  860 PROCGetNote
  870 D=1:IF RND(16)<Dotted1 THEN D=1.5
  880 IF X1=1 THEN Dur%=2^Y1*D
  890 IF X1>1 THEN Dur%=2^(RND(X1)+Y1)*D
  900 SOUND1,Env1,Pitch%,Dur%*Tempo%
  910 PROCPrint(3)
  920 ENDPROC
  930
  940 DEF PROCPlay2
  950 PROCRollDice
  960 PROCGetNote
  970 D=1:IF RND(16)<Dotted2 THEN D=1.5
  980 IF X2=1 THEN Dur%=2^Y1*D
  990 IF X2>1 THEN Dur%=2^(RND(X1)+Y1)*D
 1000 SOUND2,Env2,Pitch%,Dur%*Tempo%
```

```
 1010 PROCPrint(15)
 1020 ENDPROC
 1030
 1040 DEF PROCPlay3
 1050 PROCRollDice
 1060 PROCGetNote
 1070 D=1:IF RND(16)<Dotted3 THEN D=1.5
 1080 IF X3=1 THEN Dur%=2^Y3*D
 1090 IF X3>1 THEN Dur%=2^(RND(X3)+Y3)*D
 1100 SOUND3,Env3,Pitch%-48,Dur%*Tempo%
 1110 PROCPrint(28)
 1120 ENDPROC
 1130
 1140 DEF PROCRollDice
 1150 Chance%=RND(100)
 1160
 1170 REM This Sets the Random Factor
 1180 IF Chance%<84 Roll=1 ELSE IF Chanc
e%<88 Roll=2 ELSE IF Chance%<92 Roll=3 E
LSE IF Chance%<96 ROLL=4 ELSE Roll=5
 1190
 1200 IF Roll=1 Dice%(0)=RND(4)-1 ELSE P
ROCRoll
 1210 PROCAddRolls
 1220 ENDPROC
 1230
 1240 DEF PROCRoll
 1250 FOR I%=0 TO Roll-1
 1260 Dice%(I%)=RND(4)-1
 1270 NEXT I%
 1280 ENDPROC
 1290
 1300 DEF PROCAddRolls
 1310 Note%=0
 1320 FOR I%=0 TO 4
 1330 Note%=Note%+Dice%(I%)
 1340 NEXT I%
 1350 ENDPROC
 1360
 1370 DEF PROCPrint(Tab)
 1380 PRINTTAB(Tab)Key$(Note%);"-";Dur%*
```

```
Tempo%;
 1390 ENDPROC
 1400
 1410 DEF PROCGetNote
 1420 IF LEN(Key$(Note%))=2 THEN NoteNam
e$=LEFT$(Key$(Note%),1) ELSE NoteName$=L
EFT$(Key$(Note%),2)
 1430 Octave%=VAL(RIGHT$(Key$(Note%),1))
 1440 PositionInScale%=(INSTR(Scale$,Not
eName$))/3
 1450 Pitch%=Key+PositionInScale%*4+(Oct
ave%-1)*48
 1460 IF Pitch%<0 OR Pitch%>255 PRINT"ER
ROR IN PITCH DATA ";Key$(Note%);" Pitch
= ";Pitch%:STOP
 1470 ENDPROC
```

When run, the program will compose music according to the various parameters set within it. These can all be altered and are explained below.

Program notes

The program up to line 150 sets a screen display which gives us something to look at while the computer is churning out its composition.

The array, Dice%, gives us five dice (0 to 4) to roll and Key$ stores 16 (0 to 15) notes. PROCSetScale initiates this and line 450 is used to set the data pointer to the scale of our choice. The notes fisted in lines 530 to 650 are sample scales for you to experiment with. You can alter these and add your own. The names are just for identification as the final output relies not only upon the scale but also upon the note durations and envelope parameters.

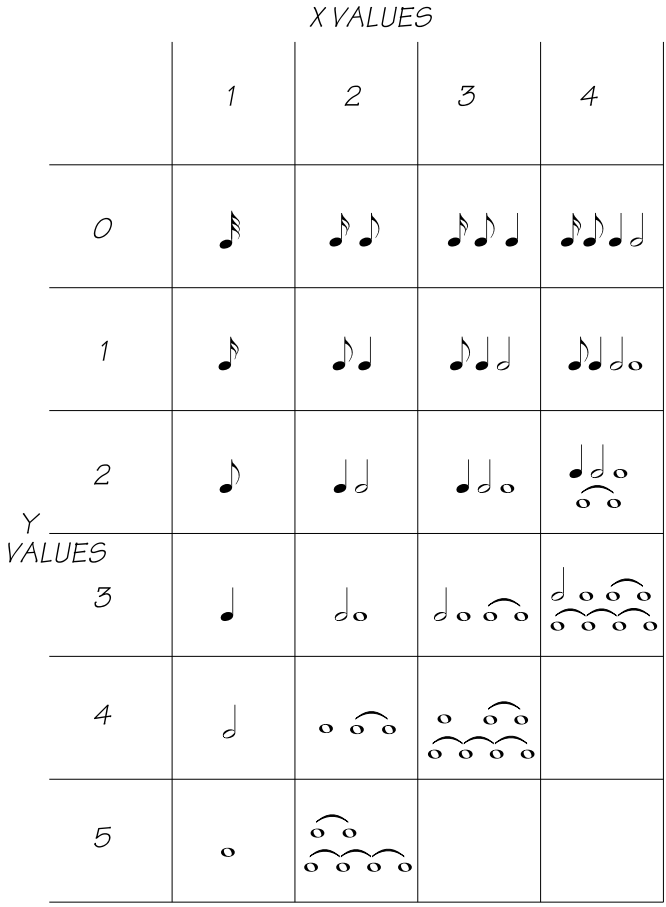
Line 300 allocates envelopes to each channel. Again, you can add to and alter each channel's envelopes as you wish.

Calculating the duration values

PROCDuration calls the procedure at fine 670 to determine the permitted set of duration values. Each channel has three duration parameters which are used to determine the range of duration values. As the operation of each channel's duration values is the same, we will only look at those of channel 1.

The permissible durations of a note are determined by the values of X1 and Y1. Cross-index them in **Figure 11.1** to see some of the possible values and ranges which can occur. The actual durations which do occur

are produced in the PROCPlay procedures. The other parameter, Dottedl, sets the probability that a note will be dotted, ie have its duration increased by half. If we look at lines 870 to 890 in PROCPlay1, we can examine a specific instance.



The probability that a note will be dotted is related to a random value between 1 and 16 generated in line 870. This seems to tie in well with music whose note values tend to be in multiples of 2, 4, 8 and 16 (like the binary system) but you can alter it to run on a straight percentage basis if you wish. If Dotted1 is greater than or equal to the random value, the next note will be dotted. If Dotted1 is set to 1 the notes will never be dotted, and if it is set to 16 they will always be dotted.

Lines 880 or 890 calculate the actual duration value depending upon the Xl and Y1 parameters. Refer to Figure 11.1 and try some examples and you will see how it works.

The REPEAT loop between lines 350 and 390 repeats the composition. The principle behind each channel's operation is the same. If there is room in the buffer, a new note is called through the PROCPlay procedures. As they are identical, we will only examine PROCPlay1.

The first thing it does is to call PROCRollDice at line 1140. This works on a percentage basis. You can alter line 1180 to determine how many 'dice' we roll and how often. The dice are assumed to be numbered 0 to 3, which will give us a value of from 0 to 15 to relate to the notes in Key$. Line 1200 will either roll one dice or call PROCRoll if more than one needs to be rolled. The dice are numbered 0 to 4.

PROCRoll at line 1240 runs through the required number of dice and allocates new random numbers to them in the range 0 to 3

PROCAddRolls at line 1300 runs through the Dice% array, adds up the total of the 'faces' and assigns it to the variable, Note'%. We are now back at line 850. The next fine calls PROCGetNote (at line 1410) with which you should now be familiar. Key$ is used to select our note.

PROCPrint at line 1370 prints out the note under the relevant channel heading and the process repeats.

Note line 1100 which plays channel 3 notes an octave lower than normal. This acts as a sort of bass which provides a tonal (providing the scale allows for a feeling of tonality) and rhythmic foundation.

Experimenting with the program

All the variable parameters and DATA statements can be altered and adjusted to produce an almost infinite range of sounds. Much of it will be music but with extreme settings, especially of the duration parameters, the results will be disjointed.

If you enter:

```
1155 Chance%=80
```

and then:

```
1155 Chance%=100
```

you will be able to hear and observe the results of rolling only one dice and then of rolling all five. You can add values in between, to simulate rolling any specific number. If you temporarily blank out PROCPrint, eg by adding:

```
1375 ENDPROC
```

and add:

```
1345 PRINT Note%
```

you will see the range of totals produced by PROCAddRolls.

The note selections from line 530 onwards are important, too. The notes should be chosen so that they don't create a severe dissonance when sounded together. In general, a cluster of semitones will not produce musical results, but even notes a tone apart (and the odd semitone) will blend when played together, if the duration values are not too long.

The duration values themselves need a lot of consideration. Mixing an extreme selection will produce music which has no rhythmic base and this can be very firing for western ears. A regular four-in-a-bar bass line, such as that provided by channel 3, can be used to provide the listener with some sort of rhythmic foundation, even though the other channels may be going their own way.

ENVELOPE 1, the scale data at line 560 and the duration. parameters at line 740 will produce quite a hypnotic composition with a very eastern flavour.

You can try extending the scale range and altering the severity of the random selection. You could give each channel its own PROCRollDice for finer tuning of the melodic output.

Using chords as a compositional base

Although the variety of compositions produced by Program 11.1 is quite large, the computer is restricted to a choice of only 16 notes (although you can increase this). This is not such a disadvantage, as many tunes have been written containing far fewer notes, but it does tend to root the compositions in a particular key. Sometimes, depending upon the notes, the key may seem to shift and then revert back•, but if we include unrelated accidentals in the scale we may hear too many dissonant sequences. Try it and see.

One way to broaden the output is to relate the choice of notes to particular chords and their relative constituent notes. Program 11.2 does this, and allows you to program the computer with any chord sequence and any chord type that you wish.

```
   10 REM PROGRAM 11.2
   20 REM Computer Composition
   30 REM Based on Chord Sequences
   40
   50 MODE7
   60
   70 Scale$="  C  C# D  D# E  F  F# G
G# A  A# B"
   80 ChordRange$="   M   7   9   min mi
n6min7min9maj6maj7aug dim"
   90
  100 DIM NotesToChooseFrom$(11,6)
```

```
  110
  120 REM RESTORE To Requied Data
  130 RESTORE 220
  140 READ NoOfChords%
  150
  160 DIM Melody$(NoOfChords%)
  170
  180 FOR Tune=1 TO NoOfChords%
  190 READ Melody$(Tune)
  200 NEXT Tune
  210
  220 DATA 12
  230 DATA C7,C7,C7,C7,F7,F7,C7,C7,G7,F7
,C7,G7
  240
  250 DATA 12
  260 DATA C7,C9,C7,C9,F7,F9,C7,C9,G9,F9
,C9,G7
  270
  280 DATA 32
  290 DATA Cmin,Cmin,G7,G7,G7,G7,Cmin
  300 DATA Cmin,Cmin,Cmin,G7,G7,G7,G7
  310 DATA Cmin,Cmin,Fmin,Fmin,Cmin,Cmin
  320 DATA G7,G7,Cmin,Cmin,Fmin,Fmin
  330 DATA Cmin,Cmin,G7,G7,Cmin,Cmin
  340
  350 DATA 16
  360 DATA Amin7,D7,Gmaj6,Emin6,Gmin9,C7
  370 DATA Fmaj7,Dmin6,Fmin7,G#min6
  380 DATA Gmaj6,D#M,Cmin6,D7,Bmin,E7
  390
  400 RESTORE 1470
  410 FOR N=1 TO 11
  420 FOR C=1 TO 6
  430 READ NotesToChooseFrom$(N,C)
  440 NEXT C
  450 NEXT N
  460
  470 ENVELOPE1,4,0,1,0,1,1,0,32,-2,0,-4
,126,100
  480 ENVELOPE2,1,0,0,0,0,0,0,126,-4,-1,
```

```
-4,126,100
  490 ENVELOPE3,4,0,0,0,0,0,0,106,-1,0,-
1,106,80
  500 Env1%=2:Env2%=2:Env3%=3
  510
  520 PRINT''
  530 PRINT"Please Enter Number of Beats
 in Bar"
  540 INPUT NoOfBeats%
  550 PRINT"Please Enter Tempo (2 or gre
ater)"
  560 INPUT Tempo1%
  570 Tempo%=Tempo1%
  580
  590 PRINT"Do You Want Rhythm Variation
 (Y/N)"
  600 INPUT Sync$
  610 IF Sync$="Y" OR Sync$="y" Sync=TRU
E ELSE Sync=FALSE
  620
  630 Comp=0
  640 REPEAT
  650
  660 Comp=Comp+1
  670 Col=128+(Comp MOD7)
  680 PRINTTAB(2,10)CHR$Col;CHR$141;"Com
posing Opus 1 Variation ";Comp
  690 PRINTTAB(2,11)CHR$Col;CHR$141;"Com
posing Opus 1 Variation ";Comp
  700
  710 FOR T%=1 TO NoOfChords%
  720 SyncPoint=RND(4)
  730 FOR Beat%=1 TO NoOfBeats%*2
  740
  750 IF Sync PROCSync
  760 PROCPlay
  770
  780 NEXT Beat%
  790 NEXT T%
  800 UNTIL FALSE
  810
```

```
  820 END
  830
  840 DEF PROCPlay
  850
  860 PROCAnalyseChord
  870 PROCGetNote(Note1$)
  880 PROCPlayChord(&101,Env1%)
  890 PROCGetNote(Note2$)
  900 PROCPlayChord(&102,Env2%)
  910 PROCBass(Key%+4)
  920
  930 ENDPROC
  940
  950 DEF PROCAnalyseChord
  960 Chord$=Melody$(T%)
  970 IF MID$(Chord$,2,1)="#" Key$=LEFT$
(Chord$,2):ChordType$=MID$(Chord$,3) ELS
E Key$=LEFT$(Chord$,1):ChordType$=MID$(C
hord$,2)
  980 Key%=1+((INSTR(Scale$,Key$))/3-1)*
4
  990 ChordNumber%=(INSTR(ChordRange$,Ch
ordType$))/4
 1000
 1010 Choice1%=RND(6)
 1020 REPEAT
 1030 Choice2%=RND(6)
 1040 UNTIL Choice2%<>Choice1%
 1050
 1060 Note1$=NotesToChooseFrom$(ChordNum
ber%,Choice1%)
 1070 Note2$=NotesToChooseFrom$(ChordNum
ber%,Choice2%)
 1080 ENDPROC
 1090
 1100 DEF PROCGetNote(Note$)
 1110 IF LEN(Note$)=2 THEN NoteName$=LEF
T$(Note$,1) ELSE NoteName$=LEFT$(Note$,2
)
 1120 Octave%=VAL(RIGHT$(Note$,1))
 1130 PositionInScale%=(INSTR(Scale$,Not
```

```
eName$))/3
 1140 Pitch%=Key%+PositionInScale%*4+(Oc
tave%-1)*48
 1150 IF Pitch%<0 OR Pitch%>255 PRINT"ER
ROR IN PITCH DATA ";Note$;" Pitch = ";Pi
tch%:STOP
 1160 ENDPROC
 1170
 1180 DEF PROCPlayChord(Chan%,Env%)
 1190 SOUNDChan%,Env%,Pitch%,Tempo%
 1200 ENDPROC
 1210
 1220 DEF PROCBass(Pit%)
 1230 IF ADVAL(-8)>0 SOUND3,Env3%,Pit%,T
empo%
 1240 ENDPROC
 1250
 1260 DEF PROCSync
 1270 REM GOTO Required Syncopation
 1280 ON SyncPoint GOTO 1300,1370,1400,1
310
 1290
 1300 IF Beat%=1 Tempo%=Tempo1%*1.5 ELSE
 IF Beat%=2 Tempo%=Tempo1%*.5 ELSE Tempo
%=Tempo1%
 1310 ENDPROC
 1320
 1330 REM Out Of Sync
 1340 IF Beat%=1 Tempo%=Tempo1%*2 ELSE I
F Beat%=2 Beat%=3 ELSE Tempo%=Tempo1%
 1350 ENDPROC
 1360
 1370 IF Beat%=NoOfBeats% OR Beat%=NoOfB
eats%+1 Tempo%=Tempo1%*.5 ELSE IF Beat%=
1 Tempo%=Tempo1%*2 ELSE Tempo%=Tempo1%
 1380 ENDPROC
 1390
 1400 IF Beat%=NoOfBeats% OR Beat%=NoOfB
eats%+1 OR Beat%=NoOfBeats%+2 OR Beat%=N
oOfBeats%+3 Tempo%=Tempo1%*.5:PROCPlay
 1410 ENDPROC
```

```
 1420
 1430 IF Beat% MOD 2=1 Tempo%=Tempo1%*.7
5 ELSE IF Beat% MOD 2=0 Tempo%=Tempo1%*.
25
 1440 ENDPROC
 1450
 1460 REM Major
 1470 DATA G1,C2,E2,G2,C3,E3
 1480
 1490 REM Seventh
 1500 DATA A#1,C2,E2,G2,A#2,C3
 1510
 1520 REM Major Ninth
 1530 DATA D2,E2,G2,A#2,C3,D3
 1540
 1550 REM Minor
 1560 DATA G1,C2,D#2,G2,C3,D#3
 1570
 1580 REM Minor 6th
 1590 DATA A1,C2,D#2,G2,A2,C3
 1600
 1610 REM Minor 7th
 1620 DATA A#1,C2,D#2,G2,A#2,C3
 1630
 1640 REM Minor Ninth
 1650 DATA D2,D#2,G2,A#2,C3,D3
 1660
 1670 REM Major 6th
 1680 DATA A1,C2,E2,G2,A2,C3
 1690
 1700 REM Major 7th
 1710 DATA B1,C2,E2,G2,B2,C3
 1720
 1730 REM Augmented
 1740 DATA G#1,C2,E2,G#2,C3,E3
 1750
 1760 REM Diminished
 1770 DATA D#2,F#2,A2,C3,D#3,F#3
```

The program will prompt for the number of beats in a bar, tempo, and

whether or not you want rhythmic variations. The compositions are based on a chord sequence resident in the DATA statements and the rhythm variations can be altered and ztdjusted within the program.

Program notes

ChordRange$ at line 80 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 1470 and relates to C chords. This information is adjusted for chords of other keys as we shall see. (See Chapter 2: 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 around octave 2 in Figure 2.4.

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 1650 have two ninths (D 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 NotesToChooseFrom$ array in fines 400 to 450.

The melody is derived from the chord sequence held in DATA statements beginning at line 220. The first figure is the number of chords which is read into the variable, NoOfChords%, at line 140. The remainder of the data are chords which are read into Melody$ at line 190.

The NoOfBeats%, asked for at line 540 represents the number of beats per bar. This is doubled in line 730 to produce a composition based around eighth notes or quavers. For example, an entry of four will produce bars containing eight quavers.

The tempo is input at line 560. The rhythmic variations are produced by altering this input value, so it is also stored in a second variable, Tempo%.

Lines 670 to 690 produce a small screen display.

The REPEAT loop running from 640 to 800 controls the tune production. A tune will play through each chord in Melody$ once: this is controlled by line 710. 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 the NoOfBeats% variable) multiplied by two.

If rhythmic variations have been selected, line 750 calls PROCSync (Sync for syncopation). Line 720 randomly selects one of four variations for one bar - unless it is selected again.

PROCPlay plays a single chord (comprised of three notes) of the tune and is called by fine 760. It has quite a lot of work to do, and tempo values of less than two will try to get hold of notes quicker than the BASIC program can supply them, causing uneven and hesitant results.

The first procedure called by PROCPlay is PROCAnalyseChord at line 950 which performs a similar analysis upon the chord to the one performed by our PROCAnalyseNote procedures 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. The key is calculated in the usual way at line 980, and line 990 gives us a chord number which simply shows how far along ChordRange$ it is. We then pick two different notes from the chord in fines 1010 to 1070 and end the procedure.

PROCGetNote at line 1100 is like the others in the book but it is called twice, once for each of the two notes we picked from the chord.

PROCPIayChord at line 1180 is called twice, too, and synchronizes the two notes so that they sound together.

The final act of PROCPlay is to call PROCBass at line 1220, which just sustains the root note of the chord, eg a C chord will produce a C bass note and a G# chord a G# bass note.

The last procedure is PROCSync at line 1260, which is only called if rhythmic variations have been asked for. If they have, this is called before every note to determine the duration - Tempo% as it is termed in this program. The variation in rhythm is produced by altering the value of Tempo%. according to the Beat% variable. Five examples are included and line 1280, along with line 720, selects the particular syncopation or variation for that bar.

Experimenting with the program

There are two main ways you can alter the output other than varying the tempo and beats per bar values input. The first is through the selection of chords used in the tune which determines the overall harmonic progression of the piece and the second involves the complexity of the rhythmic variations you introduce.

The data in lines 230 and 260 will play a 12 bar blues: the second set of data includes some ninth chords which produce a more jazzy feel. The data at fine 290 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 360 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 at the end of the program. Insert the chord name in ChordRange$ at line 80 and increase the NotesToChooseFrom$ array at line 100 and the N variable at line 410 to suit. Be careful just exactly where in ChordRange$ you put the new chord name, eg if min6 came before min the program would return an incorrect chord number for min at fine 990 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, such as C or G#, will usually stand by itself. In our notation these would be CM and G#M. Giving every chord a symbol in this way simplifies PROCAnalyseChord.

For more variation, increase the number of notes allocated to each chord. This can simply be used to increase the range so that notes occur over, say, two 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, CM would consist of a normal C scale. C7 would consist of an F major scale as it contains one flat, which is the key of F. Cmin would contain the notes of the D# (E flat) major scale. (Minor scales have been discussed in Chapter 2: you can mix harmonic and melodic scales as you wish.)

As the program picks notes from the chord at random, using complete scales may produce dissonant results. 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 four beats in the bar will produce music in 4/4 time and each chord will last for one bar. With a value of two, it will seem as if each chord lasts for half a bar, and so on. 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 that the first beat of the bar sounds louder than the others.

Adding rhythmic 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.

The variations are produced by PROCSync (assuming they are

requested) and a look at the first example, at line 1300, will show how they work.

According to certain criteria which you set up - in this case the note number contained in Beat% - the duratlon 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 1340 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. Too many out-of-sync rhythms may be too disorderly but one or two certainly add interest to a piece and are very effective.

If you look at the other examples in PROCSync, you will see how they work. The principle is easily adapted to produce almost any rhythm configuration you require. One aspect of the changes to be wary of is the eventual value of Tempo%. It will not hold a non-integer value. For example, the variation at fine 1430 will work with a Tempo1% value of four or eight but will otherwise produce strange results.

Instead of altering Tempo%, the variations could be stored in D AT A statements or in arrays. The above method, however, will adapt to whatever NoOfBeats% happens to be, although you could take this into account in any other system.

By setting more than one possible variation, which we have done, and by altering the sync pointer in line 720 and sync indicator in line 1280 the program will produce quite a varied output.

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 and could be altered so although they picked notes from the same chord, they would have independent durations. This would create a more truly polyphonic effect. The bass could also be given its own duration values and 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 add channel 0 as a drum track.

As I mentioned earlier, there is another way of choosing notes from a scale relevant 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. However, so that computer composition does not dominate this book, and as we already have several programs to experiment with, I shall only discuss the principles of this 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 have both been looking 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 11.2, you will see that the results are quite different to those produced when we limited the choice to six notes (which would probably only contain three or four different 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, eg in most man-made compositions the melody over a Cmin chord will probably contain more C, G and D# notes than any others. (This is not always true but it is a fair assumption and it makes a good starting point.) In the program, all notes would be selected indiscriminately. 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 Cmin might then look like this:

C2	20%
D2	5%
D#2	15%
F2	10%
G2	15%
G#2	10%
A#2	5%
C3	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 composition. 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. (This is, again, a generalisation but applies to most classical music, a lot of jazz and many popular and standard songs.) 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 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 be used only 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 such a program as that described, 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 a program like this would obviously be much improved if it played all three channels. As the two melody channels will be pursuing their own parts and as 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 downward melody while the other pursues an upward path.

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

Bass notes

The bass notes can be controlled in two ways. First, their durations can be fixed to play four in a bar, ie four crotchets (or three in a bar if the piece was in 3/4 time, etc) and, secondly, you can restrict their choice of note. As the purpose of a bass line is to support a harmonic framework, if it is

restricted to the root, fifth and possibly the third, the harmony will be reinforced. But by all means experiment.

Designing and developing programs

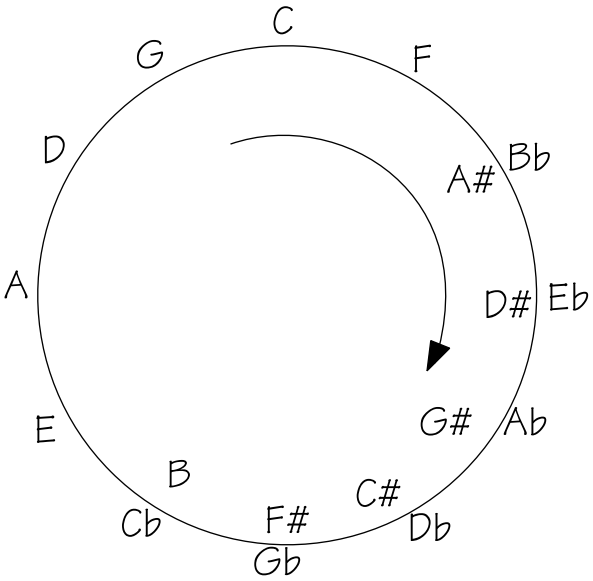
Within the framework of rules set out above, you have scope to apply your own ideas. One thing to be aware of is not to ask the computer to get a note through the stepping procedure that it has been forbidden to get through the scale table. For example, if it was currently playing a 62 and it was ordered to move down a step, it may generate a percentage on the scale table which said that F2 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 only with valid candidate notes to chose 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 simply as food for thought and as suggestions for further experiments. Detailed instructions and listings could quite easily consume the remaining pages of this 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 fantastic possibilities as it would allow anyone to affect the way the music was being 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 11.2**. Seventh chords, eg 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, an E7 will want to move to an A# (Bb) the seventh of which will want to move to a D# (Eb), etc. 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 BBC micro 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 (ie 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 you (or rather, the computer) start to produce results you like, you may want to save them. All the programs fisted so far, apart from Program 10.1, 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 BBC. Individual bars or phrases could be played again in any order you specify. The tune parameters would initially be stored in an array and later saved to tape or disk. (See the User Guide for details of file handling.)

The Amazing One Line Wonder 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 One Line Wonder, and reverses the pitch of the notes so that high notes will be played low and low notes will be played high. While it would be difficult to call the music produced by this example original (although what else can we call it?), the results are interesting and quite humorous. Other mathematical permutations will produce quite different results.

As we already have complete programs to play Mozart's Rondo, the Liberty Bell and the Dance of the Sugar-plum Fairy (from Chapter 9), 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 C4 and G#2, they would revolve around E3 which is half-way between the two. If you do not want to check through the music or DATA statements of a particular piece, you can find the highest and lowest notes by adding the following (to programs based on Program 9.2):

```
1 HiP=0:LoP=255
851 IF Pitch>HiP HiP=Pitch
852 IF Pitch<LoP LoP=Pitch
```

When the program has worked through the data, get the computer to print HiP and LoP in command mode and look up the notes in Figure 2.4.

The central point does not have to be exact and you can offset it a little but, especially in multi-part tunes, you will probably find that an exact central pitch will play in tune better than offset values. Here is the fine to insert in Rondo:

```
855 Pitch=77+77-Pitch
```

F#2 with a pitch value of 77 has been taken as the pivot point. Try also 73 and 81.

For the Dance of the Sugar-plum Fairy insert:

```
855 Pitch=117+117-Pitch
```

and this for the Liberty Bell:

```
855 Pitch=85+85-Pitch
```

Again, try plus and minus four on the central values.

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.

The above example is very simple. Try modifying the notes with SIN and COS functions or an algebraic expression. You could also reverse the sequence of notes so that the tune plays backwards, or alter the Pitch values so that the music plays in steps of two (quarter tone intervals) instead of steps of four (semitone intervals). This should produce music with an eastern flavour.

In the music business, if a singer has a hit song, the writer is often asked to produce a follow-up. This will sometimes be a re-hash of the original song, using basically the same chord progressions and melodic movements although it is unlikely that the writers ever use a computer. No doubt you can think of lots of hits and follow-up singles which were very similar.

Sing-a-long-a-matic

As speech ROMs and voice production systems increase in variety and versatility, the day may not be too far off when such voices can be programmed with a pitch to enable 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 BBC micro - 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.

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.

CHAPTER 12
Harmony and Transposition

The text books describe harmony as any combination of notes sounded simultaneously. For our purposes we will assume that this combination should produce agreeable sounds - it doesn't always!

The classical approach to and study of harmony is quite complex and beyond our scope. One of its main topics is counterpoint, which is when two or more melodic lines are played simultaneously. The canon or round is a form of counterpoint. An example is the song 'London's Burning', in which a number of voices sing exactly the same melody at staggered intervals throughout the piece.

Other examples of counterpoint include the invention and the fugue which were popular in classical music. J. S. Bach produced many brilliant examples of each. His most famous is probably his Toccata and Fugue in D minor, normally played on the organ and popularised, if that's the word, in horror films such as 'Phantom of the Opera'.

Although such studies are perhaps more relevant to modern day classical musicians and composers than to song writers and groups, many good arrangers and musicians have the ability to create interesting music with more than one melody fine. Many electronic music albums display such writing and it can also be found on some rock and progressive music albums.

The jazz musician turns harmonies and melodies upside down and the study of jazz harmony is a separate subject altogether.

Our main concern, at least in the initial stages of our musical experiments, is likely to be trying to decide what chord will fit a particular melody line and, having found the chord, how its harmony can best be brought out.

Harmonising a tune

When we put a tune into our computer, we are likely to be faced with one of two situations. We may have a melody line and a list of chord symbols or we may have a full-blooded piano or organ score with lots of notes and

secondary melody lines running through the piece. Each presents its own problems when we try to convert it for the computer.

With the first we wonder what to put in and with the second we wonder what to leave out. The best advice anyone can give is to do what sounds best; but of course we need a starting point

A melody with chord symbols: what to put in

As we have mentioned in other chapters, it is often a good idea to have a bass line running through the piece. This not only provides a good harmonic foundation but, as the notes are low, it gives the piece a sense of fullness.

As we are working with chords it is easy to use the notes of the chord in the bass. The root note is the note which gives the chord its name. This will quite definitely establish the key of the chord. For variation we can alternate the root with the fifth which should preferably be lower than the root but this is not essential. If the chord sequence; was C, Amin, Fmaj6, G? and the music was in 4/4 time and we wanted two bass notes in each bar they might look like this:

(C2 G1) (A1 El) (F1 C1) (Gl D1)

We can go on to add the third and, from there, we could add 'passing notes' which smooth out the bass line by avoiding large interval jumps. This can result in a walking bass often found in swing music such as that played in slow foxtrot or quickstep tempo.

This example is based on the above chord sequence and has four bass notes in each bar:

(C2 Gl C2 B1) (A1 E1 A1 Gl) (F1 D1 E1 Fl) (G1 G1 A1 B1)

The bass does not have to include every note in the chord and you don 't need to have a bass note on every beat. If the distance between one note and another is small, the bass itself forms a sort of melody line. Movement from one note to an adjacent note is known as stepwise movement and can generally be recommended. In this chord sequence: F, A, Dmin, C, F, this sort of movement would sound effective:

C1 C#1 D1 E1 F

Rather than construct a bass line as a separate entity, we can use the two other channels to provide an accompaniment to the melody. This was the basic premise behind the musical arrangements in the programs in Chapter 9. If you look again at Figure 9.1, you will see how channel 3 and channel 2 combine to produce the accompaniment.

Channel 2 actually has two roles. In bar 1 for example, first it plays a bass note then it joins channel 3 to play a chord. The overall accompaniment is rhythmic and is typical of a piano arrangement. There is no attempt to produce a separate bass line or a secondary melody line.

The Dance of the Sugar-plum Fairy follows a similar pattern. Only at the end does the upper melody hold on to a note while a lower line plays a downward run. Note the introduction which uses all three channels before the first one leaves to play the melody.

The Liberty Bell arrangement is also similar, but you can hear lower notes moving in upward and downward patterns to complement the main melody. This is simple counterpoint although it does not run right through the piece and you can hear how it adds another level of interest to the music.

Assuming you have decided upon a melody and basslaccompaniment line you still have another channel to arrange. If you can double up with the bass channel, as channel 2 does in Rondo, you effectively gain an extra voice. Bearing in mind the beneficial effects of stepwise movement you can proceed to fill in the gaps between melody and bass.

It is not easy to give hard and fast rules about allocation of the in between notes. Generally, any note of the chord will suffice and, with only three voices, it is usually better not to duplicate the melody or bass note unless it is as part of a rhythm accompaniment. If you are torn between one note and another in a chord, see if the previous note can move to this note by stepwise movement. Other than that, try out the alternatives and let your ears be the judges.

Working from a piano copy: what to leave out

It is sometimes more of a problem trying to decide what to leave out of an arrangement than it is to think of notes to put in. Many piano and organ arrangements have lots of stepwise movement in them, usually a good bass line and often a counter melody in places.

Again, we need a channel for the melody line. Allocation of notes to the other channels will depend upon the type of accompaniment required. If you can see two clear melody fines running through the piece, as would be evident in a fugue, etc, then obviously use these, and add the third channel either as a bass or in places between the melodies to reinforce the harmony. Which to choose should be apparent from the arrangement.

Pruning decisions must be made when a melody is supported by one or two other notes and when an accompaniment consists of a cluster of notes. In the former case, the best advice is usually to forget any supportive melody notes, concentrate on the melody itself and use the spare channel for the accompaniment.

A handful of notes m rue accompaniment section may not always have the root note of the chord in the bass. Simply using the lowest note

illustrated is not always enough. If the lowest notes form a stepwise progression, however, you can usually use them and let the third voice play the chord root.

If the chord is not named on the music and there are so many notes involved that the actual chord is difficult to work out, as a guide try placing the third voice closer to the accompaniment than to the melody. Otherwise, if you can work out what the basic chord is, you can discover which notes are part of the chord and will support the harmony, which notes are passing notes and which notes may be contributing to a little counterpoint or secondary melody line.

With only three channels it is not always possible to produce complicated harmonies and, if in doubt, the best idea is to stick to a melody line with chord notes in the bass and the third channel. You can alter them afterwards if they need adjusting. Your ears are your best guide but, initially, if you follow these general guidelines you will produce reasonable results.

Adding harmony to a melody line

You can see from the above that harmonising a tune requires a little thought. It would be nice if we could enter a melody line and let the computer work out a harmonic accompaniment for us. There are keyboards on the market which do just that. If you play a single note melody line, the keyboard will add another line to harmonise with it. The harmonic additions are controlled by a chip, of course, a computer dedicated to just that task.

The next program is presented tongue-in-cheek and adds a pseudo-harmony to a melody line.

```
    1 REM PROGRAM 12.1
    2 REM Pseudo Harmony Additions
    3 REM Insert In PROGRAM 9.1
    4
  300 PRINT Note$;TAB(6);Pitch;
  310 SOUND1,Env,Pitch,Dur
  311 PROCHarmonise
  315 SOUND2,Env,Pitch-P,Dur
  316 PRINTTAB(12);Pitch-P
  420
  430 DEF PROCHarmonise
  440 HarmPoint=RND(12)
  450 ON HarmPoint GOTO 460,470,470,470,
470,480,480,480,490,490,500,510
```

```
460 P=8:ENDPROC
470 P=12:ENDPROC
480 P=20:ENDPROC
490 P=28:ENDPROC
500 P=32:ENDPROC
510 P=48:ENDPROC
```

This adds a second 'melody' line to the single-channel version of Mozart's Rondo Alla Turca. You can see why I have called it pseudo-harmony by looking at PROCHarmonise at line 430. All it does is to add one of six different intervals to the melody fine. The result is, predictably, erratic.

We can reduce the random effect by restricting the intervals to, for example, thirds and fifths: we can take the randomness away altogether by playing a single interval, such as a third, throughout. We can do this by setting P equal to 12. The result sounds quite good but, as it lacks variation, it can soon become boring. Also, as the thirds are calculated on a semitone basis and not a scale basis, the result is a little mechanical and, at times, sounds out of key. For example, to add a third to the note E3 in this musical extract we should add C3. The program would add C#3.

The concept of a program which will harmonise any given melody line is intriguing, and although Program 12.1 harmonises on a purely random basis we can add rules to it as we did to Program 10.1. Harmonising a melody can be a creative act or it can be a mechanical process which gives us licence to produce something in between.

The construction of such a program would necessarily require a little knowledge of chords and harmony. So, rather than trying to delve too deeply into what is, perhaps, one of the most complex aspects of music, I will leave a few suggestions with which the more ambitious and musically knowledgeable programmer can experiment.

The addition of rules can be simplified in a number of ways. Restricting the melody to the key of C seems a reasonable first step and giving the computer a set of chords to work with could be the second. A list of chords might include C, C7, Dmin, Emin, F, Fmaj7, G, G7, and Amin. This could be expanded to include other chords slightly further harmonically removed from the key of C.

Rules could include using stepwise movement, either to move directly from one chord to another or as a bridge to span the notes in two different chords.

The Circle of Fifths could be used whenever possible, but further rules would have to govern movement around the Circle to avoid straying too far from the home key and to ensure that we ended up there at the end of the piece.

The application of some rules may require that the melody be scanned more than once so that the computer can make initial observations about the harmony of the piece before, for example, working out a stepwise fine.

Commercial instruments do not have this opportunity.

We could even use Program 12.1 as a starting point and add a routine to ensure that intervals were adjusted to fit the key of the piece or the scale used by the chord in use at the time. This would produce far better results than the mathematical intervals computed by the original program, as the key of the piece would not be disturbed.

Finally, if you want another tune to perform harmony experiments upon, the next program has extracted the DATA statements necessary to play a single line version of The Liberty Bell. These are for insertion in Program 12.1.

This will enable you to compare your program with the original one in Program 9.6.

```
  1 REM PROGRAM 12.2
  2 REM DATA Statements For
  3 REM Liberty Bell
  4 REM For Use in PROGRAM 12.1
  5
110 FOR N=1 TO 110
180 DATA C3,3
182 DATA A2,6,A2,3,A2,3,G#2,3,A2,3
184 DATA F3,6,C3,3,C3,6,A2,3
186 DATA A#2,6,A#2,3,A#2,6,C3,3
188 DATA D3,15,A#2,3
190 DATA G2,6,G2,3,G2,3,F#2,3,G2,3
192 DATA E3,6,D3,3,D3,6,A#2,3
194 DATA A2,6,A2,3,A2,6,A#2,3
196 DATA C3,15,C3,3
198 DATA A2,6,A2,3,A2,3,G#2,3,A2,3
200 DATA A3,6,F3,3,F3,6,C3,3
202 DATA B2,6,G3,3,G3,6,G3,3
204 DATA G3,15,F3,3
206 DATA E3,6,G3,3,G3,3,F#3,3,G3,3
208 DATA D3,6,G3,3,G3,3,F#3,3,G3,3
210 DATA C3,6,B2,3,C3,6,B2,3
212 DATA C3,9,C3,9
214 REM 2nd Part
216 DATA A2,3,G#2,3,A2,3,D3,6,C3,3
218 DATA A2,9,F2,9
220 DATA D2,9,G2,9
222 DATA F2,15,F2,3
224 DATA G2,3,A2,3,A#2,3,E3,6,D3,3
```

```
  226 DATA C3,9,F3,9
  228 DATA E3,9,D3,9
  230 DATA C3,15,C3,3
  232 DATA D3,6,D3,2,E3,1,D3,3,C#3,3,D3,
3
  234 DATA E3,9,E3,9
  236 DATA F3,6,F3,2,A3,1,G3,3,F3,3,G3,3
  238 DATA A3,15,A3,2,A3,1
  240 DATA G3,6,F3,3,D3,6,A#2,3
  242 DATA A2,9,F2,9
  244 DATA G2,9,E2,9
  246 DATA F2,12
  248
  250
  260
  270
```

Transposition

Transposition is an everyday occurrence in the music world, but it o-ften baffles new musicians. Like other musical disciplines, it requires a little study and thought but do not be put off by some of the more detailed discussions. Try the examples, run the program and refer back to the text when necessary.

Transposition is when a note or tune is played higher or lower than it is written. This normally results in a change of key, but if a tune is moved up or down a complete octave, although the key stays the same, this is still a transposition. You may have had to make such decisions in Chapter 9 when arranging tune data to play on the computer.

Referring to Figure 2.4, if the music shows a note written to correspond to, say, C4 on the keyboard, you may decide that you want the tune to be played lower. If you substitute C3 for the note, you are transposing it down an octave.

Most transpositions do not occur simply over a straight octave, but tend to move up or down within the octave. For example, playing E4 instead of C4 would be an upward transposition of two tones and a transposition from the key of C to the key of E. Although the two sets ofnotes maintain the same relationship to each other, eg Twelfth Street Rag is still recognisable as Twelfth Street Rag in whatever key it is played, the transposition is not quite so straightforward an operation to perform: we must take into account the difference in key signatures and any accidentals which occur.

Although the principle of transposition is easy to understand, the physical transfer on paper of notes from one key to another can cause

many problems for the less experienced musician. As you gain musical experience and grow} better able to recognise the difference between intervals - which are the same in whatever key you are playing - then transposition will become more instinctive and less of a mathematical chore.

Instant computer transposition

For convenience, we have included an instant transposer in most of our programs, namely the variable Key. It was originally introduced in Program 3.1. This can be increased or decreased to play the music in any key of our choice; the only restriction is that, if we try to lower the key too far, some of the pitch values produced may fall below 0. Note, this does not affect the original data in the computer, only the way the sound chip interprets it.

This demonstrates one of the many advantages of computers and digital information handling. As information is stored as a series of numbers it is easy to perform mathematical operations on the numbers to produce new values. This principle has been used in programs throughout this book, not least of all in the One Line Wonder Composer program of Chapter 11.

So, if the music is already in the computer, we can transpose it to any key simply by altering one variable but if we need a physical copy of the music there is little alternative other than to write it by hand. Even here, though, the computer can help us. Before describing how, let us see why music is transposed in the first place.

Why transpose?

Tunes are transposed for three reasons. First of all, although throughout this book we have only used the treble and bass clefs, there are other clefs. These are normally only found in orchestral scores, and most modem band arrangements only use one other clef, the Alto or C clef. It looks a little like the figure '3' preceded by a thick fine and a thin line. The middle of the '3' falls on the centre line of the stave, which in this clef represents middle C.

Alternative clefs are used for instruments whose ranges do not fit comfortably into the treble and bass clefs. The alto clef is used primarily for the viola. Unless you are involved with orchestral scores - or play the viola - you are unlikely to meet those other clefs: but if you do, you'll know what they are.

Many instruments are known as transposing instruments because they sound in a different key to that in which the music is written. Examples are the E flat Alto Saxophone, the B flat Tenor Saxophone, the B flat Clarinet and the B flat Trumpet. These names tell us what key the instruments play

in, but others such as the English horn, which sounds a fifth lower than the written music, give us no clues,

This is sometimes a difficult idea to grasp but again, unless you are involved with such instruments, you are unlikely to need to make use of such information. For the sake of completeness, however, here are two examples. Music for the E flat Alto Saxophone is written a. sixth higher than it sounds on the piano keyboard so the key signature is written a sixth higher than the concert key (ie the piano key) of the piece. Music in the key of F would be written in the key of D for the saxophone and an F note would be produced by writing a D note in this key. The B flat Trumpet is scored a tone higher so, if the piece was in the key of C, the trumpet's part would be written in the key of D.

These transpositions do not involve another clef. They are simply written in a different key and, when the instruments play them, they sound in the correct key- So, if you are writing a piece of music in a particular key and wish to include a part for a transposing instrument you will have to transpose its part to a different key.

These aspects of orchestral instruments are the first reason why music is transposed. The second reason involves altering a piece of music to fit a singer's vocal range. Quite often a piece of music in a certain key will be perfect for one singer but too high or low for another. This depends upon the vocal range of individual singers, which varies enormously from one to another. Professional singers, therefore, often need the services of a transposer who will write out a song accompaniment in the best key for their voice.

Incidentally, some less musical singers have been known to claim that they always sing in C or F or some other key. This, as I am sure you will realise, is nonsense as the range of a song depends not only upon the key but upon the highest and lowest notes of the melody.

The third reason for transposing a tune is simply because it sounds good. An upward key change of a semitone in a song gives the tune a special kind of lift and creates a musical excitement. Harmonically, a key change is often preceded by moving from the root chord to that of an augmented fifth and it sounds very effective. For example, to move from C to C# you would play the chords C, G#7, C#, The progression certainly gives the music a lift. The song 'Can't Smile Without You', popularised by Barry Manilow, contains three key changes which definitely add to its appeal.

Calculating a transposition

Transposition is a purely mechanical act. To transpose from one key to another, simply count the difference in semitones and apply this offset to the original notes. For example, to transpose from the key of A to the key of F, count how many semitones from one note to the other - in this case

four - and apply this offset to all other notes. In this way, D will become A# (Bb) and F will become C#, etc. You can use Figure 2.4 to help work this out.

Counting in semitones in this way automatically takes into account the accidentals, but we must know whether the notes are sharp, flat or natural before we begin. A C in the key of A is a C ,so we must make sure we count from that note. The key signature tells us whether a note is sharp or flat, but we must make the necessary adjustments ourselves for accidentals.

The computer as a transposition aid

Given that transposition may be desirable if not always necessary, how can we use the computer to help us? First thoughts might lead us to believe that, if we could feed in the original notes, the computer could be made to print out the new notes. In practice, however, it would probably take longer to enter the notes than it would to do the necessary calculations. Probably more useful would be a note comparison table which fists the notes in the old key against the notes in the new key.

Thirteen keys are fisted in Figure 2.5 (although F# and Gb are enharmonics, they are written differently) which gives us 156 different permutations. We can use our BBC micro to display the original key and the new key along with the note positions and the note names, and use this as a help sheet to aid conversion.

Line 70 contains an FX call to disable the ESCAPE key. Do not remove the REM until your program is fully debugged.

```
 10 REM PROGRAM 12.3
 20 REM Transposition Program
 30
 40 ON ERROR GOTO 2270
 50
 60 *TV255,1
 70 REM Disable ESCAPE key:*FX220,1
 80
 90 REM Switch on CAPS LOCK
100 *FX202,32
110
120 PROCSetUp
130
140 End=FALSE
150 MODE7
160 PROCMenu
170 IF End CLS:END
```

```
  180
  190 MODE4
  200 PROCWindows
  210 PROCDisplayScales
  220 GOTO150
  230 END
  240
  250 DEF PROCSetUp
  260 TClef=844:LClef=628
  270 TR$=CHR$130+"TRANSPOSITION PROGRAM
"
  280
  290 REM Treble Clef Characters
  300 VDU23,229,32,32,32,160,64,0,0,0
  310 VDU23,230,39,39,167,167,38,36,248,
240
  320 VDU23,231,228,230,102,51,24,12,7,3
  330 VDU23,232,160,32,32,32,120,164,38,
39
  340 VDU23,233,1,3,12,24,48,97,227,230
  350 VDU23,234,34,34,36,36,40,48,32,96
  360 VDU23,235,35,35,35,35,35,35,35,35
  370 VDU23,236,0,0,0,0,0,8,20,34
  380 TC$=CHR$9+CHR$236+CHR$8+CHR$10+CHR
$235+CHR$8+CHR$10+CHR$234+CHR$8+CHR$8+CH
R$10+CHR$233+CHR$232+CHR$8+CHR$8+CHR$10+
CHR$231+CHR$230+CHR$8+CHR$10+CHR$229
  390
  400 REM Bass Clef Characters
  410 VDU23,237,31,127,192,128,128,184,1
20,56
  420 VDU23,238,128,192,224,115,51,48,48
,48
  430 VDU23,239,48,51,51,48,48,32,64,128
  440 VDU23,240,1,2,4,8,16,32,0,0
  450 BC$=CHR$237+CHR$238+CHR$8+CHR$10+C
HR$239+CHR$8+CHR$8+CHR$10+CHR$240
  460
  470 REM Semibrieve
  480 VDU23,224,0,28,34,66,68,56,0,0
  490
```

```
  500 REM Crotchet
  510 VDU23,225,0,28,62,126,124,56,0,0
  520
  530 REM Natural
  540 VDU23,226,64,76,84,100,76,84,100,4
  550
  560 REM Flat
  570 VDU23,227,64,64,92,102,70,76,120,0
  580 F$=CHR$227
  590
  600 REM Sharp
  610 VDU23,238,36,36,126,36,36,126,36,3
6
  620 S$="#":REM S$=CHR$228
  630
  640 REM Offset for Lower Scale Notes
  650 DIM OffSetArray(12,12)
  660 RESTORE 840
  670 FOR TScale%=0 TO 12
  680 FOR LScale%=0 TO 12
  690 READ OffSetArray%(TScale%,LScale%)
  700 NEXT LScale%
  710 NEXT TScale%
  720
  730 DIM KeyArray$(12),MinKeyArray$(12)
  740 RESTORE 980
  750 FOR A=0 TO 12
  760 READ K$:IF A>7 K$=K$+CHR$227
  770 READ M$:IF A>10 M$=M$+CHR$227
  780 KeyArray$(A)=K$
  790 MinKeyArray$(A)=M$
  800 NEXT A
  810
  820 ENDPROC
  830
  840 DATA 0,3,-1,2,-2,1,-3,-3,1,-2,2,-1
,3
  850 DATA -3,0,3,-1,2,-2,1,1,-2,2,-1,3,
0
  860 DATA 1,-3,0,3,-1,2,-2,-2,2,-1,3,0,
-3
```

```
  870 DATA -2,1,-3,0,3,-1,2,2,-1,3,0,-3,
1
  880 DATA 2,-2,1,-3,0,3,-1,-1,3,0,-3,1,
-2
  890 DATA -1,2,-2,1,-3,0,3,3,0,-3,1,-2,
2
  900 DATA 3,-1,2,-2,1,-3,0,0,-3,1,-2,2,
-1
  910 DATA 3,-1,2,-2,1,-3,0,0,-3,1,-2,2,
-1
  920 DATA -1,2,-2,1,-3,0,3,3,0,-3,1,-2,
2
  930 DATA 2,-2,1,-3,0,3,-1,-1,3,0,-3,1,
-2
  940 DATA -2,1,-3,0,3,-1,2,2,-1,3,0,-3,
1
  950 DATA 1,-3,0,3,-1,2,-2,-2,2,-1,3,0,
-3
  960 DATA -3,0,3,-1,2,-2,1,1,-2,2,-1,3,
0
  970
  980 DATA C,A,G,E,D,B,A,F#,E,C#,B,G#,F#
,D#,F,D,B,G,E,C,A,F,D,B,G,E
  990
 1000 DEF PROCMenu:CLS:PROCdd(TR$,8,0)
 1010 PRINTTAB(3,3)CHR$131"Here are your
 options:"
 1020 PRINT'CHR$129"   1.."CHR$130"Trebl
e Clef."'CHR$129"   2.."CHR$130"Bass Cle
f."'CHR$129"   3.."CHR$130"End Program."
 1030 PRINT'CHR$131"   Please enter your
 choice ("CHR$129"1-3"CHR$131")?"CHR$129
;
 1040 PROCInput(48,52)
 1050 IF Key%=49 Clf%=1:NOSet=-36
 1060 IF Key%=50 Clf%=2:NOSet=-12
 1070 IF Key%=51 End=TRUE:ENDPROC
 1080
 1090 PROCGetSF("original"):OAc%=Key%-48
:Okey$=LEFT$(Ac$,1):IF Okey$="F" OAc%=OA
c%+6
```

```
 1100 PROCGetSF("new"):NAc%=Key%-48:Nkey
$=LEFT$(Ac$,1):IF Nkey$="F" NAc%=NAc%+6
 1110
 1120 OffSet%=OffSetArray%(OAc%,NAc%)*12
 1130
 1140 PRINT'TAB(12)CHR$136CHR$131"THANK
YOU":PROCD(200)
 1150 ENDPROC
 1160
 1170 REM Double Height Letters
 1180 DEF PROCdd(P$,a%,b%):FOR L%=0 TO 1
:PRINTTAB(a%,b%+L%)CHR$141;P$:NEXT:ENDPR
OC
 1190
 1200 DEF PROCInput(Min%,Max%):REPEAT:Ke
y%=GET:UNTIL Key%>Min% AND Key%<Max%:PRI
NTCHR$Key%:ENDPROC
 1210
 1220 DEF PROCGetSF(Key$)
 1230 PRINT'CHR$131"   Has the "Key$" ke
y any sharps"'CHR$131"   or flats ("CHR$
129"S, F or N"CHR$131")?"CHR$129;
 1240 REPEAT:Ans=GET:UNTIL (Ans=83 OR An
s=70 OR Ans=78):PRINTCHR$Ans:IF Ans=83 A
c$="Sharps" ELSE IF Ans=70 Ac$="Flats" E
LSEAc$="N"
 1250 IF Ans=78 Key%=48:ENDPROC
 1260
 1270 PRINT'CHR$131"How many ";Ac$;" has
 it? ("CHR$129"1-6"CHR$131")?"CHR$129;:P
OCInput(48,55)
 1280 ENDPROC
 1290
 1300 DEF PROCWindows
 1310 REM Set Graphics Window
 1320 VDU24,0;256;1279;1023;
 1330
 1340 REM Background Yellow
 1350 VDU19,129,131,0,0,0
 1360 GCOL0,129:CLG
 1370
```

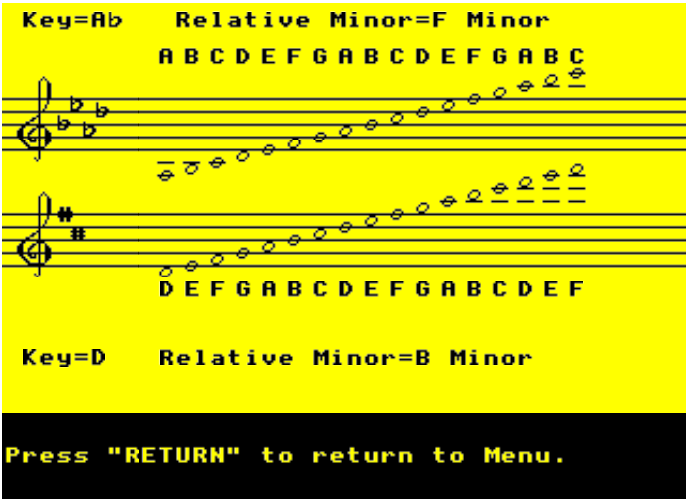
```
 1380 REM Foreground Black
 1390 GCOL0,0
 1400
 1410 REM Set Text Window
 1420 VDU28,030,39,24
 1430 ENDPROC
 1440
 1450 DEF PROCDisplayScales
 1460 PROCStave
 1470 Hinote%=904:Lonote%=712
 1480
 1490 Scal=1
 1500 PROCScale(Hinote%)
 1510 Hinote%=Hinote%-216-OffSet%:Lonote
%=Lonote%-216-OffSet%
 1520
 1530 Scal=2:NOSet=NOSet+48
 1540 PROCScale(Lonote%-64)
 1550
 1560 *FX15,1
 1570 PRINT'"Press ""RETURN"" to return
 to Menu. ";:PROCInput(12,14)
 1580 ENDPROC
 1590
 1600 REM Draw Staves
 1610 DEF PROCStave
 1620 PROCS(TClef):PROCS(LClef)
 1630 IF Clf%=1 PROCClef(TC$,TClef+62):P
ROCClef(TC$,LClef+62) ELSE PROCClef(BC$,
TClef):PROCClef(BC$,LClef)
 1640
 1650 PROCKeySig(Okey$,OAc%,TClef+14)
 1660 PROCGetKey(OAc%,TClef+160)
 1670 PROCKeySig(Nkey$,NAc%,LClef+16)
 1680 PROCGetKey(NAc%,LClef-256)
 1690
 1700 ENDPROC
 1710
 1720 REM Draw the Lines
 1730 REM Distance between a line and
 1740 REM space = 12
```

```
 1750 DEF PROCS(L%):FOR Line%=L% TO L%-9
6 STEP-24:MOVE0,Line%:DRAW1280,Line%:NEX
T:ENDPROC
 1760
 1770 REM Print Clef(s)
 1780 DEF PROCClef(C$,Pos%):VDU5:MOVE24,
Pos%:PRINTC$:VDU4:ENDPROC
 1790
 1800 DEF PROCD(Delay%):DL=TIME:REPEATUN
TILTIME>DL+Delay%:ENDPROC
 1810
 1820 REM Print Notes of Scale
 1830 DEF PROCScale(Pos%)
 1840 X%=240
 1850 Dif%=48
 1860
 1870 FOR Y%=Lonote% TO Hinote% STEP12:X
%=X%+Dif%:PROCNote("S"):N%=((Y%+NOSet)/1
2)MOD7 +65:VDU5:MOVEX%,Pos%+32:PRINTCHR$
N%:VDU4:NEXT
 1880 ENDPROC
 1890
 1900 DEF PROCNote(n$):VDU5
 1910 IF n$="S" n$=CHR$224
 1920 IF n$="C" n$=CHR$225
 1930 MOVEX%,Y%:PRINTn$
 1940
 1950 IF Y%>=TClef+36 PROCLeg(TClef+36,2
4)
 1960 IF Y%<=LClef-108 PROCLeg(LClef-108
,-24)
 1970 IF Scal=1:IF Y%<=TClef-108 PROCLeg
(TClef-108,-24)
 1980 IF Scal=2:IF Y%>=LClef-36 PROCLeg(
LClef-36,24)
 1990
 2000 ENDPROC
 2010
 2020 DEF PROCLeg(H%,Step%):FOR Le%=H% T
O Y% STEP Step%:MOVEX%,Le%-12:DRAWX%+28,
Le%-12:NEXT:ENDPROC
```

```
 2030
 2040 DEF PROCKeySig(SF$,Sig%,Y%)
 2050 IF SF$="S" n$=S$:Y%=Y%-2:RESTORE 2
230 ELSE IF SF$="F" n$=F$:Sig%=Sig%-6:Y%
=Y%-46:RESTORE 2240 ELSE ENDPROC
 2060
 2070 IF Clf%=2 Y%=Y%-24
 2080 X%=96
 2090 VDU5
 2100 FOR a%=1 TO Sig%
 2110 READ y%:MOVEX%,Y%+y%:PRINTn$
 2120 X%=X%+24
 2130 NEXT a%
 2140 VDU4
 2150 ENDPROC
 2160
 2170 DEF PROCGetKey(No,Pos)
 2180 MajKey$=KeyArray$(No)
 2190 MinKey$=MinKeyArray(No)+" Minor"
 2200 VDU5:MOVE32,Pos:PRINT"Key=";MajKey
$;"   Relative Minor=";MinKey$:VDU4
 2210 ENDPROC
 2220
 2230 DATA 0,-36,12,-24,-60,-12
 2240 DATA 0,36,-12,24,-24,12
 2250
 2260 REM ERROR Routine
 2270 REPORT:PRINTERR;" at line ";ERL
```

The program will ask if you want to use the treble or bass clef and then will ask for information about the original and new keys. It will print two staves, the clefs, the key signatures, the notes as they appear on the stave, and the note names. The notes are offset so that you can read down from the original key to find the note name and the position on the stave of the note in the new key.

From Program 12.3



Program notes

The program accepts single-key responses in upper case. Line 100 ensures the CAPS LOCK is on.

PROCSetUp at line 250 sets the relative positions of the top clef and the lower clef in line 270.

The next 350 lines create a set of user-definable characters for the clef signs and the notes. Although the notes are printed as semibreves, fine 510 defines a crotchet should you wish to use it, and line 540 defines a natural sign which you may find useful, although it is not used in the program. Line 610 defines a sharp sign but 620 selects the hash (#) for use as a sharp. Because of the small space between the two horizontal lines of the sharp sign, you will find this gap almost filled when it sits on a line and the hash seems to give better results even though it is heavier

OffsetArray% at line 650 holds the necessary information to offset the notes of the new scale against the old ones. The offset values are read from the DATA statements at line 840. The figures represent how many notes up or down we need to move to get from the old key name to the new key name. For example, cross-indexing D (two sharps) with F (one flat) will result in +2, meaning that we must move up two notes from D. Try this on Figure 2.4.

You will notice a pattern in the data which confirms some sort of mathematical relationship between the keys, although it is not easy to see exactly what the relationship is or to put it into a formula (there's a challenge - but no prizes). Lines 850 to 900 are the same as 960 to 910 (reading backwards).

The array cross-indexes each possible key against every other. The 0

subscripts represent no sharps or flats, ie the key of C. The subscripts l to 6 are the sharps and 7 to 12 are the flats.

The arrays, KeyArray$ and MinKey Array$, store the key names and their relative minor keys respectively.

Line 160 takes us to PROCMenu at line 1000. This prints the instructions then moves to PROCInput at line 1200. This runs through a REPEAT loop until a key is pressed within the range set by the calling parameters, in this case ASCII values greater than 48 and less than 52, which are the keys I, 2 and 3. Once a valid input is received, Clf% is set to indicate which clef is required, and NOSet is used to offset the notes depending upon whether the required clef is treble or bass.

PROCGetSF is then called at line 1220 to find out if the keys have any sharps or flats. It is called twice, once for the original key and once for the new key. If you indicate that there are no sharps or flats by pressing 'N' the procedure will exit at line 1250, otherwise Ac$ will have been filled with 'sharps' or 'flats' and will go on to ask how many there are.

Back at lines 1090 and 1100, OAc% and NAc% are set to register the number of sharps or flats in the key and, if flats are involved, 6 is added.

This enables the correct offset to be found in OffsetArray% at line 1130. Remember, fiats are accessed in OffsetArray% with subscript values of 7 to 12. The offset figure is multiplied by 12 so that it can be used directly when printing the new scale. See the REMs at lines 1730 and 1740.

From there, it's back to line 190, mode 4 and PROCWindows at line 1300. This is self-explanatory and you can refer to the User Guide for further information if necessary.

The last part of the main program is line 210 which calls PROCDisplaySca1es at line 1450. Its first job is to call PROCStave at line 1610. This prints the staves, the clefs and the key signatures.

PROCS at line 1750 is called twice with the top and lower stave positions which were allocated in line 260. This draws the five line staves.

PROCClef at fine 1780 draws the required clefs in their relevant positions.

PROCKeySig at line 2040 is called by lines 1650 and 1670 to print the key signatures. It uses information gained from Ac$ and passed to Okey$ and Nkey$ during PROCMenu to determine if the key contains sharps or flats. If so, positional adjustments are made to Y%, the vertical axis, to ensure that they are positioned correctly. The procedure is called with the number of sharps or flats in OAc'% or NAc% and a loop with this value is used to print the correct number. As sharps and flats are not placed on the stave in the same positions, the necessary offsets are held in DATA statements in lines 2230 and 2240 and RESTOREd according to the key being printed.

As we have not yet told the computer what keys are involved, other than by the number of sharps or flats in the key signature, this information is obtained from PROCGetKey at line 2170, which prints it on the screen beside the relevant scale.

Having drawn the staves, it's back to PROCDisplayScales to draw the notes. PROCSca1e does this at fine 1830. It is called with a value which tells it where to print the note names. Hinote% and Lonote% are set at line 1470 and these values are used as top and bottom limits on the notes being printed.

X% at line 1840 is the initial print position measured across the horizontal axis and Dif% is the difference between the notes.

PROCNote at fine 1900 is called with an 'S' for semibreve or a 'C' for crotchet, to determine which note to print. It is set to print semibreves but you can alter it. Having printed the note, fines 1950 to 1980 check to see if leger fines are required, and call PROCLeg at line 2020 if they are. There are four areas where leger lines will occur: above the top stave, below the top stave, above the lower stave and below the lower stave. Each is checked individually. The variable, Seal, is set at lines 1490 and 1530 and used in lines 1970 and 1980 to indicate if it is the top or lower scale being drawn.

PROCLeg is called with a position across the horizontal axis and a step value. This will be negative if the notes are below the stave. The lines are drawn and control passes back to PROCNote and back to PROCScale - the middle of fine 1870 to be exact - where N% is calculated.

N% is the ASCII code of the note name just printed and is calculated from the note offset variable, NOSet, given in line 1050 or 1060, and Y%, which is its vertical position on the stave.

Back to PROCDisplayScales and line 1510. Having printed the original scale, Offset% is used to adjust the values of Hinote% and Lonote%. NOSet is adjusted for the lower stave and the scale drawing procedure is repeated.

Line 1560 flushes the buffer so we don't accidentally fly back to the menu until we want to.

Using the program

The transpositions produced by the program are made in the direction which involves the least movement, eg if you enter F (one flat) as the original key and D (two sharps) as the new key the program will transpose down a third not up a sixth. If you are transposing for a singer or taking a song up a tone or semitone for a melodic keychange, this will be the most useful arrangement.

If you want or need to transpose in the opposite direction to the program, the variables held in OffSetArray% will have to be altered. This is easily done by removing '*12' from the end of line 1120 and adding another line:

```
1120 Offset%= OffSetArray%(OAc%,NAc%)
1125 OffSet%=(OffSet%-7*SGN(OffSet%))*12
```

This reverses the effect of OffSet% so -3 becomes -4, -2 becomes +5, etc. If you try this with a transposition which moves the scale too low, eg from A to Bb, you will overwrite the note names on the screen and will have to make adjustments to PROCNote to avoid it. This will only happen in extreme cases.

Because the offset is calculated from key names (ie letters), keys with the same letter in their name will not move, eg F and F# ,B and Bb, etc Effectively, in such instances all we need do is to replace the key signature (and alter accidentals) as the notes remain where they are on the stave. A transposition from A to Bb, however, although a similar transposition (ie up a semitone), will move because the note names do change.

If, for some reason, you wish to transpose between these non-moving keys in the opposite direction, eg from F downwards to F# or B upwards to Bb etc, you are really only moving the notes up or down an octave. This can be done by setting Offset% to plus or minus 7. If you seriously require such an exotic transposition, you are probably capable of doing it without the aid of a computer but, in any case, I will leave you to add the necessary modifications.

It would be easy to add another menu option to give you the choice of transposing up or down.

Accidentals

If the original key has no accidentals, the program will indicate the new notes exactly. Accidentals sometimes cause concern for the beginner. A set of rules to cover every eventuality would serve only to confuse, but I will see if I can pass on a hint or two to make it easier.

Normal sharps and flats are accounted for by the key signature, eg in the key of A with three sharps, a C note would be written just as it would in the key of C (or any other key): the fact that we play a C announced by the sharp sign in the key signature.

Sometimes you may see a note which has been sharpened or flattened by the key signature, with a sharp or flat in front of it for no apparent reason. This is sometimes done if the previous bar altered the note, or if a note of the same name but in a different octave in that bar has an accidental before it. Theoretically, such a sharp or flat is not necessary because, from our discussions in Chapter 2, we know that an accidental only applies to the bar in which it occurs and to the note on the line or space on which it occurs. Such sharps or flats are meant simply as reminders and are usually (but not always) enclosed in brackets to show this.

The best advice anyone can give to a relative newcomer to transposition is this - if in doubt, count as described earlier, using Figure 2.4 to pick out the notes.

Problems can arise because, if we sharpen or flatten some notes, we may get results like E# or Fb. For example, in transposing from C to A a

G# would appear as an E#. While not technically wrong, it would generally be written as an F natural.

If the note in question was Ab (enharmonically equivalent to G#) it would transpose as F# flat, which is a bit of a contradiction to say the least. You may be able to work out, however, that if we flatten a sharpened note it becomes a natural.

I find that the easiest way to approach accidentals is to see what the accidental is doing to the note. We already know what sharps and flats do. A natural will raise a note if the key contains flats or if the note has previously been flattened, and lower it if the key contains sharps or the note has previously been sharpened. Once we know if the accidental is raising or lowering the note we can do the same thing to our new note. In the last example we know the flat was lowering the note so lowering F $ will result in F natural. Keep in mind the key signature and apply the accidental operation to the true note rather than just the note name.

Like everything else, a little practice helps a lot.

Transposing chords

Chord names are easily converted because the chord type stays the same. If you run the program and input the two keys all you need to do is check the note names against each other. For example, in transposing from F# to Db, an chord prefixed with a G# becomes an Eb, C# becomes an Ab, etc. In these cases, you need to look at the key signature to see if the note referred to is a sharp, a natural or a flat. Then look at the lower stave and check the note against the key signature there.

As with accidentals, sometimes a little musical commonsense needs to be applied but in the case of chords we can lay down some rules. You will never see an E#, a B# , an F$ or an A$ chord. (A# is a legitimate note name and we use A# purely for the convenience of the programs. Any chord with this note as its root would be called Bb. You might see a Cb chord, which is the same as a B chord, but this only usually occurs if the key signature contains a lot of flats. Generally, if you have a choice between two chord names, one a sharp and one a flat, use whichever appears in the key signature.

The chord examples listed in Figure 2.11 are all in the key of C. If you are not too familiar with chord construction, you can use the program to help convert them to other keys.

Modifying the program

You may prefer to input the keys directly rather than as a number of sharps or flats. The program was designed to require input of a minimum of information. Reference to Figure 2.5 should clear up any confusion about

key signatures.

Continually referring to the screen during a transposition can be a little inconvenient and, until you gain some experience, you will find that you have to check the display constantly. You may find it more useful to have a hard copy of the display and, if you have a dot matrix printer, you could add a screen dump routine.

You could add an option to change the display from treble to bass clef and vice versa, without having to go back through the menu to repeat the same input information.

If you are doing transpositions for a singer, it will do no harm to remind them that musicians have to play the arrangements: avoid awkward keys if possible. That means keys with a lot of sharps or flats.

As a general rule, most keyboard players prefer flats and most guitarists prefer sharps - but that should not be your overriding concern. Although good musicians should be able to play in any key, good musicians may not always be available so it is often in the singer's best interests to keep it simple.

CHAPTER 13
The All-singing, All-dancing BBC Micro

The ideas and programs presented in this book have been developed using the BASIC language. Some musical applications are really only accessible through machine code and many of the programs could be developed beyond the bounds of BASIC by making use of the BBC micro's built-in assembler. It is hoped that, whatever your present level of computing ability, you will continue to experiment with the programs.

One such application which is best written in machine code is the production of music as a continuous background to whatever else the computer may be doing. The problem lies not in carrying out the normal sound and envelope functions but in supplying the sound chip with new note information. The system of interrupts described briefly in Appendix 1 will ensure that whatever information is stored in the sound queues will be carried out. The relative slowness of BASIC, however, sometimes leaves the sound generator waiting for a new note which causes a gap in the music production. You will notice such hiccups if you try to play a piece of music too fast, in some of the compositional programs for example. If we work at the same level as the interrupts it is easier to maintain control over information and pass on new note data.

Not to be outdone, however, we will see what we can do with BASIC.

Background music from BASIC

Background music is most evident in arcade-style games, which are usually written in machine code. With BASIC we can still achieve a certain degree of success in this area if we bear in mind two things which will set the limits on our music and our BASIC program. The optimum arrangement is for fairly long notes and fairly frequent access times between SOUND command updates. The balance between these will determine our success.

The overall limiting factor will depend upon the purpose of the BASIC program. If it is in the nature of an arcade game then the time taken by BASIC to update the SOUND command may slow down the game too

much. In such cases, rather man pray something fast and furious you could try interweaving notes on the channels by laying one sound over another to create a texture as opposed to a tune.

If the program does not require constant use of all the computer's facilities, the solution should be easier. You could program the computer to play a tune while waiting for an input to a utility program, etc. In such a case, the computer would have nothing else to do and you would not be stealing time from another operation.

If you use the negative ADVAL method to feed SOUND commands (which should at least ensure that your main program never grinds to a halt) you must also ensure that the channels never run dry for lack of note information: otherwise they may run out of time or out of sync. This may not be so important if you use only one channel, and this would be easier to control, too.

We have already used the principles just described, albeit in a very small way, when we printed out the notes the computer was playing in the programs in Chapters 10 and 11.

The ideas should be familiar to you now and, rather than illustrate the principle with another similar program, we will take a look at a slightly different aspect of the same thing involving sound and animation synchronization.

Cartoons

Cartoons are the ultimate in sound and visual synchronization. Every single action is accompanied by an over-exaggerated sound effect or snippet of music.

Sophisticated computers are already being used in cartoon studios to help draw the pictures. They are used to draw a sequence of character movements, eg walking or running. An artist will draw the first and last frames and the computer will fill in the others. This process is known as 'inbetweening' because it fills the gaps in between two pictures.

The calculation of the shape as one picture turns into the other is ideally suited to a computer and such programs can be duplicated on the BBC micro, although we will not be able to achieve the precision, speed or quality of a dedicated computer. It is quite possible for a program to illustrate a building crumbling to the ground or a man's outline dissolving into a heap or to show a square turning into a triangle.

Fascinating though this subject is, we are now encroaching upon computer graphics, which is not within our domain. We will still dabble a little in their territory, however, as we explore the next topic.

Sound and animation synchronization

If we have an animated display of a man jumping around the screen on a pogo stick it is quite easy to make the computer produce a 'boing'

whenever the pogo stick hits the ground. In the same way, it is easy to produce a bang when a gun is fired or a zap when a laser beam is fired. In these cases, the sounds are being synchronized to the animation. There are no problems because the animation is controlled from BASIC and does not run ahead as a series of SOUND commands will do,

If we try the opposite approach and attempt to synchronize a display (or anything else controlled directly from BASIC) with a series of SOUND commands, say a piece of music, then the music will tend to run ahead of the display. This is a result of the queues used by the sound generator.

However, operating solely from within BASIC, with a little care we can still achieve a fair degree of sound and animation synchronization. The principle behind this operation is to keep the sound queue as empty as possible so that each new command will be executed as soon as it is sent. In this way, as we send a SOUND command we can order a movement, so keeping the two close together. There are a few potential problems we need to be aware of which we will discuss later.

The following program illustrates how this can be done and is for insertion into Program 9.5. The additional fines are not very long and provide a good, if simple, demonstration of what is possible.

```
    1 REM PROGRAM 14.1
    2 REM Animated/Synchronised Dancer
    3 REM Insert in PROGRAM 9.5/9.2
    4
  195 MODE 5
  197 PROCFigures
  198 VDU23,1,0;0;0;0;
  634 x%=0:y%=10:m%=0:d%=0
  640
  650 Ch1=0:Ch2=0:Ch3=0
  680 IF ADVAL(-6)>14 AND Ch1<C1 Ch1=Ch1
+1:SOUNDChan1(1,Ch1)+1,Chan1(2,Ch1),Chan
1(3,Ch1),Chan1(4,Ch1)*Tempo:IF Ch1<82 OR
 Ch1>97 PROCCartoon
  690 IF ADVAL(-7)>14 AND Ch2<C2 Ch2=Ch2
+1:SOUNDChan2(1,Ch2)+2,Chan2(2,Ch2),Chan
2(3,Ch2),Chan2(4,Ch2)*Tempo:IF Ch2>69 AN
D Ch2<92 PROCCartoon
  725 VDU23,1,1,0;0;0;
 2000 DEF PROCFigures
 2010 VDU23,224,124,254,68,130,130,68,40
,100
 2020 VDU23,225,16,56,40,16,56,84,146,56
```

```
 2030 VDU23,227,40,40,40,40,40,0,0,0
 2040 VDU23,228,146,16,56,124,254,40,40,
40
 2050 VDU23,229,0,16,56,40,16,56,84,146
 2060 M1$="   "+CHR$8+CHR$8+CHR$8+CHR$10
+" "+CHR$225+" "+CHR$8+CHR$8+CHR$8+CHR$1
0+" "+CHR$224+" "
 2070 M2$=" "+CHR$229+CHR$8+CHR$8+CHR$10
+" "+CHR$228+" "+CHR$8+CHR$8+CHR$8+CHR$1
0+" "+CHR$227+" "
 2080
 2090 COLOUR 2
 2100 ENDPROC
 2110
 2120
 2130 DEF PROCCartoon
 2140 m%=m% EOR 1
 2150 IF m%=1 PRINTTAB(x%,y%)M2$ ELSE PR
INTTAB(x%,y%)M1$
 2160 IF x%>17 d%=1
 2170 IF x%<2 d%=0
 2180 IF d%=0 x%=x%+1 ELSE x%=x%-1
 2190 ENDPROC
```

Program notes

It is important that the animation be performed as quickly as possible, hence the use of short integer variables. In this short example, however, they are not essential.

PROCFigures at line 2000 designs two sets of figures which are stored in Ml$ and M2$. The putting together of figures containing more than one user-definable character is covered in the User Guide, Chapter 29.

PROCCartoon at line 2130 moves the figures around. The variables, x% and y%, are set at fine 634 and control the horizontal and vertical positions of the figure. m% is used to switch from Ml$ to M2$ using the EOR function at fine 2140. This has the same effect as writing:

IF m%=0 THEN m%=1 ELSE m%=0

EOR is one of a number of logical operators and is described in the User Guide on page 250.

d% is used to determine if the figure is to move left or right across the

screen and is set by the value of x%, in lines 2160 and 2170. Line 2180 adjusts the horizontal position.

PROCCartoon is called from fines 680 and 690. These are similar to the original lines in Programs 9.5 and 9.2 but, instead of keeping the queues full, new notes are only sent when required - as near as we can gauge with BASIC. This done by checking the state of the buffers with the negative ADVAL function. You will notice that, as we have not altered fine 700, channel 3 will still be topped up whenever there is space in its buffer, but the three remain in sync. In other pieces of music it may be necessary to adjust the sync parameters or the conditional statements which allow other notes to be sent.

Conditional statements have been added at the end of fines 680 and 690 which call PROCCartoon when a new note is sent to the SOUND command. Chl and Ch2 are used to take control of the figure at different points in the music.

The result is interesting, quite amusing and capable of much further development. The next program goes a step further and produces a more sophisticated presentation which is built around an animated character.

Hercules

The principles behind the following program are the same as those behind the previous one. It uses the routines in Program 9.2 and is for insertion into that program. It introduces another piece of music - Bizet's March of the Toreadors - and a more complex set of animated characters.

```
   10 REM PROGRAM 14.2
   20 REM "Hercules"
   30 REM Sound and Animation in Sync
   40 REM Insert in PROGRAM 9.2
   50
   90 C1=53:C2=47:C3=35
  250 ENVELOPE1,1,0,0,0,0,0,0,126,-4,-1,
-6,126,100
  260 ENVELOPE2,2,0,0,1,4,0,1,126,-3,0,-
6,126,100
  270 ENVELOPE3,3,0,8,-8,0,1,1,110,0,0,-
10,110,110
  272 ENVELOPE4,4,0,0,1,1,0,1,126,-8,-8,
-16,126,100
  274 ENVELOPE5,3,0,0,1,1,0,1,63,-8,-8,-
16,126,100
  276 ENVELOPE6,6,0,0,0,0,0,0,126,0,0,-1
```

```
,126,126
  278 ENVELOPE7,1,0,0,0,0,0,0,126,-10,-1
0,-10,126,0
  330 IF Note$="R" Env=0 ELSE IF (N=35 O
R N=42) Env=3 ELSE IF N<19 Env=1 ELSE En
v=2
  450 IF Note$="R" Env=0 ELSE Env=4
  570 IF Note$="R" Env=0 ELSE Env=5
  651 PROCMan
  652 MODE 5
  653 VDU23,1,0;0;0;0;
  654 m%=0:x%=8:y%=12
  655 COLOUR1
  656 PRINT'"Ladies and Gentlemen"
  657 PRINTTAB(5)"Presenting"
  658 PRINT''TAB(6)"HERCULES"
  659 SOUND0,6,4,40:SOUND1,0,0,180
  660 SOUND&1000,0,0,80:FOR S=1 TO 4:SOU
ND0,7,4,4:NEXT S
  661 COLOUR 2
  662
  670 REPEAT
  680 IF ADVAL(-6)>14 AND Ch1<C1 Ch1=Ch1
+1:PROCCartoon:SOUNDChan1(1,Ch1)+1,Chan1
(2,Ch1),Chan1(3,Ch1),Chan1(4,Ch1)*Tempo
  690 IF ADVAL(-7)>14 AND Ch2<C2 Ch2=Ch2
+1:SOUNDChan2(1,Ch2)+2,Chan2(2,Ch2),Chan
2(3,Ch2),Chan2(4,Ch2)*Tempo
  700 IF ADVAL(-8)>14 AND Ch3<C3 Ch3=Ch3
+1:SOUNDChan3(1,Ch3)+3,Chan3(2,Ch3),Chan
3(3,Ch3),Chan3(4,Ch3)*Tempo
  710 UNTIL Ch1=C1 AND Ch2=C2 AND Ch3=C3
  721 COLOUR1
  722 PRINTTAB(10,y%+5)"Thank You"
  723 PRINT'TAB(6)"...Signed HERC"
  724 VDU23,1,1;0;0;0;
  725 SOUND0,6,4,40
  880 REM Channel 1
  890 DATA &200,D3,8,E3,6,D3,2,B2,8,B2,8
  900 DATA &200,B2,6,A2,2,B2,6,C3,2,B2,1
6
```

```
  910 DATA &200,C3,8,A2,6,D3,2,B2,16
  920 DATA &200,G2,8,E2,6,A2,2,D2,16
  930 DATA &200,A2,20,E3,4,D3,4,C3,4
  940 DATA &200,B2,4,A2,4,B2,4,C3,4,B2,1
6
  950 DATA &200,F#2,8,B2,8,B2,8,A#2,6,C#
3,2
  960 DATA &200,F#3,32
  970 DATA &200,R,4,E3,4,D#3,4,E3,4,A2,4
,B2,4,C3,8
  980 DATA &200,R,4,B2,4,G2,4,E3,4,D3,16
  990 DATA &200,R,4,G2,4,D2,4,C3,4,B2,8,
A2,8
 1000 DATA &200,G2,16,R,16
 1010 REM Channel 2
 1020 DATA &200,G1,8,D1,8,G1,8,D1,8
 1030 DATA &200,G1,8,D1,8,G1,8,D1,8
 1040 DATA &200,A1,8,F#1,8,G1,8,F#1,8
 1050 DATA &200,E1,8,C#1,8,D1,8,F#1,8
 1060 DATA &200,A1,8,C2,8,A1,8,C2,8
 1070 DATA &200,G1,8,B1,8,E1,8,B1,8
 1080 DATA &200,F#1,8,B1,8,F#1,8,A#1,8
 1090 DATA &200,B1,8,F#1,8,D#1,8,B0,8
 1100 DATA &200,C1,8,A1,8,E1,8,A1,8
 1110 DATA &200,D1,8,G1,8,R,4,B1,4,G1,4,
E2,4
 1120 DATA &200,D2,8,R,16,F#1,8
 1130 DATA &200,G1,16,R,16
 1140 REM Channel 3
 1150 DATA &200,B1,8,R,8,B1,8,R,8
 1160 DATA &200,B1,8,R,8,B1,8,R,8
 1170 DATA &200,R,32
 1180 DATA &200,R,32
 1190 DATA &200,R,8,E2,8,R,8,E2,8
 1200 DATA &200,R,8,E2,8,R,8,E2,8
 1210 DATA &200,D2,8,R,8,C#2,8,R,8
 1220 DATA &200,D#2,8,R,24
 1230 DATA &200,R,8,C2,8,R,8,C2,8
 1240 DATA &200,R,8,B1,8,R,16
 1250 DATA &200,R,24,D1,8
 1260 DATA &200,B1,16,R,16
```

```
 1270
 1280 DEF PROCMan
 1290 VDU23,234,128,128,128,128,128,128,
128,255
 1300 VDU23,235,0,6,8,16,32,76,158,255
 1310 VDU23,236,24,60,90,255,90,36,24,25
5
 1320 VDU23,237,7,1,1,1,1,1,1,255
 1330 VDU23,238,0,96,16,8,4,50,121,255
 1340 VDU23,241,255,126,60,24,24,24,24,1
26
 1350 VDU23,242,24,24,24,24,24,24,24,126
 1360 VDU23,246,126,36,38,38,100,100,36,
231
 1370 VDU23,247,126,36,100,100,38,38,36,
231
 1380 Man1$=CHR$234+CHR$236+CHR$237+CHR$
8+CHR$8+CHR$10+CHR$241+CHR$8+CHR$10+CHR$
246
 1390 Man2$=CHR$234+CHR$236+CHR$238+CHR$
8+CHR$8+CHR$10+CHR$241+CHR$8+CHR$10+CHR$
247
 1400 Man3$=CHR$234+CHR$236+CHR$238+CHR$
8+CHR$8+CHR$10+CHR$242+CHR$8+CHR$10+CHR$
247
 1410 Man4$=CHR$234+CHR$236+CHR$237+CHR$
8+CHR$8+CHR$10+CHR$241+CHR$8+CHR$10+CHR$
247
 1420 Man5$=CHR$234+CHR$236+CHR$237+CHR$
8+CHR$8+CHR$10+CHR$242+CHR$8+CHR$10+CHR$
246
 1430 ENDPROC
 1440
 1450 DEF PROCCartoon
 1460 m%=m% EOR 1
 1470 PRINTTAB(x%,y%);
 1480 IF Ch1=1 PRINTMan4$:ENDPROC
 1490 IF Ch1=C1 Wait=INKEY(60):PRINTMan4
$:ENDPROC
 1500 IF Ch1<19 AND m%=0 PRINTMan1$:ENDP
ROC ELSE IF Ch1<19 AND m%=1 PRINTMan2$:E
```

```
NDPROC
 1510 IF Ch1<35 AND m%=0 PRINTMan4$:ENDP
ROC ELSE IF Ch1<35 AND m%=1 PRINTMan5$:E
NDPROC
 1520 IF m%=0 PRINTMan1$ ELSE IF m%=1 PR
INTMan3$
 1530 ENDPROC
```

Program notes

The salient points of the program were covered during examination of Programs 9.2 and 14.1, but we will have a closer look at some of its more important routines.

PROCMan at line 1280 constructs nine user-definable characters which are put together to form five different figures.

After the tune has been assembled into its arrays, fine 659 produces a round of applause. The second statement on this line ties up channel 1 until the applause is over to prevent the program running straight into the tune. It is interesting to realise that as the applause is sounding, channels 2 and 3 will already be waiting for channel I to empty so they can all synchronize and play.

The first statement in fine 660 incorporates a Hold parameter to let the applause die away slowly. The following statements produce the sound of the orchestra conductor rapping his baton on the podium four times.

The animation orders are taken only from' channel 1 and PROCCartoon does all the calculations regarding which figure to print. We will look at it in more detail.

PROCCartoon basically switches between two figures and the switches are activated by line 1460 as they were in Program 14.1. The switching occurs between three sets of two figures, and the remaining fines of the procedure are used to determine which set of figures to use.

Line 1470 sets the print position for the figures. Lines 1480 and 1490 print a stationary figure at the very start and end of the program.

As the program moves through the procedure, line 1500 picks up the first few bars and prints Man1$ and Man2$. When Ch1 reaches 19, line 1510 takes over and prints Man4$ and Man5$. When Ch1 reaches 35, control passes through to line 1520 which prints Man1$ and Man3$.

And so, at line 1490, when Ch1 is equal to C1, the last figure is printed and control reaches line 725 which ends the program with another tumultuous round of applause.

Further experiments in animation

Although we must be careful not to over-extend the capacity of BASIC to (apparently) control more than one sequence of events at a time, these programs have by no means overtaxed its abilities. BBC BASIC is very

fast and you could expand these ideas into more complex and complicated graphic routines - bearing in mind the restrictions of the system.

It should be possible, for example, to print more than one Hercules on the screen during PROCCartoon. Alternatively, you could design a larger figure or a multi-coloured figure made up from different coloured user-definable characters superimposed on each other. The latter is only possible in a graphics mode and would require some modifications to the printing procedure, but the overall effect would be suitably impressive.

Computer art

Other than direct animation, you could generate a pattern triggered at suitable points by the music. This could include starbursts, radiating lines, colour changes (use VDU 19), the printing of various geometrical shapes or pre-defined characters. If a programmed piece of music was playing - as opposed to a computer-generated piece - to avoid repetition the graphic designs could be selected at random. If we were running a composition program, the combination of computer music and computer-generated art would prove a very interesting spectacle. To begin with, you could add a screen display to many of the programs in this book.

Tomorrow's BBC micro

As you know, the BBC micro was designed to be capable of expansion. Aiready the use of disk drives, plug-in EPROMs and the interface capabilities of the user port and 1MHz bus give us a potentially very powerful machine. When the Tube£ and second processor are in common use, its power will increase enormously.

Not only from a computing but from a musical point of view, technology can make so many more things possible. If manufacturers take advantage of the BBC micro's expansion facilities, we can expect to see plug-in piano type keyboards and add-on synthesizer voice modules. Sound sampling devices will be able to listen to a sound and let the user play it back at any pitch - a facility at present only available on synthesisers costing several thousands of pounds.

The ever-increasing speed and versatility of the microcomputer will secure it a place in all areas of music from sound generation and production to recording and playback. Many recording studios already rely upon computers and the trend is likely to continue. There is no reason why one day a BBC micro should not be performing similar functions.

In this world of rapidly developing technology, after tomorrow - who knows?

APPENDIX 1
The Hardware and the Software

The sound chip in the BBC micro is the Texas SN76489, which is similar in specification and performance to those in many other micros. This might lead one to expect similar abilities, but the BBC micro's sound system is more powerful than other micros by virtue of the software it uses to drive the chip.

The OS (Operating System) is a machine code program roughly 16K in size which is resident in ROM inside the computer. Its overall purpose is to aid the running of high level languages such as BASIC and to provide a kind of buffer between the user and the rather unfriendly world of the CPU (Central Processing Unit). It looks after a host of jobs and, among other things, it provides software to handle the sound generator. It makes extensive use of interrupts to improve the general performance of the machine and also, of special interest to us, the sound chip.

Normally, a computer can only perform one action at a time. It could not, for example, calculate 2+2 at the same time as it was calculating 3+3. If these problems were programmed into a computer, the answers would appear instantaneously, but the computer would have worked its way through the program, one item of information at a time. So, theoretically, it could not make an explosive sound at the same time as it showered pieces of bomb around the screen. In practice, we know it can do exactly that and the reason lies in the OS. By using interrupts and a system of queues, the BBC micro can execute a sound command and apparently do something else at the same time.

Interrupts

It is not necessary to know exactly what's happening to make use of the system, but you may find the information useful at a later date during development of one of your programs.

An interrupt is a means of switching the attention of the CPU from one task to another and back again. For example, the pseudo-variable, TIME, is updated every 1/100th of a second by an interrupt from the VIA (Versatile Interface Adaptor). Whatever task the computer is about, it stops to increment the value of TIME and then continues from where it left off.

Pressing a key on the keyboard causes an interrupt to be sent to a keyboard service routine in the OS. This happens even when a program is running. The routine stores the ASCII value of the key in the keyboard buffer and then goes back to whatever task was in hand before it was interrupted.

Run the following and press some keys while it is running:

```
10 FOR X=1 TO 10000
20 NEXT X
```

When the delay loop has finished, you will see the characters you typed appear on the screen. Now type and run this:

```
10 SOUND1,-15,53,254
```

If you press some keys while this is sounding, they will immediately appear on the screen. This is because the sound chip is controlled by interrupts. Whenever the computer meets a SOUND command an interrupt directs the system to produce the required sound and immediately reverts back to its previous task. In this case it had nothing to do other than revert to command mode and you will notice the cursor prompt appear on the screen as soon as you hit RETURN.

The sound chip itself has no means of controlling the duration of a note and it will continue to sound until a software command turns it off. The computer uses the timer interrupt to check on the duration of a note. If the tone has sounded for its allotted span, it is switched off.

The ENVELOPE command relies on interrupts, too. The first parameter in the ENVELOPE command, after the envelope number, is a 'step' value. The other parameters are described as a number of changes in pitch or amplitude during each step. The step value is specified in multiples of 1/100ths of a second, which ties in very nicely with the timer interrupt. When the TIME variable is incremented, the computer checks to see if any of the envelope parameters need adjusting: if they do, the sound chip is updated with the new values.

Normally, we will not deal directly with the BBC micro's system of interrupts - this is more the domain of the assembly language programmer and needs to be handled with care - but the beauty of the BBC micro's sound system is that we have access to all the necessary functions through the BASIC language. In fact, all sounds and effects are created with only two commands - SOUND and ENVELOPE.

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. If we were only as logical and methodical as the computer! If it hasn't happened to you yet, let's hope it never does. 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 twice before running it. If you are fortunate enough to have a disk drive, this is no problem, but I know how laborious this can be with a cassette recorder. 99 times out of 100 you will probably have no problem - the BBC micro is quite tolerant of the idiosyncrasies of most recorders - but the 100th time after you've had 99 successful saves could just be the time something goes wrong.

Using a cassette recorder

When using cassettes, I find that the best thing I can do to ensure trouble-free loading and saving is to clean the recording and play-back heads regularly. Cassettes, cheap ones especially, shed their coating on to the heads, which impairs the signal and makes secure saving a problem.

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.

Verify command

The BBC micro does not have a VERIFY command to allow you to check that what you have on tape is the same as what you have in the computer. The *CAT command is not quite the same and programs which have *CATed have, sometimes, refused to load. A better method is to use the following:

```
*LOAD "program"8000
```

where 'program' is the file name. With cassettes the name can be omitted altogether.

This attempts to load the program into the ROM area (see the memory map on page 500 of the User Guide) which, of course, it cannot do: it runs through the normal loading procedures and will report any loading errors more efficiently than the *CAT command. It will not interfere with the program already in memory so, if any errors are reported, you can SAVE it again. It does not check the program byte for byte with the program in memory, but it is the closest to a VERIFY command we have, without writing a special routine, and it is very reliable.

Entering programs

The programs have been fisted directly from the computer at a print width of 40 characters per line - the number of characters per line on a mode 7 screen. This will help when checking your programs against the listings.

You will probably be aware of the programmable function keys, described in the User Guide in Chapter 25. Here is a short program to initialise the keys before starting work on a program. Some of the routines are only applicable to cassettes, but most users who have worked up to disk drives will have discovered their own favourites.

```
10 REM PROGRAM X2.1
20 REM Function Key SetUp
30
40 *KEY0RUN|M
50
60 REM CAT
70 *KEY1*.|M
80
90 *KEY2LOAD""|M
```

```
  100
  110 *KEY3AUTO
  120
  130 REM MODE7, Paging Mode On, List
  140 *KEY4MO.7|MV.14|ML.|M
  150
  160 REM Page Mode Off
  170 *KEY5V.5|M
  180
  190 REM VERIFY
  200 *KEY6*LOAD""8000|M
  210
  220 REM Print Program Length
  230 *KEY7P.LOM.-PA.|M
  240
  250 REM Print Remaining Memory
  260 *KEY8DIM P%-1:P.H.-P%|M
  270
  280 REM OLD, Page Mode Off, List
  290 *KEY9O.|MV.15|ML.|M
  300
  310 FORX%=0TO10:PRINT"SOFT KEYS SET":N
EXT
  320 END
```

A very useful facility available from OS 1.0 onwards is the ability to print teletext characters directly on to the screen. For example, pressing SHIFT and F4 simultaneously is the equivalent of PRINTing CHR$132 and produces a blue alpha-numeric character. As well as saving memory, this technique makes it harder to enter and edit menu and instruction pages, etc, and is worth cultivating.

As these codes will not fist to a printer in the usual way, they have been written out in full in the programs in this book. For further details see the User Guide page 439. Incidentally, the OSBYTE call mentioned there is incorrect. It should read A=&E2 (226).

I have tried to use meaningful variable names throughout the program and have restricted my use of integer variables (see the User Guide page 65) simply to aid readability. Many programs could be shortened and speeded up by using integer variables and shorter variable names especially where the program performs calculations before sending data to the SOUND command.

I have also refrained from using indirection operators (see the User

Guide Chapter 39) for which I may be criticised. However, my aim was always that the reader should understand how the programs work and beginners may well be confused by such 'heavy' material. Those more expert will, I hope, let loose all the tricks of the trade as they experiment with the program.

Merging programs

Many programs fisted in this book use the same or similar routines and to save time and effort some of the programs have been designed to be inserted directly into other programs. These programs will not run by themselves.

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 merged with the other programs using * SPOOL and *EXEC as described in the User Guide on page 402. It is essential that every single line be entered, including the blank ones which are sometimes there to delete unwanted lines from the original program.

Alternatively, you could LOAD the main program and type in the new fines, but be very careful of entry errors.

For disk users, I would recommend the first option but if cassette users check the fine numbers carefully the second method may be quicker.

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 the RND function with a negative number in brackets such as:

```
X=RND (-2)
```

Details are included in the User Guide on pages 84 and 342.

This can be useful if you want to check a program's output.

Other titles from Sunshine

SPECTRUM BOOKS

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

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

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

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

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

COMMODORE 64 BOOKS

Mathematics for the Commodore 64
Czes Kosniowski
ISBN 0 946408 149 **£5.95**

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

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

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

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

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

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

ELECTRON BOOKS

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

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

BBC COMPUTER BOOKS

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

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

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

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