Chapter 5 **Advanced uses of sound**

In this chapter we look at two of the more advanced uses of the SOUND statement. We first of all consider how to synchronise the playing of two and three part melodies. This is a sequential processing problem and similar problems often occur in computer science, for example, in operating systems.

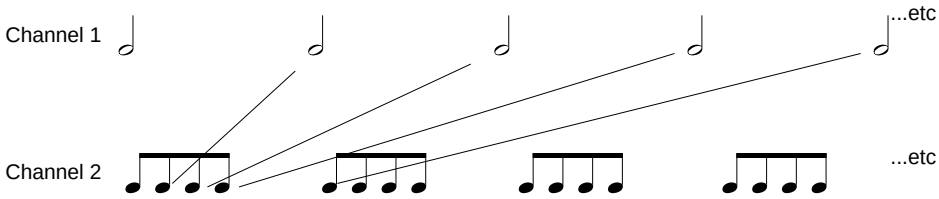
We then look at the intriguing business of getting the computer to generate or compose music. There are two aspects to this. First of all we can get the computer to imitate music of a particular style by supplying it with a sufficient number of examples of tunes in the style that we require to mimic. Secondly we can get the computer to compose its own original music, by using a constrained random number generator. Both of these methods will provide pleasing results if sufficient thought is put into the programming techniques.

5.1 Playing a two-voice melody

Although careful use of the ENVELOPE statement can produce moderately pleasing effects with a single voice (single SOUND statement), it is always obvious that the sound is generated by a fairly basic synthesiser. A lot of the resulting musical inadequacies can be overcome by using 2 or 3 voices or sound channels simultaneously. Before we can do this there are queuing and synchronisation problems that have to be overcome.

Syncrhonisation of two voices

Consider playing melodies simultaneously from parallel arrays or separate data streams containing, for each melody line, a pitch and duration value. We could fetch elements alternately from each melody array and send them alternately to two sound channels. A queuing problem arises whenever notes of different durations appear at corresponding points in each melody line - the usual situation in musical arrangements. To start with we'll consider the problem with 2 voices or channels. The following example should make things clear. A sequence of 4 minims is to be initiated in one channel at the same time as a series of quavers in the other channel:



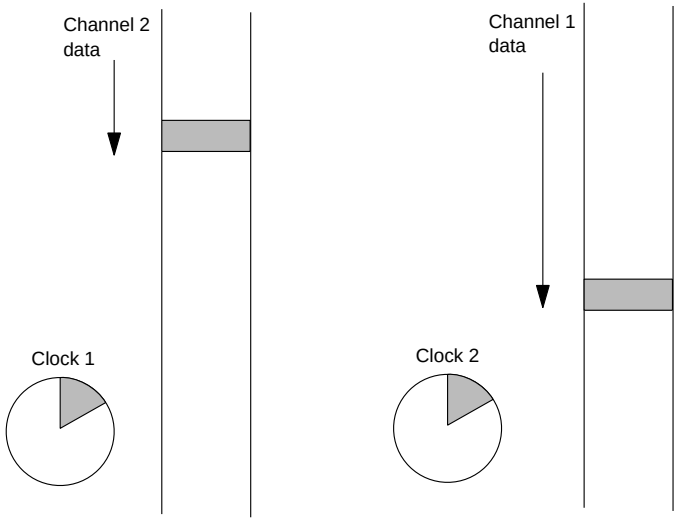
We could attempt to play the melodies by fetching a note from the channel 1 data stream or array and sending it to the channel 1 queue. We then fetch a note from the channel 2 data stream and send it to the channel 2 queue etc. (By 'send' we mean execute a SOUND statement.) This approach would be perfectly satisfactory if there were a limitless queue associated with each channel. However a channel queue can only hold a maximum of 5 requests.

By sending notes alternately to each channel, we have created the correspondence shown by the sloping lines. The program will be held up when it attempts to send the seventh minim to the channel 1 queue. The first minim will still be sounding and the next five have filled the queue. There are also five notes on the channel 2 queue but these are shorter and will be dealt with more frequently than the channel 1 notes. When the first minim on channel 1 has been played, four notes on channel 2 will have finished, leaving only two notes in the queue. The second minim on channel 1 now starts to play, making room for the seventh minim in the queue. This enables one further quaver to be added to the channel 2 queue before the program is again held up on attempting to add the eighth minim to the channel 1 queue. Thus while the second minim is being played on channel 1, only three quavers are available to be played on channel 2.

To solve this problem, we must arrange in this particular case to execute SOUND statements for channel 2 more frequently than for channel 1. Once the SOUND statement for the first note on channel 1 has been obeyed, no further channel 1 SOUND statements need be obeyed until SOUND statements have been obeyed for the first four notes on channel 2. In general, we must keep the total duration of notes for which channel 1 SOUND statements have been executed approximately equal to the total duration of notes for which channel 2 SOUND statements have been executed.

One way to solve the problem is to merge the voices into one DATA stream that contains channel numbers as well as pitch and duration values. This, however, makes the problem of transposition from the musical score to the DATA statement horrendous. The complexity of the synchronisation task has to be handled in the manual transposition process rather than in the program.

One simple way to achieve synchronisation between two or more voices is to keep a 'clock' running for each voice of the melody as shown.



In general the current note for each channel will be in a different position in the data streams. The clocks will tend to show equal elapsed times. Each time a SOUND statement is obeyed, the duration of the note is added to the clock associated with that channel. At each step we must obey a SOUND statement for the channel whose clock shows the least elapsed time. We require to repeat the following operation:

```
IF clock1 > clock2  THEN SOUND statement for channel 2

                    ELSE SOUND statement for channel 1
```

The program then selects one out of two alternative courses of action and this ensures that the channels free run and are not subject to interference from each other. Effectively we have removed the artificial connection in the parallel data streams between notes in different channels that have different duration values.

An alternative method of playing notes of a two-voice melody from separate data streams is to use the function ADVAL to test the channel queue status. For example, the expression 'ADVAL(-6)' has a value indicating the number of empty places on the channel 1 queue (-7 for channel 2 and -8 for channel 3). Thus, we could use:

```
IF ADVAL(-6)>0 AND voice 1 not yet finished THEN
                            SOUND statement for channel 1
IF ADVAL(-7)>0 AND voice 2 not yet finished THEN
                            SOUND statement for channel 2
```

This ensures that no SOUND statement is obeyed if a channel queue is full. The end effect is exactly the same as that of the clock algorithm. Using ADVAL, a separate test is needed In check whether a voice has finished, whereas with the clock method this possibility can be dealt with by setting the clock to a large value when the last SOUND statement for that voice is obeyed. The clock algorithm also makes it easy In incorporate a common musical requirement - emphasis of tho first note in every bar. Here the state of a clock could be used to recognise the first note of a bar.

Transposing

Another tedious task to be overcome before we start getting the machine to play arrangements is transposing from a musical score to a set of pitch numbers and associated notation. Transposing directly from the black dots to pitch numbers and durations in 1/20ths of a second can be tedious and error prone. You can write a graphics 'picking and dragging' program to input the music onto a screen stave, and this is a commonly adopted approach, but we have not space for that. Instead we shall adopt a character convention, and list the music in DATA statements using the following tables.

code	musical convention	duration (for metronome 150)
t	1/32	1
s	1/16	2
ds	dotted 1/16	3
e	1/8	4
de	dotted 1/8	6
q	1/4	8
dq	dotted 1/4	12
h	1/2	16
dh	dotted 1/2	24
w	whole	32

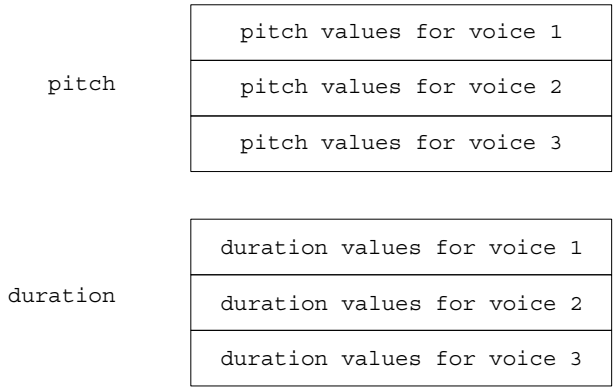
Remember that there are notes that cannot be accurately represented at this tempo. For example a dotted 1/32 is 1.5 (only 1 or 2 can be used as a duration parameter in a SOUND statement). Similarly a 1/16 triplet is 4/3 per note, an 1/8 triplet 8/3 per note and a 1/4 triplet 16/3 per note.

Pitch values arc represented using the convention :

pitch values	pitch number
C (C below middle C)	5
C' (middle C)	53
C'' (C above middle C)	101
C'''	149
C''''	197
C'# (middle C sharp)	57
C'b (middle C flat)	49
R rest	255

We do not cater for a key signature, but insert sharps and flats explicitly.

The character codes in the DATA statements will te converted into pitch and duration codes by the program. In all our programs for playing two or three part music, we shall use two two-dimensional arrays to hold up to three voices for an arrangement. These can be pictured as:



The first program uses only the first two rows of these arrays. There are also three one-dimensional arrays used to record the number of notes in each voice, a count of the notes SOUNDed for each voice and the 'clock' recording the total duration of the notes SOUNDed for each voice.

A two-voice Bach minuet

The next program is a complete program: that can be used to play two voices of a melody where the two voices are supplied separately in DATA statements using the above notation. The data in this case is a two-voice Bach minuet.

```
 10  ENVELOPE 1,1,0,0,0,0,0,0,126,-4,0,-63,126,100
 20  ENVELOPE 2,1,0,0,0,0,0,0,126,-4,0,-63,126,100
 30  ENVELOPE 3,1,0,0,0,0,0,0,128,-4,0,-63,126,100
 40  DIM pitch(3,100), duration(3,100), noofnotes(3),
       nextnote(3), clock(3)
 50  tempo=1
 60  PROCinitialise(1)
 70  PROCinitialise(2)
 80  PROCplaytwovoices
 90  END
200  DEF PROCinitialise(voice)
210  LOCAL note,pitch$,duration$,dur$,dur,
       notename$,position,prime$,octave
220    READ noofnotes(voice)
230    FOR note = 1 TO noofnotes(voice)
240      READ pitch$, duration$
250      dur$=RIGHT$(duration$,1)
255      dur =INSTR("tseqhw",dur$)
260      duration(voice,note)=2^(dur-1)*tempo
270      IF INSTR(duration$,"d") THEN
           duration(voice,note) =
           duration(voice,note)*3/2
280      notename$=LEFT$(pitch$,1)
290      position=INSTR("C-D-EF-G-A-BR",notename$)
300      IF position=13 THEN pitch(voice,note)=255
           ELSE pitch(voice,note)=1+4*position
310      IF RIGHT$(pitch$,1) = "#" THEN
           pitch(voice,note) = pitch(voice,note) + 4
320      IF RIGHT$(pitch$,1) = "b" THEN
           pitch(voice,note) = pitch(voice,note) - 4
330      prime$ = "'" : octave = 0
340      FOR j=2 TO LEN(pitch$)
350        IF MID$(pitch$,j,1) = prime$
           THEN octave = octave +1
360      NEXT j
370      pitch(voice,note) = pitch(voice,note)+octave*48
380    NEXT note
390  ENDPROC

400  DEF PROCplaytwovoices
410    nextnote(1)=0 : nextnote(2)=0
420    clock(1)=0 : clock(2)=0
430    finished=0
440    SOUND &101,0,0,8 : SOUND &102,0,0,8
450    REPEAT
460      IF clock(1) > clock(2) THENPROCsound(2)
           ELSE PROCsound(1)
470    UNTIL finished=2
480  ENDPROC

600  DEF PROCsound(voice)
610  LOCAL n ,envelope
```

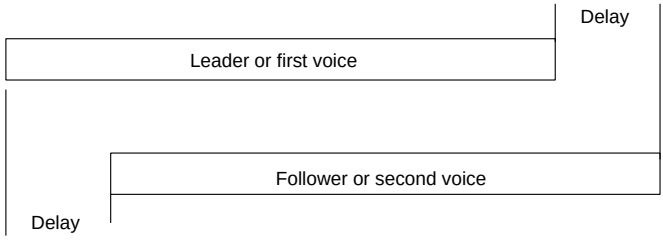
```
  620     nextnote(voice)=nextnote(voice)+1
  630     n=nextnote(voice)
  640     clock(voice)=clock(voice)+duration(voice,n)
  650     IF pitch(voice,n)=255 THEN envelope=0
            ELSE envelope=voice
  660     SOUND voice,envelope,pitch(voice,n),
            duration(voice,n)
  670     IF n=noofnotes(voice) THEN
            finished=finished+1 :clock(voice)=2000000
  680   ENDPROC

  700   DEF PROCround(leader,follower,delay)
  710   LOCAL l,f
  720     pitch(follower,1)=255 :duration(follower,1)=delay
  730     f = 1
  740     FOR l=1 TO noofnotes(leader)
  750       f = f + 1
  760       pitch(follower,f)=pitch(leader,l)
  770       duration(follower,f)=duration(leader,l)
  780     NEXT l
  790     noofnotes(follower)=f
  800   ENDPROC

 1000   DATA 74, D'',q,G',e,A',e,B',e,C'',e,D'',q,G',e,
          R,e,G',e,R,e,E'',q
 1010   DATAC'',e,D'',e,E'',e,F''#,e,G'',q,G',e,R,e,
          G',e,R,e,C'',q,D'',e
 1020   DATA C'',e,B',e,A',e,B',q,C'',e,B',e,A',e,G',e,
          F'#,q,G',e,A',e,B',e
 1030   DATA G',e,B',q,A',h,D'',q,G',e,A',e,B',e,C'',e,
          D'',q,G',e,R,e,G',e
 1040   DATA R,e,E'',q,C'',e,D'',e,E'',e,F''#,e,G'',q,
          G',e,R,e,G',e,R,e
 1050   DATA C'',q,D'',e,C'',e,B',e,A',e,B',q,C'',e,
          B',e,A',e,G',e,A',q
 1060   DATA B',e,A',e,G',e,F'#,e,G',h,G,q
 1070   DATA 37, B,h,A,q,B,dh,C',dh,B,dh,A,dh,G,dh,D',e,
          R,e,B,q,G,q,D',e,R,e
 1080   DATA D',e,C',e,B,e,A,e,B,h,A,q,G,q,B,q,G,q,C',dh,
          B,e,R,e,C',e,B,e,A,e,G,e
 1090   DATA A,h,F',q,G',h,B',q,C'',q,D'',q,D',q,G',dh
```

There is one further point to note in the above program. We have started PROCplaytwovoices with statements that play two synchronised rests, one on each of the two channels that we being used. By the time that these rests have finished being 'played', the program will have started to obey SOUND statements for the two voices and further use of synchronisation parameters is rendered unnecessary by the 100% timing accuracy of the sound generator. Provided that our two voices start in step, they will remain in step.

5.2 Simple canons or rounds

You can arrange the voices yourself if you have sufficient musical knowledge, but there are many compositions that will produce pleasing results on your micro. One particular intriguing musical form that is easy to transpose into a number of voices (because the second and third voices are derivable from the theme) is the canon.

The simplest and most familiar form of canon is the round 'Frere Jacques' is a common example. A theme (called the initiating voice or leader) enters. The second voice identical to the theme in the case of a round) enters after a time interval. The round is of course written in such a way that it harmonises with itself. Thus the theme performs two functions, firstly as a melody in its own right and secondly as a harmony or counterpoint to itself.



Now the follower is identical to the leader in the case of a round. In canons in general, it is mathematically derivable from it. Thus to play two or more voices only one theme need be transposed into a program.

A two-voice round - Frere Jacques

We can easily modify the above program to play a round. The next program plays 'Frere Jacques' as a two voice round with a two-bar delay. The procedure PROCround produces the two rows of our arrays necessary to play a round on two channels. In this procedure we effectively displace the follower by the delay, where the delay is specified to the procedure in multiples of the smallest possible note ( ).

```
10  ENVELOPE 1,1,0,0,0,0,0,0,63,10,0,-63,63,110
20  ENVELOPE 2,1,0,0,0,0,0,0,126,-4,0,-63,126,100
30  ENVELOPE 3,1,0,0,0,0,0,0,126,-4,0,-63,126,100
40  DIM pitch(3,100),duration(3,100),
      noofnotes(3), nextnote(3), clock(3)
50  tempo=1
60  PROCinitialise(1)
70  PROCround(1,2,64)
80  PROCplaytwovoices
90  END
```

```
        .
        .
        .
  700   DEF PROCround(leader,follower,delay)
  710   LOCAL l,f
  720     pitch(follower,1)=255:duration(follower,1)=delay
  730     f = 1
  740       FOR l=1 TO noofnotes(leader)
  750         f = f + 1
  760         pitch(follower,f)=pitch(leader,l)
  770         duration(follower,f)=duration(leader,l)
  780     NEXT l
  790     noofnotes(follower)=f
  800   ENDPROC

 1000   DATA 32,F',q,G',q,A',q,F',q,F',q,G',q,A',q,
           F',q,A',q,B'b,q,C'',h
 1010   DATA A',q,B'b,q,C'',h,C'',e,D'',e,C'',e,B'b,e,
           A',q,F',q,C'',e,D'',e
 1020   DATA C'',e,B'b,e,A',q,F',q,F',q,C',q,F',h,F',q,
           C',q,F',h
```

If the program doesn't sound right then you have probably made a mistake in typing the data. To check the tune through play a single voice only using a FOR loop:

```
FOR note= 1 TO noofnotes
 SOUND 1,1, pitch(1,note),duration(1,note)
NEXT note
```

These three lines should replace the call of PROCplaytwovoices.

Contrasting ENVELOPES can be used to effect, and we leave you to experiment with these. Now the theme in the above program is rather banal and boring but it is necessary to verify that your program works before moving on to the serious stuff.

5.3 Synchronizing three (or more!) voices

Before moving on to morecomplex canons, we first present a procedure to synchronise music consisting of three separate voices. In the next program, we have replaced PROCplaytwovoices with PROCharmonise which can organise the playing of three voices. In fact it will organise the playing of any number of voices from one upwards as specified by its parameter. It could be used to play more than three voices if we had more than three musical sound channels available. Each execution of the REPEAT loop in this procedure picks out a channel that has fallen behind and issues a SOUND statement for that channel.

A three-voice round - Frere Jacques

The next program indicatea thc changes needed to arrange and play a three-voice round using the same DATA as before.

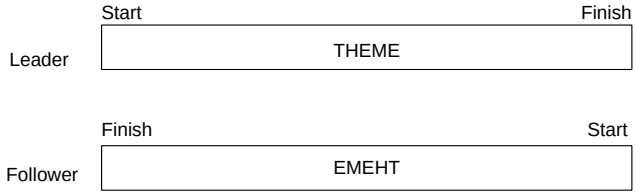
```
 10  ENVELOPE 1,1,0,0,0,0,0,0,63,10,0,-63,63,110
 20  ENVELOPE 2,1,0,0,0,0,0,0,126,-4,0,-63,126,100
 30  ENVELOPE 3,1,0,0,0,0,0,0,126,-4,0,-63,126,100
 40  DIM pitch(3,100), duration(3,100),
       noofnotes(3),nextnote(3),clock(3)
 50  tempo = 1
 60  PROCinitialise(1)
 70  PROCround(1,2,64)
 80  PROCround(2,3,64)
 90  PROCharmonise(3)
100  END
       :
       :

400  DEF PROCharmonise(noofvoices)
410  LOCAL voice,slowest,sync
420    sync = (noofvoices-1)*&100
430    FOR voice=1 TO noofvoices
440      SOUND sync+voice,0,0,8
450      clock(voice)=0 : nextnote(voice)=0
460    NEXT voice
470    finished=0
480    REPEAT
490      slowest=1
500      FOR voice=1 TO noofvoices
510        IF clock(voice)<clock(slowest)
           THEN slowest=voice
520      NEXT voice
530      PROCsound(slowest)
540    UNTIL finished=noofvoices
550  ENDPROC
       :
       :
```

5.4 Bach's 'Musical Offering'

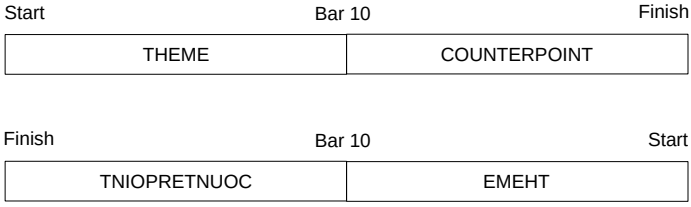
J. S. Bach's amazing work 'The Musical Offering' contains a number of canons of different forms. The American mathematical philosopher Douglas Hofstader, said of the work 'All in all, the Musical Offering represents one of Bach's supreme accomplishments in counterpoint. It is itself one large intellectual fugue, in which many ideas and forms have been woven together, and in which playful double meanings and subtle allusions are commonplace. And it is a very beautiful creation of the human intellect which we can appreciate forever'.

Crab canons or canons in retograde motion

One of the rarest forms of canon is the crab canon or canon in retrograde motion. It is a rare form presumably because it is so difficult to write. In a crab canon there is no delay, both themes enter simultaneously. The first voice plays the theme from the start and the second voice plays the same theme backwards from the end.



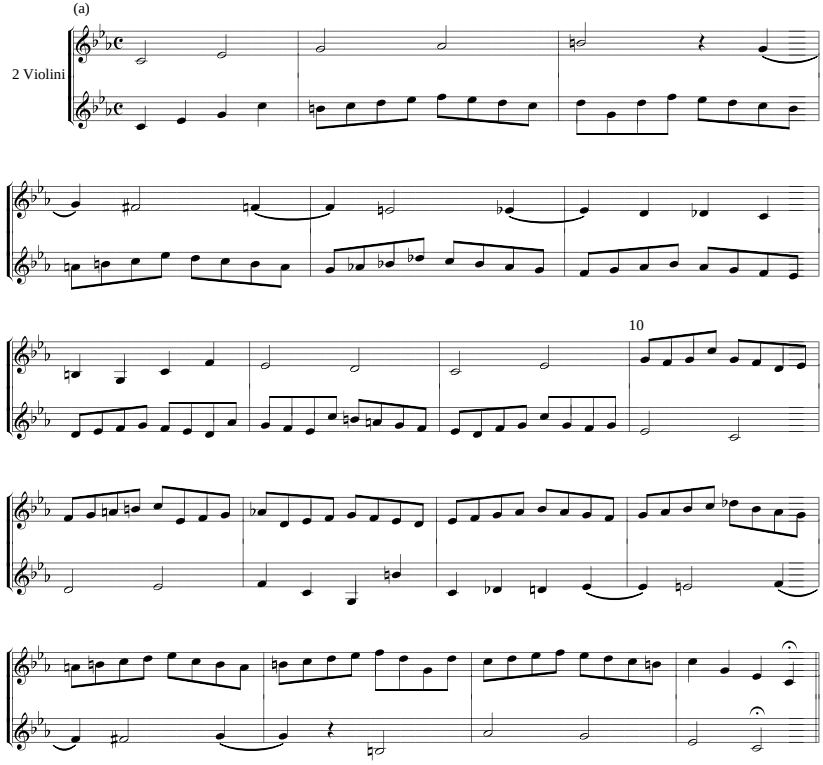
Bach's No. 9 canon frm 'The Musical Offering' is a crab canon. It contains a theme with long duration notes followed by a counterpoint. The theme is played against the reverse of the counterpoint, followed by the counterpoint playing against the reverse of the theme. This structure is obvious when you listen to the music.



The next program includes a procedure to generate the arrays for a crab canon and includes the DATA for Bach's crab canon.

```
10   ENVELOPE 1,1,0,0,0,0,0,0,63 ,10,0,-63,63,110
20   ENVELOPE 2,1,0,0,0,0,0,0,126,-4,0,-63,126,100
30   ENVELOPE 3,1,0,0,0,0,0,0,126,-4,0,-63,126,100
40   DIM pitch(3,100), duration(3,100),
       noofnotes(3), nextnote(3), clock(3)
50   tempo=1
60   PROCinitialise(1)
70   PROCcrab(1,2)
80   PROCharmonise(2)
90   END
     .
     .
     .
```

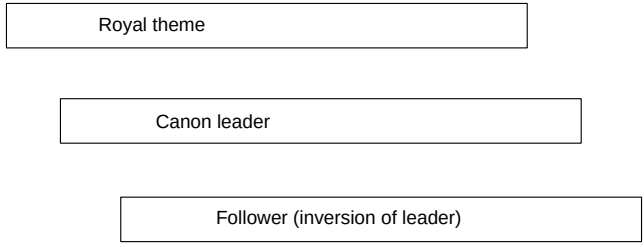
```
 700   DEF PROCcrab(voice,othervoice)
 710   LOCAL n1,n2
 720     n1=noofnotes(voice)
 730     FOR n2=1 TO noofnotes(voice)
 740       pitch(othervoice,n2)=pitch(voice,n1)
 750       duration(othervoice,n2)=duration(voice,n1)
 760       n1=n1-1
 770     NEXT n2
 780     noofnotes(othervoice )=noofnotes(voice)
 790   ENDPROC

1000   DATA 90,C',h,E'b,h,G',h,A'b,h,B,h,R,q,G',h,F'#,h,
       F',h,E',h,E'b,h,D',q,D'b,q,C',q,B,q,G,q,C' ,q,
       F',q,E'b,h,D',h,C',h,E'b,h,G',e,F',e,G',e,C'',e,
       G',e,E'b,e,D',e,E'b,e,F',e,G',e,A',e,B',e,C'',e
1010   DATA E'b,e,F',e,G',e,A'b,e,D',e,E'b,e,F',e,G',e,
       F',e,E'b,e,D',e,E'b,e,F',e,G',e,A'b,e,B'b,e,
       A'b,e,G',e,F',e,G',e,A'b,e,B'b,e,C'',e,D''b,e,
       B'b,e,A'b,e,G',e,A',e,B',e,C'',e,D'',e,E''b,e,C'',e
1020   DATA B'b,e,A'b,e,B',e,C'',e,D'',e,E''b,e,F'',e,
       D'',e,G',e,D'',e,C'',e,D'',e,E''b,e,F'',e,E''b,e,
       D'',e,C'',e,B',e,C'',q,G',q,E'b,q,C',q
```



5.5 Mirror canons or canons in contrary motion

In this form the follower is derived from the leader by inverting the intervals in the leader. This means that when the leader ascends the follower descends by exactly the same interval. A familiar tune that will work as a mirror canon is 'Good King Wenceslas'. A time delay of half a bar is needed between the leader and the follower.

We now look at canon No. 4 frcan the 'Musical Offering'. This is a three-part arrangement, a variation of the 'Royal Theme' - the centre piece of the work - providing the upper voice. The higher canonic part enters first followed by its exact inversion half a bar later (delay = 8 notes):



In a mirror canon there is a common note about which the reflection occurs. In this case it is Eb (the third degree of the C minor scale - the key of the work). Thus C in the leader becomes G in the follower and vice versa. If all that is a bit technical bear in mind that it is just a rule for deriving the first note of the follower. Once the first note of the follower is fixed we derive the remainder by inverting the intervals in the leader. The leader in this case starts as the sequence:

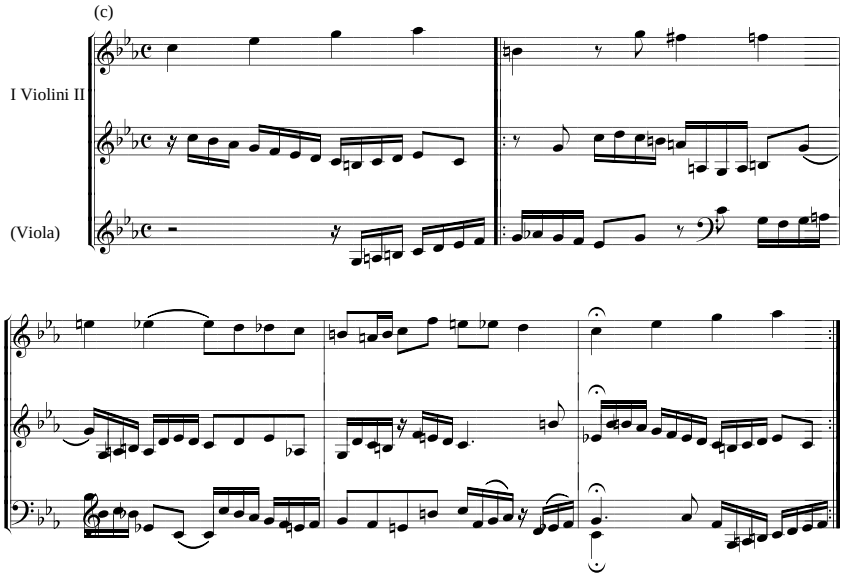
```
C  Bb  Ab  G  F ...
 \/  \/  \/ \/
 T   T   S  T  ...
```

i.e. a descending sequence of tone, tone, semitone, tone,... The follower thus begins (in the octave below):

```
G  A  B  C  D  ...
 \/ \/ \/ \/
 T  T  S  T  ...
```

i.e. an ascending sequence of tone, tone, semitone, tone,... The next program contains the procedure required to generate the arrays for the inverted part from the data for the canon. It uses the data for the 'Royal Theme' together with the canon data.

```
      :
      :
  50  tempo=2
  60  PROCinitialise(1)
  70  PROCinitialise(2)
  80  PROCinvert(2,3,-68,16*tempo)
  90  PROCharmonise(3)
 100  END
      :
      :
 700  DEF PROCinvert(voice,othervoice,shift,delay)
 710  LOCAL n1,next1,n2,lastpitchon1,lastpitchon2
 720    IF delay >0 THEN pitch(othervoice,1)=255 :
          duration(othervoice,1)=delay : n2=1
        ELSE n2=0  :REM n2 counts notes in other voice.
 730  next1=1  :REMnext1 is next note in voice.
 740  REPEAT  :REM to copy rests and find first note.
 750    IF pitch(voice,next1)=255 THEN
          n2=n2+1:pitch(othervoice,n2)=255:
          duration(othervoice,n2)=duration(voice,next1):
          next1 = next1+1
 760  UNTIL pitch(voice,next1)<>255
 770  n2=n2+1
 780  pitch(othervoice,n2)=pitch(voice,next1)+shift
 790  lastpitchon1=pitch(voice,next1)
 800  lastpitchon2=pitch(othervoice,n2)
 810  duration(othervoice,n2)=duration(voice,next1)
 820  next1=next1+1
 830  FOR n1=next1 TO noofnotes(voice)
 840    n2=n2+1
 850    nextinterval=-(pitch(voice,n1)-lastpitchon1)
 860    IF pitch(voice,n1)=255 THEN
          pitch(othervoice,n2)=255
        ELSE
          pitch(othervoice,n2) =
          lastpitchon2+nextinterval :
          lastpitchon1=pitch(voice,n1) :
          lastpitchon2=pitch(othervoice,n2)
 870  duration(othervoice,n2)=duration(voice,n1)
 880  NEXT n1
 890  noofnotes(othervoice)=n2
 900  ENDPROC

1000  DATA 22,C'',q,E''b,q,G'',q,A''b,q,B',q,R,e,G'',e,
      F''#,q,F'',q,E'',q,E''b,dq,D'',e,D''b,e,C'',e,
      B',e,A',s,G',s,C'',e,F'',e,E'',e,E''b,e,D'',q
1010  DATA 46,R,s,C'',s,B'b,s,A'b,s,G',s,F',s,E'b,s,
      D',s,C',s,B,s,C',s,D',s,E'b,e,C',e,R,e,G',e,
      C'',s,D'',s,C'',s,B'b,s,A',s,A,s,G,s,A,s,B,e,
      G',de,G,s,A,s,B,s,C',s,D',s,E'b,s,D',s,C',e,
      D',e,E'b,e,Ab,e,G,s,D',s,C',s,B,s,R,s,F',s,
      E',s,D',s,C',dq
```



Exercises

1 Arrange for the voice synchronisation procedures to recognise the first note in a bar and emphasise it slightly by playing it with a different envelope. (Define envelope 4 for this purpose).

2 Animate a piece of music by drawing vertical lines at successive x positions, one line for each note as the note is played. The height of a line should be proportional to the pitch of the corresponding note.

3 Animate a piece of music by displaying the notes on a musical stave as they are played.

4 Transpose 'Good King Wenceslas' into our musical notation and play it as a mirror canon as suggested in the text.

5 Modify PROCharmonise to handle sound channels numbered from zero upwards. Write a procedure that generates, from the voice I data, a set of pitch and duration values for playing a 'drumbeat' on Channel 0. (There is room in our arrays for these - the zero subscripts were not used.) You can experiment with different rules for generating the drumbeat. For example, you might have a drumbeat only at the start of each bar, or every time the start of a note on voice I coincides with the natural 'beat' of the music.

5.6 Automatic composition

Music has been called 'a compromise between chaos and monotony'. The next two programes provide two contrasting examples. The first is an example of 'chaos' and the second represents structured monotony. The first program selects a note or pitch, channel, envelope and duration entirely at random.

```
 10  ENVELOPE 1,1, 0,0,0,0,0,0,
       126,-4,0,-63,126,100
 20  ENVELOPE 2,1, 0,0,0,0,0,0,
       63,10,0,-63,63,110
 30  ENVELOPE 3,1, 0,0,0,0,0,0,
       126,-8,0,-10,126,50
 40  FOR note=1 TO RND(100)
 50    channel = RND(3)
 60    envelope = RND(3)
 70    pitch = RND(256)-1
 80    duration=RND(32)
 90    SOUND channel,envelope,pitch,duration
100  NEXT note
```

The second program produces a monotonous sequence, the nature of which should be clear from a reading of the program.

```
 10  ENVELOPE 1,1, 0,0,0,0,0,0,
       126,-4,0,-63,126,100
 20  ENVELOPE 2,1, 0,0,0,0,0,0,
       63,10,0,-63,63,110
 30  ENVELOPE 3,1, 0,0,0,0,0,0,
       126,-8,0,-10,126,50
 40  FOR note = 1 TO 20
 50    SOUND 1,1,53,8
 60  NEXT note
 70  key=GET
 80  FOR pitch=53 TO 101 STEP 4
 90    SOUND 1,1,pitch,8
100  NEXT pitch
110  key=GET
120  FOR phrase = 1 TO 10
130    SOUND 1,1,53,8
140    SOUND 1,1,69,8
150    SOUND 1,1,81,8
160    SOUND 1,1,101,8
170  NEXT phrase
```

In this section, we explore ways of getting the BBC micro to compose its own music. In order for a computer to compose interesting music, there must be some degree of randomness involved, otherwise the music produced would be monotonous. But the music must also satisfy certain rules that make it recognisable as music to the listener. Incidentally, the rules that make music acceptable vary from culture to culture and from period to period. For example, Oriental music sounds strange to Western ears and the music of Beethoven (18th to 19th century) would probably have shocked Palestrina (16th century). The ear needs educating in the rules that are prevalent at a particular period.

Random music is not necessarily unpleasant, particularly if the texture of the music is controlled. The next program illustrates this point and plays music selecting two random numbers to drive the pitch and duration in a SOUND statement. Superimposed on this basic method we have added three effects:

(1) An echo (Two SOUND statements referencing separate envelopes)

(2) Insertion of a glissando or slide (de rigueur in arcade games) at random instants, and

(3) Insertion of pitch distortion at random instants.

The net effect is not uninteresting. Note that the pitch distortion is inserted by changing the parameters in a single ENVELOPE statement. In this case we could have used two ENVELOPE statements with different parameters and selected, but the setting and resetting of ENVELOPE parameters (PROCpitchset and PROCpitchreset) is the general structure required for dynamically changing ENVELOPE parameters and back again in a playing loop.

```
 10  ENVELOPE 1,1, 0,0,0,0,0,0,
       126,-4,0,0,126,100
 20  ENVELOPE 2,1, 0,0,0,0,0,0,
       63,-4,0,0,63,50
 30  prevnote = 0
 40  FOR I = 1 TO 100
 50    note = RND(255)
 60    IF note MOD 11 =0 THENPROCslide(prevnote,note)
 70    IF note MOD 7 = 0 THENPROCpitchset
 80    SOUND 1,1,note,RND(8)
 90    SOUND 1,2,note,RND(8 )
100    prevnote = note
110    PROCpitchreset
120    NEXT
130  END
```

```
140  DEF PROCslide(old,new)
150    IF old>new THEN step = -1 ELSE step = 1
160    SOUND &1001,0,0,0
170    FOR i = old TO new STEP step
180      SOUND &1001,0,0,0
190      SOUND &11,1,i,2
200      PROCpitchreset
210    NEXT
220  ENDPROC

230  DEF PROCpitchset
240    pi1 = 16:pi2 = -16: pi3 = 16
250    pn1 = 2: pn2 =4: pn3 = 2
260    ENVELOPE 1,1, pi1,pi2,pi3,pn1,pn2,pn3,
         126,-4,0,0,126,100
270  ENDPROC

280  DEF PROCpitchreset
290    ENVELOPE 1,1, 0,0,0,0,0,0,
         126,-4,0,0,126,100
300  ENDPROC
```

5.7 Generating rhythms

Rhythm is a very important component of music of all cultures. Indeed in some primitive cultures, music consists of rhythm and very little else. In this section, we shall examine ways of making a computer generate a rhythmic structure that is similar to that of a simple folk tune.

In music, rhythm is concerned with the grouping of notes into beats, of beats into bars, bars into phrases and so on. In the Oxford Companion to Music, the entry under 'phrase' states that any simple four-line hymn or folk-tune falls clearly into two halves or 'sentences'. Each sentence falls into two phrases and each phrase normally consists of four bars (although this is sometimes varied). We shall use this simple model for our first attempts at automatic composition.

To a computer scientist or linguist, the above description suggests the use of a 'generative grammar' to describe the structure of a piece of music. Such grammars are used extensively by computer scientists to describe the structure of programming languages. Such structures are, for example, reflected in the construction of a compiler. In this case, we might start with the rule

```
TUNE ::= SENTENCE SENTENCE
```

which we read as 'A tune consists of a sentence followed by another sentence'. ::= is a special symbol meaning 'consists of' or 'can be rewritten as'.

We could then go on to define

```
SENTENCE ::= PHRASE PHRASE
PHRASE ::= BAR BAR BAR BAR
```

or we might decide that the last bar of a phrase should have a different structure from the other bars:

```
PHRASE ::= BAR1 BAR1 BAR1 BAR2
```

where a BAR2 will have a different definition from a BAR1. Rules like these are usually referrexi to as 'rewrite rules'. The left-hand side of the rule can be rewritten as the right-hand side.

A more complicated example of a musical grammar might start off with

```
PIECE ::= SONATA | RONDO | FUGUE
```

The sign '|' is read as 'or', thus the above rewrite rule states that a piece is either a sonata or a rondo or a fugue. The definition might continue with

```
SONATA ::= EXPOSITION DEVELOPMENT RECAPITULATION
```

Simple rewrite rules provide a concise notation for describing the structure of language or music, but they have many limitations and the system has to be 'augmented' for more advanced applications.

Returning to our simple folk-tune example, the structure of the rules constituting the grammar can be directly reflected in the structure of a BASIC program that generates a piece of music from the grammar. In the next program, the rule defining a tune has been transcribed directly into into a procedure that generates a tune.

```
DEFPROCtune                  Corresponds to rule
  PROCsentence
  PROCsentence          TUNE ::= SENTENCE SENTENCE
ENDPROC
```

PROCsentence is defined similarly. These two procedures could have been combined into one, a tune being defined as four phrases, but it is always advisable to maintain a procedure structure that reflects the structure of the process being modelled. We may decide later that the first sentence in a tune should have a slightly different structure than the second. Defining a tune in terms of sentences and a sentence in terms of phrases will make it easier to incorporate changes like this.

PROCphrase is defined in a similar way. It makes three identical calls of PROCbar and then a fourth call of PROCbar

to generate the last bar of the phrase. The type of bar to be generated has been indicated by a parameter.

```
DEF PROCphrase
  LOCAL bar                  Corresponds to rule
  FOR bar=1 TO 3
    PRCOCbar(minnote)  PHRASE ::= BAR1 BAR1 BAR1 BAR2
  NEXT bar
PROCbar(16)
ENDPROC
```

The parameter indicates the minimum duration permitted for the final note of the bar and we have created the last bar of a phrase (a BAR2) by supplying a different parameter, 16. This indicates that the bar generated by this call should have a final note of duration at least 16 time units, i.e. a minim. Forcing a phrase to end with a longish note gives an impression of rounding off the phrase. The first three bars of a phrase are allowed to terminate with the shortest permitted note available for the tune being composed. This value is called 'minnote' and is input to the program as a parameter. The value input determines the overall 'tempo' of the piece.

The 'grammar' of a bar will depend on the number of beats in a bar (another input parameter). For example, in 2/4 time, we could have

```
BAR1 ::= CROTCHETGROUP CROTCHETGROUP | MINIMGROUP
```

This means 'a bar can be a group of notes equivalent to a crotchet followed by another crotchet group, or a bar can consist of a group of notes equivalent to a minim.' We shall not allow note groupings to cut across the 'beat' structure of the bar. We could define

CROTCHETGROUP ::= 

assuming a semiquaver as the minimum permitted note (duration = 2). For convenience, we insist that notes in a group all have the same duration. We do not permit

CROTCHETGROUP ::= 

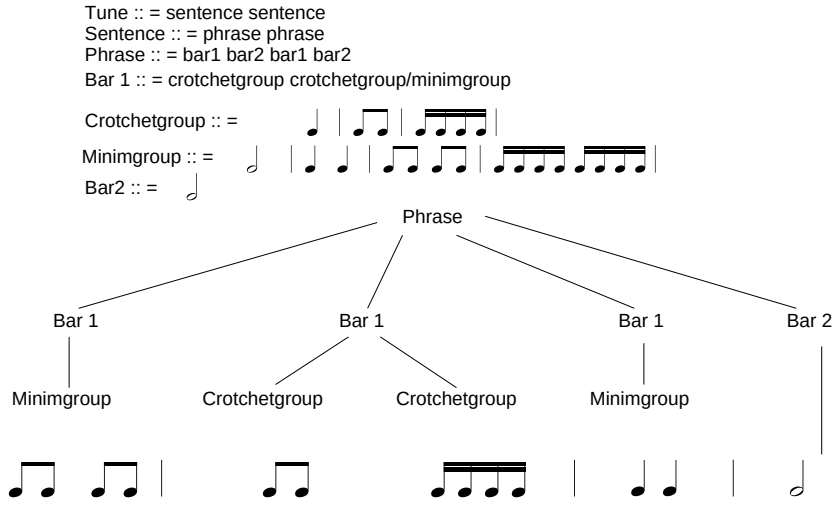
A minim group is defined as

MINIMGROUP ::= 

Recall that we require a phrase to terminate with at least a minim. With two beats to the bar, this means

BAR2 ::= 

The complete grammar for 2/4 time is listed next. The whole process of generating a sequence of symbols (in this case notes of a certain duration) using rewrite rules can be viewed as a tree structure. Using choice where choice in available we could generate the tree shown below. This particular tree is just one of a large number that could be generated from the rewrite rules. The tree structure or hierarchy is reflected directly in the procedure hierarchy or structure.



BAR1 and BAR2 would be defined slightly differently if we had three or four (or more) beats to the bar.

The definitions of BAR1. and BAR2 are implemented in a fairly ad hoc fashion in PROCbar in the program.

```
 10   ENVELOPE 1,1, 0,0,0,0,0,0,
      126,-8,0,-63,126,50
 20   ENVELOPE 2,1, 0,0,0,0,0,0,
      100,-8,0,-80,100,50
100   INPUT "Beats per bar",timesig
110   INPUT "Minimum note",minnote
120   PROCtune
130   END

200   DEF PROCtune
210     PROCsentence
220     PROCsentence
230   ENDPROC
```

```
240  DEF PROCsentence
250    PROCphrase
260    PROCphrase
270  ENDPROC

280  DEF PROCphrase
290    LOCAL bar
300    FOR bar=1 TO 3
310      PROCbar(minnote)
320    NEXT bar
330    PROCbar(16)
340  ENDPROC

350  DEF PROCbar(minfinish)
360    envelope = 1
370    beatsleft=timesig
380    REPEAT
390      PROCselectgroup
400      IF beatsleft=0
           THEN PROCsubdividegroup(minfinish)
           ELSE PROCsubdividegroup(minnote)
410      FOR note=1 TO nextgroup DIV duration
420        PROCplaynote
430      NEXT note
440    UNTIL beatsleft=0
450  ENDPROC

460  DEF PROCselectgroup
470    LOCAL g,timeleft
480    timeleft=beatsleft*8
490    IF beatsleft=1 OR timeleft=minfinish THEN
         nextgroup=timeleft :beatsleft=0:ENDPROC
500    REPEAT:g=RND(beatsleft)
510    UNTIL beatsleft-g=0 OR timeleft-g*8>=minfinish
520    nextgroup=g*8
530    beatsleft=beatsleft-g
540  ENDPROC

550  DEF PROCsubdividegroup(mindur)
560    IF nextgroup=mindur OR nextgroup MOD mindur<>0
       THEN duration=nextgroup:ENDPROC
570    REPEAT
580      duration=2*RND(5)
590    UNTIL nextgroup MOD duration=0ANDduration>=mindur
600  ENDPROC

610  DEFPROCplaynote
620    pitch=53
630    SOUND 1,envelope,pitch,duration:envelope = 2
650  ENDPROC
```

PROCbar repeatedly chooses a group consisting of a random number of whole notes that in less than or equal to the number of beats left to be played, subject to the constraint imposed by the 'minimum last note' parameter. Each group chosen is then split into an equal number of notes whose duration divides into the group chosen and whose duration is less than or equal to the minimum permitted duration. The notes of the group are then played (all on Middle C).

One further enhancement that assists the listener's perception of rhythm is to use a slightly louder envelope for the first note of a bar than is used for the remaining notes of the bar.

Listen to some of the output from this program and you will find that the 'sentence', phrase, bar and beat structure is usually fairly evident.

5.8 Generat1ng pitch values

We now turn our attention to the pitch of the notes played in the tune so that we can impose a melodic sequence on the rhythmic structure.

Playing random notes from a scale

A particular piece of music (or at least a section of a piece of music) is usually confined to notes taken from a set of notes that are closely related to each other in some way. The set of notes, or 'scale', used contributes in a large way to the 'character' of the music. We can easily alter our 'Rhythm on Middle C' program so that it selects random notes from a particular scale. The next program indicates the modifications needed to do this.

```
 10  ENVELOPE 1,1, 0,0,0,0,0,0,
     126,-4,0,-63,126,100
 20  ENVELOPE 2,1, 0,0,0,0,0,0,
     100,-4,0,-80,100,80
 30  READ scalelength
 40  DIM scalenote(scalelength)
 50  FOR n=1 TO scalelength
 60    READ scalenote(n)
 70  NEXT
 80  DATA 4, 53,69,81,101

 90  keynote=scalenote(1)
100  INPUT "Beats per bar" ,timesig
110  INPUT "Minimum note",minnote
120  PROCtune
130  END

200  DEF PROCtune
210    PROCsentence(FALSE)
220    PROCsentence(TRUE)
230  ENDPROC
```

```
240   DEF PROCsentence(finalsent)
250     PROCphrase(FALSE)
260     PROCphrase(finalsent)
270   ENDPROC

280   DEF PROCphrase(finalph)
290     LOCAL bar
300       FOR bar=1 TO 3
310         PROCbar(minnote,FALSE)
320       NEXT bar
330       PROCbar(16,finalph)
340   ENDPROC

350   DEF PROCbar(minfinish,finalbar)
360     envelope = 1
370     beatsleft=timesig
380     REPEAT
390       PROCselectgroup
400       IF beatsleft=0
            THEN PROCsubdividegroup(minfinish)
            ELSE PROCsubdividegroup(minnote)
410       FOR note=1 TO nextgroup DIV duration
420         PROCplaynote(note=nextgroup DIV duration AND
            beatsleft=0 AND finalbar)
430       NEXT note
440     UNTIL beatsleft=0
450   ENDPROC
      .
      .
      .

610   DEF PROCplaynote(finalnote)
620     IF finalnote THEN pitch=keynote
        ELSE pitch=scalenote(RND(scalelength))
630     SOUND 1,envelope,pitch,duration
640     envelope=2
650   ENDPROC
```

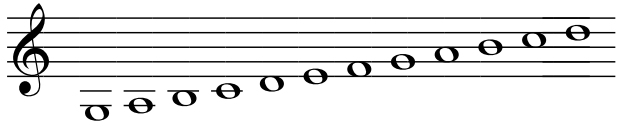
The DATA statement at line 80 defines the number of notes and the pitch values for the scale used, in this case a major arpeggio.

One further addition has been made to this program. A new parameter is passed to each of PROCsentence, PROCphrase and PROCbar to indicate whether it is the final example of that construction in the tune. This enables the program to recognise the last note of the tune and constrain it to fall on the keynote of the scale.

Try running the program with notes taken from the major arpeggio and you will obtain a moderately pleasing if rather monotonous effect. Then try some of the other scales listed here.

Scale	Interval Sequence							
major	8	8	4	8	8	8	4	
diminished	4	8	4	8	4	8	4	8
blues	12	8	4	4	12	8		
Hindu	8	8	4	8	4	8	8	
whole tone	8	8	8	8	8	8		
dorian minor	8	4	8	8	8	4	8	
aeolian minor	8	4	8	8	4	8	8	
hamonic minor	8	4	8	8	4	12	4	
pentatonic	8	8	12	8	12			

In subsequent sections, we shall use notes taken from the following extended major scale:



This is the scale of C major extended downwards by three notes to lower G and up one note to upper D.

First order probability distribution of notes

Once the set of notes (the key) on which a tune will be played has been determined, there are a number of further constraints that can be applied in order to make a tune mimic a particular style. In music of a particular style, certain notes and combinations of notes will be more common than others. It is fairly obvious that Mozart does not sound like Stravinsky and this difference can be quantified to a certain extent using the techniques now described.

One way of making our program select pitch values more systematically is to make it use probability distributions when selecting the pitch of a note to be played. The simplest (and least satisfactory) type of distribution that can be used is the first order probability distribution. The table below shows the result of a 'first order analysis' of 9 simple well-known tunes taken from a child's recorder tutor (Baa Black Sheep, Bobby Shaftoe, etc.).

g	a	b	c	d	E	F	G	A	B	C	D
3.4	0.7	3.4	15.0	12.6	13.8	8.5	14.3	8.0	11.1	6.3	2.9

If we imagine all these tunes being played in C major, on the 12 notes from lower G to upper D, then the figures in the table give, as percentages, the relative frequency of occurrence of each note over the nine tunes analysed. Thus, 3.4% of the notes in these tunes were bottom G, 15% of the notes were Middle C and so on. (These percentages were part

of the output produced by a program which is discussed later.)

We now explain how to generate a note so that the probability of getting a particular note matches its entry in the table. The tunes generated by doing this will not be much better than those obtained by choosing a random note from the scale, hut the basic technique used is easily extended to deal with higher order probability distributions discussed next.

We first set up an array containing the above percentages. In order to select a note, we generate a random number in the range 0 to 100 and add up values from the array until the total exceeds the random number generated (see below). The number of percentages added determine the note from the scale that is selected.

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
3.4	0.7	3.4	15.0	12.6	13.8	8.5	14.3	8.0	11.1	6.3	2.9

```
rand=RND(1)*100
```

Say, for example, rand=43.5. Then the first 6 values in freq1 are added before we obtain a total that exceeds 43.5. This means that we play the 6th note in the scale.

```
pitch=scalenote(6)
```

This approach is implemented in the next program.

```
  10  ENVELOPE 1,1, 0,0,0,0,0,0,
      126,-4,0,-63,126,100
  20  ENVELOPE 2,1, 0,0,0,0,0,0,
      100,-4,0,-80,100,80
  30  READ scalelength
  40  DIM scalenote(scalelength)
  50  FOR n=1 TO scalelength
  60    READ scalenote(n)
  70  NEXT
  80  DATA 12, 33,41,49,53,61,69,73,81,89,97,101,109
  90  keynote=scalenote(4)
 100  PROCsetupfreqtable1
 110  INPUT "Beats per bar",timesig
 120  INPUT "Minimum note",minnote
 130  notesplayed=0
 140  PROCtune
 150  END
      :
      :
```

```
  610  DEF PROCplaynote(finalnote)
  620    IF finalnote THEN pitch=keynote
           ELSE PROCselectpitch1
  630    SOUND 1,envelope,pitch,duration
  640    notesplayed=notesplayed+1
  650    envelope=2
  660  ENDPROC

  670  DEF PROCsetupfreqtable1
  680  LOCAL lb1,l,n,fileno
  690    DIM freq1(12)
  700    FOR n=1 TO 12
  710      READ freq1(n)
  720    NEXT n
  730  ENDPROC

  740  DEF PROCselectpitch1
  750  LOCAL rand, n, sum
  760    rand=RND(1)*100
  770    n=0 : sum=0
  780    REPEAT
  790      n=n+1 :sum=sum+freq1(n)
  800    UNTIL sum>=rand
  810    pitch=scalenote(n)
  820    lastnoteplayed=n
  830  ENDPROC
10010  DATA 3.4, 0.7, 3.4, 15.0, 12.6, 13.8,
       8.5, 14.3, 8.0, 11.1, 6.3, 2.9
```

Note sequences are much more important in composing melodies, and if we want to constrain the note sequences that are chosen by our program, we must consider higher wider probabilities.

Second order probability distributions

The use of second order probability distributions makes the choice of a note depend on the preceding note.

The next table shows the second order distributions resulting from an analysis of our 9 simple tunes.

	g	a	b	c	d	E	F	G	A	B	C	D
g	42.9	0.0	0.0	28.6	14.3	7.1	0.0	7.1	0.0	0.0	0.0	0.0
a	0.0	0.0	66.7	0.0	33.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
b	21.4	0.0	7.1	50.0	21.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
c	0.0	0.0	12.7	40.0	14.5	18.2	10.9	3.6	0.0	0.0	0.0	0.0
d	7.7	0.0	7.7	26.9	28.8	13.5	9.6	5.8	0.0	0.0	0.0	0.0
E	0.0	3.5	0.0	17.5	26.3	21.1	14.0	17.5	0.0	0.0	0.0	0.0
F	0.0	2.9	0.0	0.0	20.0	45.7	8.6	22.9	0.0	0.0	0.0	0.0
G	0.0	0.0	0.0	3.5	1.8	15.8	21.1	22.8	31.6	0.0	1.8	1.8
A	0.0	0.0	0.0	0.0	0.0	0.0	3.0	24.2	18.2	48.5	0.0	6.1
B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	13.0	13.0	43.5	23.9	6.5
C	0.0	0.0	0.0	0.0	0.0	3.8	0.0	3.8	3.8	30.8	42.3	15.4
D	0.0	0.0	0.0	0.0	0.0	0.0	0.0	33.3	16.7	8.3	25.0	16.7

One row in this table corresponds to one of our notes and the entries in a row give the percentage of occasions on which each of the other notes followed the note to which the row corresponds. For example, row 1 indicates that lower G is followed by another lower 6 on 42.9% of occasions, by Middle C on 28.6% of occasions, by D next to Middle C on 14.3% of occasions, by E on 7.1% of occasions and by upper G on 7.1% of occasions.

The modifications to the previous program needed to generate notes according to the second order probability distributions are presented in the next program.

```
        :
        :
  100   PROCsetupfreqtable1
  101   PROCsetupfreqtable2
        :
        :
  610   DEF PROCplaynote(finalnote)
  620   IF finalnote THEN pitch=keynote
          ELSE IF notesplayed=0 THEN PROCselectpitch1
          ELSE PROCselectpitch2
        :
        :
  840   DEF PROCsetupfreqtable2
  850   LOCAL l,n
  860     DIM freq2(12,12)
  870     FOR l=1 TO 12
  880       FOR n=1 TO 12
  890         READ freq2(l,n)
  900     NEXT:NEXT
  910   ENDPROC

  920   DEF PROCselectpitch2
  930   LOCAL rand, n, sum
  940     rand=RND(1)*100
  950     n=0 : sum=0
  960     REPEAT
  970       n=n+1 : sum=sum+freq2(lastnoteplayed,n)
  980     UNTIL sum>=rand
  990     pitch=scalenote(n)
 1000     lastbut1=lastnoteplayed : lastnoteplayed=n
 1010   ENDPROC

10010   DATA 3.4, 0.7, 3.4, 15.0, 12.6, 13.8,
        8.5, 14.3, 8.0, 11.1, 6.3, 2.9
20010   DATA 42.9, 0.0, 0.0, 28.6, 14.3, 7.1,
        0.0, 7.1, 0.0, 0.0, 0.0, 0.0
20020   DATA 0.0, 0.0, 66.7, 0.0, 33.3, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0
        ... (second order probabilities)
```

The first note of the tune is generated using first order probabilities (there is no previous note on which to base its selection). From then on, a note is generated using the row of the second-order probability table that corresponds to the previous note played. This eliminates the occasional violent leaps in pitch that occurred with the previous version of the program. The second-order probabilities associated with such violent leaps are mostly 0. The use of second order probability thus encourages the program to use commonly acceptable pitch intervals between consecutive notes.

Third order probability distributions

If we want the program to use commonly used sequences of notes, we can move on to third order distributions where the probability of choosing a note will depend on the two previous notes played. The next table gives the results of a third order analysis of our 9 tunes.

Each possible sequence has a probability distribution associated with it and that distribution is used to choose the next note. Many of the rows in a complete version of this table would contain all zeros - if you consult the previous table, you will see that many combinations of two notes never occur.

The next program uses the first order distribution to choose its starting note, the second order distributions to choose the second note and from then on it uses the third order distributions.

```
        :
        :
  100   PROCsetupfreqtable1
  101   PROCsetupfreqtable2
  102   PROCsetupfreqtable3
        :
        :
  610   DEF PROCplaynote(finalnote)
  620     IF finalnote THEN pitch=keynote
          ELSE IF notesplayed=0 THEN PROCselectpitch1
          ELSE IF notesplayed=1 THEN PROCselectpitch2
          ELSE PROCselectpitch3
        :
        :
 1020   DEF PROCsetupfreqtable3
 1030   LOCAL lb1,l,n
 1040     DIM freq3(12,12,12)
 1050     REPEAT
 1060       READ lb1,l
 1070       FOR n=1 TO 12
 1080         READ freq3(lb1,l,n)
 1090       NEXT n
 1100   UNTIL lb1=0
 1110   ENDPROC
```

Previous two note numbers		frequency distributions of following notes											
1	1	0.0	0.0	0.0	33.3	33.3	16.7	0.0	16.7	0.0	0.0	0.0	0.0
1	4	0.0	0.0	33.3	33.3	0.0	33.3	0.0	0.0	0.0	0.0	0.0	0.0
1	5	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	6	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0
2	3	0.0	0.0	50.0	50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	5	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1	66.7	0.0	0.0	33.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	3	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	4	0.0	0.0	0.0	50.0	16.7	16.7	16.7	0.0	0.0	0.0	0.0	0.0
3	5	33.3	0.0	33.3	0.0	0.0	0.0	33.3	0.0	0.0	0.0	0.0	0.0
4	3	28.6	0.0	0.0	42.9	28.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	4	0.0	0.0	5.0	50.0	10.0	15.0	15.0	5.0	0.0	0.0	0.0	0.0
4	5	37.5	0.0	0.0	0.0	25.0	25.0	12.5	0.0	0.0	0.0	0.0	0.0
4	6	0.0	20.0	0.0	20.0	0.0	0.0	30.0	30.0	0.0	0.0	0.0	0.0
4	7	0.0	16.7	0.0	0.0	33.3	50.0	0.0	0.0	0.0	0.0	0.0	0.0
4	8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0
5	1	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	3	25.0	0.0	0.0	50.0	25.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	4	0.0	0.0	36.4	0.0	36.4	18.2	0.0	9.1	0.0	0.0	0.0	0.0
5	5	0.0	0.0	0.0	20.0	53.3	6.7	6.7	13.3	0.0	0.0	0.0	0.0
5	6	0.0	0.0	0.0	28.6	14.3	14.3	28.6	14.3	0.0	0.0	0.0	0.0
5	7	0.0	0.0	0.0	0.0	80.0	20.0	0.0	0.0	0.0	0.0	0.0	0.0
5	8	0.0	0.0	0.0	0.0	0.0	0.0	33.3	66.7	0.0	0.0	0.0	0.0
6	2	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	4	0.0	0.0	0.0	50.0	10.0	20.0	20.0	0.0	0.0	0.0	0.0	0.0
6	5	0.0	0.0	0.0	53.3	20.0	6.7	13.3	6.7	0.0	0.0	0.0	0.0
6	6	0.0	0.0	0.0	0.0	25.0	50.0	8.3	16.7	0.0	0.0	0.0	0.0
6	7	0.0	0.0	0.0	0.0	12.5	0.0	0.0	87.5	0.0	0.0	0.0	0.0
6	8	0.0	0.0	0.0	0.0	0.0	60.0	10.0	20.0	10.0	0.0	0.0	0.0
7	2	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	5	0.0	0.0	42.9	0.0	14.3	42.9	0.0	0.0	0.0	0.0	0.0	0.0
7	6	0.0	0.0	0.0	12.5	56.3	18.7	0.0	12.5	0.0	0.0	0.0	0.0
7	7	0.0	0.0	0.0	0.0	0.0	66.7	33.3	0.0	0.0	0.0	0.0	0.0
7	8	0.0	0.0	0.0	12.5	0.0	0.0	37.5	37.5	12.5	0.0	0.0	0.0
8	4	0.0	0.0	50.0	50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	5	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	6	0.0	0.0	0.0	33.3	22.2	11.1	11.1	22.2	0.0	0.0	0.0	0.0
8	7	0.0	0.0	0.0	0.0	0.0	75.0	16.7	8.3	0.0	0.0	0.0	0.0
8	8	0.0	0.0	0.0	7.7	7.7	15.4	15.4	23.1	30.8	0.0	0.0	0.0
8	9	0.0	0.0	0.0	0.0	0.0	0.0	5.6	22.2	22.2	50.0	0.0	0.0
8	11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0
8	12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0
9	7	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0
9	8	0.0	0.0	0.0	0.0	0.0	0.0	71.4	14.3	14.3	0.0	0.0	0.0
9	9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	16.7	0.0	83.3	0.0	0.0
9	10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	25.0	6.3	0.0	68.8	0.0
9	12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	50.0	0.0	50.0
10	8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0
10	9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	16.7	33.3	33.3	0.0	16.7
10	10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	10.0	65.0	0.0	15.0
10	11	0.0	0.0	0.0	0.0	0.0	9.1	0.0	9.1	9.1	0.0	36.4	36.4
10	12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	66.7	0.0	0.0	33.3	0.0
11	6	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0
11	8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0
11	9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0
11	10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	37.5	62.5	0.0	0.0
11	11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	45.5	54.5	0.0
11	12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	50.0	50.0	0.0	0.0	0.0
12	8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	66.7	0.0	0.0	33.3
12	9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	50.0	0.0	0.0	0.0	50.0
12	10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0
12	11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0
12	12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0

```
 1120  DEF PROCselectpitch3
 1130  LOCAL rand,n,sum
 1140  rand=RND(1)*100
 1150  n=0 : sum=0
 1160  REPEAT
 1170  n=n+1
 1175  sum=sum+freq3(lastbut1,lastnoteplayed,n)
 1180  UNTIL sum>=rand
 1190  pitch=scalenote(n)
 1200  lastbut1=lastnoteplayed : lastnoteplayed=n
 1210  ENDPROC
10010  ...
20010  ...
30010  DATA 1, 1, 0.0, 0.0, 0.0, 33.3, 33.3, 16.7,
       0.0, 16.7, 0.0, 0.0, 0.0, 0.0
30020  DATA 1, 4, 0.0, 0.0, 33.3, 33.3, 0.0, 33.3,
       0.0, 0.0, 0.0, 0.0, 0.0, 0.0
       ... (third-order frequencies)
```

The music generated by the program faithfully imitates the banality of the source material.

5.9 A program generate prolatbility frequency tables

In case you want to analyse your own favourite type of music, we present the following program which was used to generate the distributions of the three frequency tables.

```
   10  DIM freq1(12), freq2(12,12), freq3(12,12,12)
   20  INPUT "Number of tunes",nooftunes
   30  FOR tune=1 TO nooftunes
   40    PROCanalysetune
   50  NEXT tune
   60  PROCstandardisetable1
   70  PROCstandardisetable2
   80  PROCstandardisetable3
   90  PROCoutputtable1
  100  PROCoutputtable2
  110  PROCoutputtable3
  120  END

  130  DEF PROCanalysetune
  140  LOCAL scale$,lastbut1$,last$,next$,
       lastbut1,last,next
  150    READ scale$
  160    READ lastbut1$,last$,next$
  170    lastbut1=INSTR(scale$,lastbut1$)
  180    last=INSTR(scale$,last$)
  190    freq1(lastbut1)=freq1(lastbut1)+1
  200    freq1(last)=freq1(last)+1
  210    freq2(lastbut1,last)=freq2(lastbut1,last)+1
```

```
220     REPEAT
230       next=INSTR(scale$,next$)
240       IF next=0 THEN PRINT "Error in DATA ";tune;
          " ";scale$;" ";next$
250       freq1(next)=freq1(next)+1
260       freq2(last,next)=freq2(last,next)+1
270       freq3(lastbut1,last,next) =
            freq3(lastbut1,last,next)+1
280       lastbut1=last:last=next
290       READ next$
300     UNTIL next$="Z"
310   ENDPROC

330   DEF PROCstandardisetable1
340   LOCAL n, total
350     total=0
360     FOR n=1 TO 12
370       total=total+freq1(n)
380     NEXT n
390     FOR n=1 TO 12
400       freq1(n)=freq1(n)*100/total
410     NEXT n
420   ENDPROC

440   DEF PROCstandardisetable2
450   LOCAL l,n
460     FOR l=1 TO 12
470     total=0
480     FOR n=1 TO 12
490       total=total+freq2(l,n)
500     NEXT n
510     IF total>0 THEN
          FOR n=1 TO 12: freq2(l,n)=freq2(l,n)*100/total:
          NEXT n
520     NEXT l
530   ENDPROC

540   DEF PROCstandardisetable3
550   LOCAL lb1,l,n,total
560     FOR lb1=1 TO 12
570       FOR l=1 TO 12
580         total=0
590         FOR n=1 TO 12
600           total=total+freq3(lb1,l,n)
610         NEXT n
620         IF total=0 THEN freq3(lb1,l,0)=0 ELSE
            FOR n=1 TO 12:
              freq3(lb1,l,n)=freq3(lb1,l,n)*100/total:
            NEXT n : freq3(lb1,l,0)=100
630       NEXT l
640     NEXT lb1
650   ENDPROC
```

```
  660  DEF PROCoutputtable1
  670  LOCAL n
  680    PRINT "10000 DATA ";
  690    @%=&20105
  700    FOR n=1 TO 12
  710      PRINT ;freq1(n); :IF n<12 THEN PRINT ",";
  720    NEXT n
  730    PRINT
  740  ENDPROC

  750  DEF PROCoutputtable2
  760  LOCAL l,n,lineno
  770    lineno=20010
  780    FOR l=1 TO 12
  790      @%=5
  800      PRINT lineno;" DATA ";
  810      @%=&20105
  820      FOR n=1 TO 12
  830        PRINT ;freq2(l,n) ; : IF n<12 THEN PRINT ",";
  840      NEXT n
  850      PRINT
  860      lineno=lineno+10
  870    NEXT l
  880  ENDPROC

  890  DEF PROCoutputtable3
  900  LOCAL lb1,l,n,lineno
  910    lineno=30010
  920    FOR lb1=1 TO 12
  930      FOR l=1 TO 12
  940        IF freq3(lb1,l,0)>0 THENPROCoutline3
  950      NEXT l
  960    NEXT lb1
  970    @%=&90A
  980    PRINT ;lineno;
           " DATA 0,0, 0,0,0,0,0,0,0,0,0,0,0,0"
  990    REM *** DATA terminator ***
 1000  ENDPROC

 1010  DEF PROCoutline3
 1020    LOCAL n
 1030    @%=5
 1040    PRINT ;lineno;" DATA ";lb1;",";l;",";
 1050    @%=&20105
 1060    FOR n=1 TO 12
 1070      PRINT ;freq3(lb1,l,n);:IF n<12 THEN PRINT ",";
 1080    NEXT n
 1090    PRINT
 1100    lineno=lineno+10
 1110  ENDPROC
```

```
1120  DATA gabcdEFGABCD, G,A,B,G,A,A,B,C,C,B,B,G,A,B,
      G,A,A,B,C,D,G,D,D,C,B,A,B,C,D,A,D,D,C,B,A,B,C,
      D,A,G,A,B,G,A,A,B,C,C,G,A,G,A,B,G,A,A,B,C,D,G,Z
1130  DATA defgaBCDEFGA, g,g,g,C,B,D,B,C,D,D,D,g,f,a,
      f,d,g,f,g,C,B,D,B,g,a,C,a,f,g,g,B,D,B,g,B,D,B,
      a,C,a,f,a,C,a,B,D,B,g,B,D,B,a,C,a,f,g,g,Z
1140  DATA defgaBCD, g,B,C,D,C,B,a,g,a,d,d,a,g,B,C,D,
      C,B,a,g,a,d,d,g,B,g,g,C,B,a,g,FALSE,a,d,d,a,g,B,C,
      D,C,B,a,g,a,d,d,g,Z
1150  DATA ***cdEFGABC,c,c,G,G,A,B,C,A,G,F,F,E,E,d,
      d,c,G,G,G,F,F,F,E,E,E,d,G,G,G,F,G,A,F,E,d,d,c,Z
1160  DATA ***cdEFGABC,E,F,G,G,A,B,C,E,E,G,G,A,B,C,
      G,C,C,B,B,A,A,G ,G,A,G,F,E,d,c,Z
1170  DATA ***gaBCDE,D,B,D,D,B,D,E,D,C,B,a,B,C,D,
      g,g,g,g,g,a,B,C,D,D,a,a,C,B,a,g,Z
1180  DATA *******GABCD,B,B,B,B,B,B,B,D,G,A,B,C,C,C,
      C,C,B,B,B,B,A,A,B,A,D,B, B,B,B,B,B,B,D,G,A,B,C,
      C,C,C,C,B,B,B,D,C,B,A,G,Z
1190  DATA defgaBCDE,D,E,D,C,B,g,g,a,B,a,g,f,d,d,
      D,E,D,C,B,g,g,B,e,f,g,B,g,C,a,B,g,g,C,e,a,g,f,
      d,d,B,g,C,a,B,g,g,B,e,f,f,g,Z
1200  DATA fgabcDEF,f,b,b,b,b,b,a,b,e,c,c,c,c,c,
      D,D,D,D,F,E,D,D,c,c,c,c,F,F,D,D,D,D,D,E,c,c,c,
      c,F,E,D,c,b,c,c,b,a,b,b,b,b,b,Z
```

The string at the start of the DATA for a tune establishes the range of notes for the tune starting three notes below the keynote. There then follow the names of the notes in the tune in the order in which they appear. The program! prints the tables in the form of DATA statements numbered from 10000 upwards that can be absorbed into another program. To do this, type

```
*SPOOL "freqtables"
RUN
*SPOOL
```

and the DATA statentents for the tables will be stored on cassette. These can be added to any program by typing

```
*EXEC "freqtables"
```

Note that as is stands the program does not cater for accidentals and would have to be extended for these.

Now the question is: does using probability tables to mimic a musical genre produce anything worth listening to. One of the problems with this method is that you can make music more and more 'Bach-like' or 'Mozart-like' by using higher and higher probability orders, but as the music becomes more and more like the target style it becomes less and less original. In the limit if you take a high enough order probability distribution, you are taking so much

information from say Bach tune that the program will eventually generate an actual Bach tune (give or take a few notes).

So finally we return to letting the computer do its own thing and get it to play some 12 bar blues.

5.10 Micro blues

The next program plays or improvises on a 12 bar blues. It does not use probability tables but selects notes from two jazz blues scales (Bb and Eb, DATA statement 860). It utilises a rhythmic chordal accompaniment and the three voices are synchronised using PROCinitialise, PROCharmonise and PROCsound which were described earlier. Voices 2 and 3 consist of a simple blues chord progression taken from DATA statements 700 and 710. These are loaded up into rows 2 and 6 of the three row pitch and duration arrays. PROCjazz initialises row 1 of this array by randomly selecting starting notes for a phrase from the appropriate scale. The rhythm for a phrase is randomly selected from a set of DATA statements (1301 onwards).

```
 10  ENVELOPE 1,1,0,0,0,0,0,0,63,10,0,-63,63,126
 20  ENVELOPE 2,1,0,0,0,0,0,0,126,-4,0,-100,126,100
 30  ENVELOPE 3,1,0,0,0,0,0,0,126,-4,0,-100,126,100
 40  DIM pitch(3,200), duration(3,200),
       noofnotes(3), nextnote(3), clock(3)
 50  tempo = 1
 60  PROCinitialise(2)
 70  PROCinitialise(3)
 80  PROCjazz
 90  PROCharmonise(3)
100  END

200  DEF PROCinitialise(voice)
       .
       . as before
       .
390  ENDPROC

400  DEF PROCharmonise(noofvoices)
       .
       . as before
       .
550  ENDPROC

600  DEF PROCsound(voice)
       .
       . as before
       .

680  ENDPROC
```

```
 700  DATA 24,A#,h,A#,dq,A#,e,R,e,A#,e,A#,e,R,q,A#,dq,
      A#,e,R,w,C'#,h,C'#,dq,C'#,e,R,w,A#,h,A#,dq,
      A#,e,R,w,D'#,w,C'#,w,A#,h,A#,dq,A#,e,A#,w
 710  DATA 24, D',h,D',dq,D',e,R,w,D',e,D',e,R,q,D',dq,
      D',e,R,w,G,h,G,dq,G,e,R,w,D',h,D',dq,D',e,R,w,
      R,w,R,w,D',h,D',dq,D',e,D',w

 800  DEF PROCjazz
 810    DIM Bb(13), Eb(13)
 820    ii = 0
 830    FOR note=1 TO 13
 840      READ Bb(note)
 850    NEXT note
 860    DATA 45,57,65,69,73,85,93,105,113,117,121,133,14
 870    FOR note = 1 TO 13
 880      Eb(note) = Bb(note) + 20
 890    NEXT note
 900    PROCplaytheblues
 910    noofnotes(1)=ii
 920    ENDPROC

 930    DEF PROCplaytheblues
 940      PROCplaybars(4,"Bb")
 950      PROCplaybars(2,"Eb")
 960      PROCplaybars(6,"Bb")
 970    ENDPROC

 980  DEF PROCplaybars(n, key$)
 990    FOR bar = 1 TO n
1000      FOR phrase = 1 TO 2
1010      PROCselectstartnote
1020      PROCselectupdown
1030      PROCselectphrase
1040      PROCplayphrase(key$)
1050    NEXT phrase
1060  NEXT bar
1070  ENDPROC

1080  DEF PROCselectstartnote
1090    startnote = RND(13)
1100  ENDPROC

1110  DEF PROCselectphrase
1120    sphrase=RND(6)
1130    restoreto = 1300 + sphrase
1140    RESTORE restoreto
1150  ENDPROC
```

```
 1160  DEF PROCplayphrase(key$)
 1170    READ noofnotes
 1180    note = startnote
 1190    FOR i = 1 TO noofnotes
 1200      READ length
 1210      ii =ii + 1: duration(1,ii) = length
 1220    IF key$ = "Eb" THEN pitch(1,ii) = Eb(note)
           ELSE pitch(1,ii) = Bb(note)
 1230      note=note+ updown
 1240      IFnote > 13 OR note < 1 THEN note=7
 1250    NEXT i
 1260  ENDPROC

 1270  DEF PROCselectupdown
 1280    IF RND(2) = 2 THEN updown= -1 ELSE updown= 1
 1290  ENDPROC

 1301  DATA 8,2,2,2,2,2,2,2,2
 1302  DATA 4,4,4,4,4
 1303  DATA 1,16
 1304  DATA 1,16
 1305  DATA 4,2,6,2,6
 1306  DATA 11,1,2,1,2,1,2,1,2,1,2,1
```

Exercises

1 Select your favourite style of music, analyse a sample, generate a set of probability frequency tables and produce a program that composes in that style. You may find that you need to experiment with the rhythm generating program in order to produce rhythms that are appropriate for your kind of music.

2 The 'creative' part of the 'micro blues' program could be significantly improved by adding more constraints. For example:

   (a) The intervals used in the phrases are all major or minor seconds (one step in the scale sequence - line 1220). The interval between notes could be varied.

   (b) Rests or gaps of silence should be introduced, space is very important in music.

   (c) Fast note phrases used consecutively should be followed by a long note.

   (d) Repetition of a phrase should be occasionally introduced.

   (e) There are further harmonic constraints if you know the blues progression.