EAR TRAINING

by
Glen B. Haydon
Epsilon Lyra, Inc.
Box 429 Star Route 2
La Honda, CA 94020

Introduction

Ear training is an important part of a musical education. It is different from music appreciation and playing instrumental music. Hearing has an analogy with vision. Most of us are sighted and we learn to appreciate many visual details. In contrast, we usually suppress the auditory details except for those associated with the spoken language. Sometimes we do not pay much attention to even the spoken language. We hear only what we want to hear.

The Problem

There are many characteristics of music including the pitch, the intervals between musical tones and the types of musical triads. We learn to recognize these characteristics in ear training. We do not need to recognize the differences between major, minor, augmented, and diminished triads to appreciate music. But to develop a better appreciation and understanding of music, we can benefit from a course in ear training.

Well my wife, after several years of piano, decided to take some additional courses in music. Among those courses she had to go to a computer laboratory for the ear training. There, students used programs running on Apple computers to provide the ear training drills, including pitches, intervals, triads, scales and so on. It would be good to extend this practice at home. I use IBM clones. Evidently the Computer Laboratory did not have similar programs to run on the IBM family of computers. I made the rounds of several computer software stores and could not find any similar programs. So what was I to do?

The Conceptual Solution

Within a half hour I had my IBM clone playing individual notes. The rest was just a matter of developing a program that would provide the ear training exercises. After nearly 20 years of my wife's being a computer widow, she is taking an interest in computers.

The problem is really simple. The IBM clones can set the pitch and turn on and off the speaker. The value 0B6h sent to port 43 allows a calculated 16 bit divisor to be sent to the 8 bit port 42 as two eight bit values, the low value first. Then toggling a bit in port 61 turns on and off the speaker. An appropriate delay time can be used.

Program Utilities

The first function to be defined is a delay. A do nothing loop will serve the purpose. The duration will depend upon the speed of the processor. The easiest way is to make a simple loop and time it with your watch. You can adjust the number of iterations to give you a sufficiently accurate measure of the time in milliseconds, or any other unit. This is the only value that you will have to adjust for this program to run on an IBM clone system at a different speed.

```
{
: MSEC     0 DO 9 0 DO LOOP    LOOP ;
}
```

Changing the value, 9, in the source and recompiling the program, is probably more difficult than changing the value of `DURATION` in the compiled program where it is used. This can be done from the run-time program. The name `MSEC` will no longer produce milliseconds, but the result can be used the same way.

For ear training, it would be desirable to present the training examples in random order. I stole the code for `RND`, `RANDOM`, and `CHOOSE` Leo Brodie's, *Starting Forth.* It is a classical reference and adequate for this purpose.

```
{
CREATE RND HERE ,
: RANDOM  RND @ 31421 * 6927 + DUP RND ! ;
: CHOOSE  (u1 --- u2 )
     RANDOM U* SWAP DROP ;
}
```

The items to be chosen are placed in arrays later in the program.
A formatting tool allows multiple carriage returns.

```
{
: CRS      0 DO CR LOOP  ;
}
```

We would also like to position the responses at the same place on the screen each time.

```
{
: POSITION     PAGE 10 CRS    15 SPACES ;
}
```

Primitives

The primitive function we need will take as parameters, a pitch in tenths of a hertz and the duration in milliseconds. As described above, we need to convert the desired pitch to the required divisor value for the 8253 chip. The required value can be calculated from a system dependent constant divided by the desired pitch. That constant is a double precision value. I guess it could be calculated easily enough, but it is far easier to find the correct value by experiment. The value can be tuned with a tuning fork or with a tuned piano. If you find the value I have used to be off for your ear, just tune it yourself. The calculated value is stored in the appropriate channel of the 8253 through ports 43 and 42 as described. Once the value is latched into the register, you need to turn the speaker on for the desired delay and then turn the speaker off through port 61. For more details of the algorithm see any manual on the IBM PC.

```
{
HEX
: NOTE    ( pitch msec - )
     B6 43 P!
     B04000. 4 ROLL U/MOD SWAP DROP
     DUP FF AND 42 P!
     100 /      42 P!
     61   P@   DUP 03 OR   61 P!  SWAP
     MSEC 61 P!     ;
DECIMAL
}
```

Try this function with 256 for middle C, and 10000 for 10 seconds. The double precision value can be adjusted for tuning and the time used will show you how different your system is from mine.

A Tempered Scale

The next problem is to set up the pitches for the 12 notes of the chromatic scale. For each musical key, the diatonic frequencies vary a little with each note. As with a piano, the scale can be tempered. The easiest way is to find a music book that discusses the tempering of the scale. This I have done and assigned the appropriate pitch to each name.

```
{
2560 CONSTANT C
2712 CONSTANT C#
2873 CONSTANT D
3044 CONSTANT D#
3225 CONSTANT E
3417 CONSTANT F
3620 CONSTANT F#
3835 CONSTANT G
4063 CONSTANT G#
4304 CONSTANT A
4561 CONSTANT A#
4832 CONSTANT B
}
```

Next we want to play the notes. For this we first define a `DURATION`.

```
{
```

```
CREATE DURATION 4000 ,
}
```

On my system this is approximately 4 seconds. The value can be easily modified from the run-time program:

```
1000 DURATION !
```

The new duration will be one quarter as long.

Next, we will eventually want to change the keys for our exercises. We do this by defining a variable that can be changed during the program.

```
{
CREATE KEYS 256 ,
}
```

Then we want to play a note of a selected pitch on the computer. The `DURATION` of each note will be the same.

```
{
: PLAY ( pitch )
KEYS @ 256 */ DURATION @ NOTE ;
}
```

Finally, we can define the 12 notes for the middle octave. By changing the value in `KEYS`, the tempered scale can be placed anywhere.

```
{
: C1      C    PLAY ;
: C#1     C#   PLAY ;
: D1      D    PLAY ;
: D#1     D#   PLAY ;
: E1      E    PLAY ;
: F1      F    PLAY ;
: F#1     F#   PLAY ;
: G1      G    PLAY ;
: G#1     G#   PLAY ;
: A1      A    PLAY ;
: A#1     A#   PLAY ;
: B1      B    PLAY ;
: C2      C 2* PLAY ;
}
```

A flat symbol is not available on my computer but a sharp symbol is. In a tempered scale a C sharp has the same pitch as a D flat. That problem is easily solved. The twelve notes of a chromatic scale beginning with middle C are given a postscript of 1 indicating the middle octave. Adjust the duration of the note to suit your problem.

The Exercises

The next goal is to develop the ear-training exercises.
The first exercise will be learning to recognize a series of triads: major, minor, augmented and diminished. Assign the appropriate pitches to functions with the appropriate name.

```
     {
     : MAJOR        C1 E1  G1  E1  C1 ;
     : MINOR        C1 D#1 G1  D#1 C1 ;
     : DIMINISHED   C1 D#1 F#1 D#1 C1 ;
     : AUGMENTED    C1 E1  G#1 E1  C1 ;
     }
```

Each name will execute the selected triad in the selected key and with the selected `DURATION` for each note.

The code fields of the names of the triads are stored in an array from which they can be randomly chosen.

```
{
CREATE triads
     ' MAJOR      CFA ,
     ' MINOR      CFA ,
     ' DIMINISHED CFA ,
     ' AUGMENTED  CFA ,
}
```

The program can then choose them randomly from that array. Also we will want to select the keys at random. Therefore, we define an array of keys. Then we can choose one randomly or we could select any one key from the array.

```
{
CREATE keys
     ' C  CFA ,
     ' C# CFA ,
     ' D  CFA ,
     ' D# CFA ,
     ' E  CFA ,
     ' F  CFA ,
     ' F# CFA ,
     ' G  CFA ,
     ' G# CFA ,
     ' A  CFA ,
     ' A# CFA ,
    ' B  CFA ,
}
```

Next we will put together an exercise that will randomly select a key, one of the triads and play it. A pause after playing the triad allows the user to decide which key and triad were played. Striking any key will then display the answer followed by another pause to let the user think about his work. Pressing any key will terminate the pause and continue the program.

```
{
: TRIAD
    POSITION
    12 CHOOSE 2*  keys   + @ DUP EXECUTE KEYS
     4 CHOOSE 2 * triads + @ DUP EXECUTE
    BEGIN ?TERMINAL UNTIL
    POSITION
    SWAP ." Key of " 2+ NFA ID. 2+ NFA ID. ." triad. " CR
    BEGIN ?TERMINAL UNTIL ;
}
```

`CHOOSE` first selects the key and then the triad to be played. The program must remember both the selected musical key and the triad so that the correct answer can be displayed when ready. Next we can write a function, `TRIADS`, to repeat the function, `TRIAD`.

```
{
: TRIADS
    CR 100 0 DO TRIAD ?TERMINAL IF LEAVE THEN LOOP ;
}
```

You can terminate the exercise by holding down any key until the menu appears. This is done by the conditional leave construct.

Intervals

A similar pattern can be used for a variety of other ear training exercises. The laboratory my wife used included intervals. For this the series of intervals is defined with the appropriate names.

```
{
: Perfect_unison   C1 C1  ;
: Minor_second     C1 C1# ;
: Major_second     C1 D1  ;
; Minor_third      C1 D#1 ;
: Major_third      C1 E1  ;
: Perfect_fourth   C1 F1  ;
: Augmented_fourth C1 F#1 ;
: Perfect_fifth    C1 G1  ;
: Minor_sixth      C1 G#1 ;
: Major_sixth      C1 A1  ;
: Minor_seventh    C1 A#1 ;
: Major_seventh    C1 B1  ;
: Perfect_octave   C1 C2  ;
}
```

The code fields of these names are placed in an array from which they can be randomly chosen.

```
{
CREATE intervals
     ' Perfect_unison   CFA ,
     ' Minor_second     CFA ,
     ' Major_second     CFA ,
     ' Minor_third      CFA ,
     ' Major_third      CFA ,
     ' Perfect_fourth   CFA ,
     ' Augmented_fourth CFA ,
     ' Perfect_fifth    CFA ,
     ' Minor_sixth      CFA ,
     ' Major_sixth      CFA ,
     ' Minor_seventh    CFA ,
     ' Major_seventh    CFA ,
     ' Perfect_octave   CFA ,
}
```

To start with, it is easier to learn the intervals in only one key. We will start with the key of C. The key of C is the first item, beginning with 0, in the array of keys. The form is maintained for later use in selecting random keys.

```
{
: INTERVAL
     POSITION
     0 2* keys + @  DUP EXECUTE    KEYS
     13 CHOOSE 2* intervals + @ DUP EXECUTE
     BEGIN ?TERMINAL UNTIL
     POSITION
     SWAP ." Key of " 2+ NFA ID. 2+ NFA ID. ." Interval. "
     CR
     BEGIN ?TERMINAL UNTIL ;
}
```

Once the function for a single `INTERVAL` is correct, we can define multiple `INTERVALS`.

```
{
: INTERVALS
     CR 100 0
     DO PAGE 12 CRS INTERVAL
```

```
          ?TERMINAL IF LEAVE THEN LOOP ;
     }
```

Mixed Keys and Intervals

Later, using a similar function we can randomly select first the musical key and then the interval.

```
     {
     : KEY&INTERVAL
          POSITION
          12 CHOOSE 2* keys       + @ DUP EXECUTE KEYS
          13 CHOOSE 2 * intervals + @ DUP EXECUTE
          BEGIN ?TERMINAL UNTIL
          POSITION
          SWAP ." Key of " 2+ NFA ID. 2+ NFA ID. ." Interval. "
          CR
          BEGIN ?TERMINAL UNTIL ;
     }
```

Then we can execute the single function repeatedly.

```
     {
     : KEY&INTERVALS
          CR 100 0
          DO KEY&INTERVAL
          ?TERMINAL IF LEAVE THEN LOOP ;
     }
```

These examples provide a few exercises for drill on the basics of an ear training program. The exercises can be extended to scales, and more complicated chords. That is for the programmer to play with. These will give the beginner to ear training a good start.

Menu

There is another problem for the music student who has never worked with a computer and does not know how to type. To solve that, the exercise options can be put into a simple menu. The menu could be patched into the program so that it is displayed at the start making it a turnkey program.

```
     {
     : MENU
     PAGE 4 CRS 10 SPACES
          ." A TRIADS        " 2 CRS 10 SPACES
          ." B INTERVALS     " 2 CRS 10 SPACES
          ." C KEY&INTERVALS " 2 CRS 10 SPACES
          ." D QUIT          " 3 CRS 10 SPACES
```

```
        ." Enter the letter for the drill of your choice."
        5 CRS ;
    }
```

Each letter is defined as an alias for the desired exercise. Adding `MENU` to the alias will return the user to the menu.

```
    {
    : A  TRIADS        MENU ;
    : B  INTERVALS     MENU ;
    : C  KEY&INTERVALS MENU ;
    : D  PAGE   BYE         ; EXIT
    }
```

After the menu is displayed, you are actually in the programming language and can execute any of the available language functions as well as the particular exercises. At this point, the value for `DURATION` can be adjusted.

Compile this paper and the program will run!

A Postscript

I developed this source code as any programmer might. It required the definition of the problem, a programmer, a writer, an editor, a graphics artist, a printer and a good proofreader. After I decided what I wanted, I first wrote the program as I do most of mine. I start with Forth screens and left them a mess. After everything was working about the way I had in mind, I reformatted the screens, set off portions with plenty of white space and often chose more descriptive names. I then took my small portable computer to bed and drafted the text that I later uploaded to my main system. Then I converted the revised Forth screens to a text file. With my word processor, I merged my text and the source code. Along the way I edited and reedited the text many times. I used my spelling and grammar programs to review the text. Finally, I moved the edited text file to my typesetting program where I formatted the output for printing. I then printed it. Finally, there were several cycles of proofs and corrections before printing the publication copy.

Conclusion

This paper is actually the source code for my musical ear training program. The text file compiles. It is written for the reader. The compiler selects only those portions enclosed in braces for compilation.

Any project requires a report. Why not make the report the source code?

Note

Since the FORML meeting a real musician found the frequencies to the nearest cycle disturbing. The original paper has been revised and now has the frequencies set to the nearest tenth of a cycle. Changes to the system dependent constant used in `NOTE` and the frequencies have been made. It is better! It could be carried to the nearest hundredth of a cycle with some more work. (22 December 1990)