ED 392 433 IR 017 724

AUTHOR	Crenshaw, John H.
TITLE	Music and Computers: Symbiotic Learning.
PUB DATE	95
NOTE	8p.; In: "Emerging Technologies, Lifelong Learning, NECC '95"; see IR 017 705.
PUB TYPE	Reports - Descriptive (141) -- Speeches/Conference Papers (150)
EDRS PRICE	MF01/PC01 Plus Postage.
DESCRIPTORS	*Computer Uses in Education; Higher Education; High Schools; *Learning Activities; *Music Activities; *Programming; Student Interests; *Student Projects
IDENTIFIERS	BASIC Programming Language; PASCAL Programming Language

ABSTRACT

Many individuals in middle school, high school, and university settings have an interest in both music and computers. This paper seeks to direct that interest by presenting a series of computer programming projects. The 53 projects fall under two categories: musical scales and musical sound production. Each group of projects is preceded by a short discussion of the relevant underlying musical concepts; in this way, a person who has an interest in music but lacks a strong background should be able to complete the projects without having to refer to outside sources. The musical scales projects do not attempt to produce any sounds and can be solved using any language on any type of hardware. Most of the projects in the musical sound production group do create sound output and require access to a microcomputer system running Pascal, Basic, or another language which allows the user to create sound output. None of the projects require the use of MIDI boards or any external instrumentation. (Author/AEF)

Music and Computers: Symbiotic Learning

by John H. Crenshaw

Paper presented at the NECC '95, the Annual National Educational Computing Conference (16th, Baltimore, MD, June 17-19, 1995.

IR017734

paper

Music and Computers: Symbiotic Learning

John H. Crenshaw
Western Kentucky University
Bowling Green, KY 42101
(502) 745-4642
crenshaw@wkuvx1.wku.edu

Key words: music, computers, programming, sound

Abstract

This paper presents a series of programming projects involving elementary concepts from the field of music. These projects are designed for individuals who possess a fundamental interest in music and computers, and wish to further develop their abilities. Since the projects are intended to be completely self-contained, students or instructors with limited musical background should be able to absorb the necessary musical concepts as they progress through the projects.

Introduction

Many individuals in middle school, high school, and university settings have an interest in both music and computers. This interest, if properly directed and developed, can lead to a deeper knowledge and appreciation of both disciplines.

This paper seeks to help direct that interest by providing a series of computer programming projects. Each project is built around an elementary musical concept. Each group of projects is preceded by a short discussion of the relevant underlying musical concepts. In this way, a person who has an interest in music but lacks a strong background should be able to complete the projects without having to refer to outside sources.

Similarly, as the projects are completed, a person will be developing a stronger ability in programming since the projects are ordered by increasing programming difficulty also.

The projects of the next section, **Musical Scales**, do not attempt to produce any sounds. These projects could be solved using any language on any type of hardware. The following section, **Musical Sound Production**, does create sound output in most of its projects. These projects will need access to a microcomputer system running Pascal, Basic, or another language which allows the user to create sound output in some relatively straightforward manner. No projects require the use of MIDI boards or any external instrumentation.

Musical Scales

A pitch in music is a continuous tone which has a specific frequency expressed as vibrations per second or hertz. In Western music, an equal temperament scale or tempered scale is composed of 12 distinct pitches such that each successive pair of pitches has an identical frequency ratio, namely the 12th root of 2. That is, if X and Y are 2 adjacent pitches with Y being higher than X, then Frequency(Y)/Frequency(X) is the 12th root of 2. An octave is defined to be any sequence of 12 successive pitches.

(1.) Write a program to input a starting frequency and then compute and print out the frequencies for each of the 12 pitches in that octave.

A piano has 88 keys tuned to the tempered scale. The human ear is able to hear sounds in the range 20-20,000 Hertz.

(2.) Design an algorithm to input a starting frequency for the lowest key on a piano and compute the resulting frequency for the highest key on a piano. (27.5 is a traditional starting frequency.)

(3.) Design an algorithm to calculate what frequencies would be possible for the lowest key on a piano in order to ensure that the highest key would still be audible.

The keys on a piano are given names:

A,(A#,Bb),B,C,(C#,Db),D,(D#,Eb),E,F,(F#,Gb),G,(G#,Ab)

The notes A,B,C, etc.. are the <u>white</u> keys. The notes in parentheses are the <u>black</u> keys. Each black key may be called by either name in parentheses in the projects defined here. Additionally, each key is identified by the octave in which it is located using the numbers 0 to 8. Thus D3 would be the D in the third octave from the bottom. Each octave begins with a C. Thus B4 would be followed by C5. The lowest piano key is A0. The highest piano key is C8.

(4.) Design an algorithm to input a starting frequency and a starting pitch name and print out a table showing the pitch names and the frequencies for the 12 pitches in that octave.

(5.) Design an algorithm to input a starting frequency and a starting pitch name for the lowest key on a piano and then print out a table showing the pitch names and the frequencies for all 88 keys. (Use A0 with a frequency of 27.5 as a starting point. In this case, A4 should end up being close to 440 Hz which is the international tuning frequency.)

(6.) Write a function or procedure which will accept a pitch name as a character string (e.g. C#5) and return the frequency of that pitch assuming A0 is 27.5.

Musical instruments have a <u>range</u> which defines the pitches they are capable of producing. For example, the piano range is A0 to C8. A violin has a range of G3 to approximately E7.

(7.) Design an algorithm to input the range of a given musical instrument by entering the lowest and highest pitch names which can be produces. Print out these ranges and the corresponding frequencies which can be produced.

(8.) Design an algorithm to input the names and ranges for several musical instruments and store them in an appropriate data structure and/or external file. Prepare and print a graph in which the horizontal axis is the range of audible frequencies 20-20,000 and each instrument is presented as a horizontal line corresponding to its frequency range. Label the axis with frequency numbers and/or pitch names.

(9.) Design an algorithm to input the names and ranges for several musical instruments and print out the range which is common to all those instruments.

(10.) Design an algorithm to input the names and ranges for several musical instruments and then input a desired range. Print out the instrument(s) which are capable of producing all pitches in the desired range.

(11.) Design an algorithm to input the names and ranges for several musical instruments. Print out the range(s) which represent the pitches which are possible by at least one of the instruments. Note that the resulting range may be continuous or fragmented; there may be gaps which are not covered by any instrument.

There are other tuning scales which predate the tempered scale which are also based on mathematical relations between different notes in the scale. The <u>just intonation scale</u> is built on the following ratios for an octave beginning with C:

C = 1/1	E = 5/4	G# = 25/16	C = 2/1
C# = 25/24	F = 4/3	A = 5/3	
D = 9/8	F# = 45/32	A# = 225/128	
D# = 75/64	G = 3/2	B = 15/8	

That is, given the frequency, FREQ, for a pitch C, the frequency for a pitch F will be 4/3 * FREQ. Similarly, the frequency for G# will be 25/16 * FREQ. This scale is seldom used today because of several difficulties. Because there is not a fixed ratio between any pair of successive notes, a user must compute all frequencies based on a starting octave pitch. The same note, say D5, will have different calculated frequencies depending on the starting octave pitch used.

(12.) Design an algorithm to enter and store the just intonation ratios. Enter a starting pitch frequency and print out a table of frequencies for each of the notes in once octave starting with the entered frequency. Print the note name and the frequency for each pitch.

(13.) Design an algorithm to enter and store the just intonation ratios. Compute and print out a table showing the frequency ratio between each of the 12 successive notes in one octave.

(14.) Design an algorithm to enter two different starting frequencies and print out a table of frequencies for one octave for each of the starting frequencies. Notice if the same pitches have different frequencies in the two tables. Use C4 = 261.63 and F4 = 348.84 in one test, and C4 = 261.63 and D4 = 294.33 for another test.

(15.) Design an algorithm to enter a starting pitch, say C4 = 261.63, and print out a table of frequencies for one octave using the just intonation scale and the tempered scale. For each note, print out also the "error" between the two different values and the percent of error assuming the tempered scale is the "correct" value.

All the previous projects have assumed a scale composed of 12 distinct pitches per octave. Musicians and cultures have built scales using other number of pitches per octave. If each such scale is constructed using the principles of a tempered scale then the calculation of frequencies will be very simple; the ratio between any successive pair of pitches in a system with N pitches per octave will be the Nth root of 2.

(16.) Design an algorithm to enter an integer, N, and a starting frequency. Compute and print out a table showing the frequency for each pitch of a one octave scale having N different pitches using the tempered scale. If N=12 then the results should match the 12-tone tempered scale.

(17.) Design an algorithm to enter an integer, N, and a starting frequency. Compute the frequencies for each pitch in a one octave scale having N different pitches using the tempered scale. For each pitch in the N-tone scale, determine which pitch in the 12-tone tempered scale is closest in frequency. Print out this pitch from the 12-tone scale and the percentage of deviation of the N-tone pitch from its closest 12-tone pitch. Run your program with N values of 13 to 40.

Musical Sound Production

Many versions of BASIC include a PLAY command which allows a user to specify a single pitch or a series of pitches to be "played" on the speaker system of the computer. A typical PLAY command might look like the following:

```
PLAY "03 C D E F G2 F2 E2 D2 C"
```

In this version, "03" defines the octave being used and the other letters specify the sequence of pitches within that octave that are to be played. The "2" following certain pitches gives a relative length for those pitches; G2 will be one half as long as the pitches with no "2" following them. Many other options are available with the PLAY commands of various vendors. The exercises in this section are designed for users with access to a BASIC system who wish to investigate the PLAY statement further.

(18.) Write a BASIC program to allow a string to be read in and used as the command string for a PLAY command. Use this program to investigate the range of pitches which can be played on that system.

(19.) Use the same program to investigate the shortest and longest duration possible for a pitch in that system.

(20.) Use the program to investigate how to produce a sequence of pitches with one PLAY command.

(21.) Use the program to investigate how to produce a sequence of pitches which encompass more than one octave.

(22.) Use the program to investigate how to create accidentals such as Bb or F#.

(23.) Use the program to investigate whether 2 identical pitches played back to back have a noticeable break in the sound between them or if the result is simply equivalent to playing one pitch for twice as long. Does the system have any facilities for inserting a break or controlling how long it is?

(24.) Use the knowledge you have learned to create a string representing a melody line from a song. Enter it into a program and debug it until it sounds correct.

(25.) Investigate whether your system could permit a tempo (speed) control to be specified or changed at run time so that a given string could be played faster or slower without reentering the entire string. This might require a considerable amount of string processing.

(26.) Design a system which would permit a given melody to be transposed to a higher or lower range automatically. The melody string would be stored in a string variable and a numeric offset value would be read in. All pitches in the melody string would be adjusted up or down the number of pitches specified by the offset value.

(27.) Design an algorithm to play "The Twelve Days of Christmas" with all 12 verses. Your program should use repeated loops as much as possible so that the number of melody strings which must be stored would be as small as possible.

(28.) Design an algorithm which would store melody strings in a data file on disk. Allow the user to specify a desired string and then retrieve it from the disk file and play it. The user should be able to add and delete melody strings to the file.

(29.) Design an algorithm to allow a user to modify melody strings stored in a disk file. The user should be able to add, change, and delete individual notes in a specified melody string as well as listen to the string played. Thus, if a melody string had been stored before it was totally correct, the user may adjust it and store it back on the disk.

(30.) Label a set of keys on the keyboard to represent the keys on a piano keyboard. For example, the keys on the second row (ASDFGHJKL;') could represent the white keys and a portion of the keys above that row could be the corresponding black keys. Design an algorithm which would read in a series of keystrokes one at a time. Each time a "white" or "black" key is pressed, have the computer play the corresponding note for a given length of time. Continue reading and playing notes until the user strikes a specified key which denotes end of song.

Many versions of BASIC and Pascal for microcomputers include a SOUND command which will produce a sound of a specified frequency. In BASIC, the typical SOUND command has two parameters: an integer frequency value and an integer length value.

Turbo Pascal has a SOUND command or procedure which includes only an integer frequency argument. A complementary NOSOUND command with no arguments is necessary to terminate the sound begun by the SOUND command. The duration of the sound may be controlled by the use of a DELAY command which includes a single argument specifying the length of the delay. Thus a typical application in Turbo Pascal would require:

```
SOUND(Frequency_Value)
DELAY(Time_Length)
NOSOUND
```

(31.) Investigate the BASIC or Pascal system available to you. Determine the time units used for the Duration parameter of the SOUND command or DELAY command.

(32.) Design an algorithm to request a frequency from the user and produce a pitch of that frequency. Use the program to determine the range of audible pitches which your computer speaker can produce.

(33.) If you are using Turbo Pascal or a similar product using a SOUND-NOSOUND combination to specify duration, write a procedure called SOUND2 which has 2 parameters: frequency and duration. In future exercises, then, you may use SOUND2 in a manner similar to the BASIC SOUND command.

(34.) Use the SOUND, (or SOUND2), command to generate a siren which produces a continuous range of increasing and then decreasing frequencies. Experiment with the speed at which the frequencies should change and the range of frequencies in order to get a "pleasing" pattern.

(35.) Design an algorithm to enter a pitch name, A to G, and produce a sound of that pitch (in any suitable octave) for a reasonable time interval using the tempered scale.

(36.) Design an algorithm to enter a pitch name with a possible accidental attached such as A#, B or Eb and produce a sound of that pitch in any suitable octave for a reasonable time interval using the tempered scale.

(37.) Design an algorithm like (36.) but include an octave number such as A#3, B5, or Eb4.

(38.) Label a set of keys on the keyboard to represent the white and black notes of a piano. Design an algorithm to input a sequence of single character inputs and have the computer produce a sound of a given duration each time an input character matches a key defined in your "piano keyboard". Define a "stopping" key which will cause the program to terminate when that key is pressed.

(39.) Redo project (38.) to have a pitch continue sounding until another key is pressed. This will allow a melody to have pitches of different time lengths. You may want to set up just one key, (such as the space bar), which would cause the previous pitch to cease but not have another pitch begin.

(40.) Design a data structure using arrays which will allow you to store a series of note names such as A#3 in one character array and a series of durations in another integer array. This will define a melody line such that the first note name is to be played for the length of time defined by the first duration value, the second note is to be played for the length of time defined by the second duration value, etc. The two arrays used to store the notes and the durations are often referred to as parallel arrays.

(41.) Using the structure of (40.), design an algorithm to read in a set of values for notes and durations and store them in the arrays. Then have the program play the melody by examining each note/duration combination one at a time.

(42.) Design an algorithm to store the arrays of project (41.) onto a disk file after they have been created. Request the user to enter the desired file name.

(43.) Design an algorithm to enter a desired file, read in the array data from that file, and play the song represen ed by the data on that file.

(44.) Write an editing program which will allow a user to examine an existing song file. Each note/duration combination should be displayed on the screen along with its sequence number. (The first note has sequence number 1, the second note is number 2, etc.)

(45.) Expand the editing program of (44.) to allow a user to change the note and/or duration for any given sequence number. After all such changes are made, rewrite the data back onto the disk file.

(46.) Expand the editing program of (44.) to allow a user to add or delete note/duration entries. When a note/duration entry is added or deleted, refresh the screen display to show the new status of the song. The sequence numbers will now be different and the internal arrays will be different. After all such changes are made, rewrite the data back onto the disk file.

(47.) Combine all of the projects from (40.)-(46.) into one project which will allow a user to enter a melody, play a melody, retrieve a melody from disk, edit a melody, or write a melody onto disk.

(48.) Combine the project of (47.) with the keyboard playing project (38.) to allow the user to enter a melody from the "piano keyboard" and have it stored in the note/duration arrays. The user will still have to enter the duration values manually, but the entry of the note names will be much faster using the "piano keyboard" technique.

(49.) Using the program from (42.), (or any subsequent variation of that program), allow the user to transpose the melody up or down by N half-steps where N is read in by the user. If N = 3, for example, then the melody would be raised 3 half-steps. One half-step represents the interval between any 2 successive keys on the keyboard. Therefore, one half-step up from D3 would be D#3. Two half-steps up from D3 would be E3. Change the note values in the array to conform to the transposed note values so that the next time that melody is played, the new transposed notes will be used.

(50.) Write a new version of the transposition program in (49.) so that the user can enter the new starting note rather than the number of half-steps. Therefore, if a melody started with a D3, then the user could get the melody transposed up 3 half-steps by entering the new starting note of F3.

(51.) Modify the program from (49.) so that the user can input the desired transpose value, N, and have the melody played immediately in that transposed system. Do not change the original array values but simply play each note after applying the correct transpose shift to it.

(52.) Examine a melody like "Three Blind Mice" which has more than one repeated melody phrase. Design a program to permit each repeated phrase to be stored only once and then play the entire song by having the program play each phrase in the desired order. One possibility would be to have a different set of array names for each phrase. A more interesting possibility would be to design a play procedure which would play all the note/duration values of an array starting with sequence number START and continuing through sequence number STOP, where START and STOP are supplied to the procedure as parameters along with the array names. In this approach, the separate phrases could be stored back to back in just one set of arrays.

(53.) Examine the melody "The Twelve Days of Christmas". In this song, the melody covers Day 1, and then Day2 and Day1, and then Day 3 and Day2 and Day1, and so on. In addition, Day6 through Day 12 are identical. Devise an efficient collection of note/duration phrases to be stored in an array, and a program segment to play these phrases in order to create the entire song melody.