Guide to BOOK Chapter 1 Examples: *Richard Dobson*

Programming in C

Building the Examples

All the programs in this chapter comprise a single source file. For convenience, they are all included in the supplied *Makefile*, which can be used on OS X and Linux, and with gcc in Windows (if MinGW is installed). Separate instructions are provided below for those using Visual C++ 2005 or 2008. In all cases, compilation is initiated at the command line.

GCC-based Environments – *OS X, Linux, MinGW*

It is possible to build all the programs in Book Chapter 1 by issuing the single command:

```
$ make
```

However, it is recommended to build (and where appropriate, modify and rebuild) each program individually, as the corresponding chapter is read. For example:

```
$ make wonder
$ make sinetext
```

Note that the name supplied to the make command is that of the "target" – the name of the executable.

To run the executable from the build directory, remember that in all native Unix-based systems such as OS X it must be prefixed with the sign for the "current directory":

```
$ ./wonder
$ ./sinetext
```

Note that some of the programs employ functions from the C maths library. In the *Makefile* this is indicated by the argument **`-lm`**.

Windows, VC++

The recommended build environment on Windows is the MinGW Unix package.

The VC++ equivalent to *make* is *nmake*, which can be used to build a single program from the command line. Note that you may need to launch the special version of the DOS console available in the VC++ installation called e.g. "Visual Studio 2005 Prompt" (typically found from the Start menu via "All Programs"). This launches a console window with all the necessary paths set up to run the various command line tools, locate the compiler libraries, etc. The command is almost identical to the Unix form, except that the full executable name needs to be given:

```
> nmake sinetext.exe
```

The program can then be run in the usual way:

```
> sinetext
```

As in the Unix environment, *nmake* is able to build such simple single-file programs without an explicit Makefile. For a compound Makefile, VC++ uses a “project” file (e.g. one with the ***.vcproj*** file extension), which is created via the Visual Studio application (the “Integrated Development Environment” or IDE). See your compiler documentation for details. The recommendation remains to use the command-line environment to build all these programs, not least, as it is the same environment in which you will be running them.

Note that the compiler (“`cl`”) can be called directly (with no need to specify any maths library). In this case, you supply the name of the source file to `cl`:

```
> cl wonder.c
```

By default, `cl` knows to build compile the source file into an executable with the same root name.

Command-line Arguments

Some programs take command-line arguments. In all such cases, typing the program name by itself will result in a short “usage message” being printed to the console, giving the function of each argument. For example, running *cpsmidi* will produce:

```
$  ./cpsmidi
Usage: cpsmidi MIDInote
 range: 0 <= MIDInote <= 127
$
```

Supporting Tools – text2sf, Gnuplot and Audacity

The final programs in Book Chapter 1 (Sections 1.8, 1.9) introduce three tools with which to view or audition text data files, text soundfiles, and raw binary soundfiles.

text2sf

The *text2sf* program is a command-line utility to convert text soundfiles to standard WAVE or AIFF soundfiles. It is presently limited to output only 16bit samples. The soundfile format is set by the filename extension – .wav, aiff, etc.

Building *text2sf* on the Mac or Linux

1. Enter the subdirectory *text2sf*:

```
      cd text2sf
      ./make
```

This will build both the required library *libportsf.a* (See Book Chapter 2), and the executable *text2sf*.

2. Copy the executable into the parent directory ("examples"):

```
      cp text2sf ..
```

3. And return to that directory:

```
      cd ..
```

Building *text2sf* on Windows

A ready-built binary is provided in the *text2sf* folder. For the purposes of this chapter, simply copy *text2sf.exe* to your working directory. If you find it is more generally useful you might put it into a general "bin" folder somewhere on your system path, so it can be accessed from anywhere.

The *text2sf* Program

Usage:

```
      text2sf infile outfile srate chans gain

      infile   : input text file
      outfile  : output soundfile, with extension .wav, .aif, .aiff, or .aifc
      srate    : sample rate
      chans    : number of channels
      gain     : scale infile amplitude values
```

Example: convert text file to stereo 44100Hz 16bit WAVE file, at half amplitude.

```
$ ./text2sf sin440.txt sin440.wav 44100 2 0.5
```

Gnuplot (and Aquaterm)

Gnulot is used to plot general 2D data (among other things), read from input text files. These may be breakpoint files or soundfiles. Installers for OS X and Windows can be found in the subfolder "*gnuplot*". Note that OS X users first need to install **Aquaterm**, so that Gnuplot can output graphs to the native graphic environment. Linux users may well find (depending on the distribution) that they already have it. Otherwise, the best way to obtain it online is via the native package manager for that Linux distribution.

Note that the Windows version is called "*wgnuplot*" – in all the examples given below, substitute "`wgnuplot`" for "`gnuplot`".

The main Gnuplot website for information, tutorials and updates is: http://www.gnuplot.info

Note that Gnuplot provides internal semi-interactive documentation: type "`help`" at the Gnuplot prompt.

Audacity

Audacity is a cross-platform open-source soundfile editor and player. It is used in this Book Chapter for its facility to read and play raw binary soundfiles. Prebuilt binaries are available in the top-level Apps folder, for Window and OS X. Again, Linux users may already have it as part of their distribution. If not, it is best obtained via their native package manager system.

The main Audacity website is: http://audacity.sourceforge.net

Updates are relatively infrequent, but always significant.

The Book Chapter 1 Programs

(Windows users: omit the ./ prefix.)

wonder.c: prints a musical message to the console.

Example:

```
./wonder
```

midi2freq.c: reports the frequency of MIDI note number 73 in Hz.

Example:

```
./midi2freq
```

freq2midi.c: an interactive program to convert frequency to MIDI note and respond to prompt by giving a MIDI note number.

Example:

```
./freq2midi
```

cpsmidi.c: convert a MIDI note number to frequency, using a command-line argument.

Usage:

```
cpsmidi MIDInote
```

Example: find the frequency of MIDI note number 60 – middle-C.

```
./cpsmidi 60
```

nscale.c: generate a list of Equal-Temperament (ET) frequencies of an N-note scale (1 octave).

Usage:
```
nscale notes MIDInote
```

Example: find the frequencies of a 19tone ET scale from the A above middle C.

```
./nscale 19 69
```

iscale.c: generate an ET scale for an N-note octave. Uses command-line with optional flags and argument. Input is either MIDI note or raw frequency.

Usage:
```
iscale [-m] [-i]  iscale N startval  [outfile.txt]
```

`-m` : startval is MIDI note (otherwise: raw frequency in Hz).
`-i` : write the calculated interval ratio as well as frequency.
`outfile.txt` : optional; give name of an output text file to receive the data; otherwise, output is written to screen.

Example: write a 12tone scale from A below middle C to file *m57-12et.txt.*

```
./iscale -m 12 57 m57-12et.txt
```

breakdur.c: report duration and maximum value of an input breakpoint file.

Usage:
```
breakdur  infile.txt
```

(Note: the text file can have any reasonable extension, e.g. *.txt* or *.brk*).

Example: to report info for the file *envelope.brk.*

```
$  ./breakdur envelope.brk
read 7 breakpoints
duration: 2.000000 seconds
maximum value: 1.000000 at 0.750000 secs
$
```

expdecay.c: generate a basic exponential decay breakpoint file.

Usage:
```
expdecay  dur T steps
```

where T = Time constant (T > 0.0, typ. 0.0 < T <= 1)

Example: create medium decay over 200 points, redirected to text file, for display via Gnuplot.

```
$ ./expdecay 1 0.5 200 >expdecay.txt
$  gnuplot
...
>  plot "expdecay.txt" with lines
[ ... any number of other commands ... type q to exit Gnuplot.]
> q
$
```

expbrk.c: generate exponential attack or decay (always decays to target level), allowing arbitrary start and end values. Data written to screen - can be redirected to a file.

Usage:

```
     expbrk duration npoints startval endval
```

Example: generate a 1-second decay from 2.0 to -2.0, written to file *exp2.txt.*

```
$ ./expbrk 1 200 2 -2 >exp2.txt
```

sinetext.c: generate sinewave text data file of 50 points. Output is written to `stdout`. It has no arguments.

Example: output written to screen, redirected to a text file (e.g. for display using Gnuplot).

```
$ ./sinetext >sinwave.txt
```

sinetext2.c: create sinewave data of given srate, frequency, length. Data is written to the screen. Use redirection to output to a text file.

Usage:

```
     sinetext2 nsamps freq srate
```

Example: generate 1 second sine wave, freq = 440Hz, srate = 44100, redirected to the file *sin440.txt*.

```
$ ./sinetext2  44100 440 44100 >sin440.txt
```

Plot the first 1000 points with Gnuplot:

```
$ gnuplot
...
> plot [t=0:1000] "sin440.txt" with lines
> q
$
```

Optional: use *text2sf* to convert the output to a soundfile (at amplitude 0.5), for viewing with Audacity:

```
$ ./text2sf sin440.txt sin440.aiff 44100 1 0.5
```

Macintosh OS X: use the `-a` option to open Audacity from the terminal, with the requested file:

```
$  open -a audacity.app sin440.aiff
```

Linux (assuming it is installed in a standard place):

```
$  audacity sin440.aiff
```

Windows: double-click audacity.exe and open the soundfile.
(Advanced: place Audacity.exe in a folder that is on your PATH. Audacity can now be launched from the command line as above).

tfork.c: exponential decay applied to a sine wave (a virtual "tuning-fork"). Output to named text file. Uses raw exponential formula with slope argument (time constant).

Usage:

```
     tfork outfile.txt dur freq srate slope
```

Example: generate a 1-second decaying sine wave at 440Hz with a medium slope.

```
$ ./tfork tf440.txt 1 440 44100 0.5
```

tfork2.c: a similar tuning fork generator with amplitude control, and using alternative formula to ensure decay to zero. No slope argument (slope is determined by duration).

Usage:

```
     tfork2 outfile.txt dur freq srate amp
```

Example: generate a 1 second decaying sine wave at 440Hz at amplitude 0.5.

```
$ ./tfork tf440.txt 1 440 44100 0.5
```

tforkraw.c: a tuning fork generator as above, but this one writes raw binary soundfiles (that can be displayed and played using Audacity). Choice of short or float samples.

Usage:

```
     tforkraw outsfile.raw dur freq srate amp isfloat
```

Example: using this program to do the same as in *tfork2*, writing floats to a raw soundfile.

```
$ ./tfork tf440.raw 1 440 44100 0.5 1
```