Guide to DVD Chapter 3 Examples: *Richard Dobson*

From C to C++

This guide only has one significant example, the extension to *tabgen* (from Book Chapter 2) to explore the *Csound* oscillator. See the comments in the source for details. Two small demo programs are provided, demonstrating bitwise operations (to print a number in binary) and the variable argument mechanism. In both cases, see the comments in the code.

Additionally, a simple (and very minimalist) soundfile play program *sfplay* is provided, which demonstrates the use of a *handler* (for CTRL-C), and a *callback function*. This program requires that the *PortAudio* library (v19) be installed on your system. This is a complex package, with potential difficulties especially under Windows (where at least three alternative audio APIs are available, depending on which version of Windows you are using).

In addition, building under Windows will require the DirectX SDK from Microsoft. See the documentation at http://www.portaudio.com for details of installation and building for your platform and compiler.

Building the Examples

The standard **gcc** command-line environment is assumed, as in previous chapters.

Windows:

If you want to build *portaudio* with ASIO support, you will need to register and download the SDK directly from Steinberg:

Go to: http://www.steinberg.net/en/company/3rd_party_developer.html

Unpack and copy the downloaded directory (currently "asiosdk2") into the *portaudio* directory as instructed in the *portaudio* documentation.

For DirectSound support you will also need to install the DirectX SDK.

Go to: http://msdn.microsoft.com/en-us/directx/dd299405.aspx

Look under the "Games" list for the current DirectX SDK. Microsoft tends to change its web pages quite frequently. If the page does not appear (note you will need to use Internet Explorer), use the search facility. Installing the SDK will automatically update any installed Microsoft compilers, such as *Visual Studio Express*.

gcc (OS X, Linux, MinGW)

To build *sfplay*, you need to first install *portaudio*.

portaudio:

Two alternative installation procedures are available; in each case, you need first to download and unpack the *portaudio* distribution into any convenient directory. You will find that double-clicking on the distribution archive (e.g. *pa_stable_v19_20071207.tar.gz*) will extract all files into a new directory without the tar.gz extensions.

NOTE: This is made easier by the standard ability of all Unix shells to perform intelligent *automatic name completion*:

To cd into the newly made directory from its parent, you type the first few characters (but do not hit `<Enter>`):

```
cd pa19
```

and then hit the TAB key – this will auto-complete the command to:

```
cd pa19stableDec07/
```

Procedure 1: Install using: `./configure`

The traditional system for building large programs on Unix/Linux is known as the “autoconf” system, that works out which variations are required in a *makefile* for the platform the software is to be built on, and for the compiler being used. From the user’s point of view, this system follows a standard three-stage sequence:

Stage 1: `./configure`

This runs a substantial script that interrogates the system, and creates the makefile dynamically that will be used in the next stage. Sometimes one or more custom commands can be supplied to `./configure.` For example, in *MinGW* you will need to tell it this fact:

```
./configure --with-host_os=mingw
```

If you want ASIO support (after installing the ASIO SDK) you would add:

```
--with-winapi=asio --with-asiodir=./asiosdk2
```

to that command.

The script then takes a while to inspect all the compiler and platform capabilities, but eventually finishes by creating a Makefile.

Stage 2: `make`

This is a step we are now all quite familiar with.

Stage 3: Install in “the usual place”: this requires administrator privileges:

```
sudo make install
```

123 Shortcut: When no options need to be given to `./configure`, all three commands can be concatenated into one line (note the semicolons):

```
./configure; make; sudo make install
```

You will notice that some of the makefiles use the same format to run multiple commands together.

Procedure 2: scons

This is a new *python*-based build system: download and install from http://www.scons.org. (Linux and OS X will very likely have Python installed already, so that at least is a step that you will probably not need to deal with.)

The program looks for a standard file called ***SConstruct*** in the current directory. It performs the same task as `./configure` and `make` together:

```
scons
```

It is steadily replacing `./configure` as the build system for *portaudio* (much as it already is for *Csound*) – the *autoconf* system is highly complex, and does not meet all contemporary needs.

The Programs

To build (in the examples directory):

```
make
```

This will build *tabgen2*, *bitwise* and *varargdemo*, together with *portsf* if required.

tabgen2 – additive synthesis using table lookup**.**

Usage:

```
tabgen2 [-sN][-b][-c][-wN][-t]outfile dur srate nchans
                              amp freq type noscs

        -s  = set sample type to N ( N= 1 to 5, for 8i,16i,24i,32i,32f)

        -b  =  use block processing

        -c  = use Csound-style table lookup (-t ignored if set)

        -wN = table length (default = 1024)

        -t  = use truncating lookup (else linear interpolation)

         type:  0 = pulse
                1 = square
                2 = triangle
                3 = saw up
                4 = saw down
```

tabgen2 extends *tabgen*

- with the option to use the *Csound* table lookup technique
- with a pulse wave option
- with the option for block-based processing

The outputs are unchanged from *tabgen* (except for the extra pulse wave option). The interest here is in comparing the table lookup methods, most especially in respect of speed. The reader is encouraged to modify the code freely in pursuit of this goal – for example to run multiple oscillators in parallel, and to output multiple channels of audio. Explore the effect (if any) of specifying high sample rates, and long output durations. Some or all of these conditions may need to be combined to detect a significant difference in performance, on modern ultra-fast machines.

Example: Generate a 30-second high-resolution pulse wave.

```
./tabgen2 -s3 -c -b hirespulse.wav 30 96000 2 0.707 220 0 100
```

bitwise – demonstrate bitwise operations, with output in decimal, hex and binary.

Usage:

```
  bitwise op operand_1 operand_2

        Operations:
        0  = AND
        1 = OR
        2 = XOR
        3 = COMP (OP_2 ignored)
        4 = Left Shift
        5 = Right Shift
```

Example: Demonstrate left shift.

```
      ./bitwise 4 16000 2
```

Output:

```
      16000 << 2 = 64000
      0x00003e80 << 0x00000002 = 0x0000fa00
      00000000 00000000 00111110 10000000
      <<
      00000000 00000000 00000000 00000010
      =
      00000000 00000000 11111010 00000000
```

varargdemo – demonstrate use of variable argument list.

Usage:

```
      ./varargdemo
```

See the source code: the program deliberately includes some errors, to demonstrate the dangers inherent in the variable argument mechanism. It is not sufficient merely to run the program and observe the output. Some of that output is incorrect!

sfplay – play a soundfile.

To build: make sure you have *portaudio* built first (see above). `cd` to the *sfplay* directory:

```
      cd sfplay
      make
```

This will (again) build *portsf* first if required.

Usage:

```
      sfplay [-dN] soundfile [from]
       -dN  : use output Device N
```

The program has but two options:

- select the output device (where more than one)
- Start playback at time given in options argument from.

use Ctrl-C to stop playback.

See the code for examples of:

- A handler function for Ctrl-C
- A callback function used by the *portaudio* framework