A fixed-point number can be represented by a 32-bit signed integer, e.g. where n bits are allocated for the integer part and (32 - n) bits for the fractional part. In audio DSP applications, most of the "signals" should be normalized, i.e. range of values is ±1.0. The integer part could be as little as 2 or 3 bits. However, it is wise to leave more "headroom" for computations in which intermediate results may greatly exceed ±1.0.

Using 12 bits for the (signed) integer part and 20 bits for the fractional part gives a magnitude range of ±2047 and a precision of 1 / (2 ^ 20) which is about 0.000001 (decimal). This works well for audio DSP applications, e.g. a "bi-quad" resonant filter, reverberation, etc.

Fixed-point arithmetic is not complicated. Fixed-point addition and subtraction is the same as for ordinary signed integers, e.g.

```
int32_t  h,  j,  k;   // fixed-point variables

h = j + k;
```

To add a constant number, the constant is shifted left 20 bit places to convert to fixed-point, e.g:

```
h = k + (100 << 20);    // add constant 100 to k
```

Scalar multiplication and division are also the same as for ordinary signed integers, e.g.

```
h = k * 10;   // multiply k by 10
j = h / 100;  // divide h by 100
```

To multiply two fixed-point numbers together, it is necessary to use 64-bit multiplication, because the result (in general) will not fit into 32 bits. One of the numbers must be cast into a 64 bit (long long) integer to force the compiler to use a 64 bit multiply function. The result must be divided by (2 ^ 20), i.e. shifted right 20 bits, to obtain the correct result, e.g.

```
h = ((int64) j * k) >> 20;   // h = j x k
```

Programs using fixed-point math may be made more readable by defining a few macros, as follows...

```
#define IntToFixedPt(i)     (i << 20)                  // convert int (i) to fixed-pt
#define FloatToFixed(r)     (int32_t)(r * 1048576)     // convert float (r) to fixed-pt
#define FixedToFloat(z)     ((float)z / 1048576)       // convert fixed-pt (z) to float
#define IntegerPart(z)      (z >> 20)                  // get integer part of fixed-pt
#define FractionPart(z,n)   ((z & 0xFFFFF) >> (20 - n))   // get n MS bits of the
                                                          // fractional part
#define MultiplyFixed(v,w)  (((int64)v * w) >> 20)        // product of two numbers
```

Note: The above constant 1048576 is 2 ^ 20 (i.e. 2 raised to the power 20).