Untitled-1

```
//float
/*
 * floatScale2 - Return bit-level equivalent of expression 2*f for
 *   floating point argument f.
 *   Both the argument and result are passed as unsigned int's, but
 *   they are to be interpreted as the bit-level representation of
 *   single-precision floating point values.
 *   When argument is NaN, return argument
 *   Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while
 *   Max ops: 30
 *   Rating: 4
 */
unsigned floatScale2(unsigned uf) {
  /*
  1 10101010 00000000000000000000000
  1 10101010 10100000000000000000000

  1111111111111111111111111111111
  */

  // The point of this function is bascially to "scale" the input, unsigned integer uf,
by a factor of 2
  // The return value is the bit-level representation of the input scaled by a factor of
2

  // Scaling a float by a factor of 2 is very straightforward
  // If the input is a normalized float, we simply add one to the exponent
  // When the input is denormalized, we can scale it by shifting the mantissa one bit to
the left (effectivly multiplying it by 2)

  // structure of single precision floating point
  // Sign | Exponent | Mantissa
  // 1    | 8        | 23
  // Bias = 127

  // extract the sign and exp from uf
  unsigned sign = (uf >> 31);
  unsigned exp = (~(0x1 << 31) & uf) >> 23;

  // 0xF = 1111
  // since the mantissa is 23 bits long, we need 5 0xF and a 0x7 (0111)
  // Mantissa mask => 0x7FFFFF;
  unsigned frac = (uf & 0x7FFFFF);

  // if uf is NaN or inf, then return uf
  if(exp == 0xFF) return uf;

  if(exp == 0) {
    frac <<= 1;
  } else {
    exp += 1;
  }

  return (sign << 31) | (exp << 23) | frac;
```

```
    // return 2;
  }
```