```
/*
 * CS:APP Data Lab
 *
 * <Please put your name and userid here>
 *
 * bits.c - Source file with your solutions to the Lab.
 *          This is the file you will hand in to your instructor.
 *
 * WARNING: Do not include the <stdio.h> header; it confuses the dlc
 * compiler. You can still use printf for debugging without including
 * <stdio.h>, although you might get a compiler warning. In general,
 * it's not good practice to ignore compiler warnings, but in this
 * case it's OK.
 */


#if 0
/*
 * Instructions to Students:
 *
 * STEP 1: Read the following instructions carefully.
 */

You will provide your solution to the Data Lab by
editing the collection of functions in this source file.

INTEGER CODING RULES:

  Replace the "return" statement in each function with one
  or more lines of C code that implements the function. Your code
  must conform to the following style:

  int Funct(arg1, arg2, ...) {
      /* brief description of how your implementation works */
      int var1 = Expr1;
      ...
      int varM = ExprM;

      varJ = ExprJ;
      ...
      varN = ExprN;
      return ExprR;
  }

  Each "Expr" is an expression using ONLY the following:
  1. Integer constants 0 through 255 (0xFF), inclusive. You are
     not allowed to use big constants such as 0xffffffff.
  2. Function arguments and local variables (no global variables).
  3. Unary integer operations ! ~
  4. Binary integer operations & ^ | + << >>

  Some of the problems restrict the set of allowed operators even further.
  Each "Expr" may consist of multiple operators. You are not restricted to
  one operator per line.

  You are expressly forbidden to:
  1. Use any control constructs such as if, do, while, for, switch, etc.
  2. Define or use any macros.
  3. Define any additional functions in this file.
  4. Call any functions.
  5. Use any other operations, such as &&, ||, -, or ?:
  6. Use any form of casting.
  7. Use any data type other than int.  This implies that you
     cannot use arrays, structs, or unions.
```

```
   You may assume that your machine:
   1. Uses 2s complement, 32-bit representations of integers.
   2. Performs right shifts arithmetically.
   3. Has unpredictable behavior when shifting an integer by more
      than the word size.

 EXAMPLES OF ACCEPTABLE CODING STYLE:
   /*
    * pow2plus1 - returns 2^x + 1, where 0 <= x <= 31
    */
   int pow2plus1(int x) {
      /* exploit ability of shifts to compute powers of 2 */
      return (1 << x) + 1;
   }

   /*
    * pow2plus4 - returns 2^x + 4, where 0 <= x <= 31
    */
   int pow2plus4(int x) {
      /* exploit ability of shifts to compute powers of 2 */
      int result = (1 << x);
      result += 4;
      return result;
   }


 FLOATING POINT CODING RULES

 For the problems that require you to implent floating-point operations,
 the coding rules are less strict.  You are allowed to use looping and
 conditional control.  You are allowed to use both ints and unsigneds.
 You can use arbitrary integer and unsigned constants.

 You are expressly forbidden to:
   1. Define or use any macros.
   2. Define any additional functions in this file.
   3. Call any functions.
   4. Use any form of casting.
   5. Use any data type other than int or unsigned.  This means that you
      cannot use arrays, structs, or unions.
   6. Use any floating point data types, operations, or constants.


 NOTES:
   1. Use the dlc (data lab checker) compiler (described in the handout) to
      check the legality of your solutions.
   2. Each function has a maximum number of operators (! ~ & ^ | + << >>)
      that you are allowed to use for your implementation of the function.
      The max operator count is checked by dlc. Note that '=' is not
      counted; you may use as many of these as you want without penalty.
   3. Use the btest test harness to check your functions for correctness.
   4. Use the BDD checker to formally verify your functions
   5. The maximum number of ops for each function is given in the
      header comment for each function. If there are any inconsistencies
      between the maximum ops in the writeup and in this file, consider
      this file the authoritative source.

 /*
  * STEP 2: Modify the following functions according the coding rules.
  *
  *   IMPORTANT. TO AVOID GRADING SURPRISES:
  *   1. Use the dlc compiler to check that your solutions conform
  *      to the coding rules.
  *   2. Use the BDD checker to formally verify that your solutions produce
  *      the correct answers.
```

```
   */


 #endif
/* Copyright (C) 1991-2012 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   The GNU C Library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with the GNU C Library; if not, see
   <http://www.gnu.org/licenses/>.  */
/* This header is separate from features.h so that the compiler can
   include it implicitly at the start of every compilation.  It must
   not itself include <features.h> or any other header that includes
   <features.h> because the implicit include comes before any feature
   test macros that may be defined in a source file before it first
   explicitly includes a system header.  GCC knows the name of this
   header in order to preinclude it.  */
/* We do support the IEC 559 math functionality, real and complex.  */
/* wchar_t uses ISO/IEC 10646 (2nd ed., published 2011-03-15) /
   Unicode 6.0.  */
/* We do not support C11 <threads.h>.  */
/*
 * ezThreeFourths - multiplies by 3/4 rounding toward 0,
 *   Should exactly duplicate effect of C expression (x*3/4),
 *   including overflow behavior.
 *   Examples: ezThreeFourths(11) = 8
 *             ezThreeFourths(-9) = -6
 *             ezThreeFourths(1073741824) = -268435456 (overflow)
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 12
 *   Rating: 3
 */
int ezThreeFourths(int x) {
  int threex = ((x << 1) + x);
  int bias = (threex >> 31) & 3;
  return (threex + bias) >> 2;
}
```