I'm working on modernizing Rosetta Code's infrastructure. Starting with communications. Please accept this time-limited open invite to RC's Slack. (https://join.slack.com/t/rosettacode/shared_invite/zt-glwmugtu-xpMPcqHs0u6MsK5zCmJF~Q). --Michael Mol (/wiki/User:Short_Circuit) (talk (/wiki/User_talk:Short_Circuit)) 20:59, 30 May 2020 (UTC)

Least common multiple

Task

Compute the least common multiple (LCM) of two integers.

Given *m* and *n*, the least common multiple is the smallest positive integer that has both *m* and *n* as factors.

Example

The least common multiple of **12** and **18** is **36**, because:

- **12** is a factor (**12** × **3** = **36**), and
- **18** is a factor (**18** × **2** = **36**), and
- there is no positive integer less than **36** that has both factors.

As a special case, if either *m* or *n* is zero, then the least common multiple is zero.

One way to calculate the least common multiple is to iterate all the multiples of *m*, until you find one that is also a multiple of *n*.

If you already have *gcd* for greatest common divisor (/wiki/Greatest_common_divisor), then this formula calculates *lcm*.

(/wiki/Category:Solutions_by_Programming

Least common multiple
You are encouraged to solve this task (/wiki/Rosetta_Code:Solve_a_Task) according to the task description, using any language you may know.

$$\operatorname{lcm}(m, n) = \frac{|m \times n|}{\gcd(m, n)}$$

One can also find *lcm* by merging the prime decompositions (/wiki/Prime_decomposition) of both *m* and *n*.

Related task

- **greatest common divisor (https://rosettacode.org/wiki/Greatest_common_divisor).**

See also

- MathWorld entry: Least Common Multiple (http://mathworld.wolfram.com/LeastCommonMultiple.html).
- Wikipedia entry: Least common multiple (https://en.wikipedia.org/wiki/Least_common_multiple).

Contents

11l (/wiki/Category:11l)

```
F gcd(=a, =b)
   L b != 0
      (a, b) = (b, a % b)
   R a

F lcm(m, n)
   R m I/ gcd(m, n) * n

print(lcm(12, 18))
```

Output:

```
36
```

360 Assembly (/wiki/Category:360_Assembly)

Translation of: PASCAL
For maximum compatibility, this program uses only the basic instruction set (S/360) with 2 ASSIST macros (XDECO,XPRNT).

```
LCM      CSECT
         USING  LCM,R15           use calling register
         L      R6,A              a
         L      R7,B              b
         LR     R8,R6             c=a
LOOPW    LR     R4,R8               c
         SRDA   R4,32               shift to next reg
         DR     R4,R7               c/b
         LTR    R4,R4             while c mod b<>0
         BZ     ELOOPW              leave while
         AR     R8,R6               c+=a
         B      LOOPW             end while
ELOOPW   LPR    R9,R6             c=abs(u)
         L      R1,A              a
         XDECO  R1,XDEC           edit a
         MVC    PG+4(5),XDEC+7    move a to buffer
         L      R1,B              b
         XDECO  R1,XDEC           edit b
         MVC    PG+10(5),XDEC+7   move b to buffer
         XDECO  R8,XDEC           edit c
         MVC    PG+17(10),XDEC+2  move c to buffer
         XPRNT  PG,80             print buffer
         XR     R15,R15           return code =0
         BR     R14               return to caller
A        DC     F'1764'           a
B        DC     F'3920'           b
PG       DC     CL80'lcm(00000,00000)=0000000000'  buffer
XDEC     DS     CL12              temp for edit
         YREGS
         END    LCM
```

Output:

```
lcm( 1764, 3920)=     35280
```

8th (/wiki/Category:8th)

```
: gcd \ a b -- gcd
        dup 0 n:= if drop ;; then
        tuck \ b a b
        n:mod \ b a-mod-b
        recurse ;

: lcm \ m n
        2dup \ m n m n
        n:* \ m n m*n
        n:abs \ m n abs(m*n)
        -rot \ abs(m*n) m n
        gcd \ abs(m*n) gcd(m.n)
        n:/mod \ abs / gcd
        nip \ abs div gcd
;

: demo \ n m --
        2dup "LCM of " . . " and " . . " = " . lcm . ;

12 18 demo cr
-6 14 demo cr
35  0 demo cr


bye
```

Output:

```
LCM of 18 and 12 = 36
LCM of 14 and -6 = 42
LCM of 0 and 35 = 0
```

Ada (/wiki/Category:Ada)

lcm_test.adb:

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Lcm_Test is
   function Gcd (A, B : Integer) return Integer is
      M : Integer := A;
      N : Integer := B;
      T : Integer;
   begin
      while N /= 0 loop
         T := M;
         M := N;
         N := T mod N;
      end loop;
      return M;
   end Gcd;

   function Lcm (A, B : Integer) return Integer is
   begin
      if A = 0 or B = 0 then
         return 0;
      end if;
      return abs (A) * (abs (B) / Gcd (A, B));
   end Lcm;
begin
   Put_Line ("LCM of 12, 18 is" & Integer'Image (Lcm (12, 18)));
   Put_Line ("LCM of -6, 14 is" & Integer'Image (Lcm (-6, 14)));
   Put_Line ("LCM of 35, 0 is" & Integer'Image (Lcm (35, 0)));
end Lcm_Test;
```

Output:

```
LCM of 12, 18 is 36
LCM of -6, 14 is 42
LCM of 35, 0 is 0
```

ALGOL 68 (/wiki/Category:ALGOL_68)

```
BEGIN
   PROC gcd = (INT m, n) INT :
   BEGIN
      INT a := ABS m, b := ABS n;
      IF a=0 OR b=0 THEN 0 ELSE
         WHILE b /= 0 DO INT t = b; b := a MOD b; a := t OD;
         a
      FI
   END;
   PROC lcm = (INT m, n) INT : ( m*n = 0 | 0 | ABS (m*n) % gcd (m, n));
   INT m=12, n=18;
   printf ((($gxg(0)3(xgxg(0))l$,
            "The least common multiple of", m, "and", n, "is", lcm(m,n),
            "and their greatest common divisor is", gcd(m,n)))
END
```

Output:

```
The least common multiple of 12 and 18 is 36 and their greatest common divisor is 6
```

Note that either or both PROCs could just as easily be implemented as OPs but then the operator priorities would also have to be declared.

ALGOL W (/wiki/Category:ALGOL_W)

```
begin
    integer procedure gcd ( integer value a, b ) ;
        if b = 0 then a else gcd( b, a rem abs(b) );

    integer procedure lcm( integer value a, b ) ;
        abs( a * b ) div gcd( a, b );

    write( lcm( 15, 20  ) );
end.
```

APL (/wiki/Category:APL)

APL provides this function.

```
      12^18
36
```

If for any reason we wanted to reimplement it, we could do so in terms of the greatest common divisor by transcribing the formula set out in the task specification into APL notation:

```
      LCM←{(|α×ω)÷α∨ω}
      12 LCM 18
36
```

AppleScript (/wiki/Category:AppleScript)

```
------------------ LEAST COMMON MULTIPLE -----------------

-- lcm :: Integral a => a -> a -> a
on lcm(x, y)
    if 0 = x or 0 = y then
        0
    else
        abs(x div (gcd(x, y)) * y)
    end if
end lcm


--------------------------- TEST -------------------------
on run

    lcm(12, 18)

    --> 36
end run


-------------------- GENERIC FUNCTIONS -------------------

-- abs :: Num a => a -> a
on abs(x)
    if 0 > x then
        -x
    else
        x
    end if
end abs


-- gcd :: Integral a => a -> a -> a
on gcd(x, y)
    script
        on |λ|(a, b)
            if 0 = b then
                a
            else
                |λ|(b, a mod b)
            end if
        end |λ|
    end script

    result's |λ|(abs(x), abs(y))
end gcd
```

Output:

```
36
```

Arendelle (/wiki/Category:Arendelle)

For GCD function check out here (https://rosettacode.org/wiki/Greatest_common_divisor#Arendelle)

```
< a , b >

( return ,

        abs ( @a * @b ) /
        !gcd( @a , @b )

)
```

Arturo (/wiki/Category:Arturo)

```
lcm: function [x,y][
    x * y / gcd @[x y]
]

print lcm 12 18
```

Output:

```
36
```

Assembly (/wiki/Category:Assembly)

x86 Assembly (/wiki/Category:X86_Assembly)

```
; lcm.asm: calculates the least common multiple
; of two positive integers
;
; nasm x86_64 assembly (linux) with libc
; assemble: nasm -felf64 lcm.asm; gcc lcm.o
; usage: ./a.out [number1] [number2]

    global main
    extern printf ; c function: prints formatted output
    extern strtol ; c function: converts strings to longs

    section .text

main:
    push rbp    ; set up stack frame

    ; rdi contains argc
    ; if less than 3, exit
    cmp rdi, 3
    jl incorrect_usage

    ; push first argument as number
    push rsi
    mov rdi, [rsi+8]
    mov rsi, 0
    mov rdx, 10 ; base 10
    call strtol
    pop rsi
    push rax

    ; push second argument as number
    push rsi
    mov rdi, [rsi+16]
    mov rsi, 0
    mov rdx, 10 ; base 10
    call strtol
    pop rsi
    push rax

    ; pop arguments and call get_gcd
    pop rdi
    pop rsi
    call get_gcd

    ; print value
    mov rdi, print_number
    mov rsi, rax
    call printf

    ; exit
    mov rax, 0  ; 0--exit success
    pop rbp
    ret
```

```
incorrect_usage:
    mov rdi, bad_use_string
    ; rsi already contains argv
    mov rsi, [rsi]
    call printf
    mov rax, 0  ; 0--exit success
    pop rbp
    ret

bad_use_string:
    db "Usage: %s [number1] [number2]",10,0

print_number:
    db "%d",10,0

get_gcd:
    push rbp    ; set up stack frame
    mov rax, 0
    jmp loop

loop:
    ; keep adding the first argument
    ; to itself until a multiple
    ; is found. then, return
    add rax, rdi
    push rax
    mov rdx, 0
    div rsi
    cmp rdx, 0
    pop rax
    je gcd_found
    jmp loop

gcd_found:
    pop rbp
    ret
```

AutoHotkey (/wiki/Category:AutoHotkey)

```
LCM(Number1,Number2)
{
 If (Number1 = 0 || Number2 = 0)
  Return
 Var := Number1 * Number2
 While, Number2
  Num := Number2, Number2 := Mod (http://www.autohotkey.com/docs/Functions.htm#BuiltIr
 Return, Var // Number1
}

Num1 = 12
Num2 = 18
MsgBox (http://www.autohotkey.com/docs/commands/MsgBox.htm) % LCM(Num1,Num2)
```

AutoIt (/wiki/Category:AutoIt)

```
Func (https://www.autoitscript.com/autoit3/docs/keywords.htm) _LCM($a, $b)
        Local (https://www.autoitscript.com/autoit3/docs/keywords.htm) $c, $f, $m = $a
        $c = 1
        While (https://www.autoitscript.com/autoit3/docs/keywords.htm) $c <> 0
                $f = Int (https://www.autoitscript.com/autoit3/docs/functions/Int.htm)
                $c = $a - $b * $f
                If (https://www.autoitscript.com/autoit3/docs/keywords.htm) $c <> 0 Th
                        $a = $b
                        $b = $c
                EndIf (https://www.autoitscript.com/autoit3/docs/keywords.htm)
        WEnd (https://www.autoitscript.com/autoit3/docs/keywords.htm)
        Return (https://www.autoitscript.com/autoit3/docs/keywords.htm) $m * $n / $b
EndFunc (https://www.autoitscript.com/autoit3/docs/keywords.htm)   ;==>_LCM
```

Example

```
ConsoleWrite (https://www.autoitscript.com/autoit3/docs/functions/ConsoleWrite.htm)(_L
ConsoleWrite (https://www.autoitscript.com/autoit3/docs/functions/ConsoleWrite.htm)(_L
ConsoleWrite (https://www.autoitscript.com/autoit3/docs/functions/ConsoleWrite.htm)(_L
```

```
36
60
0
```

--BugFix (/mw/index.php?title=User:BugFix&action=edit&redlink=1) (talk (/mw/index.php?title=User_talk:BugFix&action=edit&redlink=1)) 14:32, 15 November 2013 (UTC)

AWK (/wiki/Category:AWK)

```
# greatest common divisor
function gcd(m, n,    t) {
        # Euclid's method
        while (n != 0) {
                t = m
                m = n
                n = t % n
        }
        return m
}

# least common multiple
function lcm(m, n,    r) {
        if (m == 0 || n == 0)
                return 0
        r = m * n / gcd(m, n)
        return r < 0 ? -r : r
}

# Read two integers from each line of input.
# Print their least common multiple.
{ print lcm($1, $2) }
```

Example input and output:

```
$ awk -f lcd.awk
12 18
36
-6 14
42
35 0
0
```

BASIC (/wiki/Category:BASIC)

Applesoft BASIC (/wiki/Category:Applesoft_BASIC)

ported from BBC BASIC

```
10 DEF FN MOD(A) = INT((A / B - INT(A / B)) * B + .05) * SGN(A / B)
20 INPUT"M=";M%
30 INPUT"N=";N%
40 GOSUB 100
50 PRINT R
60 END

100 REM LEAST COMMON MULTIPLE M% N%
110 R = 0
120 IF M% = 0 OR N% = 0 THEN RETURN
130 A% = M% : B% = N% : GOSUB 200"GCD
140 R = ABS(M%*N%)/R
150 RETURN

200 REM GCD ITERATIVE EUCLID A% B%
210 FOR B = B% TO 0 STEP 0
220     C% = A%
230     A% = B
240     B = FN MOD(C%)
250 NEXT B
260 R = ABS(A%)
270 RETURN
```

BBC BASIC (/wiki/Category:BBC_BASIC)

Works with: BBC BASIC for Windows (/wiki/BBC_BASIC_for_Windows)

```
      DEF FN_LCM(M%,N%)
      IF M%=0 OR N%=0 THEN =0 ELSE =ABS(M%*N%)/FN_GCD_Iterative_Euclid(M%, N%)

      DEF FN_GCD_Iterative_Euclid(A%, B%)
      LOCAL C%
      WHILE B%
        C% = A%
        A% = B%
        B% = C% MOD B%
      ENDWHILE
      = ABS(A%)
```

IS-BASIC (/wiki/Category:IS-BASIC)

```
100 DEF LCM(A,B)=(A*B)/GCD(A,B)
110 DEF GCD(A,B)
120   DO WHILE B>0
130     LET T=B:LET B=MOD(A,B):LET A=T
140   LOOP
150   LET GCD=A
160 END DEF
170 PRINT LCM(12,18)
```

Tiny BASIC (/wiki/Category:Tiny_BASIC)

```
10 PRINT "First number"
20 INPUT A
30 PRINT "Second number"
40 INPUT B
42 LET Q = A
44 LET R = B
50 IF Q<0 THEN LET Q=-Q
60 IF R<0 THEN LET R=-R
70 IF Q>R THEN GOTO 130
80 LET R = R - Q
90 IF Q=0 THEN GOTO 110
100 GOTO 50
110 LET U = (A*B)/R
111 IF U < 0 THEN LET U = - U
112 PRINT U
120 END
130 LET C=Q
140 LET Q=R
150 LET R=C
160 GOTO 70
```

Batch File (/wiki/Category:Batch_File)

```
@echo (https://www.ss64.com/nt/echo.html) off
setlocal (https://www.ss64.com/nt/setlocal.html) enabledelayedexpansion
set (https://www.ss64.com/nt/set.html) num1=12
set (https://www.ss64.com/nt/set.html) num2=18

call (https://www.ss64.com/nt/call.html) :lcm %num1% %num2%
exit (https://www.ss64.com/nt/exit.html) /b

:lcm <input1> <input2>
if (https://www.ss64.com/nt/if.html) %2 equ (https://www.ss64.com/nt/equ.html) 0 (
        set (https://www.ss64.com/nt/set.html) /a lcm = %num1%*%num2%/%1
        echo (https://www.ss64.com/nt/echo.html) LCM = !lcm!
        pause>nul (https://www.ss64.com/nt/nul.html)
        goto (https://www.ss64.com/nt/goto.html) :EOF
)
set (https://www.ss64.com/nt/set.html) /a res = %1 %% %2
call (https://www.ss64.com/nt/call.html) :lcm %2 %res%
goto (https://www.ss64.com/nt/goto.html) :EOF
```

Output:

```
LCM = 36
```

bc (/wiki/Category:Bc)

Translation of: AWK

```
/* greatest common divisor */
define g(m, n) {
        auto t

        /* Euclid's method */
        while (n != 0) {
                t = m
                m = n
                n = t % n
        }
        return (m)
}

/* least common multiple */
define l(m, n) {
        auto r

        if (m == 0 || n == 0) return (0)
        r = m * n / g(m, n)
        if (r < 0) return (-r)
        return (r)
}
```

Befunge (/wiki/Category:Befunge)

Inputs are limited to signed 16-bit integers.

```
&>:0`2*1-*:&>:#@!#._:0`2*1v
>28*:*:**+:28*>:*:*/\:vv*-<
|<:%/*:*:*82\%*:*:*82<<>28v
>$/28*:*:*/*.@^82::+**:*:*<
```

Input:

```
12345
-23044
```

Output:

```
345660
```

BQN (/wiki/Category:BQN)

```
Lcm ← ×÷{𝕨(|𝕊⍟(>⊸0)⊣)𝕩}
```

Example:

```
12 Lcm 18
```

```
36
```

Bracmat (/wiki/Category:Bracmat)

We utilize the fact that Bracmat simplifies fractions (using Euclid's algorithm). The function `den$`*`number`* returns the denominator of a number.

```
(gcd=
  a b
.   !arg:(?a.?b)
  &   den$(!a*!b^-1)
    * (!a:<0&-1|1)
    * !a
);
out$(gcd$(12.18) gcd$(-6.14) gcd$(35.0) gcd$(117.18))
```

Output:

```
36 42 35 234
```

Brat (/wiki/Category:Brat)

```
gcd = { a, b |
  true? { a == 0 }
    { b }
    { gcd(b % a, a) }
}

lcm = { a, b |
  a * b / gcd(a, b)
}

p lcm(12, 18) # 36
p lcm(14, 21) # 42
```

C (/wiki/Category:C)

```c
#include <stdio.h>

int gcd(int m, int n)
{
        int tmp;
        while(m) { tmp = m; m = n % m; n = tmp; }
        return n;
}

int lcm(int m, int n)
{
        return m / gcd(m, n) * n;
}

int main()
{
        printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)(
        return 0;
}
```

C# (/wiki/Category:C_sharp)

```c#
Using System;
class Program
{
    static int gcd(int m, int n)
    {
        return n == 0 ? Math.Abs(m) : gcd(n, n % m);
    }
    static int lcm(int m, int n)
    {
        return Math.Abs(m * n) / gcd(m, n);
    }
    static void Main()
    {
        Console.WriteLine("lcm(12,18)=" + lcm(12,18));
    }
}
```

Output:

```
lcm(12,18)=36
```

C++ (/wiki/Category:C%2B%2B)

Library: Boost (/wiki/Category:Boost)

```
#include <boost/math/common_factor.hpp>
#include <iostream>

int main( ) {
   std::cout << "The least common multiple of 12 and 18 is " <<
      boost::math::lcm( 12 , 18 ) << " ,\n"
      << "and the greatest common divisor " << boost::math::gcd( 12 , 18 ) << " !" <<
   return 0 ;
}
```

Output:

```
The least common multiple of 12 and 18 is 36 ,
and the greatest common divisor 6 !
```

Alternate solution

Works with: C++11 (/wiki/C%2B%2B11)

```
#include <cstdlib>
#include <iostream>
#include <tuple>

int gcd(int a, int b) {
    a = abs(a);
    b = abs(b);
    while (b != 0) {
        std::tie(a, b) = std::make_tuple(b, a % b);
    }
    return a;
}

int lcm(int a, int b) {
    int c = gcd(a, b);
    return c == 0 ? 0 : a / c * b;
}

int main() {
    std::cout << "The least common multiple of 12 and 18 is " << lcm(12, 18) << ",\n"
        << "and their greatest common divisor is " << gcd(12, 18) << "!"
        << std::endl;
    return 0;
}
```

Clojure (/wiki/Category:Clojure)

```
(defn gcd
      [a b]
      (if (zero? b)
      a
      (recur b, (mod a b))))

(defn lcm
      [a b]
      (/ (* a b) (gcd a b)))
;; to calculate the lcm for a variable number of arguments
(defn lcmv [& v] (reduce lcm v))
```

COBOL (/wiki/Category:COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. show-lcm.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    FUNCTION lcm
    .
PROCEDURE DIVISION.
    DISPLAY "lcm(35, 21) = " FUNCTION lcm(35, 21)
    GOBACK
    .
END PROGRAM show-lcm.

IDENTIFICATION DIVISION.
FUNCTION-ID. lcm.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    FUNCTION gcd
    .
DATA DIVISION.
LINKAGE SECTION.
01  m                      PIC S9(8).
01  n                      PIC S9(8).
01  ret                    PIC S9(8).

PROCEDURE DIVISION USING VALUE m, n RETURNING ret.
    COMPUTE ret = FUNCTION ABS(m * n) / FUNCTION gcd(m, n)
    GOBACK
    .
END FUNCTION lcm.

IDENTIFICATION DIVISION.
FUNCTION-ID. gcd.

DATA DIVISION.
LOCAL-STORAGE SECTION.
01  temp                   PIC S9(8).

01  x                      PIC S9(8).
01  y                      PIC S9(8).

LINKAGE SECTION.
01  m                      PIC S9(8).
01  n                      PIC S9(8).
01  ret                    PIC S9(8).

PROCEDURE DIVISION USING VALUE m, n RETURNING ret.
    MOVE m to x
    MOVE n to y

    PERFORM UNTIL y = 0
```

```
            MOVE x TO temp
            MOVE y TO x
            MOVE FUNCTION MOD(temp, y) TO Y
        END-PERFORM

        MOVE FUNCTION ABS(x) TO ret
        GOBACK
        .
    END FUNCTION gcd.
```

Common Lisp (/wiki/Category:Common_Lisp)

Common Lisp provides the `lcm` function. It can accept two or more (or less) parameters.

```
CL-USER> (lcm 12 18)
36
CL-USER> (lcm 12 18 22)
396
```

Here is one way to reimplement it.

```
CL-USER> (defun my-lcm (&rest args)
           (reduce (lambda (m n)
                     (cond ((or (= m 0) (= n 0)) 0)
                           (t (abs (/ (* m n) (gcd m n))))))
                   args :initial-value 1))
MY-LCM
CL-USER> (my-lcm 12 18)
36
CL-USER> (my-lcm 12 18 22)
396
```

In this code, the `lambda` finds the least common multiple of two integers, and the `reduce` transforms it to accept any number of parameters. The `reduce` operation exploits how *lcm* is associative, `(lcm a b c) == (lcm (lcm a b) c)`; and how 1 is an identity, `(lcm 1 a) == a`.

D (/wiki/Category:D)

```
import std.stdio, std.bigint, std.math;

T gcd(T)(T a, T b) pure nothrow {
    while (b) {
        immutable t = b;
        b = a % b;
        a = t;
    }
    return a;
}

T lcm(T)(T m, T n) pure nothrow {
    if (m == 0) return m;
    if (n == 0) return n;
    return abs((m * n) / gcd(m, n));
}

void main() {
    lcm(12, 18).writeln;
    lcm("256204778801521550085490633230958956l".BigInt,
        "679545449426828292043156566l684282819".BigInt).writeln;
}
```

Output:

```
36
1566925124003829826223212517517200259473120608l193527869
```

Dart (/wiki/Category:Dart)

```
main() {
        int x=8;
  int y=12;
int z= gcd(x,y);
  var lcm=(x*y)/z;
  print('$lcm');
  }

int gcd(int a,int b)
{
  if(b==0)
    return a;
  if(b!=0)
    return gcd(b,a%b);
}
```

Delphi (/wiki/Category:Delphi)

See Pascal (https://rosettacode.org/wiki/Least_common_multiple#Pascal).

DWScript (/wiki/Category:DWScript)

```
PrintLn(Lcm(12, 18));
```

Output:

```
36
```

EchoLisp (/wiki/Category:EchoLisp)

(lcm a b) is already here as a two arguments function. Use foldl to find the lcm of a list of numbers.

```
(lcm 0 9) → 0
(lcm 444 888)→ 888
(lcm 888 999) → 7992

(define (lcm* list) (foldl lcm (first list) list)) → lcm*
(lcm* '(444 888 999)) → 7992
```

Elena (/wiki/Category:Elena)

Translation of: C#
ELENA 4.x :

```
import extensions;
import system'math;

gcd = (m,n => (n == 0) ? (m.Absolute) : (gcd(n,n.mod:m)));

lcm = (m,n => (m * n).Absolute / gcd(m,n));

public program()
{
    console.printLine("lcm(12,18)=",lcm(12,18))
}
```

Output:

```
lcm(12,18)=36
```

Elixir (/wiki/Category:Elixir)

```
defmodule RC do
  def gcd(a,0), do: abs(a)
  def gcd(a,b), do: gcd(b, rem(a,b))

  def lcm(a,b), do: div(abs(a*b), gcd(a,b))
end

IO.puts RC.lcm(-12,15)
```

Output:

```
60
```

Erlang (/wiki/Category:Erlang)

```
% Implemented by Arjun Sunel
-module(lcm).
-export([main/0]).

main() ->
        lcm(-3,4).

gcd(A, 0) ->
        A;

gcd(A, B) ->
        gcd(B, A rem B).

lcm(A,B) ->
        abs(A*B div gcd(A,B)).
```

Output:

```
12
```

ERRE (/wiki/Category:ERRE)

```
PROGRAM LCM

PROCEDURE GCD(A,B->GCD)
    LOCAL C
    WHILE B DO
        C=A
        A=B
        B=C MOD B
    END WHILE
    GCD=ABS(A)
END PROCEDURE

PROCEDURE LCM(M,N->LCM)
    IF M=0 OR N=0 THEN
        LCM=0
        EXIT PROCEDURE
      ELSE
        GCD(M,N->GCD)
        LCM=ABS(M*N)/GCD
    END IF
END PROCEDURE

BEGIN
    LCM(18,12->LCM)
    PRINT("LCM of 18 AND 12 =";LCM)
    LCM(14,-6->LCM)
    PRINT("LCM of 14 AND -6 =";LCM)
    LCM(0,35->LCM)
    PRINT("LCM of 0 AND 35 =";LCM)
END PROGRAM
```

Output:

```
LCM of 18 and 12 = 36
LCM of 14 and -6 = 42
LCM of 0 and 35 = 0
```

Euphoria (/wiki/Category:Euphoria)

```
function gcd(integer m, integer n)
    integer tmp
    while m do
        tmp = m
        m = remainder(n,m)
        n = tmp
    end while
    return n
end function

function lcm(integer m, integer n)
    return m / gcd(m, n) * n
end function
```

Excel (/wiki/Category:Excel)

Excel's LCM can handle multiple values. Type in a cell:

```
=LCM(A1:J1)
```

This will get the LCM on the first 10 cells in the first row. Thus :

```
12      3       5       23      13      67      15      9       4       2

3605940
```

Ezhil (/wiki/Category:Ezhil)

```
## இந்த நிரல் இரு எண்களுக்கு இடையிலான மீச்சிறு பொது மடங்கு (LCM), மீப்பெரு பொது வகுத்தி (GCD) என்ன

நிரல்பாகம் மீபொம(எண்1, எண்2)

        @(எண்1 == எண்2) ஆனால்

  ## இரு எண்களும் சமம் என்பதால், மீபொம அந்த எண்ணேதான்

                பின்கொடு எண்1

        @(எண்1 > எண்2) இல்லைஆனால்

                சிறியது = எண்2
                பெரியது = எண்1

        இல்லை

                சிறியது = எண்1
                பெரியது = எண்2

        முடி

        மீதம் = பெரியது % சிறியது

        @(மீதம் == 0) ஆனால்

  ## பெரிய எண்ணில் சிறிய எண் மீதமின்றி வகுபடுவதால், பெரிய எண்தான் மீபொம

                பின்கொடு பெரியது

        இல்லை

                தொடக்கம் = பெரியது + 1
                நிறைவு = சிறியது * பெரியது

                @(எண் = தொடக்கம், எண் <= நிறைவு, எண் = எண் + 1) ஆக

    ## ஒவ்வோர் எண்ணாக எடுத்துக்கொண்டு தரப்பட்ட இரு எண்களாலும் வகுத்துப் பார்க்கின்றோம். முதலாவதாக இ

                        மீதம்1 = எண் % சிறியது
                        மீதம்2 = எண் % பெரியது

                        @((மீதம்1 == 0) && (மீதம்2 == 0)) ஆனால்
                                பின்கொடு எண்
                        முடி

                முடி

        முடி

முடி

அ = int(உள்ளீடு("ஓர் எண்ணைத் தாருங்கள் "))
```

```
ஆ = int(உள்ளீடு("இன்னோர் எண்ணைத் தாருங்கள் "))

பதிப்பி "நீங்கள் தந்த இரு எண்களின் மீபொம (மீச்சிறு பொது மடங்கு, LCM) = ", மீபொம(அ, ஆ)
```

F# (/wiki/Category:F_Sharp)

```
let rec gcd x y = if y = 0 then abs x else gcd y (x % y)

let lcm x y = x * y / (gcd x y)
```

Factor (/wiki/Category:Factor)

The vocabulary *math.functions* already provides *lcm*.

```
USING: math.functions prettyprint ;
26 28 lcm .
```

This program outputs *364*.

One can also reimplement *lcm*.

```
USING: kernel math prettyprint ;
IN: script

: gcd ( a b -- c )
    [ abs ] [
        [ nip ] [ mod ] 2bi gcd
    ] if-zero ;

: lcm ( a b -- c )
    [ * abs ] [ gcd ] 2bi / ;

26 28 lcm .
```

Forth (/wiki/Category:Forth)

```
: gcd ( a b -- n )
  begin dup while tuck mod repeat drop ;

: lcm ( a b -- n )
  over 0= over 0= or if 2drop 0 exit then
  2dup gcd abs */ ;
```

Fortran (/wiki/Category:Fortran)

This solution is written as a combination of 2 functions, but a subroutine implementation would work great as well.

```
    integer function lcm(a,b)
    integer:: a,b
        lcm = a*b / gcd(a,b)
    end function lcm

    integer function gcd(a,b)
    integer :: a,b,t
        do while (b/=0)
            t = b
            b = mod(a,b)
            a = t
        end do
        gcd = abs(a)
    end function gcd
```

FreeBASIC (/wiki/Category:FreeBASIC)

Iterative solution

```
' FB 1.05.0 Win64

Function lcm (m As Integer, n As Integer) As Integer
  If m = 0 OrElse n = 0 Then Return 0
  If m < n Then Swap m, n '' to minimize iterations needed
  Var count = 0
  Do
    count +=1
  Loop Until (m * count) Mod n  = 0
  Return m * count
End Function

Print "lcm(12, 18) ="; lcm(12, 18)
Print "lcm(15, 12) ="; lcm(15, 12)
Print "lcm(10, 14) ="; lcm(10, 14)
Print
Print "Press any key to quit"
Sleep
```

Output:

```
lcm(12, 18) = 36
lcm(15, 12) = 60
lcm(10, 14) = 70
```

Recursive solution

Reuses code from Greatest_common_divisor#Recursive_solution (/wiki/Greatest_common_divisor#Recursive_solution) and correctly handles negative arguments

```
function gcdp( a as uinteger, b as uinteger ) as uinteger
    if b = 0 then return a
    return gcdp( b, a mod b )
end function

function gcd(a as integer, b as integer) as uinteger
    return gcdp( abs(a), abs(b) )
end function

function lcm(a as integer, b as integer) as uinteger
    return abs(a*b)/gcd(a,b)
end function

print "lcm( 12, -18) = "; lcm(12, -18)
print "lcm( 15,  12) = "; lcm(15, 12)
print "lcm(-10, -14) = "; lcm(-10, -14)
print "lcm(  0,   1) = "; lcm(0,1)
```

Output:

```
lcm( 12, -18) = 36
lcm( 15,  12) = 60
lcm(-10, -14) = 70
lcm(  0,   1) = 0
```

Frink (/wiki/Category:Frink)

Frink has a built-in LCM function that handles arbitrarily-large integers.

```
println[lcm[256204778801521550085490633230958956 1, 679545449426828292043156566168428 28
```

FunL (/wiki/Category:FunL)

FunL has function `lcm` in module `integers` with the following definition:

```
def
  lcm( _, 0 ) =  0
  lcm( 0, _ ) =  0
  lcm( x, y ) =  abs( (x\gcd(x, y)) y )
```

GAP (/wiki/Category:GAP)

```
# Built-in
LcmInt(12, 18);
# 36
```

Go (/wiki/Category:Go)

```
package main

import (
    "fmt"
    "math/big"
)

var m, n, z big.Int (https://golang.org/search?q=big.Int)

func init() {
    m.SetString("256204778801521550085490633230958956l", 10)
    n.SetString("679545449426828292043156566168428281 9", 10)
}

func main() {
    fmt.Println(z.Mul(z.Div(&m, z.GCD(nil, nil, &m, &n)), &n))
}
```

Output:

```
1566925124003829826223212517517200259473120608119352786 9
```

Groovy (/wiki/Category:Groovy)

```
def (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20def) gcd
gcd = { m, n -> m = m.abs(); n = n.abs(); n == 0 ? m : m%n == 0 ? n : gcd(n, m % n) }

def (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20def) lcd = { m, n ->

[[m: 12, n: 18, l: 36],
 [m: -6, n: 14, l: 42],
 [m: 35, n: 0, l: 0]].each (https://www.google.de/search?q=site%3Agroovy.codehaus.org/
    println (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20println) "LC
    assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) lcd(t
}
```

Output:

```
LCD of 12, 18 is 36
LCD of -6, 14 is 42
LCD of 35, 0 is 0
```

GW-BASIC (/wiki/Category:GW-BASIC)

Translation of: C
Works with: PC-BASIC (/mw/index.php?title=PC-BASIC&action=edit&redlink=1) version any

```
10 PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) "LCM(35, 21) = ";
20 LET (http://www.qbasicnews.com/qboho/qcklet.shtml) MLCM = 35
30 LET (http://www.qbasicnews.com/qboho/qcklet.shtml) NLCM = 21
40 GOSUB 200: ' Calculate LCM
50 PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) LCM
60 END (http://www.qbasicnews.com/qboho/qckend.shtml)

195 ' Calculate LCM
200 LET (http://www.qbasicnews.com/qboho/qcklet.shtml) MGCD = MLCM
210 LET (http://www.qbasicnews.com/qboho/qcklet.shtml) NGCD = NLCM
220 GOSUB 400: ' Calculate GCD
230 LET (http://www.qbasicnews.com/qboho/qcklet.shtml) LCM = MLCM / GCD * NLCM
240 RETURN

395 ' Calculate GCD
400 WHILE MGCD <> 0
410  LET (http://www.qbasicnews.com/qboho/qcklet.shtml) TMP = MGCD
420  LET (http://www.qbasicnews.com/qboho/qcklet.shtml) MGCD = NGCD MOD (http://www.qb
430  LET (http://www.qbasicnews.com/qboho/qcklet.shtml) NGCD = TMP
440 WEND
450 LET (http://www.qbasicnews.com/qboho/qcklet.shtml) GCD = NGCD
460 RETURN
```

Haskell (/wiki/Category:Haskell)

That is already available as the function *lcm* in the Prelude. Here's the implementation:

```
lcm (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:lcm) :: (I
lcm (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:lcm) _ 0 =
lcm (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:lcm) 0 _ =
lcm (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:lcm) x y =
```

Icon (/wiki/Category:Icon) and Unicon (/wiki/Category:Unicon)

The lcm routine from the Icon Programming Library uses gcd. The routine is

```
link numbers
procedure main()
write("lcm of 18, 36 = ",lcm(18,36))
write("lcm of 0, 9 = ",lcm(0,9))
end
```

Library: Icon Programming Library (/wiki/Category:Icon_Programming_Library)
numbers provides lcm and gcd (https://www.cs.arizona.edu/icon/library/src/procs/numbers.icn) and looks like this:

```
procedure lcm(i, j)             #: least common multiple
   if (i =  0) | (j = 0) then return 0
   return abs(i * j) / gcd(i, j)
end
```

J (/wiki/Category:J)

J provides the dyadic verb `*.` which returns the least common multiple of its left and right arguments.

```
      12 *. 18
36
   12 *. 18 22
36 132
   */ 12 18 22
396
   0 1 0 1 *. 0 0 1 1  NB. for truth valued arguments (0 and 1) it is equivalent to "a
0 0 0 1
   */~ 0 1
0 0
0 1
```

Note: least common multiple is the original boolean multiplication. Constraining the universe of values to 0 and 1 allows us to additionally define logical negation (and boolean algebra was redefined to include this constraint in the early 1900s - the original concept of boolean algebra is now known as a boolean ring).

Java (/wiki/Category:Java)

```
import java.util.Scanner;

public class LCM{
   public static void main(String (https://www.google.com/search?hl=en&q=allinurl%3Ast
      Scanner aScanner = new Scanner(System (https://www.google.com/search?hl=en&q=all

      //prompts user for values to find the LCM for, then saves them to m and n
      System (https://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btr
      int m = aScanner.nextInt();
      System (https://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btr
      int n = aScanner.nextInt();
      int lcm = (n == m || n == 1) ? m :(m == 1 ? n : 0);
      /* this section increases the value of mm until it is greater
      / than or equal to nn, then does it again when the lesser
      / becomes the greater--if they aren't equal. If either value is 1,
      / no need to calculate*/
      if (lcm == 0) {
         int mm = m, nn = n;
         while (mm != nn) {
             while (mm < nn) { mm += m; }
             while (nn < mm) { nn += n; }
         }
         lcm = mm;
      }
      System (https://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btr
   }
}
```

JavaScript (/wiki/Category:JavaScript)

ES5

Computing the least common multiple of an integer array, using the associative law:

$\mathrm{lcm}(a, b, c) = \mathrm{lcm}(\mathrm{lcm}(a, b), c)$,

$\mathrm{lcm}(a_1, a_2, ..., a_n) = \mathrm{lcm}(\mathrm{lcm}(a_1, a_2, ..., a_{n-1}), a_n)$.

```
function LCM(A)  // A is an integer array (e.g. [-50,25,-45,-18,90,447])
{
    var n = A.length, a = Math.abs(A[0]);
    for (var i = 1; i < n; i++)
     { var b = Math.abs(A[i]), c = a;
       while (a && b){ a > b ? a %= b : b %= a; }
       a = Math.abs(c*A[i])/(a+b);
     }
    return a;
}

/* For example:
   LCM([-50,25,-45,-18,90,447]) -> 67050
*/
```

ES6

Translation of: Haskell

```
(() => {
    'use strict';

    // gcd :: Integral a => a -> a -> a
    let gcd = (x, y) => {
        let _gcd = (a, b) => (b === 0 ? a : _gcd(b, a % b)),
            abs = Math.abs;
        return _gcd(abs(x), abs(y));
    }

    // lcm :: Integral a => a -> a -> a
    let lcm = (x, y) =>
        x === 0 || y === 0 ? 0 : Math.abs(Math.floor(x / gcd(x, y)) * y);

    // TEST
    return lcm(12, 18);

})();
```

Output:

```
36
```

jq (/wiki/Category:Jq)

Direct method

```
# Define the helper function to take advantage of jq's tail-recursion optimization
def lcm(m; n):
  def _lcm:
    # state is [m, n, i]
    if (.[2] % .[1]) == 0 then .[2] else (.[0:2] + [.[2] + m]) | _lcm end;
  [m, n, m] | _lcm;
```

Julia (/wiki/Category:Julia)

Built-in function:

```
lcm(m,n)
```

K (/wiki/Category:K)

```
   gcd:{:[~x;y;_f[y;x!y]]}
   lcm:{_abs _ x*y%gcd[x;y]}

   lcm .'(12 18; -6 14; 35 0)
36 42 0

   lcm/1+!20
232792560
```

Klingphix (/wiki/Category:Klingphix)

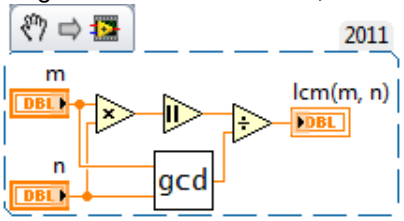
```
:gcd { u v -- n }
    abs int swap abs int swap

    [over over mod rot drop]
    [dup]
    while
    drop
;

:lcm { m n -- n }
    over over gcd rot swap div mult
;

12 18 lcm print nl  { 36 }

"End " input
```

Kotlin (/wiki/Category:Kotlin)

```
fun main(args: Array<String>) {
    fun gcd(a: Long, b: Long): Long = if (https://scala-lang.org) (b == 0L) a else (ht
    fun lcm(a: Long, b: Long): Long = a / gcd(a, b) * b
    println(lcm(15, 9))
}
```

LabVIEW (/wiki/Category:LabVIEW)

Requires GCD (/wiki/Greatest_common_divisor#LabVIEW). This image is a VI Snippet (http://zone.ni.com/devzone/cda/tut/p/id/9330), an executable image of LabVIEW (/wiki/LabVIEW) code. The LabVIEW version is shown on the top-right hand corner. You can download it, then drag-and-drop it onto the LabVIEW block diagram from a file browser, and it will appear as runnable, editable code.



(/wiki/File:LabVIEW_Least_common_multiple.png)

Lasso (/wiki/Category:Lasso)

```
define gcd(a,b) => {
        while(#b != 0) => {
                local(t = #b)
                #b = #a % #b
                #a = #t
        }
        return #a
}
define lcm(m,n) => {
         #m == 0 || #n == 0 ? return 0
         local(r = (#m * #n) / decimal(gcd(#m, #n)))
         return integer(#r)->abs
}

lcm(-6, 14)
lcm(2, 0)
lcm(12, 18)
lcm(12, 22)
lcm(7, 31)
```

Output:

```
42
0
36
132
217
```

Liberty BASIC (/wiki/Category:Liberty_BASIC)

```
print "Least Common Multiple of 12 and 18 is "; LCM(12, 18)
end

function LCM(m, n)
    LCM = abs(m * n) / GCD(m, n)
end function

function GCD(a, b)
    while b
        c = a
        a = b
        b = c mod b
    wend
    GCD = abs(a)
end function
```

Logo (/wiki/Category:Logo)

```
to abs :n
  output sqrt product :n :n
end

to gcd :m :n
  output ifelse :n = 0 [ :m ] [ gcd :n modulo :m :n ]
end

to lcm :m :n
  output quotient (abs product :m :n) gcd :m :n
end
```

Demo code:

```
print lcm 38 46
```

Output:

```
874
```

Lua (/wiki/Category:Lua)

```
function gcd( m, n )
    while n ~= 0 do
        local q = m
        m = n
        n = q % n
    end
    return m
end

function lcm( m, n )
    return ( m ~= 0 and n ~= 0 ) and m * n / gcd( m, n ) or 0
end

print( lcm(12,18) )
```

Maple (/wiki/Category:Maple)

The least common multiple of two integers is computed by the built-in procedure ilcm in Maple. This should not be confused with lcm, which computes the least common multiple of polynomials.

```
> ilcm( 12, 18 );
                                36
```

Mathematica (/wiki/Category:Mathematica)

/Wolfram Language (/wiki/Category:Wolfram_Language)

```
LCM[18,12]
-> 36
```

MATLAB (/wiki/Category:MATLAB) / Octave (/wiki/Category:Octave)

```
lcm (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/lcm.html)(a,b)
```

Maxima (/wiki/Category:Maxima)

```
lcm(a, b);   /* a and b may be integers or polynomials */

/* In Maxima the gcd of two integers is always positive, and a * b = gcd(a, b) * lcm(a
so the lcm may be negative. To get a positive lcm, simply do */

abs(lcm(a, b))
```

Microsoft Small Basic (/wiki/Category:Microsoft_Small_Basic)

Translation of: C

```
Textwindow.Write("LCM(35, 21) = ")
mlcm = 35
nlcm = 21
CalculateLCM()
TextWindow.WriteLine(lcm)

Sub CalculateLCM
  mgcd = mlcm
  ngcd = nlcm
  CalculateGCD()
  lcm = mlcm / gcd * nlcm
EndSub

Sub CalculateGCD
  While mgcd <> 0
    tmp = mgcd
    mgcd = Math.Remainder(ngcd, mgcd)
    ngcd = tmp
  EndWhile
  gcd = ngcd
EndSub
```

MiniScript (/wiki/Category:MiniScript)

```
gcd = function(a, b)
    while b
        temp = b
        b = a % b
        a = temp
    end while
    return abs(a)
end function

lcm = function(a,b)
    if not a and not b then return 0
    return abs(a * b) / gcd(a, b)
end function

print lcm(18,12)
```

Output:

```
36
```

min (/wiki/Category:Min)

Works with: min (/wiki/Min) version 0.19.6

```
((0 <) (-1 *) when) :abs
((dup 0 ==) (pop abs) (swap over mod) () linrec) :gcd
(over over gcd '* dip div) :lcm
```

МК-61/52 (/wiki/Category:%D0%9C%D0%9A-61/52)

```
ИПА     ИПВ     *       |x|     ПС      ИПА     ИПВ     /       [x]     П9
ИПА     ИПВ     ПА      ИП9     *       -       ПВ      x=0     05      ИПС
ИПА     /       С/П
```

ML (/wiki/Category:ML)

mLite (/wiki/Category:MLite)

```
fun gcd (a, 0) = a
      | (0, b) = b
      | (a, b) where (a < b)
               = gcd (a, b rem a)
      | (a, b) = gcd (b, a rem b)

fun lcm (a, b) = let val d = gcd (a, b)
                 in a * b div d
                 end
```

Modula-2 (/wiki/Category:Modula-2)

Translation of: C
Works with: ADW Modula-2 (/wiki/ADW_Modula-2) version any (Compile with the linker option *Console Application*).

```
MODULE LeastCommonMultiple;

FROM STextIO IMPORT
  WriteString, WriteLn;
FROM SWholeIO IMPORT
  WriteInt;

PROCEDURE GCD(M, N: INTEGER): INTEGER;
VAR
  Tmp: INTEGER;
BEGIN
  WHILE M <> 0 DO
    Tmp := M;
    M := N MOD M;
    N := Tmp;
  END;
  RETURN N;
END GCD;

PROCEDURE LCM(M, N: INTEGER): INTEGER;
BEGIN
  RETURN M / GCD(M, N) * N;
END LCM;

BEGIN
  WriteString("LCM(35, 21) = ");
  WriteInt(LCM(35, 21), 1);
  WriteLn;
END LeastCommonMultiple.
```

Nanoquery (/wiki/Category:Nanoquery)

```
def gcd(a, b)
        if (a < 1) or (b < 1)
                throw new(InvalidNumberException, "gcd cannot be calculated on values
        end

        c = 0
        while b != 0
                c = a
                a = b
                b = c % b
        end

        return a
end

def lcm(m, n)
        return (m * n) / gcd(m, n)
end

println lcm(12, 18)
println lcm(6, 14)
println lcm(1,2) = lcm(2,1)
```

Output:

```
36
42
true
```

NetRexx (/wiki/Category:NetRexx)

```
/* NetRexx */
options replace format comments java crossref symbols nobinary

numeric digits 3000

runSample(arg)
return

-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
method lcm(m_, n_) public static
  L_ = m_ * n_ % gcd(m_, n_)
  return L_

-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
-- Euclid's algorithm - iterative implementation
method gcd(m_, n_) public static
  loop while n_ > 0
    c_ = m_ // n_
    m_ = n_
    n_ = c_
    end
  return m_

-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
method runSample(arg) private static
  parse arg samples
  if samples = '' | samples = '.' then
    samples = '-6 14 =    42 |' -
              '3  4 =    12 |' -
             '18 12 =    36 |' -
              '2  0 =     0 |' -
              '0 85 =     0 |' -
             '12 18 =    36 |' -
              '5 12 =    60 |' -
             '12 22 =   132 |' -
              '7 31 =   217 |' -
            '117 18 =   234 |' -
             '38 46 =   874 |' -
          '18 12 -5 =   180 |' -
          '-5 18 12 =   180 |' - -- confirm that other permutations work
          '12 -5 18 =   180 |' -
       '18 12 -5 97 = 17460 |' -
             '30 42 =   210 |' -
             '30 42 =     . |' - -- 210; no verification requested
             '18 12'             -- 36

  loop while samples \= ''
    parse samples sample '|' samples
    loop while sample \= ''
      parse sample mnvals '=' chk sample
      if chk = '' then chk = '.'
      mv = mnvals.word(1)
      loop w_ = 2 to mnvals.words mnvals
        nv = mnvals.word(w_)
```

```
        mv = mv.abs
        nv = nv.abs
        mv = lcm(mv, nv)
        end w_
      lv = mv
      select case chk
        when '.' then state = ''
        when lv  then state = '(verified)'
        otherwise     state = '(failed)'
        end
      mnvals = mnvals.space(1, ',').changestr(',', ', ')
      say 'lcm of' mnvals.right(15.max(mnvals.length)) 'is' lv.right(5.max(lv.length))
      end
    end

  return
```

Output:

```
lcm of          -6, 14 is    42 (verified)
lcm of            3, 4 is    12 (verified)
lcm of          18, 12 is    36 (verified)
lcm of            2, 0 is     0 (verified)
lcm of           0, 85 is     0 (verified)
lcm of          12, 18 is    36 (verified)
lcm of           5, 12 is    60 (verified)
lcm of          12, 22 is   132 (verified)
lcm of           7, 31 is   217 (verified)
lcm of         117, 18 is   234 (verified)
lcm of          38, 46 is   874 (verified)
lcm of      18, 12, -5 is   180 (verified)
lcm of      -5, 18, 12 is   180 (verified)
lcm of      12, -5, 18 is   180 (verified)
lcm of  18, 12, -5, 97 is 17460 (verified)
lcm of          30, 42 is   210 (verified)
lcm of          30, 42 is   210
lcm of          18, 12 is    36
```

Nim (/wiki/Category:Nim)

The standard module "math" provides a function "lcm" for two integers and for an open array of integers. If we absolutely want to compute the least common multiple with our own procedure, it can be done this way (less efficient than the function in the standard library which avoids the modulo):

```
proc gcd(u, v: int): auto =
  var
    u = u
    v = v
  while v != 0:
    u = u %% v
    swap u, v
  abs(u)

proc lcm(a, b: int): auto = abs(a * b) div gcd(a, b)

echo lcm(12, 18)
echo lcm(-6, 14)
```

Output:

```
36
42
```

Objeck (/wiki/Category:Objeck)

Translation of: C

```
class LCM {
  function : Main(args : String[]) ~ Nil {
    IO.Console->Print("lcm(35, 21) = ")->PrintLine(lcm(21,35));
  }

  function : lcm(m : Int, n : Int) ~ Int {
    return m / gcd(m, n) * n;
  }

  function : gcd(m : Int, n : Int) ~ Int {
    tmp : Int;
    while(m <> 0) { tmp := m; m := n % m; n := tmp; };
    return n;
  }
}
```

OCaml (/wiki/Category:OCaml)

```
let rec gcd u v =
  if v <> 0 then (gcd v (u mod v))
  else (abs (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALabs)

let lcm m n =
  match m, n with
  | 0, _ | _, 0 -> 0
  | m, n -> abs (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VAL

let () =
  Printf (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Printf.html).printf "lcm(3
```

Oforth (/wiki/Category:Oforth)

lcm is already defined into Integer class :

```
12 18 lcm
```

ooRexx (/wiki/Category:OoRexx)

```
say lcm(18, 12)

-- calculate the greatest common denominator of a numerator/denominator pair
::routine gcd private
  use arg x, y

  loop while y \= 0
      -- check if they divide evenly
      temp = x // y
      x = y
      y = temp
  end
  return x

-- calculate the least common multiple of a numerator/denominator pair
::routine lcm private
  use arg x, y
  return x / gcd(x, y) * y
```

Order (/wiki/Category:Order)

Translation of: bc

```
#include <order/interpreter.h>

#define ORDER_PP_DEF_8gcd ORDER_PP_FN( \
8fn(8U, 8V,                            \
    8if(8isnt_0(8V), 8gcd(8V, 8remainder(8U, 8V)), 8U)))

#define ORDER_PP_DEF_8lcm ORDER_PP_FN( \
8fn(8X, 8Y,                            \
    8if(8or(8is_0(8X), 8is_0(8Y)),     \
        0,                             \
        8quotient(8times(8X, 8Y), 8gcd(8X, 8Y)))))
// No support for negative numbers

ORDER_PP( 8to_lit(8lcm(12, 18)) )   // 36
```

PARI/GP (/wiki/Category:PARI/GP)

Built-in function:

```
lcm
```

Pascal (/wiki/Category:Pascal)

```
Program LeastCommonMultiple(output);

{$IFDEF FPC}
  {$MODE DELPHI}
{$ENDIF}

function lcm(a, b: longint): longint;
begin
  result := a;
  while (result mod b) <> 0 do
    inc(result, a);
end;

begin
  writeln('The least common multiple of 12 and 18 is: ', lcm(12, 18));
end.
```

Output:

```
The least common multiple of 12 and 18 is: 36
```

Perl (/wiki/Category:Perl)

Using GCD:

```
sub gcd {
        my ($x, $y) = @_;
        while ($x) { ($x, $y) = ($y % $x, $x) }
        $y
}

sub lcm {
        my ($x, $y) = @_;
        ($x && $y) and $x / gcd($x, $y) * $y or 0
}

print (https://perldoc.perl.org/functions/print.html) lcm(1001, 221);
```

Or by repeatedly increasing the smaller of the two until LCM is reached:

```
sub lcm {
        use integer;
        my ($x, $y) = @_;
        my ($f, $s) = @_;
        while ($f != $s) {
                ($f, $s, $x, $y) = ($s, $f, $y, $x) if $f > $s;
                $f = $s / $x * $x;
                $f += $x if $f < $s;
        }
        $f
}

print (https://perldoc.perl.org/functions/print.html) lcm(1001, 221);
```

Phix (/wiki/Category:Phix)

```
function lcm(integer m, integer n)
    return m / gcd(m, n) * n
end function
```

Phixmonti (/wiki/Category:Phixmonti)

```
def gcd /# u v -- n #/
        abs int swap abs int swap

        dup
        while
                over over mod rot drop dup
        endwhile
        drop
enddef

def lcm /# m n -- n #/
        over over gcd rot swap / *
enddef

12345 50 lcm print
```

PHP (/wiki/Category:PHP)

Translation of: D

```
echo lcm(12, 18) == 36;

function lcm($m, $n) {
    if ($m == 0 || $n == 0) return 0;
    $r = ($m * $n) / gcd($m, $n);
    return abs (http://www.php.net/abs)($r);
}

function gcd($a, $b) {
    while ($b != 0) {
        $t = $b;
        $b = $a % $b;
        $a = $t;
    }
    return $a;
}
```

PicoLisp (/wiki/Category:PicoLisp)

Using 'gcd' from Greatest common divisor#PicoLisp (/wiki/Greatest_common_divisor#PicoLisp):

```
(de lcm (A B)
   (abs (*/ A B (gcd A B))) )
```

PL/I (/wiki/Category:PL/I)

```
/* Calculate the Least Common Multiple of two integers. */

LCM: procedure options (main);          /* 16 October 2013 */
   declare (m, n) fixed binary (31);

   get (m, n);
   put edit ('The LCM of ', m, ' and ', n, ' is', LCM(m, n)) (a, x(1));

LCM: procedure (m, n) returns (fixed binary (31));
   declare (m, n) fixed binary (31) nonassignable;

   if m = 0 | n = 0 then return (0);
   return (abs(m*n) / GCD(m, n));
end LCM;

GCD: procedure (a, b) returns (fixed binary (31)) recursive;
   declare (a, b) fixed binary (31);

   if b = 0 then return (a);

   return (GCD (b, mod(a, b)) );

end GCD;
end LCM;
```

```
The LCM of             14  and             35  is            70
```

PowerShell (/wiki/Category:PowerShell)

version 1

```
function gcd ($a, $b)  {
    function pgcd ($n, $m)  {
        if($n -le $m) {
            if($n -eq 0) {$m}
            else{pgcd $n ($m-$n)}
        }
        else {pgcd $m $n}
    }
    $n = [Math]::Abs($a)
    $m = [Math]::Abs($b)
    (pgcd $n $m)
}
function lcm ($a, $b)  {
    [Math]::Abs($a*$b)/(gcd $a $b)
}
lcm 12 18
```

version 2

version2 is faster than version1

```
function gcd ($a, $b)  {
    function pgcd ($n, $m)  {
        if($n -le $m) {
            if($n -eq 0) {$m}
            else{pgcd $n ($m%$n)}
        }
        else {pgcd $m $n}
    }
    $n = [Math]::Abs($a)
    $m = [Math]::Abs($b)
    (pgcd $n $m)
}
function lcm ($a, $b)  {
    [Math]::Abs($a*$b)/(gcd $a $b)
}
lcm 12 18
```

Output:

```
36
```

Prolog (/wiki/Category:Prolog)

SWI-Prolog knows gcd.

```
lcm(X, Y, Z) :-
        Z is (http://pauillac.inria.fr/~deransar/prolog/bips.html) abs (http://pauilla
```

Example:

```
 ?- lcm(18,12, Z).
Z = 36.
```

PureBasic (/wiki/Category:PureBasic)

```
Procedure GCDiv(a, b); Euclidean algorithm
  Protected r
  While b
    r = b
    b = a%b
    a = r
  Wend
  ProcedureReturn a
EndProcedure

Procedure LCM(m,n)
  Protected t
  If m And n
    t=m*n/GCDiv(m,n)
  EndIf
  ProcedureReturn t*Sign(t)
EndProcedure
```

Python (/wiki/Category:Python)

Functional

gcd

Using the fractions libraries gcd (https://docs.python.org/library/fractions.html?highlight=fractions.gcd#fractions.gcd) function:

```
>>> import fractions
>>> def lcm(a,b): return abs(a * b) / fractions.gcd(a,b) if a and b else 0

>>> lcm(12, 18)
36
>>> lcm(-6, 14)
42
>>> assert lcm(0, 2) == lcm(2, 0) == 0
>>>
```

Or, for compositional flexibility, a curried **lcm**, expressed in terms of our own **gcd** function:

```
'''Least common multiple'''

from inspect import signature


# lcm :: Int -> Int -> Int
def lcm(x):
    '''The smallest positive integer divisible
       without remainder by both x and y.
    '''
    return lambda y: 0 if 0 in (x, y) else abs(
        y * (x // gcd_(x)(y))
    )


# gcd_ :: Int -> Int -> Int
def gcd_(x):
    '''The greatest common divisor in terms of
       the divisibility preordering.
    '''
    def go(a, b):
        return go(b, a % b) if 0 != b else a
    return lambda y: go(abs(x), abs(y))


# TEST ----------------------------------------------------
# main :: IO ()
def main():
    '''Tests'''

    print(
        fTable(
            __doc__ + 's of 60 and [12..20]:'
        )(repr)(repr)(
            lcm(60)
        )(enumFromTo(12)(20))
    )

    pairs = [(0, 2), (2, 0), (-6, 14), (12, 18)]
    print(
        fTable(
            '\n\n' + __doc__ + 's of ' + repr(pairs) + ':'
        )(repr)(repr)(
            uncurry(lcm)
        )(pairs)
    )


# GENERIC -------------------------------------------------

# enumFromTo :: (Int, Int) -> [Int]
def enumFromTo(m):
    '''Integer enumeration from m to n.'''
    return lambda n: list(range(m, 1 + n))
```

```
# uncurry :: (a -> b -> c) -> ((a, b) -> c)
def uncurry(f):
    '''A function over a tuple, derived from
       a vanilla or curried function.
    '''
    if 1 < len(signature(f).parameters):
        return lambda xy: f(*xy)
    else:
        return lambda xy: f(xy[0])(xy[1])


# unlines :: [String] -> String
def unlines(xs):
    '''A single string derived by the intercalation
       of a list of strings with the newline character.
    '''
    return '\n'.join(xs)


# FORMATTING ---------------------------------------------

# fTable :: String -> (a -> String) ->
#                     (b -> String) -> (a -> b) -> [a] -> String
def fTable(s):
    '''Heading -> x display function -> fx display function ->
                    f -> xs -> tabular string.
    '''
    def go(xShow, fxShow, f, xs):
        ys = [xShow(x) for x in xs]
        w = max(map(len, ys))
        return s + '\n' + '\n'.join(map(
            lambda x, y: y.rjust(w, ' ') + ' -> ' + fxShow(f(x)),
            xs, ys
        ))
    return lambda xShow: lambda fxShow: lambda f: lambda xs: go(
        xShow, fxShow, f, xs
    )


# MAIN ---
if __name__ == '__main__':
    main()
```

Output:

```
Least common multiples of 60 and [12..20]:
12 -> 60
13 -> 780
14 -> 420
15 -> 60
16 -> 240
17 -> 1020
18 -> 180
19 -> 1140
20 -> 60

Least common multiples of [(0, 2), (2, 0), (-6, 14), (12, 18)]:
  (0, 2) -> 0
  (2, 0) -> 0
(-6, 14) -> 42
(12, 18) -> 36
```

Procedural

Prime decomposition

This imports Prime decomposition#Python (/wiki/Prime_decomposition#Python)

```
from prime_decomposition import decompose
try:
    reduce
except NameError:
    from functools import reduce

def lcm(a, b):
    mul = int.__mul__
    if a and b:
        da = list(decompose(abs(a)))
        db = list(decompose(abs(b)))
        merge= da
        for d in da:
            if d in db: db.remove(d)
        merge += db
        return reduce(mul, merge, 1)
    return 0

if __name__ == '__main__':
    print( lcm(12, 18) )    # 36
    print( lcm(-6, 14) )    # 42
    assert lcm(0, 2) == lcm(2, 0) == 0
```

Iteration over multiples

```
>>> def lcm(*values):
        values = set([abs(int(v)) for v in values])
        if values and 0 not in values:
                n = n0 = max(values)
                values.remove(n)
                while any( n % m for m in values ):
                        n += n0
                return n
        return 0

>>> lcm(-6, 14)
42
>>> lcm(2, 0)
0
>>> lcm(12, 18)
36
>>> lcm(12, 18, 22)
396
>>>
```

Repeated modulo

Translation of: Tcl

```
>>> def lcm(p,q):
        p, q = abs(p), abs(q)
        m = p * q
        if not m: return 0
        while True:
                p %= q
                if not p: return m // q
                q %= p
                if not q: return m // p


>>> lcm(-6, 14)
42
>>> lcm(12, 18)
36
>>> lcm(2, 0)
0
>>>
```

Qi (/wiki/Category:Qi)

```
(define gcd
  A 0 -> A
  A B -> (gcd B (MOD A B)))

(define lcm A B -> (/ (* A B) (gcd A B)))
```

Quackery (/wiki/Category:Quackery)

```
[ [ dup while
    tuck mod again ]
  drop abs ]         is gcd ( n n --> n )

[ 2dup and iff
    [ 2dup gcd
      / * abs ]
  else
    [ 2drop 0 ] ]    is lcm ( n n --> n )
```

R (/wiki/Category:R)

```
"%gcd%" <- function(u, v) {ifelse(u %% v != 0, v %gcd% (u%%v), v)}

"%lcm%" <- function(u, v) { abs(u*v)/(u %gcd% v)}

print (50 %lcm% 75)
```

Racket (/wiki/Category:Racket)

Racket already has defined both lcm and gcd funtions:

```
#lang racket
(lcm 3 4 5 6)        ;returns 60
(lcm 8 108)          ;returns 216
(gcd 8 108)          ;returns 4
(gcd 108 216 432)    ;returns 108
```

Raku (/wiki/Category:Raku)

(formerly Perl 6) This function is provided as an infix so that it can be used productively with various metaoperators.

```
say 3 lcm 4;            # infix
say [lcm] 1..20;        # reduction
say ~(1..10 Xlcm 1..10) # cross
```

Output:

```
12
232792560
1 2 3 4 5 6 7 8 9 10 2 2 6 4 10 6 14 8 18 10 3 6 3 12 15 6 21 24 9 30 4 4 12 4 20 12 2
```

Retro (/wiki/Category:Retro)

This is from the math extensions library included with Retro.

```
: gcd ( ab-n ) [ tuck mod dup ] while drop ;
: lcm ( ab-n ) 2over gcd [ * ] dip / ;
```

REXX (/wiki/Category:REXX)

version 1

The **lcm** subroutine can handle any number of integers and/or arguments.

The integers (negative/zero/positive) can be (as per the **numeric digits**) up to ten thousand digits.

Usage note: the integers can be expressed as a list and/or specified as individual arguments (or as mixed).

```
/*REXX program finds the  LCM  (Least Common Multiple)  of any number of integers.
numeric digits 10000                              /*can handle 10k decimal digit numbe
say 'the LCM of      19  and   0                    is ⟶   '     lcm(19    0
say 'the LCM of       0  and  85                    is ⟶   '     lcm( 0   85
say 'the LCM of      14  and  -6                    is ⟶   '     lcm(14,  -6
say 'the LCM of      18  and  12                    is ⟶   '     lcm(18   12
say 'the LCM of      18  and  12  and  -5           is ⟶   '     lcm(18   12,   -5
say 'the LCM of      18  and  12  and  -5  and  97 is ⟶   '     lcm(18,  12,   -5,
say 'the LCM of 2**19-1  and  2**521-1              is ⟶   '     lcm(2**19-1    2**52
                                                  /* [↑]   7th  &  13th  Mersenne prime
exit                                              /*stick a fork in it,  we're all done
/*──────────────────────────────────────────────────────────────────────────────────────
lcm: procedure; parse arg $,_; $=$ _;           do i=3  to arg();  $=$ arg(i);  end  /
     parse var $ x $                                  /*obtain the first value in args
     x=abs(x)                                         /*use the absolute value of  X.
               do  while $\==''                       /*process the remainder of args.
               parse var $ ! $;    if !<0  then !=-!  /*pick off the next arg (ABS val
               if !==0  then return 0                 /*if zero, then LCM is also zero
               d=x*!                                  /*calculate part of the LCM here
                     do  until !==0;    parse  value   x//!  !     with     !  x
                     end   /*until*/                 /* [↑]  this is a short & fast G
               x=d%x                                  /*divide the pre-calculated valu
               end   /*while*/                       /* [↑]  process subsequent args.
     return x                                         /*return with the LCM of the arg
```

output when using the (internal) supplied list:

```
the LCM of      19  and   0                    is ⟶   0
the LCM of       0  and  85                    is ⟶   0
the LCM of      14  and  -6                    is ⟶   42
the LCM of      18  and  12                    is ⟶   36
the LCM of      18  and  12  and  -5           is ⟶   180
the LCM of      18  and  12  and  -5  and  97 is ⟶   17460
the LCM of 2**19-1  and  2**521-1              is ⟶   359912417083689697563871582424
```

version 2

Translation of: REXX version 0
using different argument handling-
Use as lcm(a,b,c,---)

```
lcm2: procedure
x=abs(arg(1))
do k=2 to arg() While x<>0
  y=abs(arg(k))
  x=x*y/gcd2(x,y)
  end
return x

gcd2: procedure
x=abs(arg(1))
do j=2 to arg()
  y=abs(arg(j))
  If y<>0 Then Do
    do until z==0
      z=x//y
      x=y
      y=z
      end
    end
  end
return x
```

Ring (/wiki/Category:Ring)

```
see lcm(24,36)

func lcm m,n
     lcm = m*n / gcd(m,n)
     return lcm

func gcd gcd, b
     while b
           c   = gcd
           gcd = b
           b   = c % b
     end
     return gcd
```

Ruby (/wiki/Category:Ruby)

Ruby has an `Integer#lcm` method, which finds the least common multiple of two integers.

```
irb(main):001:0> 12.lcm 18
=> 36
```

I can also write my own `lcm` method. This one takes any number of arguments.

```
def gcd(m, n)
  m, n = n, m % n until n.zero?
  m.abs
end

def lcm(*args)
  args.inject(1) do |m, n|
    return 0 if n.zero?
    (m * n).abs / gcd(m, n)
  end
end

p lcm 12, 18, 22
p lcm 15, 14, -6, 10, 21
```

Output:

```
396
210
```

Run BASIC (/wiki/Category:Run_BASIC)

This example is **incorrect**. Please fix the code and remove this message.
Details: This example computes GCD not LCM.

```
print lcm(22,44)

function lcm(m,n)
 while n
   t = m
   m = n
   n = t mod n
 wend
lcm = m
end function
```

Rust (/wiki/Category:Rust)

This implementation uses a recursive implementation of Stein's algorithm to calculate the gcd.

```
use std::cmp::{max, min};

fn gcd(a: usize, b: usize) -> usize {
    match ((a, b), (a & 1, b & 1)) {
        ((x, y), _) if x == y => y,
        ((0, x), _) | ((x, 0), _) => x,
        ((x, y), (0, 1)) | ((y, x), (1, 0)) => gcd(x >> 1, y),
        ((x, y), (0, 0)) => gcd(x >> 1, y >> 1) << 1,
        ((x, y), (1, 1)) => {
            let (x, y) = (min(x, y), max(x, y));
            gcd((y - x) >> 1, x)
        }
        _ => unreachable!(),
    }
}

fn lcm(a: usize, b: usize) -> usize {
    a * b / gcd(a, b)
}

fn main() {
    println!("{}", lcm(6324, 234))
}
```

Scala (/wiki/Category:Scala)

```
def (https://scala-lang.org) gcd(a: Int, b: Int):Int=if (https://scala-lang.org) (b==0
def (https://scala-lang.org) lcm(a: Int, b: Int)=(a*b).abs/gcd(a,b)
```

```
lcm(12, 18)   // 36
lcm( 2,  0)   // 0
lcm(-6, 14)   // 42
```

Scheme (/wiki/Category:Scheme)

```
>(define gcd (lambda (a b)
         (if (zero? b)
             a
             (gcd b (remainder a b)))))
>(define lcm (lambda (a b)
         (if (or (zero? a) (zero? b))
             0
             (abs (* b (floor (/ a (gcd a b))))))))
>(lcm 12 18)
36
```

Seed7 (/wiki/Category:Seed7)

```
$ include "seed7_05.s7i";

const func integer: gcd (in var integer: a, in var integer: b) is func
  result
    var integer: gcd is 0;
  local
    var integer: help is 0;
  begin
    while a <> 0 do
      help := b rem a;
      b := a;
      a := help;
    end while;
    gcd := b;
  end func;

const func integer: lcm (in integer: a, in integer: b) is
  return a div gcd(a, b) * b;

const proc: main is func
  begin
    writeln("lcm(35, 21) = " <& lcm(21, 35));
  end func;
```

Original source: [1] (http://seed7.sourceforge.net/algorith/math.htm#lcm)

SenseTalk (/wiki/Category:SenseTalk)

```
function gcd m, n
        repeat while m is greater than 0
                put m into temp
                put n modulo m into m
                put temp into n
        end repeat
        return n
end gcd

function lcm m, n
        return m divided by gcd(m, n) times n
end lcm
```

Sidef (/wiki/Category:Sidef)

Built-in:

```
say Math.lcm(1001, 221)
```

Using GCD:

```
func gcd(a, b) {
    while (a) { (a, b) = (b % a, a) }
    return b
}

func lcm(a, b) {
    (a && b) ? (a / gcd(a, b) * b) : 0
}

say lcm(1001, 221)
```

Output:

```
17017
```

Smalltalk (/wiki/Category:Smalltalk)

Smalltalk has a built-in `lcm` method on `SmallInteger`:

```
12 lcm: 18
```

Sparkling (/wiki/Category:Sparkling)

```
function factors(n) {
        var f = {};

        for var i = 2; n > 1; i++ {
                while n % i == 0 {
                        n /= i;
                        f[i] = f[i] != nil ? f[i] + 1 : 1;
                }
        }

        return f;
}

function GCD(n, k) {
        let f1 = factors(n);
        let f2 = factors(k);

        let fs = map(f1, function(factor, multiplicity) {
                let m = f2[factor];
                return m == nil ? 0 : min(m, multiplicity);
        });

        let rfs = {};
        foreach(fs, function(k, v) {
                rfs[sizeof rfs] = pow(k, v);
        });

        return reduce(rfs, 1, function(x, y) { return x * y; });
}

function LCM(n, k) {
        return n * k / GCD(n, k);
}
```

Standard ML (/wiki/Category:Standard_ML)

```
val rec gcd = fn (x, 0) => abs x | p as (_, y) => gcd (y, Int.rem p)

val lcm = fn p as (x, y) => Int.quot (abs (x * y), gcd p)
```

Swift (/wiki/Category:Swift)

Using the Swift GCD function.

```
func lcm(a:Int, b:Int) -> Int {
    return abs(a * b) / gcd_rec(a, b)
}
```

Tcl (/wiki/Category:Tcl)

```
proc lcm {p q} {
    set m [expr {$p * $q}]
    if {!$m} {return 0}
    while 1 {
        set p [expr {$p % $q}]
        if {!$p} {return [expr {$m / $q}]}
        set q [expr {$q % $p}]
        if {!$q} {return [expr {$m / $p}]}
    }
}
```

Demonstration

```
puts [lcm 12 18]
```

Output:

```
36
```

TI-83 BASIC (/wiki/Category:TI-83_BASIC)

```
lcm(12,18
              36
```

TSE SAL (/wiki/Category:TSE_SAL)

```
// library: math: get: least: common: multiple <description></description> <version co
INTEGER PROC FNMathGetLeastCommonMultipleI( INTEGER x1I, INTEGER x2I )
 //
 RETURN( x1I * x2I / FNMathGetGreatestCommonDivisorI( x1I, x2I ) )
 //
END

// library: math: get: greatest: common: divisor <description>greatest common divisor
INTEGER PROC FNMathGetGreatestCommonDivisorI( INTEGER x1I, INTEGER x2I )
 //
 IF ( x2I == 0 )
  //
  RETURN( x1I )
  //
 ENDIF
 //
 RETURN( FNMathGetGreatestCommonDivisorI( x2I, x1I MOD x2I ) )
 //
END

PROC Main()
 //
 STRING s1[255] = "10"
 STRING s2[255] = "20"
 REPEAT
  IF ( NOT ( Ask( "math: get: least: common: multiple: x1I = ", s1, _EDIT_HISTORY_ ) )
  IF ( NOT ( Ask( "math: get: least: common: multiple: x2I = ", s2, _EDIT_HISTORY_ ) )
  Warn( FNMathGetLeastCommonMultipleI( Val( s1 ), Val( s2 ) ) ) // gives e.g. 10
 UNTIL FALSE
END
```

TXR (/wiki/Category:TXR)

```
$ txr -p '(lcm (expt 2 123) (expt 6 49) 17)'
43259338018880832376582582128138484281161556655442781051813888
```

uBasic/4tH (/wiki/Category:UBasic/4tH)

Translation of: BBC BASIC

```
Print "LCM of 12 : 18 = "; FUNC(_LCM(12,18))

End


_GCD_Iterative_Euclid Param(2)
  Local (1)
  Do While b@
    c@ = a@
    a@ = b@
    b@ = c@ % b@
  Loop
Return (ABS(a@))


_LCM Param(2)
If a@*b@
  Return (ABS(a@*b@)/FUNC(_GCD_Iterative_Euclid(a@,b@)))
Else
  Return (0)
EndIf
```

Output:

```
LCM of 12 : 18 = 36

0 OK, 0:330
```

UNIX Shell (/wiki/Category:UNIX_Shell)

$\operatorname{lcm}(m, n) = \left|\frac{m \times n}{\gcd(m, n)}\right|$

Works with: Bourne Shell (/wiki/Bourne_Shell)

```
gcd() {
        # Calculate $1 % $2 until $2 becomes zero.
        until test 0 -eq "$2"; do
                # Parallel assignment: set -- 1 2
                set -- "$2" "`expr "$1" % "$2"`"
        done

        # Echo absolute value of $1.
        test 0 -gt "$1" && set -- "`expr 0 - "$1"`"
        echo "$1"
}

lcm() {
        set -- "$1" "$2" "`gcd "$1" "$2"`"
        set -- "`expr "$1" \* "$2" / "$3"`"
        test 0 -gt "$1" && set -- "`expr 0 - "$1"`"
        echo "$1"
}

lcm 30 -42
# => 210
```

C Shell (/wiki/Category:C_Shell)

```
alias gcd eval \''set gcd_args=( \!*:q )        \\
        @ gcd_u=$gcd_args[2]                    \\
        @ gcd_v=$gcd_args[3]                    \\
        while ( $gcd_v != 0 )                   \\
                @ gcd_t = $gcd_u % $gcd_v       \\
                @ gcd_u = $gcd_v                \\
                @ gcd_v = $gcd_t                \\
        end                                     \\
        if ( $gcd_u < 0 ) @ gcd_u = - $gcd_u    \\
        @ $gcd_args[1]=$gcd_u                   \\
'\'

alias lcm eval \''set lcm_args=( \!*:q )        \\
        @ lcm_m = $lcm_args[2]                  \\
        @ lcm_n = $lcm_args[3]                  \\
        gcd lcm_d $lcm_m $lcm_n                 \\
        @ lcm_r = ( $lcm_m * $lcm_n ) / $lcm_d  \\
        if ( $lcm_r < 0 ) @ lcm_r = - $lcm_r    \\
        @ $lcm_args[1] = $lcm_r                 \\
'\'

lcm result 30 -42
echo $result
# => 210
```

Ursa (/wiki/Category:Ursa)

```
import "math"
out (lcm 12 18) endl console
```

Output:

```
36
```

Vala (/wiki/Category:Vala)

```
int lcm(int a, int b){
    /*Return least common multiple of two ints*/
    // check for 0's
    if (a == 0 || b == 0)
        return 0;

    // Math.abs(x) only works for doubles, Math.absf(x) for floats
    if (a < 0)
        a *= -1;
    if (b < 0)
        b *= -1;

    int x = 1;
    while (true){
        if (a * x % b == 0)
            return a*x;
        x++;
    }
}

void main(){
    int a = 12;
    int b = 18;

    stdout.printf("lcm(%d, %d) = %d\n", a, b, lcm(a, b));
}
```

VBA (/wiki/Category:VBA)

```
Function gcd(u As Long, v As Long) As Long
    Dim t As Long
    Do While v
        t = u
        u = v
        v = t Mod v
    Loop
    gcd = u
End Function
Function lcm(m As Long, n As Long) As Long
    lcm = Abs(m * n) / gcd(m, n)
End Function
```

VBScript (/wiki/Category:VBScript)

```
Function LCM(a,b)
        LCM = POS((a * b)/GCD(a,b))
End Function

Function GCD(a,b)
        Do
                If a Mod b > 0 Then
                        c = a Mod b
                        a = b
                        b = c
                Else
                        GCD = b
                        Exit Do
                End If
        Loop
End Function

Function POS(n)
        If n < 0 Then
                POS = n * -1
        Else
                POS = n
        End If
End Function

i = WScript.Arguments(0)
j = WScript.Arguments(1)

WScript.StdOut.Write "The LCM of " & i & " and " & j & " is " & LCM(i,j) & "."
WScript.StdOut.WriteLine
```

Output:

```
C:\>cscript /nologo lcm.vbs 12 18
The LCM of 12 and 18 is 36.

C:\>cscript /nologo lcm.vbs 14 -6
The LCM of 14 and -6 is 42.

C:\>cscript /nologo lcm.vbs 0 35
The LCM of 0 and 35 is 0.

C:\>
```

Wortel (/wiki/Category:Wortel)

Operator

```
@lcm a b
```

Number expression

```
!#~km a b
```

Function (using gcd)

```
&[a b] *b /a @gcd a b
```

Wren (/wiki/Category:Wren)

```
var gcd = Fn.new { |x, y|
    while (y != 0) {
        var t = y
        y = x % y
        x = t
    }
    return x
}

var lcm = Fn.new { |x, y| (x*y).abs / gcd.call(x, y) }

var xys = [[12, 18], [-6, 14], [35, 0]]
for (xy in xys) {
    System.print("lcm(%(xy[0]), %(xy[1]))\t%("\b"*5) = %(lcm.call(xy[0], xy[1]))")
}
```

Output:

```
lcm(12, 18) = 36
lcm(-6, 14) = 42
lcm(35, 0)  = 0
```

XBasic (/wiki/Category:XBasic)

Translation of: C

Works with: Windows XBasic (/wiki/Windows_XBasic)

```
PROGRAM "leastcommonmultiple"
VERSION "0.0001"

DECLARE FUNCTION Entry()
INTERNAL FUNCTION Gcd(m&, n&)
INTERNAL FUNCTION Lcm(m&, n&)

FUNCTION Entry()
  PRINT "LCM(35, 21) ="; Lcm(35, 21)
END FUNCTION

FUNCTION Gcd(m&, n&)
  DO WHILE m& <> 0
    tmp& = m&
    m& = n& MOD (http://www.xbasic.org) m&
    n& = tmp&
  LOOP
  RETURN n&
END FUNCTION

FUNCTION Lcm(m&, n&)
  RETURN m& / Gcd(m&, n&) * n&
END FUNCTION

END PROGRAM
```

Output:

```
LCM(35, 21) = 105
```

XPL0 (/wiki/Category:XPL0)

```
include c:\cxpl\codes;

func GCD(M,N);  \Return the greatest common divisor of M and N
int  M, N;
int  T;
[while N do     \Euclid's method
    [T:= M;  M:= N;  N:= rem(T/N)];
return M;
];

func LCM(M,N);  \Return least common multiple
int  M, N;
return abs(M*N) / GCD(M,N);

\Display the LCM of two integers entered on command line
IntOut(0, LCM(IntIn(8), IntIn(8)))
```

Yabasic (/wiki/Category:Yabasic)

```
sub gcd(u, v)
    local t

    u = int(abs(u))
    v = int(abs(v))
    while(v)
        t = u
        u = v
        v = mod(t, v)
    wend
    return u
end sub

sub lcm(m, n)
    return m / gcd(m, n) * n
end sub

print "Least common multiple: ", lcm(12345, 23044)
```

zkl (/wiki/Category:Zkl)

```
fcn lcm(m,n){ (m*n).abs()/m.gcd(n) }  // gcd is a number method
```

Output:

```
zkl: lcm(12,18)
36
zkl: lcm(-6,14)
42
zkl: lcm(35,0)
0
```

Categories (/wiki/Special:Categories): Recursion (/wiki/Category:Recursion)
| Programming Tasks (/wiki/Category:Programming_Tasks)
| Solutions by Programming Task (/wiki/Category:Solutions_by_Programming_Task) | 11l (/wiki/Category:11l)
| 360 Assembly (/wiki/Category:360_Assembly) | 8th (/wiki/Category:8th) | Ada (/wiki/Category:Ada)
| ALGOL 68 (/wiki/Category:ALGOL_68) | ALGOL W (/wiki/Category:ALGOL_W) | APL (/wiki/Category:APL)
| AppleScript (/wiki/Category:AppleScript) | Arendelle (/wiki/Category:Arendelle)
| Arturo (/wiki/Category:Arturo) | Assembly (/wiki/Category:Assembly)
| X86 Assembly (/wiki/Category:X86_Assembly) | AutoHotkey (/wiki/Category:AutoHotkey)
| AutoIt (/wiki/Category:AutoIt) | AWK (/wiki/Category:AWK) | BASIC (/wiki/Category:BASIC)
| Applesoft BASIC (/wiki/Category:Applesoft_BASIC) | BBC BASIC (/wiki/Category:BBC_BASIC)
| IS-BASIC (/wiki/Category:IS-BASIC) | Tiny BASIC (/wiki/Category:Tiny_BASIC)
| Batch File (/wiki/Category:Batch_File) | Bc (/wiki/Category:Bc) | Befunge (/wiki/Category:Befunge)
| BQN (/wiki/Category:BQN) | Bracmat (/wiki/Category:Bracmat) | Brat (/wiki/Category:Brat)
| C (/wiki/Category:C) | C sharp (/wiki/Category:C_sharp) | C++ (/wiki/Category:C%2B%2B)
| Boost (/wiki/Category:Boost) | Clojure (/wiki/Category:Clojure) | COBOL (/wiki/Category:COBOL)
| Common Lisp (/wiki/Category:Common_Lisp) | D (/wiki/Category:D) | Dart (/wiki/Category:Dart)
| Delphi (/wiki/Category:Delphi) | DWScript (/wiki/Category:DWScript) | EchoLisp (/wiki/Category:EchoLisp)
| Elena (/wiki/Category:Elena) | Elixir (/wiki/Category:Elixir) | Erlang (/wiki/Category:Erlang)
| ERRE (/wiki/Category:ERRE) | Euphoria (/wiki/Category:Euphoria) | Excel (/wiki/Category:Excel)
| Ezhil (/wiki/Category:Ezhil) | F Sharp (/wiki/Category:F_Sharp) | Factor (/wiki/Category:Factor)
| Forth (/wiki/Category:Forth) | Fortran (/wiki/Category:Fortran) | FreeBASIC (/wiki/Category:FreeBASIC)
| Frink (/wiki/Category:Frink) | FunL (/wiki/Category:FunL) | GAP (/wiki/Category:GAP)
| Go (/wiki/Category:Go) | Groovy (/wiki/Category:Groovy) | GW-BASIC (/wiki/Category:GW-BASIC)
| Haskell (/wiki/Category:Haskell) | Icon (/wiki/Category:Icon) | Unicon (/wiki/Category:Unicon)
| Icon Programming Library (/wiki/Category:Icon_Programming_Library) | J (/wiki/Category:J)
| Java (/wiki/Category:Java) | JavaScript (/wiki/Category:JavaScript) | Jq (/wiki/Category:Jq)
| Julia (/wiki/Category:Julia) | K (/wiki/Category:K) | Klingphix (/wiki/Category:Klingphix)
| Kotlin (/wiki/Category:Kotlin) | LabVIEW (/wiki/Category:LabVIEW) | Lasso (/wiki/Category:Lasso)
| Liberty BASIC (/wiki/Category:Liberty_BASIC) | Logo (/wiki/Category:Logo) | Lua (/wiki/Category:Lua)
| Maple (/wiki/Category:Maple) | Mathematica (/wiki/Category:Mathematica)
| *Wolfram Language (/wiki/Category:Wolfram_Language)* | MATLAB (/wiki/Category:MATLAB)
| Octave (/wiki/Category:Octave) | Maxima (/wiki/Category:Maxima)
| Microsoft Small Basic (/wiki/Category:Microsoft_Small_Basic) | MiniScript (/wiki/Category:MiniScript)
| Min (/wiki/Category:Min) | MK-61/52 (/wiki/Category:%D0%9C%D0%9A-61/52) | ML (/wiki/Category:ML)
| MLite (/wiki/Category:MLite) | Modula-2 (/wiki/Category:Modula-2) | Nanoquery (/wiki/Category:Nanoquery)
| NetRexx (/wiki/Category:NetRexx) | Nim (/wiki/Category:Nim) | Objeck (/wiki/Category:Objeck)
| OCaml (/wiki/Category:OCaml) | Oforth (/wiki/Category:Oforth) | OoRexx (/wiki/Category:OoRexx)
| Order (/wiki/Category:Order) | PARI/GP (/wiki/Category:PARI/GP) | Pascal (/wiki/Category:Pascal)
| Perl (/wiki/Category:Perl) | Phix (/wiki/Category:Phix) | Phixmonti (/wiki/Category:Phixmonti)
| PHP (/wiki/Category:PHP) | PicoLisp (/wiki/Category:PicoLisp) | PL/I (/wiki/Category:PL/I)

PowerShell (/wiki/Category:PowerShell) | Prolog (/wiki/Category:Prolog)
PureBasic (/wiki/Category:PureBasic) | Python (/wiki/Category:Python) | Qi (/wiki/Category:Qi)
Quackery (/wiki/Category:Quackery) | R (/wiki/Category:R) | Racket (/wiki/Category:Racket)
Raku (/wiki/Category:Raku) | Retro (/wiki/Category:Retro) | REXX (/wiki/Category:REXX)
Ring (/wiki/Category:Ring) | Ruby (/wiki/Category:Ruby) | Run BASIC (/wiki/Category:Run_BASIC)
Run BASIC examples needing attention (/wiki/Category:Run_BASIC_examples_needing_attention)
Examples needing attention (/wiki/Category:Examples_needing_attention) | Rust (/wiki/Category:Rust)
Scala (/wiki/Category:Scala) | Scheme (/wiki/Category:Scheme) | Seed7 (/wiki/Category:Seed7)
SenseTalk (/wiki/Category:SenseTalk) | Sidef (/wiki/Category:Sidef) | Smalltalk (/wiki/Category:Smalltalk)
Sparkling (/wiki/Category:Sparkling) | Standard ML (/wiki/Category:Standard_ML)
Swift (/wiki/Category:Swift) | Tcl (/wiki/Category:Tcl) | TI-83 BASIC (/wiki/Category:TI-83_BASIC)
TSE SAL (/wiki/Category:TSE_SAL) | TXR (/wiki/Category:TXR) | UBasic/4tH (/wiki/Category:UBasic/4tH)
UNIX Shell (/wiki/Category:UNIX_Shell) | C Shell (/wiki/Category:C_Shell) | Ursa (/wiki/Category:Ursa)
Vala (/wiki/Category:Vala) | VBA (/wiki/Category:VBA) | VBScript (/wiki/Category:VBScript)
Wortel (/wiki/Category:Wortel) | Wren (/wiki/Category:Wren) | XBasic (/wiki/Category:XBasic)
XPL0 (/wiki/Category:XPL0) | Yabasic (/wiki/Category:Yabasic) | Zkl (/wiki/Category:Zkl)