

```
[> restart;
[> interface(warnlevel=0) : # Maple 12
```

Greatest Common Divisor Worksheet Euclid's Algorithm

The Maple functions `igcd()` and `igcdex()` implement Euclid's algorithm
`igcd()` computes the gcd
`igcdex()` computes the gcd using the Extended Euclidean Algorithm

```
> igcd(1071, 1029);
igcdex(1071, 1029);
```

21
21 (1)

Euclid treated the problem geometrically, using repeated subtractions. Noting that the gcd of two integers also divides the difference between the two integers.
We use Maple's `trace()` function to see how it works.

```
> Egcd0 := proc(a, b)
    local x, y;
    x := a; y := b;
    if x = 0 then return y end if;
    while y <> 0 do
        if x > y then
            x := x - y;
        else
            y := y - x;
        end if;
    end do;
    return x;
end proc;
```

```
> # trace(Egcd0) : # turning on the trace function.
> Egcd0(1071, 1029);
```

21 (2)

```
> Egcd0(1029, 1071);
```

21 (3)

```
> # untrace(Egcd0) : # turning trace off
```

This version is a recursive version of the algorithm using mod arithmetic. Egcd(x,y), where x and y are the integers of interest.

```
> Egcd := proc(x, y)
    if y = 0 then return x
    else
        return Egcd(y, x mod y)
    end if;
end proc :
```

```
> Egcd(1071, 1029);
21 (4)
```

```
> Egcd(1029, 1071);
21 (5)
```

Notice that in this second call to Egcd(x,y) we have reversed the number such that $x < y$ and the proc re-ordered the call such that $x > y$; second line of the trace.

This is the non-recursive version of the procedure

```
> Egcd2 := proc (a, b)
    local temp, x, y;
    x := a : y := b :
    while y <> 0 do
        temp := y;
        y := x mod y;
        x := temp
    end do;
end proc :
```

```
> Egcd2(1071, 1029);
21 (6)
```

```
> Egcd2(1029, 1071);
21 (7)
```

Bézout's identity states that if a and b are nonzero integers whose greatest common divisor is given by gcd(a,b) then there exist integers x and y such that

$$ax + by = \text{gcd}(a,b)$$

the integers x and y can be determined using the Extended Euclidean algorithm. Let a and b be two non-negative where $a \geq b$.

If b equal 0 then set gcd(a,b)=a, x=1, y=0, and Return(gcd(a,b),x,y)

Set $x_2 = 1$, $x_1 = 0$, $y_2 = 0$, $y_1 = 1$.

While $b > 0$ do the following

$q = \text{Integer}(a/b)$, $r = a - q \cdot b$, $x = x_2 - q \cdot x_1$, $y = y_2 - q \cdot y_1$.

Set $a = b$, $b = r$, $x_2 = x_1$, $x_1 = x$, $y_2 = y_1$, $y_1 = y$.

end do

Set gcd(a,b) = a,

Set $x = x_2$ and $y = y_2$

Return(gcd(a,b), x, y).

[illegible]

$$L := [3, -14, 33] \quad (8)$$

$$\begin{aligned} &> \quad 'L[2]\cdot a + L[3]\cdot b'=L[2]\cdot a + L[3]\cdot b; \\ &\qquad\qquad\qquad L_2 a + L_3 b = 3 \end{aligned} \tag{9}$$

$$\begin{aligned} & \textcolor{red}{>} \quad a := 1071 : b := 1029 : \\ & \quad L := \textit{EgcdX}(a, b); \\ & \quad 'L[2] \cdot a + L[3] \cdot b' = L[2] \cdot a + L[3] \cdot b; \\ & \quad \quad \quad \textcolor{blue}{L := [21, -24, 25]} \\ & \quad \quad \quad \textcolor{blue}{L_2 a + L_3 b = 21} \end{aligned} \tag{10}$$

$$\begin{aligned}
> \quad a &:= 2070 : b := 2035 : \\
&L := \text{EgcdX}(a, b); \\
&'L[2] \cdot a + L[3] \cdot b' = L[2] \cdot a + L[3] \cdot b; \\
&\qquad\qquad\qquad L := [5, -58, 59] \\
&\qquad\qquad\qquad L_2 a + L_3 b = 5
\end{aligned} \tag{11}$$

$$\begin{aligned} & \textcolor{red}{>} \quad a := 4864 : b := 3458 : \\ & \quad L := \textit{EgcdX}(a, b); \\ & \quad 'L[2] \cdot a + L[3] \cdot b' = L[2] \cdot a + L[3] \cdot b; \\ & \quad \quad \quad \textcolor{blue}{L := [38, 32, -45]} \\ & \quad \quad \quad \textcolor{blue}{L_2 a + L_3 b = 38} \end{aligned} \tag{12}$$

```
> a := 219 : b := 93 :
'gcd' = igcdex(a, b, 'x', 'y');
'x' = x;
'y' = y;
'a · x + b · y' = a · x + b · y;
```

```
gcd = 3
x = -14
y = 33
a x + b y = 3
```

(13)

```

> a := 1071 : b := 1029 :
'gcd' = igcdex(a, b, 'x', 'y'); 'x' = x; 'y' = y;
'a·x + b·y' = a·x + b·y;

gcd = 21
x = -24
y = 25
a x + b y = 21

```

(14)

```

> a := 2070 : b := 2035 :
'gcd' = igcdex(a, b, 'x', 'y'); 'x' = x; 'y' = y;
'a·x + b·y' = a·x + b·y;

gcd = 5
x = -58
y = 59
a x + b y = 5

```

(15)

```

> a := 4864 : b := 3458 :
'gcd' = igcdex(a, b, 'x', 'y'); 'x' = x; 'y' = y;
'a·x + b·y' = a·x + b·y;

gcd = 38
x = 32
y = -45
a x + b y = 38

```

(16)

The modular inverse of the integer ($p \bmod N$) exists iff $\gcd(p, N) = 1$, and is defined as the value of x such that

$$p \cdot x = 1 \bmod N$$

We can write it as $x = p^{-1} \bmod n$. We can use the Extended Euclidean algorithm to determine the integers x and y in the following equation

$$px + Ny = 1$$

where x is the modular inverse

Notice:

invmod(p, N) returns $[x, \gcd(p, N)]$
returns $x = 0$ when the $\gcd(p, N) \neq 1$

```

> invmod := proc(p, N)
local a, b, x, y, x1, x2, y1, y2, r, q;
a := N; b := p;
x2 := 1; y2 := 0;
x1 := 0; y1 := 1;
while (b ≠ 0) do
q := trunc( (a/b) );
r := a - q·b;
x := x2 - q·x1;
y := y2 - q·y1;
a := b; b := r; x2 := x1; x1 := x; y2 := y1; y1 := y;
end do;
if (a ≠ 1) then return [0, a] end if;
if (y2 < 0) then y2 := N + y2 end if;
return [y2, a];
end proc;

```

```
> invmod(3, 11); #  $y \cdot 3 = 1 \pmod{20}$ 
[4, 1] (17)
```

```
> invmod(3, 20);
[7, 1] (18)
```

```
> invmod(1254761, 2042796);
[824693, 1] (19)
```

```
> invmod(619313, 2042796);
[1090625, 1] (20)
```

```
> invmod(1032299, 1759320);
[72539, 1] (21)
```

```
> invmod(111, 421);
[110, 1] (22)
```

```
> invmod(93, 219);
[0, 3] (23)
```

Using Maple igcdex() function

```
> 'gcd(3, 11)' = igcdex(3, 11, 'x', 'y'); #  $y \cdot 3 = 1 \pmod{20}$ 
'x' = x;
'y' = y;
gcd(3, 11) = 1
x = 4
y = -1 (24)
```

```
> 'gcd(3, 20)' = igcdex(3, 20, 'x', 'y');
'x' = x;
'y' = y;
gcd(3, 20) = 1
x = 7
y = -1 (25)
```

```
> 'gcd(1254761, 2042796)' = igcdex(1254761, 2042796, 'x', 'y');
'x' = x;
'y' = y;
gcd(1254761, 2042796) = 1
x = 824693
y = -506557 (26)
```

This calculation will generate a negative x. We can use $x + N$ to work with a positive number

```
> 'gcd(619313, 2042796)' = igcdex(619313, 2042796, 'x', 'y');
'x' = x;
'y' = y;
'x' = x + 2042796;
gcd(619313, 2042796) = 1
x = -952171
y = 288669
x = 1090625 (27)
```

```

> 'gcd(1032299, 1759320)' = igcdex(1032299, 1759320, 'x','y');
'x' = x;
'y' = y;

gcd(1032299, 1759320) = 1
x = 72539
y = -42563

```

(28)

```

> 'gcd(111, 421)' = igcdex(111, 421, 'x','y');
'x' = x;
'y' = y;

gcd(111, 421) = 1
x = 110
y = -29

```

(29)

There is no mod inverse here because the gcd(93,219) ≠ 1

```

> 'gcd(93, 219)' = igcdex(93, 219, 'x','y');
'x' = x;
'y' = y;
'93 x mod 219' = 33·93 mod 219 ;

gcd(93, 219) = 3
x = 33
y = -14
93 x mod 219 = 3

```

(30)

A variation of the above algorithm is shown below. Knuth; Algorithm X, Vol 2 p 342

Notice:

invmod(x, Φ) returns [y, gcd(x, Φ)]

where y's are positive integers. compare

invmod2(619313, 2042796) & invmod(619313, 2042796)

returns y=0 when the gcd(x,Φ) ≠ 1

```

> invmod2 := proc(x, Φ)
    local i, a1, a2, b1, b2, c1, c2, q;
    a1 := 1; a2 := x;
    b1 := 0; b2 := Φ;
    i := 1;
    while (b2 ≠ 0) do
        q := trunc( a2 / b2 );
        c2 := a2 mod b2;
        c1 := a1 + q · b1;
        a1 := b1; b1 := c1; a2 := b2; b2 := c2;
        i := -i;
    end do;
    if (a2 ≠ 1) then return [0, a2] end if;
    if (i < 0) then a1 := Φ - a1 end if;
    return [a1, a2];
end proc;

```

```

[> # trace(invmod2) : # turning on the trace function.
[> invmod2(3, 11);
[4, 1]
[> # untrace(invmod2) : # turning off the trace function.

```

(31)

```

[> invmod2(3, 20); # y·3 = 1 mod 20
[7, 1]

```

(32)

```

[> invmod2(1254761, 2042796);
[824693, 1]

```

(33)

```

[> invmod2(619313, 2042796);
[1090625, 1]

```

(34)

```

[> invmod2(1032299, 1759320);
[72539, 1]

```

(35)

```

[> invmod2(111, 421);
[110, 1]

```

(36)

```

[> invmod2(93, 219);
[0, 3]

```

(37)

```

[> # trace(invmod) :
[> invmod(619313, 2042796);
[1090625, 1]
[> # untrace(invmod) :

```

(38)

```

[> # trace(invmod2) :
[> invmod2(619313, 2042796);
[1090625, 1]
[> # untrace(invmod2) :

```

(39)