# **Linear Diophantine Equations: ax + by = c — One‑page revision (Extended Euclid)**

**Goal:** Solve the linear Diophantine equation ax + by = c when g = gcd(a,b) divides c. This note gives a compact derivation, a single-step method to find one solution (x1,y1) using the Extended Euclidean Algorithm, the general solution, worked examples, implementation (C++ & Python), and bullet‑point checklists for fast revision.

## **1. Problem statement (single line)**

Find integer solutions (x, y) to ax + by = c where a, b, c are given integers and g = gcd(a, b) divides c (otherwise no integer solution exists).

## **2. Quick answer (the end formula you must remember)**

Let g = gcd(a,b). Suppose (x0, y0) is one pair of integers satisfying a x0 + b y0 = g (found by Extended Euclid). Then scale by c/g to get a particular solution for the target equation:

(x1, y1) = (x0 \* (c/g), y0 \* (c/g))

All integer solutions are then given by the one-parameter family:

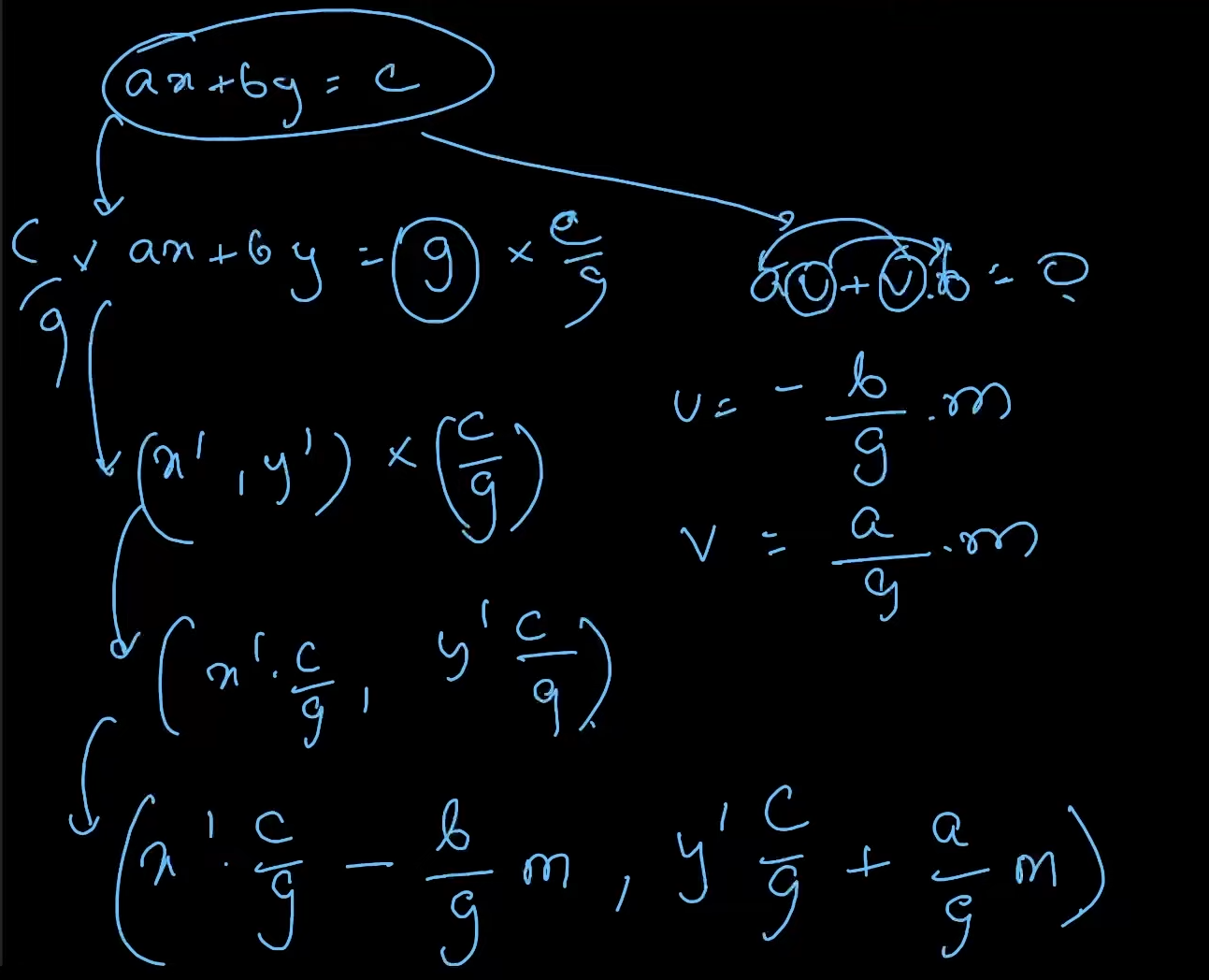
x = x1 - (b/g) \* t

y = y1 + (a/g) \* t

for any integer t.

**3. Why this works (short derivation)**

1. If g = gcd(a,b), then integers x0,y0 exist so that a x0 + b y0 = g (Bezout's identity). Extended Euclid returns such (x0,y0).
2. Multiply both sides by c/g to get a (x0\*c/g) + b (y0\*c/g) = c — so (x1,y1) = (x0\*c/g, y0\*c/g) is a particular solution.
3. Any two solutions (x,y) and (x',y') satisfy a(x-x') + b(y-y') = 0 → a(x-x') = -b(y-y'). So b/g divides (x-x') and a/g divides (y'-y). Parameterizing by integer t gives the general formula above.



## **4. How to find (x0,y0) in practice — Extended Euclidean Algorithm (compact)**

Extended Euclid computes g = gcd(a,b) together with integers (x0,y0) such that a x0 + b y0 = g.

Algorithm (iterative):

1. Start with pairs (r0,r1) = (a,b) and coefficients (s0,s1) = (1,0), (t0,t1) = (0,1).
2. While r1 != 0: perform quotient q = r0 // r1; update (r0,r1) = (r1, r0 - q\*r1) and similarly update s and t: (s0,s1) = (s1, s0 - q\*s1), (t0,t1) = (t1, t0 - q\*t1).
3. When loop ends g = r0, and coefficients (s0,t0) satisfy a\*s0 + b\*t0 = g.

That pair (s0,t0) is the (x0,y0) we need.

## **5. Implementation (copy-paste ready)**

### **C++ (iterative, long long safe for typical contest constraints)**



| #include <bits/stdc++.h> using namespace std;  // returns gcd(a,b), and sets x,y such that a\*x + b\*y = gcd(a,b) long long extgcd(long long a, long long b, long long &x, long long &y) {  x = 1; y = 0;  long long x1 = 0, y1 = 1;  while (b != 0) {  long long q = a / b;  long long r = a - q \* b; a = b; b = r;  long long nx = x - q \* x1; x = x1; x1 = nx;  long long ny = y - q \* y1; y = y1; y1 = ny;  }  return a; // gcd }  // solve ax + by = c ; returns true if solution exists, and sets x0,y0 (one solution) bool solveDiophantine(long long a, long long b, long long c, long long &x0, long long &y0) {  long long x, y;  long long g = extgcd(llabs(a), llabs(b), x, y);  if (c % g != 0) return false;  // adjust signs because we used absolute values  if (a < 0) x = -x;  if (b < 0) y = -y;  long long mul = c / g;  x0 = x \* mul;  y0 = y \* mul;  return true; }  // Example usage int main(){  long long a = 14, b = 21, c = 35; // example  long long x0, y0;  if (solveDiophantine(a,b,c,x0,y0)) {  cout << "One solution: x=" << x0 << " y=" << y0 << '\n';  // general form: x = x0 - (b/g)\*t, y = y0 + (a/g)\*t  } else {  cout << "No integer solution\n";  } } |
| --- |

### 

## **6. Worked example (step-by-step)**

Solve 14x + 21y = 35.

1. g = gcd(14,21) = 7. Since 7 | 35, solutions exist.
2. Extended Euclid on (14,21) gives x0 = -1, y0 = 1 for 14\*(-1) + 21\*1 = 7.
3. Scale by 35/7 = 5: a particular solution is x1 = -5, y1 = 5.
4. General solution: x = -5 - (21/7)t = -5 - 3t, y = 5 + (14/7)t = 5 + 2t, for t in Z.

Check: t=0 → x=-5,y=5 → 14\*(-5)+21\*5 = -70+105 = 35.

## **7. Useful notes & pitfalls (bullet list for fast revision)**

* **Existence:** solution exists iff g = gcd(a,b) divides c.
* **Sign handling:** Extended Euclid usually runs on non-negative inputs; if you change sign, remember to fix (x,y) signs accordingly.
* **Scaling:** Do *not* forget to multiply the Bezout coefficients by c/g.
* **General solution:** step size for x is b/g, for y is a/g (note signs in formula).
* **Minimal positive solution:** to find minimal non-negative x, choose t such that x is in desired range via modular arithmetic: reduce x1 mod (b/g).
* **Overflow:** use 64-bit integers (long long) in contests if a,b,c can be up to 1e12.
* **Multiple gcd forms:** if extgcd(a,b) returns g,x,y with original signs, you can directly use x\*(c/g).

## **8. Quick checklist to solve any ax+by=c (one-pass)**

1. Compute g = gcd(a,b).
2. If c%g != 0 → NO SOLUTION.
3. Run Extended Euclid to get (x0,y0) with a\*x0 + b\*y0 = g.
4. Set x1 = x0 \* (c/g), y1 = y0 \* (c/g).
5. General solution: x = x1 - (b/g)\*t, y = y1 + (a/g)\*t for t in Z.
6. If you want solutions within ranges, use t and integer division/mod operations to pick t.

## **9. One-line memory aid**

*Extended Euclid gives the Bezout pair for g = gcd(a,b) → scale by c/g → shift by multiples of (b/g, a/g).*

* add a short section about how to find **non-negative** solutions or smallest positive x.