

```

/*
 * gcd.c
 *
 * This file contains the kernel functions
 *
 *     long int gcd(long int a, long int b);
 *     long int euclidean_algorithm(long int m, long int n, long int *a, long int *b);
 *     long int Zq_inverse(long int p, long int q);
 *
 * gcd() returns the greatest common divisor of two long integers,
 * at least one of which is nonzero.
 *
 * euclidean_algorithm() returns the greatest common divisor of two long
 * integers m and n, and also finds long integers a and b such that
 *  $am + bn = \text{gcd}(m,n)$ . The integers m and n may be negative, but cannot
 * both be zero.
 *
 * Zq_inverse() returns the inverse of p in the ring  $\mathbb{Z}/q$ . Assumes p and q
 * are relatively prime integers satisfying  $0 < p < q$ .
 *
 * 97/3/31 gcd() modified to accept negative integers.
 */

```

```

#include "kernel.h"

```

```

long int gcd(
    long int    a,
    long int    b)
{
    a = ABS(a);
    b = ABS(b);

    if (a == 0)
    {
        if (b == 0)
            uFatalError("gcd", "gcd");
        else
            return b;
    }

    while (TRUE)
    {
        if ((b = b%a) == 0)
            return a;
        if ((a = a%b) == 0)
            return b;
    }
}

```

```

long int euclidean_algorithm(
    long int    m,
    long int    n,
    long int    *a,
    long int    *b)
{
    /*
     * Given two long integers m and n, use the Euclidean algorithm to
     * find integers a and b such that  $a*m + b*n = \text{g.c.d.}(m,n)$ .
     *
     * Recall the Euclidean algorithm is to keep subtracting the
     * smaller of {m, n} from the larger until one of them reaches
     * zero. At that point the other will equal the g.c.d.
     *
     * As the algorithm progresses, we'll use the coefficients
     * mm, mn, nm, and nn to express the current values of m and n
     * in terms of the original values:
     *
     *     current m = mm*(original m) + mn*(original n)
     *     current n = nm*(original m) + nn*(original n)
     */

    long int    mm,
                mn,

```

```
        nm,
        nn,
        quotient;

/*
 * Begin with a quick error check.
 */

if (m == 0 && n == 0)
    uFatalError("euclidean_algorithm", "gcd");

/*
 * Initially we have
 *
 *      current m = 1 (original m) + 0 (original n)
 *      current n = 0 (original m) + 1 (original n)
 */

mm = nn = 1;
mn = nm = 0;

/*
 * It will be convenient to work with nonnegative m and n.
 */

if (m < 0)
{
    m = - m;
    mm = -1;
}

if (n < 0)
{
    n = - n;
    nn = -1;
}

while (TRUE)
{
    /*
     * If m is zero, then n is the g.c.d. and we're done.
     */
    if (m == 0)
    {
        *a = nm;
        *b = nn;
        return n;
    }

    /*
     * Let n = n % m, and adjust the coefficients nm and nn accordingly.
     */
    quotient = n / m;
    nm -= quotient * mm;
    nn -= quotient * mn;
    n -= quotient * m;

    /*
     * If n is zero, then m is the g.c.d. and we're done.
     */
    if (n == 0)
    {
        *a = mm;
        *b = mn;
        return m;
    }

    /*
     * Let m = m % n, and adjust the coefficients mm and mn accordingly.
     */
    quotient = m / n;
    mm -= quotient * nm;
    mn -= quotient * nn;
    m -= quotient * n;
```

```

    }

    /*
     * We never reach this point.
     */
}

long int Zq_inverse(
    long int    p,
    long int    q)
{
    long int    a,
               b,
               g;

    /*
     * Make sure 0 < p < q.
     */

    if (p <= 0 || p >= q)
        uFatalError("Zq_inverse", "gcd");

    /*
     * Find a and b such that ap + bq = gcd(p,q) = 1.
     */

    g = euclidean_algorithm(p, q, &a, &b);

    /*
     * Check that p and q are relatively prime.
     */

    if (g != 1)
        uFatalError("Zq_inverse", "gcd");

    /*
     *      ap + bq = 1
     * =>  ap = 1 (mod q)
     * =>  The inverse of p in Z/q is a.
     *
     * Normalize a to the range (0, q).
     *
     * [My guess is that a must always fall in the range -q < a < q,
     * in which case the following code would simplify to
     *
     *      if (a < 0)
     *          a += q;
     *
     * but I haven't worked out a proof.]
     */

    while (a < 0)
        a += q;
    while (a > q)
        a -= q;

    return a;
}

```