```c
/*****************************************************************************
*   FILE: keyEligible.c
*   AUTHOR: Connor Beardsmore - 15504319
*   UNIT: FCC200
*   PURPOSE: Check the eligibility of keys a and b
*   LAST MOD: 28/03/17
*   REQUIRES: keyEligible.h
*****************************************************************************/

#include "keyEligible.h"

//----------------------------------------------------------------------------
// FUNCTION: keyEligible
// IMPORT: a (int), b (int)
// EXPORT: eligible (int)
// PURPOSE: Check that the two given keys are eligible via coprime check

int keyEligible( int a, int b, int alphabet )
{
    int eligible = 1;

    // a must be positive and less than the alpabet (26)
    if ( ( a < 0 ) || ( b > ( alphabet - 1 ) ) )
        eligible = 0;
    // a must be coprime to the alphabet length (26)
    if ( gcdFunction( a, alphabet ) != 1 )
        eligible = 0;
    // b must be positive and less than the alphabet (26)
    if ( ( b < 0 ) || ( b > ( alphabet - 1 ) ) )
        eligible = 0;
    return eligible;
}

//----------------------------------------------------------------------------
// FUNCTION: gcd
// IMPORT: a (int), b (int)
// PURPOSE: Find greatest common denominator of 2 numbers

int gcdFunction( int a, int b )
{
    int quotient, residue, temp, gcd = 1;
    // SWAP ELEMENTS TO GET THE MAX
    if ( a < b )
    {
        temp = a;
        a = b;a
        b = temp;
    }
    // CHECK IF EITHER NUMBER IS 0
    if ( a == 0 )   return b;
    if ( b == 0 )   return a;
    // SATISFY THE EQUATION: A = B * quotient + residue
    quotient = a / b;
    residue = a - ( b * quotient );
    // RECURSIVELY CALL GCD
    gcd = gcdFunction( b, residue );
    return gcd;
}

//----------------------------------------------------------------------------
// FUNCTION: extendEuclid
// IMPORT: a (int), n (int)
// PURPOSE: Extended Euclidean algorithm to find inverse modular

int extendEuclid( int a, int n )
{
    int t = 0, newt = 1;
    int r = n, newr = a;
    int q = 0, temp = 0;

    // IF GCD IS NOT 1 THEN NO COPRIME EXISTS
    if ( gcdFunction( a, n ) != 1 )
```

```
73          return -1;
74
75      // PERFORM EXTENDED EUCLIDEAN
76      while ( newr != 0 )
77      {
78          q = r / newr;
79          temp = t;
80          t = newt;
81          newt = temp - ( q * newt );
82          temp = r;
83          r = newr;
84          newr = temp - ( q * newr );
85      }
86
87      // MAKE SURE T IS NOT NEGATIVE
88      if ( t < 0 )
89          t = t + n;
90
91      return t;
92  }
93
94  //-------------------------------------------------------------------------
95
```