```
// BigInteger by Jane Alam Jan

struct Bigint {
   string a;                  // to store the digits
   int sign;                  // sign = -1 for negative numbers, sign = 1 otherwise

   Bigint() {}                // default constructor
   Bigint( string b ) { (*this) = b; }          // constructor for string

   int size() {               // returns number of digits
      return a.size();
   }

   Bigint inverseSign() {     // changes the sign
      sign *= -1;
      return (*this);
   }

   // --------- leading zero remover
   Bigint normalize( int newSign ) {               // removes leading 0, fixes sign
      for( int i = a.size() - 1; i > 0 && a[i] == '0'; i-- )
         a.erase(a.begin() + i);
      sign = ( a.size() == 1 && a[0] == '0' ) ? 1 : newSign;
      return (*this);
   }

   //--------- assignment operator
   void operator = ( string b ) {                  // assigns a string to Bigint
      a = b[0] == '-' ? b.substr(1) : b;
      reverse( a.begin(), a.end() );
      this->normalize( b[0] == '-' ? -1 : 1 );
   }

   //---------- conditional operators
   bool operator < ( const Bigint &b ) const {     // less than operator
      if( sign != b.sign ) return sign < b.sign;
      if( a.size() != b.a.size() )
         return sign == 1 ? a.size() < b.a.size() : a.size() > b.a.size();
      for( int i = a.size() - 1; i >= 0; i-- ) if( a[i] != b.a[i] )
         return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
      return false;
   }

   bool operator == ( const Bigint &b ) const {          // operator for equality
      return a == b.a && sign == b.sign;
   }

   //---------- mathematical operators
   Bigint operator + ( Bigint b ) {                    // addition operator overloading
      if( sign != b.sign ) return (*this) - b.inverseSign();
      Bigint c;
      for(int i = 0, carry = 0; i<a.size() || i<b.size() || carry; i++ ) {
         carry+=(i<a.size() ? a[i]-48 : 0)+(i<b.a.size() ? b.a[i]-48 : 0);
         c.a += (carry % 10 + 48);
         carry /= 10;
      }
      return c.normalize(sign);
   }
   Bigint operator - ( Bigint b ) {                    // subtraction operator overloading
```

```
        if( sign != b.sign ) return (*this) + b.inverseSign();
        int s = sign; sign = b.sign = 1;
        if( (*this) < b ) return ((b - (*this)).inverseSign()).normalize(-s);
        Bigint c;
        for( int i = 0, borrow = 0; i < a.size(); i++ ) {
            borrow = a[i] - borrow - (i < b.size() ? b.a[i] : 48);
            c.a += borrow >= 0 ? borrow + 48 : borrow + 58;
            borrow = borrow >= 0 ? 0 : 1;
        }
        return c.normalize(s);
    }
    Bigint operator * ( Bigint b ) {                    // multiplication operator overloading
        Bigint c("0");
        for( int i = 0, k = a[i] - 48; i < a.size(); i++, k = a[i] - 48 ) {
            while(k--) c = c + b;                       // ith digit is k, so, we add k times
            b.a.insert(b.a.begin(), '0');               // multiplied by 10
        }
        return c.normalize(sign * b.sign);
    }
    Bigint operator / ( Bigint b ) {                    // division operator overloading
        if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
        Bigint c("0"), d;
        for( int j = 0; j < a.size(); j++ ) d.a += "0";
        int dSign = sign * b.sign; b.sign = 1;
        for( int i = a.size() - 1; i >= 0; i-- ) {
            c.a.insert( c.a.begin(), '0');
            c = c + a.substr( i, 1 );
            while( !( c < b ) ) c = c - b, d.a[i]++;
        }
        return d.normalize(dSign);
    }
    Bigint operator % ( Bigint b ) { // modulo operator overloading
        if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0] - 48 );
        Bigint c("0");
        b.sign = 1;
        for( int i = a.size() - 1; i >= 0; i-- ) {
            c.a.insert( c.a.begin(), '0');
            c = c + a.substr( i, 1 );
            while( !( c < b ) ) c = c - b;
        }
        return c.normalize(sign);
    }

    //----------------output method
    void print() {
        if( sign == -1 ) putchar('-');
        for( int i = a.size() - 1; i >= 0; i-- ) putchar(a[i]);
    }
};



int main() {
    Bigint a, b, c;         // declared some Bigint variables
    string input;           // string to take input
    cin >> input;           // take the Big integer as string
    a = input;              // assign the string to Bigint a

    cin >> input;           // take the Big integer as string
```

```
    b = input;              // assign the string to Bigint b

    // Using mathematical operators
    c = a + b;              // adding a and b
    c.print();              // printing the Bigint
    puts("");               // newline

    c = a - b;              // subtracting b from a
    c.print();              // printing the Bigint
    puts("");               // newline

    c = a * b;              // multiplying a and b
    c.print();              // printing the Bigint
    puts("");               // newline

    c = a / b;              // dividing a by b
    c.print();              // printing the Bigint
    puts("");               // newline

    c = a % b;              // a modulo b
    c.print();              // printing the Bigint

    puts("");               // newline

    // Using conditional operators
    if( a == b )            // checking equality
      puts("equal");
    else
      puts("not equal");

    if( a < b )             // checking less than operator
      puts("a is smaller than b");
    return 0;
}


#include <bits/stdc++.h>
using namespace std;
#define MAX                 1e6
#define EPS                 1e-9
#define INF                 1e9+10
#define MOD                 1000000007
#define pb                  push_back
#define mp                  make_pair
#define fi                  first
#define se                  second
#define pi                  acos(-1)
#define sf                  scanf
#define pf                  printf
#define SIZE(a)             ((int)a.size())
#define Equal(a, b)         (abs(a-b) < EPS)
#define Greater(a, b)       (a >= (b+EPS))
#define GreaterEqual(a, b)  (a > (b-EPS))
#define fr(i, a, b)         for(register int i = (a); i < (int)(b); i++)
#define FastRead            ios_base::sync_with_stdio(false); cin.tie(NULL);
#define dbug(vari)          cerr << #vari << " = " << (vari) <<endl
#define isOn(S, j)          (S & (1 << j))
#define setBit(S, j)        (S |= (1 << j))
#define clearBit(S, j)      (S &= ~(1 << j))
```

```
#define toggleBit(S, j)     (S ^= (1 << j))
#define lowBit(S)           (S & (-S))
#define setAll(S, n)        (S = (1 << n) - 1)
#define fileRead(S)         freopen(S, "r", stdin);
#define fileWrite(S)        freopen(S, "w", stdout);
#define Unique(X)           X.erase(unique(X.begin(), X.end()), X.end())

typedef unsigned long long ull;
typedef long long ll;
typedef map<int, int> mii;
typedef map<ll, ll>mll;
typedef map<string, int> msi;
typedef vector<int> vi;
typedef vector<long long>vl;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
typedef vector<pair<int, int> > vii;
typedef vector<pair<ll, ll> >vll;

vi DecimalVal(int a, int b) {           // Calculate Decimal values (after .) of a/b
   vi v;
   a %= b;
   if(a == 0) {
      v.pb(0);
      return v;
   }
   bool first = 1;
   while(SIZE(v) <= 200) {              // Define the Maximum Length of decimal values
      if(a == 0)
         return v;                      // If any Zero divisor is found (then, rest all will be Zero) return values
      else if(a < b && !first) {        // If we need to add another zero (add zero after first time)
         a*=10;
         v.pb(0);
      }
      else if(a < b && first) {         // If we need to add a extra zero (adding zero first time)
         first = 0;
         a *= 10;
         continue;
      }
      else {
         v.pb(a/b);
         a%=b;
         first = 1;
      }
   }
   return v;
}

// Repetation (PunoPonik) is also calculated
vi dec1, dec2;                          // Before . (decimal), after . (decimal)
int DecimalRepeated(int a, int b) {     // Calculate Decimal values (after .) of a/b
   unordered_map<int, int>mp;
   int k = 0, point = -1;
   bool divisable = 0;

   if(a >= b) {                         // Before Decimal Calculation
      dec1.push_back(a/b);
      a %= b;
   }
```

```
    if(dec1.size() == 0)
        dec1.push_back(0);

    while(a != 0) {
        if(mp.find(a) != mp.end()) {        // if the remainder is found again, there exists a loop
            point = mp[a];
            break;
        }
        if(a%b == 0) {
            dec2.push_back(a/b);
            break;
        }
        mp[a] = k++;
        int cnt = 0;
        while(a < b) {
            a *= 10;
            if(cnt != 0) {
                dec2.push_back(0);
                k++;
            }
            ++cnt;
        }
        if(cnt != 0 && mp.find(a) != mp.end()) {
            point = mp[a];
            break;
        }
        if(cnt == 1)
            mp[a] = (k-1);
        dec2.push_back(a/b);
        a %= b;
        if(a == 0) {
            divisable = 1;
            break;
        }
    }
    return divisable == 1 ? 1:((int)dec2.size()-point);
}

int main() {
    int a, b;
    cin >> a >> b;
    vi v = DecimalVal(a, b);                    // only the fraction part
    for(auto it : v)
        cout << it;
    cout << endl;

    int Cycle = DecimalRepeated(a, b);
    for(auto it : dec1)
        cout << it;
    cout << ".";
    for(auto it : dec2)
        cout << it;
    cout << "\n\n";
    cout << "Last Repeating Cycle " << Cycle << endl;
    return 0;
}

struct fraction {
    int a, b;
```

```
  fraction() {
    a = 1;
    b = 1;
  }
  fraction(int x, int y) : a(x), b(y) {}
  flip() {swap(a, b);}
  fraction operator + (fraction other) {
    fraction temp;
    temp.b = ((b)*(other.b))/(__gcd((b), other.b));
    temp.a = (temp.b/b)*a + (temp.b/other.b)*other.a;
    int x = __gcd(temp.a, temp.b);
    if(x != 1) {temp.a/=x; temp.b/=x;}
    return temp;
  }
  fraction operator - (fraction other) {
    fraction temp;
    temp.b = (b*other.b)/__gcd(b, other.b);
    temp.a = (temp.b/b)*a - (temp.b/other.b)*other.a;
    int x = __gcd(temp.a, temp.b);
    if(x != 1) {temp.a/=x; temp.b/=x;}
    return temp;
  }
  fraction operator / (fraction other) {
    fraction temp;
    temp.a = a*other.b;
    temp.b = b*other.a;
    int x = __gcd(temp.a, temp.b);
    if(x != 1) {temp.a/=x; temp.b/=x;}
    return temp;
  }
  fraction operator * (fraction other) {
    fraction temp;
    temp.a = a*other.a;
    temp.b = b*other.b;
    int x = __gcd(temp.a, temp.b);
    if(x != 1) {temp.a/=x; temp.b/=x;}
    return temp;
  }
};
```