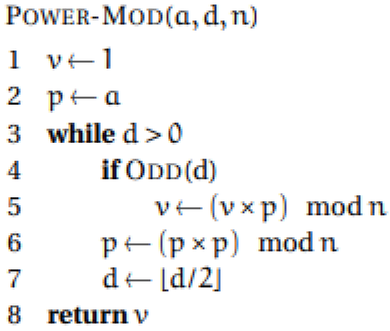
Randstate.c

```
Void randstate_init(uint64_t seed) {
        Use gmp_randinit_mt to set the extern variable state to the Mersenne Twister algorithm
        Use gmp_randseed_ui to set the seed to random
}
```

```
Void randstate_clear(void) {
        Call to gmp_randclear() to clear the state.
}
```

Numtheory.c

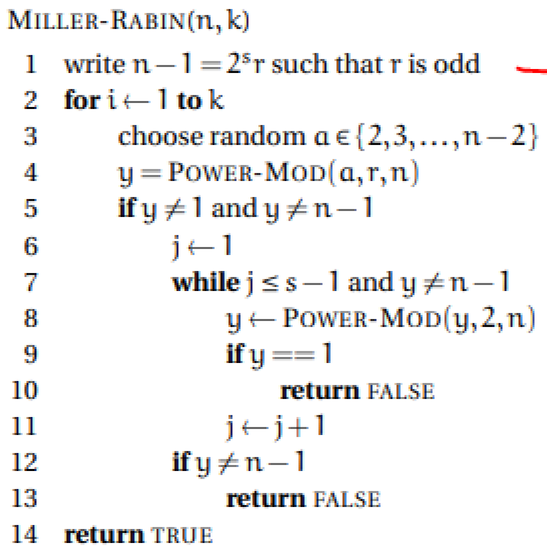
```
//Pseudo code provided in asgn pdf
void pow_mod(mpz_t o, mpz_t a, mpz_t d, mpz_t n) {
```



```
POWER-MOD(a, d, n)
  v ← 1
  p ← a
  while d > 0
      if ODD(d)
          v ← (v × p)  mod n
      p ← (p × p)  mod n
      d ← ⌊d/2⌋
  return v
```

```
}
```

```
Bool isPrime(n, iters)
//Uses Miller-Rabin method to see if n is prime or not
//Write n - 1 = 2 ^ s * r such that r is odd
//Loop, start with s = 0 and r = n-1
while(r is even) {
        Increment s by 1
        Divide r by 2 (This should run until r is odd)
}
```

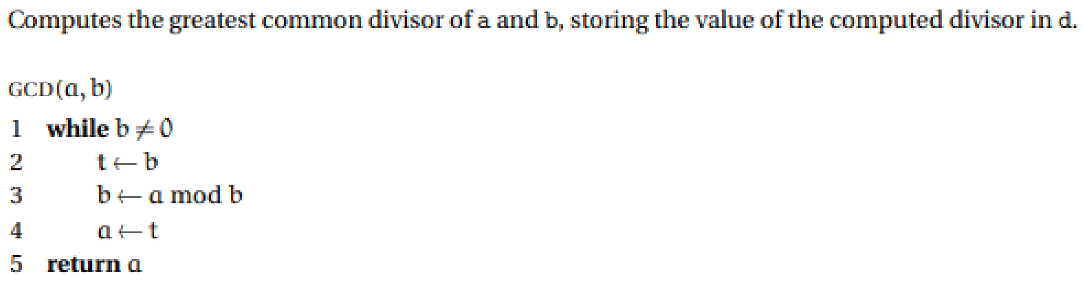


```
MILLER-RABIN($n, k$)
write $n-1 = 2^s r$ such that $r$ is odd
for $i \leftarrow 1$ to $k$
    choose random $a \in \{2,3,\ldots,n-2\}$
    $y = $ POWER-MOD($a, r, n$)
    if $y \neq 1$ and $y \neq n-1$
        $j \leftarrow 1$
        while $j \leq s-1$ and $y \neq n-1$
            $y \leftarrow$ POWER-MOD($y, 2, n$)
            if $y == 1$
                return FALSE
            $j \leftarrow j+1$
        if $y \neq n-1$
            return FALSE
return TRUE
```

Void make_prime(mpz_t p, uint64_t bits, uint64_t iters)

Randomly generates numbers that should be *bits* long, and then makes sures the generated number is prime using *isPrime()* function.

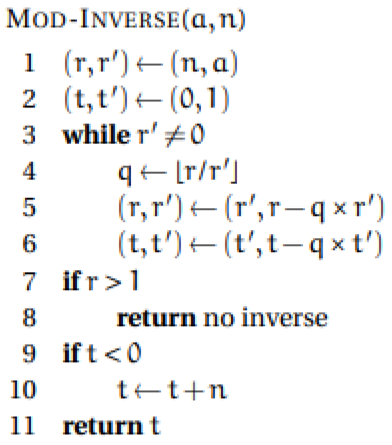
Void gcd(mpz_t d, mpz_t a, mpz_t b)



Computes the greatest common divisor of a and b, storing the value of the computed divisor in d.

```
GCD($a, b$)
while $b \neq 0$
    $t \leftarrow b$
    $b \leftarrow a \bmod b$
    $a \leftarrow t$
return $a$
```

void mod_inverse(mpz_t i, mpz_t a, mpz_t n)



```
MOD-INVERSE(a, n)
$(r, r') \leftarrow (n, a)$
$(t, t') \leftarrow (0, 1)$
while $r' \neq 0$
    $q \leftarrow \lfloor r/r' \rfloor$
    $(r, r') \leftarrow (r', r - q \times r')$
    $(t, t') \leftarrow (t', t - q \times t')$
if $r > 1$
    return no inverse
if $t < 0$
    $t \leftarrow t + n$
return t
```

RSA

rsa_make_pub()

Use make prime to generate p and q.

To get number of bits to generate p, get a random num from $[\mathtt{nbits}/4, (3 \times \mathtt{nbits})/4)$ and q_bits is nbits - pbits

N is p * q

Calculate totient of n by using $(p-1)(q-1)$.

To find e, generate random numbers using mpz_urandomb, and when it has a greatest common divisor with the totient, stop loop.

rsa_write_pub()

Use gmp_fprintf to write n, e, s, and the username to the outfile

(n, e, s, should be in hexstrings)

rsa_read_pub()

Same as write_pub, but now use gmp_fscanf to scan in from file

rsa_make_priv()

Find d, the private k, using the mod_inverse function we made using the e modulo totient of n= (q-1)(p-1)

rsa_write_priv()

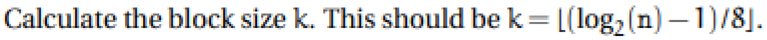
Write out n and d like in write pub

rsa_read_priv()

```
    Read in n and d, like in read_pub

rsa_encrypt()
    Perform pow mod using m, e, n and set c equal to the result

rsa_encrypt_file()
```

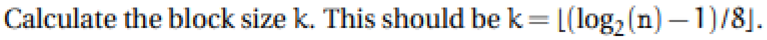


Calculate the block size k. This should be $k = \lfloor (\log_2(n) - 1)/8 \rfloor$.

```
    Allocate an uint8_t array of size k
    Set the 0th index of the array to 0xff
    While there are still bytes in file
        Use fread to get the number of bytes read
        Mpz_import them to convert into an mpz_t int. Make sure 1 for most significant word first, 1 for the endian parameter, and 0 for the nails parameter
        Encrypt the file using rsa_encrypt
        Print the ciphertext, c, to the outfile

Rsa_decrypt()
    Use pow_mod and c as base, d as exponent, and n as modulo, set results to m

Rsa_decrypt_file()
```



Calculate the block size k. This should be $k = \lfloor (\log_2(n) - 1)/8 \rfloor$.

```
    Allocate an uint8_t array of size k
    While there are still bytes in file
        Use fread to get the number of bytes read
        rsa_Decrypt()
        Mpz_export
        And then write to the outfile using fwrite

Rsa_sign()
    Pow_mod of m, d, n, result = x

rsa_verify()
    Pow_mod of s, e, n result = to a mpz
    If that mpz == m
        Then return true
    Else
        false
```

Keygen:

-b : specifies the minimum bits needed for the public modulus n.

-i : specifies the number of Miller-Rabin iterations for testing primes (default: 50).

-n pbfile : specifies the public key file (default: rsa.pub).

-d pvfile : specifies the private key file (default: rsa.priv).

-s : specifies the random seed for the random state initialization (default: the seconds since the UNIX epoch, given by time(NULL)).

-v : enables verbose output.

-h : displays program synopsis and usage.

```
Parse command line options
Open the public and private key files using fopen().
Use fchmod() and fileno(), make sure that the private key file permissions are set to 0600
Initialize the random state using randstate_init() using the seed
Use rsa_make_pub() and rsa_make_priv() to make public and private keys
Convert the username into an mpz_t with mpz_set_str(), specifying the base as 62. Then, use rsa_sign() to compute the signature of the username
Write the public and private keys to the files specified
If verbose
    Print the stats of username, s, p, q, n, e, d
Close files and randstate_clear()
```

Encrypt:

-i : specifies the input file to encrypt (default: stdin).

-o : specifies the output file to encrypt (default: stdout).

-n : specifies the file containing the public key (default: rsa.pub).

-v : enables verbose output.

-h : displays program synopsis and usage.

```
Parse command line options using getopt()
Open public key file using fopen
Read the public key
If verbose:
    Print username
    Print n
    Print e
Convert username to mpz and verify it using rsa_verify
Then use rsa_encrypt_file to encrypt
Close files and clear mpzs
```

Decrypt:

-i : specifies the input file to decrypt (default: stdin).

-o : specifies the output file to encrypt (default: stdout).

-n : specifies the file containing the private key (default: rsa.priv).

-v : enables verbose output.

-h : displays program synopsis and usage.

```
Parse command line options using getopt()
Open private key file using fopen
Read private key
```

```
If Verbose is true
    Print n
    Print e
Decrypt file using rsa_decrypt_file
Close the private key file and clear any mpz_t variables you have used.
```