1. **digit\_counter.m**

function n = digit\_counter(fname)

n = -1;

fid = fopen(fname,'r');

if fid >= 0

n = sum(isstrprop(fread(fid,inf,'char=>char'),'digit'));

fclose(fid);

end

end

1. **day\_counter.m**

function n = day\_counter(year)

months = [31 28 31 30 31 30 31 31 30 31 30 31]; % dyas of the months

start = 1776;

y = year - start; % years since 1776

% number of days since January 1, 1776:

ndays = y\*365 + ceil(y/4) - floor((year-1)/100) + floor(start/100) + floor((year-1)/2000);

% if leap year, need to adjust February:

if mod(year,4) == 0 && (mod(year,100) ~= 0 || mod(year,400) == 0)

months(2) = 29;

end

months = [0 cumsum(months(1:end-1))]; % number of days in the previous months

n = sum(mod(ndays+months,7) == 0); % mod() == 0 means Monday

end

1. **huge\_add.m**

function summa = huge\_add(a,b)

if ~ischar(a) || ~ischar(b) || sum(isstrprop(a,'digit')) ~= length(a) || ...

sum(isstrprop(b,'digit')) ~= length(b)

summa = -1;

return;

end

lng = max([length(a) length(b)]);

a = [a(end:-1:1) '0'+zeros(1,lng-length(a))]; % flip and pad with zeros if necessary

b = [b(end:-1:1) '0'+zeros(1,lng-length(b))]; % flip and pad with zeros if necessary

carry = 0;

for ii = 1:lng

c = carry + str2num(a(ii)) + str2num(b(ii)) % add the digits plus the caryy

carry = c > 9; % is there carry? (0 or 1)

summa(ii) = num2str(mod(c,10)); % put the character in the result

end

if carry

summa(end+1) = '1'; % need a leading 1 if carry

end

summa = summa(end:-1:1); % flip it back

end

1. **smallest\_multiple.m**

function mul = smallest\_multiple(n)

facts = zeros(1,n); % store the exponents of various factors

for ii = 2:n

f = factor(ii); % get factors for current integer

for jj = 2:ii

k = sum(f == jj); % what's the exponent of this factor?

if k > facts(jj) % if it is greater than what we have so far

facts(jj) = k; % update to this new value

end

end

end

% Compute the result with one command.

% The 'native' option tells MATLAB to work in uint64

mul = prod(uint64((1:n).^facts),'native');

if mul == intmax('uint64')

mul = uint64(0);

end

end

OR

function r = smallest\_multiple(k)

r = uint64(1);

for n = 1:k

r = r \* (n / gcd(r,n)); % Include factor not already present in r

end

if r == intmax('uint64')

r = uint64(0);

end

end

1. **max\_product.m**

function B = maxproduct(A,n)

% After checking that we do not have to return an empty array, we initialize a row vector

% for remembering a product, home row and column, and one of four direction codes.

% Home is a location in the matrix from which we can move in one of the directions

% (1=East, 2=South, 3=SouthEast, 4=SouthWest). For each direction, home is systematically

% moved to everywhere it legally can be the start of n values. For each location, we

% potentially update the L vector with the largest product so far, the home location and

% direction that corresponded to that product. After scanning in all four directions, we

% use L to reconstruct the required indices based on the last updated home and direction.

[r,c] = size(A);

if n>r && n>c

B = []; % cannot be solved

return

end

L = [-Inf,0,0,0]; % [product, home-row, home-col, direction]

for i=1:r

for j=1:c-n+1

L = check(A(i,j:j+n-1),[i,j,1],L); % row, right case

end

end

for i=1:r-n+1

for j=1:c

L = check(A(i:i+n-1,j),[i,j,2],L); % column, down case

end

end

for i=1:r-n+1

for j=1:c-n+1

S=A(i:i+n-1,j:j+n-1);

L = check(diag(S),[i,j,3],L); % diagonal, down case

L = check(diag(flip(S,2)),[i,j,4],L); % reverse diagonal, down case

end

end

i=L(2); j=L(3); % reconstruct coordinates

switch L(4)

case 1, B = [ones(n,1)\*i,(j:j+n-1)'];

case 2, B = [(i:i+n-1)',ones(n,1)\*j];

case 3, B = [(i:i+n-1)',(j:j+n-1)'];

case 4, B = [(i:i+n-1)',(j+n-1:-1:j)'];

end

end

function L = check(V,d,L)

p = prod(V);

if p>L(1) % if new product larger than any previous

L = [p,d]; % then update product, home and direction

end

end

1. **number2letters.m**

function m = number2letters (n)

% A is a 3x10 array of the letter counts of the words used to form various numbers, with

% units in row 1, "teens" in row 2 and tens in row 3. For the supplied number, we first

% determine the number of units, tens and hundreds. These three values determine the

% words and thus the letter count to be returned.

A = [ 0 3 3 5 4 4 3 5 5 4; ... % units

3 6 6 8 8 7 7 9 8 8; ... % "teens"

0 0 6 6 5 5 5 7 6 6]; % tens

for i = 1:n

h = fix(i/100);

t = fix(rem(i,100)/10);

u = rem(i,10);

if h>0, m = A(1,h+1)+7; % h 'hundred'

else m = 0;

end

switch t

case 0, m = m+A(1,u+1); % units only

case 1, m = m+A(2,u+1); % "teens" only

otherwise m = m+A(3,t+1)+A(1,u+1); % tens and units

end

end

end

1. **circular\_primes.m**

function n = circular\_primes(k)

n = 0;

for p = primes(k-1)

if circular\_prime(p)

n = n + 1;

end

end

end

function yes = circular\_prime(k)

d = num2str(k);

for ii = 1:length(d)

d = circshift(d,[0 1]); % try next circular shifted version

if ~isprime(str2num(d))

yes = false; % not circular prime

return; % so we can return

end

end

yes = true; % if we made it here, it is a circular prime

end

1. **cyclotron.m**

function [E,n] = cyclotron (V)

m = 3.344e-27; % mass of deuteron [kg]

q = 1.603e-19; % charge of deuteron [c]

B = 1.600; % magnetic field strength [t]

z = m\*V/(q\*B^2); % initial trajectory radius squared [m^2]

n = 0; % pass number

d = 2; % diameter/radius, sign alternates

r = sqrt(z); % initial radius of curvature

x = -r/2.0; % initial x-coordinate

while x>=-0.5 % while deuteron has not emerged

x = x+d\*r; % spiral to next x-coordinate ;

n = n+1; % update pass number

d = -d; % ... and direction

r = sqrt(r^2+2\*z); % calculate new radius in case needed

end

E = V\*n\*1e-6; % determine energy

end