



COMPUTER SKILLS

LESSON 10

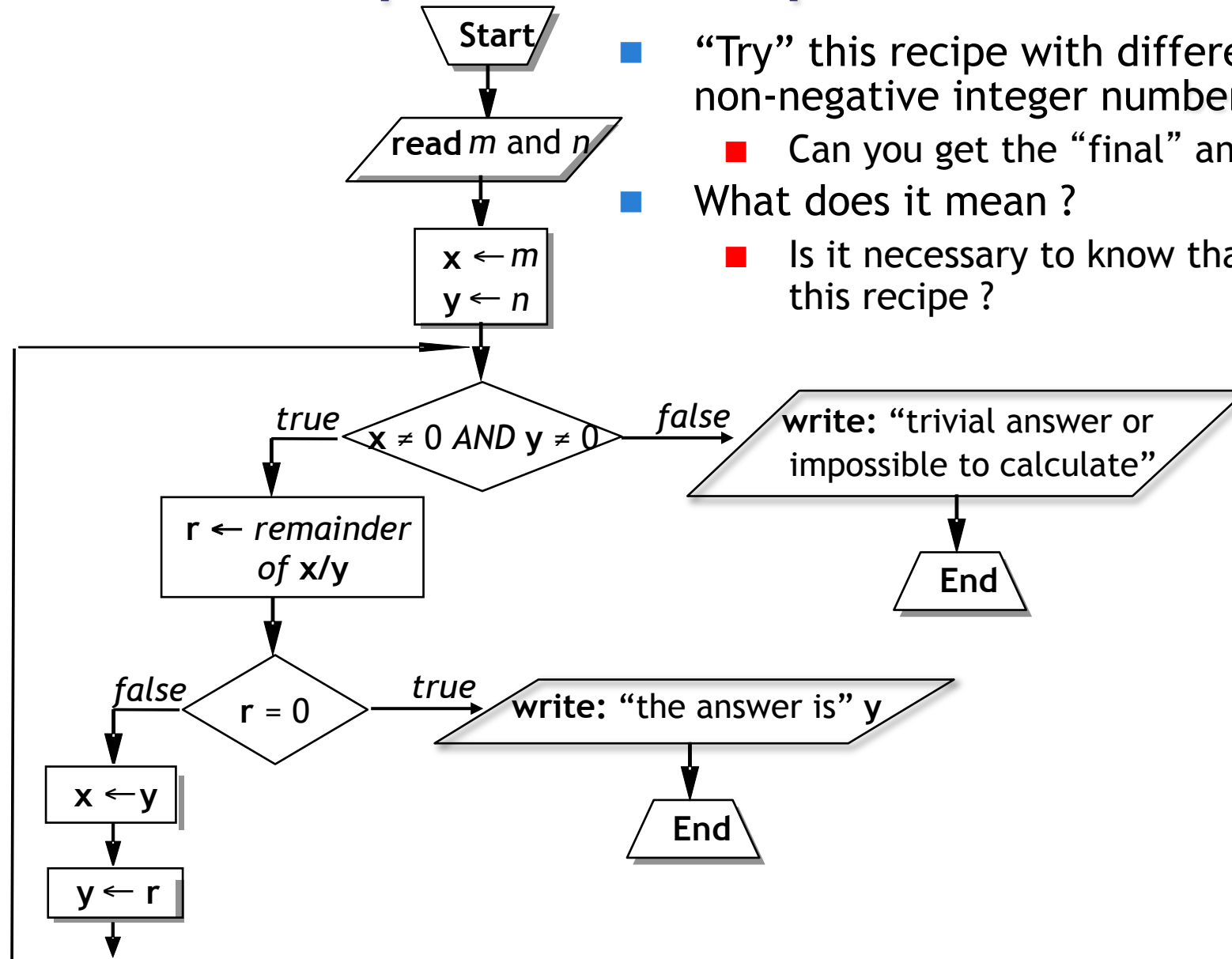
Valeria Cardellini
cardellini@ing.uniroma2.it
A.Y. 2015/16

Objectives of this lesson

We'll discuss

- Coding the GCD flow diagram
- More examples on loop statements
- Simple plots
- Basic file input/output

An example of “recipe”

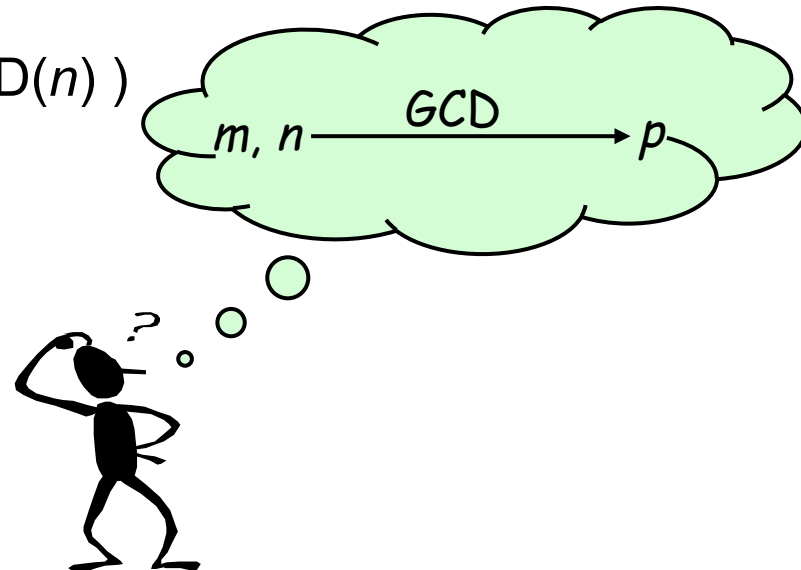


- “Try” this recipe with different pairs of non-negative integer numbers
 - Can you get the “final” answer?
- What does it mean ?
 - Is it necessary to know that, to process this recipe ?

A mathematical problem: Greatest Common Divisor (GCD)

- Determine $z = \text{GCD}(m, n)$ $m, n \in \mathbb{N}$
- $D(m) \stackrel{\text{def}}{=} \text{the set of integer divisors of } m$
- $D(m) = \{ k \mid m = k \cdot q, k \in \mathbb{N}^+, q \in \mathbb{N} \}, \mathbb{N}^+ = \mathbb{N} - \{0\}$
- $p = \text{GCD}(m, n) = \max(D(m) \cap D(n))$

Greatest Common Divisor (GCD) of two or more integers, when at least one of them is not zero, is the largest positive integer that divides the numbers without a remainder.



MATLAB code for GCD

```
function res = findgcd(m, n)
% Find the GCD of the input arguments using Euclidean algorithm.
% The two inputs must be positive integers.

if (m ~= floor(m)) || (m < 0) || (n ~= floor(n)) || (n < 0)
    fprintf('Enter two positive integers!\n');
    fprintf('You entered m=%.2f, n=%.2f\n', m, n);
    return
end

while n ~= 0
    r = mod(m,n);
    m = n;
    n = r;
end

res=m

end
```

See findgcd.m

Error-checking user input

- How to prompt the user until the user enters a value in a correct range?
- Loop until the user enters the correct value

```
% Loop until the user enters a positive number
```

```
inputnum = input('Enter a positive number: ');
```

```
while inputnum < 0
```

```
    inputnum = input('Invalid! Enter a positive  
                    number: ');
```

```
end
```

```
fprintf('Thanks, you entered a %.1f \n',inputnum)
```

See readonenum.m

Error-checking user input

- How to prompt the user to enter positive numbers n times?

```
% Loop until the user enters n positive numbers
n=3;
fprintf('Please enter %d positive numbers\n\n', n)
for i=1:n
    inputnum = input('Enter a positive number: ');
    while inputnum < 0
        inputnum = input('Invalid! Enter a positive
                           number: ');
    end
    fprintf('Thanks, you entered a %.1f \n', inputnum);
end
```

See readnnums.m

Error-checking for integers

- How to check the user has entered an integer?
 - Recall `int32` built-in function to convert to 32-bit integer type

```
% Error checks until the user enters a positive integer
inputnum = input('Enter a positive integer: ');
num2 = int32(inputnum);
while num2 ~= inputnum || num2 < 0
    inputnum = input('Invalid! Enter a positive
                    integer: ');
    num2 = int32(inputnum);
end
fprintf('Thanks, you entered %d\n', inputnum)
```

See `readoneposint.m`

Input in a while loop

- A **while** loop can be used to process input from the user as long as the user is entering data in a correct way

```
% Prompts the user and echo prints the numbers
% entered until the user enters a negative number
numvec = [];
inputnum=input('Enter a positive number: ');
while inputnum >= 0
    numvec = [numvec inputnum];
    fprintf('You entered a %d.\n\n', inputnum)
    inputnum = input('Enter a positive number: ');
end
fprintf('OK!\n')
```

See whileposnum.m

Counting in a while loop

- Sometimes it is useful to know how many times a given action has been executed within a loop
 - We need a **counter**

```
% Prompts the user for positive numbers and echo prints as  
% long as the user enters positive numbers
```

```
% Counts the positive numbers entered by the user
```

```
counter=0;
```

```
inputnum=input('Enter a positive number: ');
```

```
while inputnum >= 0
```

```
    fprintf('You entered a %d.\n\n', inputnum)
```

```
    counter = counter + 1;
```

```
    inputnum = input('Enter a positive number: ');
```

```
end
```

```
fprintf('Thanks, you entered %d positive numbers.\n',  
counter)
```

See countposnum.m

Simple plots in MATLAB

- MATLAB has many graphing capabilities
- For help on graph functions:

```
>> help graph2d      %Two dimensional graphs  
>> help graph3d      %Three dimensional graphs
```
- **plot** function: to plot what is specified as input parameter(s)
- To put labels on the x-axis, y-axis and the graph itself use the functions **xlabel**, **ylabel** and **title**
- To control axis scaling and appearance use **axis**

Our first plot

```
% This is a really simple plot of just one point!

% Create coordinate variables and plot a red '*'
x = 11;
y = 48;
plot(x,y,'r*')

% Put a title on the plot
title('Time and Temp')

% Label the axes
xlabel('Time')
ylabel('Temperature')
% Change the axes
axis([9 12 35 55])
```

See [plotonepoint.m](#)

Basic file input/output

- Input to a script or a function can come from a data file created by another source
- Output can be stored in an external file to be manipulated and/or printed later
- Three different operations on files:
 - **Reading** from a file
 - **Writing** to a file: writing to a file from the beginning
 - **Appending** to a file: it is also writing, but starting at the end of the file rather than at the beginning
- For now, we will consider only the built-in functions **load** and **save**
 - **load** to read from a file
 - **save** to write to a file

Writing data to a file

- The **save** command can be used to write data from a matrix to a new data file (overwritten in case it already exists), or to append data to an existing data file

save filename matrixvariablename -ascii

- Example

```
>> mymat = rand(2,3)
mymat =
    0.223623    0.338097    0.896119
    0.182871    0.025797    0.667263
>> save testfile.dat mymat -ascii
```

Appending data to a file

- Data can be appended to an existing file
- Use same format of **save** function, just add the qualifier **-append**

- Example

```
>> mymat2 = rand(3,3)
```

```
mymat2 =
```

```
    0.489620    0.610633    0.630217
```

```
    0.813602    0.075313    0.085837
```

```
    0.139874    0.063809    0.531430
```

```
>> save testfile.dat mymat2 -ascii -append
```

To be able to read the data file back into a matrix later, the appended matrix would have the same number of columns

Reading from a file

- Use **load** built-in function: once a file has been created, it can be read into a matrix variable
- Example:

```
testfile.dat
```

```
0.4565    0.8214    0.6154
```

```
0.0815    0.4447    0.7919
```

```
>> load testfile.dat
```

```
>> testfile
```

```
testfile =
```

```
0.456500    0.821400    0.615400
```

```
0.081500    0.444700    0.791900
```

load works well only if there is the same number of values in each line of the file (i.e., the same number of columns)

Reading from a file and then plot

```
% This reads time and temperature data for a day  
% from a file and plots the data
```

```
load timetemp.dat
```

```
% The times are in the first row, temps in the  
second row
```

```
time = timetemp(1,:);
```

```
temp = timetemp(2,:);
```

```
% Plot the data and label the plot
```

```
plot(time,temp,'k*')
```

```
xlabel('Time')
```

```
ylabel('Temperature')
```

```
title('Temperatures one day')
```

See timetempprob.m

Reading from a file using `while` loop

- Example: read data from a file into a vector up to, but not including, the -99 value

- -99 is an example of *sentinel*, a marker in between data sets

% Reads data from a file, but only plots the numbers
% up to a flag of -99. Uses a while loop.

```
load experd.dat
```

```
i = 1;
```

```
while experd(i) ~= -99
```

```
    newvec(i) = experd(i);
```

```
    i = i + 1;
```

```
end
```

```
plot(newvec, 'ko');
```

```
xlabel('Reading #');
```

```
ylabel('Weight(pounds)');
```

```
title('First Data Set');
```

See `findvalwhile.m`

Code the example in a more efficient way

```
% Reads data from a file, but only plots  
% the numbers up to a flag of -99.  
% Uses find and the colon operator
```

```
load experd.dat
```

```
where = find(experd == -99);  
newvec = experd(1:where-1);
```

```
plot(newvec, 'ko');  
xlabel('Reading #');  
ylabel('Weight(pounds)');  
title('First Data Set');
```

See findval.m