# ENGR1510J — Accelerated Introduction to Computers and Programming

## MATLAB Midterm Review

Zhang Kaiwen

UM-JI (Fall 2021)

October 22, 2021

# Number base conversion

Check Slides p33

- Decimal: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} human language
- Binary: {0,1} computer language
- Hexadecimal: {0, 1, $\cdots$ , 9, A, B, C, D, E, F} compromise between both

From base $b$ into decimal: evaluate the polynomial:

$(11111101)_2 = 1\cdot 2^7 + 1\cdot 2^6 + 1\cdot 2^5 + 1\cdot 2^4 + 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 = 253$

$(\mathrm{FD})_{16} = \mathrm{F}\cdot 16^1 + \mathrm{D}\cdot 16^0 = 15\cdot 16^1 + 13\cdot 16^0 = 253$

From decimal into base $b$: repeatedly divide $n$ by $b$ until the quotient is 0. The remainders are the numbers from right to left:

rem(253,2)=1, rem(126,2)=0, rem(63,2)=1, rem(31,2)=1, rem(15,2)=1, rem(7,2)=1, rem(3,2)=1, rem(1,2)=1

rem(253,16)=13=D, rem(15,16)=15=F

# Algorithm and Program

Algorithm $\longrightarrow$ Programming Language $\longrightarrow$ Interpreter / Compiler $\longrightarrow$ Machine Code

Example: Recursion Algorithm $\longrightarrow$ Matlab $\longrightarrow$ Matlab Interpreter $\longrightarrow$ Machine Code

**Algorithm**: recipe telling the computer how to solve a problem.

**Programming Language**: a formal language that specifies a set of instructions that can be used to produce various kinds of output.

**Interpreter**: a computer program that directly executes a programming or scripting language.

**Compiler**: computer software that transforms computer code written in one programming language (the source language) into another programming language (the target language).

**Program**: a technical setting stored in the memory of a machine or piece of hardware to be executed, including computers.

**Machine code**: a set of instructions executed directly by a computer's central processing unit (CPU).

# Source Code and Pseudocode

**Source Code**: a sequence of statements and/or declarations written in some human-readable (usually as a text) computer programming language

```
1  function M = density_sun(r, c, G, T)
2      V = 4 / 3 * pi * (c / (2 * pi)) ^ 3
3      M = 4 * pi ^ 2 * r ^ 3 / (G * T ^ 2)
4  end
```

**Pseudocode**: an informal high-level description of the operating principle of a computer program or other algorithm.

---

**Input:** $r, c, G, T$
**Output:** Density of the Sun

1: **function** DENSITY SUN($r, c, G, T$)
2: $V \leftarrow \frac{4}{3}\pi\left(\frac{c}{2\pi}\right)^3$
3: $M \leftarrow \frac{4\pi^2 r^3}{GT^2}$
4: **return** $M$
5: **end function**

# Calculation Operators

Check whether you use correct division operator!

- (Right) division: a / b $= \frac{a}{b}$
- Left division: a \b $= \frac{b}{a}$

For matrix, pay attention to the difference between the operators with . and without .

Element by element operator: (applicable to array or matrices?)

- .*
- ./
- .\
- .^

# Arrays and Matrices

Generation of Matrices slide p59:

- Obtain a sequence of numbers: `a:b` or `a:b:c`
- Concatenate (join) elements: `[ ]`
- Define a 1-dimensional array: `[a:b]` or `[a:b:c]`
- Define a 2-dimensional array: `[a b c; d e f;]`
- Get n equidistant elements in [a, b]: `linspace(a, b, n)`
- Get an n $\times$ m array of 0: `zeros(a,b)`
- Get an n $\times$ m array of 1: `ones(a,b)`

# Accessing elements in a matrix[2]

- Coordinates: using their (row, column) position
- Use ":" to refer to the whole row / column
- "1" is the top / left index
- "end" is the bottom / right index
- Use `size(A)` to get the number of rows and columns in A
- use `length(A)` to get `max(size(A))`
- use an expression of A (i.e., `A==x`) to generate a logical array depending on some condition
- let B be the generated logical array above, then use `A(B)` to generate a new array with elements in A that satisfy this condition

# Expressions

Comparative operators:

- $<$ / $<=$ less than / less than or equal to
- $>$ / $>=$ greater than / greater than or equal to
- $==$ / $\sim=$ equal to / not equal to (Don't confuse with C's not equal to)

Logical operators:

- And: & , &&
- Or: |, ||
- Not: $\sim$
- Xor: `xor(.,.)` exclusive or of A and B

What are short-circuit operators? Why do we need to use them?

# If Statements & Switch Statements

`if-elseif-if` statement:

```
1  if expression1
2      statements1
3  elseif expression2
4      statements2
5  else
6      statements
7  end
```

`switch` statement:

```
1  switch variable
2      case value1
3          statements1
4      case value2
5          statements2
6      otherwise
7          statements
8  end
```

Pay attacking to the position of end and keywords `elseif`!

# Loop Statements[2]

The `while` loop:

```
while expression
    statements
end
```

The `for` loop:

```
for i=start:increment:end
    statements
end
```

The `continue` and `break` commands:

- `continue`: skip the remaining statements in the loop to go to the next iteration.
- `break`: exit the loop and execute the next statements outside the loop.

Efficiency in multilevel loops: MATLAB uses the "column-major order", so column should be in the outer loop.
When to use which? Slide p69.

# Script vs Function

Check slides p83. Script:

- Sequence of MATLAB statements
- No input/output arguments
- Operates on data on the workspace

Function:

- Sequence of MATLAB statements
- Accepts input/output arguments
- Variable are not created on the workspace

# Functions and Sub-functions[2][3]

- Function saved in a .m file
- The .m file must be in the `Path`
- The function name must be the same as the filename
- `function [output1, output2,...] = Functionname(input1,input2,...)`
- The function can be called from another .m file or from the workspace

A .m file can contain:

- A "main function"
- Several sub-functions, only visible to other functions in the same file

For a function in the form:

`function [output1, output2] = Functionname(input1,input2)` We call it by `Functionname(input1,input2)`

# Variable Scope

Consider the code below:

```
1       function output=out_func
2       input=1;
3       funct(input);
4       fprintf("outside the funct: %d\n",input);
5       output=input;
6       end
7
8       function funct(input)
9       input=input+1;
10      fprintf("inside the funct: %d\n",input);
11      end
```

We call it by output=out_func, and then ·····

- What will be the printed information in the terminal?
- What will be the value of output?
- Why?

# Useful Matlab Functions

Mathematical functions:

- Defining an anonymous function: `f=@(x) x^2-1`
- Rounding: `mod(a, b)`, `floor(a)`, `ceil(a)`, `round(x, n)`

Random-related functions:

- n x m matrix of random numbers: `rand(n,m)`
- An n $\times$ n matrix of random integers between m and M: `randi([m M],n)`
- A random permutation: `randperm(n)`

Print functions: Search for their documents for more information about format.

- Format data into string or character vector:
  `sprintf(formatSpec,A1,...)`
- Formats data and displays the results on the screen:
  `fprintf(formatSpec,A1,...)`

# File I/O Functions

- `fid = fopen('filename', 'permission')`
- `fgetl(fid)`: read one line from the file
- `fscanf(fileID,formatSpec)`: read data from text file
- `fprintf(fileID,formatSpec,A1,...)`: write data to text file
- `fclose(fid)`: **Double check this!**

# Recursion

**Recursion**: repeating items in a self-similar way (from difficult to relatively easy)
**Implementation**: a function calling itself
**Key Steps**:

- Find the repeated steps
- Find the terminate situation

# Example 1: Calculate Factorial

```
1  function f = fact(n)
2      if n == 0 || n == 1 % terminate situation
3          f = 1;
4      else
5          f = n * fact(n - 1); %repeated steps
6      end
7  end
```

- Repeated steps: fact(n) = n * fact(n-1)
- Terminate situation: n = 0 || n = 1

# Example 2: Smallest Element in a Non-Empty array[4]

Find the smallest element in a non-empty array V.
Two recursive methods:

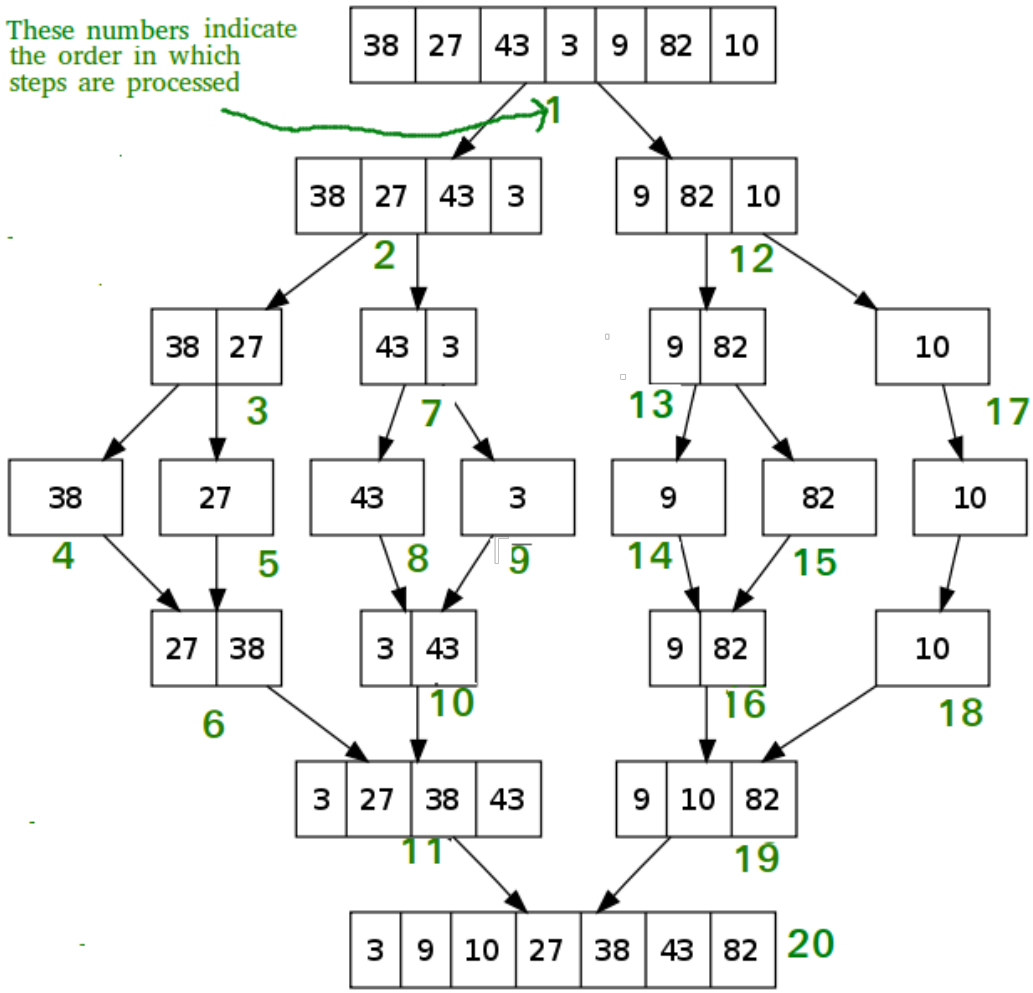
- Method1:
  We have: F(V(1),V(2),V(3),...)=min(V(1),F(V(2),V(3),...))=.....
  Initial: src=1,des=array.size; (applicable to Matlab only!)
  Repeated steps: F(array,src,des)=min(array(src),F(array,src+1,des))
  Terminate situation: src==des
- Method2:
  Divide into left and right part
  Repeated step: divided the array into left part and right part. Find the min respectively.
  Terminate step: current subarray has size 1.

# Example 3: Binary Search

Find the location of a certain integer(existed) i in an sorted ascending array V.

- Repeated Steps:
  - get the middle location of current array. Compare V(middle) with i.
  - If V(middle) >i, we search in the left sub-array. If V(middle) <i, we search in the right sub-array.
- Terminate situation:We find the location of i.

# Example 4: Merge Sort




Picture from [5].

# Example 4: Merge Sort

- Repeated Steps:
  - Split the array into two sub-arrays with roughly equal length.
  - Merge sort the sub-arrays respectively.
  - Merge the sorted sub-arrays.
- Terminate situation:The split arrays are of size 1 or 0.

# Good Luck and Enjoy Programming!

# Reference



[1] Manuel,Charlemagne. ENGR1510J 21FA slides.


[2] Yihao,Liu. VG101 MATLAB Midterm Review. 2018 SU


[3] Bowen,Wang. VG101 MATLAB Midterm Review. 2019 FA


[4] Shixin,Song. VG101 Midterm Review Chapter 3. 2020 FA


[5] Merge Sort https://www.geeksforgeeks.org/merge-sort/