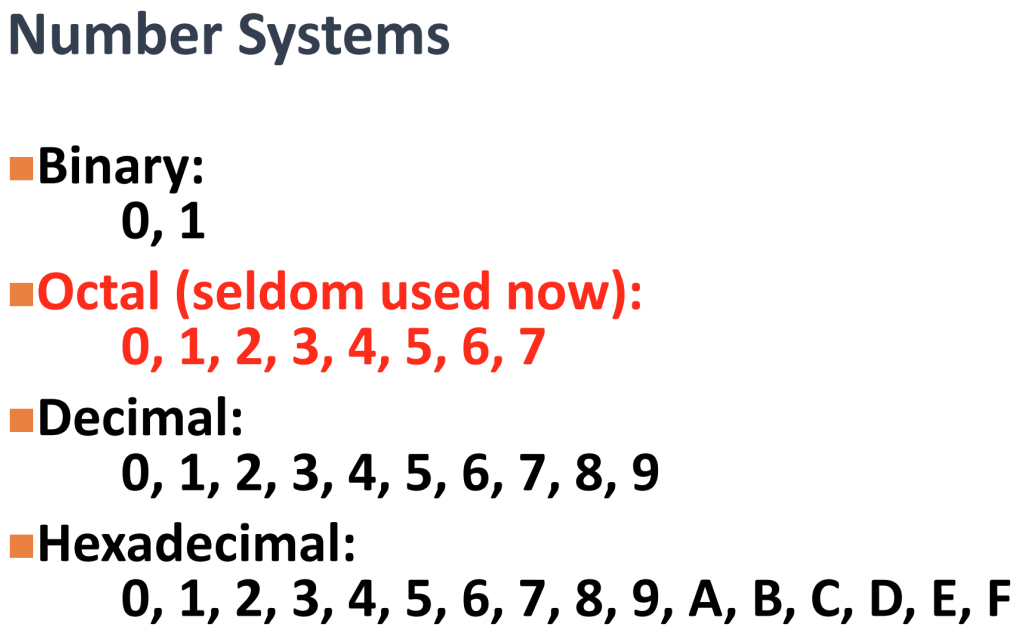
VG101 Mid 1 Review Part1

Author: Lu Haorong

Base Conversion

- **BASIC:** Get familiar with the form of **binary and hexadecimal** number system



- **BASIC:** Get familiar with the conversion mechanisms between **binary, decimal and hexadecimal** numbers
  - Bin/Hex --> Dec: $\sum_{i=0}^{n} digit(i) \times (2 \ or \ 16)^i$
  - Dec --> Bin/Hex: **Successive dividing + remainder** (check Page 31/33, Lecture 1 for detail)
  - Bin <--> Hex: **4 digits in binary <--> 1 digit in hexadecimal**
    - This is a quite useful trick in future ECE courses, like in VE270 and VE370.
- **IMPORTANT:** Some built-in base conversion functions in MATLAB
  - `bin2dec('1111') = 15` **string to double**
  - `hex2dec('FF') = 255` **string to double**
  - `dec2bin(15) = '1111'` **num (double, int32, int64...) to string**
  - `dec2bin(15, 8) = '00001111'` **Pads the output to the specified minimum digits**
  - `dec2hex(255) = 'FF'` **num (double, int32, int64...) to string**
  - `dec2hex(255, 4) = '00FF'` **Pads the output to the specified minimum digits**
  - `D = base2dec(baseStr, n)` **Convert base n to decimal**
  - `baseStr = dec2base(D, n, minDigits)` **Convert decimal to base n (`minDigits` can be omitted)**

Examples

- ***2019 Q3***

3. (20 points) A series of hexadecimal numbers can be represented as a character string such as '1A 22 30 4F', where the numbers are separated by spaces. Note that there is only one space between two numbers and no space at the beginning or at the end of the string. Write a Matlab function with prototype

function *ret* = HexSum(*str*)

to calculate and return the sum of all the hexadecimal numbers contains in the string *str* as described above. The returned variable *ret* should be a hexadecimal number represented as a string. For example, HexSum('1A 2B') should return '45'. Hint: to make your life easier, you can choose to use the Matlab functions *hex2dec* and *dec2hex*.

- ***2018 Q4***

to convert hexadecimal numbers in a text file with file name of *filename* into decimal numbers and save them to another text file. The input text file contains a list of hexadecimal numbers where each line contains only one number. Similarly the output text file contains a list of the corresponding decimal numbers where each line also contains only one number. The following shows the contents of an example input file and the corresponding output file after calling the function:

Input file 'hexnumber.txt'

1
FF
20F

Output file 'dec_hexnumber.txt'

1
255
527

Furthermore, the output file name should be 'dec_[*filename*]', where [*filename*] should be replaced with the input file name. For example if the input *filename* is 'hexnumber.txt', then the output file name should be 'dec_hexnumber.txt'.

Function

Syntax

GENERAL prototype:

```
function [rets...] = name(args...)
    % do something
end
```

In the case of **SINGLE** return value:

```
function ret = name(args...)
    % do something
end
```

In the case of **NO** return value:

```
function name(args...)
    % do something
end
```

In EXAM, just copy the prototype given by Jigang.

Something You Need To Know

- If it requires to write a function on the paper, and you write a script, there will be a great deduction.
- If you give value to some return values, when the function returns, the value will be returned.
- Use `return` to directly return the function.

Why we need a function?

Function is the abstraction of real implementation.

For example, sum an array

- Without function:

```
sum1 = 0;
for i = 1:length(arr1)
    sum1 = sum1 + arr1(i);
end

sum2 = 0;
for i = 1:length(arr2)
    sum2 = sum2 + arr2(i);
end

sum3 = 0;
for i = 1:length(arr3)
    sum3 = sum3 + arr3(i);
end

% ...
```

- With function:

```
function ret = sum(arr)
    ret = 0;
    for i = 1:length(arr)
        ret = ret + arr(i);
    end
end

sum1 = sum(arr1); % No need to implement it again!
sum2 = sum(arr2);
sum3 = sum(arr3);
```

How to implement/invoke a new function?

Take **lab2** as an example,

First, carefully read through the **function description and prototype**, and after reading, you should be able to conclude following messages

- What does the function do?
- What's the meaning of each input and return value?

```
% Return a 0/1 matrix given the size of the matrix and propotion of alive cells
% 0 represents a dead cell, 1 represents an alive cell
% a: Board rows
% b: Board columns
% p: Propotion of alive cells
% board: Initialized lifegame matrix
function board = init_board(a, b, p)

% Return the board of next step given the current board
% The board of next step is generated through the rule of lifegame
% in_board: Current board
% board: The board of next step
function board = update_board(in_board)

% Display the board in a picture
% in_board: The board you want to display
function view_board(in_board)
```

Now you should be quite clear about what the function does, and when you implement the function, always remember to keep it **independent and invariant!** You should test each function separately with several test cases, just like what TA did in lab2.

Then, when you want to invoke your new functions, just refer to **its high-level definition instead of its low-level implementation**. Function is the abstraction of the implementation, for example, when you invoke `fprintf("%d\n", num)` to print something, you never consider how it is implemented (OS system call, very complex). You just know it's a function that can **print something to the command window given a format string and some variables**. The same is true when you invoke a function written by yourself.

For example, you want to write a script initializing a 5×8 lifegame board with proportion 0.3, and display the original board and the board after one step.

```
clear;clf;
orig_board = init_board(5, 8, 0.3);   % initialize the original board
view_board(orig_board);               % display the original board
new_board = update_board(orig_board); % update the original board for one step
view_board(new_board);                % display the new board
```

Data Types

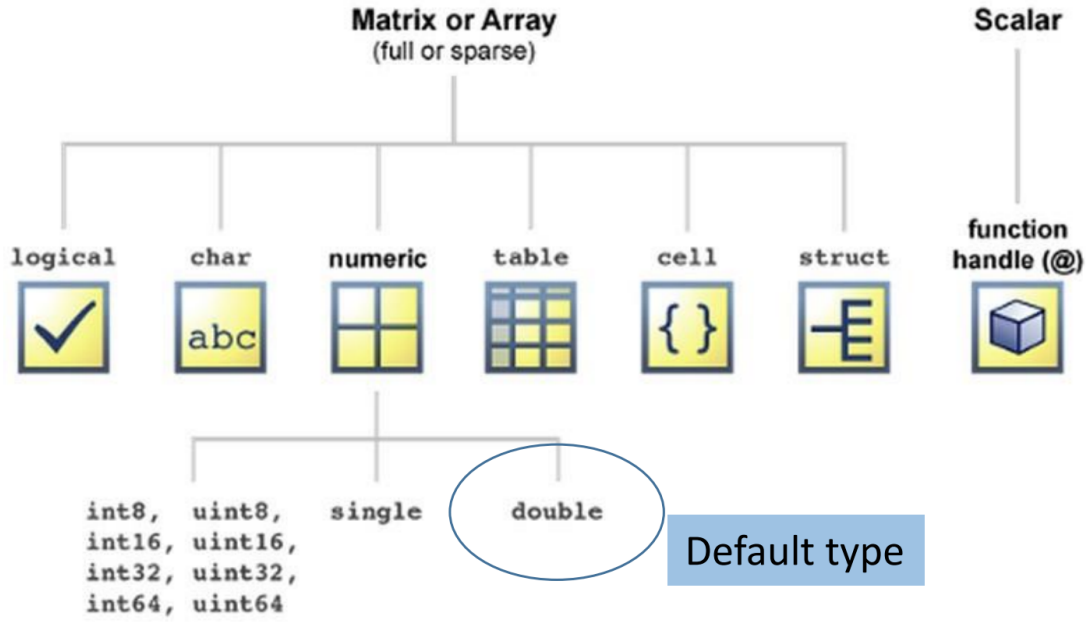
Why is data type needed?

It tells the compiler/interpreter how to interpret the data, for example,

For a variable `var`, its data (binary form) in the memory is `01100001`

- If the data type for `var` is `uint8, uint16, ...`, it's value is `97`
- If the data type for `var` is `double` (MATLAB default data type), it's value is $4.792436764660091478512717290 82 \times 10^{-322}$
- If the data type for `var` is `char`, it's value is `'a'` (ASCII Code)

Data Types in MATLAB



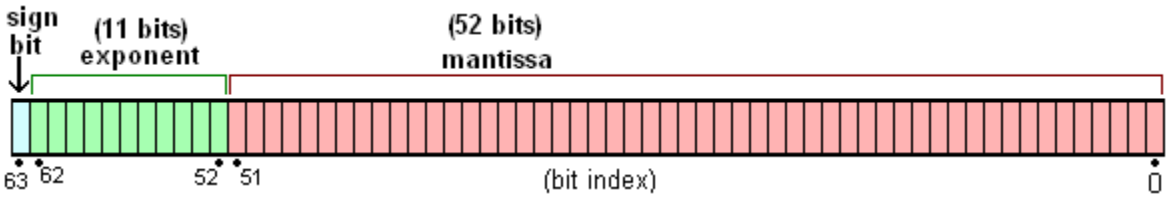
Data Type	Range of Values	Conversion Function
Signed 8-bit integer	-2^7 to 2^7-1	int8
Signed 16-bit integer	-2^{15} to $2^{15}-1$	int16
Signed 32-bit integer	-2^{31} to $2^{31}-1$	int32
Signed 64-bit integer	-2^{63} to $2^{63}-1$	int64
Unsigned 8-bit integer	0 to 2^8-1	uint8 Very useful,
Unsigned 16-bit integer	0 to $2^{16}-1$	uint16
Unsigned 32-bit integer	0 to $2^{32}-1$	uint32
Unsigned 64-bit integer	0 to $2^{64}-1$	uint64

Signed integer: Use the first bit to represent positive/negative, 2's complement (detail in VE270)

Unsigned integer: Only represent non-negative values, natural binary form

Float-Point Numbers (Just for fun)

Scientific notation in binary form



$$Float = (-1)^S \times 2^{E-1023} \times (1+M)$$

Since it use 52 bits to represent the mantissa, its relative accuracy (`eps` in MATLAB) will be greater than 1 when the exponent is greater than 52, namely when the number is greater than 2^{52}. Therefore, it's not precise to use `single/double` to represent big numbers, though their ranges are quite huge.

Application in MATLAB (Only for Lab 1)

Since MATLAB is a weakly and dynamically typed language, usually we do not need to care about the actual data type, especially in VG101. However, there are some special cases that MATLAB's default data type will cause problem, for example,

In Lab1, we have to deal with a lot of huge integers, but we know that the `double` data type will have an `eps` greater than 1 when the number is greater than 2^{52}, namely it cannot precisely represent a huge integer. To extend the range of integers that can be accepted for this program, we may convert the integer from `double` to `uint64`, and then the upper limit (precisely) of integer will become $2^{64}-1$, which is much larger than `double`.

Characters and strings

Overview

- In MATLAB, the term *string* traditionally (before R2016b) refers to an array of Unicode characters.
- Specify character data by placing characters inside a pair of single quotes.
- In MATLAB version after R2017a, you can create strings using double quotes.
- We will focus on the traditional string.

- Use single quotation for string

- String is an array of characters, so you can connect several strings through matrix ways
  - `['ab', 'cd', 'ef'] = 'abcdef'`
- Traverse a string char by char,

```
str = 'abc1def';
for i = 1:length(str)
    fprintf("%s ", str(i));
end
fprintf("\n");
% a b c 1 d e f
```

- In MATLAB, char is actually encoded using the *UTF-16* encoding, which occupies 2 bytes in the memory, but in VG101, you can always treat it as the *ASCII* code.
  - https://www.ascii-code.com/
- As it's *ASCII* code, there are some useful tricks,

```
% check if a char is a digit
if ch >= '0' && ch <= '9'
    disp('is digit');
end

% calculate the sum of digits
% 1. without char
sumD = 0;
while num > 1
    sumD = sumD + mod(num, 10);
    num = floor(num / 10);
end

% 2. with char
sumD = sum(num2str(num) - '0');
```

Useful functions for strings

```
tf = strcmp(s1, s2);
k = strfind(str,pat);
newStr = strrep(str,old,new);
s = num2str(A);
X = str2double(str);
```

Use `help` function in MATLAB to get more information about these functions

Example:

- Parse the input command:
  keyword parameter1 parameter2 … add 1 2

```
% parse
command = input('Please type a command: ','s');
space = strfind(command,' ');
if strcmp(command(1:space(1)-1),'add')
   parameter1 = str2double(command(space(1)+1:space(2)-1));
   parameter2 = str2double(command(space(2)+1:end));
   result = parameter1+parameter2
end
```

Another useful function to split/count words

- *2020 Q3*

3. (20 points) Write a Matlab function with prototype

function *count* = CountWords(str)

to count the number of words in a character string *str*. Here the words in the string are separated by spaces, where there can be **one or multiple spaces** between the two words. For example, CountWords('happy new year.') should return 3.

```
function count = CountWords(str)
    arr = split(str, ' ');
    count = length(arr);
end
% What happens if there are spaces at the begin/end of the string?
```

Control Statements and Loops

Actually this is the most important part in the exam, and nearly appear in all exam problems, but it's hard to teach how to use them without real practice. Practice on every sample problems in VG101, and then I believe you can master control statements and loops.

IF/ELSEIF/ELSE Branch

Syntax:

```
if BOOLEAN
    BLOCK1
elseif BOOLEAN % be careful about the syntax
    BLOCK2
else
    BLOCK3
end
```

Switch/Case Branch

Syntax:

```
switch EXPRESSION
case VALUE1
    BLOBK1
case {VALUE2, VALUE3, VALUE4}
    BLOBK2
otherwise
    BLOCK3
end
```

When to use switch/case

1. Condition is simple
2. Duplicated expressions
3. Long branches

While Loop

Syntax:

```
while CONDITION
    BLOCK
end
```

For Loop

Syntax:

```
for element = ARRAY
    BLOCK
end
```

Example for `break` and `continue`

```
% Find the first Palindrome with even digits in an array
% e.g.
% arr = [12321 4423 1221 3232 121]
% ret = 1221
function ret = EvenPalindrome(arr)
    for i = 1:length(arr)
        str = num2str(arr(i)); % convert double into string
        n = length(str);
        if mod(n, 2) == 1 % if odd digits, jump to next iteration
            continue;
        end
        % check Palindrome
```

```
        isPalindrome = true;
        for j = 1:n/2
            if str(i) ~= str(n+1-i)   % if one char pair is not equivalent,
                isPalindrome = false; % directly exit the check loop
                break;
            end
        end
        if isPalindrome   % if one Palindrome with even digits is found,
            ret = arr(i); % give the value to ret and exit the loop,
            break;        % since we only need to find the first one
        end
    end
end
```

Reminder

- Read the problem carefully before you start writing the code, as it's an open-internet exam, you may use translation tools to help you understand the problem. (But do not blindly trust translation tools)
- For all functions appear in today's RC, go checking the MATLAB official documentation if you do not understand how to use them.
- If you cannot guarantee the correctness of your code, you can add some simple comments to describe the function of each part of your code, because we will give partial scores based on your code.

Reference

1. Kaibin Wang (TA in VG101 2020FA), VG101 Jigang RC2 - Control statements, loops, functions and MATLAB in practice.md
2. Kaibin Wang, VG101 Mid1 Review.md
3. Yuexi Du (TA in VG1012020FA), ExampleExam.pdf
4. Jigang Wu, VG101 2021FA Lecture Slides