Computer Programming
Bachelor in Biomedical Engineering
Bachelor in Applied Mathematics and Computing

Exercise Sheet
Revision Exercises II
- SOLUTIONS -

Content Table

Exercise 1

Write a program that asks the user to enter a character string, and then displays the string diagonally.

Example of execution:

```
Please enter a string: Hello
H
  e
   l
    l
      o
```

SOLUTION

```
s = input('Please enter a String: ', 's');
for i=1:length(s)
    for j=1:i
        fprintf(' ');
    end
    fprintf('%s\n', s(i));
end
```

Exercise 2

Write a program that asks the user to introduce a sentence and a word, and removes all the occurrences of the word in the sentence.
Note: the program should make no difference between characters in upper- and lowercase.

Example of execution:

```
Enter a sentence: The cat was the animal that caught the mouse
Enter a word: the
Result: cat was animal that caught mouse
```

SOLUTION

```
sSentence = input ('Introduce a sentence: ', 's');
sWord = input ('Introduce a word: ', 's');
auxSentence = lower(sSentence);
auxWord = lower(sWord);
positions = strfind(auxSentence, auxWord);
for i=length(positions):-1:1
    index = positions(i);
    if index > 1
        sSentence = [sSentence(1:index-1)
sSentence(index+length(sWord):length(sSentence))];
    else
        sSentence = [sSentence(index+length(sWord):length(sSentence))];
    end
end
fprintf('%s\n', sSentence);
```

Exercise 3

Write a function named *deleteRepeated* that receives a string and returns the same string without consecutive repeated characters.

Example of execution:

```
>> deleteRepeated('Thisssswworrkssperfeccctlyyy')

ans =

     'Thisworksperfectly'
```

SOLUTION

```
function [strdo] = deleteRepeated (st1)
strdo = '';
if isempty(st1)==0
    strdo(1) = st1(1);
    for i=2:length(st1)
        if st1(i) ~= strdo(length(strdo))
            strdo = [strdo st1(i)];
            last = st1(i);
        end
    end
end
end
```

Exercise 4

Write a function that receives a sentence, and returns the same sentence without the words that contain consecutive repeated characters.

Example of execution:

```
>> removeWordsCons('Hello run warrior Matlab')

ans =

     'run Matlab'
```

SOLUTION

```
function [rdo] = containscons (st1)
rdo = 0;
cont = 2;
while (cont <= length(st1)) && (rdo == 0)
    if st1(cont) == st1(cont-1)
        rdo = 1;
    else
        cont = cont+1;
    end
end
end
```

```
function [strdo] = removeWordsCons (st)
strdo = '';
[word,rest] = strtok(st);
while ~isempty(word)
    if ~containscons(word)
        if isempty(strdo)
            strdo = word;
        else
            strdo = [strdo ' ' word];
        end
    end
    [word,rest] = strtok(rest);
end
end
```

Exercise 5

Write a function named *addDigits* that receives a string and returns the sum of all digits contained in it. For instance, if the string is “a8bc4fdeh9s0", the answer should be 21 (8+4+9+0=21).

Example of execution:

```
>> addDigits('a8bc4fdeh9s0')

ans =

     21
```

Note: To solve this problem you need to use the function *double*. This function receives a character and returns the number of this character in the ASCII table. For example, if you execute double(‘A’), the function returns the number of the character ‘A’ in the ASCII table, which is 65. You should also know that the ASCII numbers for the characters ‘0’ is 48, ‘1’ is 49, ‘2’ is 50 … and finally ‘9’ is 57.

SOLUTION

```
function [ sumdigits ] = addDigits ( instring )
sumdigits = 0;
for cont=1:length(instring)
    if ((double(instring(cont)) >= double('0')) && (double(instring(cont))
<= double('9')))
        sumdigits = sumdigits + (double(instring(cont)) - double('0'));
    end
end
end
```

Exercise 6

Create a function that checks whether a character string that the function received as parameter (input) is a palindrome or not, returning either true or false, respectively. A palindrome is a phrase that can be read in either direction (backwards and forwards).
For example: ‘a nut for a jar of tuna’.

Create a program that allows the user to enter a sentence and then calculates whether this sentence is palindrome or not

Examples of execution:

```
Enter a sentence: a santa at nasa
The phrase is a palindrome.

Enter a String: stressed or desserts
The phrase is not a palindrome.
```

MAIN PROGRAM

```
instring = input('Enter a String: ', 's');
if not(isempty(instring)) && isPalindrome(instring)
    fprintf('The phrase is a palindrome.\n');
else
    fprintf('The phrase is not a palindrome.\n');
end
```

FUNCTIONS

```
function [ palindrome ] = isPalindrome( inphrase )
%inphrase(inphrase == ' ') = '';    This also removes all whitespaces
inphrase = removeWhiteSpaces(inphrase);
comparestr = backwards(inphrase);
if (strcmpi(inphrase, comparestr) == 1)
    palindrome = true; % 1
else
    palindrome = false; % 0
end
end


function [ backwards_string ] = backwards( instring )
% return string backwards
backwards_string = [];
for i=length(instring):-1:1
    backwards_string = [backwards_string, instring(i)];
end
end


function [ outphrase ] = removeWhiteSpaces( inphrase )
% remove white spaces
outphrase = [];
for i=1:length(inphrase)
    % you can also do if ~(isspace(inphrase(i)))
    if inphrase(i) ~= ' '
        outphrase = [outphrase, inphrase(i)];
    end
end
end
```

Exercise 7

Write a function named *myCompareTo*, which receives two strings as input, and returns an integer from **0** to **3** according to the following rules:

- If the two strings are different: the function returns **0**
- If the two strings are equal: the function returns **1**
- If the first string is the prefix of the second one: the function returns **2**
- If the second string is the prefix of the first one: the function returns **3**

Write a program to test the function. The program asks the user to introduce two strings, calls the function and prints whether the two strings are equal, different or whether one is the prefix of the other on screen.

Note: Solve the exercise without using the functions strcmp, strcmpi, strncmp or strfind

Example of execution:

```
Introduce a sentence: dog
Introduce a sentence: cat
The two sentences are different.
```

Example of execution:

```
Introduce a sentence: good
Introduce a sentence: good morning
The first sentence is the prefix of the second one.
```

Example of execution:

```
Introduce a sentence: virtually
Introduce a sentence: virtual
The second sentence is the prefix of the first one.
```

Example of execution:

```
Introduce a sentence: biomedical
Introduce a sentence: biomedical
The two sentences are equal.
```

MAIN PROGRAM

```
st1 = input('Introduce a sentence: ','s');
st2 = input('Introduce a sentence: ','s');
vRdo = myCompare(st1, st2);
switch vRdo
    case 0
        disp('The two sentences are different');
    case 1
        disp('The first sentence is a prefix of the second');
    case 2
        disp('The second sentence is a prefix of the first');
    case 3
        disp('The two sentences are equal')
end
```

FUNCTION

```
function [result] = myCompare(st1, st2)
result = -1;
len1 = length(st1);
len2 = length(st2);
cont = 1;
equal = 1;
while (cont<=len1) && (cont<=len2) && (equal == 1)
    if st1(cont) ~= st2(cont)
        equal = 0;
    else
        cont = cont +1;
    end
end
if equal == 0
    result = 0;
elseif len1 > len2
    result = 2;
elseif len2 > len1
    result = 1;
else
    result = 3;
end
end
```

Exercise 8

Inside an application of data transmission, a program has to be created that whether information received at a destination point is equal to the information sent at the origin. The control mechanism consists in verifying whether for every text line the first 4 characters contain an integer number called control number, and the rest of the line contains its content. The program should verify that each text line is at least 5 characters long, and that the first 4 characters (the control number) are all numeric. This control number is created by adding the numeric ASCII value of each character in the text.

Write a function called *validateLine* which receives a string and returns 1 when the string is correct according to the validation mechanism, and 0 otherwise.

The following example illustrates the correct functionality of the method: A string is received with the following content:
'1330 Meat is murder'
This string's length is 18, 4 for the control number and 14 for the text content. The sum of the numeric values of each character in the text content is calculated as follows:

M	e	a	t		i	s		m	u	r	d	e	r
77	101	97	116	32	105	115	32	109	117	114	100	101	114

When adding 77 + 101 + … + 101 + 114 the result is 1330, so the control number is correct and the method should return a positive validation.

Write a program for testing the previous function. The script will ask the user to introduce sentences and control numbers until the user introduces a blank sentence.

Next, the program validates the introduced lines and prints on screen the ones which are valid.

Example:

```
Introduce a sentence: Meat is murder
Introduce a control number: 1330
VALID line
Introduce a sentence: Bilbao Madrid Barcelona
Introduce a control number: 3333
INVALID line
Introduce a sentence: The chain value
Introduce a control number: 1409
VALID Line
Introduce a sentence:
Ok, bye!
```

MAIN PROGRAM

```
cont = 0;
sentence = input('Introduce a sentence: ','s');
while not(isempty(sentence))
    cont = cont + 1;
    control = input('Introduce a control number: ','s');
    sentence = [control sentence];
    if validateLine(sentence)
        disp('Valid line');
    else
        disp('Invalid line');
    end
    sentence = input('Introduce a sentence: ','s');
end
disp('Ok, bye! ');
```

FUNCTION

```
function [ outvalid ] = validateLine ( instring )
if length(instring) < 5
    outvalid = 0;
else
    control = str2num(instring(1:4));
    valchar = sum(double(instring(5:end)));
    if control == valchar
        outvalid = 1;
    else
        outvalid = 0;
    end
end
end
```

Exercise 9

Write a function that verifies the balancing of parentheses in a string. The parentheses in a string are balanced when each opening parenthesis has a corresponding closing parenthesis. For example, the parenthesis of the following strings are balanced:

(if (zero? x) max (/ 1 x))
I told him (that it's not (yet) done). (But he wasn't listening)

while the parentheses in these strings are not balanced

if (zero? x) max (/ 1 x)))
this is)another) boring (example(
: -)

To solve this problem, write the following functions:

1. A function *numParenthesis* that receives a string and returns a vector containing as many values as there are characters in the string, where each of them represents the number of parentheses that remain opened at each position of the string.

   For example:

```
>> numParenthesis('1+(2+(1+1))')

ans =

       0  0  1  1  1  2  2  2  2  1  0
```

   The function indicates that in positions 1 and 2 there are 0 parentheses that remain opened, in positions 3, 4, and 5 1 parentheses remains opened, in positions 6, 7, 8, and 9 2 parentheses are opened, and finally, in positions 10 and 11 there are 1 and 0 parentheses that remain opened respectively.

2. A function called *checkBalance* that receives a string and returns 1 if the parentheses in the strings are balanced and 0 otherwise.

   For example:

```
>> checkBalance(('1+(2+(1+1))')

ans =

       1

>> checkBalance('this is )another)boring(example(')

ans =

       0
```

SOLUTION

```
function [vectpare] = numParentheses (inphrase)
vectpare = zeros(length(inphrase),1);
count = 0;
for i=1:length(inphrase)
    switch (inphrase(i))
        case '('
            count = count + 1;
        case ')'
            count = count -1;
    end
    vectpare(i) = count;
end
end
```

```
function [isBalanced] = checkBalance(inVector)
isBalanced = 1;
vect = numParentheses(inVector);
if (vect(length(vect)) ~= 0)
    isBalanced = 0;
else
    count = 1;
    while ((isBalanced) && (count <=length(vect)))
        if (vect(count) < 0)
            isBalanced = 0;
        else
            count = count + 1;
        end
    end
end
end
```

Exercise 10

Create a function called notBad that receives a single argument, a string. The function should find the first appearance of the substring 'not' and 'bad'. If 'bad' follows 'not', then it should replace the whole 'not'...'bad' substring with 'good' and return the result. If it doesn't find 'not' and 'bad' in the right sequence (or at all), just return the original sentence.

```
>> notBad('This dinner is not that bad!')

ans =

      'This dinner is good!'

>> notBad ('This movie is not so bad!')

ans =

      'This movie is good!'

>> notBad ('This dinner is bad!')

ans =

      'This dinner is bad!'
```

SOLUTION

```
function [out] = notbad (in)
i_not = strfind(in,'not');
i_bad = strfind(in,'bad');
out = '';
if i_bad > i_not
    for i = 1:i_not-1
        out = [out in(i)];
    end
    out = [out 'good'];
    for i = i_bad+3:length(in)
        out = [out in(i)];
    end
else
    out = in;
end
end
```

Exercise 11

Write a function *changeCase* that receives a sentence that contains some words marked with a special character at the beginning of the word. The function should return the same sentence with the marked words modified according to these rules:

- - : words whose first character is – should be converted to uppercase
- _ : words whose first character is _ should be converted to lowercase
- *: words whose first character is * should have the first character of the word converted to uppercase and the rest of them to lowercase
- the rest of the words should not change

Example of execution:

```
>> changeCase('-Hello *world, my _nAMe is *BOB')
ans =

    'HELLO World, my name is Bob'
```

SOLUTION

```
function [sentence_out] = changeCase (sentence_in)
sentence_out = '';
% break sentence into words
[word,rest] = strtok(sentence_in);
while ~isempty(word)
    firstchar = word(1);
    % find out if word should be transformed
    switch(firstchar)
        case '_'
            word(1)='';
            word = lower(word);
        case '-'
            word(1)='';
            word = upper(word);
        case '*'
            word(1)='';
```

```
            word = lower(word);
            word(1) = upper(word(1));
    end
    sentence_out = [sentence_out ' ' word];
    [word,rest] = strtok(rest);
end
% add to sentence_out
sentence_out(1) = '';
end
```

Exercise 12

Write a function *addVariables* that imitates the way matlab's fprintf function works. The function receives one sentence. This sentence has two separate parts: first some text, and then a list of numbers. These two parts are separated with a comma. The different numbers are also separated by commas. The character * indicates the position in the text where the numbers must go. The function should return a new version of the sentence where the character * in the text has been replaced by the numbers, following their order of appearance: the first number should take the place of the first *, the second number should replace the second occurrence of *, and so on.

If the number of stars (*) is not the same as the number of numbers, the program will return the sentence 'ERROR'

Examples of execution:

```
>> addVariables('there is * thing * say: * words *
you,1,2,3,4')

ans =

      'there is 1 thing 2 say: 3 words 4 you'


>> addVariables('there is * thing * say: * words *
you,1,2,3,4,5,6')

ans =

      'ERROR'
```

SOLUTION

```
function [sentence_out] = addVariables (sentence_in)
sentence_out = '';
% get nums
nums = [];
[text, vars] = strtok(sentence_in,',');
rest = vars;
while ~isempty(rest)
    [new_num,rest] = strtok(rest,',');
    nums = [nums new_num];
end
if length(strfind(text,'*')) == length(nums)
% replace characters with numbers
    next_num = 1;
    for letter = text
```

```
        if letter == '*'
            sentence_out = [sentence_out nums(next_num)];
            next_num = next_num+1;

        else
            sentence_out = [sentence_out letter];
        end
    end
else
    sentence_out = 'ERROR';
end
end
```

Exercise 13

Ubbi dubbi is a language game spoken in the English language. Ubbi dubbi works by adding -ub- before each group of vowels. Write a function named ubbidubbi that receives a sentence and returns its “translation” to Ubbi Dubbi.

Note: When adding -ub- before a vowel in uppercase, Ub will now be in upper case, and the vowel will be in lower case.

Example of execution:

```
>> ubbidubbi ('Does Sheldon know I''m lying?')

ans =

     Duboes Shubeldubon knubow Ubi'm lyubing?
```

As you can see in the example, for the group of vowels oe, ‘ub’ only needs to be added to the o.

SOLUTION

```
function [stout] = ubbidubbi(stin)
stout = [];
bVowel = 0;
for i=1:length(stin)
switch stin(i)
    case {'a','e','i','o','u'}
        if bVowel == 0
            bVowel = 1;
            stout = [stout 'ub' stin(i)];
        else
            stout = [stout stin(i)];
        end
    case {'A','E','I','O','U'}
        if bVowel == 0
            bVowel = 1;
            stout = [stout 'Ub' lower(stin(i))];
        else
            stout = [stout stin(i)];
        end
    otherwise
        stout = [stout stin(i)];
        bVowel = 0;
end
```

```
end
end
```

Exercise 14

Write a function *smallestWord* that receives a string and returns the smallest word in that string.

Note: You can assume that there is going to be a word that is smaller than the rest.

Example of execution:

```
>> smallestWord('The most important example is trust')

ans =
      'is'
```

SOLUTION

```
function[ smallest ] = smallestWord( sentence )
% initially the smallest word is the first one
[smallest,remain] = strtok(sentence,' ');
% then, we check the other words
sentence = remain;
while isempty(sentence)== 0
    [word,remain] = strtok(sentence, ' ');
    if(length(word)<length(smallest))
        smallest = word;
    end
    sentence = remain;
end
end
```

Exercise 15

Write a function *reverseCase* that receives a string and replaces lowercase characters by uppercase and vice-versa, and inverts the order of the words.

Example of execution:

```
>> reverseCase('A CAT is An aniMAL')

ans =
      'ANImal aN IS cat a'
```

SOLUTION

```
function[ sentence_out ] = reverseCase( sentence_in )
sentence_out ='';
while isempty(sentence_in)==0
%split text into words
    [word_in,sentence_in] = strtok(sentence_in,' ');
    word_out = '';
    for char = word_in
        % if it's uppercase
```

```
        if char == upper(char)
            % add in lower case
            word_out = [word_out lower(char)];
        else
            % add in upper case
            word_out = [word_out upper(char)];
        end
    end
    sentence_out = [word_out ' ' sentence_out];
    % add word at start instead of end
end
end
```

Exercise 16

Write a function *countNumbers* that receives a string and returns how many numbers there are in that sentence. Note that numbers can have decimal and thousands separators, in which dots separate decimals, and commas separate thousands.

Example of execution:

```
>> countNumbers('I went shopping and bought 3 apples and 12 eggs
for a total of 3.12 euros')

ans =
      3
```

SOLUTION

```
function[count] = countNumbers(sentence)
count = 0;
lastWasNum = 0;
for char = sentence
    switch char
        case {'0','1','2','3','4','5','6','7','8','9'}
            if lastWasNum == 0
                count = count+1;
                lastWasNum = 1;
            end
        otherwise
            if char ~= '.' && char ~= ','
                lastWasNum = 0;
            end
    end
end
end
```

Exercise 17

Write a function *frequentChars* that receives a string and returns the most and least frequent characters in that string.

Example of execution:

```
>> [max,min] = frequentChars('A cat is an animal')
```

```
max =
     'a'
min =
     'c'
```

SOLUTION

```
function[ max_char,min_char ] = frequentChars( sentence_in )
sentence_in = lower(sentence_in);
sentence_in = strrep(sentence_in,' ','');
max_char = '';
max_count = 0;
min_char = '';
min_count = 0;
% for each character of the sentence
for char=sentence_in
    % count how many times that character appears
    count = 0;
    for char2=sentence_in
        if char==char2
            count = count+1;
        end
    end
    % check whether count is greater that the maximum
    if isempty(max_char) || count>max_count
        max_count = count;
        max_char = char;
    elseif isempty(min_char) || count<min_count
        min_count = count;
        min_char = char;
    end
end
end
```

Exercise 18

Write a function *reverseWords* that receives a string and prints the sentence on screen, reversing each of the words (without using the reverse function in Matlab).

Example of execution:

```
>> reverseWords('A cat is an animal')

ans =

      'A tac si na lamina'
```

SOLUTION

```
function[outstring] = reverseWords(instring)
outstring = [];
while(isempty(instring) == 0)
    [word, instring] = strtok(instring);
    lastl = length(word);
    wordr = word;
    for i = 1:floor(lastl/2)
        word(i) = word(lastl - i + 1);
        word(lastl - i + 1) = wordr(i);
    end
    outstring = [outstring ' ' word];
end
end
```

Exercise 19

Write a function *vowelRepeat* that receives a string and checks which words in this string contain vowels that are repeated. The function will print the start position of these words in the sentence on screen.

Example of execution:

```
>> vowelRepeat('The book is on the armchair')

ans =

      5 20
```

SOLUTION

```
function[vectwords] = vowelRepeat(instring)
count = 0;
vowels = ['a' 'e' 'i' 'o' 'u' 'A' 'E' 'I' 'O' 'U'];
vectwords = [];
sentence = instring;
while(isempty(instring) == 0)
    [word, instring] = strtok(instring);
    for i = vowels
        vpos = strfind(word, i);
        if(length(vpos) > 1)
```

```
            count = count + 1;
            vectwords(count) = strfind(sentence, word);
        end
    end
end
end
```

Exercise 20

Write a function *wordRepeat* that receives a string of words and returns the words on screen that are not repeated.

Example of execution:

```
>> wordRepeat('book table book car book arm')

ans =

     'table car arm'
```

SOLUTION

```
function[outstring] = wordRepeat(instring)
vpos = [];
outstring = [];
sentence = instring;
while(isempty(instring) == 0)
    [word, instring] = strtok(instring);
    vpos = strfind(sentence, word);
    if(length(vpos) < 2)
        outstring = [outstring ' ' word];
    end
end
end
```

Exercise 21

Write a function *wordsInSentence* that receives a sentence and prints on screen the words in the sentence, and returns the number of words found. Words in the sentence might be separated by spaces, points or semicolons (;).

Example of execution:

```
>> wordsInSentence ('I found a book, a cat and a pen.
Nothing else')

ans =
      I
      found
      a
      book
      a
      cat
      and
      a
```

```
pen
nothing
else
11
```

SOLUTION

```
function [wcount] = wordsInSentence(st1)
wcount=0;
i= 1;
while i<= length(st1)
    word = '';
    bStop = false;
    while (st1(i) ~=' ') && (st1(i) ~=',') && (st1(i) ~='.') && (st1(i)
~=';') && (i< length(st1))
        word = [word st1(i)];
        i = i +1;
    end
    if i == length(st1)
        word = [word st1(i)];
    end
    fprintf('%s\n', word);
    wcount = wcount + 1;
    i =i + 1;
end
fprintf('%d\n', wcount);
end
```

Exercise 22

Write a function that receives two strings as parameters and returns a string with the words that appear in the two strings.

Example of execution:

```
>> repeatedWords('cat dog duck squirrel frog tiger', 'cat frog
table duck')

ans =

    'cat duck frog'
```

SOLUTION

```
function [outst] = repeatedWords(st1, st2)
outst = [];
while (isempty(st1) == 0)
    [word1, st1] = strtok(st1);
    aux = st2;
    bFound = 0;
    while (isempty(aux) == 0) && (bFound == 0)
        [word2, aux] = strtok(aux);
        if (strcmpi(word1, word2) == 1)
            bFound = 1;
        end
    end
```

```
    if bFound == 1
        outst = [outst ' ' word1];
    end
end
end
```

Exercise 23

Write a function *mixWords* that receives two strings and returns another string that contains the words of the two strings intermixed, so that words that occupied the same position in each string (first word, second word, etc) appear one after the other, the shortest one first.

Example of execution:

```
mixWords('cat dog duck squirrel frog tiger', 'cat frog table
duck')

ans =
    'cat cat dog frog duck table duck squirrel frog tiger'
```

SOLUTION

```
function [outst] = mixWords(st1, st2)
outst = [];
while (isempty(st1) == 0) && (isempty(st2) == 0)
    [word1, st1] = strtok(st1);
    [word2, st2] = strtok(st2);
    l1 = length(word1);
    l2 = length(word2);
    if l2 >= l1
        outst = [outst ' ' word1 ' ' word2];
    else
        outst = [outst ' ' word2 ' ' word1];
    end
end
if isempty(st1) == 0
    outst = [outst st1];
end
if isempty(st2) == 0
    outst = [outst st2];
end
outst(1) = [];
end
```