NTNU
Kunnskap for en bedre verden

Department of Computer and Information Science

Final Examination in TDT4105 “Information Technology, Introduction”, with Matlab

Contact during the exam:	Rune Sætre	Mobile: 452 18103
	Anders Christensen	Mobile: 918 97181
	Terje Rydland	Mobile: 957 73463
	Benjamin A. Bjørnseth	Mobile: 478 32483

Exam date:	**2015-12-16**
Exam time (from-to):	**09:00 – 13:00**
Allowed aids:	**Specified simple calculator**

Other information:

The exam contains 4 problems. A percentage score is given to show how much each problem and sub-problem counts when the exams are graded. Read through all the problems before you start solving them. Be smart and make good use of your time! If you feel the problems are not fully specified, please write your assumptions explicitly.

Answer briefly and clearly, and write so that the text is easy to read. If the text is ambiguous or longer than necessary, points will be deducted.

Language:	**English**
Number of pages:	**19 (including front-page, forms and appendix)**

Contents:

- Problem 1: Multiple Choice Questions (25%)
- Problem 2: Understanding Code (15%)
- Program 3: Programming Travel (20%)
- Problem 4: Programming Evaluation (40%)
- Appendix: Useful functions
- Forms for answering multiple choice questions (2 forms)

Controlled by:

9.des. 2015	Alf Inge Wang
Date	Sign

Problem 1: Multiple Choice Questions (25%)

Use the two enclosed forms to solve this exercise (take one home). You can get a new form if you need it. Only one answer is completely correct. For each question, a correct answer counts 1 point. Wrong answer or more than one answer counts -1/2 point. No answer counts 0 points. You get no less than 0 points total for this problem.

1. What does Random Access Memory mean?
    a. It is random where the computer stores information.
    b. Memory cells can be accessed directly in random order.
    c. The memory is located at various random locations at the motherboard.
    d. That random errors can occur in parts of the memory.
2. When is photolithography used in the production of computers?
    a. When the names of the ports are etched at the back of the computer.
    b. Under production of integrated circuits.
    c. In the creation of pictures for the user manual.
    d. When integrated circuits are fixed to the circuit cards.
3. What is “pipelining”?
    a. A term for what happens if too much data is simultaneously written to the hard drive.
    b. A technique where data is sent between different parts of the computer in “pipes”.
    c. A technique where the CPU can execute several instructions in parallel.
    d. A technique that acts as a “secure tunnel” between your computer and a server.
4. What detects all burst errors with length n bits using an n-bit mask, but cannot be used for cryptography?
    a. Simple checksum.
    b. HASH-functions.
    c. Parity.
    d. Cyclic Redundancy Check (CRC).
5. In the TCP/IP protocol…
    a. all the packets will follow the same route to the receiver.
    b. packet switching is used.
    c. nothing is received until the last IP-packet has arrived.
    d. there is less interference as smaller pieces are sent separately.
6. What is the binary number `10101010` in decimal?
    a. 170
    b. 180
    c. 190
    d. 200
7. Which of the following RGB-codings produces blue?
    a. `f1faf0`
    b. `120012`
    c. `0000ff`
    d. `884311`

8. We have a list of names, such as **list = { 'Jo Å', 'Geir Li', 'Ine By' }** but with many more names in practice. The list is not sorted and can contain duplicates (e.g. the same name can be present at several locations in the list). We shall write a function **antall (list, name)** which will return an integer which represents how many times a name is found in the list. We have the following sketch for the pseudo code:

```
function antall (list, name):
    amount ← 0
    let n go from first to last element in a list:
        if n == name:
            amount ←amount + 1
    return amount
```

   Question: The run-time complexity to the pseudo code above will be?
   a. Θ(n)
   b. Θ(n log n)
   c. Θ(log n)
   d. Θ(n^2)

9. For a sorted list we could have used a binary search instead for the loop **"let n go… "** in the pseudo code from question 8. An alternative algorithm which first sorts the list, and then uses binary search for finding the names will have…
   a. lower complexity (meaning faster) than the pseudo code given above.
   b. higher complexity than the pseudo code above.
   c. the same complexity as the pseudo code above.
   d. higher complexity if the name can be found zero or one time in the list, and lower if the name can be found several times.

10. The function **antall** for an unsorted list as given in question 8, can in Matlab be implemented by using the built-in functions **sum** and **strcmp**, which makes it possible to write the code for the body of the function in one single line of code. For example

```
function ant = antall (list, name)
    ant = sum( strcmp( list, name) )
```

   Question: What is the run-time complexity of this code?
   a. Θ(1)
   b. Θ(log n)
   c. Θ(n)
   d. Θ(n^2)

11. What are the benefits of using an SSD instead of an ordinary magnetic hard drive?
   a. An SSD increases the memory of the graphics card to produce a smoother execution of games and similar programs.
   b. In an SSD the data is stored in electronic circuits. There are no movable parts, and thus the disk will be faster and more reliable.
   c. It is easier to store permanent data on an SSD.
   d. An SSD is not as sensitive to current peaks and thus is more sustainable than a magnetic disk.

12. What is the motivation for the discipline software engineering?
   a. Faster code.
   b. Develop software with best quality independent of budget and time.
   c. Give the foundation for all software to be developed in specific phases in sequence.
   d. Develop software with sufficiently good quality within time and budget.

13. What does modulation describe in e.g. FM and AM?
    a. It describes how a signal can be sent over a carrier wave.
    b. It describes how to get an overview of the whole Internet.
    c. It describes how the power of the electricity can be increase to give more access.
    d. It describes how the Internet can be grouped in reasonable parts.
14. If the text string 'Hallo' in ASCII is represented by the following in hexadecimal: `48 61 6c 6c 6f`, how will 'Morna' be represented?
    a. `4e 65 69 64 61`
    b. `4e 54 4e 55 21`
    c. `4d 6f 72 6e 61`
    d. `55 66 6g 7h 61`
15. An advantage of the waterfall model can be:
    a. Easier to deal with instantaneous requirements from customers.
    b. Easier for a project manager to follow progress according to a project plan.
    c. The system will reflect a gradually better understanding of the user's needs.
    d. Results in faster delivery and shorter time to start using working parts of the system.
16. How many bytes are needed to store a 24-bits picture with 1280x1024 pixels without compression?
    a. Ca. 3,8MB
    b. Ca. 1,2MB
    c. Ca. 24MB
    d. Ca. 24GB
17. What is the first activity in the requirement engineering process according to the textbook?
    a. Feasibility Study.
    b. Requirements Elicitation and Analysis.
    c. Requirements Specification.
    d. Requirements Validation.
18. What is an Acceptance Test?
    a. Test how different parts of the system will interact?
    b. Test that every function in the system works as they should.
    c. Test that the operating system accepts the system on the platform.
    d. Test with user data to check that the system meets the customers' needs.
19. Which of the following techniques are lossless compression?
    a. Run-length encoding.
    b. Analog-to-digital conversion.
    c. JPEG-encoding.
    d. Check-sum generation.
20. What does Boehm's spiral model contain, that is not found in the Waterfall model or incremental development?
    a. Risk analysis.
    b. Testing/Validation.
    c. Requirement specification.
    d. Maintenance.

Problem 2 Understanding Code (15%)

Problem 2a (5%)

What is printed to the screen when the function **main** in the code shown below is executed? (3%)
Explain with one sentence what the function **`mystery`** does (2%)

```
function main()
A='SUNEAILSUN';
B='JALTNCSAES';
D=mystery(A,B);
disp(D);
end

function z = mystery(x,y)
    z='';
    for i = 1:length(x)
        if  mod(i,2)
            z(i) = y(i);
        else
            z(i) = x(i);
        end %if
    end %for
end %function
```

Problem 2b (5%)

What will be returned if the code **`compute(1)`** as shown below is executed? (3%)
Explain with one sentence what the function **`compute`** does (2%)

```
function c = compute(x)
    if x<10
        c = x*compute(x*2);
    else
        c = 1;
    end
end %function
```

Problem 2c (5%)

What will be returned if the code **`a([2,5,3,8,6,1,7])`** as shown below is executed? (3%)
Explain with one sentence what the function **`a`** does (2%)

```
function c = a( c )
   for b=1:length(c)
      d = b ;
      for e=b+1:length(c)
         if c(e)>c(d)
            d = e ;
         end
      end
      f = c(b) ;
      c(b) = c(d) ;
      c(d) = f ;
   end
end
```

Problem 3 Programming Travel (20%)

Write the functions so they can be reused. You can reuse functions from previous problems even though you are not able to solve them.

Problem 3a (5%)

Create the function **readTime** (no input parameters) which asks the user to write in the time using hour, minute and second separately as shown in the example below. The function should ensure that the user enters valid time (you can assume the the user enters numbers and not text). It must give error message for wrong user input and ask the user again to enter hour, minute or second if they are incorrect. The function shall return a list with three elements [hour, minute, sec].

Example of execution and what is printed to the screen (user input is shown with underlining and bold text):

```
>> time = readTime()
Enter hour: 24
- ERROR: Hour must be between 0 and 23!
Enter hour: 12
Enter minute: -5
- ERROR: Minute must be between 0 and 59!
Enter minute: 34
Enter second: 65
- ERROR: Second must be between 0 and 59!
Enter second: 20
time =
    12, 34, 20
>>
```

Problem 3b (5%)

Create the function **convertTime** which takes two input parameters **time** and **mode**. The parameter **mode** can have two values: ’time’ or ’sec’. If mode has the value ‘time’, the function shall convert time given in seconds to time given as a vector formatted as [hour, min, sec] and return the vector. If mode has the value ‘sec’, the function shall return number of seconds a vector in the format [hour, min, sec] corresponds to. The parameter **time** can both be an integer as well as a vector of integers depending of the value of **mode**.

Example of execution:

```
>> convertTime(3857, 'time')
ans =
     1     4    17
>> time = convertTime(3857,'time')
time =
     1     4    17
>> time = convertTime([1,4,17],'sec')
time =
        3857
>>
```

Problem 3c (5%)
Create the function **travelTime** that does not have any input or output. The function will first ask the user about a start time and then an end time for a travel. The function must ensure that a valid time will be entered. If the user tries to enter an end time earlier than the start time, the function shall print the following error message:*"- ERROR: Arrival time must be later than Departure time"*, as well as ask the user to enter another end time. The function does not need to take travels that last after midnight into account. The function shall not return anything, but print the travel time given in hours, minutes and seconds to screen as shown below.

Example of execution (user input is underlined and bold):

```
>> travelTime()
Give departure time in hour, minute and second:
Enter hour: 26
- ERROR: Hour must be between 0 and 23!
Enter hour: 15
Enter minute: 20
Enter second: 20
Give arrival time in hour, minute and second:
Enter hour: 13
Enter minute: 15
Enter second: 39
- ERROR: Arrival time must be later than Departure time
Give arrival time in hour, minute and second:
Enter hour: 18
Enter minute: 59
Enter second: 59
Traveltime: 3 hr, 39 min, 39 sec
>>
```

Problem 3d (5%)
Create the function **analyzeBusRoutes** which has one input parameter **BusTables**, which is a two-dimensional table of integers where each row contains the following: number of bus route, start time (given by hour and minute), and end time (given by hour and minute). The function will no return anything, but will print to the screen the number and travel time for the bus route that takes the longest time, as well as the number and travel time for the bus route that takes the shortest time as shown in the example of execution below. If there are more than one bus route with the same travel time, the function shall print the first it finds.

If the function is executed with the following information:

- Bus nr. 1, Start time 15:00, End time: 15:19
- Bus nr. 3, Start time 15:32, End time: 16:45
- Bus nr. 4, Start time 15:45, End time: 16:23
- Bus nr. 5, Start time 15:55, End time: 16:11

the execution will be as follow:

```
>> busses=[1,15,0,15,19;
       3,15,32,16,45;
       4,15,45,16,23;
       5,15,55,16,11];
>> analyzeBusRoutes(busses)
The slowest bus route is bus nr. 3 and it takes 1 hour, 13 min.
The fastest bus route is bus nr. 5 and it takes 0 hour, 16 min.
>>
```

Problem 4 Programming Evaluation (40%)

Every year about 2000 students takes the ITGK exam, and a lot of time is used to evaluate and award grades. Thus, it is decided to program a system that will help doing the job. To ensure that other universities and courses can use the program, the program must be generalized.

You should create a system to register evaluation of exams. An exam normally consists of four problems where each counts 25%, 15%, 20% and 40% of the grade. Problem two has three sub-problems, problem three has four, and problem four has eight sub-problems. The sub-problems are weighted so each counts an amount percentage of hundred.

Re-use previous sub-problem whenever possible. You can assume that all functions receive valid arguments (inputs) if nothing else is given.

Problem 4 a) (5%)

Create the start of the main program **grades**. The start should only contain the constants (variables) the program needs.

Example on running the program and what it prints to the screen:

```
>> grades
NTNU_scores =
    89    77    65    53    41     0
NTNU_letters =
    'A'    'B'    'C'    'D'    'E'    'F'
TASKS =
  Columns 1 through 8
    '1'    '2a'    '2b'    '2c'    '3a'    '3b'    '3c'    '3d'
  Columns 9 through 16
    '4a'    '4b'    '4c'    '4d'    '4e'    '4f'    '4g'    '4h'
WEIGHTS =
  Columns 1 through 12
    25     5     5     5     5     5     5     5     5     5     5     5
  Columns 13 through 16
     5     5     5     5
```

Problem 4 b) (5%)

Create the function **`makeArray`** with two input parameters (**`numbers`** and **`texts`**), where **numbers** is a list of numbers, and **texts** is a list of characters (both lists have the same length). The function shall return a CellArray which contains all information from the two input parameters as shown in the example of execution below:

Example of execution of the function and what it will return (with input from Problem 4a)):

```
>> limitLetters = makeArray( NTNU_scores, NTNU_letters )
limitLetters =
    [89]    'A'
    [77]    'B'
    [65]    'C'
    [53]    'D'
    [41]    'E'
    [ 0]    'F'
```

```
>> weightTasks = makeArray( WEIGHTS, TASKS )
weightTasks =
    [25]    '1'
    [ 5]    '2a'
    [ 5]    '2b'
... Skipping some lines here ... Vi hopper over noen linjer her ...
    [ 5]    '4g'
    [ 5]    '4h'
```

Problem 4 c) (5%)

A sensor evaluates an exam and assign points to each problem with a score from 0 to 10, where 0 is the worst (0% score) and 10 is the best (100% score). The evaluation of an exam consists of a vector with one number for each sub-problem, in the same sequence as they come (e.g. `1`,`2a`,`2b`, etc.)

Create the function **computeScore** with the input parameters **Points** which is a vector with scores for sub-problems, and the constant **WEIGHTS** from Problem 4 a). The function shall compute a total score in percentage for one exam, based on the weighting of the problems.
Example on execution of the function and what it returns:

```
>> computeScore([10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10], WEIGHTS)
ans =
   100
>> computeScore([10 0 0 0 10 10 10 10 0 0 0 0 0 0 0 0], WEIGHTS)
ans =
    45
>> computeScore([5 0 0 0 10 10 10 10 0 0 0 0 0 0 0 0], WEIGHTS)
ans =
   32.5000
>> computeScore([4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4], WEIGHTS)
ans =
    40
```

Problem 4 d) (5%)

Write the function **score2Letter** with two input parameters **scoreSum** (percent-vice total score for a candidate from Problem 4 c)) and **limitLetters**-tabellen from Problem 4b).
The function shall return the highest corresponding letter grade (without rounding). For example, will 40.9 points give F, while 41.0 points will give E.
Example of execution of the function and what it will return:

```
>> score2letter( 88.9, limitLetters )
ans =
B
```

Problem 4 e) (5%)

Write the function **addCandidate** which has three input parameters: **candidateNumber**, **Scores** (a list), and the constant **WEIGHTS** from Problem 4a). The function shall add the following (separated with tab) as a line at the end of a file with the name '**eksamen.txt**':

candidateNumber comes first in the line, then all the part-scores from **Scores**, and at the end the percent-vise score with one decimal accuracy and a new line. If the file does not exist, it must be created. If it cannot be created, the function should end with an error message.

Example of execution of the function and what it returns:

```
>> addCandidate(12392, [10 0  0  0 10 10 10 10 10 0 0 0 0 0 0 0], WEIGHTS)
>> addCandidate(33322, [0 10 10 10 0 0 0 0 0 10 10 10 10 10 10 10], WEIGHTS)

% Etter at minnepinnen som inneholder filen er fjernet ...
>> addCandidate(12492, [0 10 10 10 0 0 0 0 0 10 10 10 10 10 10 10], WEIGHTS)
Error using addCandidate (line 4)
Kan ikke åpne filen eksamen.txt
>>

% Innholdet i filen eksamen.txt på minnepinnen er nå:
12392 10    0     0     0     10    10    10    10    10    0     ...   0     50.0
33322 0     10    10    10    0     0     0     0     0     10    ...   10    50.0
```

Problem 4 f) (5%)

Write the function **readResultFile** with one input parameter **filename**. The function shall read the content of the file with the name **filename** which is formatted as described in Problem 4 e) and put the content in a two-dimensional table where each row contains a candidate number, all sub-scores, and the percent-vise score. The candidate number and sub-scores shall be integers, while the percent-vise score shall be a float number. You are not allowed to use the load-function in Matlab for this sub-problem. If the file does not exist, the function should be stopped with an error message.

The file eksamen2.txt has the following content where the numbers are separated with tabs

```
12300   0   10  10  10  0   0   0   0   0   10  10  10  10  10  10  10 50.0
44400   4   4   11  0   0   0   0   0   0   0   0   0   0   0   0   0  19.0
12300   9   0   0   0   10  10  10  10  10  0   0   0   0   0   0   0  47.5
```

Example of execution with the file “eksamen2.txt”:

```
>> readResultFile('eksamen2.tekst')
Error using readResultFile (line 4)
Kunne ikke åpne fila
>>
>> readResultFile('eksamen2.txt')
ans =
  Columns 1 through 6
        12300            0           10           10           10            0
        44400            4            4           11            0            0
        12300            9            0            0            0           10
  Columns 7 through 12
            0            0            0            0           10           10
            0            0            0            0            0            0
           10           10           10           10            0            0
  Columns 13 through 18
           10           10           10           10           10           50
            0            0            0            0            0           19
            0            0            0            0            0         47.5
>>
```

Problem 4 g) (5%)

Write the function **checkResultOK** with two input parameters **filename** (the name of the file with exam results to be checked) and **WEIGHTS** (from Problem 4a)). The function will read the file and return **true** if the following is fulfilled:

- no candidate is listed more than once in the same file
- no one has got less than 0 or more than 10 points on any problem
- the percent-vise score for all candidates is correctly computed as in Problem 4c) above

The function shall print error message if it discovers any errors.

Example of execution of the function that shows what it prints and what is returns:

```
>> checkResultOK( 'eksamen.txt', WEIGHTS )
ans =
     1
```

```
>> checkResultOK( 'eksamen2.txt' , WEIGHTS )
ERROR: Candidate 44400 scores are not between 0-10!
ERROR: Candidate 44400 has wrong total score!
ERROR: Candidate 12300 appears more than once!
ans =
     0
```

Problem 4 h) (5%)

Write the function **listAll** with the input parameters **filename** and **limitLetters**. The function shall read the results from the file **filename** formatted as in Problem 4e), and print to screen a list where every line contains a candidate number with five digits, the percent-vise score with one digit accuracy, and the corresponding letter grade according to the rules from Problem 4d). The columns should be adjusted as shown below:

```
10004   7.4 F
10000  75.4 C
10098  87.4 B
10008  88.1 B
10017  94.9 A
10019 100.0 A
```

The print-out must be sorted by the increasing percent-vise score. The return value of the function shall be the number of candidates printed to the screen.

An example of execution of the function, what it prints and what it returns:

```
>> listAll('eksamen2.txt', limitLetters)
44400  19.0 F
12300  47.5 E
12300  50.0 E
ans =
     3
```

Appendix: Some useful functions

FIX Round towards zero.
FIX(X) rounds the elements of X to the nearest integers towards zero.

FLOOR Round towards minus infinity.
FLOOR(X) rounds the elements of X to the nearest integers towards minus infinity.

FCLOSE Close file.
ST = FCLOSE(FID) closes the file associated with file identifier FID, which is an integer value obtained from an earlier call to FOPEN. FCLOSE returns 0 if successful or -1 if not.

FEOF Test for end-of-file.
ST = FEOF(FID) returns 1 if the end-of-file indicator for the file with file identifier FID has been set, and 0 otherwise.
The end-of-file indicator is set when a read operation on the file associated with the FID attempts to read past the end of the file.

FGETL Read line from file, discard newline character.
TLINE = FGETL(FID) returns the next line of a file associated with file identifier FID as a MATLAB string. The line terminator is NOT included. Use FGETS to get the next line with the line terminator INCLUDED. If just an end-of-file is encountered, -1 is returned.

FIND Returns the linear indexes of non-zero elements in a matrix.
FIND([0 1 0 1 0]) returns [2 4]. If the first parameter has more than one row, a column vector containing the linear indexes of non-zero elements are returned. An optional second parameter set the maximum number of indexes to return.

FOPEN Open file.
FID = FOPEN(FILENAME,PERMISSION) opens the file FILENAME in the mode specified by PERMISSION:

'r'	open file for reading
'w'	open file for writing; discard existing contents
'a'	open or create file for writing; append data to end of file
'r+'	open (do not create) file for reading and writing
'w+'	open or create file for reading and writing; discard existing contents
'a+'	open or create file for reading and writing; append data to end of file

FPRINTF Write formatted data to file.
COUNT = FPRINTF(FID,FORMAT,A,...) formats the data in the real part of array A (and in any additional array arguments), under control of the specified FORMAT string, and writes it to the file associated with file identifier FID. COUNT is the number of bytes successfully written. FID is an integer file identifier obtained from FOPEN. It can also be 1 for standard output (the screen) or 2 for standard error. If FID is omitted, output goes to the screen.

FORMAT is a string containing ordinary characters and/or C language conversion specifications. Conversion specifications involve the character %, optional flags, optional width and precision fields, optional subtype specifier, and conversion characters d, i, o, u, x, X, f, e, E, g, G, c, and s.

The special formats \n,\r,\t,\b,\f can be used to produce linefeed, carriage return, tab, backspace, and formfeed characters respectively. Use \\ to produce a backslash character and %% to produce the percent character.

INPUT Read a value from the keyboard and into a variable
ANSWER=INPUT(STR) prints STR as a prompt, reads a number and assigns it to ANSWER. If character string are to be read, use the optional second parameter ‘s’.

ISEMPTY - Determine whether array is empty
This MATLAB function returns logical 1 (true) if A is an empty array and logical 0 (false) otherwise.
TF = isempty(A)

LENGTH The length of vector.
LENGTH(X) returns the length of vector X. It is equivalent to MAX(SIZE(X)) for non-empty arrays and 0 for empty ones.

LOAD   Loads data from filename.
load(filename) loads data from filename.
If filename is a MAT-file, then load(filename) loads variables in the MAT-File into the MATLAB® workspace.
If filename is an ASCII file, then load(filename) creates a double-precision array containing data from the file.

MAX finds the highest element in a vector, or the highest element in each column of a matrix.

MIN finds the lowest element in a vector, or the lowest element in each column of a matrix.

MOD Modulus after division.
MOD(x,y) is x - n.*y where n = floor(x./y) if y ~= 0.

RANDI Pseudorandom integers from a uniform discrete distribution.
R = RANDI(IMAX,N) returns an N-by-N matrix containing pseudorandom integer values drawn from the discrete uniform distribution on 1:IMAX.
RANDI(IMAX,M,N) or RANDI(IMAX,[M,N]) returns an M-by-N matrix.

REM Remainder after division.
REM(x,y) is x - n.*y where n = fix(x./y) if y ~= 0.

ROUND Rounds to nearest decimal or integer
Y = round(X) rounds each element of X to the nearest integer. If an element is exactly between two integers, the round function rounds away from zero to the integer with larger magnitude.
Y = round(X,N) rounds to N digits:

SIZE The size of array.
D = SIZE(X), for M-by-N matrix X, returns the two-element row vector
D = [M,N] containing the number of rows and columns in the matrix.

SORTROWS Sort array rows
This MATLAB function sorts the rows of A in ascending order, based on column.
B = sortrows(A)
B = sortrows(A,column)

SQRT Square root.
SQRT(X) is the square root of the elements of X.

SSCANF Extracts values from a string according to a format string. Opposite of FPRINTF.
A=SSCANF(’12/11-2014’,’%d/%d-%d’) returns a column vector containing the values 12, 11, and 2014.

STRSPLIT Splits the first (string) parameter into a cell array of substrings, according to the delimiter string given as the second parameter. STRSPLIT('one, two, three', ', ') results in {'one', 'two', 'three'}. Multiple alternative delimiters can be specified using a cell array as the second parameter.

STRTOK separates the first token of a string from the rest of that string.
[TOKEN, REST]=STRTOK(‘ first second’, DELIM) sets TOKEN to ‘first’ and REST to ‘ second’. The optional parameter DELIM contains a list of delimiter characters – where the space character is default. Any delimiter characters before the first token are ignored.

STR2NUM Convert string matrix to numeric array.
X = STR2NUM(S) converts a character array representation of a matrix of numbers to a numeric matrix. For example, S=['12'; '34']   str2num(S) => [12; 34]

SUM The sum of elements.
S = SUM(X) is the sum of the elements of the vector X. If X is a matrix, S is a row vector with the sum over each column.

This page is on purpose empty!

Answer form for Multiple Choice Questions

Candidate number:	______________________	Program:	______________________
Course code:	______________________	Date:	______________________
Total no of pages:	______________________	Page:	______________________

Problem	***A***	***B***	***C***	***D***
1.1				
1.2				
1.3				
1.4				
1.5				
1.6				
1.7				
1.8				
1.9				
1.10				
1.11				
1.12				
1.13				
1.14				
1.15				
1.16				
1.17				
1.18				
1.19				
1.20				

This page is on purpose empty!

Answer Form for Multiple Choice Questions

Candidate number:	______________________	Program:	______________________
Course code:	______________________	Date:	______________________
Total no of pages:	______________________	Page:	______________________

Problem	***A***	***B***	***C***	***D***
1.1				
1.2				
1.3				
1.4				
1.5				
1.6				
1.7				
1.8				
1.9				
1.10				
1.11				
1.12				
1.13				
1.14				
1.15				
1.16				
1.17				
1.18				
1.19				
1.20				