Middle East Technical University Department of Computer Engineering

# CENG 140

## C Programming

Spring '2019-2020
Take-Home Exam 2

Emre Külah
kulah@ceng.metu.edu.tr
Due date: May 10, 2020, Sunday, 23:59

## 1 Overview

Data processing is, generally, "the collection and manipulation of items of data to produce meaningful information." In this sense, it can be considered a subset of information processing, "the change (processing) of information in any manner detectable by an observer." [1]

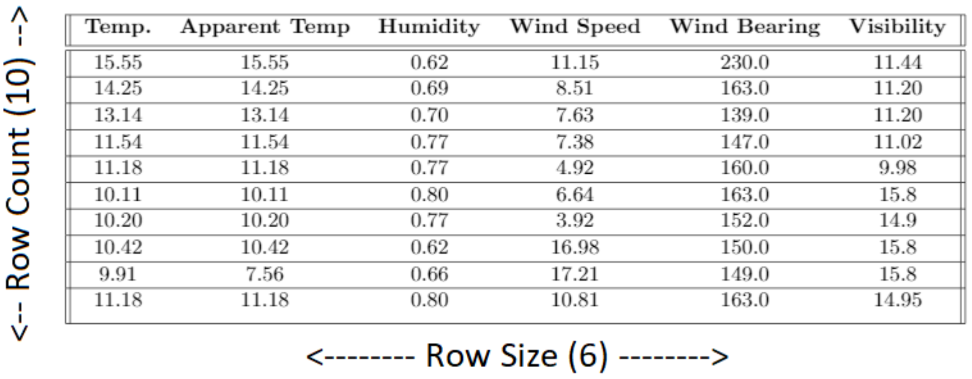
In this assignment, you are expected to implement several data processing methods. These methods include revealing statistical information from the data and manipulating the data to extract new features. There will be sufficient information for each task and to make the description process easy, you will be given a small, simple data. This data as input text and its corresponding outputs for each task will be supplied.

Signatures of the functions are given in *the2.h* header file. Additionally, stub codes are available for the source file.

## 2 Tasks

There are 7 tasks in this assignment. One of them is printing the array, one is the initializing the matrix, and six tasks are about the data processing.

Above all, let's start with a simple, understandable data in a 2D array structure. The data in the following table contains weather information from a couple of sample days. Each column gives different values about the day. In machine learning terminology, each column represents a feature (i.e. temperature, humidity, etc.) and each row called a feature vector. This data is going to be stored in a double double pointer (double **).


&lt;-- Row Count (10) --&gt;
&lt;-------- Row Size (6) --------&gt;

| Temp. | Apparent Temp | Humidity | Wind Speed | Wind Bearing | Visibility |
|---|---|---|---|---|---|
| 15.55 | 15.55 | 0.62 | 11.15 | 230.0 | 11.44 |
| 14.25 | 14.25 | 0.69 | 8.51 | 163.0 | 11.20 |
| 13.14 | 13.14 | 0.70 | 7.63 | 139.0 | 11.20 |
| 11.54 | 11.54 | 0.77 | 7.38 | 147.0 | 11.02 |
| 11.18 | 11.18 | 0.77 | 4.92 | 160.0 | 9.98 |
| 10.11 | 10.11 | 0.80 | 6.64 | 163.0 | 15.8 |
| 10.20 | 10.20 | 0.77 | 3.92 | 152.0 | 14.9 |
| 10.42 | 10.42 | 0.62 | 16.98 | 150.0 | 15.8 |
| 9.91 | 7.56 | 0.66 | 17.21 | 149.0 | 15.8 |
| 11.18 | 11.18 | 0.80 | 10.81 | 163.0 | 14.95 |

Figure 1: Weather information data

- You will implement following seven methods;
  - **double** initialize_the_matrix(int* row_count, int* row_size)**,
  - **void print_first_n_row(double** matrix, int n, int row_size)**,
  - **void calculate_dot_product(double** matrix, int row_size, int row1, int row2)**,
  - **double** calculate_covariance_matrix(double** matrix, int row_count, int row_size)**,
  - **double** calculate_x_transpose_times_x(double** matrix, int row_count, int row_size)**,

  - **double** group_by(double** matrix, int* group_count, int row_count, int row_size, int group_column, int operation)**,
  - **double** convolution(double** matrix, int row_count, int row_size, double** kernel, int kernel_size_height, int kernel_size_width)**.

- **double** initialize_the_matrix(int* row_count, int* row_size)** function gets two "integer pointer" parameters and returns double double pointer. The content of the matrix will be streamed via standard input (stdin) and this method will initialize the matrix. The data will be streamed as following format:

  ```
  15.55 15.55 0.63 11.15 230.0 11.44 -1
  14.25 14.25 0.69 8.51  163.0 11.20 -1
  ...
  ...
  11.18 11.18 0.80 10.81 163.0 14.95 -1
  -1
  ```

  "-1"s represent the end of the rows (vectors) and one more "-1" at the end represents the end of the matrix. While collecting the values, you have to calculate row count (it is 10 for this particular example) and row size (it is 6 for this particular example). These values are going to be assigned to function parameters row_count and row_size respectively.

- **void print_first_n_row(double** matrix, int n, int row_size)** function gets matrix itself, number of the rows to print and row size (vector size). It is going to print first $n$ row of the matrix.

  ```
  int main()
      ...
      /* initialize the weather information matrix */
      print_first_n_row(matrix, 3, row_size);
  ```

  Output will be:

  ```
  15.5500 15.5500 0.6300 11.1500 230.0000 11.4400
  14.2500 14.2500 0.6900 8.5100 163.0000 11.2000
  13.1400 13.1400 0.7000 7.6300 139.0000 11.2000
  ```

  **NOTE: There will be no extra whitespace at the end of the lines.**

- **void calculate_dot_product(double** matrix, int row_size, int row1, int row2)** method gets the matrix, row size and desired vector indexes to calculate the dot product. It prints the dot product as follows:

  ```
  Dot product of rows <row1>, <row2>: <dot_product>
  ```

  Dot product is used to calculate the angle between two vectors [2]. You can calculate dot product using following equation:

  $$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} (a_i \times b_i); \tag{1}$$

  where n=6 for weather information data.

  ```
  int main()
      ...
      /* initialize the weather information matrix */
      calculate_dot_product(matrix, row_size, 4, 6)
      /* row#4 -> 11.18 11.18 0.77 4.92 160.0 9.98 */
      /* row#6 -> 10.20 10.20 0.77 3.92 152.0 14.9 */
  ```

  The calculation is as follows:

  $$\vec{row}\#4 \cdot \vec{row}\#6 = 11.18 \times 10.20 + 11.18 \times 11.20 + ... + 9.98 \times 14.9 = 24716.6533 \tag{2}$$

  So output will be:

  ```
  Dot product of rows 4, 6: 24716.6533
  ```

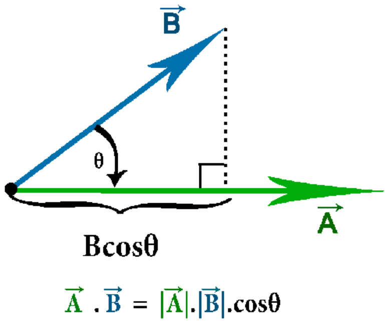


Figure 2: Dot Product

- **double** calculate_covariance_matrix(double** matrix, int row_count, int row_size)** function gets matrix and its dimensions as parameter. It calculates covariance matrix of the data and returns it. Variance means the average of the squared differences from the mean of the column values.

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n} \tag{3}$$

where $\mu$ is mean of a column (i.e. wind speed), n is the sample count (it is 10 for our example). For example, the mean of the temperature column (its column index → 0) is:

$$\mu_0 = \frac{15.55 + 14.25 + 13.14 + 11.54 + 11.18 + 10.11 + 10.20 + 10.42 + 9.91 + 11.18}{10} = 11.748 \tag{4}$$

Now how we calculate the variance for column#0:

$$\sigma_0^2 = \frac{\sum_{i=1}^{N}(X_i - \overline{x}) \times (X_i - \overline{x})}{N-1} \tag{5}$$

where N is the number of rows which is 10, $X_i$ is the value of row#i, and $\overline{x}$ is the mean value which is 11.748 for our example.

Covariance calculation is similar. Instead of multiplying same column values, we multiply two different columns. For example, if we want to calculate covariance between column#0 (temperature) and column#1 (apparent temperature):

$$\sigma_{01}^2 = \frac{\sum_{i=1}^{N}(X_i - \overline{x}) \times (Y_i - \overline{y})}{N-1} \tag{6}$$

where N is the number of rows which is 10, $X_i$ is the value of row#i, and $\overline{x}$ is the mean value of temperature column which is 11.748 for our example. $Y_i$ is the value of row#i, and $\overline{y}$ is the mean value of apparent temperature column which is 11.513 for our example

This $\sigma_{01}^2$ gives us the covariance of the temperature and apparent temperature columns on the data. Covariance is the co-variance between the features which gives the directional relationship between two features, `https://www.geeksforgeeks.org/mathematics-covariance-and-correlation/`.

Covariance matrix $\Sigma$ keeps all of the covariances between all features.

$$\Sigma = \begin{pmatrix} \sigma_{00}^2 & \sigma_{01}^2 & \sigma_{02}^2 & \sigma_{03}^2 & \sigma_{04}^2 & \sigma_{05}^2 \\ \sigma_{10}^2 & \sigma_{11}^2 & \sigma_{12}^2 & \sigma_{13}^2 & \sigma_{14}^2 & \sigma_{15}^2 \\ \sigma_{20}^2 & \sigma_{21}^2 & \sigma_{22}^2 & \sigma_{23}^2 & \sigma_{24}^2 & \sigma_{25}^2 \\ \sigma_{30}^2 & \sigma_{31}^2 & \sigma_{32}^2 & \sigma_{33}^2 & \sigma_{34}^2 & \sigma_{35}^2 \\ \sigma_{40}^2 & \sigma_{41}^2 & \sigma_{42}^2 & \sigma_{43}^2 & \sigma_{44}^2 & \sigma_{45}^2 \\ \sigma_{50}^2 & \sigma_{51}^2 & \sigma_{52}^2 & \sigma_{53}^2 & \sigma_{54}^2 & \sigma_{55}^2 \end{pmatrix}$$

- **double** calculate_x_transpose_times_x(double** matrix, int row_count, int row_size)** function calculates $x^T \times x$ where $x$ is the original matrix and $x^T$ is the transpose of the original matrix. It returns the result of this operation as a double double pointer. The transpose of a matrix is an operator which flips a matrix over its diagonal [3]. The method generates transpose of the matrix and applies matrix multiplication. Be careful about the order of the multiplication since matrix multiplication is not commutative.

$$x^T \times x \neq x \times x^T \tag{7}$$

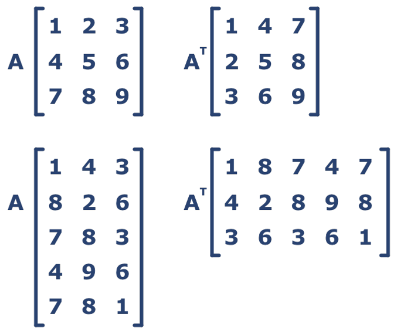


Figure 3: Transpose operation

- **double** group_by(double** matrix, int* group_count, int row_count, int row_size, int group_column, int operation)** function gets matrix, group_count (it is going to be calculated and assigned in this method), dimensions of the matrix, a column index to apply grouping over and an operation index. This function applies a grouping over matrix according to the column index given as a parameter. The function returns grouped matrix.

  This method makes grouping on the matrix according to the given column index. You find out the number of groups exists in the matrix for a given column index, set the group_count parameter to send back this value to the caller function. Rows with the same data in the given column index are grouped and converted into a single row according to the desired operation. There are 4 operations used:

  - SUM (0)
  - AVERAGE (1)
  - MAX (2)
  - MIN (3)

  This operation is applied to all columns except the column index. The grouped matrices that are formed according to all operation types are shown below. Here grouping is done according to the humidity column (column index = 2). Group rows sorted by the first occurrences of the values.

  For example, grouping over humidity column with SUM operation;

```
int main(){
    int group_count;
    ...

    /* group_column = 2 */
    /* operation = 0 */
    group_by(matrix, &group_count, row_count, row_size, 2, 0)
}
```

  Let's find all unique values in the matrix in the order of first occurrences.

| ... | **Humidity** | ... |
|---|---|---|
| ... | 0.62 | ... |
| ... | 0.69 | ... |
| ... | 0.70 | ... |
| ... | 0.77 | ... |
| ... | 0.77 | ... |
| ... | 0.80 | ... |
| ... | 0.77 | ... |
| ... | 0.62 | ... |
| ... | 0.66 | ... |
| ... | 0.80 | ... |

  List of all unique values is [0.62, 0.69, 0.70, 0.77, 0.80, 0.66]. We are going to group all rows according to their humidity value.

| Temp. | Apparent Temp | Humidity | Wind Speed | Wind Bearing | Visibility |
|---|---|---|---|---|---|
| 15.55 | 15.55 | 0.62 | 11.15 | 230.0 | 11.44 |
| 10.42 | 10.42 | 0.62 | 16.98 | 150.0 | 15.8 |
| 14.25 | 14.25 | 0.69 | 8.51 | 163.0 | 11.20 |
| 13.14 | 13.14 | 0.70 | 7.63 | 139.0 | 11.20 |
| 11.54 | 11.54 | 0.77 | 7.38 | 147.0 | 11.02 |
| 11.18 | 11.18 | 0.77 | 4.92 | 160.0 | 9.98 |
| 10.20 | 10.20 | 0.77 | 3.92 | 152.0 | 14.9 |
| 10.11 | 10.11 | 0.80 | 6.64 | 163.0 | 15.8 |
| 11.18 | 11.18 | 0.80 | 10.81 | 163.0 | 14.95 |
| 9.91 | 7.56 | 0.66 | 17.21 | 149.0 | 15.8 |

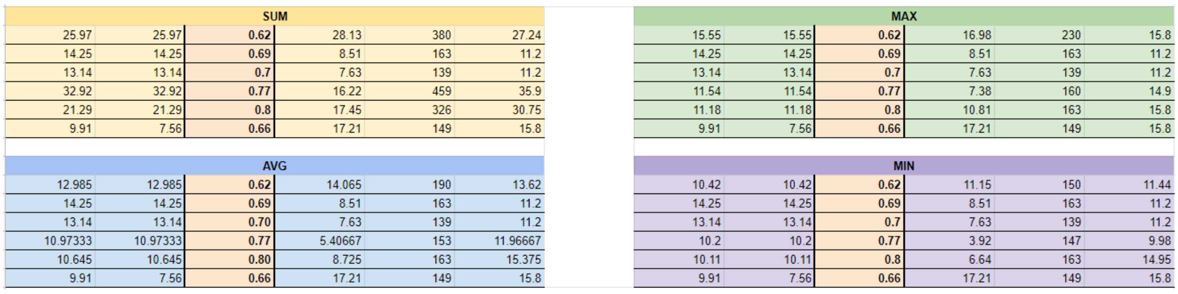
After grouping, we are going to apply given operation SUM to merge all rows in each group except column index we apply grouping over (it is humidity in this example). First group in this data is $humidity = 0.62$:

| Temp. | Apparent Temp | Humidity | Wind Speed | Wind Bearing | Visibility |
|---|---|---|---|---|---|
| 15.55 | 15.55 | 0.62 | 11.15 | 230.0 | 11.44 |
| 10.42 | 10.42 | 0.62 | 16.98 | 150.0 | 15.8 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

If we apply SUM operation on this ($humidity = 0.62$) group, then it will be:

| Temp. | Apparent Temp | Humidity | Wind Speed | Wind Bearing | Visibility |
|---|---|---|---|---|---|
| 25.97 | 25.97 | 0.62 | 28.13 | 380.00 | 27.24 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

As shown, SUM operation is not applied to humidity column.



| SUM | | | | | |
|---|---|---|---|---|---|
| 25.97 | 25.97 | **0.62** | 28.13 | 380 | 27.24 |
| 14.25 | 14.25 | **0.69** | 8.51 | 163 | 11.2 |
| 13.14 | 13.14 | **0.7** | 7.63 | 139 | 11.2 |
| 32.92 | 32.92 | **0.77** | 16.22 | 459 | 35.9 |
| 21.29 | 21.29 | **0.8** | 17.45 | 326 | 30.75 |
| 9.91 | 7.56 | **0.66** | 17.21 | 149 | 15.8 |

| MAX | | | | | |
|---|---|---|---|---|---|
| 15.55 | 15.55 | **0.62** | 16.98 | 230 | 15.8 |
| 14.25 | 14.25 | **0.69** | 8.51 | 163 | 11.2 |
| 13.14 | 13.14 | **0.7** | 7.63 | 139 | 11.2 |
| 11.54 | 11.54 | **0.77** | 7.38 | 160 | 14.9 |
| 11.18 | 11.18 | **0.8** | 10.81 | 163 | 15.8 |
| 9.91 | 7.56 | **0.66** | 17.21 | 149 | 15.8 |

| AVG | | | | | |
|---|---|---|---|---|---|
| 12.985 | 12.985 | **0.62** | 14.065 | 190 | 13.62 |
| 14.25 | 14.25 | **0.69** | 8.51 | 163 | 11.2 |
| 13.14 | 13.14 | **0.70** | 7.63 | 139 | 11.2 |
| 10.97333 | 10.97333 | **0.77** | 5.40667 | 153 | 11.96667 |
| 10.645 | 10.645 | **0.80** | 8.725 | 163 | 15.375 |
| 9.91 | 7.56 | **0.66** | 17.21 | 149 | 15.8 |

| MIN | | | | | |
|---|---|---|---|---|---|
| 10.42 | 10.42 | **0.62** | 11.15 | 150 | 11.44 |
| 14.25 | 14.25 | **0.69** | 8.51 | 163 | 11.2 |
| 13.14 | 13.14 | **0.7** | 7.63 | 139 | 11.2 |
| 10.2 | 10.2 | **0.77** | 3.92 | 147 | 9.98 |
| 10.11 | 10.11 | **0.8** | 6.64 | 163 | 14.95 |
| 9.91 | 7.56 | **0.66** | 17.21 | 149 | 15.8 |

Figure 4: Grouped matrices by column index 2

- **double** convolution(double** matrix, int row_count, int row_size, double** kernel, int kernel_size_height, int kernel_size_width)** function gets matrix, dimensions of the matrix, another smaller matrix and its dimensions. This function applies a convolution operation on the original matrix and returns created convoluted matrix.

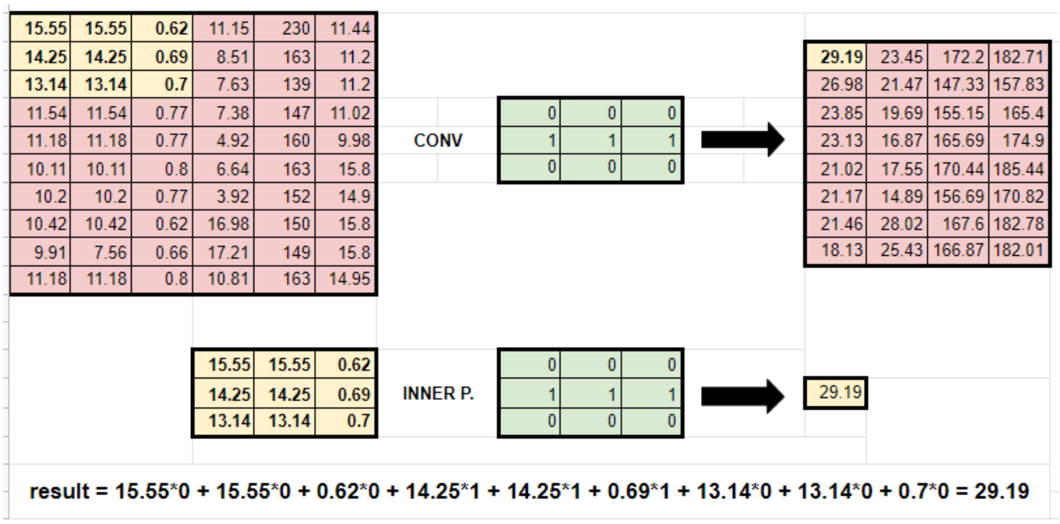
  Convolution is a mathematical operation often used in image processing. An inner product operation is performed by sliding a matrix called kernel (or filter or window) that is smaller than the original matrix, on the original matrix. In this way, a common value of neighboring cells that influence each other is extracted. You can find a good animation on the following link: `https://miro.medium.com/max/1400/1*D6iRfzDkz-sEzyjYoVZ73w.gif`

  Inner product is the generalized version of dot product. It is summation of the multiplication of elements side by side.

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \cdot \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4$$

Figure 5: Inner product

A kernel and its dimensions will be given to the function as parameters. By sliding the given kernel on the original matrix, the output matrix will be calculated and will be returned.



|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 15.55 | 15.55 | 0.62 | 11.15 | 230 | 11.44 |
| 14.25 | 14.25 | 0.69 | 8.51 | 163 | 11.2 |
| 13.14 | 13.14 | 0.7 | 7.63 | 139 | 11.2 |
| 11.54 | 11.54 | 0.77 | 7.38 | 147 | 11.02 |
| 11.18 | 11.18 | 0.77 | 4.92 | 160 | 9.98 |
| 10.11 | 10.11 | 0.8 | 6.64 | 163 | 15.8 |
| 10.2 | 10.2 | 0.77 | 3.92 | 152 | 14.9 |
| 10.42 | 10.42 | 0.62 | 16.98 | 150 | 15.8 |
| 9.91 | 7.56 | 0.66 | 17.21 | 149 | 15.8 |
| 11.18 | 11.18 | 0.8 | 10.81 | 163 | 14.95 |

CONV

|  |  |  |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

|  |  |  |  |
|---|---|---|---|
| 29.19 | 23.45 | 172.2 | 182.71 |
| 26.98 | 21.47 | 147.33 | 157.83 |
| 23.85 | 19.69 | 155.15 | 165.4 |
| 23.13 | 16.87 | 165.69 | 174.9 |
| 21.02 | 17.55 | 170.44 | 185.44 |
| 21.17 | 14.89 | 156.69 | 170.82 |
| 21.46 | 28.02 | 167.6 | 182.78 |
| 18.13 | 25.43 | 166.87 | 182.01 |

|  |  |  |
|---|---|---|
| 15.55 | 15.55 | 0.62 |
| 14.25 | 14.25 | 0.69 |
| 13.14 | 13.14 | 0.7 |

INNER P.

|  |  |  |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

|  |
|---|
| 29.19 |

**result = 15.55*0 + 15.55*0 + 0.62*0 + 14.25*1 + 14.25*1 + 0.69*1 + 13.14*0 + 13.14*0 + 0.7*0 = 29.19**

Figure 6: Convolution operation on a 2D matrix

## 3 Specifications

- The values in the matrices will be in the range $(-999999, 999999)$. These two values are defined in *the2.h* header file as:

  ```
  MIN_VALUE = -999999
  MAX_VALUE = 999999
  ```

- The number of columns of the matrix can be in the range $[1, 20]$ and there can be finitely many rows.
- You do not have to check whether the row or column indexes that are required to perform operations on the functions are valid. This kind of edge cases will not be tried. Example:

  ```
  calculate_dot_product(matrix, row_size, -1, -5)
  ```

- Kernel size in the convolution method will not exceed the original matrix size.
- In print_first_n_row and calculate_dot_product functions, you are going to print double values with 4-digits precision. There will be no extra newline character at the end of these print functions.
- In calculate_x_transpose_times_x function, group rows sorted by the first occurrences of the values.
- In print_first_n_row function, there will be no extra whitespace at the end of the lines and there will be no extra newline character at the end.

## 4 Regulations

- **Programming Language:** C
- **Libraries and Language Elements:**
  You should not use any library other than *"stdio.h"*, *"stdlib.h"* and *"math.h"*. You can use conditional clauses (switch/if/else if/else), loops (for/while), allocation methods (malloc, calloc, realloc). **You can NOT use any further elements beyond that (this is for students who repeat the course).** You can define your own helper functions.
- **Compiling and running:**
  **DO NOT FORGET! YOU WILL USE ANSI-C STANDARTS.** You should be able to compile your codes and run your program with given **Makefile**:

  ```
  >_ make the2
  >_ ./the2
  ```

  **If you are working with ineks or you are working on Ubuntu OS**, you can feed your program with input files instead of typing inputs. This gives the input from stdin and an equivalent of typing inputs. This way you can test your inputs faster:

  ```
  >_ ./the2 < inp1.txt
  >_ ./the2 < inp2.txt
  ```

- **Submission:**
You will use CengClass system for the homework just like Lab Exams. You can use the system as an editor or work locally and upload the source files. Late submission IS NOT allowed, it is not possible to extend the deadline and **please do not ask for any deadline extensions**.
- **Evaluation:** Your codes will be evaluated based on several input files including, but not limited to the test cases given to you as an example. You can check your grade with sample test cases via CengClass system but do not forget it is not your final grade. Your output must give the exact output of the expected outputs. It is your responsibility to check the correctness of the output with the invisible characters. Otherwise, you can not get a grade from that case. If your program gives correct outputs for all cases, you will get 100 points.
- **Cheating: We have zero-tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations and will get 0. Sharing code between each other or using third party code is strictly forbidden. Even if you take a "part" of the code from somewhere/somebody else - this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar. So please do not think you can get away with by changing a code obtained from another source.

# References

[1] Wikipedia contributors. Data processing — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Data_processing`, 2020.

[2] Wikipedia contributors. Dot product — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Dot_product`, 2020.

[3] Wikipedia contributors. Transpose — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Transpose`, 2020.