T.C.
DOKUZ EYLUL UNIVERSITY

FACULTY OF
ENGINEERING

DEPARTMENT OF
COMPUTER ENGINEERING

2025 - 2026
FALL SEMESTER

CME 3205
OPERATING SYSTEMS

ASSIGNMENT
PRICE PREDICTION SERVER

DUE DATE
23:55 – 17.12.2025

In this assignment you are asked to implement a multi threaded Multiple Linear Regression (MLR) system in C that trains on three CSV datasets, performs mandatory min–max normalization on all numeric attributes, encodes categorical attributes as specified, computes model coefficients using multiple threads, and predicts for a new instance by printing both the normalized and reverse-normalized (real-scale) target value. This program will open a port and accept connection from another console using telnet for usage. The original console will be used for program management and debug.

You will be given three small datasets in CSV format:

Housing.csv
Student_Performance.csv
multiple_linear_regression_dataset.csv

You are required to download/extract these files from Sakai assignment page, examine their columns and types, and use them in your program.

Your program must be written in C programming language and it must work on a standard Linux computer. You should create and test inside the same VirtualBox Linux setup used in lab sessions. You must compile using console commands (using gcc as in lab sessions).

You are also required to use the following global variables to make your code more understandable, modifiable and standardized.

```
int PORT_NUMBER = 60000;
```
The port number that will be used by your program. You may need to change it after closing your program, because OS thinks it is still in use by your previous execution.

```
int MAX_SAMPLES = 10000;
```
The maximum number of rows your program will store and process.

```
int MAX_FEATURES = 100;
```
The maximum number of input attributes (Attributes in this case are your columns).

```
int STRING_BUFFER_LIMIT = 100;
```
A safe upper bound for reading token/field strings from CSV and user input (Upper limit for string size of every row value).

```
int PREPROC_THREAD_LIMIT = 128;
```
An upper bound for the number of threads you may spawn for attribute-level preprocessing.

```
int COEFF_THREAD_LIMIT = 128;
```
An upper bound for the number of threads you may spawn for coefficient computation.

These are the basic variables and names you should use in your program. A standard pseudo-workflow of your program is given below.

MULTI THREADED REGRESSION BASIC EXECUTION STEPS:

1. Dataset presence check and load:
At program start, check that all three CSV files exist in the working directory.

If any file is missing, output an error (e.g., ERROR: Dataset file “Housing.csv” not found!) and terminate.

If found, parse the CSV, detect column types (numeric vs categorical) and store the data into memory-resident arrays suitable for numeric processing (doubles for numeric columns; string buffers for raw categorical values).

2. Preprocessing with per-attribute threads (mandatory):
For EACH numeric attribute, create a thread that:

a) Scans the column to find xmin and xmax,
b) Applies min–max normalization: x_norm = (x − xmin) / (xmax − xmin),
c) Stores xmin and xmax for later use during prediction.

For EACH categorical attribute, create a thread that:

a) Scans the column and applies the required encoding rule(s),
b) Writes encoded numeric columns into the design matrix X.

2.1. Housing.csv furnishingstatus encoding (special rule in this assignment):
For a house with furnishingstatus = "semi-furnished":

furnishing_furnished = 2
furnishing_semi_furnished = 1
furnishing_unfurnished = 0

(Apply this rule consistently; do not replace it with generic one-hot unless explicitly stated in the dataset’s instructions.)

After all attribute threads finish, join them and proceed. Use mutexes only where threads write to shared tables (e.g., the global min/max table or shared encoded matrix slices). Otherwise, prefer per-column ownership to avoid unnecessary locking.

3. Model training on normalized data (required):
Train a multiple linear regression model on the normalized feature matrix X_norm and normalized target y_norm. You must use threads for coefficient computation:

Option A (Normal Equations, recommended for small datasets):

Construct X^TX and X^Ty on normalized data.

Create at least one thread per coefficient βj that is responsible for producing the j-th row (or its assigned block) of X^TX and its entry in X^Ty. Each thread writes only to its own block to avoid races; if you aggregate into shared accumulators, protect them with a mutex.

After all threads finish, solve $(X^TX)\ \beta = X^Ty$ in the main thread (e.g., Gaussian elimination or Cholesky). Store β (including intercept β0 if you included a bias column of 1s).

Option B (Iterative, e.g., batch gradient descent):

Create at least one thread per coefficient βj. In each iteration, threads compute their gradient components in parallel; the main thread (or a single updater thread) applies the synchronized β update after a barrier. Repeat until convergence or max iterations.

The printed regression equation must use normalized variables (e.g., price_norm = β0 + β1*area_norm + …).

4. Prediction workflow (interactive):

Ask the user to choose which dataset to use for prediction (one of the three).

Read raw user input values for ALL attributes of the chosen dataset.

Normalize numeric inputs using the stored xmin/xmax of the chosen dataset.

Encode categorical inputs using the same rules used in training.

Compute $\hat{y}\_norm = \beta 0 + \Sigma\ \beta j\ xj\_norm$.

Reverse-normalize the model output to real scale using the target's ymin,ymax from training:
    $y = \hat{y}\_norm * (ymax - ymin) + ymin$

Print BOTH values:
    Predicted normalized target: <value>
    Predicted real target: <value with units if applicable>

5. Output requirements (training and prediction):

After training on normalized data, print the full equation with coefficient names aligned to feature names.

During prediction, echo the normalized feature vector you fed into the model (for debugging), the ŷ_norm, and the reverse-normalized y on the real scale.

6. Error handling:

If a column is constant (xmin == xmax), handle safely:
    Either drop that feature and clearly indicate in output/report,
    Or map all normalized values to 0 and explain in comments.

If categorical input has an unseen label at prediction time, either reject with a clear error message or map to a documented default.

If memory limits (MAX_SAMPLES, MAX_FEATURES) are exceeded, print an informative error and terminate gracefully.

7. Concurrency and synchronization (operating systems focus):

You must use POSIX threads (pthreads) and, where needed, pthread_mutex_t and/or POSIX semaphores to prevent races.

Required threading:
    One thread per numeric attribute (normalization).
    One thread per categorical attribute (encoding).
    At least one thread per coefficient βj for the training phase.

Demonstrate correct joining, absence of data races, and avoidance of deadlocks. Use barriers or thread-counting schemes where iterations or multi-stage work require synchronization.

8. Compilation and environment:

Compile on Linux with:

```
gcc assignment.c -o assignment.o -lpthread
```

(This is an example gcc command that you can use, you can also add additional debug flags to help you debug easier. Search about this online.)

Your program must run correctly under the lab's VirtualBox Linux image.

9. Notes for students (background formulas):

Min–max normalization:

x_norm = (x − x_min) / (x_max − x_min)

Reverse transform for target y uses:

y = y_norm*(y_max − y_min) + y_min

Ordinary Least Squares normal equations:

$(X^TX)\beta = X^Ty$, solution $\beta = (X^TX)^{-1} X^Ty$ (when X^TX is invertible)

POSIX threads, mutexes, and semaphores are the standard C concurrency primitives you will use.

PROGRAM REQUIREMENTS (CHECKLIST):

Read and parse the three CSV files.

Detect numeric vs categorical attributes and build an internal schema.

Preprocess ALL numeric columns with min–max (store per-column x_min/x_max).

Apply the special furnishing status encoding rule for Housing.csv exactly as specified.

Create separate threads for:
- Each numeric attribute's normalization,
- Each categorical attribute's encoding,
- Each coefficient's computation during training.

Train on normalized data and print the normalized equation.

Accept a new raw instance from the user; normalize and encode it.

Compute ŷ_norm and reverse-normalize to print y on the real scale.

Handle errors and edge cases gracefully; exit cleanly.

A sample port/socket communication between your assignment and telnet program is given below. You do now have to follow this exact output, however you should follow the steps and output or input the required information at correct points. The text with yellow background are inputs by user.

```
telnet localhost 60000
```

With this command, you will be connected to port 60000 of your localhost (your own computer). Assuming this is the port that your assignment is currently working on, you should receive the following messages.

```
WELCOME TO PRICE PREDICTION SERVER

Enter CSV file name to load:
Housing.csv

Checking dataset...

[OK] File "Housing.csv" found.
Reading file...
   545 rows loaded.
   13 columns detected.

Column analysis:
   area             : numeric
   bedrooms         : numeric
   bathrooms        : numeric
   stories          : numeric
   parking          : numeric
   mainroad         : categorical (yes/no)
   guestroom        : categorical (yes/no)
   basement         : categorical (yes/no)
   furnishingstatus : categorical (furnished, semi-furnished,
unfurnished)
   price            : numeric (TARGET)

Starting attribute-level categorical encoding...

[Thread C1] mainroad: yes/no → 1/0
[Thread C2] guestroom: yes/no → 1/0
[Thread C3] basement: yes/no → 1/0

[Thread C4] furnishingstatus (SPECIAL RULE):
   furnished        → 2
   semi-furnished   → 1
   unfurnished      → 0

[OK] All categorical encoding threads completed.

Starting numeric normalization threads...
```

```
[Thread N1] Normalizing area... xmin=1650 xmax=16200
[Thread N2] Normalizing bedrooms... xmin=1 xmax=6
[Thread N3] Normalizing bathrooms... xmin=1 xmax=4
[Thread N4] Normalizing stories... xmin=1 xmax=4
[Thread N5] Normalizing parking... xmin=0 xmax=3
[Thread N6] Normalizing price (TARGET)... ymin=1250000
ymax=13300000

[OK] All normalization threads completed.

Building normalized feature matrix X_norm...
Building normalized target vector y_norm...

Spawning coefficient calculation threads...

[β-Thread 1] Calculating β1 (area_norm)...
[β-Thread 2] Calculating β2 (bedrooms_norm)...
[β-Thread 3] Calculating β3 (bathrooms_norm)...
[β-Thread 4] Calculating β4 (stories_norm)...
[β-Thread 5] Calculating β5 (parking_norm)...
[β-Thread 6] Calculating β6 (mainroad)...
[β-Thread 7] Calculating β7 (guestroom)...
[β-Thread 8] Calculating β8 (basement)...
[β-Thread 9] Calculating β9 (furn_furnished)...
[β-Thread 10] Calculating β10 (furn_semifurnished)...
[β-Thread 11] Calculating β11 (furn_unfurnished)...

All coefficient threads joined.
Solving (XᵀX)β = Xᵀy ...

Training completed.

FINAL MODEL (Normalized Form)

price_norm =
  0.1123
+ 0.3551 * area_norm
+ 0.0814 * bedrooms_norm
+ 0.0662 * bathrooms_norm
+ 0.0440 * stories_norm
+ 0.1204 * parking_norm
+ 0.0511 * mainroad
+ 0.0201 * guestroom
+ 0.0332 * basement
+ 0.0910 * furn_furnished
+ 0.0320 * furn_semifurnished
+ 0.0000 * furn_unfurnished
```

```
Enter new instance for prediction:

area (xmin=1650 xmax=16200):
9000

bedrooms (xmin=1 xmax=6):
3
bathrooms (xmin=1 xmax=4):
2

stories (xmin=1 xmax=4):
2

parking (xmin=0 xmax=3):
1

mainroad (yes/no):
yes

guestroom (yes/no):
no

basement (yes/no):
yes

furnishingstatus (furnished, semi-furnished, unfurnished):
semi-furnished

Normalizing new input...

[Thread N1] Normalizing area...
[Thread N2] Normalizing bedrooms...
[Thread N3] Normalizing bathrooms...
[Thread N4] Normalizing stories...
[Thread N5] Normalizing parking...
[Thread N6] Normalizing price (TARGET)... ymin=1250000
ymax=13300000

area_norm        = 0.5220
bedrooms_norm    = 0.4000
bathrooms_norm   = 0.3333
stories_norm     = 0.3333
parking_norm     = 0.3333
mainroad         = 1
guestroom        = 0
basement         = 1
furn_furnished=0  furn_semifurnished=1  furn_unfurnished=0
```

```
Predicted normalized price: 0.63481

Reverse-normalizing target...

price = 0.63481 * (13300000 − 1250000) + 1250000
price ≈ 9,782,545.05

PRICE PREDICTION RESULTS:

Normalized prediction : 0.63481
Real-scale prediction : 9,782,545.05 TL

Do you want to continue? (Y/n):
n

Thank you for using PRICE PREDICTION SERVER! Good Bye!
```

The above example shows how a communication between your server and telnet command should be handled. In addition to this you should also send error messages. There could be more error messages you may need to send, to make your program more user friendly.

If there is an error that is not covered here, you can write your own error message with similar style to given error messages above. After sending an error message, you should close the connection and wait for new telnet connection to be made. You are not required to ask for correct input again, however if you wish, you are allowed to create a program that works like that, rather than closing down.

You are advised but not required to write your code from simplest implementation step by step. For example, if you know Python or another high level language, write this program there first. This will allow you to quickly develop correct working program. After that, translate that program to C language, because you know your design is correct, all emerging errors will be caused by C language rules, data types and functionalities. After that add multi threading and debug emerged errors. Lastly, add port communication to finish your program and satisfy the requirements.

You are also required to come to code control during your lab sessions in 18.12.2025, 25.12.2025 and 30.12.2025. The code control appointment table is included with this assignment paper, please check it and make sure you attend to your code control appointment. If you do not come to code control, even if you made an upload with perfectly working code, you grade will be zero.

We are planing to do your code controls in Lab 3 and 4 (the labs were “CME 1251 – Project Based Learning – I” class is held), however if it there is a change of location, you will be informed in Sakai announcements. If there is an extreme scenario and you will not be able to come to your code control, notify us immediately of this situation. You are also advised to remind your group mates of your code control to prevent them from forgetting and getting a zero.

Your code control will be 20 minutes in total, 5 minutes for presenting your code, 5 minutes for executing and showing different outputs, 5 minutes for questions and evaluation by your teacher and lastly 5 minutes as an extra in case your code control took too long or you experienced technical problems during your code control. If the code control of previous group finish early, you are free to do your code control early if you want.

GRADING REQUIREMENTS:

File and Code Criteria:

Correct naming of upload file.
Correct English variable and function names.
Clear English comments.
Code quality, indentation, and readability.
Using required global variables with correct names.
Correct, reproducible console compilation on Linux.

Preprocessing & Encoding Criteria:

Correct detection of numeric vs categorical columns.
Correct per-column min–max normalization with stored x_min/x_max.
Correct Housing.csv furnishing status encoding rule.
Clear handling of constant numeric columns (xmin == xmax).
Consistent preprocessing between training and prediction.

Concurrency Criteria:

Creating one thread per attribute for normalization/encoding.
Creating at least one thread per coefficient βj for training.
Correct use of shared memory and synchronization (mutex/semaphore/barriers).
No data races, deadlocks, or resource leaks; all threads joined.

Regression & Prediction Criteria:

Correct training on normalized data (Normal Equations or Iterative method).
Printing the regression equation in normalized space.
Correct normalized prediction ŷ_norm.
Correct reverse-normalized prediction y on real scale.
Robust handling of invalid/unseen categorical inputs.

Execution Criteria:

Successful load of all datasets; informative error on missing files.
Opening port communication to port 60000.
Interactive prediction flow working as described.
Clear, user-friendly error messages.
Program exits cleanly.

Code Control/Presentation Criteria:

Individual:

Able to explain design choices (thread roles, data layout, synchronization).
Able to answer questions correctly.

Group:

Program works correctly on instructor’s machine.
Correct behavior with different datasets and edge cases.
Demonstrates concurrency visibly (e.g., logs, thread IDs, timing).

UPLOAD REQUIREMENTS:

You are required to upload the C source file you have written to SAKAI. You must compile and test it via console (gcc and ./) before upload. If we cannot compile and execute your code on our machines, your grade will be significantly reduced. Do not upload binaries.

You may work as a group of two (2) or three (3) students. Please do not use spaces or Turkish characters in your file name.

For 3 Student Groups:
GROUP_XX_(S_NUMBER_1)_(S_NUMBER_2)_(S_NUMBER_3).c
Example: GROUP_01_2024510123_2024510124_2024510125.c

For 2 Student Groups:
GROUP_XX_(S_NUMBER_1)_(S_NUMBER_2).c
Example: GROUP_01_2024510123_2024510124.c

Late or no submissions will be graded zero. In a rare case of system or upload failure, email your code file to the research assistants as a record; acceptance is not guaranteed. Prefer uploading one or two days before the deadline. If you later improve your code before the deadline, delete the original upload and submit a new file with a datetime suffix “YYYYMMDD_HHMM”.

Example (2-person group):
GROUP_01_2024510123_2024510124_20251217_1920.c

Academic Integrity:
Your uploads will be analyzed for cheating and plagiarism. Learning from online resources is allowed, but write your own code. Copy-pasted or AI-generated code without understanding will be considered plagiarism.

Questions:
If you have questions or problems regarding this assignment paper, ask them during lab sessions so everyone benefits from the answers. Please avoid email for routine questions.

GOOD LUCK TO YOU ALL!