# Final Project – SW Test course (67778) – 2021

The project should be submitted in groups of 4-5 students. The amount of work requires a group effort. We assume you will split the work between you so that not everything falls on one person.

The project is based on Tesseract, v5.0.0-alpha.20200328, running on Windows 10. Tesseract must not only be installed but also configured to run from a cmd shell (put the location of tesseract.exe in your PATH; define TESSDATA_PREFIX). See detailed instructions in Lesson 01 pre-class exercise.

The executable part of your submission will be tested on Windows 10.

You can write the answers in Hebrew or English.

## Requirements

> Note: Some of the requirements and limitations listed are artificially set by us for this project, to make it easy to decide what is considered a bug. These are not necessarily "real life" requirements of Tesseract, just the ones to be followed in the exercise.

**Tesseract shall convert text from images, under the following specifications:**

1. The supported fonts are the ones listed in "font list" at the end of this document.
2. Supported languages: English, Hebrew, Swedish, Spanish.
   2.1. Note: get language packs from here.
3. Supported font sizes: All font sizes between 8 and 72 (sizes are in points, the common measurement you get from editing applications).
4. There is no guarantee on OCR results for images using fonts not in the list, font size out of the range, or in a different language.
5. The text can be anywhere in the image, without limitations, as long as the full characters are in the image and are not overlapping other characters. There is no guarantee on results for characters that are partially in the image (truncated characters) or overlapped.
6. The text in the image can have any number of lines and the lines can be of any length (number of characters). Any length of text is supported.
7. The OCR is not supported on "text in the wild" (text from pictures taken from real-life, e.g. street signs). The text for OCR should be text from a text editors, using the supported fonts.
8. The OCR is limited to text angled at 0°, 90°, 180° and 270° degrees rotation (0° means horizontal text with no rotation).
9. The OCR supports only black text on white background (i.e. don't test with colors – bugs involving colors won't be accepted for this exercise).

**OCR tuning**

10. Tesseract's behavior can be modified by **configuration parameters** (CP). Each configuration parameter has a default value. The full list with the default values can be seen by running this command:

    ```
    tesseract --print-parameters
    ```

    The value of CPs can be changed from the default in two ways:

    - By specifying the new value in a config file (a text file)
    - By specifying the new value on the command line using the '-c' option

    If unspecified, Tesseract uses the default.

    Options on command line must occur before the specification of the config file.

    The format for the command is:

    ```
    tesseract imagename outputFileName  [options...] [configfile...]
    ```

    For this project, you should consider only the following configuration parameters:

    tessedit_char_blacklist
    tessedit_char_unblacklist
    tessedit_char_whitelist
    page_separator
    tessedit_load_sublangs

11. The only command line options to cover by testing are:
    - **-l** (specifying the languages)
    - **-c** (specifying configuration parameters).

    You can neglect testing any of the other options. We will not accept bugs that are connected to options other than the two above.

## Project Tasks

1. [10%] Use the “feature breakdown” technique to generate a breakdown of the various ways configuration parameters can be specified and the interactions between these options. The output should be a list of aspects or items to test, as demonstrated in class. It should include not more than 20 lines. There is no need to do prioritization.

   Guidance:
   - More about feature breakdown: See lesson 11's material.
   - This task refers to the mechanism of setting the value of configuration parameters; it should not relate or mention any specific parameter.

2. [15%] Use exploratory testing to investigate how the **page_separator** configuration parameter behaves.

   In your answer:
   - Explain the **page_separator** configuration parameter (what it does; how it behaves; Use no more than 5 sentences).
   - If you find any anomalies, write them as a bug report, following the template given in lesson 11.

   More about exploratory testing: See lesson 1’s material.
   More about bug reports: See lesson 11’s material.

3. [15%] Propose three Metamorphic Relations that exist in Tesseract. For each, write a test case that shows how the relation can be used for generating many tests.
   Write the test case in the format taught in class.

   **Warning:** A single relation can be used to generate many slightly different tests. If the way you generate the new tests is following a similar principle, you are probably using the same relation. Don’t count it as many relations; We won’t… Also, we will only check the first three proposals. There is no point writing more than three.

   More about writing test cases: See lesson’s 11 material.
   More about metamorphic relations: See lesson 5’s material.

4. [60%; see breakdown inside]

   Our approach to test tesseract in this exercise follows this simple form:

   (a) Perform OCR on an image.
   (b) Compare the results to what is known to be in the image.

   However, the number of different tests that can be executed is infinite and testers must decide how much to test and what to test, to achieve a reasonable confidence that the tested program works well, while not testing forever.

For this project, we focus on the following input parameters:

- Font
- Font size
- The configuration parameters mentioned in the requirements (req. 10)
- Variations in the text in the image:
  - Length of text
  - Position of text in the image
  - Language
  - Space between words and lines
  - The content of the text

4.1 [5%] For each of the text variations, suggest a partition to equivalence classes. List only the VALID classes.

More about equivalence partitions and equivalence classes: See lesson 4’s material.

4.2 [20%] Propose a Test Strategy for testing tesseract focusing on the parameters listed above. Write an explanation of your strategy and explain why you think it is sound, efficient, and effective. Use maximum two A4 page, Calibri font, 12pt.

More about test strategy: See lesson 11’s material.

**NOTES:**

a) Read the test-strategy example file and the NOTE for Students at its end!

b) There is no one right strategy. You need to develop a proposal for testing that is logical and has good reasoning behind it. Something you can explain and defend when asked “why did you decide to test in this way”. Also: testing ALL tesseract is way beyond the scope of a final project. We are not asking for this – just the scope we outlined above and in the requirements. Explains why you will NOT test some aspects, if you consciously made such decisions.
Again: **By definition** your strategy will not be perfect. Don’t kill yourself, but don’t write nonsense, it is easy to see through.

c) Engineers, developers and testers need to write clearly. Explain things in a clear language that is easy to follow and read. Break overly long sentences to short sentences. Perform a review of this text with your team members. We will not try to guess what a complicated text tries to say, or what you neglected to say clearly.

4.3 [35%] Implement your strategy:

4.3.1 Create images that fit the strategy. Use the code we provide (Test2Image.py) as a starting point for automated generation of images. You can also use Paint, or any other image-editing application to create images.
4.3.1.1 The images should fit the strategy you defined
4.3.1.2 You are allowed maximum of 300 images

4.3.2 Write test automation (in Python 3) that will test tesseract using these images

4.3.3 Run the tests. Ensure it does not take more than 15 minutes to run all the tests. Running much faster is OK. While performance will differ on different machines, we assume you have access to a 2-3 years old PC or newer, with at least 8GB RAM, running Windows 10.

4.3.4 Write bug reports for any anomaly you find. For the most critical five bugs, write a full report, per the template taught in lesson 11. For the rest of the bugs, create a table with just a title for the bug, the test number (see below) and the tesseract command to run.

More about test automation: See lesson 7's material, and exercise 3.

More about bug reports: See lesson 11's material.

**Guidelines and Rules:**

- The test automation must work like in Exercise 3 (we will use the command below, using Python 3.8.x):

```
python project_<teamName>.py input_<teamName>.csv
```

- Note that the CSV format is similar but not exactly the same as what we used in Exercise 3. You need to add two columns: Test Number, and Image Description.
    - Test number: an integer. Different number for each test. Does not have to be consecutive numbers.
    - Image description: A **very** short description what's in the image. Examples of descriptions: "One line of text"; "Multiple lines in two languages: Eng and Swe".
      You can choose to add new columns to accommodate other options. If you completed Exercise 3, this change should be easily accomplished.

- Mandatory output to the screen:
    - A header
    - A line that clearly identifies the test being run and the result (test number, and the whole command line).

EXAMPLE:

| **Test#** | **Command** | **Result** |
|---|---|---|
| 1 | tesseract Arial_16pt.jpg out | PASS |

Any printout that tesseract does while running can appear in the output. Don't waste time trying to suppress it. This is why we ask for the header to be printed out on each test.

- Each image should test a specific aspect, or some well-known combination of aspects of the strategy. While you can put all the tests into one huge image, it will be VERY difficult to know what the bug is, when the test fails. Hint: If the Image description is very long, you probably test too many things are once.

- The test run time for all the tests together is limited to 15 minutes.

- If your team finds a unique bug that other teams did not, you will get 2 bonus points in the grade, up to 6 bonus points for the project.

We strongly suggest you do NOT look for "ready made" images on Google. It will take you a lot of time to find images that fit your strategy. It will likely take you less time to just create the images.
You can create images with Word, with MS Paint, or other text/image editor of your liking. You can also use the python package we prepared to generate images (**Text2Image.py** - available in moodle).

## Submission

Submit in **Project_<teamName>.zip**: (you get to select your team name)

ZIP file contents:

- **README_<teamName>.txt**: A file with:
  - The names, email and IDs of all the team members
  - The details on the PC on which you ran your tests: CPU number and speed, RAM size (In Windows machines you get this info by right-click on "This PC" icon (or "This PC" in File Explorer window) -> Properties ).
  - Any important information you think we need to know about your submission, that is not part of the Test strategy.
- **Tasks_<teamName>.docx:** A file with answers to tasks 1, 2, 3, 4.
- For task 4, make sure to include:
  - Test strategy
  - Bug reports (if you found any bugs)
- Your automation code in files:
  - **project_<teamName>.py**
  - **input_<teamName>.csv**
- Your test images in a folder called "**images**"
- **testResults_<teamName>.csv**

A file with the test results of each test in task 4. This can be a copy of **input_<teamName>.csv**, with an additional column with the results (PASS/FAIL).

## Supported fonts list:

1. Aharoni Bold
2. Arial
3. Arial Black
4. Bahnschrift
5. Calibri
6. Cambria
7. Cambria Math
8. Candara
9. Comic Sans MS
10. Consolas
11. Constantia
12. Corbel
13. Courier New
14. David
15. David Bold
16. Ebrima
17. Forte
18. Franklin Gothic Medium
19. FrankRuehl
20. Gabriola
21. Gadugi
22. Georgia
23. Gisha
24. Gisha Bold
25. Impact
26. Ink Free
27. Levenim MT
28. Levenim MT Bold
29. Lucida Console
30. Lucida Sans Unicode
31. Marlett
32. Microsoft Sans Serif
33. Miriam
34. Miriam Fixed
35. MS Gothic
36. Narkisim
37. Rod
38. Segoe Print
39. Segoe Script
40. Segoe UI
41. Tahoma
42. Times New Roman