Instructions for Automated Testing

- Add your assembler program file in the Simple-Assembler Directory.

- Add the following line(s) to your **run** file in the **Simple-Assembler** directory of the **CO_A_P1** directory:

    I) Compile and run Java program:

    javac your_assembler.java

    java your_assembler

    II) Compile and run C++ program:

    g++ -o <output_file_name> your_assembler.cpp
    ./<output_file_name>

    III) Run Python program:

    python3 your_assembler.py

- To run the executable (**run**) file on Windows OS, you may seek one of the methods listed below:
    1. Use a Bash emulator: Install a Bash emulator such as Git Bash. Following are the links to the installation and the procedure to follow, respectively.
       https://gitforwindows.org/
       https://stackoverflow.com/questions/36401147/running-sh-scripts-in-git-bash
       (your run files usually have .sh extension in Linux. They can execute even without the extension specified, like in the case of the run file provided to you)

    2. Use Windows Subsystem for Linux (WSL) on VSCode: you can enable the Windows Subsystem for Linux, which allows you to run a Linux distribution such as Ubuntu directly on Windows. Once you have WSL installed, you can execute the run file from within the Linux environment by moving your testing directory to the Linux file system in WSL. It's recommended you do this on VSCode for a better experience. Refer to the following link:
       https://code.visualstudio.com/docs/remote/wsl-tutorial

    3. Use a virtual machine: Install a virtual machine software like VirtualBox (https://www.virtualbox.org/) or VMware (https://www.vmware.com/) and set up a Linux distribution inside the virtual machine. This will provide you with a full Linux

environment where you can execute the run file. Obviously, you'll have to move the entire testing directory to the OS run by your VM.

- Then go to the **automatedTesting** directory in your terminal and execute **./run** command (make sure your run file is made an executable file before running the run file)
- Also, make sure to run it with the **--no-sim** argument.

```
ullas@ullas-aspire5:~$ cd CO_M21_Assignment-main/
ullas@ullas-aspire5:~/CO_M21_Assignment-main$ ls
Assignment.pdf  automatedTesting  README.md  Simple-Assembler  SimpleSimulator
ullas@ullas-aspire5:~/CO_M21_Assignment-main$ cd automatedTesting/
ullas@ullas-aspire5:~/CO_M21_Assignment-main/automatedTesting$ ./run --no-sim
================================================
================ TESTING ASSEMBLER ===============
================================================

Runing simple tests
[FAILED] test01
[FAILED] test02
[FAILED] test03
[FAILED] test04
[FAILED] test05
```

The printed output will look somewhat like this (if your assembler program failed the test cases)

```
ullas@ullas-aspire5:~/CO_M23_AA/automatedTesting$ ./run --no-sim
================================================
================ TESTING ASSEMBLER ===============
================================================

Runing simple tests
[PASSED] test1
[PASSED] test2
[PASSED] test3
[PASSED] test4
[PASSED] test5

Running hard tests
[PASSED] test1
[PASSED] test2
[PASSED] test3
[PASSED] test4
[PASSED] test5
```

The printed output will look somewhat like this (if your assembler program passed the test cases)

- The errorGen files are run automatically, but they don't have a correct solution to compare with. Make sure your program prints any one valid error statement for the erroneous files and that you handle all possible corner cases for errors.

All the best!