TAYLOR SERIES EXPANSION in MIPS Assembly
by **Stefan Goldschmidt** and **Oliver Rudzinski**

- Project Submission & Documentation-

This submission for the graded MIPS assembly project has been written inside the *Microsoft Visual Studio Code* editor and tested with the *QtSpim* assembly emulator. Formatting of the source code and output in different editors and emulators may vary.

Interval definition
The finite Taylor series expansion for the **exponential function** always contains 30 summands. This yields an interval for an accurate estimation of $[-8.1, 8.1]$.

The finite Taylor series expansion for the **natural logarithmic function** on the interval also always contains 30 summands. This provides sufficient accuracy for its convergence interval. With the implemented reformulation of every number outside this interval, the calculation works for arbitrarily large numbers up to the representation limit of single-precision floating point numbers.

Required tasks
All required tasks have been completed.

Optional tasks
Additionally, to the required tasks, the following tasks and improvements have been implemented:

- Check in each function if the argument x can be calculated regarding the limitations of the approximation and the function itself
- Error handling regarding false input (lower bound being larger than upper bound)
- Warning that step sizes below 0.1 can yield bad results because of lacking precision of floats.
- Possibility to repeat the program at the end.
- Improved table-like user interface.

Confusion
Task 3 states that output should contain $ln(y)$, y being $exp(x)$. This would yield x again since those functions are inverse to each other. Even though we believe that this was just a typo in the assignment description, you can uncomment line 147 inside the source code to calculate this value instead of $ln(x)$

Remarks

- The interval to be put in by the user is meant to be inclusive but will be exclusive in terms of its upper bound in reality. This is because adding a certain value in single-precision floating point format often contains round-up errors. Therefore, the upper bound of the interval cannot be reached exactly by the program and will therefore be neglected.
- The optional improvements are not included in the C code.