This code was developed by Anamarija Fofonjka in the lab of Michel Milinkovitch (LANE)
https://www.lanevol.org/
Corresponding author email : Michel.Milinkovitch@unige.ch

Description:
    RD-3D is a GPU-based finite-difference implementation used to simulate the reaction-diffusion process
    on a regular network of spacing ε with periodic or no-flux boundary conditions at the domain borders.

Files:
    code folder:
        3D-RD.cu - contains the main function and multithreaded implementation for running GPU kernel functions and writing the data
        kernel.cu - implementation of finite-difference GPU kernel functions
        loading.h - contains functions used to load the data and set up the simulation context
        Makefile - used to compile and run the code
        3D-RD - compiled binary file
        3D-RD.o - intermediate compiling stage
    networks folder:
        Contains data for test examples below.
        Contains python script to generate gaussian bumps networks of different heights and standard deviations (requiers python3 interpreter + numpy library).
    output folder:
        Contains test examples outputs.

Compile:
    Requirements:
        NVIDIA graphics card, NVIDIA C Compiler (nvcc), Nvidia toolkit, GNU c++ compiler (g++)
        Installation guides:
            https://docs.nvidia.com/cuda/index.html#installation-guides
            https://gcc.gnu.org/install/index.html
        Required software installation takes a couple of minutes.

    To compile the code, enter the directory containing the code and type:

        make

    To clean the last build type:

        make clean

Run:
    To run the test example on a regular 10 by 10 network of hexagonal prisms or gaussian bumps use the following command line arguments:

    Argument 1: network type (1 - hexagonal prisms, 0 - gaussian bumps)
    Argument 2: initial conditions if network type is 1 (1 - uniformly colored scales, 0 - randomized uniform steady state)
    Argument 2: gaussian bumps sigma (integer) if network type is 0
    Argument 3: total height of the simulation domain
    Argument 4: domain thickness between prisms/gaussian bumps
    Argument 5: mesh spacing epsilon

    Examples:

        Top simulation view will be outputed for hexagonal prisms and bottom view for gaussian bumps.
        Image outputs come in .ppm format and are stored every 10000 iterations.
        Inital conditions and the final state will be stored as point clouds in .ply files.
        Hexagonal prisms are simulated with periodic and gaussian bumps with no-flux boundary conditions.
        All examples were ran on NVIDIA Titan V graphics card.

Example 1: Uniformly colored prisms as initial condition
To run this example type:
./3D-RD 1 1 20 6 1
The results will be stored in output/prisms_1_6_20_1 folder and can be compared with provided expected output in output/prisms_1_6_20_1_out.
The run takes approximately 1 minute.

Example 2: Randomised uniform steady state as initial condition
./3D-RD 1 0 20 6 1
The results will be stored in output/prisms_0_6_20_1 folder and can be compared with provided expected output in output/prisms_0_6_20_1_out.
The run takes approximately 1.5 minutes.

Example 3: Randomised uniform steady state as initial condition (always used for gaussian bumps).
./3D-RD 0 7 20 6 1
The results will be stored in output/gauss_6_20_7_1 folder and can be compared with provided expected output in output/gauss_6_20_7_1_out.
The run takes approximately 1 minute.

To generate gaussian bumps network:

Python script gaussian_bumpy.py generates a network of 10 by 10 gaussian bumps. Command line arguments are:

Argument 1: Maximum network height (integer)
Argument 2: Domain thickness between gaussian bumps (integer)
Argument 3: Standard deviation of the bumps (integer)

Example:
python gaussian_bumps.py 20 6 7

Bottom simulation domain is always defined as z = 0.
Top layer of the generated network is gauss10by10_6_20_7_top_layer.ply while gauss10by10_6_20_7.txt is used for simulation.