Exercise 9

Task 1 (`ping_pong.c`)

Write an **MPI program** `ping_pong.c` in which two processes send numbers to each other.

1. Initialize the MPI environment with `MPI_Init()` and finalize it with `MPI_Finalize()`.
2. Store information about the ranks and the total number of processes using `MPI_Comm_rank` and `MPI_Comm_size`.
3. The program must be executed with exactly two processes. If more than two processes are used, abort the program and display an appropriate error message.
4. Define a constant `PING_PONG_LIMIT`. Up to this limit, the two processes should continuously send each other a number, incrementing it each time.
5. Implement the sending and receiving logic for both processes. One process should send all odd numbers, while the other sends even numbers. For each send and receive operation, print a corresponding message to the console. The message should include the sender rank, receiver rank, the action (send/receive), and the number. Example:
   ```
   Rank 0 sent ping_pong_count 1 to Rank 1
   Rank 1 received ping_pong_count 1 from Rank 0
   Rank 1 sent ping_pong_count 2 to Rank 0
   Rank 0 received ping_pong_count 2 from Rank 1
   ```
6. Add timing calls before and after the loop using `double MPI_Wtime(void)`

Hints: Use blocking `MPI_Send`/`MPI_Recv` with a fixed `tag` (e.g., 0). Decide whose "turn" it is from `ping_pong_count % 2` and the rank.

Compile & Run (locally)

```
mpicc -O2 -Wall -Wextra -o ping_pong ping_pong.c
mpirun -n 2 ./ping_pong
```

Run on the Cluster (Slurm)

Execute `ping_pong.c` on the Fulda HPC cluster via a customized Slurm script.

Task 2 (`numbers.c`)

Write an **MPI program** `numbers.c` in which one process sends a list of numbers to another process. The receiving process calculates the sum of the numbers and sends the result back to the original process, which then outputs it. Execute `numbers.c` on the Fulda HPC Cluster.