Homework 6, due: 03/10

MATH 9830, Spring 2015 Timo Heister, heister@clemson.edu

1. Write a program using MPI that sends a token around in a ring pattern from one processor to the next:

   - The token is of type integer starting with the value 0 on rank 0.
   - The token is send from the process with rank i to the process with rank $1+i$ (wrapping around to 0). Before every send, the value of the token is incremented by one.
   - The process is repeated for three rounds and process 0 announces the current round.
   - Each process prints to the screen when a message is received (include the rank and the token value). Finally, process 0 outputs the final value before exiting.
   - Use MPI_Barrier to ensure the output appears in order.

   Example output (try to match this):

```
$ mpirun -n 3 ./main
We are having a party with 3 processes!
ROUND 0
 process 1 got token = 1 from rank 0
 process 2 got token = 2 from rank 1
 process 0 got token = 3 from rank 2
ROUND 1
 process 1 got token = 4 from rank 0
 process 2 got token = 5 from rank 1
 process 0 got token = 6 from rank 2
ROUND 2
 process 1 got token = 7 from rank 0
 process 2 got token = 8 from rank 1
 process 0 got token = 9 from rank 2
 final answer: 9
```

2. Manufactured solutions in step-5:

   (a) Make yourself familiar with the concepts in step-5 (documentation, video lectures, read the code).

   (b) Change the output to vtk and visualize the solutions (submit two images on different mesh levels that elevate the 2d solution and also show the grid – similar to the images in the tutorial).

   (c) The method of manufactured solutions allows us to check our code for correctness by solving a PDE where we know the answer. We construct this problem by making up the correct solution
   $$u_{ref} = \sin(2\pi x)\cos(4\pi y)$$
   and computing a function f such that u_{ref} fulfills the PDE. We will choose $a = 0.1$ for the diffusion coefficient everywhere. Show me your f.

   (d) Now change to the diffusion coefficient in the code, change the starting mesh to a hyper_cube with 3 initial refinements, and implement a class `Solution` (derived from `Function<dim>` like `Coefficient`) with the answer above. Change the boundary conditions to use the `Solution` class (we obviously need the correct boundary conditions!). Check your results visually.

(e) deal.II allows you to compute the error $\|u - u_{ref}\|_0$ between a given reference and the computed solution:

```
Vector<float> difference_per_cell (triangulation.n_active_cells());
VectorTools::integrate_difference (dof_handler,
                                   solution,
                                   Solution<dim>(),
                                   difference_per_cell,
                                   QGauss<dim>(3),
                                   VectorTools::L2_norm);
double L2_error = difference_per_cell.l2_norm();
```

You will need to include `<deal.II/numerics/error_estimator.h>`.

Create a table with the refinement level (1-5) and the L2 error and check that the convergence order is consistent with the finite element theory.