C++11 code 9.1.24: Implementation of `solveHeatEvolution()` ↬ GitLab

```
Eigen::VectorXd solveHeatEvolution(const lf::assemble::DofHandler &dofh,
                                   unsigned int m, double final_time) {
  Eigen::VectorXd discrete_heat_sol(dofh.NumDofs());
  double tau = final_time / m;                          // step size
  const lf::uscalfe::size_type N_dofs(dofh.NumDofs());  // dim. of FE space

  std::cout << "*********************************************************"
            << std::endl;
  std::cout << "\n>>> SolveHeatEvolution: m = " << m << ", N = " << N_dofs
            << std::endl;
  /* Setting up the problem information */
  // Precomputing the required data for the Runge-Kutta method
  // Assemble the Runge-Kutta Radau IIA 2-stages method solver (order 3)
  Radau3MOLTimestepper radau_solver(dofh);
  // Starting with the zero initial condition vector
  Eigen::VectorXd discrete_solution_cur =
      radau_solver.discreteEvolutionOperator(0.0, tau,
                                             Eigen::VectorXd::Zero(N_dofs));

  std::cout << "\n>> Iterating the action of discreteEvolutionOperator"
            << std::endl;
  /* Evolving the parabolic heat system */
  // While less elegant, we use a current and next step solution vector in the
  // iteration to stay away from potential harming aliasing effects of putting
  // an Eigen::Vector on both sides of an assignment statement.
  Eigen::VectorXd discrete_solution_next;
  for (int i = 1; i < m; i++) {
    discrete_solution_next = radau_solver.discreteEvolutionOperator(
        i * tau, tau, discrete_solution_cur);
    discrete_solution_cur = discrete_solution_next;
  }
  discrete_heat_sol = discrete_solution_cur;
  return discrete_heat_sol;
}
```