**C++ code 5.5.4: Function implementing the fixed-point iteration**

```cpp
template <typename FUNCTOR_F, typename RECORDER = std::function<
                                  void(const Eigen::VectorXd &, double)>>
Eigen::VectorXd solveSemilinearBVP(
    std::shared_ptr<const lf::uscalfe::FeSpaceLagrangeO1<double>> fes_p,
    FUNCTOR_F f, double rtol = 1.0E-4, double atol = 1.0E-8,
    unsigned int itmax = 100,
    RECORDER &&rec = [](const Eigen::VectorXd &, double) -> void {}) {
  // Wrap right hand side source function into a Mesh Function
  lf::mesh::utils::MeshFunctionGlobal mf_f(f);
  // Reference to current mesh
  const lf::mesh::Mesh &mesh{*(fes_p->Mesh())};
  // Obtain local->global index mapping for current finite element space
  const lf::assemble::DofHandler &dofh{fes_p->LocGlobMap()};
  // Dimension of finite element space'
  const std::size_t N_dofs(dofh.NumDofs());

  // Assemble right-hand side vector
  Eigen::VectorXd phi(N_dofs);
  phi.setZero();
  // Assemble volume part of right-hand side vector depending on the
     source
  // function f.
  // Initialize object taking care of local computations on all cells.
  lf::uscalfe::ScalarLoadElementVectorProvider<double, decltype(mf_f)>
      elvec_builder(fes_p, mf_f);
  // Invoke assembly on cells (codim == 0)
  lf::assemble::AssembleVectorLocally(0, dofh, elvec_builder, phi);

  // Coefficient vector for approximate solution
  Eigen::VectorXd mu_vec(N_dofs);
  // Initial guesd = 0
  mu_vec.setZero();
  // Main fixed-point iteration loop
  Eigen::VectorXd mu_old(N_dofs);
  double diff_norm;  // Norm of correction
  unsigned int steps = 0;
  do {
    mu_old = mu_vec;
    // Solve linear fixed-point system and return solution in mu_vec
    fixedPointNextIt(fes_p, mu_vec, phi);
    diff_norm = (mu_vec - mu_old).norm();
    rec(mu_vec, diff_norm);
    steps++;
    // Correction-based termination
  } while ((diff_norm >= rtol * mu_vec.norm()) && (diff_norm >= atol) &&
           (steps < itmax));
  return mu_vec;
}
```