```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2014-2017 OpenFOAM Foundation
    Copyright (C) 2018-2020 OpenCFD Ltd.
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "mySolidificationMeltingSource.H"
#include "fvMatrices.H"
#include "basicThermo.H"
#include "gravityMeshObject.H"
#include "zeroGradientFvPatchFields.H"
#include "extrapolatedCalculatedFvPatchFields.H"
#include "addToRunTimeSelectionTable.H"
#include "geometricOneField.H"
#include <cmath>
// * * * * * * * * * * * * * Static Member Functions * * * * * * * * * * * * //

namespace Foam
{
namespace fv
{
    defineTypeNameAndDebug(mySolidificationMeltingSource, 0);
    addToRunTimeSelectionTable(option, mySolidificationMeltingSource, dictionary);
}
}


// * * * * * * * * * * * * * Private Member Functions  * * * * * * * * * * * //

void Foam::fv::mySolidificationMeltingSource::update()
{
    if (curTimeIndex_ == mesh_.time().timeIndex())
    {
        return;
    }

    if (debug)
    {
        Info<< type() << ": " << "alpha.liquid" << " - updating phase indicator" << endl;
    }

    // update old time alpha1 field
    alphaC_.oldTime();
    const auto& CpVoF = mesh_.lookupObject<volScalarField>(CpName_);

    const auto& T = mesh_.lookupObject<volScalarField>(TName_);
    scalar Tsol = Tmelt_-0.25;
    scalar Tliq = Tmelt_+0.75;
    scalar eps = 0.0001;
    forAll(cells_, i)
    {
        label celli = cells_[i];

        scalar Tc = T[celli];
        scalar Cpc = CpVoF[celli];
        scalar alpha1New = alphaC_[celli] + relax_*Cpc*(Tc - Tmelt_)/L_;
        // scalar alpha1New = 0.5 + 0.5*std::erf(4 * ((Tc - (Tliq + Tsol)/2)/(Tliq - Tsol + eps)));
        alphaC_[celli] = max(0, min(alpha1New, 1));
    }

    alpha1_.correctBoundaryConditions();

    curTimeIndex_ = mesh_.time().timeIndex();
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::fv::mySolidificationMeltingSource::mySolidificationMeltingSource
(
    const word& sourceName,
    const word& modelType,
    const dictionary& dict,
    const fvMesh& mesh
)
:
    cellSetOption(sourceName, modelType, dict, mesh),
    Tmelt_(coeffs_.get<scalar>("Tmelt")),
    L_(coeffs_.get<scalar>("L")),
    relax_(coeffs_.getOrDefault<scalar>("relax", 0.9)),
    TName_(coeffs_.getOrDefault<word>("T", "T")),
    CpName_(coeffs_.getOrDefault<word>("Cp", "Cp")),
    UName_(coeffs_.getOrDefault<word>("U", "U")),
    rhoCpPhiName_(coeffs_.getOrDefault<word>("rhoCpPhi", "rhoCpPhi")),
    alphaC_
    (
        IOobject
        (
            "alphaPCM",
            mesh.time().timeName(),
            mesh,
            IOobject::READ_IF_PRESENT,
            IOobject::NO_WRITE
        ),
        mesh,
        dimensionedScalar(dimless, Zero),
        zeroGradientFvPatchScalarField::typeName
    ),
    alpha1_
    (
        IOobject
        (
            "alpha.liquid",
            mesh.time().timeName(),
            mesh,
            IOobject::READ_IF_PRESENT,
            IOobject::AUTO_WRITE
        ),
        mesh,
        dimensionedScalar(dimless, Zero),
        zeroGradientFvPatchScalarField::typeName
    ),
    curTimeIndex_(-1)
{
    fieldNames_.resize(2);
    fieldNames_[0] = UName_;
    fieldNames_[1] = TName_;

    fv::option::resetApplied();
}


// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

void Foam::fv::mySolidificationMeltingSource::addSup
(
    fvMatrix<scalar>& eqn,
    const label fieldi
)
{
    apply(geometricOneField(), eqn);
}


void Foam::fv::mySolidificationMeltingSource::addSup
(
    const volScalarField& rho,
    fvMatrix<scalar>& eqn,
    const label fieldi
)
{
    apply(rho, eqn);
}
```

```
bool Foam::fv::mySolidificationMeltingSource::read(const dictionary& dict)
{
    if (cellSetOption::read(dict))
    {
        coeffs_.readEntry("Tmelt", Tmelt_);
        coeffs_.readEntry("L", L_);

        coeffs_.readIfPresent("relax", relax_);
        coeffs_.readIfPresent("T", TName_);
        coeffs_.readIfPresent("U", UName_);
        coeffs_.readIfPresent("Cp", CpName_);
        coeffs_.readIfPresent("rhoCpPhi", rhoCpPhiName_);

        return true;
    }

    return false;
}


// ************************************************************************* //
```