```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2013 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
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 "myIncompressibleTwoPhaseMixture.H"
#include "addToRunTimeSelectionTable.H"
#include "surfaceFields.H"
#include "fvc.H"


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

//- Calculate and return the laminar viscosity
void Foam::myIncompressibleTwoPhaseMixture::calcNu()
{
    nuModel1_->correct();
    nuModel2_->correct();

    const volScalarField limitedAlpha1
    (
        "limitedAlpha1",
        min(max(alpha1_, scalar(0)), scalar(1))
    );

    // Average kinematic viscosity calculated from dynamic viscosity
    nu_ = mu()/(limitedAlpha1*rho1_ + (scalar(1) - limitedAlpha1)*rho2_);
}


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

Foam::myIncompressibleTwoPhaseMixture::myIncompressibleTwoPhaseMixture
(
    const volVectorField& U,
    const surfaceScalarField& phi
)
:
    IOdictionary
    (
        IOobject
        (
            "transportProperties",
            U.time().constant(),
            U.db(),
            IOobject::MUST_READ_IF_MODIFIED,
            IOobject::NO_WRITE
        )
```

```
    ),
    twoPhaseMixture(U.mesh(), *this),

    nuModel1_
    (
        viscosityModel::New
        (
            "nu1",
            subDict(phase1Name_),
            U,
            phi
        )
    ),
    nuModel2_
    (
        viscosityModel::New
        (
            "nu2",
            subDict(phase2Name_),
            U,
            phi
        )
    ),

    rho1_("rho", dimDensity, nuModel1_->viscosityProperties().lookup("rho")),
    rho2_("rho", dimDensity, nuModel2_->viscosityProperties().lookup("rho")),

    // ADDITION
    cp1_("cp", dimensionSet(0, 2, -2, -1, 0, 0, 0), nuModel1_->viscosityProperties().lookup("cp")),
    cp2_("cp", dimensionSet(0, 2, -2, -1, 0, 0, 0), nuModel2_->viscosityProperties().lookup("cp")),
    Pr1_("Pr", dimensionSet(0, 0, 0, 0, 0, 0, 0), nuModel1_->viscosityProperties().lookup("Pr")),
    Pr2_("Pr", dimensionSet(0, 0, 0, 0, 0, 0, 0), nuModel2_->viscosityProperties().lookup("Pr")),
    // END OF ADDITION

    U_(U),
    phi_(phi),

    nu_
    (
        IOobject
        (
            "nu",
            U_.time().timeName(),
            U_.db()
        ),
        U_.mesh(),
        dimensionedScalar("nu", dimensionSet(0, 2, -1, 0, 0), 0),
        calculatedFvPatchScalarField::typeName
    )
{
    calcNu();
}


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

Foam::tmp<Foam::volScalarField>
Foam::myIncompressibleTwoPhaseMixture::mu() const
{
    const volScalarField limitedAlpha1
    (
        min(max(alpha1_, scalar(0)), scalar(1))
    );

    return tmp<volScalarField>
    (
        new volScalarField
        (
```

コンストラクタに比熱とプラントル数の初期化を追加

```
            "mu",
            limitedAlpha1*rho1_*nuModel1_->nu()
          + (scalar(1) - limitedAlpha1)*rho2_*nuModel2_->nu()
        )
    );
}


Foam::tmp<Foam::surfaceScalarField>
Foam::myIncompressibleTwoPhaseMixture::muf() const
{
    const surfaceScalarField alpha1f
    (
        min(max(fvc::interpolate(alpha1_), scalar(0)), scalar(1))
    );

    return tmp<surfaceScalarField>
    (
        new surfaceScalarField
        (
            "muf",
            alpha1f*rho1_*fvc::interpolate(nuModel1_->nu())
          + (scalar(1) - alpha1f)*rho2_*fvc::interpolate(nuModel2_->nu())
        )
    );
}

//ADDITION
Foam::tmp<Foam::surfaceScalarField>
Foam::myIncompressibleTwoPhaseMixture::kappaf() const
{
    const surfaceScalarField alpha1f
    (
        min(max(fvc::interpolate(alpha1_), scalar(0)), scalar(1))
    );

    return tmp<surfaceScalarField>
    (
        new surfaceScalarField
        (
            "kappaf",
            alpha1f*rho1_*cp1_/Pr1_*fvc::interpolate(nuModel1_->nu())
          + (scalar(1) - alpha1f)*rho2_*cp2_/Pr2_*fvc::interpolate(nuModel2_->nu())
        )
    );
}
// END of ADDITION


Foam::tmp<Foam::surfaceScalarField>
Foam::myIncompressibleTwoPhaseMixture::nuf() const
{
    const surfaceScalarField alpha1f
    (
        min(max(fvc::interpolate(alpha1_), scalar(0)), scalar(1))
    );

    return tmp<surfaceScalarField>
    (
        new surfaceScalarField
        (
            "nuf",
            (
                alpha1f*rho1_*fvc::interpolate(nuModel1_->nu())
              + (scalar(1) - alpha1f)*rho2_*fvc::interpolate(nuModel2_->nu())
            )/(alpha1f*rho1_ + (scalar(1) - alpha1f)*rho2_)
        )
    );
```

kappaf() 関数の定義を追加

```
}


bool Foam::myIncompressibleTwoPhaseMixture::read()
{
    if (regIOobject::read())
    {
        if
        (
            nuModel1_().read
            (
                subDict(phase1Name_ == "1" ? "phase1": phase1Name_)
            )
         && nuModel2_().read
            (
                subDict(phase2Name_ == "2" ? "phase2": phase2Name_)
            )
        )
        {
            nuModel1_->viscosityProperties().lookup("rho") >> rho1_;
            nuModel2_->viscosityProperties().lookup("rho") >> rho2_;

// ADDITION
            nuModel1_->viscosityProperties().lookup("cp") >> cp1_;
            nuModel2_->viscosityProperties().lookup("cp") >> cp2_;
            nuModel1_->viscosityProperties().lookup("Pr") >> Pr1_;
            nuModel2_->viscosityProperties().lookup("Pr") >> Pr2_;
// END of ADDITION

            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}


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

read()関数に，比熱とプラントル数の読み込みを追加