```
    Info<< "Reading field p_rgh\n" << endl;
    volScalarField p_rgh
    (
        IOobject
        (
            "p_rgh",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );

// ADDITION
    Info<< "Reading field T\n" << endl;
    volScalarField T
    (
        IOobject
        (
            "T",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
// END of ADDITION

    Info<< "Reading field U\n" << endl;
    volVectorField U
    (
        IOobject
        (
            "U",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );

    #include "createPhi.H"


    Info<< "Reading transportProperties\n" << endl;
    myIncompressibleTwoPhaseMixture twoPhaseProperties(U, phi);

    volScalarField& alpha1(twoPhaseProperties.alpha1());
    volScalarField& alpha2(twoPhaseProperties.alpha2());

    const dimensionedScalar& rho1 = twoPhaseProperties.rho1();
    const dimensionedScalar& rho2 = twoPhaseProperties.rho2();

// ADDITION
    const dimensionedScalar& cp1 = twoPhaseProperties.cp1();
    const dimensionedScalar& cp2 = twoPhaseProperties.cp2();
// END of ADDITION

    // Need to store rho for ddt(rho, U)
    volScalarField rho
    (
        IOobject
        (
            "rho",
            runTime.timeName(),
```

温度を宣言

改造したクラス（ライブラリ）を指定

比熱を宣言

```
            mesh,
            IOobject::READ_IF_PRESENT
        ),
        alpha1*rho1 + alpha2*rho2,
        alpha1.boundaryField().types()
    );
    rho.oldTime();

// ADDITION
    // Need to store rhoCp for ddt(rhoCp, T)
    volScalarField rhoCp
    (
        IOobject
        (
            "rhoCp",
            runTime.timeName(),
            mesh,
            IOobject::READ_IF_PRESENT
        ),
        alpha1*rho1*cp1 + alpha2*rho2*cp2,
        alpha1.boundaryField().types()
    );
    rhoCp.oldTime();
// END of ADDITION

    // Mass flux
    surfaceScalarField rhoPhi
    (
        IOobject
        (
            "rhoPhi",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        fvc::interpolate(rho)*phi
    );

// ADDITION
    surfaceScalarField rhoCpPhi
    (
        IOobject
        (
            "rhoCpPhi",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        fvc::interpolate(rhoCp)*phi
    );
// END of ADDITION

    // Construct interface from alpha1 distribution
    interfaceProperties interface(alpha1, U, twoPhaseProperties);


    // Construct incompressible turbulence model
    autoPtr<incompressible::turbulenceModel> turbulence
    (
        incompressible::turbulenceModel::New(U, phi, twoPhaseProperties)
    );

    #include "readGravitationalAcceleration.H"

    /*
    dimensionedVector g0(g);
```

rhoCp を宣言

rhoCpPhi を宣言

```
// Read the data file and initialise the interpolation table
interpolationTable<vector> timeSeriesAcceleration
(
    runTime.path()/runTime.caseConstant()/"acceleration.dat"
);
*/

Info<< "Calculating field g.h\n" << endl;
volScalarField gh("gh", g & mesh.C());
surfaceScalarField ghf("ghf", g & mesh.Cf());

volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    p_rgh + rho*gh
);

label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell
(
    p,
    p_rgh,
    mesh.solutionDict().subDict("PIMPLE"),
    pRefCell,
    pRefValue
);

if (p_rgh.needReference())
{
    p += dimensionedScalar
    (
        "p",
        p.dimensions(),
        pRefValue - getRefCellValue(p, pRefCell)
    );
    p_rgh = p - rho*gh;
}

fv::IOoptionList fvOptions(mesh);


tmp<surfaceScalarField> tphiAlphaCorr0;
```