beginnerFOAM

이 상 봉

Taylor theorem

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k + h_k(x)(x-a)^k$$

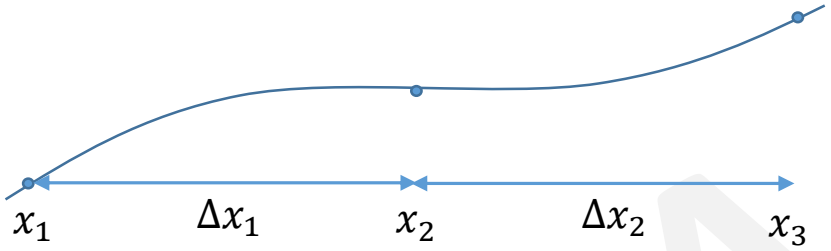
If k≥1 and f is k-times differentiable at the point a

$$\lim_{x \to a} h_k(x) = 0$$

Tayor series

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k + \cdots$$

Taylor Series



expression of f

No. points	Order of accuracy
1	1
2	2
3	3

$$f_1 = f_2 - \Delta x_1 \left.\frac{\partial f}{\partial x}\right|_2 + \left.\frac{\Delta x_1^2}{2!}\frac{\partial^2 f}{\partial x^2}\right|_2 - \left.\frac{\Delta x_1^3}{3!}\frac{\partial^3 f}{\partial x^3}\right|_2 + \cdots$$

$$f_2 \approx f_1 \qquad err = \Delta x_1 \left(-\left.\frac{\partial f}{\partial x}\right|_2 + \left.\frac{\Delta x_1}{2!}\frac{\partial^2 f}{\partial x^2}\right|_2 + \cdots \right) \propto \Delta x$$

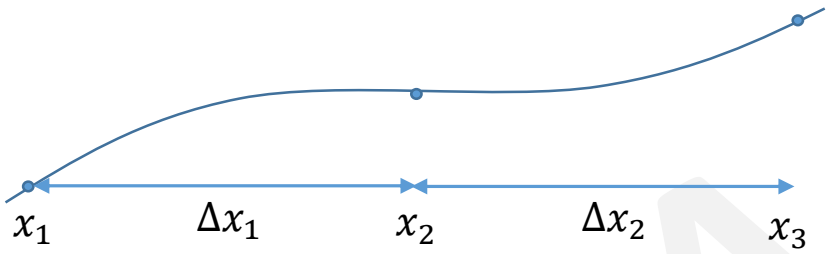
1st order accuracy

$$f_3 = f_2 + \Delta x_2 \left.\frac{\partial f}{\partial x}\right|_2 + \left.\frac{\Delta x_2^2}{2!}\frac{\partial^2 f}{\partial x^2}\right|_2 + \left.\frac{\Delta x_2^3}{3!}\frac{\partial^3 f}{\partial x^3}\right|_2 + \cdots$$

$$f_2 \approx \frac{\Delta x_2 f_1 + \Delta x_1 f_3}{\Delta x_1 + \Delta x_2} \qquad err = \frac{\Delta x_1 \Delta x_2}{\Delta x_1 + \Delta x_2} \left(\frac{\Delta x_1 + \Delta x_2}{2} \left.\frac{\partial^2 f}{\partial x^2}\right|_2 + \cdots \right) \propto \Delta x^2$$

2nd order accuracy

Taylor Series



expression of f'

No. points	Order of accuracy
2	1 (2)
3	2
4	3

$$f_1 = f_2 - \Delta x_1 \left.\frac{\partial f}{\partial x}\right|_2 + \left.\frac{\Delta x_1^2}{2!}\frac{\partial^2 f}{\partial x^2}\right|_2 - \left.\frac{\Delta x_1^3}{3!}\frac{\partial^3 f}{\partial x^3}\right|_2 + \cdots$$

$$\left.\frac{\partial f}{\partial x}\right|_2 \approx \frac{f_2 - f_1}{\Delta x_1} \qquad err = \Delta x_1 \left(\left.\frac{1}{2!}\frac{\partial^2 f}{\partial x^2}\right|_2 + \cdots \right) \propto \Delta x$$

1st order accuracy

$$f_3 = f_2 + \Delta x_2 \left.\frac{\partial f}{\partial x}\right|_2 + \left.\frac{\Delta x_2^2}{2!}\frac{\partial^2 f}{\partial x^2}\right|_2 + \left.\frac{\Delta x_2^3}{3!}\frac{\partial^3 f}{\partial x^3}\right|_2 + \cdots$$

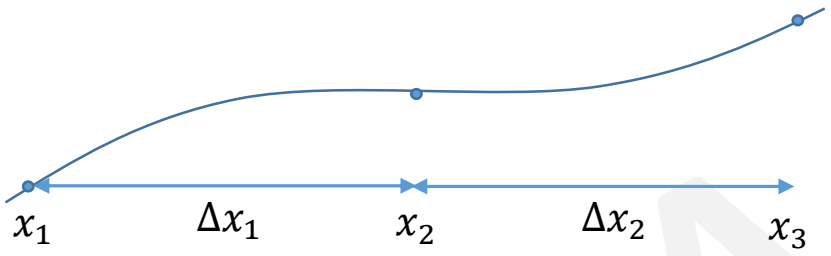
$$\left.\frac{\partial f}{\partial x}\right|_2 \approx \frac{\Delta x_1^2 f_3 - \Delta x_2^2 f_1 + (\Delta x_2^2 - \Delta x_1^2) f_2}{\Delta x_1 \Delta x_2 (\Delta x_1 + \Delta x_2)}$$

2nd order accuracy

$$if\, \Delta x_1 = \Delta x_2, \qquad \left.\frac{\partial f}{\partial x}\right|_2 \approx \frac{f_3 - f_1}{2\Delta x}$$

2nd order accuracy by using 2 points

Taylor Series



expression of f''

No. points	Order of accuracy
3	1 (2)
4	2
5	3

$$f_1 = f_2 - \Delta x_1 \left.\frac{\partial f}{\partial x}\right|_2 + \left.\frac{\Delta x_1^2}{2!}\frac{\partial^2 f}{\partial x^2}\right|_2 - \left.\frac{\Delta x_1^3}{3!}\frac{\partial^3 f}{\partial x^3}\right|_2 + \cdots$$

$$f_3 = f_2 + \Delta x_2 \left.\frac{\partial f}{\partial x}\right|_2 + \left.\frac{\Delta x_2^2}{2!}\frac{\partial^2 f}{\partial x^2}\right|_2 + \left.\frac{\Delta x_2^3}{3!}\frac{\partial^3 f}{\partial x^3}\right|_2 + \cdots$$

$$\left.\frac{\partial^2 f}{\partial x^2}\right|_2 \approx \frac{2\{\Delta x_1 f_3 - 2(\Delta x_1 + \Delta x_2) f_2 + \Delta x_2 f_1\}}{\Delta x_1 \Delta x_2 (\Delta x_1 + \Delta x_2)}$$

$$err = \left.\frac{(\Delta x_2^2 - \Delta x_1^2)}{(\Delta x_1 + \Delta x_2)}\frac{\partial^3 f}{\partial x^3}\right|_2 + \cdots \propto \Delta x$$

1st order accuracy

$$if\ \Delta x_1 = \Delta x_2, \quad \left.\frac{\partial f}{\partial x}\right|_2 \approx \frac{f_3 - 2f_2 + f_1}{\Delta x^2}$$

2nd order accuracy by using 3 points

Taylor Series

Relationship between number of points and accuracy of expression

No. points	f accuracy	f’ accuracy	f’’ accuracy
1	1		
2	2	1 (2)	
3	3	2	1 (2)
4	4	3	2
5	5	4	3
6	6	5	4

Finite Difference Method

$$\frac{\partial \phi}{\partial t} - \alpha \frac{\partial^2 \phi}{\partial x^2} = 0$$

$$\left.\frac{\partial \phi}{\partial t}\right|_i^{(n)} - \alpha \left.\frac{\partial^2 \phi}{\partial x^2}\right|_i^{(n)} = 0$$

$$\frac{\phi_i^{(n+1)} - \phi_i^{(n)}}{\Delta t} - \alpha \frac{\phi_{i+1}^{(n)} - 2\phi_i^{(n)} + \phi_{i-1}^{(n)}}{\Delta x^2} = 0$$

explicit
time accuracy : 1st order
spatial accuracy : 2nd order
simple calculation (no matrix)
limitation of time step

$$\Delta t < \frac{\alpha}{2} \Delta x^2$$

$$\left.\frac{\partial \phi}{\partial t}\right|_i^{(n+1)} - \alpha \left.\frac{\partial^2 \phi}{\partial x^2}\right|_i^{(n+1)} = 0$$

$$\frac{\phi_i^{(n+1)} - \phi_i^{(n)}}{\Delta t} - \alpha \frac{\phi_{i+1}^{(n+1)} - 2\phi_i^{(n+1)} + \phi_{i-1}^{(n+1)}}{\Delta x^2} = 0$$

implicit
time accuracy : 1st order
spatial accuracy : 2nd order
matrix solver
no limitation of time step

Finite Difference Method

$$\frac{\partial \phi}{\partial t} - \alpha \frac{\partial^2 \phi}{\partial x^2} = 0$$

$$\left.\frac{\partial \phi}{\partial t}\right|_i^{(n+1)} - \alpha \left.\frac{\partial^2 \phi}{\partial x^2}\right|_i^{(n+1)} = 0$$

$$\frac{3\phi_i^{(n+1)} - 4\phi_i^{(n)} + \phi_i^{(n-1)}}{2\Delta t} - \alpha \frac{\phi_{i+1}^{(n+1)} - 2\phi_i^{(n+1)} + \phi_{i-1}^{(n+1)}}{\Delta x^2} = 0$$

backward
time accuracy : 2nd order
spatial accuracy : 2nd order
matrix solver
old-old phi (n-1 step)

$$\left.\frac{\partial \phi}{\partial t}\right|_i^{(n+1/2)} - \alpha \left.\frac{\partial^2 \phi}{\partial x^2}\right|_i^{(n+1/2)} = 0$$

$$\frac{\phi_i^{(n+1)} - \phi_i^{(n)}}{\Delta t} - \frac{\alpha}{2}\left(\left.\frac{\partial^2 \phi}{\partial x^2}\right|_i^{(n+1)} + \left.\frac{\partial^2 \phi}{\partial x^2}\right|_i^{(n)}\right) = 0$$

Crank-Nicolson
time accuracy : 2nd order
spatial accuracy : 2nd order
matrix solver
dependency of discretization (time & space)

Finite Difference Method

$$\frac{\partial \phi}{\partial t} - \alpha \frac{\partial^2 \phi}{\partial x^2} = 0$$

$$\left.\frac{\partial \phi}{\partial t}\right|_i^{(n+1)} - \alpha \left.\frac{\partial^2 \phi}{\partial x^2}\right|_i^{(n+1)} = 0$$

$$\frac{\phi_i^{(n+1)} - \phi_i^{(n)}}{\Delta t} - \alpha \frac{\phi_{i+1}^{(n+1)} - 2\phi_i^{(n+1)} + \phi_{i-1}^{(n+1)}}{\Delta x^2} = 0$$

implicit
time accuracy : 1st order
spatial accuracy : 2nd order
matrix solver
no limitation of time step

$$\begin{bmatrix} \frac{1}{\Delta t} + \frac{2\alpha}{\Delta x^2} & -\frac{\alpha}{\Delta x^2} & 0 & 0 & 0 \\ -\frac{\alpha}{\Delta x^2} & \frac{1}{\Delta t} + \frac{2\alpha}{\Delta x^2} & -\frac{\alpha}{\Delta x^2} & 0 & 0 \\ 0 & -\frac{\alpha}{\Delta x^2} & \frac{1}{\Delta t} + \frac{2\alpha}{\Delta x^2} & -\frac{\alpha}{\Delta x^2} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha}{\Delta x^2} & \frac{1}{\Delta t} + \frac{2\alpha}{\Delta x^2} \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

Finite Volume Method

$$\frac{\partial \phi}{\partial t} - \alpha \frac{\partial^2 \phi}{\partial x^2} = 0$$

$$\int_V \frac{\partial \phi}{\partial t} dV - \int_V \alpha \frac{\partial^2 \phi}{\partial x^2} dV = \int_V \frac{\partial \phi}{\partial t} dV - \int_S \alpha \frac{\partial \phi}{\partial x} \cdot dS = 0$$

$$\frac{\phi_i^{(n+1)} - \phi_i^{(n)}}{\Delta t} V_i - \alpha \left\{ \left(\frac{\phi_{i+1}^{(n+1)} - \phi_i^{(n+1)}}{\Delta x_{i+1}} \right) \cdot A_{f_{i+1}} - \left(\frac{\phi_i^{(n+1)} - \phi_{i-1}^{(n+1)}}{\Delta x_i} \right) \cdot A_{f_i} \right\} = 0$$

$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha \left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2} \right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha \left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3} \right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha \left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4} \right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha \left(\frac{A_{f_n}}{\Delta x_n} + \frac{A_{f_{n+1}}}{\Delta x_{n+1}} \right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

beginnerFOAM

Transformation of Procedure-Oriented Code to Object-Oriented Code

Procedure-Oriented

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source);
      calc_coef((double*)coef,source,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

beginnerFOAM

```
int main()
{


   fvMesh mesh;
   GeometricField<double> phi
   (
      IOobject("phi", ".", true, true)
      ,mesh
   );
   for(int nTime=0;nTime<NTST;nTime++)
   {
      fvMatrix phiEqn
      (
         fvm::ddt(phi)-fvm::laplacian(phi)
      );
      phiEqn.solve();
      phi.correctBoundaryConditions();
   }
   return 0;
}
```

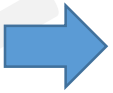
```
$ cat phi
internalField uniform 0.0
boundaryField
fixedValue -100
zeroGradient
```

beginnerFOAM

Transformation of Procedure-Oriented Code to Object-Oriented Code

OpenFOAM

```
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    GeometricField<scalar,fvPatchField,volMesh> phi
    (
        IOobject(“phi", runTime.timeName(), mesh, IOobject::MUST_READ, IOobject::AUTO_WRITE),
        mesh
    );
    while (runTime.loop())
    {
        fvMatrix<scalar> phiEqn
        (
            fvm::ddt(phi) - fvm::laplacian(coef, phi)
        );
        phiEqn.solve();
        phi.correctBoundaryConditions();
    }
    return 0;
}
```



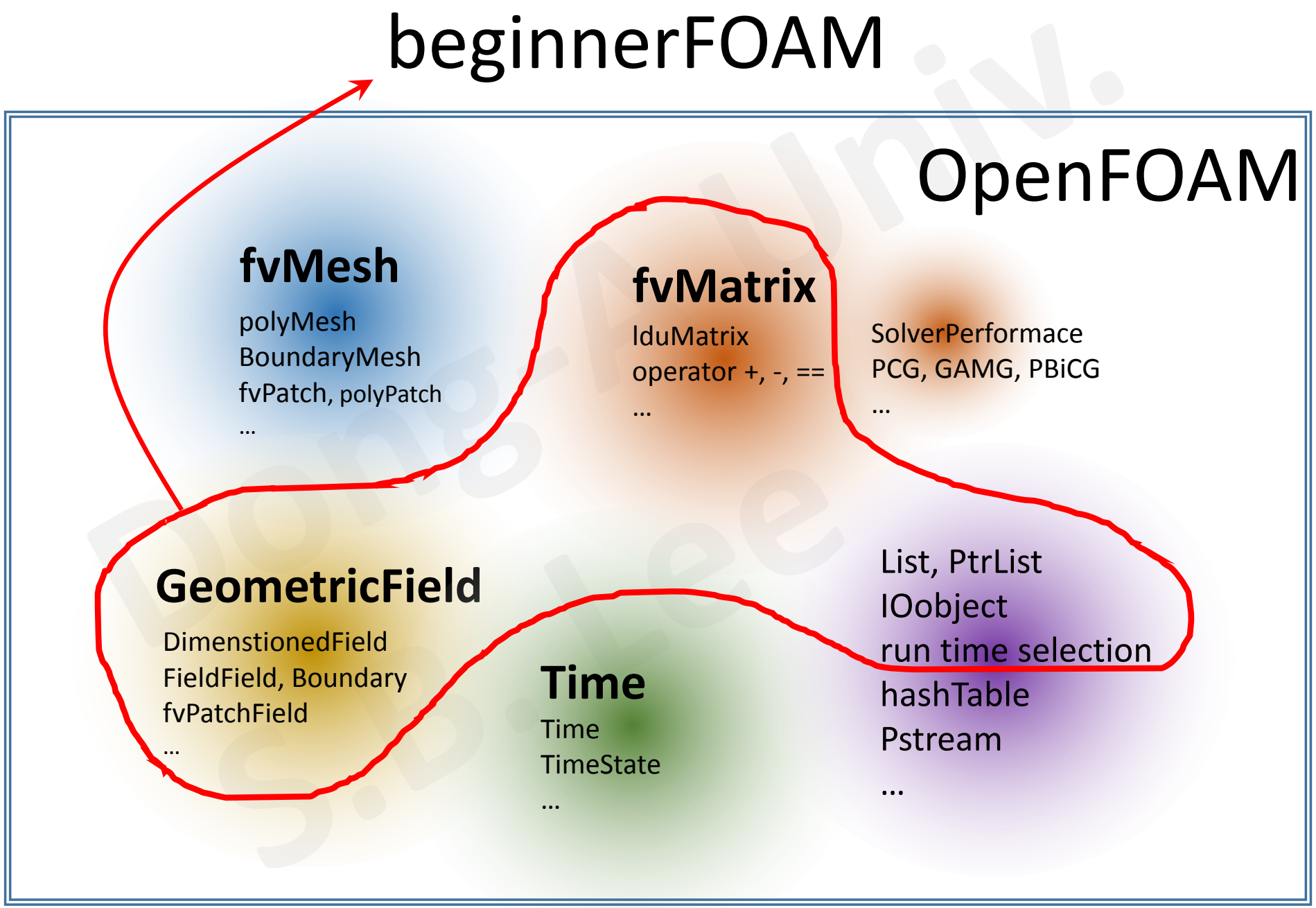
beginnerFOAM

```
int main()
{


    fvMesh mesh;
    GeometricField<double> phi
    (
        IOobject("phi", ".", true, true)
        ,mesh
    );
    for(int nTime=0;nTime<NTST;nTime++)
    {
        fvMatrix phiEqn
        (
            fvm::ddt(phi)-fvm::laplacian(phi)
        );
        phiEqn.solve();
        phi.correctBoundaryConditions();
    }
    return 0;
}
```

beginnerFOAM

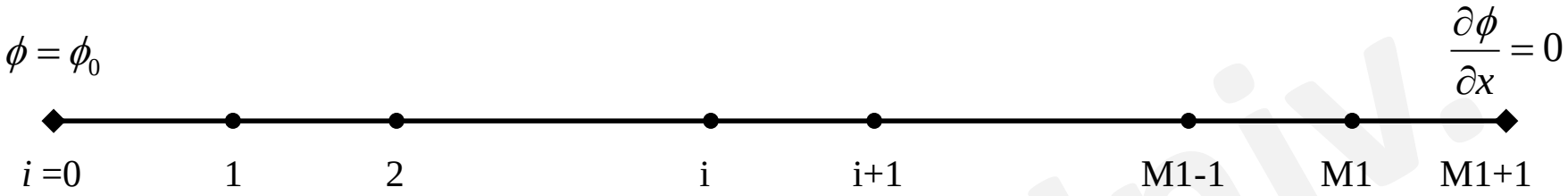
Transformation of Procedure-Oriented Code to Object-Oriented Code

Step 1. Procedure-oriented coding
Step 2. Object-oriented variables (List class)
Step 3. Object-oriented matrix (fvMatrix class at a low level)
Step 4. Object-oriented variables (GeometricField class)
Step 5. Object-oriented variables with I/O object (IOobject class)
Step 6. Object-oriented matrix (fvm namespace)
Step 7. Object-oriented matrix (mathematical operation overloading)
Step 8. Object-oriented matrix based on BC (separation of matrices)
Step 9. Object-oriented variables with boundary (Boundary class inside GeometricField class)
Step 10. Improvement of Boundary class (PtrList class instead of List class)
Step 11. Object-oriented boundary (fixedValue, zeroGradient fvPatchField class)
Step 12. Improvement of each B/C class
Step 13. Improvement of B/C class (fvPatchField class)
Step 14. Impovement of fvPatchField class (virtual evaluate & updateCoeffs)
Step 15. Mechanism of Run Time Selection


beginnerFOAM
OpenFOAM
fvMesh
polyMesh
BoundaryMesh
fvPatch, polyPatch
…
fvMatrix
lduMatrix
operator +, -, ==
…
SolverPerformace
PCG, GAMG, PBiCG
…
GeometricField
DimenstionedField
FieldField, Boundary
fvPatchField
…
Time
Time
TimeState
…
List, PtrList
IOobject
run time selection
hashTable
Pstream
…

Step 1. Procedure-oriented coding

- dual array of double type : coefficient matrix
- single array of double type : variable & source matrix
- procedure-oriented functions in main function



$$\begin{bmatrix} \frac{1}{\Delta t}+\frac{2\alpha}{\Delta x^2} & -\frac{\alpha}{\Delta x^2} & 0 & 0 & 0 \\ -\frac{\alpha}{\Delta x^2} & \frac{1}{\Delta t}+\frac{2\alpha}{\Delta x^2} & -\frac{\alpha}{\Delta x^2} & 0 & 0 \\ 0 & -\frac{\alpha}{\Delta x^2} & \frac{1}{\Delta t}+\frac{2\alpha}{\Delta x^2} & -\frac{\alpha}{\Delta x^2} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha}{\Delta x^2} & \frac{1}{\Delta t}+\frac{\alpha}{\Delta x^2} \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FDM

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source);
      calc_coef((double*)coef,source,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

$\phi_i = \text{phi}[i]$ boundary field : i = 0, M1+1

internal field : i = 1 ~ M1

$S_i = \text{source}[i]$ internal field : i = 1 ~ M1

$A_{ij} = \text{coef}[i][j]$ internal field : i = 1 ~ M1

j = 1 ~ M1

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
    init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source);
      calc_coef((double*)coef,source,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

$$\phi_i^0 = 0$$

$$\text{phi}[i] = 0 \quad \text{for i} = 1 \sim \text{M1}$$

```
void init_phi(double* phi)
{
   for(int i=1;i<=M1;i++) phi[i]=0.0;
}
```

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
       bcond(phi);
      calc_source(phi,source);
      calc_coef((double*)coef,source,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

$$\begin{cases} \phi_0^{n+1} = -100 \\ \phi_{M1+1}^{n+1} = 200 \quad \text{or} \quad \left.\dfrac{\partial \phi}{\partial x}\right|_{M1+!}^{\text{n+1}} = 0 \end{cases}$$

$$\begin{cases} \text{phi}[0] = -100 \\ \text{phi}[M1+1] = 200 \quad \text{or} \quad \text{phi}[M1+1] = \text{phi}[M1] \end{cases}$$

```
void bcond(double* phi)
{
   phi[0]=-100.0;
//    phi[M1+1]=200.0;  // for Dirichlet at i=M1
   phi[M1+1]=phi[M1];  // for Neumann at i=M1
}
```

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
       calc_source(phi,source);
      calc_coef((double*)coef,source,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

$$S_i^{(n+1)} = \frac{\phi_i^{(n)}}{\Delta t}$$

$$S_1^{(n+1)} = \frac{\phi_1^{(n)}}{\Delta t} + \alpha \frac{\phi_0^{(n+1)}}{\Delta x^2}$$

```
void calc_source(double* phi, double* source)
{
   for(int i=1;i<=M1;i++) source[i]=1.0/DT*phi[i];
   source[1]+=ALPHA*phi[0]/DX2;
   // source[M1]+=0.5*ALPHA*phi[M1+1]/DX2;
   // for Dirichlet at i=M1
}
```

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source);
       calc_coef((double*)coef,source,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

$$A_{ii} = \text{coef}[i][i] = \frac{1}{\Delta t} + \frac{2\alpha}{\Delta x^2}$$

$$A_{i-1,i} = \text{coef}[i-1][i] = \frac{\alpha}{\Delta x^2}$$

$$A_{i+1,i} = \text{coef}[i+1][i] = \frac{\alpha}{\Delta x^2}$$

$$A_{M1M1} = \text{coef}[M1][M1] = \frac{1}{\Delta t} + \frac{\alpha}{\Delta x^2}$$

```
void calc_coef(double* coef, double* source, const int& nTime)
{
   double coefRef[M1+1][M1+1];
   for(int i=1;i<=M1;i++) for(int j=1;j<=M1;j++) coefRef[i][j]=0.0;
   for(int i=1;i<=M1;i++)
   {
      coefRef[i][i]=1.0/DT+ALPHA*2.0/DX2;
      if(i!=1) coefRef[i-1][i]=-ALPHA/DX2;
      if(i!=M1) coefRef[i+1][i]=-ALPHA/DX2;
   }
   // for Neumann at i=M1
   coefRef[M1][M1]=1.0/DT+ALPHA*1.0/DX2;
   for(int i=1;i<=M1;i++) for(int j=1;j<=M1;j++)
      coef[i*M1+j]=coefRef[i][j];
}
```

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source);
      calc_coef((double*)coef,source,nTime);
       solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

Gauss - Seidel solver

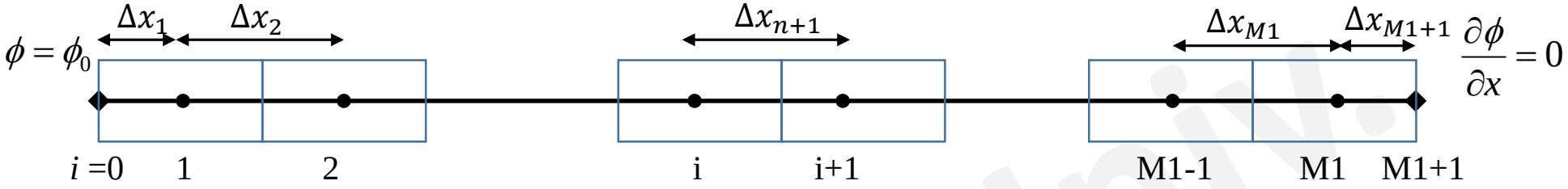
```
void solve(double* coef, double* phi, double* source, const int&
nTime)
{
   int nIter=1;
   double coefRef[M1+1][M1+1], oldPhi[M1+2], residual;

   for(int i=1;i<=M1;i++) for(int j=1;j<=M1;j++) coefRef[i][j]=coef[i*M1+j];
   do
   {
      for(int i=1;i<=M1;i++) oldPhi[i]=phi[i];
      for(int i=1;i<=M1;i++)
      {
         double sumLHS=0.0;
         for(int iLHS=1;iLHS<=M1;iLHS++)
            sumLHS+=coefRef[iLHS][i]*oldPhi[iLHS];
         sumLHS-=coefRef[i][i]*oldPhi[i];
         phi[i]=(source[i]-sumLHS)/coefRef[i][i];
      }
      residual=0.0;
      for(int i=1;i<=M1;i++) residual+=fabs(phi[i]-oldPhi[i])/double(M1);
      cout << nTime << " : RES = " << residual << ", ITER = "
           << nIter << endl;
      nIter++;
   } while(residual>0.0001);
}
```

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source);
      calc_coef((double*)coef,source,nTime);
      solve((double*)coef,phi,source,nTime);
   }
    write_phi(phi);
   return 0;
}
```

writing function

```
void write_phi(double* phi)
{
   ofstream os("1D.dat");
   for(int i=1;i<=M1;i++) os << phi[i] << endl;
   os.close();
}
```

- If we does not want to use an array type for variables?
  : creating a new structure for variables, which has the same function with the array



$$\begin{bmatrix} \frac{V_1}{\Delta t}+\alpha\left(\frac{A_{f_1}}{\Delta x_1}+\frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t}+\alpha\left(\frac{A_{f_2}}{\Delta x_2}+\frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t}+\alpha\left(\frac{A_{f_3}}{\Delta x_3}+\frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t}+\alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   double dx[M1+2],V[M1+1], Af[M1+2];
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source,dx,V,Af);
      calc_coef((double*)coef,source,dx,V,Af,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

$\phi_i = \text{phi}[i]$ boundary field : i = 0, M1+1

internal field : i = 1 ~ M1

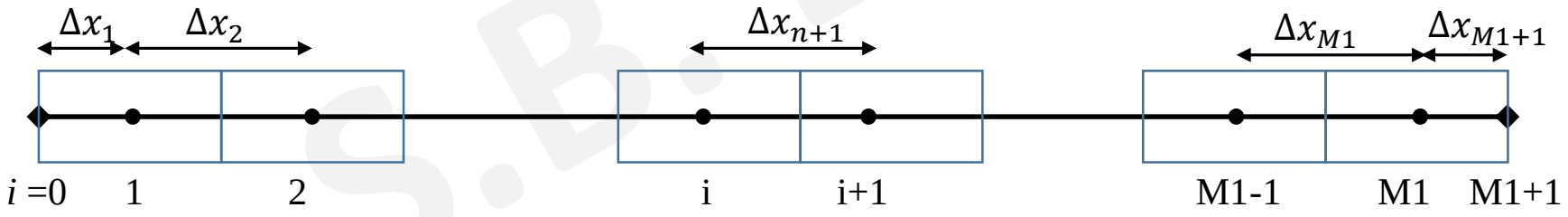
$S_i = \text{source}[i]$ internal field : i = 1 ~ M1

$A_{ij} = \text{coef}[i][j]$ internal field : i = 1 ~ M1

j = 1 ~ M1

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   double dx[M1+2], V[M1+1], Af[M1+2];
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source,dx,V,Af);
      calc_coef((double*)coef,source,dx,V,Af,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

```
void init_mesh(double* dx, double* V, double* Af)
{
   for(int i=1;i<=M1+1;i++) dx[i]=1.0/double(M1);
   dx[1]=0.5*dx[1];
   dx[M1+1]=0.5*dx[M1+1];
   for(int i=1;i<=M1;i++)
   {
      Af[i]=DY*DZ;
      V[i]=Af[i]*1.0/double(M1);
   }
   Af[M1+1]=DY*DZ;
}
```



```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   double dx[M1+2], V[M1+1], Af[M1+2];
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source,dx,V,Af);
      calc_coef((double*)coef,source,dx,V,Af,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

$$\phi_i^0 = 0$$

$$\text{phi}[i] = 0 \quad \text{for i} = 1 \sim \text{M1}$$

```
void init_phi(double* phi)
{
   for(int i=1;i<=M1;i++) phi[i]=0.0;
}
```

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   double dx[M1+2], V[M1+1], Af[M1+2];
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source,dx,V,Af);
      calc_coef((double*)coef,source,dx,V,Af,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

$$\begin{cases} \phi_0^{n+1} = -100 \\ \phi_{M1+1}^{n+1} = 200 \quad \text{or} \quad \left.\dfrac{\partial \phi}{\partial x}\right|_{M1+!}^{\text{n+1}} = 0 \end{cases}$$

$$\begin{cases} \text{phi}[0] = -100 \\ \text{phi}[M1+1] = 200 \quad \text{or} \quad \text{phi}[M1+1] = \text{phi}[M1] \end{cases}$$

```
void bcond(double* phi)
{
   phi[0]=-100.0;
//    phi[M1+1]=200.0;  // for Dirichlet at i=M1
   phi[M1+1]=phi[M1];  // for Neumann at i=M1
}
```

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   double dx[M1+2], V[M1+1], Af[M1+2];
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source,dx,V,Af);
      calc_coef((double*)coef,sourcedx,V,Af,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

$$S_i^{(n+1)} = \frac{\phi_i^{(n)}}{\Delta t} V_i$$

$$S_1^{(n+1)} = \frac{\phi_1^{(n)}}{\Delta t} V_i + \alpha \frac{A_{f_1}}{\Delta x_1} \phi_0^{(n+1)}$$

```
calc_source(double* phi, double* source, double* dx, double* V, double* Af)
{
   for(int i=1;i<=M1;i++) source[i]=1.0/DT*phi[i]*V[i];
   source[1]+=ALPHA*Af[1] *phi[0]/dx[1];
}
```

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   double dx[M1+2], V[M1+1], Af[M1+2];
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source,dx,V,Af);
      calc_coef((double*)coef,source,dx,V,Af,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

$$A_{ii} = \text{coef}[i][i] = \frac{1}{\Delta t} V_i + \frac{\alpha(A_{f_{i+1}} + A_{f_i})}{\Delta x}$$

$$A_{i-1,i} = \text{coef}[i-1][i] = -\frac{\alpha A_{f_i}}{\Delta x}$$

$$A_{i+1,i} = \text{coef}[i+1][i] = -\frac{\alpha A_{f_{i+1}}}{\Delta x}$$

$$A_{M1M1} = \text{coef}[M1][M1] = \frac{1}{\Delta t} V_{M1} + \frac{\alpha A_{f_{M1}}}{\Delta x}$$

```
void calc_coef(double* coef, double* source, double* dx, double* V, double* Af, const int& nTime)
{
   double coefRef[M1+1][M1+1];
   for(int i=1;i<=M1;i++) for(int j=1;j<=M1;j++) coefRef[i][j]=0.0;
   for(int i=1;i<=M1;i++)
   {
      coefRef[i][i]=1.0*V[i]/DT+ALPHA*(Af[i+1]/dx[i+1]+Af[i]/dx[i]);
      if(i!=1) coefRef[i-1][i]=-ALPHA*Af[i]/dx[i];
      if(i!=M1) coefRef[i+1][i]=-ALPHA*Af[i+1]/dx[i+1];
   }
   // for Neumann at i=M1
   coefRef[M1][M1]=1.0*V[M1]/DT+ALPHA*Af[M1]/dx[M1];
   for(int i=1;i<=M1;i++) for(int j=1;j<=M1;j++)
      coef[i*M1+j]=coefRef[i][j];
}
```

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   double dx[M1+2], V[M1+1], Af[M1+2];
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source,dx,V,Af);
      calc_coef((double*)coef,source,dx,V,Af,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

Gauss - Seidel solver

```
void solve(double* coef, double* phi, double* source, const int&
nTime)
{
   int nIter=1;
   double coefRef[M1+1][M1+1], oldPhi[M1+2], residual;

   for(int i=1;i<=M1;i++) for(int j=1;j<=M1;j++) coefRef[i][j]=coef[i*M1+j];
   do
   {
      for(int i=1;i<=M1;i++) oldPhi[i]=phi[i];
      for(int i=1;i<=M1;i++)
      {
         double sumLHS=0.0;
         for(int iLHS=1;iLHS<=M1;iLHS++)
            sumLHS+=coefRef[iLHS][i]*oldPhi[iLHS];
         sumLHS-=coefRef[i][i]*oldPhi[i];
         phi[i]=(source[i]-sumLHS)/coefRef[i][i];
      }
      residual=0.0;
      for(int i=1;i<=M1;i++) residual+=fabs(phi[i]-oldPhi[i])/double(M1);
      cout << nTime << " : RES = " << residual << ", ITER = "
           << nIter << endl;
      nIter++;
   } while(residual>0.0001);
}
```

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   double dx[M1+2], V[M1+1], Af[M1+2];
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source,dx,V,Af);
      calc_coef((double*)coef,source,dx,V,Af,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

writing function

```
void write_phi(double* phi)
{
   ofstream os("1D.dat");
   for(int i=1;i<=M1;i++) os << phi[i] << endl;
   os.close();
}
```

- If we does not want to use an array type for variables?
  : creating a new structure for variables, which has the same function with the array

Step 2. Object-oriented variables at a low level

- List class for variables
- procedure-oriented functions in main function

```
template <class T>
class List
{
private:
   T* v_;                  // pointer type for memory allocation
   int size_;              // list size (the same concept with array size)
public:
   List()
   :
      size_(0),
      v_(NULL)
   {}
   List(int s)
   :
      size_(0),
      v_(NULL)
   {
      this->setSize(s);
      size_=s;
   }
   List(int s, T v)
   :
      size_(0),
      v_(NULL)
   {
      this->setSize(s);
      size_=s;
      for(int i=0;i<size_;i++) v_[i]=v;
   }
   void setSize(const int newSize)
   {
      T* vv=new T[newSize];                  // memory allocation
      if(this->size_)                        // if resizing the previous List
      {
         int minSize = min(this->size_, newSize);
         T* v1=&this->v_[minSize];
         T* v2=&vv[minSize];
         while(minSize--) *--v2=*--v1;       // copy the previous data
      }
      if(this->v_) delete[] this->v_;        // delete the previous memory allocation
      this->size_=newSize;                   // size = new size
      this->v_=vv;                           // completion
   }
```

constructor without memory allocation

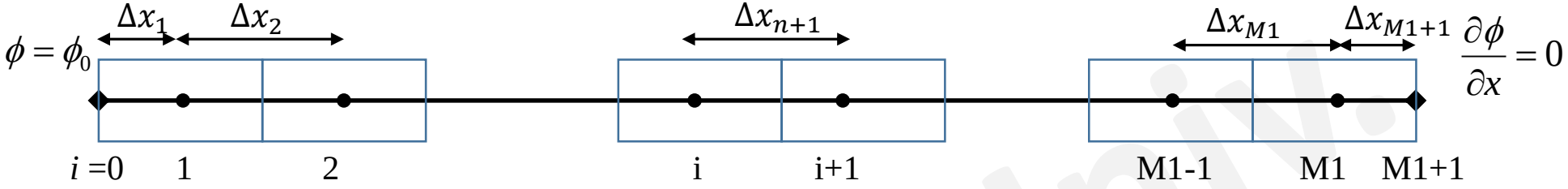
constructor with memory allocation

constructor with memory allocation and intialization

example

```
List<double> phi1;
List<int> phi2(10);
List<List<double> > phi3;

phi1.setSize(20);
phi3.setSize(15);

phi1[0]=1.5;
phi1[19]=28.7;
phi2[0]=2;
phi2[9]=100;

for(int i=0;i<phi3.size();i++) phi3[i].setSize(7);

phi3[0][0]=12.78;
phi3[14][6]=-213.123;
```

```
   T& operator[](int s)          // operator overloading
   {                             // for the same expression with array
      return v_[s];
   }
   int size()
   {
      return size_;
   }
};
```



$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
   List<double> phi(M1+2), source(M1+1), dx(M1+2), V(M1+1), Af(M1+2);
   List<List<double> > coef(M1+1);
   for(int i=1;i<coef.size();i++) coef[i].setSize(M1+1);
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source,dx,V,Af);
      calc_coef(coef,source,dx,V,Af,nTime);
      solve(coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

```
int main()
{
   double phi[M1+2], source[M1+1], coef[M1+1][M1+1];
   double dx[M1+2],V[M1+1], Af[M1+2];
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source,dx,V,Af);
      calc_coef((double*)coef,source,dx,V,Af,nTime);
      solve((double*)coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

```
void init_mesh(List<double>& dx, List<double>& V, List<double>& Af)
{
   for(int i=1;i<dx.size();i++) dx[i]=1.0/double(V.size()-1);
   dx[1]=0.5*dx[1];
   dx[dx.size()-1]=0.5*dx[dx.size()-1];
   for(int i=1;i<V.size();i++)
   {
      Af[i]=DY*DZ;
      V[i]=Af[i]*1.0/double(V.size()-1);
   }
   Af[Af.size()-1]=DY*DZ;
}

void init_phi(List<double>& phi)
{
   for(int i=1;i<phi.size()-1;i++) phi[i]=0.0;
}  // phi.size()=M1+2

void bcond(List<double>& phi)
{
   phi[0]=-100.0;
   phi[phi.size()-1]=200.0;    // for Dirichlet at i=M1
//    phi[phi.size()-1]=phi[phi.size()-2];     // for Neumann at i=M1
}

void calc_source(List<double> phi, List<double>& source)
{
   for(int i=1;i<source.size();i++)  // soure.size()=M1+1
   {
      source[i]=1.0/DT*phi[i]*V[i];
   }
   source[1]+=ALPHA*Af[1]*phi[0]/dx[1];
}
```

```
void init_mesh(double* dx, double* V, double* Af)
{
   for(int i=1;i<=M1+1;i++) dx[i]=1.0/double(M1);
   dx[1]=0.5*dx[1];
   dx[M1+1]=0.5*dx[M1+1];
   for(int i=1;i<=M1;i++)
   {
      Af[i]=DY*DZ;
      V[i]=Af[i]*1.0/double(M1);
   }
   Af[M1+1]=DY*DZ;
}

void init_phi(double* phi)
{
   for(int i=1;i<=M1;i++) phi[i]=0.0;
}

void bcond(double* phi)
{
   phi[0]=-100.0;
//    phi[M1+1]=200.0;  // for Dirichlet at i=M1
   phi[M1+1]=phi[M1];  // for Neumann at i=M1
}

void calc_source(double* phi, double* source)
{
   for(int i=1;i<=M1;i++)
   {
      source[i]=1.0/DT*phi[i]*V[i];
   }
   source[1]+=ALPHA*Af[1]*phi[0]/dx[1];
}
```

```
void calc_coef(List<List<double> >& coef, List<double> source,
List<double>& dx, List<double>& V, List<double>& Af, const int& nTime)
{  // coef.size()=M1+1
   for(int i=1;i<coef.size();i++) for(int j=1;j<coef[i].size();j++)
      coef[i][j]=0.0;
   for(int i=1;i<coef.size();i++)
   {
      coef[i][i]=1.0*V[i]/DT+ALPHA*(Af[i+1]/dx[i+1]+Af[i]/dx[i]);
      if(i!=1) coef[i-1][i]=-ALPHA*Af[i]/dx[i];
      if(i!=coef.size()-1) coef[i+1][i]=-ALPHA*Af[i+1]/dx[i+1];
   }
   coef[coef.size()-1][coef.size()-1]=1.0*V[coef.size()-1]/DT+
                                      ALPHA*Af[coef.size()-1]/dx[coef.size()-1];
}
```

```
void calc_coef(double* coef, double* source, const int& nTime)
{
   double coefRef[M1+1][M1+1];
   for(int i=1;i<=M1;i++) for(int j=1;j<=M1;j++)
      coefRef[i][j]=0.0;
   for(int i=1;i<=M1;i++)
   {
      coefRef[i][i]=1.0*V[i]/DT+ALPHA*(Af[i+1]/dx[i+1]+Af[i]/dx[i]);
      if(i!=1) coefRef[i-1][i]=-ALPHA*Af[i]/dx[i];
      if(i!=M1) coefRef[i+1][i]=-ALPHA*Af[i+1]/dx[i+1];
   }
   coefRef[M1][M1]=1.0*V[M1]/DT+ALPHA*Af[M1]/dx[M1];

for(int i=1;i<=M1;i++) for(int j=1;j<=M1;j++)
      coef[i*M1+j]=coefRef[i][j];
}
```

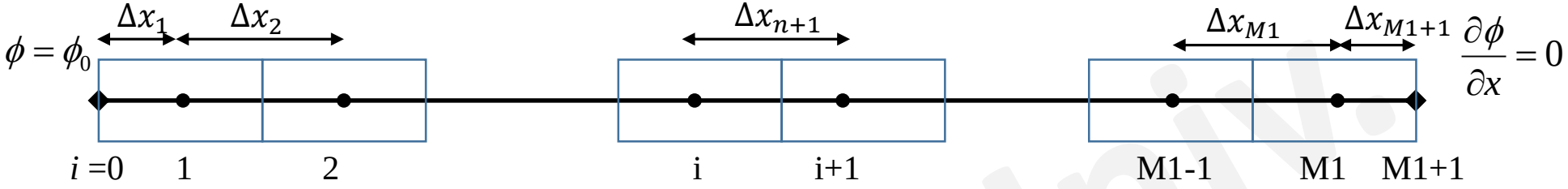

similar coding in solve, write functions

- If we want to use an object for matrice?

Step 3. Object-oriented matrix

- fvMatrix class including coefficient & source matrices
- procedure oriented functions for initializing & writing variables
- fvMatrix object oriented functions

```
class fvMatrix
{
private:
   List<List<double> > coef_;                                  // coefficient matrix
   List<double> source_;                                       // source matrix
public:
   fvMatrix()
   {}
   fvMatrix(List<double> phi)
   {
      coef_.setSize(phi.size()-1);                             // set the size of coefficient matrix
      for(int i=0;i<coef_.size();i++) coef_[i].setSize(phi.size()-1);
      source_.setSize(phi.size()-1);                           // set the size of source matrix
      for(int i=1;i<coef_.size();i++)
      {
         source_[i]=0.0;                                       // initialize the source matrix
         for(int j=1;j<coef_[i].size();j++) coef_[i][j]=0.0;   // initialize the coefficient matrix
      }
   }
   void bcond(List<double>& phi)
   {
   }
   void calc_source(List<double>& phi)
   {
   }
   void calc_coef(const int& nTime)
   {
   }
   void solve(List<double>& phi, const int& nTime)
   {
   }
};
```



$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
   List<double> phi(M1+2), source(M1+1), dx(M1+2), V(M1+1), Af(M1+2);
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      fvMatrix phiEqn(phi);
      phiEqn.bcond(phi);
      phiEqn.calc_source(phi,dx,V,Af);
      phiEqn.calc_coef(dx,V,Af,nTime);
      phiEqn.solve(phi,nTime);
   }
   write_phi(phi);
   return 0;
}
```

```
int main()
{
   List<double> phi(M1+2), source(M1+1), dx(M1+2), V(M1+1), Af(M1+2);
   List<List<double> > coef(M1+1);
   for(int i=1;i<coef.size();i++) coef[i].setSize(M1+1);
   init_mesh(dx,V,Af);
   init_phi(phi);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      bcond(phi);
      calc_source(phi,source,dx,V,Af);
      calc_coef(coef,source,dx,V,Af,nTime);
      solve(coef,phi,source,nTime);
   }
   write_phi(phi);
   return 0;
}
```

```
 void bcond(List<double>& phi)
   {
      phi[0]=-100.0;
      phi[phi.size()-1]=phi[phi.size()-2];
   }
   void calc_source(List<double>& phi, List<double>& dx, List<double>& V,
List<double>& Af)
   {
      for(int i=1;i<source_.size();i++)
      {
         source_[i]=1.0/DT*phi[i]*V[i];
      }
      source_[1]+=ALPHA*Af[1]*phi[0]/dx[1];
   }
   void calc_coef(List<double>& dx, List<double>& V, List<double>& Af, const int&
nTime)
   {
      for(int i=1;i<coef_.size();i++)
      {
         coef_[i][i]=1.0*V[i]/DT+ALPHA*(Af[i+1]/dx[i+1]+Af[i]/dx[i]);
         if(i!=1) coef_[i-1][i]=-ALPHA*Af[i]/dx[i];
         if(i!=coef_.size()-1) coef_[i+1][i]=-ALPHA*Af[i+1]/dx[i+1];
      }
      coef_[coef_.size()-1][coef_.size()-1]=1.0*V[coef_.size()-
1]/DT+ALPHA*Af[coef_.size()-1]/dx[coef_.size()-1];   // for Neumann at i=M1
   }
```

in fvMatrix class

```
void bcond(List<double>& phi)
{
   phi[0]=-100.0;
   phi[phi.size()-1]=phi[phi.size()-2];      // for Neumann at i=M1
}
void calc_source(List<double> phi, List<double>& source, List<double>& dx,
List<double>& V, List<double>& Af)
{
   for(int i=1;i<source.size();i++)
   {
      source[i]=1.0/DT*phi[i]*V[i];
   }
   source[1]+=ALPHA*Af[1]*phi[0]/dx[1];
}
void calc_coef(List<List<double> >& coef, List<double> source, List<double>& dx,
List<double>& V, List<double>& Af, const int& nTime)
{
   for(int i=1;i<coef.size();i++) for(int j=1;j<coef[i].size();j++) coef[i][j]=0.0;
   for(int i=1;i<coef.size();i++)
   {
      coef[i][i]=1.0*V[i]/DT+ALPHA*(Af[i+1]/dx[i+1]+Af[i]/dx[i]);
      if(i!=1) coef[i-1][i]=-ALPHA*Af[i]/dx[i];
      if(i!=coef.size()-1) coef[i+1][i]=-ALPHA*Af[i+1]/dx[i+1];
   }
   coef[coef.size()-1][coef.size()-1]=1.0*V[coef.size()-1]/DT+ALPHA*Af[coef.size()-
1]/dx[coef.size()-1];   // for Neumann at i=M1
}
```

in main function

```
void solve(List<double>& phi, const int& nTime)
{
   int nIter=1;
   double residual;
   List<double> oldPhi(phi.size());
   do
   {
      for(int i=1;i<phi.size()-1;i++) oldPhi[i]=phi[i];
      for(int i=1;i<phi.size()-1;i++)
      {
         double sumLHS=0.0;
         for(int iLHS=1;iLHS<oldPhi.size()-1;iLHS++)
            sumLHS+=coef_[iLHS][i]*oldPhi[iLHS];
         sumLHS-=coef_[i][i]*oldPhi[i];
         phi[i]=(source_[i]-sumLHS)/coef_[i][i];
      }
      residual=0.0;
      for(int i=1;i<phi.size()-1;i++)
         residual+=fabs(phi[i]-oldPhi[i])/double(phi.size()-2);
      cout << nTime << " : RES = " << residual << ", ITER = " << nIter << endl;
      nIter++;
   } while(residual>0.0001);
}
```

in fvMatrix class

```
void solve(List<List<double> > coef, List<double>& phi, List<double> source,
           const int& nTime)
{
   int nIter=1;
   double residual;
   List<double> oldPhi(phi.size());
   do
   {
      for(int i=1;i<phi.size()-1;i++) oldPhi[i]=phi[i];
      for(int i=1;i<phi.size()-1;i++)
      {
         double sumLHS=0.0;
         for(int iLHS=1;iLHS<oldPhi.size()-1;iLHS++)
            sumLHS+=coef[iLHS][i]*oldPhi[iLHS];
         sumLHS-=coef[i][i]*oldPhi[i];
         phi[i]=(source[i]-sumLHS)/coef[i][i];
      }
      residual=0.0;
      for(int i=1;i<phi.size()-1;i++)
         residual+=fabs(phi[i]-oldPhi[i])/double(phi.size()-2);
      cout << nTime << " : RES = " << residual << ", ITER = " << nIter << endl;
      nIter++;
   } while(residual>0.0001);
}
```

in main function

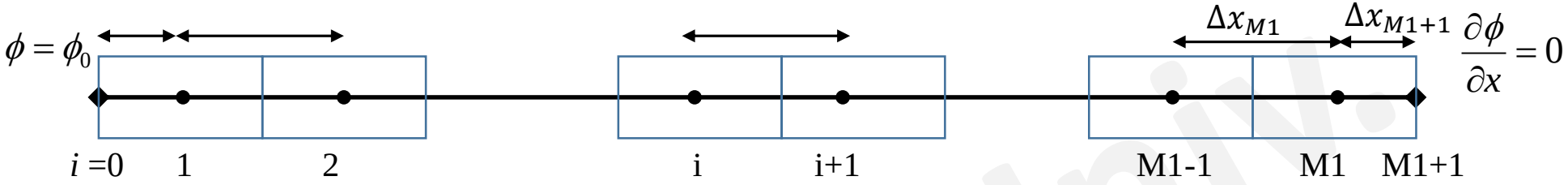
- If we want to distinguish mesh & variables from the main function?

Step 4. Object-oriented variables at a middle level

- fvMesh class for grid information
- GeometricField class for variables

```
class fvMesh
{
private:
   int nCells_;
   double DY_;
   double DZ_;
   List<double> dx_;
   List<double> V_;
   List<double> Af_;
public:
   fvMesh()
   :
      nCells_(10),
      DY_(1.0),
      DZ_(1.0),
      dx_(nCells_+2),
      V_(nCells_+1),
      Af_(nCells_+2)
   {
      for(int i=1;i<dx_.size();i++) dx_[i]=1.0/double(nCells_);
      dx_[1]=0.5*dx_[1];
      dx_[dx_.size()-1]=0.5*dx_[dx_.size()-1];
      for(int i=1;i<V_.size();i++)
      {
         Af_[i]=DY_*DZ_;
         V_[i]=Af_[i]*1.0/double(nCells_);
      }
      Af_[Af_.size()-1]=DY_*DZ_;
   }
   List<double>& dx()
   {
      return dx_;
   }
   List<double>& V()
   {
      return V_;
   }
   List<double>& Af()
   {
      return Af_;
   }
   int nCells()
   {
      return nCells_;
   }
};
```

```
template <class T>
class GeometricField
:
   public List<T>
{
private:
   fvMesh mesh_;
public:
   GeometricField(fvMesh& mesh)
   :
      List<T>(mesh.nCells()+2),
      mesh_(mesh)
   {}
   fvMesh& mesh()
   {
      return mesh_;
   }
   void bcond()
   {
      this->operator[](0)=-100.0;
      this->operator[](this->size()-1)=this->operator[](this->size()-2);
   }
   void init()
   {
      for(int i=1;i<this->size()-1;i++) this->operator[](i)=0.0;
   }
   void write()
   {
      ofstream os("phi");
      for(int i=1;i<this->size()-1;i++) os << this->operator[](i) << endl;
      os.close();
   }
};
```



$$\begin{bmatrix} \frac{V_1}{\Delta t}+\alpha\left(\frac{A_{f_1}}{\Delta x_1}+\frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t}+\alpha\left(\frac{A_{f_2}}{\Delta x_2}+\frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t}+\alpha\left(\frac{A_{f_3}}{\Delta x_3}+\frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t}+\alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
  fvMesh mesh;
  GeometricField<double> phi(mesh);
  phi.init();
  for(int nTime=0;nTime<NTST;nTime++)
  {
    phi.bcond();
    fvMatrix phiEqn(phi);
    phiEqn.calc_source(phi);
    phiEqn.calc_coef(nTime);
    phiEqn.solve(phi,nTime);
  }
  phi.write();
  return 0;
}
```

```
int main()
{
  List<double> phi(M1+2), source(M1+1), dx(M1+2), V(M1+1), Af(M1+2);
  init_mesh(dx,V,Af);
  init_phi(phi);
  for(int nTime=0;nTime<NTST;nTime++)
  {
    fvMatrix phiEqn(phi);
    phiEqn.bcond(phi);
    phiEqn.calc_source(phi,dx,V,Af);
    phiEqn.calc_coef(dx,V,Af,nTime);
    phiEqn.solve(phi,nTime);
  }
  write_phi(phi);
  return 0;
}
```

```
fvMatrix(GeometricField<double> phi)
:
    mesh_(phi.mesh())
{
    coef_.setSize(phi.size()-1);
    for(int i=0;i<coef_.size();i++) coef_[i].setSize(phi.size()-1);
    source_.setSize(phi.size()-1);
    for(int i=1;i<coef_.size();i++)
    {
        source_[i]=0.0;
        for(int j=1;j<coef_[i].size();j++) coef_[i][j]=0.0;
    }
}
void calc_source(GeometricField<double>& phi)
{
    for(int i=1;i<source_.size();i++)
    {
        source_[i]=1.0/DT*phi[i]*mesh_.V()[i];
    }
    source_[1]+=ALPHA*mesh_.Af()[1]*phi[0]/mesh_.dx()[1];
}
void calc_coef(const int& nTime)
{
    for(int i=1;i<coef_.size();i++)
    {
        coef_[i][i]=1.0*mesh_.V()[i]/DT
            +ALPHA*(mesh_.Af()[i+1]/mesh_.dx()[i+1]+mesh_.Af()[i]/mesh_.dx()[i]);
        if(i!=1) coef_[i-1][i]=-ALPHA*mesh_.Af()[i]/mesh_.dx()[i];
        if(i!=coef_.size()-1) coef_[i+1][i]=-ALPHA*mesh_.Af()[i+1]/mesh_.dx()[i+1];
    }
    coef_[coef_.size()-1][coef_.size()-1]=1.0*mesh_.V()[coef_.size()-1]/DT
        +ALPHA*mesh_.Af()[coef_.size()-1]/mesh_.dx()[coef_.size()-1];
}
```

in fvMatrix class

```
fvMatrix(List<double> phi)
{
    coef_.setSize(phi.size()-1);
    for(int i=0;i<coef_.size();i++) coef_[i].setSize(phi.size()-1);
    source_.setSize(phi.size()-1);
    for(int i=1;i<coef_.size();i++)
    {
        source_[i]=0.0;
        for(int j=1;j<coef_[i].size();j++) coef_[i][j]=0.0;
    }
}
void bcond(List<double>& phi)
{
    phi[0]=-100.0;
    // phi[phi.size()-1]=200.0;      // for Dirichlet
    phi[phi.size()-1]=phi[phi.size()-2];
}
void calc_source(List<double>& phi, List<double>& dx, List<double>& V,
List<double>& Af)
{
    for(int i=1;i<source_.size();i++)
    {
        source_[i]=1.0/DT*phi[i]*V[i];
    }
    source_[1]+=ALPHA*Af[1]*phi[0]/dx[1];
}
void calc_coef(List<double>& dx, List<double>& V, List<double>& Af, const int&
nTime)
{
    for(int i=1;i<coef_.size();i++)
    {
        coef_[i][i]=1.0*V[i]/DT+ALPHA*(Af[i+1]/dx[i+1]+Af[i]/dx[i]);
        if(i!=1) coef_[i-1][i]=-ALPHA*Af[i]/dx[i];
        if(i!=coef_.size()-1) coef_[i+1][i]=-ALPHA*Af[i+1]/dx[i+1];
    }
    coef_[coef_.size()-1][coef_.size()-1]=1.0*V[coef_.size()-1]/DT
        +ALPHA*Af[coef_.size()-1]/dx[coef_.size()-1];
}
```

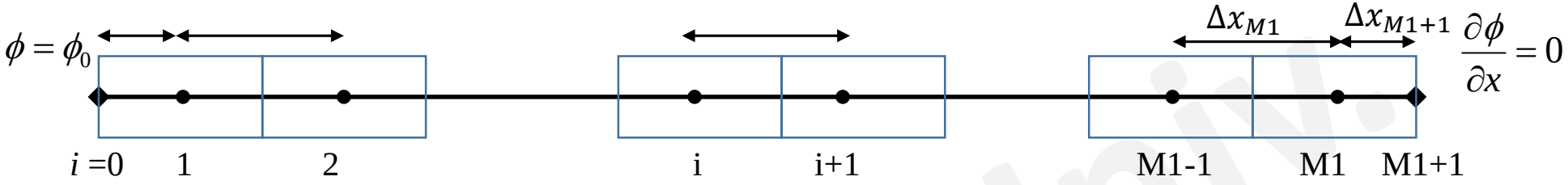
- If we want to read information of variables from files?

Step 5. Object-oriented variables with I/O object class

- IOobject class for input/output function
- GeometricField class for reading fields from the IOobject file

```
class IOobject
{
private:
   string fileName_;
   string pathName_;
   bool readOption_;
   bool writeOption_;
public:
   IOobject
   (
      string fileName,
      string pathName,
      bool readOption=false,
      bool writeOption=true
   )
   :
      fileName_(fileName),
      pathName_(pathName),
      readOption_(readOption),
      writeOption_(writeOption)
   {}
   string fileName()
   {
      return fileName_;
   }
   string pathName()
   {
      return pathName_;
   }
   bool readOption()
   {
      return readOption_;
   }
   bool writeOption()
   {
      return writeOption_;
   }
};
```

constructor with initialization of private variables



$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
   fvMesh mesh;
   GeometricField<double> phi(IOobject("phi",  ".", true, true) ,mesh);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      phi.bcond();
      fvMatrix phiEqn(phi);
      phiEqn.calc_source(phi);
      phiEqn.calc_coef(nTime);
      phiEqn.solve(phi,nTime);
   }
   phi.write();
   return 0;
}
```

```
int main()
{
   fvMesh mesh;
   GeometricField<double> phi(mesh);
   phi.init();
   for(int nTime=0;nTime<NTST;nTime++)
   {
      phi.bcond();
      fvMatrix phiEqn(phi);
      phiEqn.calc_source(phi);
      phiEqn.calc_coef(nTime);
      phiEqn.solve(phi,nTime);
   }
   phi.write();
   return 0;
}
```

```
template <class T>
class GeometricField
:
   public List<T>
{
private:
   fvMesh mesh_;
   IOobject obj_;
public:
   GeometricField(const IOobject& obj, fvMesh& mesh)
   :
      List<T>(mesh.nCells()+2),
      mesh_(mesh),
      obj_(obj)
   {
      readField("internalField");  // the same function with init()
   }
   fvMesh mesh()
   {
      return mesh_;
   }
   void readField(const string& fieldDictEntry)
   {
      if(obj_.readOption())
      {
         string firstToken;
         string fileName=obj_.pathName()+"/"+obj_.fileName();
         ifstream is(fileName.c_str());
         is >> firstToken;
         if(firstToken==fieldDictEntry)
         {
            is >> firstToken;
            if(firstToken=="uniform") operator=(is);
            else if(firstToken=="nonUniform") operator>>(is);
         }
         is.close();
      }
   }
```

"phi" file : internalField uniform 0.0

```
template <class T>
class GeometricField
:
   public List<T>
{
private:
   fvMesh mesh_;
public:
   GeometricField(fvMesh& mesh)
   :
      List<T>(mesh.nCells()+2),
      mesh_(mesh)
   {}
   fvMesh mesh()
   {
      return mesh_;
   }
   void init()
   {
      for(int i=1;i<this->size()-1;i++) this->operator[](i)=0.0;
   }
};
```

no input file,
to change the initialization value
the source file has to be re-compiled

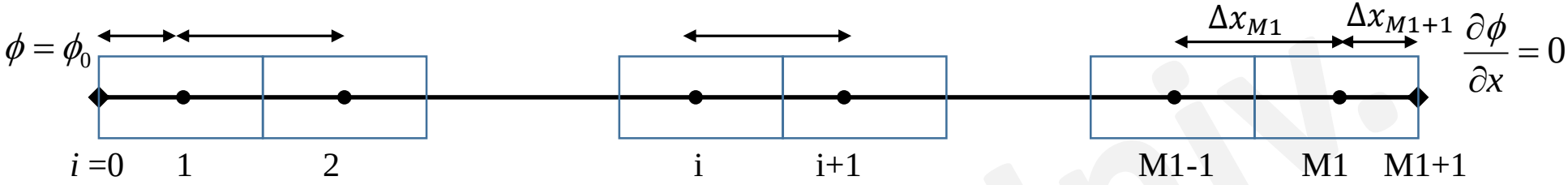
```
ifstream& operator=(ifstream& is)          // operator overloading for uniform value
 {
   T v;
   is >> v;
   for(int i=1;i<this->size()-1;i++) this->operator[](i)=v;
   return is;
 }
 ifstream& operator>>(ifstream& is)        // operator overloading for non-uniform values
 {
   T v;
   for(int i=1;i<this->size()-1;i++)
   {
      is >> v;
      this->operator[](i)=v;
   }
   return is;
 }
```

in GeometricField class

- If we want to use a mathematical description for govering equations?

Step 6. Object-oriented matrix

- fvMatrix class for using mathematical description



$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
   fvMesh mesh;
   GeometricField<double> phi(IOobject("phi",  ".", true, true) ,mesh);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      phi.bcond();
      fvMatrix phiEqn(phi);
      phiEqn.ddt(phi);
      phiEqn.laplacian(phi);
      phiEqn.solve(phi,nTime);
   }
   phi.write();
   return 0;
}
```

```
int main()
{
   fvMesh mesh;
   GeometricField<double> phi(IOobject("phi",  ".", true, true) ,mesh);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      phi.bcond();
      fvMatrix phiEqn(phi);
      phiEqn.calc_source(phi);
      phiEqn.calc_coef(nTime);
      phiEqn.solve(phi,nTime);
   }
   phi.write();
   return 0;
}
```

$$\begin{bmatrix} \frac{V_1}{\Delta t}+\alpha\left(\frac{A_{f_1}}{\Delta x_1}+\frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t}+\alpha\left(\frac{A_{f_2}}{\Delta x_2}+\frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t}+\alpha\left(\frac{A_{f_3}}{\Delta x_3}+\frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & & \vdots & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t}+\alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

$$\left(A\Big|_{ddt} - A\Big|_{laplacian}\right) \times \phi = \left(S\Big|_{ddt} - S\Big|_{laplacian}\right)$$

$$\left(A\Big|_{ddt} - A\Big|_{laplacian}\right) \times \phi = \left(S\Big|_{ddt} - S\Big|_{laplacian}\right)$$

$$A\Big|_{ddt} = \begin{bmatrix} \frac{V_1}{\Delta t} & 0 & 0 & 0 & 0 \\ 0 & \frac{V_2}{\Delta t} & 0 & 0 & 0 \\ 0 & 0 & \frac{V_3}{\Delta t} & 0 & 0 \\ & & & \vdots & \\ 0 & 0 & 0 & 0 & \frac{V_{M1}}{\Delta t} \end{bmatrix} \quad -A\Big|_{laplacian} = \begin{bmatrix} -\alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & \frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ \frac{\alpha A_{f_2}}{\Delta x_2} & -\alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & \frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & \frac{\alpha A_{f_3}}{\Delta x_3} & -\alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & \frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & & \vdots & \\ 0 & 0 & 0 & \frac{\alpha A_{f_{M1}}}{\Delta x} & -\alpha\left(\frac{A_{f_{M1}}}{\Delta x_{M1}}\right) \end{bmatrix}$$

$$S\Big|_{ddt} = \begin{bmatrix} \frac{V_1}{\Delta t}\phi_1^{(n)} \\ \frac{V_2}{\Delta t}\phi_2^{(n)} \\ \frac{V_3}{\Delta t}\phi_3^{(n)} \\ \vdots \\ \frac{V_n}{\Delta t}\phi_{M1}^{(n)} \end{bmatrix} \quad -S\Big|_{laplacian} = \begin{bmatrix} -\frac{\alpha A_{f_1}}{\Delta x_1}\phi_0^{(n+1)} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

```
void ddt(GeometricField<double>& phi)
 {
   for(int i=1;i<source_.size();i++)
   {
     source_[i]+=1.0/DT*phi[i]*mesh_.V()[i];
     coef_[i][i]+=1.0*mesh_.V()[i]/DT;
   }
 }
 void laplacian(GeometricField<double>& phi)
 {
   for(int i=1;i<coef_.size();i++)
   {
     source_[i]+=0.0;
     coef_[i][i]
          +=ALPHA*(mesh_.Af()[i+1]/mesh_.dx()[i+1]+mesh_.Af()[i]/mesh_.dx()[i]);
     if(i!=1) coef_[i-1][i]+=-ALPHA*mesh_.Af()[i]/mesh_.dx()[i];
     if(i!=coef_.size()-1) coef_[i+1][i]+=-ALPHA*mesh_.Af()[i+1]/mesh_.dx()[i+1];
   }
   source_[1]+=ALPHA*mesh_.Af()[1]*phi[0]/mesh_.dx()[1];
   coef_[coef_.size()-1][coef_.size()-1]+=
          -ALPHA*mesh_.Af()[coef_.size()]/mesh_.dx()[coef_.size()];
 }
```

```
void calc_source(GeometricField<double>& phi)
 {
   for(int i=1;i<source_.size();i++)
   {
     source_[i]=1.0/DT*phi[i]*mesh_.V()[i];
   }
   source_[1]+=ALPHA*mesh_.Af()[1]*phi[0]/mesh_.dx()[1];
 }
 void calc_coef(const int& nTime)
 {
   for(int i=1;i<coef_.size();i++)
   {
     coef_[i][i]=1.0*mesh_.V()[i]/DT
          +ALPHA*(mesh_.Af()[i+1]/mesh_.dx()[i+1]+mesh_.Af()[i]/mesh_.dx()[i]);
     if(i!=1) coef_[i-1][i]=-ALPHA*mesh_.Af()[i]/mesh_.dx()[i];
     if(i!=coef_.size()-1) coef_[i+1][i]=-ALPHA*mesh_.Af()[i+1]/mesh_.dx()[i+1];
   }
   coef_[coef_.size()-1][coef_.size()-1]=1.0*mesh_.V()[coef_.size()-1]/DT
          +ALPHA*mesh_.Af()[coef_.size()-1]/mesh_.dx()[coef_.size()-1];
 }
```

in fvMatrix class

- If we want to mimic a mathematical formulation?

Step 7. Object-oriented matrix

- tmp class instead of using pointer variables
- fvMatrix class for mimicking a mathematical formulation

```
namespace fvm
{
  fvMatrix& ddt(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    for(int i=1;i<rfvm.source().size();i++)
    {
      rfvm.source()[i]+=1.0/DT*phi[i]*phi.mesh().V()[i];
      rfvm.coef()[i][i]+=1.0*phi.mesh().V()[i]/DT;
    }
    return rfvm;
  }
  fvMatrix& laplacian(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    for(int i=1;i<rfvm.source().size();i++)
    {
      rfvm.source()[i]=0.0;
      rfvm.coef()[i][i]=-ALPHA*(phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1]+phi.mesh().Af()[i]/phi.mesh().dx()[i]);
      if(i!=1) rfvm.coef()[i-1][i]=ALPHA*phi.mesh().Af()[i]/phi.mesh().dx()[i];
      if(i!=rfvm.coef().size()-1) rfvm.coef()[i+1][i]=ALPHA*phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1];;
    }
    rfvm.source()[1]+=-ALPHA*phi.mesh().Af()[1]*phi[0]/phi.mesh().dx()[1];
    rfvm.coef()[rfvm.coef().size()-1][rfvm.coef().size()-1]
        +=ALPHA*phi.mesh().Af()[rfvm.coef().size()]/phi.mesh().dx()[rfvm.coef().size()];     // for Neumann at i=M1
    return rfvm;
  }
};
```

fvm namespace

: collection of mathematical formulations

```
template<class T>
class tmp
{
private:
  T* ptr_;
public:
  tmp(T* tPtr)
  :
    ptr_(tPtr)
  {}
  T& ref() const
  {
    return *ptr_;
  }
};
```

tmp class

: instead of using pointer variables

fvMatrix phiEqn

coef_, source_ = [0]

ddt()

$$coef\_ += A|_{ddt}$$

$$source\_ += S|_{ddt}$$

laplacian()

$$coef\_ -= A|_{laplacian}$$

$$source\_ -= S|_{laplacian}$$

```
int main()
{
    fvMatrix phiEqn;
    phiEqn.ddt();
    phiEqn.laplacian();
}
```

fvm:ddt()

tmp<fvMatrix> tFvMatrix

coef_, source_ = [0]

$$coef\_ = A|_{ddt}$$

$$source\_ = S|_{ddt}$$

return tFvMatrix.ref()

\-

operator overloading

fvm:laplacian()

tmp<fvMatrix> tFvMatrix

coef_, source_ = [0]

$$coef\_ = A|_{laplacian}$$

$$source\_ = S|_{laplacian}$$

return tFvMatrix.ref()

```
int main()
{
    fvMatrix phiEqn(fvm::ddt()-fvm::laplacian());
}
```

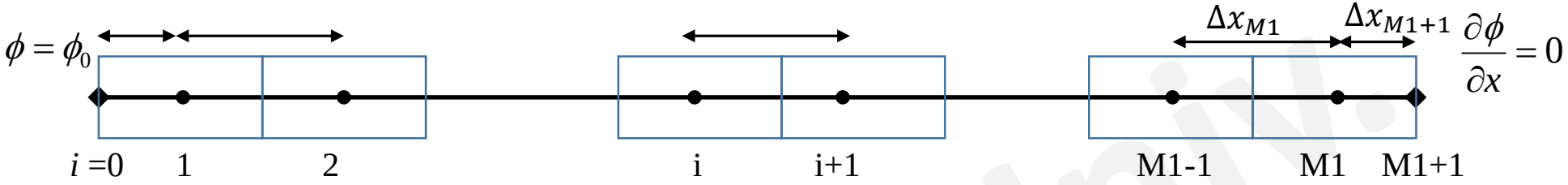
fvMatrix& operator-(fvmA, fvmB)

```
fvmA.coef_ = fvmA.coef_ - fvmB.coef_
fvmA.source_ = fvmA.source_ - fvmB.source_
return fvmA
```

fvMatrix phiEqn

coef_, source_

```
fvMatrix(fvMatrix& rfvm)
coef_=rfvm.coef_, source_=rfvm.source_
```



$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        phi.bcond();
        fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
        phiEqn.solve(phi,nTime);
    }
    phi.write();
    return 0;
}
```

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        phi.bcond();
        fvMatrix phiEqn(phi);
        phiEqn.ddt(phi);
        phiEqn.laplacian(phi);
        phiEqn.solve(phi,nTime);
    }
    phi.write();
    return 0;
}
```

```
class fvMatrix
{
private:
   List<List<double> > coef_;
   List<double> source_;
public:
   fvMatrix()
   {}
   fvMatrix(GeometricField<double> phi)
   {
      coef_.setSize(phi.size()-1);
      for(int i=0;i<coef_.size();i++) coef_[i].setSize(phi.size()-1);
      source_.setSize(phi.size()-1);
      for(int i=1;i<coef_.size();i++)
      {
         source_[i]=0.0;
         for(int j=1;j<coef_[i].size();j++) coef_[i][j]=0.0;
      }
   }
   fvMatrix(fvMatrix& rfvm)
   {
      coef_.setSize(rfvm.coef().size());
      for(int i=0;i<coef_.size();i++) coef_[i].setSize(rfvm.coef()[i].size());
      source_.setSize(rfvm.source().size());
      for(int i=1;i<coef_.size();i++)
      {
         source_[i]=rfvm.source()[i];
         for(int j=1;j<coef_[i].size();j++) coef_[i][j]=rfvm.coef()[i][j];
      }
   }
   friend fvMatrix& operator-(fvMatrix& rfvmA, fvMatrix& rfvmB)
   {
      for(int i=1;i<rfvmA.coef().size();i++)
      {
         rfvmA.source()[i]-=rfvmB.source()[i];
         for(int j=1;j<rfvmA.coef()[i].size();j++) rfvmA.coef()[i][j]-=rfvmB.coef()[i][j];
      }
      return rfvmA;
   }
```

in fvMatrix class

```
class fvMatrix
{
private:
   List<List<double> > coef_;
   List<double> source_;
   fvMesh mesh_;
public:
   fvMatrix()
   {}
   fvMatrix(GeometricField<double> phi)
   :
      mesh_(phi.mesh())
   {
      coef_.setSize(phi.size()-1);
      for(int i=0;i<coef_.size();i++) coef_[i].setSize(phi.size()-1);
      source_.setSize(phi.size()-1);
      for(int i=1;i<coef_.size();i++)
      {
         source_[i]=0.0;
         for(int j=1;j<coef_[i].size();j++) coef_[i][j]=0.0;
      }
   }
   void ddt(GeometricField<double>& phi)
   {
      for(int i=1;i<source_.size();i++)
      {
         source_[i]+=1.0/DT*phi[i]*mesh_.V()[i];
         coef_[i][i]+=1.0*mesh_.V()[i]/DT;
      }
   }
   void laplacian(GeometricField<double>& phi)
   {
      for(int i=1;i<coef_.size();i++)
      {
         source_[i]+=0.0;
         coef_[i][i]+=ALPHA*(mesh_.Af()[i+1]/mesh_.dx()[i+1]+mesh_.Af()[i]/mesh_.dx()[i]);
         if(i!=1) coef_[i-1][i]+=-ALPHA*mesh_.Af()[i]/mesh_.dx()[i];
         if(i!=coef_.size()-1) coef_[i+1][i]+=-ALPHA*mesh_.Af()[i+1]/mesh_.dx()[i+1];
      }
      source_[1]+=ALPHA*mesh_.Af()[1]*phi[0]/mesh_.dx()[1];
      coef_[coef_.size()-1][coef_.size()-1]+=-ALPHA*mesh_.Af()[coef_.size()]/mesh_.dx()[coef_.size()];
   }
```

- If we want to use several types for boundary?

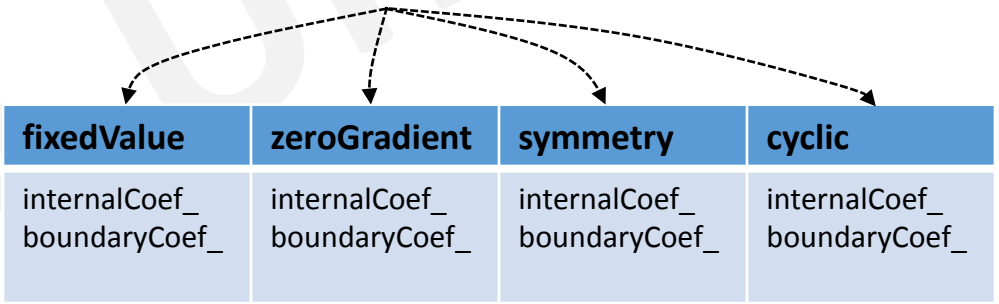
➢ If we want to use several types for boundary?

procedure-oriented coding

```
if(type==“fixedValue”)
  … modifying source_ & coef_ matrices

else if(type==“zeroGradient”)
  … modifying source_ & coef_ matrices

else if(type==“symmetry”)
  … modifying source_ & coef_ matrices

else if(type==“cyclic”)
  … modifying source_ & coef_ matrices
.
.
.
```

Object-oriented coding

variable.boundary[i].type()



fixedValue	zeroGradient	symmetry	cyclic
internalCoef_ boundaryCoef_	internalCoef_ boundaryCoef_	internalCoef_ boundaryCoef_	internalCoef_ boundaryCoef_

1. Two matrices determined by boundary conditions
2. Matrix coefficients = mesh information × boundary fields
3. Variable field = internal field + boundary field (Bounary class in Geometric class)
4. Pointer List for boundaries
5. Class for each boundary type

Step 8. Object-oriented matrix based on BC

- source matrix for Dirichlet boundary condition
- coefficient matrix for Neumann boundary condition

$$\left(A\Big|_{ddt} - A\Big|_{laplacian}\right) \times \phi = \left(S\Big|_{ddt} - S\Big|_{laplacian}\right)$$

$$\left(A\Big|_{ddt} + ic\Big|_{ddt} - A\Big|_{laplacian} - ic\Big|_{laplacian}\right) \times \phi = \left(S\Big|_{ddt} + bc\Big|_{ddt} - S\Big|_{laplacian} - bc\Big|_{laplacian}\right)$$

$$A\Big|_{ddt} = \begin{bmatrix} \frac{V_1}{\Delta t} & 0 & 0 & 0 & 0 \\ 0 & \frac{V_2}{\Delta t} & 0 & 0 & 0 \\ 0 & 0 & \frac{V_3}{\Delta t} & 0 & 0 \\ & & & \vdots & \\ 0 & 0 & 0 & 0 & \frac{V_{M1}}{\Delta t} \end{bmatrix} \quad A\Big|_{laplacian} = \begin{bmatrix} -\alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & \frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ \frac{\alpha A_{f_2}}{\Delta x_2} & -\alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & \frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & \frac{\alpha A_{f_3}}{\Delta x_3} & -\alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & \frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & & \vdots & \\ 0 & 0 & 0 & \frac{\alpha A_{f_{M1}}}{\Delta x} & -\alpha\left(\frac{A_{f_{M1}}}{\Delta x_{M1}} + \frac{A_{f_{M1+1}}}{\Delta x_{M1+1}}\right) \end{bmatrix}$$

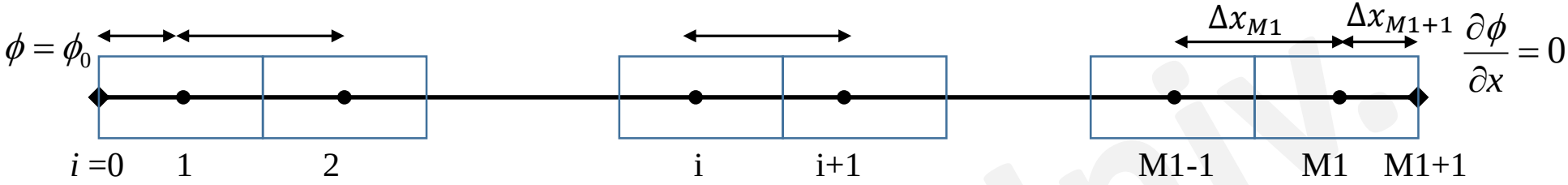
$$ic\Big|_{ddt} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ & & & \vdots & \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad ic\Big|_{laplacian} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ & & & \vdots & \\ 0 & 0 & 0 & 0 & \alpha\left(\frac{A_{f_{M1+1}}}{\Delta x_{M1+1}}\right) \end{bmatrix}$$

$$\left(A\Big|_{ddt} - A\Big|_{laplacian}\right) \times \phi = \left(S\Big|_{ddt} - S\Big|_{laplacian}\right)$$

$$\left(A\Big|_{ddt} + ic\Big|_{ddt} - A\Big|_{laplacian} - ic\Big|_{laplacian}\right) \times \phi = \left(S\Big|_{ddt} + bc\Big|_{ddt} - S\Big|_{laplacian} - bc\Big|_{laplacian}\right)$$

$$S\Big|_{ddt} = \begin{bmatrix} \frac{V_1}{\Delta t}\phi_1^{(n)} \\ \frac{V_2}{\Delta t}\phi_2^{(n)} \\ \frac{V_3}{\Delta t}\phi_3^{(n)} \\ \vdots \\ \frac{V_n}{\Delta t}\phi_{M1}^{(n)} \end{bmatrix} \qquad S\Big|_{laplacian} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$bc\Big|_{ddt} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad bc\Big|_{laplacian} = \begin{bmatrix} -\frac{\alpha A_{f_1}}{\Delta x_1}\phi_0^{(n+1)} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$



$$\begin{bmatrix} \frac{V_1}{\Delta t}+\alpha\left(\frac{A_{f_1}}{\Delta x_1}+\frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t}+\alpha\left(\frac{A_{f_2}}{\Delta x_2}+\frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t}+\alpha\left(\frac{A_{f_3}}{\Delta x_3}+\frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t}+\alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
   fvMesh mesh;
   GeometricField<double>
      phi(IOobject("phi",  ".", true, true) ,mesh);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      phi.bcond();
      fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
      phiEqn.solve(phi,nTime);
   }
   phi.write();
   return 0;
}
```

```
int main()
{
   fvMesh mesh;
   GeometricField<double>
      phi(IOobject("phi",  ".", true, true) ,mesh);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      phi.bcond();
      fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
      phiEqn.solve(phi,nTime);
   }
   phi.write();
   return 0;
}
```

```
class fvMatrix
{
private:
   List<List<double> > coef_;
   List<double> source_;
   List<double> internalCoef_;
   List<double> boundaryCoef_;
public:
   fvMatrix()
   {}
   fvMatrix(GeometricField<double> phi)
   {
      coef_.setSize(phi.size()-1);
      for(int i=0;i<coef_.size();i++) coef_[i].setSize(phi.size()-1);
      source_.setSize(phi.size()-1);
      boundaryCoef_.setSize(phi.size()-1);
      internalCoef_.setSize(phi.size()-1);
      for(int i=1;i<coef_.size();i++)
      {
         source_[i]=0.0;
         boundaryCoef_[i]=0.0;
         internalCoef_[i]=0.0;
         for(int j=1;j<coef_[i].size();j++) coef_[i][j]=0.0;
      }
   }
   fvMatrix(fvMatrix& rfvm)
   {
      coef_.setSize(rfvm.coef().size());
      for(int i=0;i<coef_.size();i++) coef_[i].setSize(rfvm.coef()[i].size());
      source_.setSize(rfvm.source().size());
      boundaryCoef_.setSize(rfvm.source().size());
      internalCoef_.setSize(rfvm.source().size());
      for(int i=1;i<coef_.size();i++)
      {
         source_[i]=rfvm.source()[i];
         boundaryCoef_[i]=rfvm.boundaryCoef()[i];
         internalCoef_[i]=rfvm.internalCoef()[i];
         for(int j=1;j<coef_[i].size();j++) coef_[i][j]=rfvm.coef()[i][j];
      }
   }
```

```
class fvMatrix
{
private:
   List<List<double> > coef_;
   List<double> source_;
public:
   fvMatrix()
   {}
   fvMatrix(GeometricField<double> phi)
   {
      coef_.setSize(phi.size()-1);
      for(int i=0;i<coef_.size();i++) coef_[i].setSize(phi.size()-1);
      source_.setSize(phi.size()-1);
      for(int i=1;i<coef_.size();i++)
      {
         source_[i]=0.0;
         for(int j=1;j<coef_[i].size();j++) coef_[i][j]=0.0;
      }
   }
   fvMatrix(fvMatrix& rfvm)
   {
      coef_.setSize(rfvm.coef().size());
      for(int i=0;i<coef_.size();i++) coef_[i].setSize(rfvm.coef()[i].size());
      source_.setSize(rfvm.source().size());
      for(int i=1;i<coef_.size();i++)
      {
         source_[i]=rfvm.source()[i];
         for(int j=1;j<coef_[i].size();j++) coef_[i][j]=rfvm.coef()[i][j];
      }
   }
```

in fvMatrix class

```
class fvMatrix
{
 void solve(GeometricField<double>& phi, const int& nTime)
  {
    int nIter=1;
    double residual;
    List<double> oldPhi(phi.size());
    do
    {
      for(int i=1;i<phi.size()-1;i++) oldPhi[i]=phi[i];
      for(int i=1;i<phi.size()-1;i++)
      {
        double sumLHS=0.0;
        for(int iLHS=1;iLHS<oldPhi.size()-1;iLHS++)
        {
          if(i==iLHS) sumLHS+=(coef_[iLHS][i]+internalCoef_[i])*oldPhi[iLHS];
          else sumLHS+=coef_[iLHS][i]*oldPhi[iLHS];
        }
        sumLHS-=(coef_[i][i]+internalCoef_[i])*oldPhi[i];
        phi[i]=(boundaryCoef_[i]+source_[i]-sumLHS)/(coef_[i][i]+internalCoef_[i]);
      }
      residual=0.0;
      for(int i=1;i<phi.size()-1;i++) residual+=fabs(phi[i]-oldPhi[i])/double(phi.size()-2);
      cout << nTime << " : RES = " << residual << ", ITER = " << nIter << endl;
      nIter++;
    } while(residual>0.0001);
  }
  List<double>& internalCoef()
  {
    return this->internalCoef_;
  }
  List<double>& boundaryCoef()
  {
    return this->boundaryCoef_;
  }
  friend fvMatrix& operator-(fvMatrix& rfvmA, fvMatrix& rfvmB)
  {
    for(int i=1;i<rfvmA.coef().size();i++)
    {
      rfvmA.source()[i]-=rfvmB.source()[i];
      rfvmA.boundaryCoef()[i]-=rfvmB.boundaryCoef()[i];
      rfvmA.internalCoef()[i]-=rfvmB.internalCoef()[i];
      for(int j=1;j<rfvmA.coef()[i].size();j++) rfvmA.coef()[i][j]-=rfvmB.coef()[i][j];
    }
    return rfvmA;
  }
};
```

```
class fvMatrix
{
  void solve(GeometricField<double>& phi, const int& nTime)
  {
    int nIter=1;
    double residual;
    List<double> oldPhi(phi.size());
    do
    {
      for(int i=1;i<phi.size()-1;i++) oldPhi[i]=phi[i];
      for(int i=1;i<phi.size()-1;i++)
      {
        double sumLHS=0.0;
        for(int iLHS=1;iLHS<oldPhi.size()-1;iLHS++)
          sumLHS+=coef_[iLHS][i]*oldPhi[iLHS];
        sumLHS-=coef_[i][i]*oldPhi[i];
        phi[i]=(source_[i]-sumLHS)/coef_[i][i];
      }
      residual=0.0;
      for(int i=1;i<phi.size()-1;i++) residual+=fabs(phi[i]-oldPhi[i])/double(phi.size()-2);
      cout << nTime << " : RES = " << residual << ", ITER = " << nIter << endl;
      nIter++;
    } while(residual>0.0001);
  }
  friend fvMatrix& operator-(fvMatrix& rfvmA, fvMatrix& rfvmB)
  {
    for(int i=1;i<rfvmA.coef().size();i++)
    {
      rfvmA.source()[i]-=rfvmB.source()[i];
      for(int j=1;j<rfvmA.coef()[i].size();j++) rfvmA.coef()[i][j]-=rfvmB.coef()[i][j];
    }
    return rfvmA;
  }
};
```

in fvMatrix class

```
namespace fvm
{
  fvMatrix& ddt(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    for(int i=1;i<rfvm.source().size();i++)
    {
      rfvm.source()[i]+=1.0/DT*phi[i]*phi.mesh().V()[i];
      rfvm.coef()[i][i]+=1.0*phi.mesh().V()[i]/DT;
    }
    return rfvm;
  }
  fvMatrix& laplacian(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    for(int i=1;i<rfvm.source().size();i++)
    {
      rfvm.source()[i]=0.0;
      rfvm.coef()[i][i]=
        -ALPHA*(phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1]+phi.mesh().Af()[i]/phi.mesh().dx()[i]);
      if(i!=1) rfvm.coef()[i-1][i]=ALPHA*phi.mesh().Af()[i]/phi.mesh().dx()[i];
      if(i!=rfvm.coef().size()-1) rfvm.coef()[i+1][i]=ALPHA*phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1];
    }
    rfvm.boundaryCoef()[1]=-ALPHA*phi.mesh().Af()[1]*phi[0]/phi.mesh().dx()[1];
    rfvm.internalCoef()[rfvm.coef().size()-1]=
      ALPHA*phi.mesh().Af()[rfvm.coef().size()]/phi.mesh().dx()[rfvm.coef().size()];
    return rfvm;
  }
};
```

```
namespace fvm
{
  fvMatrix& ddt(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    for(int i=1;i<rfvm.source().size();i++)
    {
      rfvm.source()[i]+=1.0/DT*phi[i]*phi.mesh().V()[i];
      rfvm.coef()[i][i]+=1.0*phi.mesh().V()[i]/DT;
    }
    return rfvm;
  }
  fvMatrix& laplacian(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    for(int i=1;i<rfvm.source().size();i++)
    {
      rfvm.source()[i]=0.0;
      rfvm.coef()[i][i]=
        -ALPHA*(phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1]+phi.mesh().Af()[i]/phi.mesh().dx()[i]);
      if(i!=1) rfvm.coef()[i-1][i]=ALPHA*phi.mesh().Af()[i]/phi.mesh().dx()[i];
      if(i!=rfvm.coef().size()-1) rfvm.coef()[i+1][i]=ALPHA*phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1];
    }
    rfvm.source()[1]+=-ALPHA*phi.mesh().Af()[1]*phi[0]/phi.mesh().dx()[1];
    rfvm.coef()[rfvm.coef().size()-1][rfvm.coef().size()-1]
      +=ALPHA*phi.mesh().Af()[rfvm.coef().size()]/phi.mesh().dx()[rfvm.coef().size()];
    return rfvm;
  }
};
```

in fvm namespace

- If we want to use several types for boundary?

Step 9. Object-oriented variables with boundary

- Boundary class inside GeometricField class

$$\left(A\big|_{ddt} + ic\big|_{ddt} - A\big|_{laplacian} - ic\big|_{laplacian}\right) \times \phi = \left(S\big|_{ddt} + bc\big|_{ddt} - S\big|_{laplacian} - bc\big|_{laplacian}\right)$$

$$\left(A\big|_{ddt} + ic\big|_{ddt} - A\big|_{laplacian} - ic\big|_{laplacian}\right) \times \phi = \left(S\big|_{ddt} - S\big|_{laplacian}\right) + \left(bc\big|_{ddt} - bc\big|_{laplacian}\right) \times \phi\big|_{boundary}^{(n+1)}$$

Boundary boundaryField_

$$bc\big|_{ddt} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad bc\big|_{laplacian} = \begin{pmatrix} -\frac{\alpha A_{f_1}}{\Delta x_1}\phi_0^{(n+1)} \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$bc\big|_{ddt} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad bc\big|_{laplacian} = \begin{pmatrix} -\frac{\alpha A_{f_1}}{\Delta x_1} \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad boundaryField\_ = \begin{pmatrix} \phi_0^{(n+1)} \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

- grid information
- determined by fvMesh class

- variable information
- determined by GeometricField class

```
class Boundary
  :
    public List<T>
  {
  private:
    List<string> fvPatchType_;
  public:
    Boundary()
    {
      this->setSize(2);
      fvPatchType_.setSize(2);
    }
    void readField(GeometricField& vf, const string& fieldDictEntry)
    {
      IOobject& obj = vf.obj();
      if(obj.readOption())
      {
        string firstToken;
        string fileName=obj.pathName()+"/"+obj.fileName();
        ifstream is(fileName.c_str());
        while(!is.eof())
        {
          is >> firstToken;
          if(firstToken==fieldDictEntry)
          {
            for(int iBoundary=0;iBoundary<this->size();iBoundary++)
            {
              is >> fvPatchType_[iBoundary];
              if(fvPatchType_[iBoundary]=="fixedValue") is >> this->operator[](iBoundary);
            }
          }
        }
        is.close();
      }
    }
    List<string>& fvPatchType()
    {
      return fvPatchType_;
    }
```

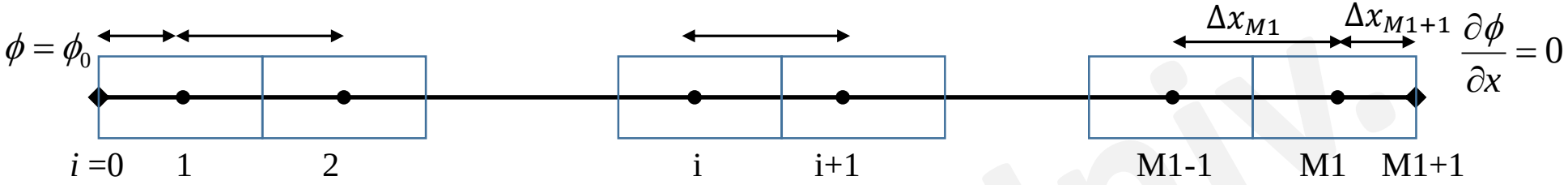
Boundary class

```
ostream& write(ostream& os)
    {
      os << "boundaryField" << endl;
      for(int i=0;i<this->size();i++)
      {
        if(fvPatchType_[i]=="zeroGradient") os << fvPatchType_[i] << endl;
        else if(fvPatchType_[i]=="fixedValue")
          os << fvPatchType_[i] << " " << this->operator[](i) << endl;
      }
      return os;
    }
};
```

structure of variable files (“phi”)

internalField uniform 0.0

internalField uniform 0.0
boundaryField
fixedValue -100
zeroGradient



$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
   fvMesh mesh;
   GeometricField<double>
      phi(IOobject("phi",  ".", true, true) ,mesh);    // w/w BC
   for(int nTime=0;nTime<NTST;nTime++)
   {
      fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
      phiEqn.solve(phi,nTime);
      phi.correctBoundaryConditions();     // updating BC
   }
   phi.write();
   return 0;
}
```

```
int main()
{
   fvMesh mesh;
   GeometricField<double>
      phi(IOobject("phi",  ".", true, true) ,mesh);    // w/o BC
   for(int nTime=0;nTime<NTST;nTime++)
   {
      phi.bcond();      // imposing BC
      fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
      phiEqn.solve(phi,nTime);
   }
   phi.write();
   return 0;
}
```

```
template <class T>
class GeometricField
:
    public List<T>
{
public:
    GeometricField(const IOobject& obj, fvMesh& mesh)
    :
        List<T>(mesh.nCells()+2),
        mesh_(mesh),
        obj_(obj)
    {
        readField("internalField");
        boundaryField_.readField(*this, "boundaryField");
    }
    IOobject& obj()
    {
        return obj_;
    }
    void write()
    {
        if(obj_.writeOption())
        {
            string fileName=obj_.pathName()+"/"+obj_.fileName();
            ofstream os(fileName.c_str());
            os << "internalField nonUniform " << endl;
            for(int i=1;i<this->size()-1;i++) os << this->operator[](i) << endl;
            boundaryField_.write(os);
            os.close();
        }
    }
    void correctBoundaryConditions()
    {
        if(boundaryField_.fvPatchType()[0]=="zeroGradient")
            boundaryField_[0]=this->operator[](1);
        if(boundaryField_.fvPatchType()[1]=="zeroGradient")
            boundaryField_[1]=this->operator[](this->size()-2);
    }
    Boundary& boundaryField()
    {
        return boundaryField_;
    }
private:
    fvMesh mesh_;
    IOobject obj_;
    Boundary boundaryField_;
};
```

```
template <class T>
class GeometricField
:
    public List<T>
{
private:
    fvMesh mesh_;
    IOobject obj_;
public:
    GeometricField(const IOobject& obj, fvMesh& mesh)
    :
        List<T>(mesh.nCells()+2),
        mesh_(mesh),
        obj_(obj)
    {
        readField("internalField");
    }
void write()
    {
        if(obj_.writeOption())
        {
            string fileName=obj_.pathName()+"/"+obj_.fileName();
            ofstream os(fileName.c_str());
            os << "internalField nonUniform " << endl;
            for(int i=1;i<this->size()-1;i++) os << this->operator[](i) << endl;
            os.close();
        }
    }
    void bcond()
    {
        this->operator[](0)=-100.0;
        //this->operator[](this->size()-1)=200.0;     // for Dirichlet
        this->operator[](this->size()-1)=this->operator[](this->size()-2);
    }
};
```

in GeometricField class

```
namespace fvm
{
  fvMatrix& ddt(GeometricField<double>& phi)
  {
    ……
  }
  fvMatrix& laplacian(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    const int& mSize=rfvm.source().size();
    for(int i=2;i<mSize-1;i++)
    {
      rfvm.source()[i]=0.0;
      rfvm.coef()[i][i]=
            -ALPHA*(phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1]+phi.mesh().Af()[i]/phi.mesh().dx()[i]);
      rfvm.coef()[i-1][i]=ALPHA*phi.mesh().Af()[i]/phi.mesh().dx()[i];
      rfvm.coef()[i+1][i]=ALPHA*phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1];
    }
    rfvm.source()[1]=0.0;
    rfvm.coef()[1][1]=
      -ALPHA*(phi.mesh().Af()[2]/phi.mesh().dx()[2]+phi.mesh().Af()[1]/phi.mesh().dx()[1]);
    rfvm.coef()[2][1]=ALPHA*phi.mesh().Af()[2]/phi.mesh().dx()[2];
    rfvm.boundaryCoef()[1]=-ALPHA*phi.mesh().Af()[1]/phi.mesh().dx()[1]*phi.boundaryField()[0];

    rfvm.source()[mSize-1]=0.0;
    rfvm.coef()[mSize-1][mSize-1]=-ALPHA*(phi.mesh().Af()[mSize]/phi.mesh().dx()[mSize]
                                          +phi.mesh().Af()[mSize-1]/phi.mesh().dx()[mSize-1]);
    rfvm.coef()[mSize-2][mSize-1]=ALPHA*phi.mesh().Af()[mSize-1]/phi.mesh().dx()[mSize-1];
    rfvm.internalCoef()[rfvm.coef().size()-1]=
        ALPHA*phi.mesh().Af()[rfvm.coef().size()]/phi.mesh().dx()[rfvm.coef().size()];
    return rfvm;
  }
};
```

internal cells

inlet cell

outlet cell

```
namespace fvm
{
  fvMatrix& ddt(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    for(int i=1;i<rfvm.source().size();i++)
    {
      rfvm.source()[i]+=1.0/DT*phi[i]*phi.mesh().V()[i];
      rfvm.coef()[i][i]+=1.0*phi.mesh().V()[i]/DT;
    }
    return rfvm;
  }
  fvMatrix& laplacian(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    for(int i=1;i<rfvm.source().size();i++)
    {
      rfvm.source()[i]=0.0;
      rfvm.coef()[i][i]=
        -ALPHA*(phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1]+phi.mesh().Af()[i]/phi.mesh().dx()[i]);
      if(i!=1) rfvm.coef()[i-1][i]=ALPHA*phi.mesh().Af()[i]/phi.mesh().dx()[i];
      if(i!=rfvm.coef().size()-1) rfvm.coef()[i+1][i]=ALPHA*phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1];
    }
    rfvm.boundaryCoef()[1]=-ALPHA*phi.mesh().Af()[1]*phi[0]/phi.mesh().dx()[1];
    rfvm.internalCoef()[rfvm.coef().size()-1]=
        ALPHA*phi.mesh().Af()[rfvm.coef().size()]/phi.mesh().dx()[rfvm.coef().size()];
    return rfvm;
  }
};
```

in fvm namespace

- If the number of boundaries is not fixed?
- If each boundary has several cells?

Step 10. Object-oriented variables with boundary

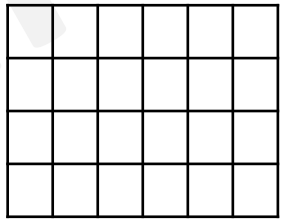
- Boundary class with PtrList class

- If the number of boundaries is not fixed?
- If each boundary has several cells?

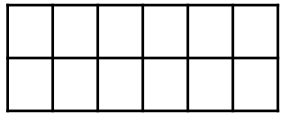
```
class GeometricField
:
   public List<T>
{
public:
   class Boundary
   :
      public List<T>
   {
      …
   }

private:
   Boundary boundaryField_;
}
```

```
class GeometricField
:
   public List<T>
{
public:
   class Boundary
   :
      public PtrList<List<T> >
   {
      …
   }

private:
   Boundary boundaryField_;
}
```

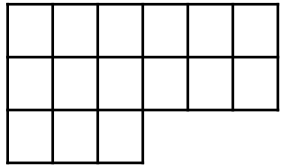
address 0 : 0x7f01



address 1 : 0x7fa2



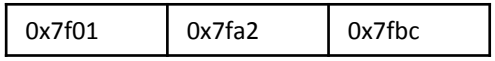
address 2 : 0x7fbc



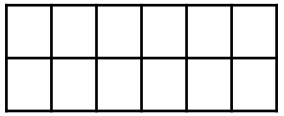

```
class GeometricField
:
   public List<T>
{
public:
   class Boundary
   {
   private:
      List<List<T> > bField_;
      …
   }
private:
   Boundary boundaryField_;
}
```

phi.boundaryField().bField().setSize(No. boundaries)
phi.boundaryField().bField()[i].setSize(No. faces on i-th boundary)

phi.boundaryField().setSize(No. boundaries)
phi.boundaryField()



phi.boundaryField()[1]



List<T> vs PtrList<List<T> >

```
class GeometricField
:
   public List<T>
{
public:
   class Boundary
   :
      public List<T>
   {
   public:
      Boundary()
      {
         this->setSize(2);
      }
      void readField(…)
      {
         …
         is >> this->operator[](0);
         is >> this->operator[](1);
      }
   };
   GeometricField(…)
   :
      List<T>(mesh.nCells()+2),
      mesh_(mesh),
      obj_(obj)
   {
      readField("internalField");
      boundaryField_.readField(*this, "boundaryField");
   }
.
.
private:
   fvMesh mesh_;
   IOobject obj_;
   Boundary boundaryField_;
}
```

default initialization of boundaryField_ by using Boundary() & List()

```
private:
   T* v_;
   int size_;
public:
   List()
   :
      size_(0),
      v_(NULL)
   {}
```

boundaryField_

T=double	T=double

```
class GeometricField
:
   public List<T>
{
public:
   class Boundary
   :
      public PtrList<List<T> >
   {
   public:
      Boundary()
      {
         this->setSize(2);
      }
      void readField(…)
      {
         …
         this->operator[](i=0,1).setSize(1);
         is >> this->operator[](i=0,1)[0];
      }
   };
   GeometricField(…)
   :
      List<T>(mesh.nCells()+2),
      mesh_(mesh),
      obj_(obj)
   {
      readField("internalField");
      boundaryField_.readField(*this, "boundaryField");
   }
.
.
private:
   fvMesh mesh_;
   IOobject obj_;
   Boundary boundaryField_;
}
```

→ Segmentation fault because List<T>* is not allocated in memory

default initialization of boundaryField_ by using Boundary() & PtrList()

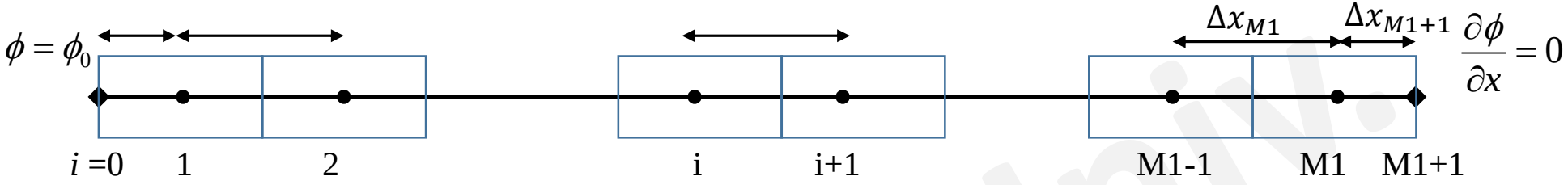
```
private:
   List<T*> ptrs_;
public:
   PtrList()
   :
      ptrs_()
   {}
```

boundaryField_

List<T>* = 0x00	List<T>* = 0x00

```
template <class T>
class PtrList
{
private:
   List<T*> ptrs_;
public:
   PtrList()
   :
      ptrs_()
   {}
   PtrList(int s)
   :
      ptrs_()
   {
      this->setSize(s);
   }

   void setSize(const int newSize)
   {
      int oldSize = size();
      if(newSize < oldSize) ptrs_.setSize(newSize);
      else if(newSize > oldSize)
      {
         ptrs_.setSize(newSize);
         for(int i=oldSize;i<newSize;i++) ptrs_[i]=NULL;
      }
   }
   T& operator[](int s)
   {
      return *(ptrs_[s]);
   }
   int size()
   {
      return ptrs_.size();
   }
   void set(const int i, T* ptr)
   {
      this->ptrs_[i] = ptr;
   }
};
```

PtrList class



$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
        phiEqn.solve(phi,nTime);
        phi.correctBoundaryConditions();
    }
    phi.write();
    return 0;
}
```

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
        phiEqn.solve(phi,nTime);
        phi.correctBoundaryConditions();
    }
    phi.write();
    return 0;
}
```

```
#include "PtrList.H"
template <class T>
class GeometricField
:
    public List<T>
{
public:
    class Boundary
    :
        public PtrList<List<T> >
    {
    private:
        List<string> fvPatchType_;
    public:
        Boundary()
        {
            this->setSize(2);
            fvPatchType_.setSize(2);
        }
        void readField(GeometricField& vf, const string& fieldDictEntry)
        {
            IOobject& obj = vf.obj();
            if(obj.readOption())
            {
                string firstToken;
                string fileName=obj.pathName()+"/"+obj.fileName();
                ifstream is(fileName.c_str());
                while(!is.eof())
                {
                    is >> firstToken;
                    if(firstToken==fieldDictEntry)
                    {
                        for(int iBoundary=0;iBoundary<this->size();iBoundary++)
                        {
                            List<T>* pList = new List<T>(1);
                            this->set(iBoundary,pList);
                            is >> fvPatchType_[iBoundary];
                            if(fvPatchType_[iBoundary]=="fixedValue") is >> this->operator[](iBoundary)[0];
                        }
                    }
                }
                is.close();
            }
        }
```

```
template <class T>
class GeometricField
:
    public List<T>
{
public:
    class Boundary
    :
        public List<T>
    {
    private:
        List<string> fvPatchType_;
    public:
        Boundary()
        {
            this->setSize(2);
            fvPatchType_.setSize(2);
        }
        void readField(GeometricField& vf, const string& fieldDictEntry)
        {
            IOobject& obj = vf.obj();
            if(obj.readOption())
            {
                string firstToken;
                string fileName=obj.pathName()+"/"+obj.fileName();
                ifstream is(fileName.c_str());
                while(!is.eof())
                {
                    is >> firstToken;
                    if(firstToken==fieldDictEntry)
                    {
                        for(int iBoundary=0;iBoundary<this->size();iBoundary++)
                        {
                            is >> fvPatchType_[iBoundary];
                            if(fvPatchType_[iBoundary]=="fixedValue") is >> this->operator[](iBoundary);
                        }
                    }
                }
                is.close();
            }
        }
```

in Boundary class inside GeometricField class

```
#include "List.H"
#include "PtrList.H"
template <class T>
class GeometricField
:
    public List<T>
{
public:
    class Boundary
    :
        public PtrList<List<T> >
    {
    private:
        List<string> fvPatchType_;
    public:
        List<string>& fvPatchType()
        { … }
        ostream& write(ostream& os)
        {
            os << "boundaryField" << endl;
            for(int i=0;i<this->size();i++)
            {
                if(fvPatchType_[i]=="zeroGradient") os << fvPatchType_[i] << endl;
                else if(fvPatchType_[i]=="fixedValue")
                    os << fvPatchType_[i] << " " << this->operator[](i)[0] << endl;
            }
            return os;
        }
    };
    GeometricField(const IOobject& obj, fvMesh& mesh)
    :
        List<T>(mesh.nCells()+2),
        mesh_(mesh),
        obj_(obj)
    { … }
    void correctBoundaryConditions()
    {
        if(boundaryField_.fvPatchType()[0]=="zeroGradient")
            boundaryField_[0][0]=this->operator[](1);
        if(boundaryField_.fvPatchType()[1]=="zeroGradient")
            boundaryField_[1][0]=this->operator[](this->size()-2);
    }
private:
    fvMesh mesh_;
    IOobject obj_;
    Boundary boundaryField_;
};
```

```
#include "List.H"
template <class T>
class GeometricField
:
    public List<T>
{
public:
    class Boundary
    :
        public List<T>
    {
    private:
        List<string> fvPatchType_;
    public:
        List<string>& fvPatchType()
        { … }
        ostream& write(ostream& os)
        {
            os << "boundaryField" << endl;
            for(int i=0;i<this->size();i++)
            {
                if(fvPatchType_[i]=="zeroGradient") os << fvPatchType_[i] << endl;
                else if(fvPatchType_[i]=="fixedValue")
                    os << fvPatchType_[i] << " " << this->operator[](i) << endl;
            }
            return os;
        }
    };
    GeometricField(const IOobject& obj, fvMesh& mesh)
    :
        List<T>(mesh.nCells()+2),
        mesh_(mesh),
        obj_(obj)
    { … }
    void correctBoundaryConditions()
    {
        if(boundaryField_.fvPatchType()[0]=="zeroGradient")
            boundaryField_[0]=this->operator[](1);
        if(boundaryField_.fvPatchType()[1]=="zeroGradient")
            boundaryField_[1]=this->operator[](this->size()-2);
    }
private:
    fvMesh mesh_;
    IOobject obj_;
    Boundary boundaryField_;
};
```

in Boundary class inside GeometricField class

```
namespace fvm
{
  fvMatrix& ddt(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    for(int i=1;i<rfvm.source().size();i++)
    {
      rfvm.source()[i]+=1.0/DT*phi[i]*phi.mesh().V()[i];
      rfvm.coef()[i][i]+=1.0*phi.mesh().V()[i]/DT;
    }
    return rfvm;
  }
  fvMatrix& laplacian(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    const int& mSize=rfvm.source().size();
    for(int i=2;i<mSize-1;i++)
    {
      rfvm.source()[i]=0.0;
      rfvm.coef()[i][i]=
        -ALPHA*(phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1]+phi.mesh().Af()[i]/phi.mesh().dx()[i]);
      rfvm.coef()[i-1][i]=ALPHA*phi.mesh().Af()[i]/phi.mesh().dx()[i];
      rfvm.coef()[i+1][i]=ALPHA*phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1];
    }
    rfvm.source()[1]=0.0;
    rfvm.coef()[1][1]=
      -ALPHA*(phi.mesh().Af()[2]/phi.mesh().dx()[2]+phi.mesh().Af()[1]/phi.mesh().dx()[1]);
    rfvm.coef()[2][1]=ALPHA*phi.mesh().Af()[2]/phi.mesh().dx()[2];
    rfvm.boundaryCoef()[1]=
      -ALPHA*phi.mesh().Af()[1]/phi.mesh().dx()[1]*phi.boundaryField()[0][0];

    rfvm.source()[mSize-1]=0.0;
    rfvm.coef()[mSize-1][mSize-1]=-ALPHA*(phi.mesh().Af()[mSize]/phi.mesh().dx()[mSize]
                                          +phi.mesh().Af()[mSize-1]/phi.mesh().dx()[mSize-1]);
    rfvm.coef()[mSize-2][mSize-1]=ALPHA*phi.mesh().Af()[mSize-1]/phi.mesh().dx()[mSize-1];
    rfvm.internalCoef()[rfvm.coef().size()-1]=
      ALPHA*phi.mesh().Af()[rfvm.coef().size()]/phi.mesh().dx()[rfvm.coef().size()];
    return rfvm;
  }
};
```

```
namespace fvm
{
  fvMatrix& ddt(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    for(int i=1;i<rfvm.source().size();i++)
    {
      rfvm.source()[i]+=1.0/DT*phi[i]*phi.mesh().V()[i];
      rfvm.coef()[i][i]+=1.0*phi.mesh().V()[i]/DT;
    }
    return rfvm;
  }
  fvMatrix& laplacian(GeometricField<double>& phi)
  {
    tmp<fvMatrix> tFvMatrix = new fvMatrix(phi);
    fvMatrix& rfvm = tFvMatrix.ref();
    const int& mSize=rfvm.source().size();
    for(int i=2;i<mSize-1;i++)
    {
      rfvm.source()[i]=0.0;
      rfvm.coef()[i][i]=
        -ALPHA*(phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1]+phi.mesh().Af()[i]/phi.mesh().dx()[i]);
      rfvm.coef()[i-1][i]=ALPHA*phi.mesh().Af()[i]/phi.mesh().dx()[i];
      rfvm.coef()[i+1][i]=ALPHA*phi.mesh().Af()[i+1]/phi.mesh().dx()[i+1];
    }
    rfvm.source()[1]=0.0;
    rfvm.coef()[1][1]=
      -ALPHA*(phi.mesh().Af()[2]/phi.mesh().dx()[2]+phi.mesh().Af()[1]/phi.mesh().dx()[1]);
    rfvm.coef()[2][1]=ALPHA*phi.mesh().Af()[2]/phi.mesh().dx()[2];
    rfvm.boundaryCoef()[1]=-ALPHA*phi.mesh().Af()[1]/phi.mesh().dx()[1]*phi.boundaryField()[0];

    rfvm.source()[mSize-1]=0.0;
    rfvm.coef()[mSize-1][mSize-1]=-ALPHA*(phi.mesh().Af()[mSize]/phi.mesh().dx()[mSize]
                                          +phi.mesh().Af()[mSize-1]/phi.mesh().dx()[mSize-1]);
    rfvm.coef()[mSize-2][mSize-1]=ALPHA*phi.mesh().Af()[mSize-1]/phi.mesh().dx()[mSize-1];
    rfvm.internalCoef()[rfvm.coef().size()-1]=
      ALPHA*phi.mesh().Af()[rfvm.coef().size()]/phi.mesh().dx()[rfvm.coef().size()];
    return rfvm;
  }
};
```

in fvm name space

Step 11. Object-oriented variables with boundary

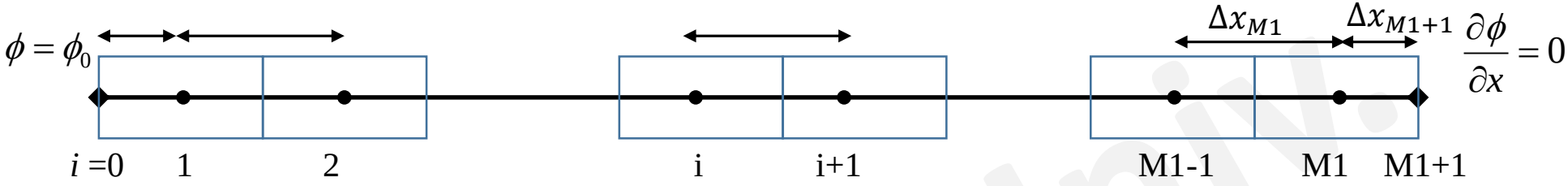
- fixedValueFvPatchField & zeroGradientFvPatchField class

```
template <class T>
class fixedValueFvPatchField
:
   public List<T>
{
public:
   fixedValueFvPatchField(const List<T>& field, const int& nPatch)
   {
      this->setSize(nPatch);
   }
};
```

fixedValueFvPatchField class

```
template <class T>
class zeroGradientFvPatchField
:
   public List<T>
{
public:
   zeroGradientFvPatchField(const List<T>& field, const
int& nPatch)
   {
      this->setSize(nPatch);
   }
};
```

zeroGradientFvPatchField class



$$\begin{bmatrix} \frac{V_1}{\Delta t}+\alpha\left(\frac{A_{f_1}}{\Delta x_1}+\frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t}+\alpha\left(\frac{A_{f_2}}{\Delta x_2}+\frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t}+\alpha\left(\frac{A_{f_3}}{\Delta x_3}+\frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t}+\alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
   fvMesh mesh;
   GeometricField<double>
      phi(IOobject("phi",  ".", true, true) ,mesh);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
      phiEqn.solve(phi,nTime);
      phi.correctBoundaryConditions();
   }
   phi.write();
   return 0;
}
```

```
int main()
{
   fvMesh mesh;
   GeometricField<double>
      phi(IOobject("phi",  ".", true, true) ,mesh);
   for(int nTime=0;nTime<NTST;nTime++)
   {
      fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
      phiEqn.solve(phi,nTime);
      phi.correctBoundaryConditions();
   }
   phi.write();
   return 0;
}
```

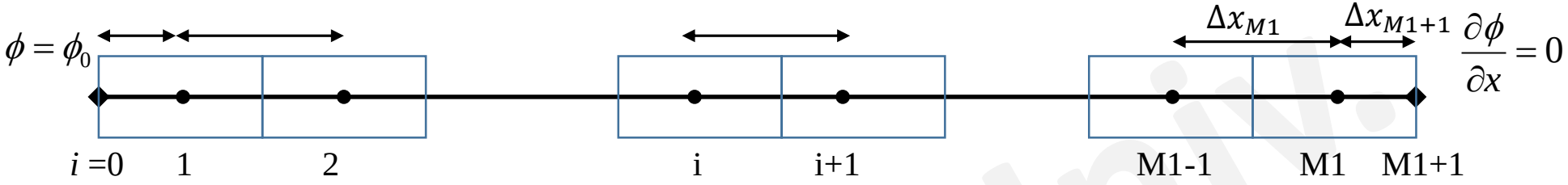
```
#include “fixedValueFvPatchFieldFor11st.H"
#include "zeroGradientFvPatchFieldFor11st.H"
template <class T>
class GeometricField
:
   public List<T>
{
public:
   class Boundary
   :
      public PtrList<List<T> >
   {
   private:
      List<string> fvPatchType_;
   public:
      Boundary()
      {
         this->setSize(2);
         fvPatchType_.setSize(2);
      }
      void readField(GeometricField& vf, const string& fieldDictEntry)
      {
         IOobject& obj = vf.obj();
         if(obj.readOption())
         {
            string firstToken;
            string fileName=obj.pathName()+"/"+obj.fileName();
            ifstream is(fileName.c_str());
            while(!is.eof())
            {
               is >> firstToken;
               if(firstToken==fieldDictEntry)
               {
                  for(int iBoundary=0;iBoundary<this->size();iBoundary++)
                  {
                     is >> fvPatchType_[iBoundary];
                     if(fvPatchType_[iBoundary]=="fixedValue")
                     {
                        fixedValueFvPatchField<T>* pList = new fixedValueFvPatchField<T>(vf,1);
                        this->set(iBoundary,pList);
                        is >> this->operator[](iBoundary)[0];
                     }
                     else if(fvPatchType_[iBoundary]=="zeroGradient")
                     {
                        zeroGradientFvPatchField<T>* pList = new zeroGradientFvPatchField<T>(vf,1);
                        this->set(iBoundary,pList);
                     }
                  }
               }
            }
            is.close();
         }
      }
```

```
template <class T>
class GeometricField
:
   public List<T>
{
public:
   class Boundary
   :
      public PtrList<List<T> >
   {
   private:
      List<string> fvPatchType_;
   public:
      Boundary()
      {
         this->setSize(2);
         fvPatchType_.setSize(2);
      }
      void readField(GeometricField& vf, const string& fieldDictEntry)
      {
         IOobject& obj = vf.obj();
         if(obj.readOption())
         {
            string firstToken;
            string fileName=obj.pathName()+"/"+obj.fileName();
            ifstream is(fileName.c_str());
            while(!is.eof())
            {
               is >> firstToken;
               if(firstToken==fieldDictEntry)
               {
                  for(int iBoundary=0;iBoundary<this->size();iBoundary++)
                  {
                     List<T>* pList = new List<T>(1);
                     this->set(iBoundary,pList);
                     is >> fvPatchType_[iBoundary];
                     if(fvPatchType_[iBoundary]=="fixedValue") is >> this->operator[](iBoundary)[0];
                  }
               }
            }
            is.close();
         }
      }
```

in Boundary class inside GeometricField class

Step 12. Object-oriented variables with boundary

- improving fixedValueFvPatchField & zeroGradientFvPatchField class



$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
        phiEqn.solve(phi,nTime);
        phi.correctBoundaryConditions();
    }
    phi.write();
    return 0;
}
```

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
        phiEqn.solve(phi,nTime);
        phi.correctBoundaryConditions();
    }
    phi.write();
    return 0;
}
```

```
template <class T>
class fixedValueFvPatchField
:
   public List<T>
{
private:
   T fixedValue_;
public:
   fixedValueFvPatchField(const List<T>& field, const int& nPatch, istream& is)
   {
      this->setSize(nPatch);
      is >> fixedValue_;
      for(int i=0;i<this->size();i++) this->operator[](i)=fixedValue_;
   }
};
```

fixedValueFvPatchField class

```
template <class T>
class fixedValueFvPatchField
:
   public List<T>
{
public:
   fixedValueFvPatchField(const List<T>& field, const int& nPatch)
   {
      this->setSize(nPatch);
   }
};
```

fixedValueFvPatchField class

```
template <class T>
class zeroGradientFvPatchField
:
   public List<T>
{
public:
   zeroGradientFvPatchField(const List<T>& field, const int& nPatch, istream& is)
   {
      this->setSize(nPatch);
   }
};
```

zeroGradientFvPatchField class

```
template <class T>
class zeroGradientFvPatchField
:
   public List<T>
{
public:
   zeroGradientFvPatchField(const List<T>& field, const int& nPatch)
   {
      this->setSize(nPatch);
   }
};
```

zeroGradientFvPatchField class

✓ dictionary in OpenFOAM instead of istream

```
template <class T>
class GeometricField
:
   public List<T>
{
public:
   class Boundary
   :
      public PtrList<List<T> >
   {
   private:
      List<string> fvPatchType_;
   public:
      Boundary()
      {
         this->setSize(2);
         fvPatchType_.setSize(2);
      }
      void readField(GeometricField& vf, const string& fieldDictEntry)
      {
         IOobject& obj = vf.obj();
         if(obj.readOption())
         {
            string firstToken;
            string fileName=obj.pathName()+"/"+obj.fileName();
            ifstream is(fileName.c_str());
            while(!is.eof())
            {
               is >> firstToken;
               if(firstToken==fieldDictEntry)
               {
                  for(int iBoundary=0;iBoundary<this->size();iBoundary++)
                  {
                     is >> fvPatchType_[iBoundary];
                     if(fvPatchType_[iBoundary]=="fixedValue")
                     {
                        fixedValueFvPatchField<T>* pList = new fixedValueFvPatchField<T>(vf,1,is);
                        this->set(iBoundary,pList);
                     }
                     else if(fvPatchType_[iBoundary]=="zeroGradient")
                     {
                        zeroGradientFvPatchField<T>* pList = new zeroGradientFvPatchField<T>(vf,1,is);
                        this->set(iBoundary,pList);
                     }
                  }
               }
               is.close();
            }
      }
```

- calling all fvPatchField classes in the same format
- using member function for a specific activity, if necessary

```
template <class T>
class GeometricField
:
   public List<T>
{
public:
   class Boundary
   :
      public PtrList<List<T> >
   {
   private:
      List<string> fvPatchType_;
   public:
      Boundary()
      {
         this->setSize(2);
         fvPatchType_.setSize(2);
      }
      void readField(GeometricField& vf, const string& fieldDictEntry)
      {
         IOobject& obj = vf.obj();
         if(obj.readOption())
         {
            string firstToken;
            string fileName=obj.pathName()+"/"+obj.fileName();
            ifstream is(fileName.c_str());
            while(!is.eof())
            {
               is >> firstToken;
               if(firstToken==fieldDictEntry)
               {
                  for(int iBoundary=0;iBoundary<this->size();iBoundary++)
                  {
                     is >> fvPatchType_[iBoundary];
                     if(fvPatchType_[iBoundary]=="fixedValue")
                     {
                        fixedValueFvPatchField<T>* pList = new fixedValueFvPatchField<T>(vf,1);
                        this->set(iBoundary,pList);
                        is >> this->operator[](iBoundary)[0];
                     }
                     else if(fvPatchType_[iBoundary]=="zeroGradient")
                     {
                        zeroGradientFvPatchField<T>* pList = new zeroGradientFvPatchField<T>(vf,1);
                        this->set(iBoundary,pList);
                     }
                  }
               }
               is.close();
            }
      }
```

in Boundary class inside GeometricField class

```
template <class T>
class GeometricField
:
    public List<T>
{
public:
    class Boundary
    :
        public PtrList<List<T> >
    {
    private:
        List<string> fvPatchType_;
    public:
        Boundary()
        {
            this->setSize(2);
            fvPatchType_.setSize(2);
        }
        void readField(GeometricField& vf, const string& fieldDictEntry)
        {
            IOobject& obj = vf.obj();
            if(obj.readOption())
            {
                string firstToken;
                string fileName=obj.pathName()+"/"+obj.fileName();
                ifstream is(fileName.c_str());
                while(!is.eof())
                {
                    is >> firstToken;
                    if(firstToken==fieldDictEntry)
                    {
                        for(int iBoundary=0;iBoundary<this->size();iBoundary++)
                        {
                            is >> fvPatchType_[iBoundary];
                            if(fvPatchType_[iBoundary]=="fixedValue")
                                this->set(iBoundary, new fixedValueFvPatchField<T>(vf,1,is));
                            else if(fvPatchType_[iBoundary]=="zeroGradient")
                                this->set(iBoundary, new zeroGradientFvPatchField<T>(vf,1,is));
                        }
                    }
                    is.close();
                }
            }
```

- more simplified format

```
template <class T>
class GeometricField
:
    public List<T>
{
public:
    class Boundary
    :
        public PtrList<List<T> >
    {
    private:
        List<string> fvPatchType_;
    public:
        Boundary()
        {
            this->setSize(2);
            fvPatchType_.setSize(2);
        }
        void readField(GeometricField& vf, const string& fieldDictEntry)
        {
            IOobject& obj = vf.obj();
            if(obj.readOption())
            {
                string firstToken;
                string fileName=obj.pathName()+"/"+obj.fileName();
                ifstream is(fileName.c_str());
                while(!is.eof())
                {
                    is >> firstToken;
                    if(firstToken==fieldDictEntry)
                    {
                        for(int iBoundary=0;iBoundary<this->size();iBoundary++)
                        {
                            is >> fvPatchType_[iBoundary];
                            if(fvPatchType_[iBoundary]=="fixedValue")
                            {
                                fixedValueFvPatchField<T>* pList = new fixedValueFvPatchField<T>(vf,1,is);
                                this->set(iBoundary,pList);
                            }
                            else if(fvPatchType_[iBoundary]=="zeroGradient")
                            {
                                zeroGradientFvPatchField<T>* pList = new zeroGradientFvPatchField<T>(vf,1,is);
                                this->set(iBoundary,pList);
                            }
                        }
                    }
                    is.close();
                }
            }
```

in Boundary class inside GeometricField class

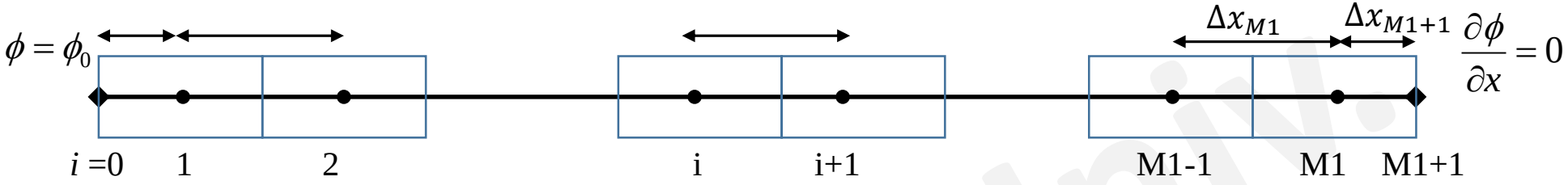
Step 13. Object-oriented variables with boundary

- fvPatchField class for zeroGradientFvPatchField & fixedValueFvPatchField class

```
template <class T>
class fvPatchField
:
   public List<T>
{
public:
   fvPatchField(const int& nPatch)
   {
      this->setSize(nPatch);
   }

   static fvPatchField<T>* New(const List<T>& field, const int& nPatch, istream& is, const string& patchFieldType)
   {
      if(patchFieldType==fixedValueFvPatchField<T>::typeName())
      {
         fvPatchField<T>* pList = new fixedValueFvPatchField<T>(field,1,is);
         return pList;
      }
      else if(patchFieldType==zeroGradientFvPatchField<T>::typeName())
      {
         fvPatchField<T>* pList = new zeroGradientFvPatchField<T>(field,1,is);
         return pList;
      }
   }
};
```

role of selector

fvPatchField class



$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
        phiEqn.solve(phi,nTime);
        phi.correctBoundaryConditions();
    }
    phi.write();
    return 0;
}
```

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
        phiEqn.solve(phi,nTime);
        phi.correctBoundaryConditions();
    }
    phi.write();
    return 0;
}
```

```
template <class T>
class fvPatchField;

template <class T>
class fixedValueFvPatchField
:
    public fvPatchField<T>
{
private:
    T fixedValue_;
public:
    static const string typeName()
    {
        return "fixedValue";
    }
    fixedValueFvPatchField(const List<T>& field, const int& nPatch, istream& is)
    :
        fvPatchField<T>(nPatch)
    {
        is >> fixedValue_;
        for(int i=0;i<this->size();i++) this->operator[](i)=fixedValue_;
    }
};
```

fixedValueFvPatchField class

```
template <class T>
class fixedValueFvPatchField
:
    public List<T>
{
private:
    T fixedValue_;
public:
    fixedValueFvPatchField(const List<T>& field, const int& nPatch, istream& is)
    {
        this->setSize(nPatch);
        is >> fixedValue_;
        for(int i=0;i<this->size();i++) this->operator[](i)=fixedValue_;
    }
};
```

```
template <class T>
class fvPatchField;

template <class T>
class zeroGradientFvPatchField
:
    public fvPatchField<T>
{
public:
    static const string typeName()
    {
        return "zeroGradient";
    }
    zeroGradientFvPatchField(const List<T>& field, const int& nPatch, istream& is)
    :
        fvPatchField<T>(nPatch)
    {}
};
```

```
template <class T>
class zeroGradientFvPatchField
:
    public List<T>
{
public:
    zeroGradientFvPatchField(const List<T>& field, const int& nPatch, istream& is)
    {
        this->setSize(nPatch);
    }
};
```


zeroGradientFvPatchField class

```
#include "List.H"
#include "PtrList.H"
#include "fixedValueFvPatchFieldFor13rd.H"
#include "zeroGradientFvPatchFieldFor13rd.H"
#include "fvPatchFieldFor13rd.H"

template <class T>
class GeometricField
:
   public List<T>
{
public:
   class Boundary
   :
      public PtrList<List<T> >
   {
   private:
      List<string> fvPatchType_;
   public:
      Boundary()
      {
         this->setSize(2);
         fvPatchType_.setSize(2);
      }
      void readField(GeometricField& vf, const string& fieldDictEntry)
      {
         IOobject& obj = vf.obj();
         if(obj.readOption())
         {
            string firstToken;
            string fileName=obj.pathName()+"/"+obj.fileName();
            ifstream is(fileName.c_str());
            while(!is.eof())
            {
               is >> firstToken;
               if(firstToken==fieldDictEntry)
               {
                  for(int iBoundary=0;iBoundary<this->size();iBoundary++)
                  {
                     is >> fvPatchType_[iBoundary];
                     this->set(iBoundary,fvPatchField<T>::New(vf,1,is,fvPatchType_[iBoundary]));
                  }
               }
            }
            is.close();
         }
      }
```

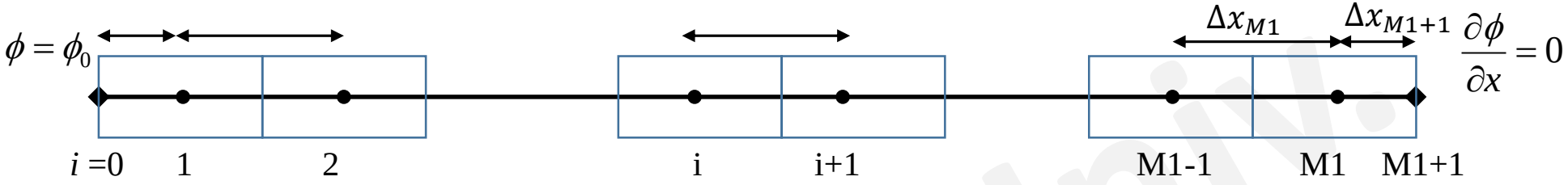
object-oriented selection of boundary condition,
: selector New(…)

```
template <class T>
class GeometricField
:
   public List<T>
{
public:
   class Boundary
   :
      public PtrList<List<T> >
   {
   private:
      List<string> fvPatchType_;
   public:
      Boundary()
      {
         this->setSize(2);
         fvPatchType_.setSize(2);
      }
      void readField(GeometricField& vf, const string& fieldDictEntry)
      {
         IOobject& obj = vf.obj();
         if(obj.readOption())
         {
            string firstToken;
            string fileName=obj.pathName()+"/"+obj.fileName();
            ifstream is(fileName.c_str());
            while(!is.eof())
            {
               is >> firstToken;
               if(firstToken==fieldDictEntry)
               {
                  for(int iBoundary=0;iBoundary<this->size();iBoundary++)
                  {
                     is >> fvPatchType_[iBoundary];
                     if(fvPatchType_[iBoundary]=="fixedValue")
                        this->set(iBoundary, new fixedValueFvPatchField<T>(vf,1,is));
                     else if(fvPatchType_[iBoundary]=="zeroGradient")
                        this->set(iBoundary, new zeroGradientFvPatchField<T>(vf,1,is));
                  }
               }
               is.close();
            }
      }
```

in Boundary class inside GeometricField class

Step 14. Object-oriented variables with boundary

- evaluate() & updateCoeffs() in each fvPatchField class
- virtual function



$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
        phiEqn.solve(phi,nTime);
        phi.correctBoundaryConditions();
    }
    phi.write();
    return 0;
}
```

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
        phiEqn.solve(phi,nTime);
        phi.correctBoundaryConditions();
    }
    phi.write();
    return 0;
}
```

```
#include "List.H"
#include "PtrList.H"
#include "fixedValueFvPatchFieldFor14th.H"
#include "zeroGradientFvPatchFieldFor14th.H"
#include "fvPatchFieldFor14th.H"

template <class T>
class GeometricField
:
    public List<T>
{
public:
    class Boundary
    :
        public PtrList<fvPatchField<T> >
    {
    private:
        List<string> fvPatchType_;
    public:
        void evaluate(const double& value)
        {
            for(int i=0;i<this->size();i++) this->operator[](i).evaluate(value);
        }
    };

 GeometricField(const IOobject& obj, fvMesh& mesh)
    :
        List<T>(mesh.nCells()+2),
        mesh_(mesh),
        obj_(obj)
    {
        readField("internalField");
        boundaryField_.readField(*this, "boundaryField");
    }
    void correctBoundaryConditions()
    {
        // internalCell has not been defined yet, so this is only valid for zeroGradient outlet
        boundaryField_.evaluate(this->operator[](this->size()-2));
    }
```

```
#include "List.H"
#include "PtrList.H"
#include "fixedValueFvPatchFieldFor13rd.H"
#include "zeroGradientFvPatchFieldFor13rd.H"
#include "fvPatchFieldFor13rd.H"

template <class T>
class GeometricField
:
    public List<T>
{
public:
    class Boundary
    :
        public PtrList<List<T> >
    {
    private:
        List<string> fvPatchType_;
    public:
        ...
    };

 GeometricField(const IOobject& obj, fvMesh& mesh)
    :
        List<T>(mesh.nCells()+2),
        mesh_(mesh),
        obj_(obj)
    {
        readField("internalField");
        boundaryField_.readField(*this, "boundaryField");
    }
    void correctBoundaryConditions()
    {
        if(boundaryField_.fvPatchType()[0]=="zeroGradient")
            boundaryField_[0][0]=this->operator[](1);
        if(boundaryField_.fvPatchType()[1]=="zeroGradient")
            boundaryField_[1][0]=this->operator[](this->size()-2);
    }
```


Boundary class & GeometricField class

```
template <class T>
class fvPatchField
:
   public List<T>
{
private:
   bool updated_;
public:
   fvPatchField(const int& nPatch)
   :
      updated_(false)
   {
      this->setSize(nPatch);
   }

   static fvPatchField<T>* New(const List<T>& field, const int& nPatch, istream& is,
const string& patchFieldType)
   {
      if(patchFieldType==fixedValueFvPatchField<T>::typeName())
      {
         fvPatchField<T>* pList = new fixedValueFvPatchField<T>(field,1,is);
         return pList;
      }
      else if(patchFieldType==zeroGradientFvPatchField<T>::typeName())
      {
         fvPatchField<T>* pList = new zeroGradientFvPatchField<T>(field,1,is);
         return pList;
      }
   }
   void evaluate(const double& value)
   {
      if(!updated_) updateCoeffs(value);
      updated_ = false;
   }
   virtual void updateCoeffs(const double& value)
   {
      updated_ = false;
   }
};
```

```
template <class T>
class fvPatchField
:
   public List<T>
{
public:
   fvPatchField(const int& nPatch)
   {
      this->setSize(nPatch);
   }

   static fvPatchField<T>* New(const List<T>& field, const int& nPatch, istream&
is, const string& patchFieldType)
   {
      if(patchFieldType==fixedValueFvPatchField<T>::typeName())
      {
         fvPatchField<T>* pList = new fixedValueFvPatchField<T>(field,1,is);
         return pList;
      }
      else if(patchFieldType==zeroGradientFvPatchField<T>::typeName())
      {
         fvPatchField<T>* pList = new zeroGradientFvPatchField<T>(field,1,is);
         return pList;
      }
   }
};
```

fvPatchField class

```
template <class T>
class fvPatchField;

template <class T>
class fixedValueFvPatchField
:
    public fvPatchField<T>
{
private:
    T fixedValue_;
public:
    static const string typeName()
    {
        return "fixedValue";
    }
    fixedValueFvPatchField(const List<T>& field, const int& nPatch, istream& is)
    :
        fvPatchField<T>(nPatch)
    {
        is >> fixedValue_;
        for(int i=0;i<this->size();i++) this->operator[](i)=fixedValue_;
    }
    virtual void updateCoeffs(const double& value)
    {
        for(int i=0;i<this->size();i++) this->operator[](i)=fixedValue_;
    }
};
```

fixedValueFvPatchField class


```
template <class T>
class fvPatchField;

template <class T>
class fixedValueFvPatchField
:
    public fvPatchField<T>
{
private:
    T fixedValue_;
public:
    static const string typeName()
    {
        return "fixedValue";
    }
    fixedValueFvPatchField(const List<T>& field, const int& nPatch, istream& is)
    :
        fvPatchField<T>(nPatch)
    {
        is >> fixedValue_;
        for(int i=0;i<this->size();i++) this->operator[](i)=fixedValue_;
    }
};
```

```
template <class T>
class fvPatchField;

template <class T>
class zeroGradientFvPatchField
:
    public fvPatchField<T>
{
public:
    static const string typeName()
    {
        return "zeroGradient";
    }
    zeroGradientFvPatchField(const List<T>& field, const int& nPatch, istream& is)
    :
        fvPatchField<T>(nPatch)
    {}
    virtual void updateCoeffs(const double& value)
    {
        for(int i=0;i<this->size();i++) this->operator[](i)=value;
    }
};
```

zeroGradientFvPatchField class

Step 15. Mechanism of Run Time Selection

- pointer to function
- register & hashTable

Lv 1: pointer to function

```
#include <iostream>
using namespace std;
void RTS(int iRTS)
{
   cout << "This is RTS function\n";
}
int main()
{
   int temp=5;
   void (*RTSfunc)(int);
   RTSfunc=&RTS;
   (*RTSfunc)(temp);
   return 0;
}
```

>> This is RTS function

Lv 2: pointer to member function of a class

```
#include <iostream>
using namespace std;
class RTS
{
public:
   static int New(int nT)
   {
      cout << "This is New function in RTS class\n";
      return nT;
   }
   RTS(int nT)
   {
      cout << "This is RTS class\n";
   }
};
int main()
{
   int temp=5;
   int (*RTSclass)(int);
   RTSclass=&RTS::New;
   (*RTSclass)(temp);
   return 0;
}
```

>> This is New function in RTS class

Lv 3: pointer to New (pointer type)

```
#include <iostream>
using namespace std;
class RTS
{
public:
   static int* New(int nT)
   {
      cout << "This is New function in RTS class\n";
      return &nT;
   }
   RTS(int nT)
   {
      cout << "This is RTS class\n";
   }
};
int main()
{
   int temp=5;
   int* (*RTSclass)(int);
   RTSclass=&RTS::New;
   (*RTSclass)(temp);
   return 0;
}
```

[COMPILE warning]
RTS_15c.cpp: In static member function `static int* RTS::New(int)':
RTS_15c.cpp:8:25: warning: address of local variable `nT' returned [-Wreturn-local-addr]
    static int* New(int nT)
                        ^

>> This is New function in RTS class

Lv 4: pointer to New & self-construction

```
#include <iostream>
using namespace std;
class RTS
{
public:
   static RTS* New(int nT)
   {
      cout << "This is New function in RTS class\n";
      return (new RTS(nT));
   }
   RTS(int nT)
   {
      cout << "This is RTS class\n";
   }
};
int main()
{
   int temp=5;
   RTS* (*RTSfunction)(int);
   RTSfunction=RTS::New;
   (*RTSfunction)(temp);
   return 0;
}
```

>> This is New function in RTS class
>> This is RTS class

Lv 5: relationship between two classes

not inherited,
very dangerous
but working

```
#include <iostream>
using namespace std;
class childRTS
{
public:
   childRTS(int nT)
   {
      cout << "This is childRTS class\n";
   }
};
class parentRTS
{
public:
   parentRTS(int nT)
   {
      cout << "This is parentRTS class\n";
   }
   static parentRTS* New(int nT)
   {
      cout << "This is New function in parentRTS class\n";
      return (parentRTS*)(new childRTS(nT));
   }
};
int main()
{
   int temp=5;
   parentRTS* (*RTSfunction)(int);
   RTSfunction=parentRTS::New;
   (*RTSfunction)(temp);
   return 0;
}
```

>> This is New function in parentRTS class
>> This is childRTS class

Lv 5: relationship between two classes

not inherited,
very dangerous
but working

```
#include <iostream>
using namespace std;
class childRTS
{
public:
   childRTS(int nT)
   {
      cout << "This is childRTS class\n";
   }
};
class parentRTS
{
public:
   parentRTS(int nT)
   {
      cout << "This is parentRTS class\n";
   }
   static parentRTS* New(int nT)
   {
      cout << "This is New function in parentRTS class\n";
      return (parentRTS*)(new childRTS(nT));
   }
};
int main()
{
   int temp=5;
   parentRTS* (*RTSfunction)(int);
   RTSfunction=parentRTS::New;
   (*RTSfunction)(temp);
   return 0;
}
```

>> This is New function in parentRTS class
>> This is childRTS class

Lv 6: register and pointer to function

not inherited,
very dangerous
but working

```
#include <iostream>
using namespace std;
#include "List.H"
class parentRTS;
List<parentRTS* (*)(int)> parentRTSFuncPtr_(0);
class childRTS
{
~~~~~
};
class parentRTS
{
~~~~
};
template<class Type>
class addpatchConstructorToTable
{
public:
   addpatchConstructorToTable()
   {
      Type* (*RTSfunction)(int);
      RTSfunction=parentRTS::New;
      parentRTSFuncPtr_.setSize(parentRTSFuncPtr_.size()+1);
      parentRTSFuncPtr_[parentRTSFuncPtr_.size()-1]=RTSfunction;
   }
};
addpatchConstructorToTable<parentRTS> reg_parentRTS;
int main()
{
   int temp=5;
   (*parentRTSFuncPtr_[0])(temp);
   return 0;
}
```

>> This is New function in parentRTS class
>> This is childRTS class

(1) parentRTSFuncPtr_

type	value
parentRTS* (*)(int)	0x00
...	0x00

(2) reg_parentRTS

Type=parentRTS

```
parentRTS* (*RTSfunction)(int);
RTSfunction=parentRTS::New;
parentRTSFuncPtr_.setSize(1);
parentRTSFuncPtr_[0]=RTSfunction
```

ex. RTSfunction=0x7ff80

(1) parentRTSFuncPtr_

type	value
parentRTS* (*)(int)	0x7ff80 ← *parentRTS::New
...	0x00

register

(3) parentRTSFuncPtr_

(*parentRTSFuncPtr_[0])(temp)
→ parentRTS::New(temp)

Lv 6: register and pointer to function

not inherited, very dangerous but working

```
#include <iostream>
using namespace std;
#include "List.H“
class parentRTS;
(1) List<parentRTS* (*)(int)> parentRTSFuncPtr_(0);
class childRTS
{
~~~~~
};
class parentRTS
{
~~~~
};
template<class Type>
class addpatchConstructorToTable
{
public:
   addpatchConstructorToTable()
   {
      Type* (*RTSfunction)(int);
      RTSfunction=parentRTS::New;
      parentRTSFuncPtr_.setSize(parentRTSFuncPtr_.size()+1);
      parentRTSFuncPtr_[parentRTSFuncPtr_.size()-1]=RTSfunction;
   }
};
(2) addpatchConstructorToTable<parentRTS> reg_parentRTS;
int main()
{
   int temp=5;
   (3) (*parentRTSFuncPtr_[0])(temp);
   return 0;
}
```

>> This is New function in parentRTS class
>> This is childRTS class

Lv 7: improvement of register

inherited

```
        class parentRTS;
        List<parentRTS* (*)(int)> parentRTSFuncPtr_(0);

        class parentRTS
        {
        public:
          parentRTS()
          {
            cout << "This is parentRTS class\n";
          }
          static parentRTS* New(int nT)
          {
6)          parentRTS* pList = (*parentRTSFuncPtr_[0])(nT);
            return pList;
          }
        };

        class childRTS
        :
          public parentRTS
        {
        public:
          childRTS(int nT)
          {
            cout << "This is childRTS class\n";
          }
        };

        template<class Type>
        class addpatchConstructorToTable
        {
        public:
          static parentRTS* New(int nT)
          {
            cout << "This is New function in addpatchConstructorToTable class\n";
            return (new Type(nT));
          }
          addpatchConstructorToTable()
          {
2)          parentRTS* (*RTSfunction)(int);
3)          RTSfunction=New;
4)          parentRTSFuncPtr_.setSize(parentRTSFuncPtr_.size()+1);
5)          parentRTSFuncPtr_[parentRTSFuncPtr_.size()-1]=RTSfunction;
          }
        };

1)      addpatchConstructorToTable<childRTS> reg_childRTS;

        int main()
        {
          int temp=5;
0)        parentRTS::New(temp);

          return 0;
        }
```

0) when parentRTS::New(temp) is activated,

1) Type=childRTS
2) parentRTS* (*RTSfunction)(int)
3) RTSfunction=parentRTS* addpatchConstructorToTable::New(int)
   it will return (new childRTS)
4) Increase the size of parentRTSFuncPtr_
5) parentRTSFuncPtr_[0]=RTSfunction
6) parentRTS* pList = (*parentRTSFuncPTr_[0])(nT)
   = (*RTSfunction)(nT)
   = (*(new childRTS))(nT)

Finally childRTS(nT) will be activated includint the activation of its parent class, parentRTS!

```
>> This is New function in addpatchConstructorToTable class
>> This is parentRTS class
>> This is childRTS class
```

Lv 8: run-time selection

```
#include <iostream>
using namespace std;

#include "List.H"
class parentRTS;
List<parentRTS* (*)(int)> parentRTSFuncPtr_(0);
List<string> parentRTSString_(0);

class parentRTS
{
public:
   parentRTS()
   {
      cout << "This is parentRTS class\n";
   }
   static parentRTS* New(string lookup)
   {
      int nT;
      for(int i=0;i<parentRTSFuncPtr_.size();i++)
      {
         if(lookup==parentRTSString_[i])
         {
            parentRTS* pList = (*parentRTSFuncPtr_[i])(nT);
            return pList;
         }
      }
   }
};

class child1RTS
:
   public parentRTS
{
public:
   static string typeName()
   {
      return "child1";
   }
   child1RTS(int nT)
   {
      cout << "This is child1RTS class\n";
   }
};

class child2RTS
:
   public parentRTS
{
public:
   static string typeName()
   {
      return "child2";
   }
   child2RTS(int nT)
   {
      cout << "This is child2RTS class\n";
   }
};
```

```
template<class Type>
class addpatchConstructorToTable
{
public:
   static parentRTS* New(int nT)
   {
      cout << "This is New function in addpatchConstructorToTable class\n";
      return (new Type(nT));
   }
   addpatchConstructorToTable()
   {
      parentRTS* (*RTSfunction)(int);
      RTSfunction=New;
      parentRTSFuncPtr_.setSize(parentRTSFuncPtr_.size()+1);
      parentRTSFuncPtr_[parentRTSFuncPtr_.size()-1]=RTSfunction;
      parentRTSString_.setSize(parentRTSString_.size()+1);
      parentRTSString_[parentRTSString_.size()-1]=Type::typeName();
   }
};

addpatchConstructorToTable<child1RTS> reg_child1RTS;
addpatchConstructorToTable<child2RTS> reg_child2RTS;

int main()
{
   int temp=5;
   parentRTS::New("child2");
   parentRTS::New("child1");

   return 0;
}
```

concept of hashTable

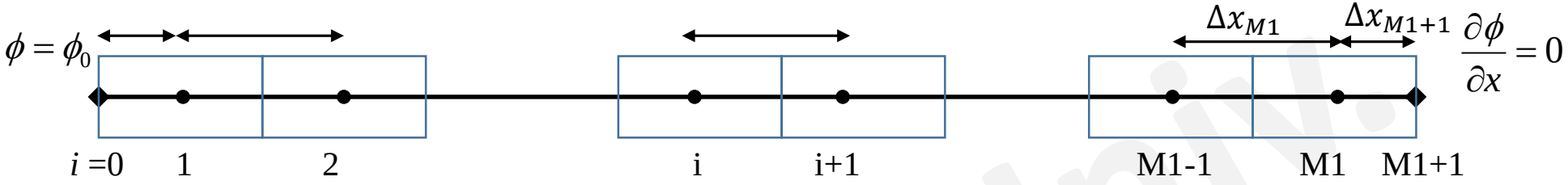
type	value	typeName	hash key
parentRTS* (*)(int)	0x7ff80	“child1”	1
…	0x7ff90	“child2”	2

```
>> This is New function in addpatchConstructorToTable class
>> This is parentRTS class
>> This is child2RTS class
>> This is New function in addpatchConstructorToTable class
>> This is parentRTS class
>> This is child1RTS class
```

Now you can add child3, child4, …

```
template <class Type, template<class> class fvPatchFieldType>
class addpatchConstructorToTable
{
public:
   static fvPatchField<Type>* New(const List<Type>& field, const int& nPatch, istream& is)
   {
      return (new fvPatchFieldType<Type>(field,nPatch,is));
   }
   addpatchConstructorToTable(const string& lookup=fvPatchFieldType<Type>::typeName())
   {
      fvPatchField<Type>* (*runTimeSelectionFunction)(const List<Type>&, const int&, istream&);
      runTimeSelectionFunction=New;
      patchConstructorName.setSize(patchConstructorName.size()+1);
      patchConstructorName[patchConstructorName.size()-1]=lookup;
      patchConstructorPtr.setSize(patchConstructorPtr.size()+1);
      patchConstructorPtr[patchConstructorPtr.size()-1]=runTimeSelectionFunction;
   }
};
```

addpatchConstructorToTable class



$$\begin{bmatrix} \frac{V_1}{\Delta t} + \alpha\left(\frac{A_{f_1}}{\Delta x_1} + \frac{A_{f_2}}{\Delta x_2}\right) & -\frac{\alpha A_{f_2}}{\Delta x_2} & 0 & 0 & 0 \\ -\frac{\alpha A_{f_2}}{\Delta x_2} & \frac{V_2}{\Delta t} + \alpha\left(\frac{A_{f_2}}{\Delta x_2} + \frac{A_{f_3}}{\Delta x_3}\right) & -\frac{\alpha A_{f_3}}{\Delta x_3} & 0 & 0 \\ 0 & -\frac{\alpha A_{f_3}}{\Delta x_3} & \frac{V_3}{\Delta t} + \alpha\left(\frac{A_{f_3}}{\Delta x_3} + \frac{A_{f_4}}{\Delta x_4}\right) & -\frac{\alpha A_{f_4}}{\Delta x_4} & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & -\frac{\alpha A_{f_n}}{\Delta x} & \frac{V_n}{\Delta t} + \alpha\left(\frac{A_{f_n}}{\Delta x_n}\right) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{bmatrix}$$

$$A \times \phi = S$$

FVM

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
        phiEqn.solve(phi,nTime);
        phi.correctBoundaryConditions();
    }
    phi.write();
    return 0;
}
```

```
int main()
{
    fvMesh mesh;
    GeometricField<double>
        phi(IOobject("phi",  ".", true, true) ,mesh);
    for(int nTime=0;nTime<NTST;nTime++)
    {
        fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
        phiEqn.solve(phi,nTime);
        phi.correctBoundaryConditions();
    }
    phi.write();
    return 0;
}
```

beginnerFoam_1D_14th.cpp

```
#include "List.H"
#include "PtrList.H"
#include "tmpFor14th.H"
#include "fvMeshFor14th.H"
#include "IOobjectFor14th.H"
#include "GeometricFieldFor14th.H"
#include "fvMatrixFor14th.H"
#include "fvmFor14th.H"

int main()
{
   fvMesh mesh;
   GeometricField<double> phi
   (
      IOobject
      (
         "phi",
         ".",
         true,
         true
      )
     ,mesh
   );

   for(int nTime=0;nTime<NTST;nTime++)
   {
      fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
      phiEqn.solve(phi,nTime);
      phi.correctBoundaryConditions();
   }
   phi.write();
   return 0;
}
```

beginnerFoam_1D_15th.cpp

```
#include "List.H"
#include "PtrList.H"
#include "tmpFor15th.H"

template <class T>
class fvPatchField;

static List<fvPatchField<double>* (*)(const List<double>&, const int&, istream&) >
patchConstructorPtr(0);
static List<string> patchConstructorName(0);

#include "fvMeshFor15th.H"
#include "IOobjectFor15th.H"
#include "GeometricFieldFor15th.H"
#include "fvMatrixFor15th.H"
#include "fvmFor15th.H"
#include "addpatchConstructorToTableFor15th.H"

addpatchConstructorToTable<double, fixedValueFvPatchField> reg_fixedValueFvPatchField_;
addpatchConstructorToTable<double, zeroGradientFvPatchField> reg_zeroGradientFvPatchField_;

int main()
{
   fvMesh mesh;
   GeometricField<double> phi
   (
      IOobject
      (
         "phi",
         ".",
         true,
         true
      )
     ,mesh
   );

   for(int nTime=0;nTime<NTST;nTime++)
   {
      fvMatrix phiEqn(fvm::ddt(phi)-fvm::laplacian(phi));
      phiEqn.solve(phi,nTime);
      phi.correctBoundaryConditions();
   }
   phi.write();
   return 0;
}
```

```
template <class T>
class fvPatchField
:
   public List<T>
{
private:
   bool updated_;
public:
   fvPatchField(const int& nPatch)
   :
      updated_(false)
   {
      this->setSize(nPatch);
   }

   static fvPatchField<T>* New(const List<T>& field, const int& nPatch, istream& is,
const string& patchFieldType)
   {
      for(int i=0;i<patchConstructorPtr.size();i++)
      {
         if(patchFieldType==patchConstructorName[i])
         {
            fvPatchField<T>* pList = (*patchConstructorPtr[i])(field,1,is);
            return pList;
         }
      }
   }
   void evaluate(const double& value)
   {
      if(!updated_) updateCoeffs(value);
      updated_ = false;
   }
   virtual void updateCoeffs(const double& value)
   {
      updated_ = false;
   }
};
```

```
template <class T>
class fvPatchField
:
   public List<T>
{
private:
   bool updated_;
public:
   fvPatchField(const int& nPatch)
   :
      updated_(false)
   {
      this->setSize(nPatch);
   }

   static fvPatchField<T>* New(const List<T>& field, const int& nPatch, istream&
is, const string& patchFieldType)
   {
      if(patchFieldType==fixedValueFvPatchField<T>::typeName())
      {
         fvPatchField<T>* pList = new fixedValueFvPatchField<T>(field,1,is);
         return pList;
      }
      else if(patchFieldType==zeroGradientFvPatchField<T>::typeName())
      {
         fvPatchField<T>* pList = new zeroGradientFvPatchField<T>(field,1,is);
         return pList;
      }
   }
   void evaluate(const double& value)
   {
      if(!updated_) updateCoeffs(value);
      updated_ = false;
   }
   virtual void updateCoeffs(const double& value)
   {
      updated_ = false;
   }
};
```

fvPatchField class

Construction of New selector in OpenFOAM

fvPatchField class

```
template<class Type>
class fvPatchField
:
   public Field<Type>
{
   // Private data
   const fvPatch& patch_;
   const DimensionedField<Type, volMesh>& internalField_;
   bool updated_;
   bool manipulatedMatrix_;
   word patchType_;
public:
   typedef fvPatch Patch;
   typedef calculatedFvPatchField<Type> Calculated;
   TypeName("fvPatchField");

   static int disallowGenericFvPatchField;

   declareRunTimeSelectionTable
      (
         tmp,
         fvPatchField,
         patch,
         (
            const fvPatch& p,
            const DimensionedField<Type, volMesh>& iF
         ),
         (p, iF)
      );
...
};
```

fvPatchField.H

```
TypeName("fvPatchField");
```

```
static const char* typeName_()
{
   return “fvPatchField";
}
static const ::Foam::word typeName
static int debug
virtual const word& type() const
{
   return typeName;
}
```

```
#define TypeName(TypeNameString)
   ClassName(TypeNameString);
   virtual const word& type() const { return typeName; }

#define ClassName(TypeNameString)
   ClassNameNoDebug(TypeNameString);
   static int debug

#define ClassNameNoDebug(TypeNameString)
   static const char* typeName_() { return TypeNameString; }
   static const ::Foam::word typeName
```

Construction of New selector in OpenFOAM

fvPatchField class

```
template<class Type>
class fvPatchField
:
   public Field<Type>
{
   // Private data
   const fvPatch& patch_;
   const DimensionedField<Type, volMesh>& internalField_;
   bool updated_;
   bool manipulatedMatrix_;
   word patchType_;
public:
   typedef fvPatch Patch;
   typedef calculatedFvPatchField<Type> Calculated;
   TypeName("fvPatchField");

   static int disallowGenericFvPatchField;

   declareRunTimeSelectionTable
     (
        tmp,
        fvPatchField,
        patch,
        (
           const fvPatch& p,
           const DimensionedField<Type, volMesh>& iF
        ),
        (p, iF)
     );
...
};
```

fvPatchField.H

```
declareRunTimeSelectionTable
    (tmp, fvPatchField, patch, (const fvPatch& p, const DimensionedField<Type, volMesh>& iF), (p, iF));
```

```
typedef tmp<fvPatchField> (*patchConstructorPtr)(const fvPatch& p, const DimensionedField<Type, volMesh>& iF);
typedef HashTable<patchConstructorPtr, word, string::hash> patchConstructorTable;
static patchConstructorTable* patchConstructorTablePtr_;
static void constructpatchConstructorTables();
static void destroypatchConstructorTables();
template<class fvPatchFieldType>
class addpatchConstructorToTable
{
public:
   static tmp<fvPatchField> New (const fvPatch& p, const DimensionedField<Type, volMesh>& iF)
   {
      return tmp<fvPatchField>(new fvPatchFieldType (p, iF));
   }
   addpatchConstructorToTable (const word& lookup = fvPatchFieldType::typeName)
   {
      constructpatchConstructorTables();
      if (!patchConstructorTablePtr_->insert(lookup, New))
      {
         std::cerr<< "Duplicate entry " << lookup << " in runtime selection table " << fvPatchField << std::endl;
         error::safePrintStack(std::cerr);
      }
   }
   ~addpatchConstructorToTable()
   {
      destroypatchConstructorTables();
   }
};
```

Construction of New selector in OpenFOAM

totalPressureFvPatchField class

```
namespace Foam
{
class totalPressureFvPatchScalarField
:
   public fixedValueFvPatchScalarField
{
   // Private data
   …
public:
   TypeName("totalPressure");
   totalPressureFvPatchScalarField
   (
      const fvPatch&,
      const DimensionedField<scalar, volMesh>&
   );
   ….
};
```

totalPressureFvPatchField.H

```
#include "totalPressureFvPatchScalarField.H"
#include "addToRunTimeSelectionTable.H"
#include "fvPatchFieldMapper.H"
#include "volFields.H"
#include "surfaceFields.H“
Foam::totalPressureFvPatchScalarField::totalPressureFvPatchScalarField
(const fvPatch& p, const DimensionedField<scalar, volMesh>& iF}
…
{}
.
.
namespace Foam
{
   makePatchTypeField
   (
      fvPatchScalarField,
      totalPressureFvPatchScalarField
   );
}
```

totalPressureFvPatchField.C

```
makePatchTypeField(fvPatchScalarField, totalPressureFvPatchScalarField);
```

↓

```
#define makePatchTypeField(fvPatchScalarField, totalPressureFvPatchScalarField)
   defineTypeNameAndDebug(totalPressureFvPatchScalarField, 0);
   addToPatchFieldRunTimeSelection(fvPatchScalarField, totalPressureFvPatchScalarField)
```

fvPatchField.H

↓

```
#define defineTypeNameAndDebug(totalPressureFvPatchScalarField, 0)
   defineTypeName(totalPressureFvPatchScalarField);
   defineDebugSwitch(totalPressureFvPatchScalarField, 0)
```

className.H

↓

```
defineTypeNameWithName(totalPressureFvPatchScalarField, totalPressureFvPatchScalarField::typeName_())
defineDebugSwitchWithName(Type, Type::typeName_(), DebugSwitch);
registerDebugSwitchWithName(Type, Type, Type::typeName_())
```

className.H & defineDebugSwitch.H

↓

```
const ::Foam::word totalPressureFvPatchScalarField::typeName(“totalPressure”)
```

className.H

Construction of New selector in OpenFOAM

totalPressureFvPatchField class

```
namespace Foam
{
class totalPressureFvPatchScalarField
:
   public fixedValueFvPatchScalarField
{
   // Private data
   …
public:
   TypeName("totalPressure");
   totalPressureFvPatchScalarField
   (
      const fvPatch&,
      const DimensionedField<scalar, volMesh>&
   );
   ….
};
```

totalPressureFvPatchField.H

```
#include "totalPressureFvPatchScalarField.H"
#include "addToRunTimeSelectionTable.H"
#include "fvPatchFieldMapper.H"
#include "volFields.H"
#include "surfaceFields.H“
Foam::totalPressureFvPatchScalarField::totalPressureFvPatchScalarField
(const fvPatch& p, const DimensionedField<scalar, volMesh>& iF}
…
{}
.
.
namespace Foam
{
   makePatchTypeField
   (
      fvPatchScalarField,
      totalPressureFvPatchScalarField
   );
}
```

totalPressureFvPatchField.C

```
makePatchTypeField(fvPatchScalarField, totalPressureFvPatchScalarField);
```

```
#define makePatchTypeField(fvPatchScalarField, totalPressureFvPatchScalarField)
   defineTypeNameAndDebug(totalPressureFvPatchScalarField, 0);
   addToPatchFieldRunTimeSelection(fvPatchScalarField, totalPressureFvPatchScalarField)
```

fvPatchField.H

```
addToRunTimeSelectionTable(fvPatchScalarField,totalPressureFvPatchScalarField,patch);
addToRunTimeSelectionTable(fvPatchScalarField,totalPressureFvPatchScalarField,patchMapper);
addToRunTimeSelectionTable(fvPatchScalarField,totalPressureFvPatchScalarField,dictionary);
```

fvPatchField.H

```
fvPatchScalarField::addpatchConstructorToTable<totalPressureFvPatchScalarField>
   addtotalPressureFvPatchScalarFieldpatchConstructorTofvPatchScalarFieldTable_
```

```
typedef tmp<fvPatchField> (*patchConstructorPtr)(const fvPatch& p, const DimensionedField<Type, volMesh>& iF);
typedef HashTable<patchConstructorPtr, word, string::hash> patchConstructorTable;
static patchConstructorTable* patchConstructorTablePtr_;
static void constructpatchConstructorTables();
static void destroypatchConstructorTables();
template<class fvPatchFieldType>                    // fvPatchFieldType = totalPressureFvPatchScalarField
class addpatchConstructorToTable
{
public:
   static tmp<fvPatchField> New (const fvPatch& p, const DimensionedField<Type, volMesh>& iF)
   {
      return tmp<fvPatchField>(new fvPatchFieldType (p, iF));
   }
   addpatchConstructorToTable (const word& lookup = fvPatchFieldType::typeName)
   {
      constructpatchConstructorTables();
      if (!patchConstructorTablePtr_->insert(lookup, New))
      { … }
   }
};
```

Construction of New selector in OpenFOAM

totalPressureFvPatchField class

```
namespace Foam
{
class totalPressureFvPatchScalarField
:
   public fixedValueFvPatchScalarField
{
   // Private data
   …
public:
   TypeName("totalPressure");
   totalPressureFvPatchScalarField
   (
      const fvPatch&,
      const DimensionedField<scalar, volMesh>&
   );
   ….
};
```

totalPressureFvPatchField.H

```
#include "totalPressureFvPatchScalarField.H"
#include "addToRunTimeSelectionTable.H"
#include "fvPatchFieldMapper.H"
#include "volFields.H"
#include "surfaceFields.H“
Foam::totalPressureFvPatchScalarField::totalPressureFvPatchScalarField
(const fvPatch& p, const DimensionedField<scalar, volMesh>& iF)
…
{}
.
.
namespace Foam
{
   makePatchTypeField
   (
      fvPatchScalarField,
      totalPressureFvPatchScalarField
   );
}
```

totalPressureFvPatchField.C

```
makePatchTypeField(fvPatchScalarField, totalPressureFvPatchScalarField);
```

```
fvPatchScalarField::addpatchConstructorToTable<totalPressureFvPatchScalarField>
   addtotalPressureFvPatchScalarFieldpatchConstructorTofvPatchScalarFieldTable_
```

```
typedef tmp<fvPatchField> (*patchConstructorPtr)(const fvPatch& p, const DimensionedField<Type, volMesh>& iF);
static HashTable<patchConstructorPtr, word, string::hash>* patchConstructorTablePtr_;
static void constructpatchConstructorTables();
static void destroypatchConstructorTables();
template<class totalPressureFvPatchScalarField>
class addpatchConstructorToTable
{
public:
   static tmp<fvPatchField> New(const fvPatch& p, const DimensionedField<Type, volMesh>& iF)
   {
      return tmp<fvPatchField> (new totalPressureFvPatchScalarField (p, iF));
   }
   addpatchConstructorToTable (const word& lookup = totalPressureFvPatchScalarField::typeName)
   {
      constructpatchConstructorTables();
      if (!patchConstructorTablePtr_->insert(lookup, New))
      { … }
   }
};
```

Mechanism of New selector in OpenFOAM

totalPressureFvPatchField.C

```
namespace Foam
{
   makePatchTypeField
   (
      fvPatchScalarField,
      totalPressureFvPatchScalarField
   );
}
```

insert New function with “totalPressure” keyword into HashTable
{ return new totalPressureFvPatchScalarField(~~); }

Boundary::readField() in GeometricBoundaryField.C

```
this->set(patchi, PatchField<Type>::New(bmesh_[patchi], field, iter().dict()));
```

```
template<class Type>
Foam::tmp<Foam::fvPatchField<Type>> Foam::fvPatchField<Type>::New
(
   const fvPatch& p,
   const DimensionedField<Type, volMesh>& iF,
   const dictionary& dict
)
{
   const word patchFieldType(dict.lookup("type"));
   typename dictionaryConstructorTable::iterator cstrIter = dictionaryConstructorTablePtr_->find(patchFieldType);
   return cstrIter()(p, iF, dict);
}
```

find New function with “totalPressure” keyword in HashTable
{ return new totalPressureFvPatchScalarField(~~); }

return New function { return new totalPressureFvPatchScalarField(~~); }

setting boundary condition on i^{th} boundary patch
void set(const int i, T* ptr) { this->ptrs_[i] = ptr; }

UPtrList.C