Assignment 2 program:

- A standard "functional" implementation

- The main(...) routine
  - obtains program parameters
  - declares and initializes variables
  - runs timestepping loop
  - calls output routines

- External Functions (subroutines) called to perform required computations.

ConvectionTest.cpp

Include statements : .h, “header”, files containing external function declarations.

Local definitions : local function declarations. Implementations at the end of the file.

Main routine : Read in program parameters
Initialize variables

```
for(k = 1; k <= stepCount; k++)
{
advanceTimeStep(S, U, V, gridData, runData);
totalTime += runData.dt;

if((k%outputCount) == 0)
{
     outputData(S,gridData,k);
}

errorS(S,gridData,totalTime,errorL2,errorMax);
cout << "ERR L2 : " <<  errorL2 <<  " ERR MAX. : " <<
errorMax << endl;
}
```

advanceTimeStep in ConvectionRoutines.cpp

```
void advanceTimeStep(DoubleArray2D& S, const DoubleArray2D& U, const DoubleArray2D& V,
const GridParameters& gridData, const RunParameters& runData)
{
double dt        = runData.dt;
long rkOrder     = runData.rkOrder;
                 *
                 *
long m = gridData.m;
long n = gridData.n;

DoubleArray2D Sn(m+1,n+1);
DoubleArray2D Snp1(m+1,n+1);
DoubleArray2D Sstar(m+1,n+1);
DoubleArray2D FS(m+1,n+1);

Sn     = S; Snp1   = S; Sstar  = S;

// RK loop - loop over stages

for (i=0; i < rkOrder; i++)
{
FS    = evaluateConvectionOp(Sstar,U,V,gridData,runData);
Snp1  = Snp1 + dt*rkk(i)*FS;
if((i+1) < rkOrder)
{
 Sstar  = Sn + dt*rk(i+1)*FS;
}}
S = Snp1;
}
```

Specific C++ constructs used

- "Container" classes for parameters (= C struct)

  GridParameters : specify parameters associated with the grid, e.g. hx,hy.

  RunParameters : specify parameters associated with the particular run, e.g. the ODE method

- 2D array class:

  DoubleArray2D : stores grid node values and provides array operations

  Indexing starts at 0!

Why container classes?

- When the set of parameters changes, one doesn't have to change routine argument lists!

  Fortran 77

```
subroutine convct(s,u,v,con,m,n,a,b,c,d,testrn)
implicit real*8 (a-h,o-z)|
real*8 s(m+1,n+1)
real*8 con(m+1,n+1)
real*8 u(m+1,n+1)
real*8 v(m+1,n+1)
character*1 testrn
```

  C++

```
DoubleArray2D evaluateConvectionOp(DoubleArray2D& s, DoubleArray2D& u,
DoubleArray2D& v, GridParameters& gridData, RunParameters& runData)
```

If another method or grid parameter needs to be passed in, the Fortran calling sequence must change.

What is in a container class?

Class declaration in GridParameters.h = class definition

```
class GridParameters
{
public :

double xMin;   // Computational Region is [xMin,xMax]X,[yMin,yMax]
double xMax;
double yMin;
double yMax;
long   m;      // Number of panels in the x direction
long   n;      // Number of panels in the y direction
double hx;     // mesh width in x direction
double hy;     // mesh width in y direction
};
```

```
GridParameters gridData;

gridData.xMin = 4.0;
gridData.xMax = 4.0;
          *
          *

x1 = gridData.xMin;
x2 = gridData.xMax;
```

Why an array class?

- Dynamic memory allocation (e.g. one can specify array sizes as the program runs).
- Takes care of new/delete --- reduces likelihood of memory leaks.
- One can "overload" algebraic operators to support array operations. This enables Matlab style programming, e.g.

```
A = B + C;
```

where A, B, and C are DoubleArray2D instances.

Default indexing starts at 0!

What's in DoubleArray2D?

Class declaration* in DoubleArray2D.h

Constructors/Initializers: Specification of how you declare (instantiate) instances

```
DoubleArray2D();
DoubleArray2D(long m, long n);
DoubleArray2D(const DoubleArray2D& D);

DoubleArray2D   A;          //  DoubleArray2D of zero size

DoubleArray2D A(5,6);      // DoubleArray2D of 5 by 6

DoubleArray2D B(A);        // DoubleArray2D B is created as a copy of A;


void initialize(long m, long n);

A.initialize(5,6);              // An existing DoubleArray2D is set to a 5 by 6 instance.
```

*and implementation

What's in DoubleArray2D?

Structure Information:

```
long getIndex1Begin() const;
long getIndex2Begin() const;

long getIndex1End() const;
long getIndex2End() const;

long getIndex1Size() const;
long getIndex2Size() const;
```

```
long mStart = A.getIndex1Begin();
long nStart =  A.getIndex2Begin();

long mEnd = A.getIndex1End();
long nEnd   =A.getIndex2End() ;

for(j = nStart; j <= nEnd; j++)
{
   for(i = mStart; i <= mEnd; i++)
   {
      A(i,j) = 0.0;
}}
```

What's in DoubleArray2D?

Element Access and Output:

```
double&  operator()(long i1, long i2);
const double&  operator()(long i1, long i2) const;
```

z = A(2,3);

A(0,1) = 5.0:

```
friend ostream&  operator <<(ostream& outStream, const DoubleArray2D& A);
```

cout << A << endl;   // dump A to console

What's in DoubleArray2D?

Array operations:

`DoubleArray2D operator+(const DoubleArray2D& D);`	**A+B**
`DoubleArray2D operator-(const DoubleArray2D& D);`	**A-B**
`DoubleArray2D operator*(double alpha);`	**A*6.0**
`friend DoubleArray2D operator*(double alpha, const DoubleArray2D& D);`	**6.0*A**
`DoubleArray2D operator/(double alpha);`	**A/6.0**
`void operator=(const DoubleArray2D& D);`	**A=B**
`void operator*=(double alpha);`	**A*=6.0 (A=A*6.0)**
`void operator+=(const DoubleArray2D& D);`	**A+=B (A=A+B)**
`void operator-=(const DoubleArray2D& D);`	**A-=B (A=A-B)**
`void setToValue(double d);`	**A.setToValue(6.0)**
`double dot(const DoubleArray2D& D) const;`	**A.dot(B) (un-scaled by mesh size)**

Array class in action...

```
DoubleArray2D Sn(m+1,n+1);
DoubleArray2D Snp1(m+1,n+1);
DoubleArray2D Sstar(m+1,n+1);
DoubleArray2D FS(m+1,n+1);

Sn     = S;
Snp1   = S;
Sstar  = S;
//
// RK loop - loop over stages
//
long i;
for (i=0; i < rkOrder; i++)
{
    FS    = evaluateConvectionOp(Sstar,U,V,gridData,runData);
    Snp1  = Snp1 + dt*rkk(i)*FS;

     if((i+1) < rkOrder)
     {
        Sstar  = Sn + dt*rk(i+1)*FS;
     }
}
S = Snp1;
```

Other C++ information

Math 157 PPT notes: ".h, .cpp and .lib files"

Why do we use .h and .cpp files?

Math 157 PPT notes: "Preprocessor Directives"

About pre-processor directives (e.g. #ifdef ...)

Available as .pdf's from 270e class web page.