PROGRAM #2

Converting program 1 => program 2

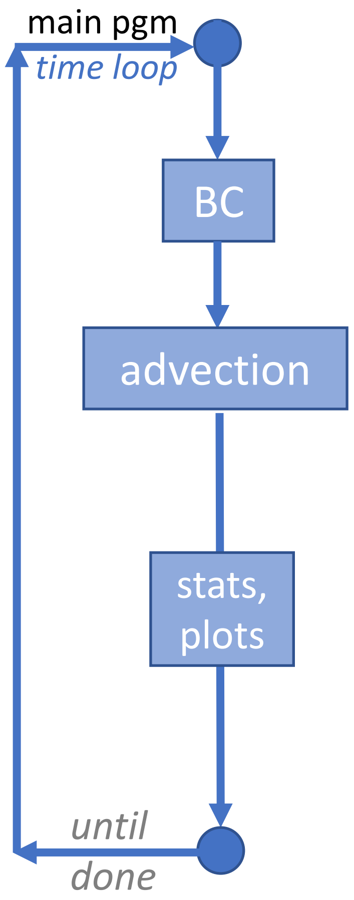
in C and Fortran

Program 1 – *overview* - C

-- Program 1 and its subroutines do this --

- calls <u>ic</u> routine to set s1() initial conditions
- calls <u>bc</u> routine to set periodic BCs
- variables passed to the advection routine:
  - 1d “time level n” scalar field s1[NXDIM]
  - 1d “time level n+1” field s2[NXDIM]
  - *fixed* flow speed c, time step dt, spatial increment dx
- <u>advection</u> routine:
  - takes as input: flow speed c, grid spacing dx and time step dt
  - courant = c*dt/dx (can be set *before* the for-loop, since c is constant)
  - *loop* uses Lax-Wendroff scheme , loop I1 ..I2

    s2[i] = s1[i] – courant * …
  - s2 array has updated values returned to main program.



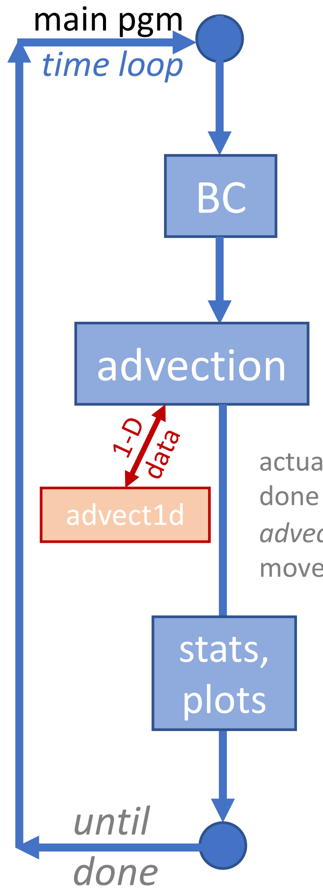
why "-I1" here? because the for-loop is over grid values *with* ghost points, but our u1d[] array – like our 2d u[][] and v[][] arrays – have *no* ghost points. u1d[0] is one-half grid length to the left of s1d[I1] !!

Program 2 – changes in **bold!** - C

-- Program 2 and its subroutines do this --

- dimension **2D arrays** s1, s2 and **2D velocity component arrays** U, V
  - **delete** history[] and "c" variables – no longer needed
- pass s1, s2 to ic() and bc() routines;
  - implement **two-dimensional** IC, as well as **0-gradient** BCs
- **new *2-D* advection routine**: *calls advect1d.*
  - input from pgm2.c: **2-D arrays** s1, s2, U, V; only s1,s2 have ghost points.
  - also input: dt, dx, and the advection-type choice
  - declare **new 1-D arrays** s1d_in(), s1d_out(), u1d()
  - for X & Y advection: copy s1 to s1d_in(), U-or-V to u1d(), pass 1D arrays to *advect1d*, copy s1d_out back to s1()
- **advect1d() routine:** *start this with copy of old advection routine!*
  - input: **constants** (dt, dx, advection type), **1-D arrays** (s1d_in, s1d_out, u1d)
  - uses Lax-Wendroff scheme , (still) *1-D* for-loop I1 ..I2
    - set *courant number* inside do-loop:
      *courant = dt/dx*0.5*(u1d[i-I1]+u1d[i+1-I1])*
    - s1d_out(i) = s1d_in(i) – courant * …

loop



actual "work" is done by *advect1d*. *advection* just moves data.

Program 2 – summary - C

*If you haven't already started. This makes a complete copy of one folder (pgm1) and puts it in the other (pgm2).

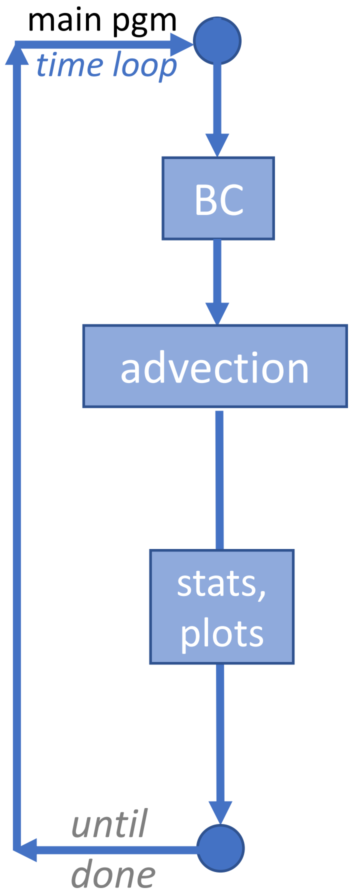
- Make a copy* of your pgm1 folder and call it pgm2: *cp -R pgm1 pgm2*
- In pgm2.c add #define for *J1, J2, NYDIM* similar to *I1, I2, NXDIM.*
- Change *BC_WIDTH* to 2 or 3 (3 if planning to do extra-credit)
- Implement 2D arrays! *s1, s2, strue arrays will be [NXDIM][NYDIM] & and have ghost points*
  - *Remember later in the class, NXDIM will not equal NYDIM*.
  - add 2-D velocity arrays *u* and *v* – neither will have any ghost points – remember staggering!!
  - change your pgm2.c call to *advection()* to also pass velocity arrays u, v.
- Implement your 2-D initial condition inside *ic()*, plot it, compare to mine.
- Implement your 2-D 0-gradient boundary conditions inside *bc().*
- Copy *advection.c* to *advect1d.c* *advect1d.c is most easily started as a copy of pgm1's advection.c!*
  - Make the changes to *advect1d* shown in the previous slide: no "c" variable, pass a 1-D *u1d* (or velocity1D or whatever you call it) array containing the 1D flow speed.
  - Move courant number math *inside* your Lax-Wendroff loop as shown on prior slide.
- Change *advection.c :* Make old s1, s2 arrays to be 2-D, add 2-D velocity arrays, add new 1-D arrays, pass 1-D slices of *s1* and of velocity to *advect1d*.
- Try *pgm2* first by doing 2D contour plots *every* time step.

Program 1 – overview – Fortran 90

- Global_data module:

  contains these variables -

  - grid dimension nx
  - grid spacing dx
  - flow speed c
  - history array()
  - 1D arrays:
    - s1, s2, strue



-- Program 1 and its subroutines do this --

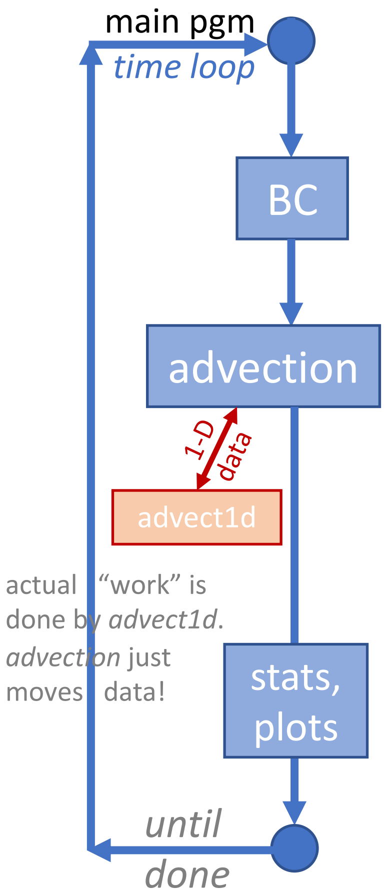
- calls ic to set *s1*, bc to set periodic BCs
- calls the advection() routine:
  - passes only dt and advection_type to *advection*
- advection() routine: *does all the "work"*
  - advection() does "USE global_data" for 1D arrays
  - there is only (1-D) X-advection here
  - can set *courant number* before do-loop:
    - *courant = dt/dx*c* (since c = constant)
  - *loop* uses Lax-Wendroff scheme , 1-D loop 1…nx
    - s1d_out(i) = s1d_in(i) – courant * …

Program 2 – changes in **bold**! – Fortran 90

- Global_data module:

  contains these variables -

  - grid dimension nx
  - **add**: grid dim *ny*
  - grid spacing dx
  - ~~flow speed c~~ *--delete--*
  - ~~history array()~~ *these vars!*
  - **now 2d** arrays:
    - s1, s2, strue
  - **add 2d** arrays:
    - u, v flow arrays



-- Program 2 and its subroutines do this --

- calls ic to set **2-D *s1***, bc to set **0-gradient** BCs
- calls the **(now 2-D)** advection() routine:
  - passes only dt and advection_type to *advection*
- advection() routine: ***now handles 2D+1D arrays***
  - advection() still does "USE global_data" for 2D arrays
  - declare **new 1-D arrays** s1d_in(), s1d_out(), u1d()
  - for X & Y advection: copy S1 & U-or-V to s1d_in(), u1d(), pass 1D arrays to *advect1d*, copy s1d_out back to s1()
- **advect1d() routine:** *start this with copy of old advection routine!*
  - do Not "*USE global_data*" here! *everything passed*
  - uses Lax-Wendroff scheme , (still) *1-D* loop 1…nx

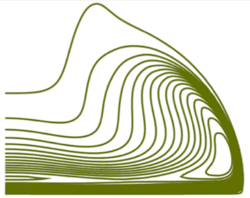
    loop
    - set *courant number* inside do-loop:

      *courant = dt/dx*0.5*(u1d(i)+u1d(i+1))*
    - s1d_out(i) = s1d_in(i) – courant * …

Program 2 – summary – Fortran90

*If you haven't already started. This makes a complete copy of one folder (pgm1) and puts it in the other (pgm2).

- Make a copy* of your pgm1 folder and call it pgm2: *cp -R pgm1 pgm2*
- In global_data.f90:
  - add 2nd dimension "ny", set equal to nx. *later in the semester nx will not equal ny!*
  - make scalar arrays 2D! *s1, s2, strue arrays will be (-2:nx+3,-2:ny+3) if you use 3 ghost points*
  - add 2-D velocity arrays *u* and *v* – neither will have any ghost points – remember staggering!!
- Implement your 2-D initial condition inside *ic()*, plot it, compare to mine.
- Implement your 2-D 0-gradient boundary conditions inside *bc().*
- Copy *advection.f90* to *advect1d.f90* *advect1d.f90 is most easily started as a copy of pgm1's advection.f90!*
  - Make the changes to *advect1d* shown in the previous slide: no "c" variable, pass a 1-D *u1d* (or velocity1D or whatever you call it) array containing the 1D flow speed.
  - Move courant number math *inside* your Lax-Wendroff loop as shown on prior slide.
- Change *advection.f90* : Change s1, s2 arrays to be 2-D, add 2-D velocity arrays, add new 1-D arrays, pass 1-D slices of *s1* and of velocity to *advect1d*.
- Try *pgm2* first by doing 2D contour plots *every* time step.

 *Either language:* 2-D Advection routine 

I call the first dimension (columns) "i" and 2nd dimension "j" (rows). You don't have to do that if you prefer a different convention!!

- Advecting rows (X) *all j (rows)*
  - Loop over *all rows* (2nd dimension, **j**)
    - Loop over all columns *i* *with ghost points*
      - ✓ copy **s1**(i,j) to s1d_in
    - Loop over all columns **i**=1,nx+1 *(no ghost points)*
      - ✓ copy **u**(i,j) to u1d
    - call ***advect1d***
      - ✓ pass s1d_in, u1d to advect1d
      - ✓ advect1d returns updated *s1d_out*
    - Loop over all columns **i** =1,nx *(no ghost points)*
      - ✓ copy s1d_out to **s1**(i,j)

- Advecting columns (Y) *all i (columns)*
  - Loop over *all columns* (1st dim., **i**)
    - Loop over all rows *j* *with ghost points*
      - ✓ copy **s1**(i,j) to s1d_in
    - Loop over all rows **j**=1,ny+1 *(no ghost points)*
      - ✓ copy **v**(i,j) to u1d
    - call ***advect1d***
      - ✓ pass s1d_in, u1d to advect1d
      - ✓ advect1d returns updated *s1d_out*
    - Loop over all rows **j** =1,ny *(no ghost points)*
      - ✓ copy s1d_out to **s1**(i,j)

Since nx=ny, you can use *s1d_in, s1d_out, u1d* for both X- and Y- data slices. Later when nx, ny differ, you declare based on the larger dimension.