MESA

run_star_extras

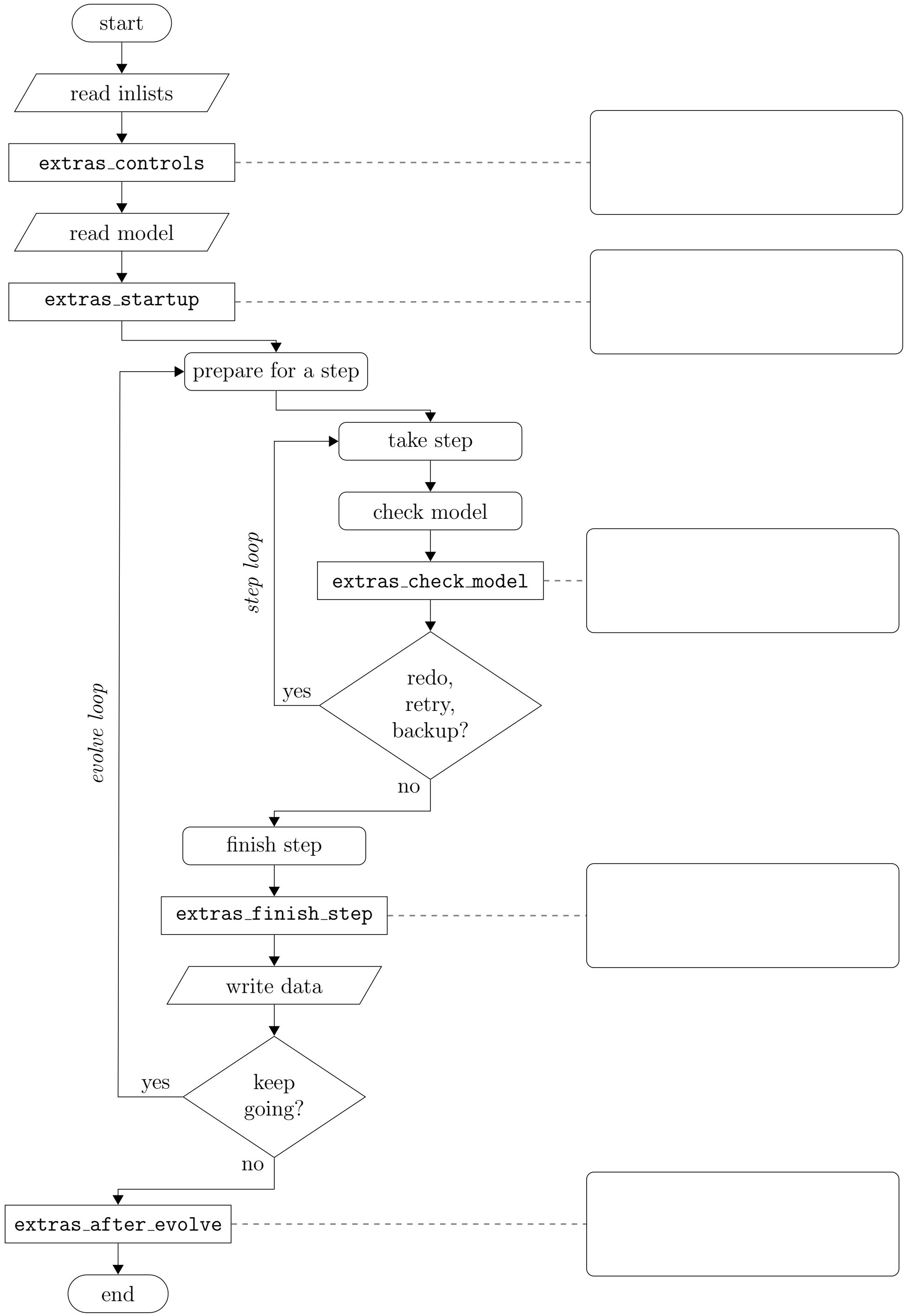
Ilaria Caiazzo, Jeremy Heyl. TAs: Amber Lauer, Kang Liu

A special thanks to Josiah Schwab for letting us use his examples!

run_star_extras gives you the possibility to add commands that aren’t already available. Moreover, you can use it to override MESA’s built-in physics routines.

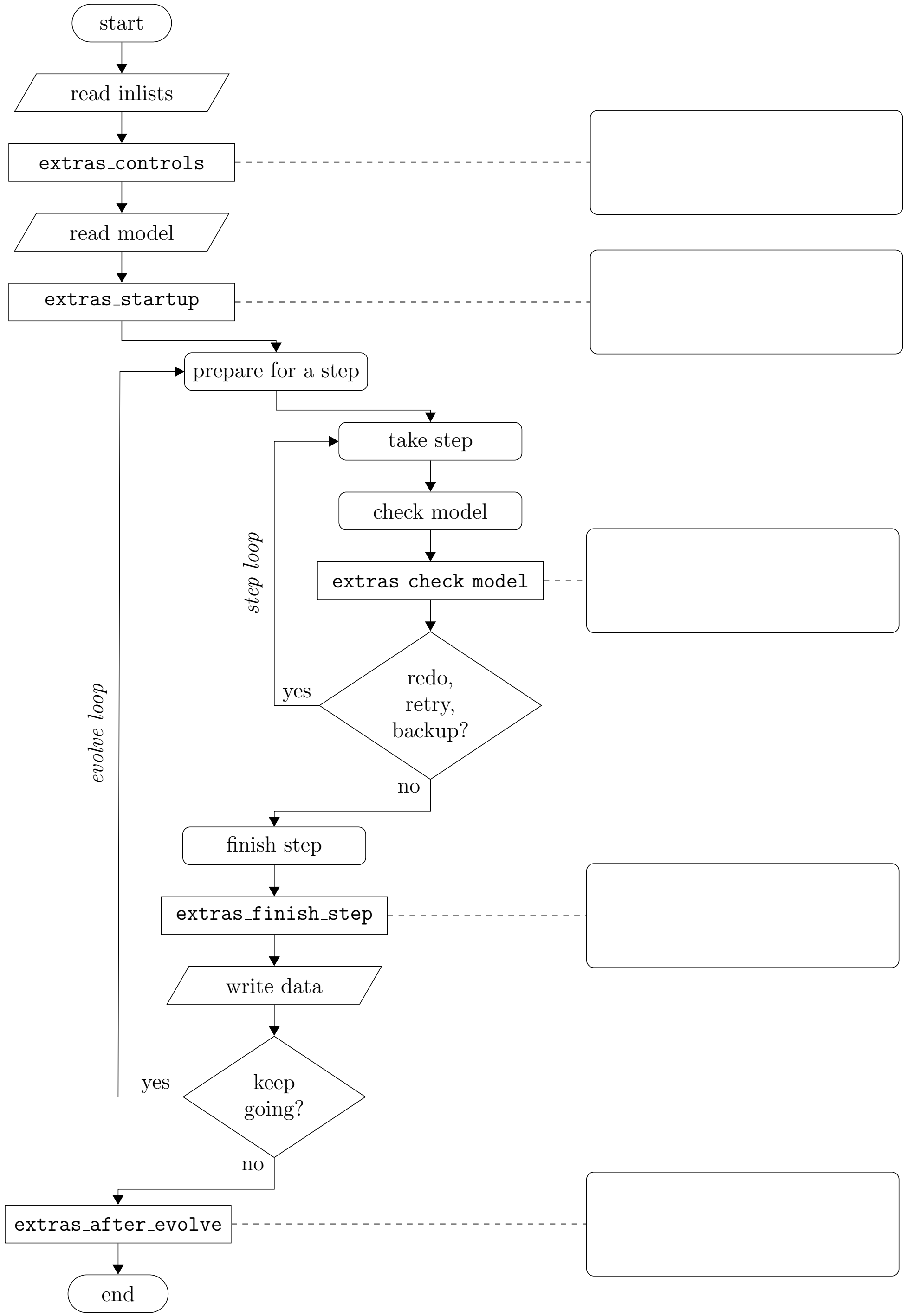
run_star_extras is written in Fortran. However, don’t be afraid if you don’t know fortran. If you have experience in coding in any other language you will be fine. If you get stuck on Fortran, just ask your fellow students and TAs for help!

You can find a short introduction to Fortran commands that you may need in the appendix of the pdf file



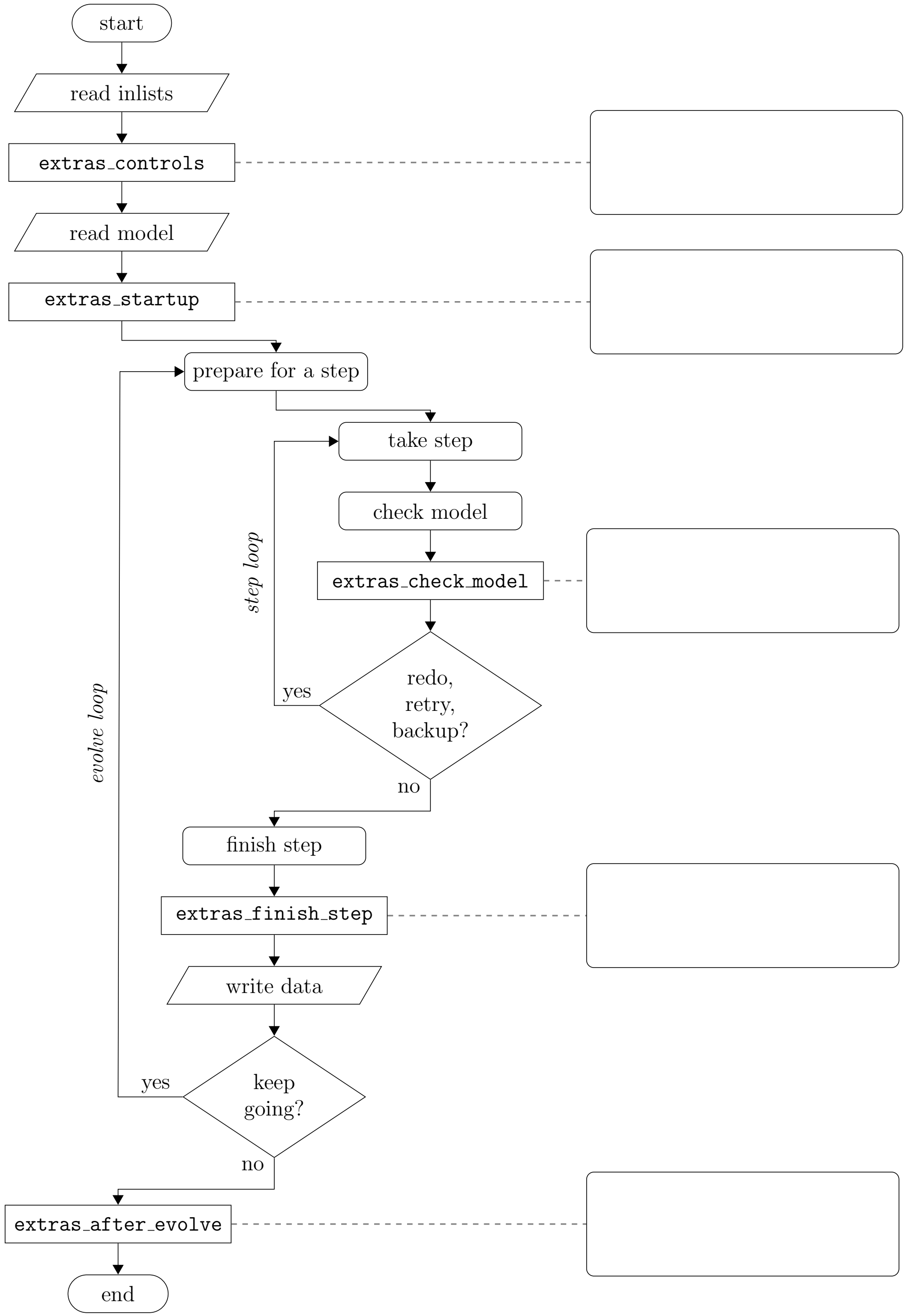
The different run_star_extras.f routines get called at different points during MESA execution.

They give you hooks to customize the software at different stages of the execution.



Let’s say that we want to stop the run when the stars is greater than a certain radius.

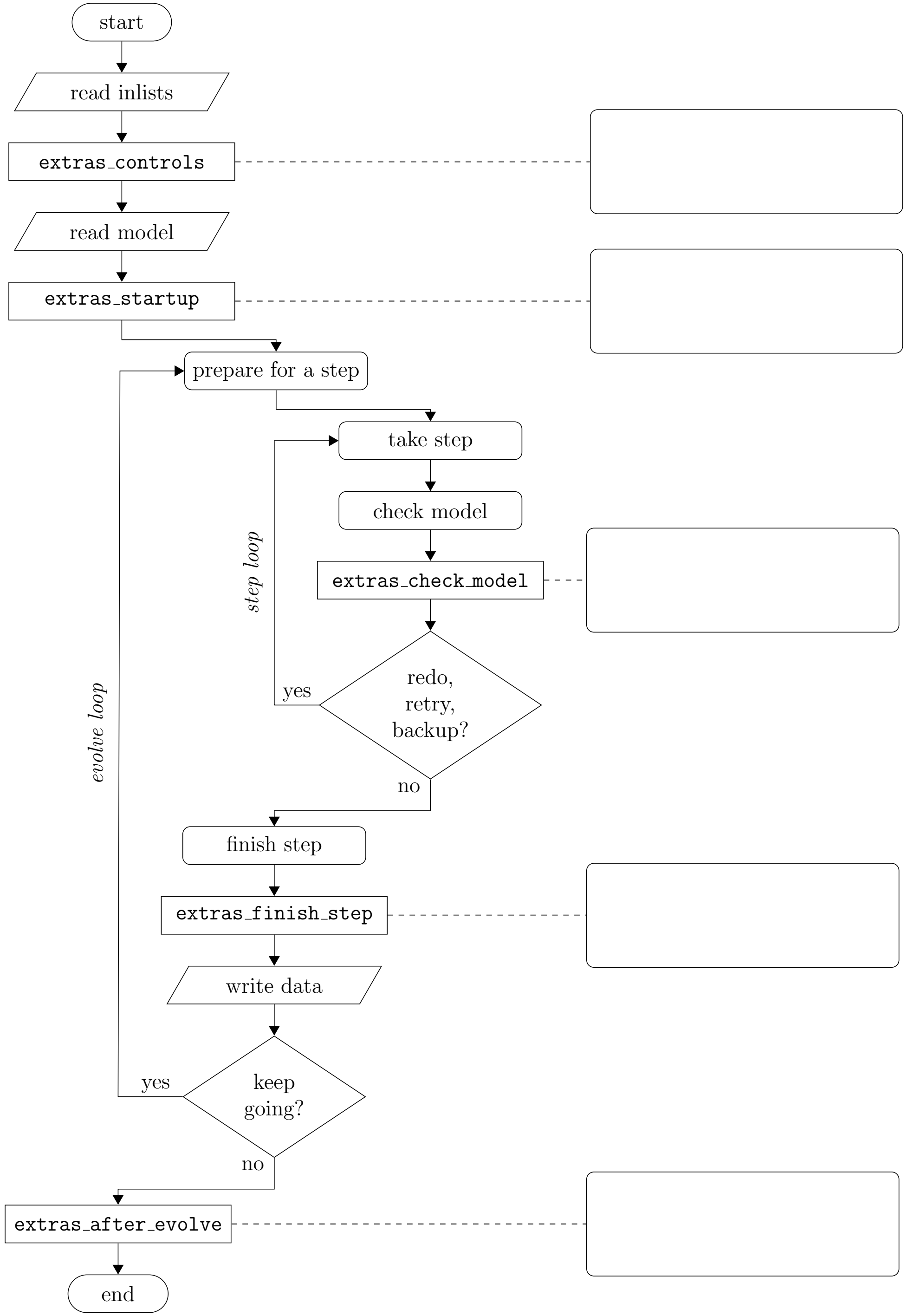
Which hook would you use?



Let’s say that we want to stop the run when the stars is greater than a certain radius.

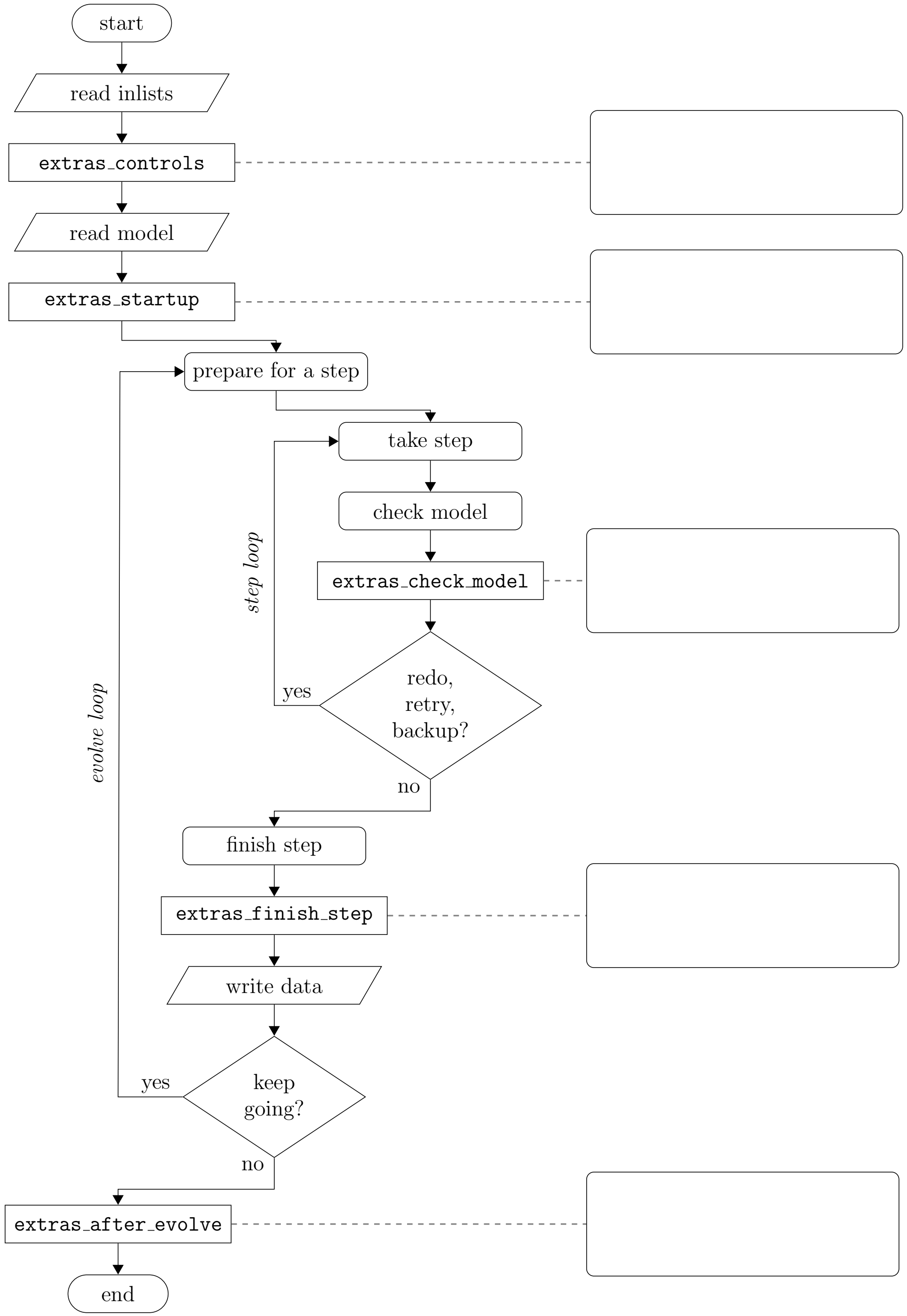
Which hook would you use?

In the extra_finish_step routine you can check the star’s radius and stop the run.



What if you wanted the run to stop at a specific radius (a certain values +or- something)?

Which hook would you use?



What if you wanted the run to stop at a specific radius (a certain values +or- something)?

Which hook would you use?

In this case, you have to use the extra_check_model.

Example - adding a stopping condition

Ok, let's say that we want the run to terminate when the star's radius is bigger than a certain value and we don't care how close the final model is to that radius. As we saw in the previous slide, the right routine to use is extras_finish_step

- You can use the same folder you were using for pgplot, your 1M_pre_ms_to_wd.
- open src/run_star_extra.f
- If you look at the names the subroutines and functions defined, they are the same as the ones listed in the flowchart.
- look for extras_finish_step
- The output by default is keep_going.

Example - adding a stopping condition

We want MESA to check at the end of each step if the value of the star’s radius is greater than a certain value, and if so, to stop the run.

First, we need to understand the language for that. At each step, MESA stores all the information about the star in the structure called star_info. In run_star_extras.f this structure is simply referred to with the variable named s. Other than the stellar model itself (i.e., mass, luminosity, composition profile, thermodinamics profile and so on) the star_info structure contains, as components, all the commands you can set from the control section of the inlist (i.e., initial_mass, xa_central_lower_limit etc).

The components of star_info are listed in the file $MESA_DIR/star/public/star_data.inc. In Fortran, the percent (%) operator is used to access the components of the structure. (So you can read s% x = 3 in the same way that you would read s.x = 3 in C.).

Example - adding a stopping condition

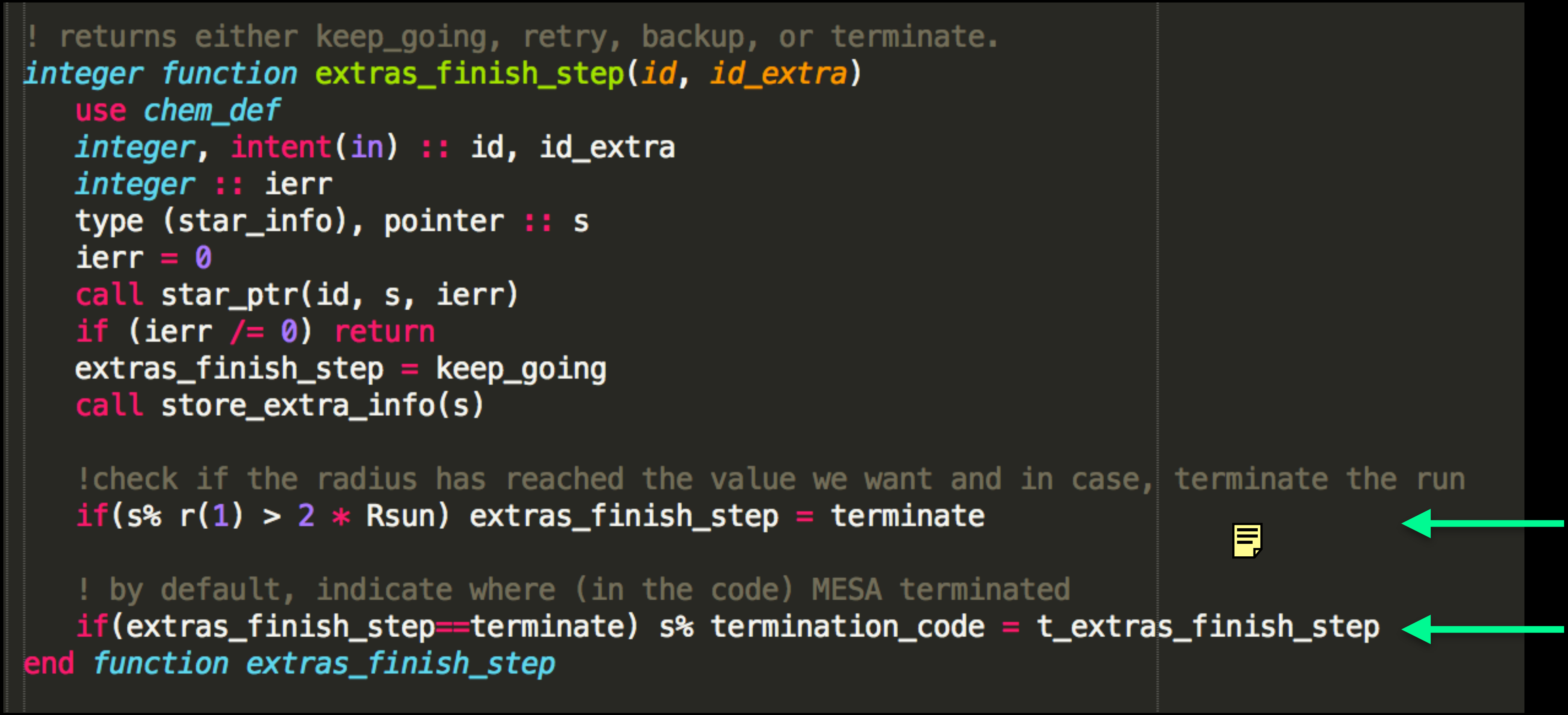
So let’s take a look at what’s in star_info.

- open $MESA_DIR/star/public/star_data.inc and start looking around.
- if you search for the word radius, you can read "r(k) is radius at outer edge of cell k". (In MESA, the outermost zone is at k=1 and the innermost zone is at k=s% nz.) Therefore, the radius of the star is s% r(1).

MESA uses cgs units unless otherwise noted. The most common non-cgs units are solar units. MESA defines its constants in $MESA_DIR/const/public/const_def.f. If we use the built in constants we are sure we're using exactly the same definitions as MESA. The constant with the value of the solar radius (in cm) is named Rsun.

Example - adding a stopping condition

Now we check the value of the radius at the end of each step and, if it’s greater than 2 Rsun, we tell MESA to terminate the run:



```
! returns either keep_going, retry, backup, or terminate.
integer function extras_finish_step(id, id_extra)
   use chem_def
   integer, intent(in) :: id, id_extra
   integer :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   extras_finish_step = keep_going
   call store_extra_info(s)

   !check if the radius has reached the value we want and in case, terminate the run
   if(s% r(1) > 2 * Rsun) extras_finish_step = terminate

   ! by default, indicate where (in the code) MESA terminated
   if(extras_finish_step==terminate) s% termination_code = t_extras_finish_step
end function extras_finish_step
```

Example - adding a stopping condition

Whenever you change run_star_extras.f, you have to ./mk the folder again.

If you try it out, the run will end immediately after the relaxation steps because the model during the pre-main sequence has a radius bigger than 2 solar radii. Indeed, you can read in the output “termination code: extras_finish_step”.

What we want is for MESA to stop on the RGB.

Example - adding a stopping condition

In order to do that, we can add another condition to the if statement.

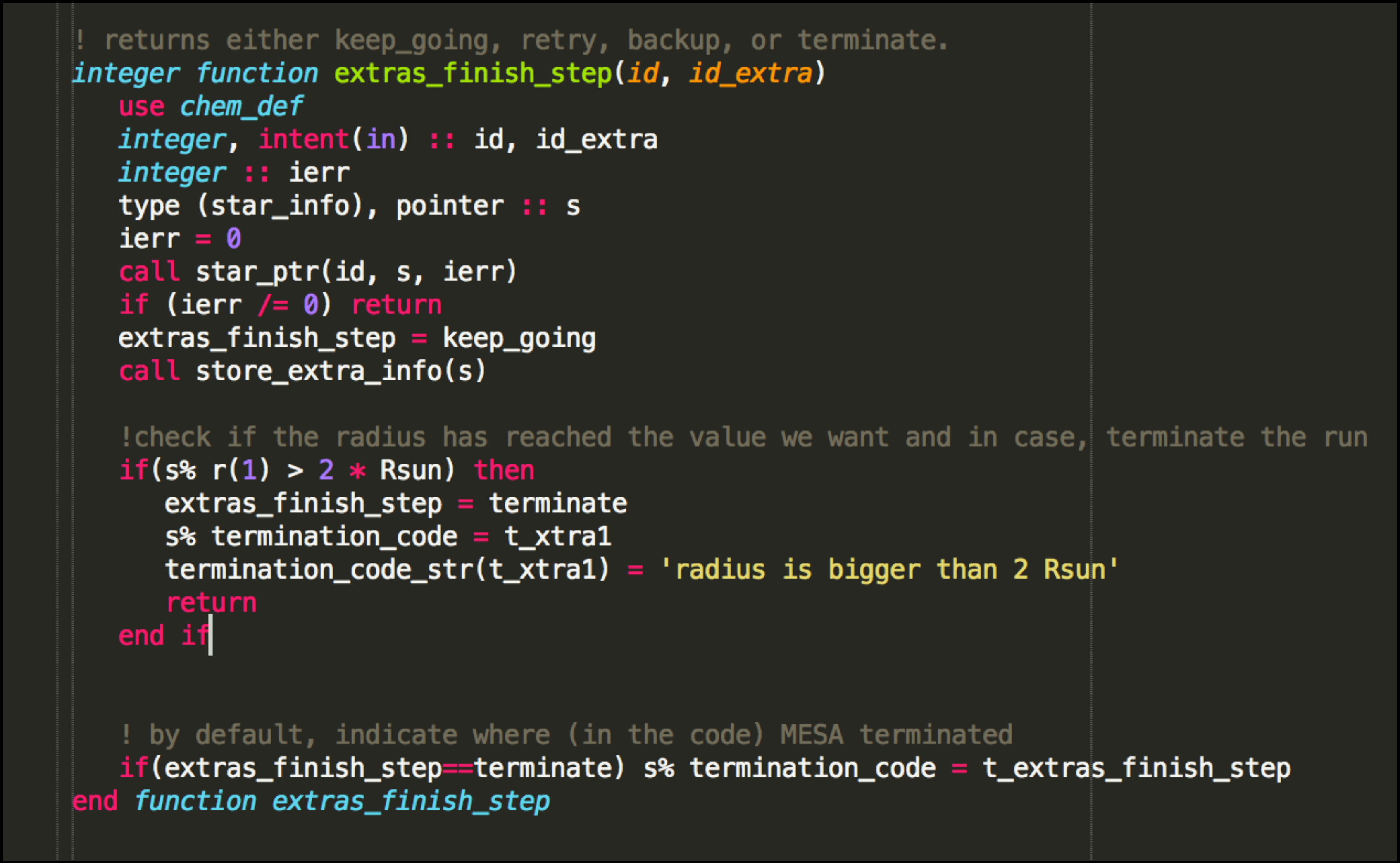
For example, you can ask MESA to stop the run if the radius is greater than 2 solar radii AND if the hydrogen at the centre of the star is less than a small value (like 10^-4). This will make sure that the stopping condition applies after the end of the main sequence phase.

In order to do this, you have to find which component of star_info stores the value of the central hydrogen at each step.

Example - adding a stopping condition

As you move to more complicated customizing, you might have different stopping conditions in the same routine. In this case, you might want to have specific names for the termination_code. MESA provides 9 customizeable termination codes, named t_xtra1 .. t_xtra9. You can customize the messages that will be printed upon exit by setting the corresponding termination_code_str value:

Example - adding a stopping condition



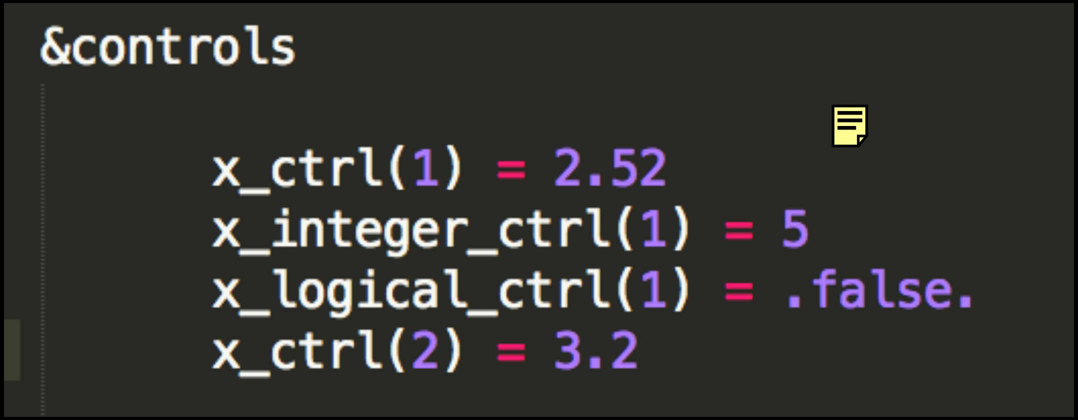
```fortran
! returns either keep_going, retry, backup, or terminate.
integer function extras_finish_step(id, id_extra)
   use chem_def
   integer, intent(in) :: id, id_extra
   integer :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   extras_finish_step = keep_going
   call store_extra_info(s)

   !check if the radius has reached the value we want and in case, terminate the run
   if(s% r(1) > 2 * Rsun) then
      extras_finish_step = terminate
      s% termination_code = t_xtra1
      termination_code_str(t_xtra1) = 'radius is bigger than 2 Rsun'
      return
   end if


   ! by default, indicate where (in the code) MESA terminated
   if(extras_finish_step==terminate) s% termination_code = t_extras_finish_step
end function extras_finish_step
```

Changing run_star_extras variables from the inlist

If you want to change the values of your conditions without having to compile every time, you can use a set of controls that are part of the star_info structure and that are set in the inlist. They are called x_ctrl, x_integer_ctrl, and x_logical_ctrl and they are arrays of length 100, respectively of double, integer and boolean precision. You can assign a value to them in the inlist:



```
&controls

      x_ctrl(1) = 2.52
      x_integer_ctrl(1) = 5
      x_logical_ctrl(1) = .false.
      x_ctrl(2) = 3.2
```

and access them later on as part of the star structure (like s% x_ctrl(1), etc.).

“Other” physics routines

MESA provides hooks to override its built-in physics routines. These hooks are referred to as "other" routines. The other routines are listed in the folder $MESA_DIR/star/other. You can look for the one you want, from changing the diffusion or the neutrino physics to changing the winds and so on, and copy and paste it into your run_star_extras.f, where you can modify it as you please.