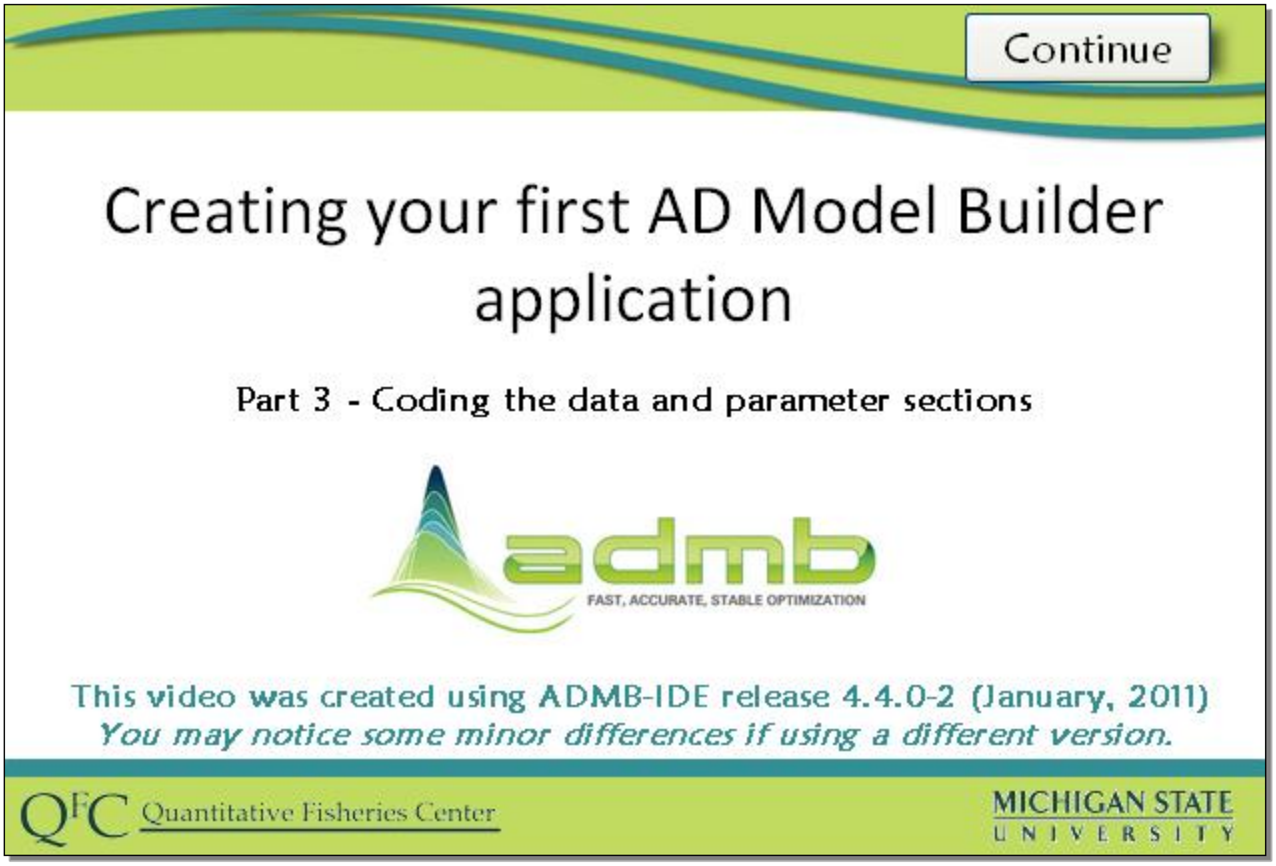
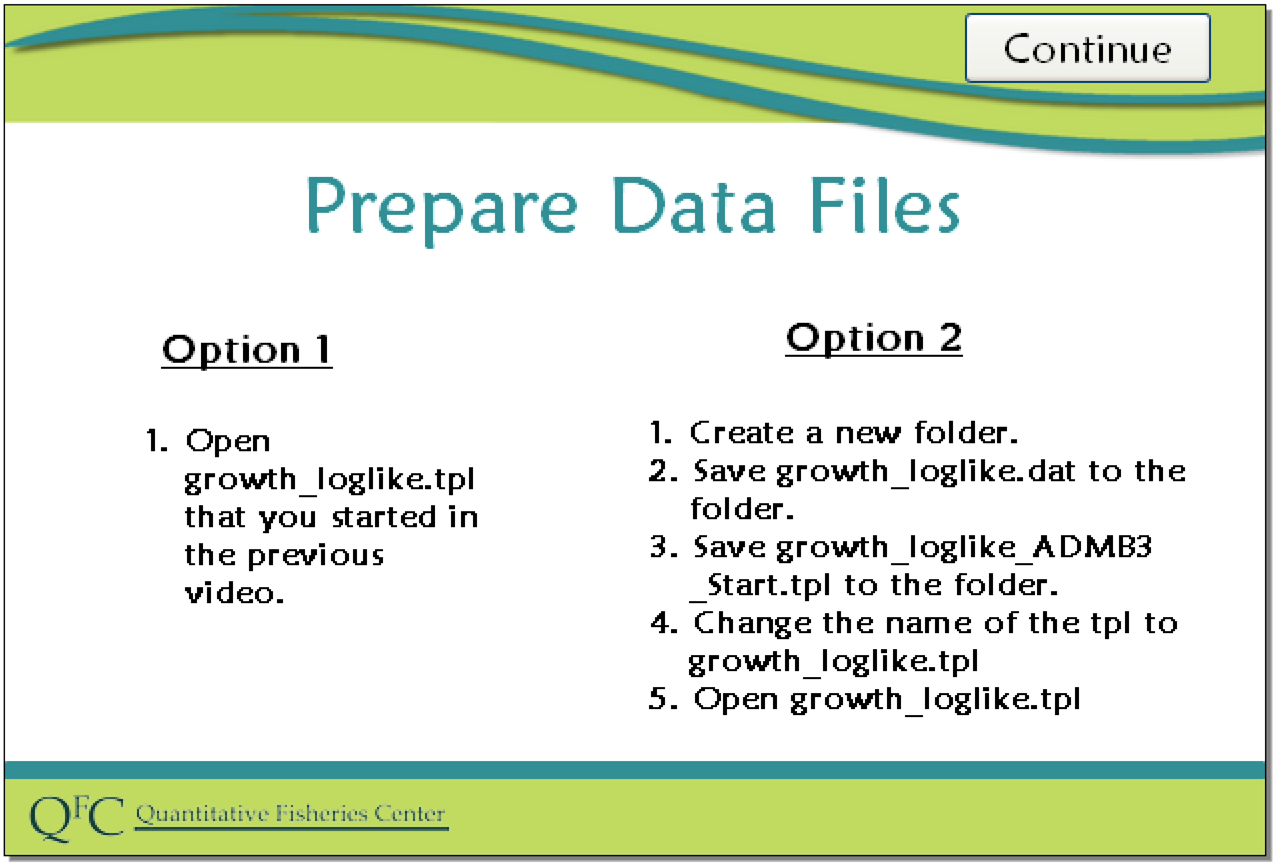
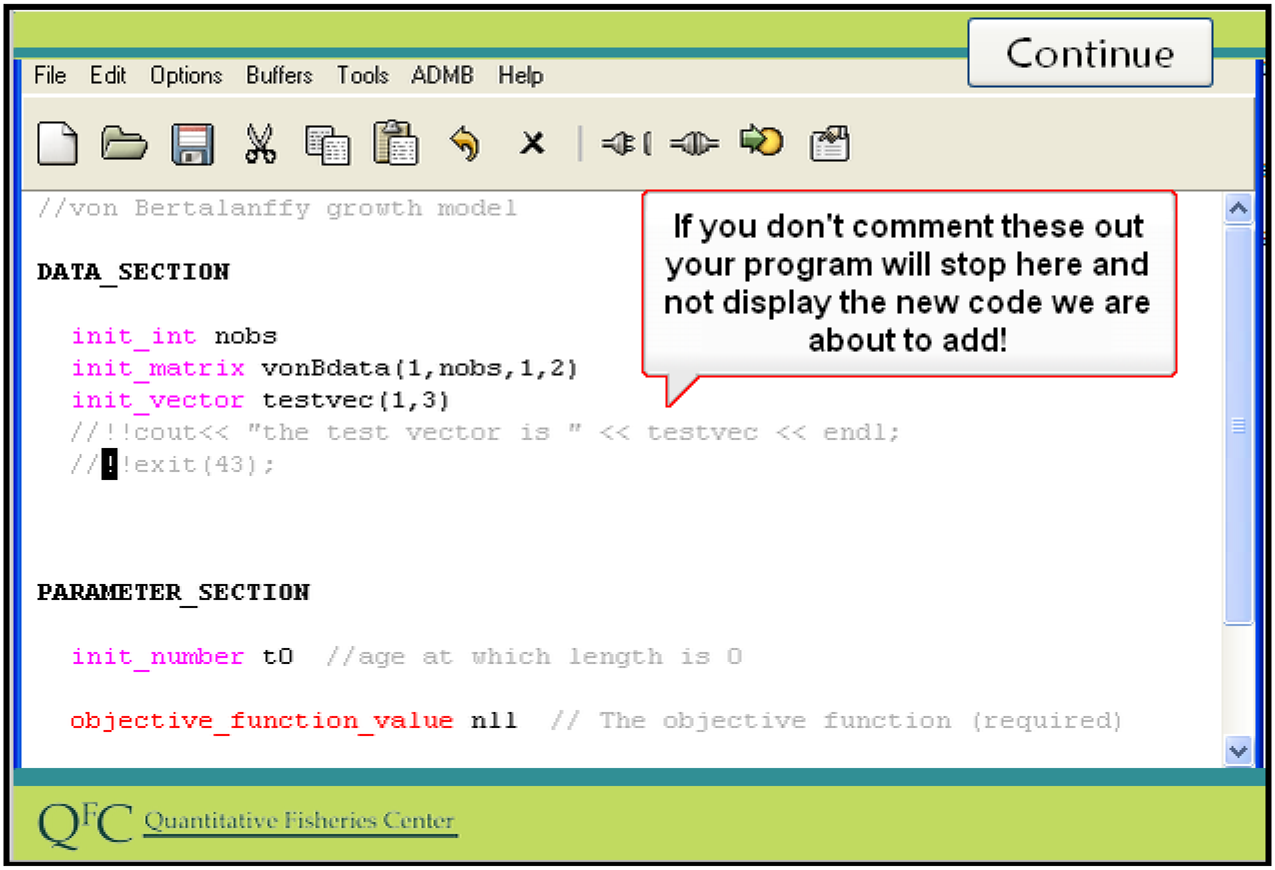



In this video we complete the data and parameter sections, add some code to the procedure section and write out some results to see where we have gotten too.




You have two options for preparing your data. If you worked through the previous video, you can simply open your template and continue working. Otherwise, you will need to create a new folder and save the documents. You must rename the growth_loglike_ADMB3_Start.tpl to be growth_loglike.tpl because remember, the file names need to be the same for the tpl and the dat file. Click next when you are ready.

Either open growth_loglike.tpl that you worked with in the last video, or create a new folder, save growth_loglike.dat to the folder, save growth_loglike_ADMB3_Start.tpl to the new folder and change the name of that tpl to growth_loglike.tpl then open it.




Before proceeding with coding changes, the first thing we do is "comment out" the cout and exit statements we previously included in the code.

We previously included these to test our program by writing out a result and then quitting. Putting double slashes in front of these lines turns them into comments. Instead of turning them into comments you could have just deleted these lines.

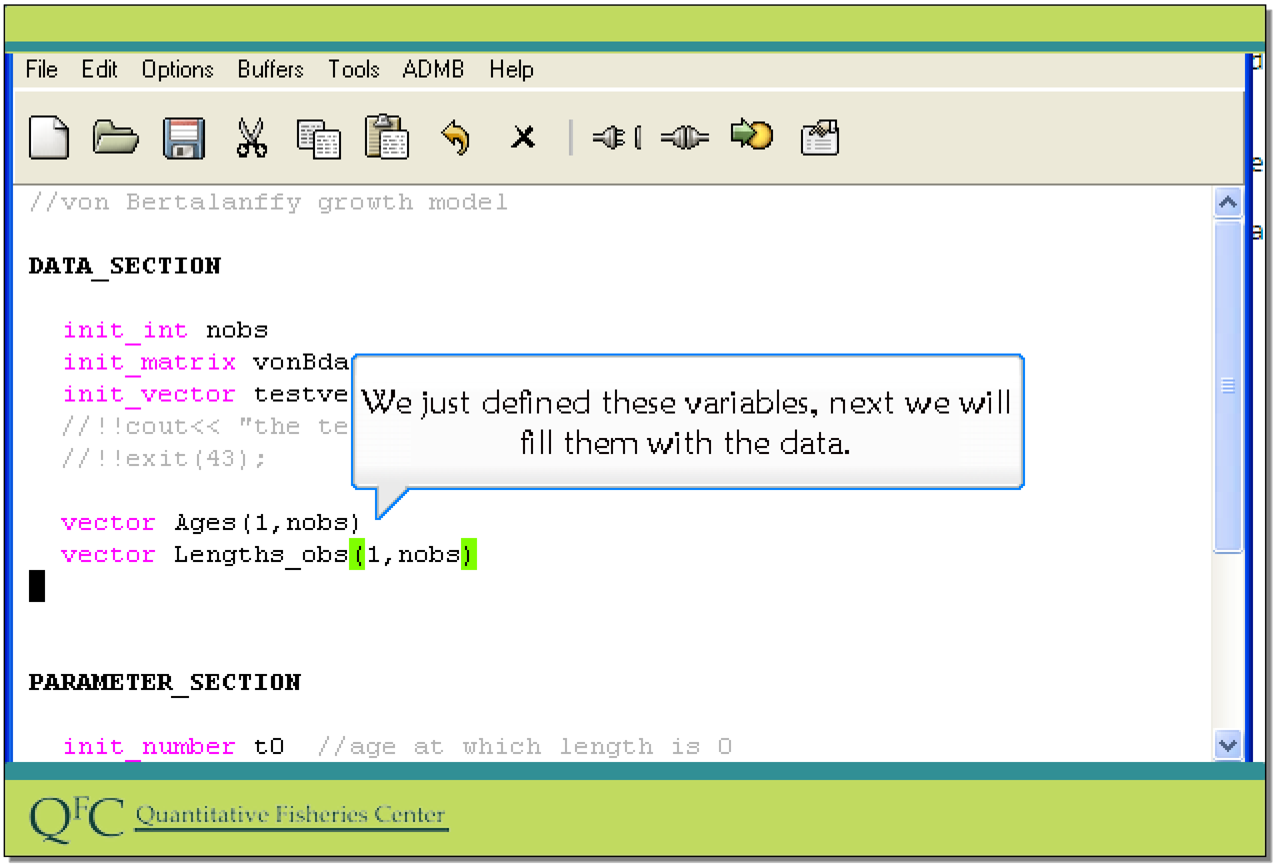
Commenting out lines can sometimes be faster than deleting them and is often wise when you think you might want to change the code back. If I didn't comment out or delete the exit line the program would continue to quit when it got to that line in the code and all your new work would be for nothing. Not commenting out the old cout statement means that the distracting test you previously sent to the screen would continue to be sent there even though it is no longer needed.

**Slide Code:**

**// !!cout<<…………**

**// !!exit……………..**

# Data_Section



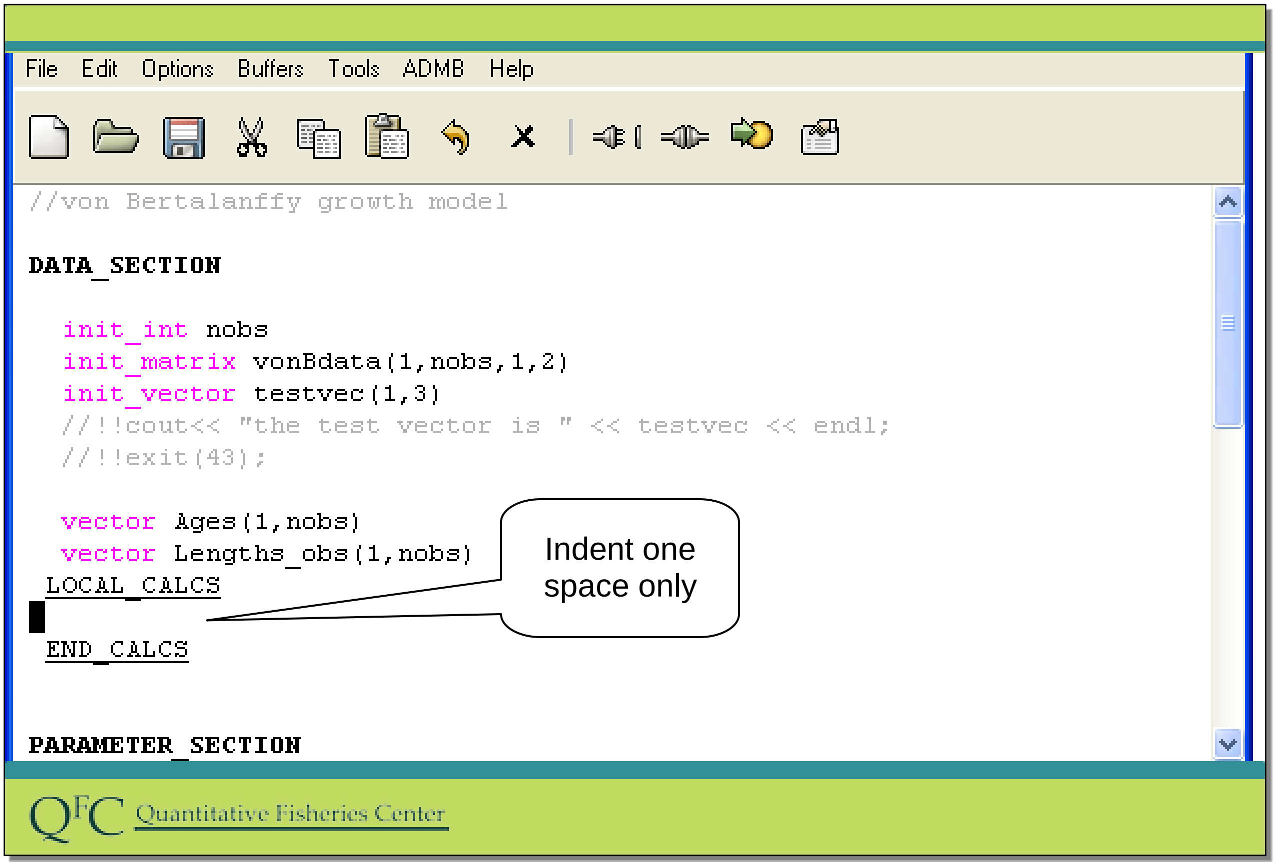

We complete the data section by creating vectors to hold age and length data and filling them with the right columns from the vonBdata matrix. First we need to define the variables Ages and Lengths_obs to be vectors of the right dimension.

First we create a vector called Ages, with minimum dimension of 1 and maximum dimension of nobs which happens to be 60 when read from the current dat file. Because we will be extracting this vector as a column from the vonBdata matrix its minimum and maximum index need to correspond to the minimum and maximum row index for that matrix.

We write a similar line to create the vector variable for observed lengths.

**Slide Code:**

```
vector  Ages (1,nobs)
vector  Lengths_obs(1,nobs)
```

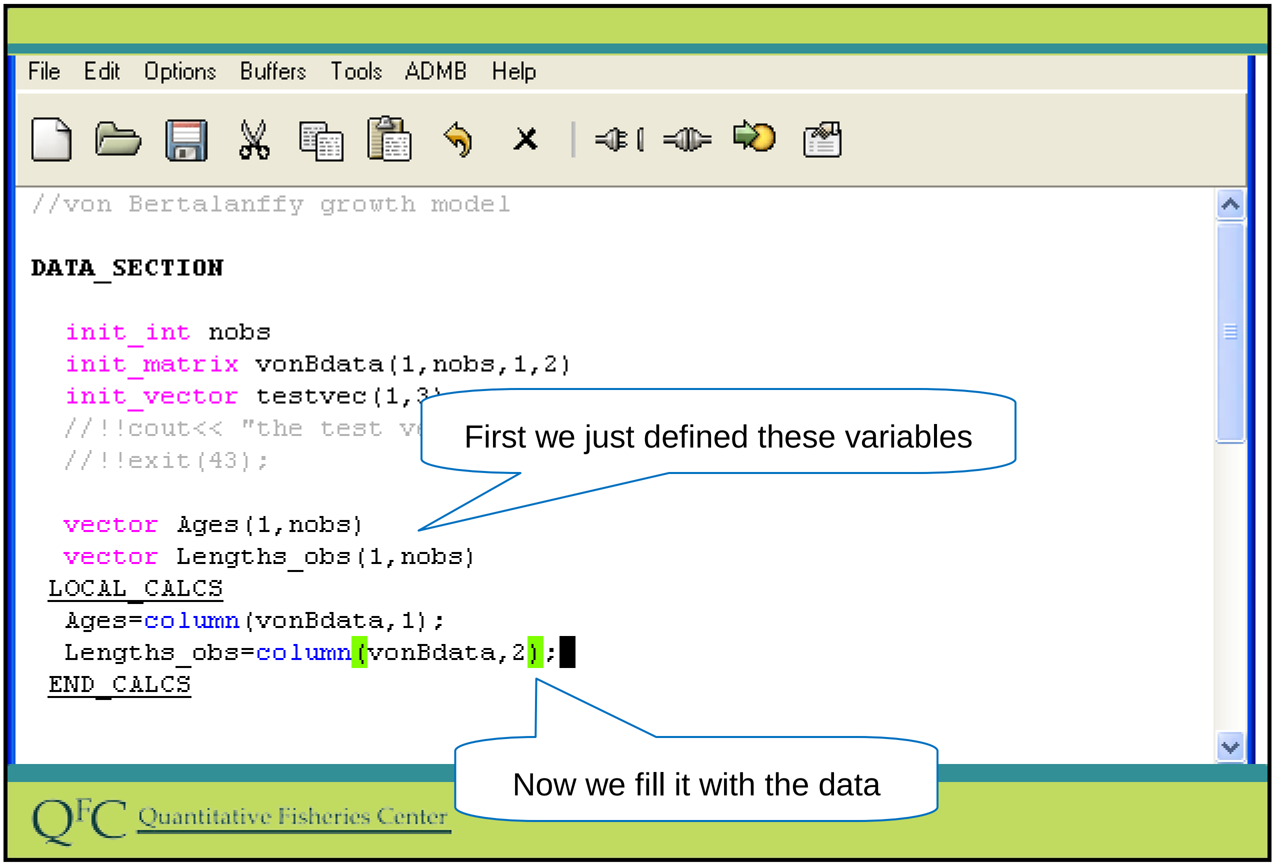



Next we write two lines that actually fill these new vector variables with the desired contents. C plus-plus, and admb which is based on it, is a highly structured programming language. What this means is you always need to define a variable before it gets used.

We introduce a new special way to block code in the data section when you have a sequence of lines you want to be interpreted as c plus-plus code with no translation. This is known as a local calcs block. You start the block with Local underscore calcs all capitalized and end it with end underscore calcs also all capitalized. Both lines are special lines that need to be indented exactly one space.

**Slide Code:**

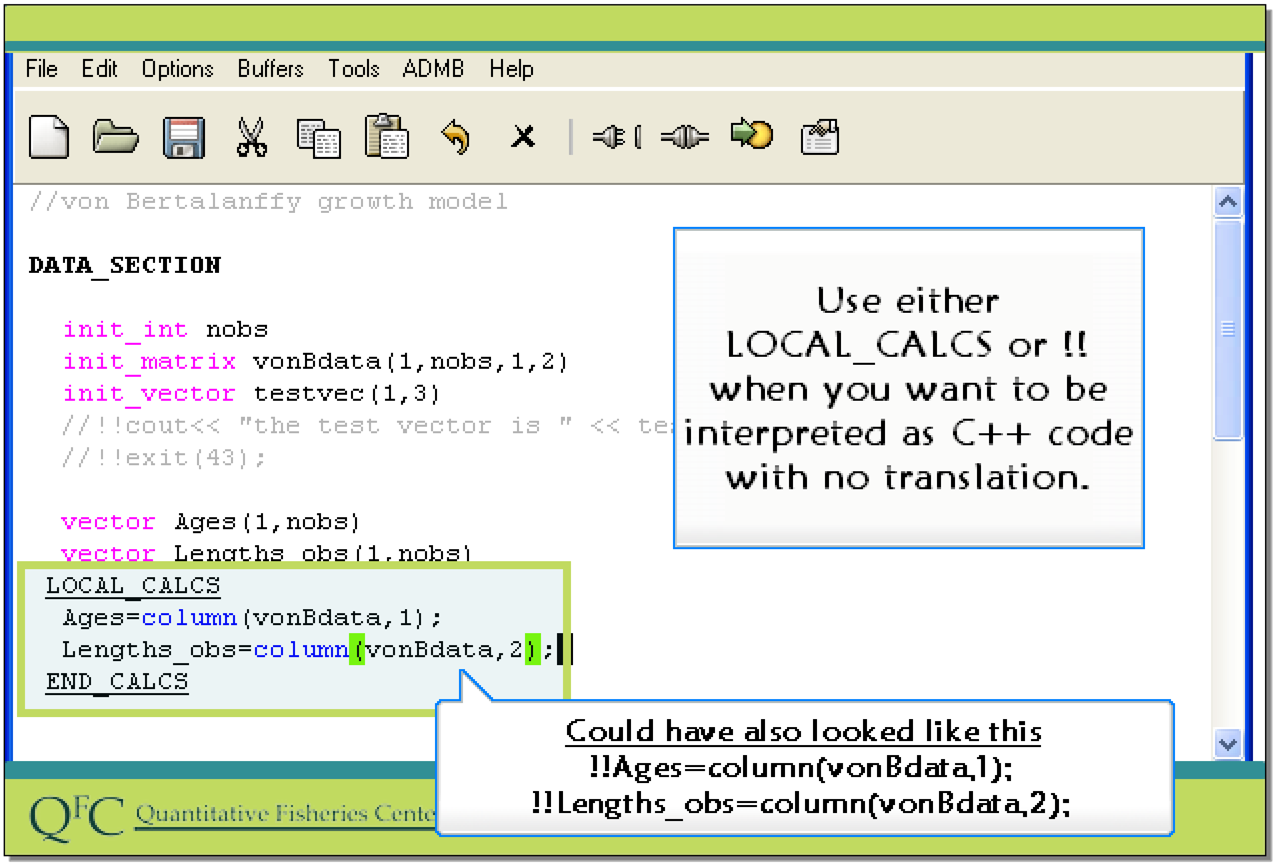
```
 LOCAL_CALCS


 END_CALCS
```



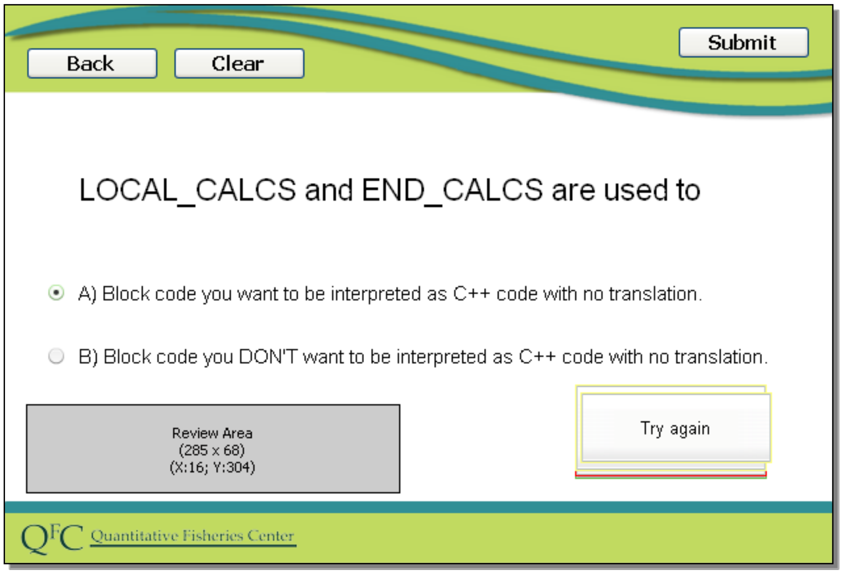

Between these lines go your lines of code. Here we add two lines: Ages=column(vonBdata,1); and Lengths_obs=column(vonBdata,2); Notice because these will be interpreted as c plus-plus code they need a semi-colon at the end. Each line uses the column function to extract that column so we can assign the contents of those columns to our new vector variables for ages and observed lengths. The column function takes two arguments. The first is the name of the matrix we will extract the column from and the second is the column index we want. Our second argument is 1 in the first line because the age data were in that column and its 2 in the second because the length data were in that column.

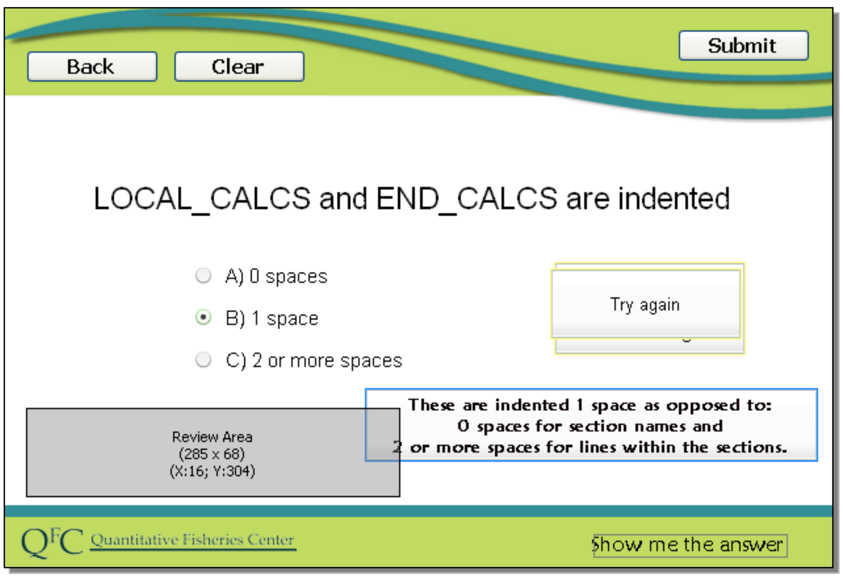
**Slide Code:**

```
Ages=column(vonBdata,1);
Lengths_obs=column(vonBdata,2);
```




We could have achieved the same result as we did with the local calcs block here by just putting two exclamation marks in front of each line within the block. In fact, for this case using the exclamation marks would actually save on typing. But sometimes you will want more than a couple of lines of c plus plus code in the data section and this is where using local calcs blocks can be very useful.


Back
Clear
Submit
LOCAL_CALCS and END_CALCS are used to
A) Block code you want to be interpreted as C++ code with no translation.
B) Block code you DON'T want to be interpreted as C++ code with no translation.
Review Area
(285 x 68)
(X:16; Y:304)
Try again
QFC Quantitative Fisheries Center

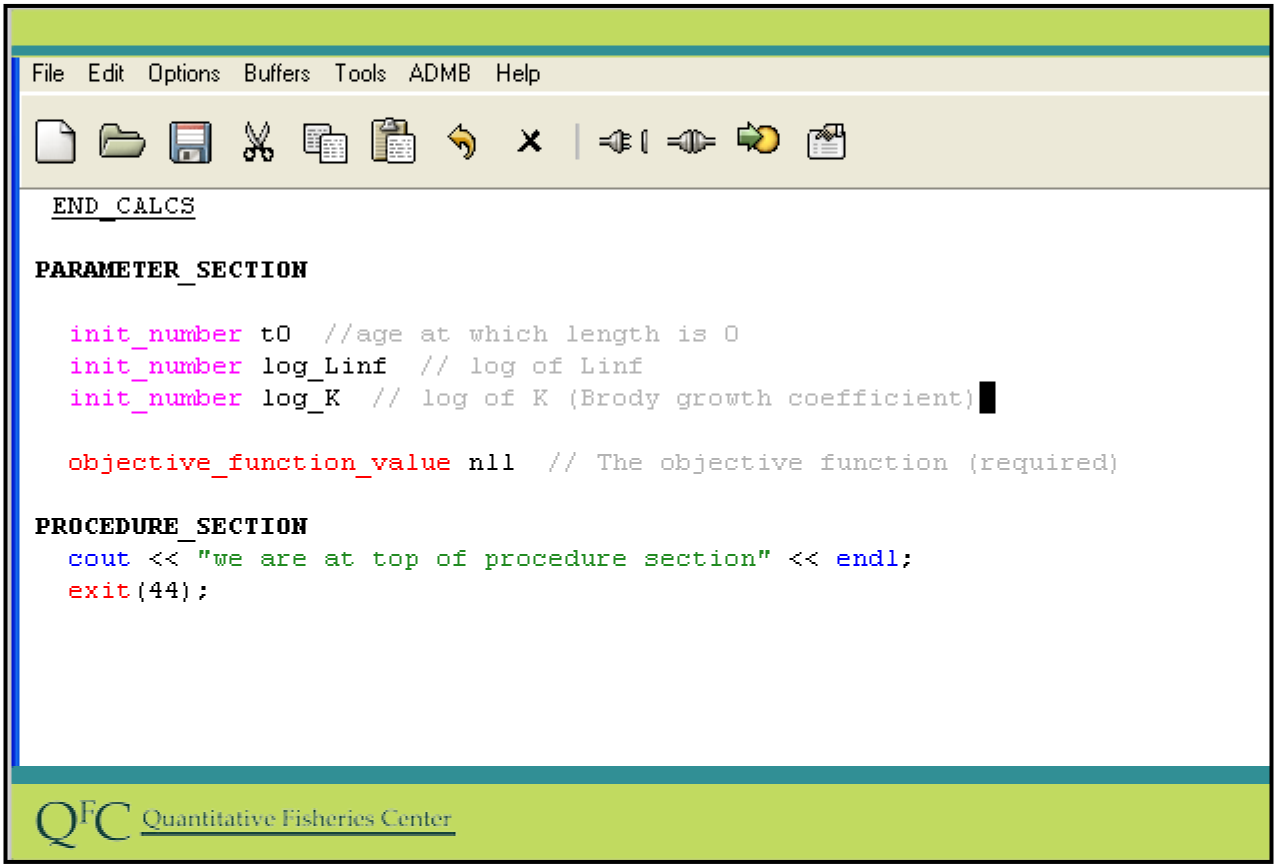

Back
Clear
Submit
LOCAL_CALCS and END_CALCS are indented
A) 0 spaces
B) 1 space
C) 2 or more spaces
Try again
Review Area
(285 x 68)
(X:16; Y:304)
These are indented 1 space as opposed to:
0 spaces for section names and
2 or more spaces for lines within the sections.
QFC Quantitative Fisheries Center
Show me the answer

# Parameter_Section

```
 END_CALCS

PARAMETER_SECTION

  init_number t0  //age at which length is 0

  objective_function_value nll  // The objective function (required)

PROCEDURE_SECTION
  cout << "we are at top of procedure section" << endl;
  exit(44);
```

We now complete the parameter section.

Previously we had defined one parameter, t0. This is actually one of the parameters for the model so we will keep it and add similar definitions for the other parameters we want to estimate. t naught can take negative or positive values so it is estimated directly as a parameter.

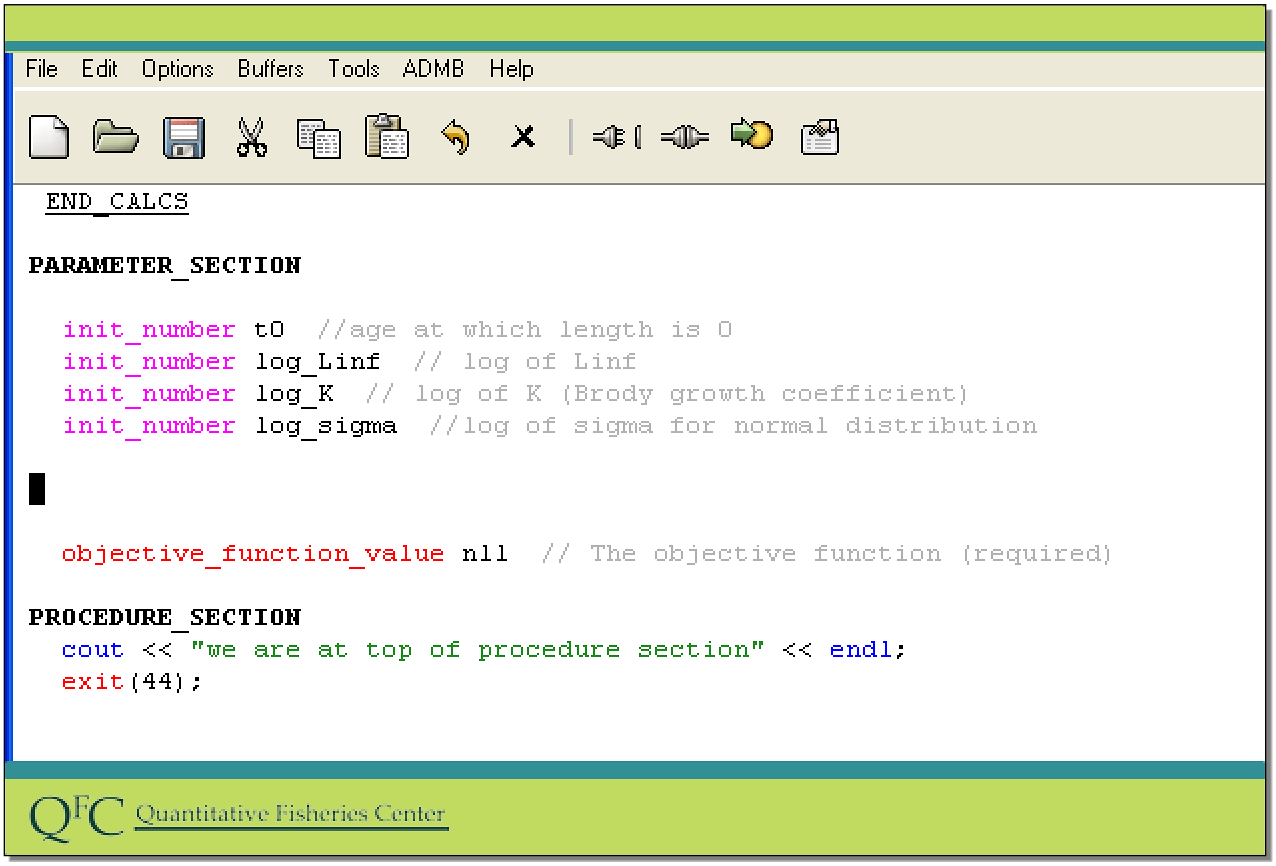



L-infinity and K, however can only be positive. During the search process, however, the search algorithm does not know these have to be positive and this can lead to problems. To avoid this we estimate the natural log of these parameters, which can take any real value and still correspond to positive values for L-infinity and K when back transformed. While estimating on log-scale can be helpful for quantities that on their original scale were restricted to be positive, this can cause serious problems if the original quantity should be able to take negative values. Thus you cannot automatically estimate all parameters on a log-scale. You need to think about what range of values is possible.

An added advantage of estimating parameters on the log-scale is it will tend to put the quantities being adjusted automatically during the fitting of your model on a more similar scale, which makes it easier for the automated searches to find the minimum of the objective function.

**Slide Code:**

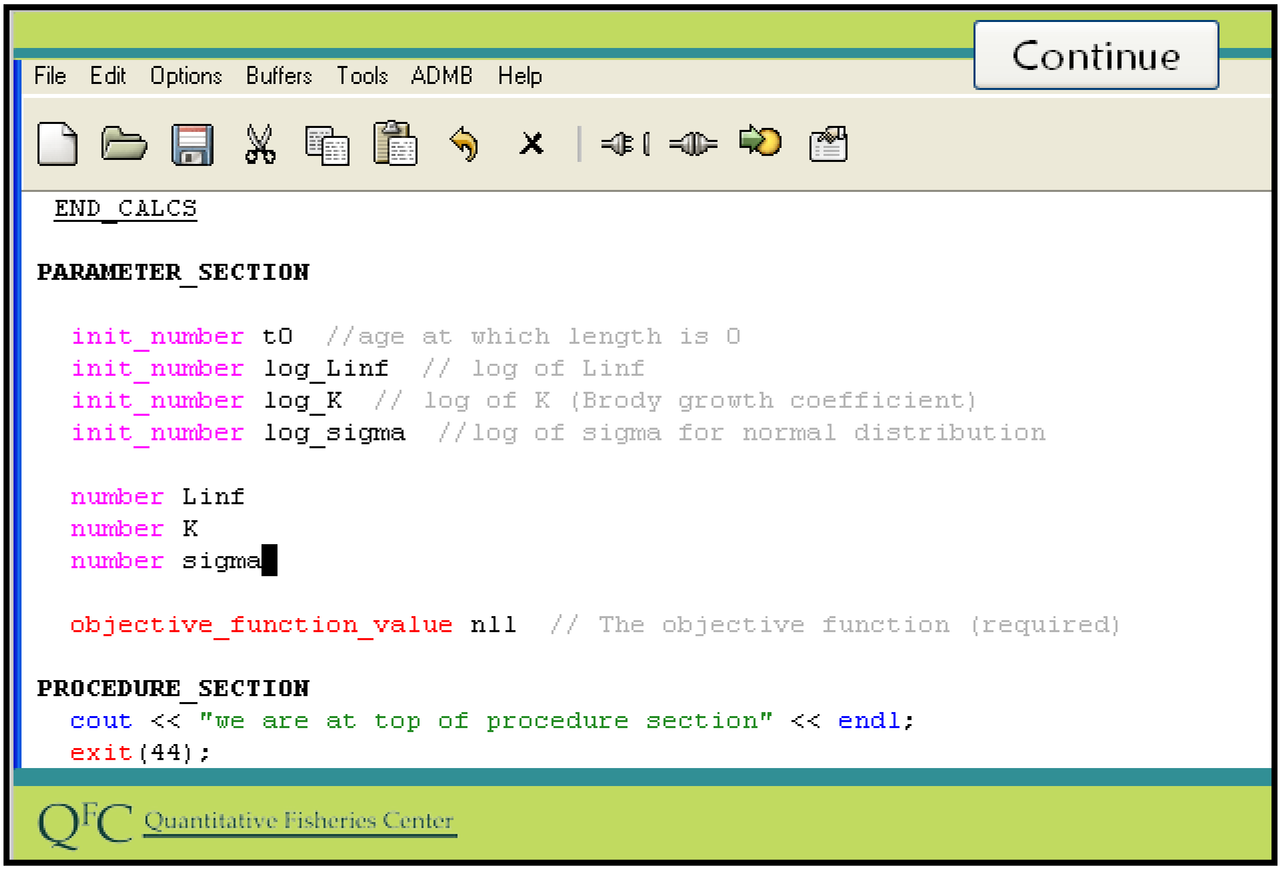
```
init_number log_Linf // log of Linf
init_number log_K // log of K (Brody growth coefficient)
```




Next we add the parameter for the variance. Here we estimate the log of sigma rather than sigma squared. Sigma rather than sigma squared is not on the same scale (with same units) as the data, which is a generally a good thing. As with L infinity and K it must be positive so we will estimate it on a log scale.

**Slide Code:**

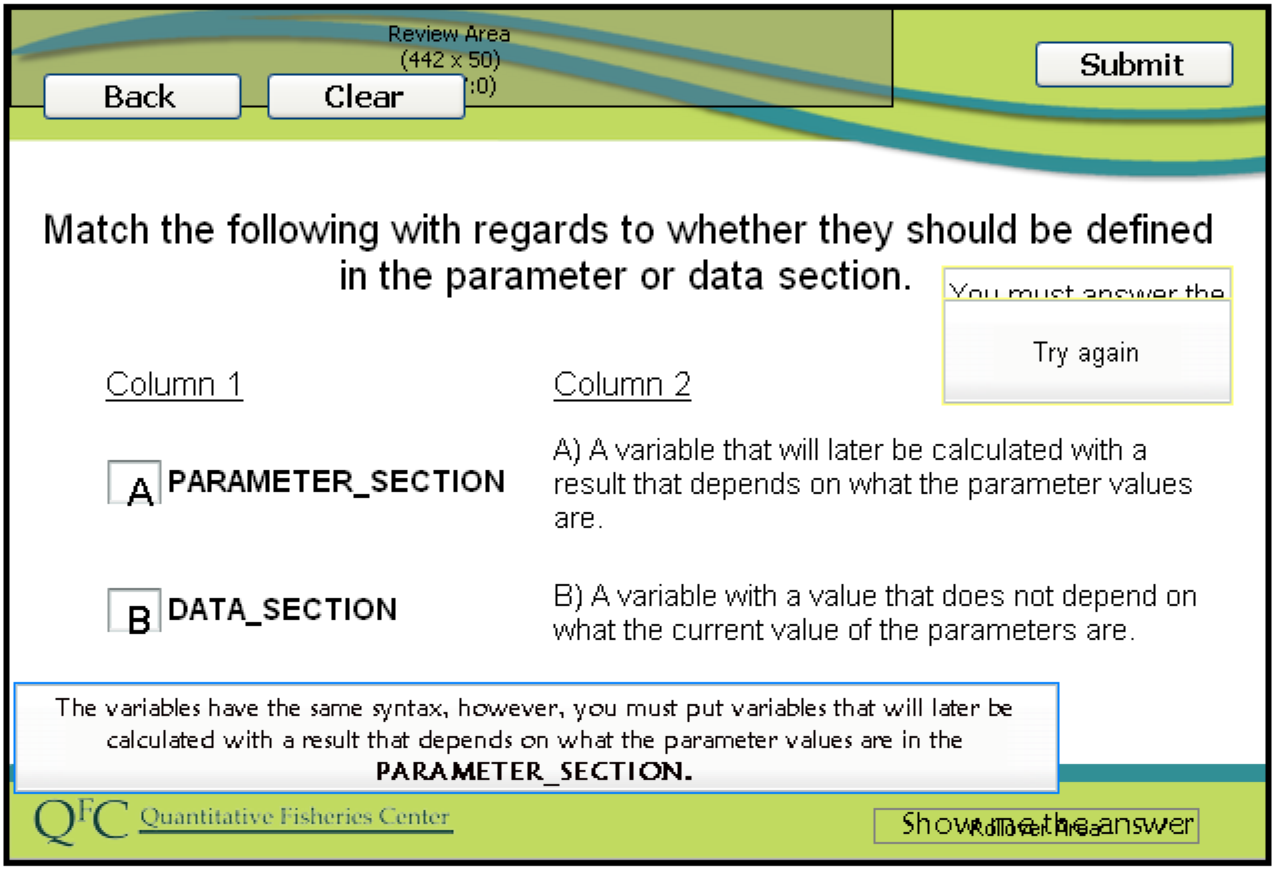
```
init_number  log_sigma  // log of sigma for normal distribution
```

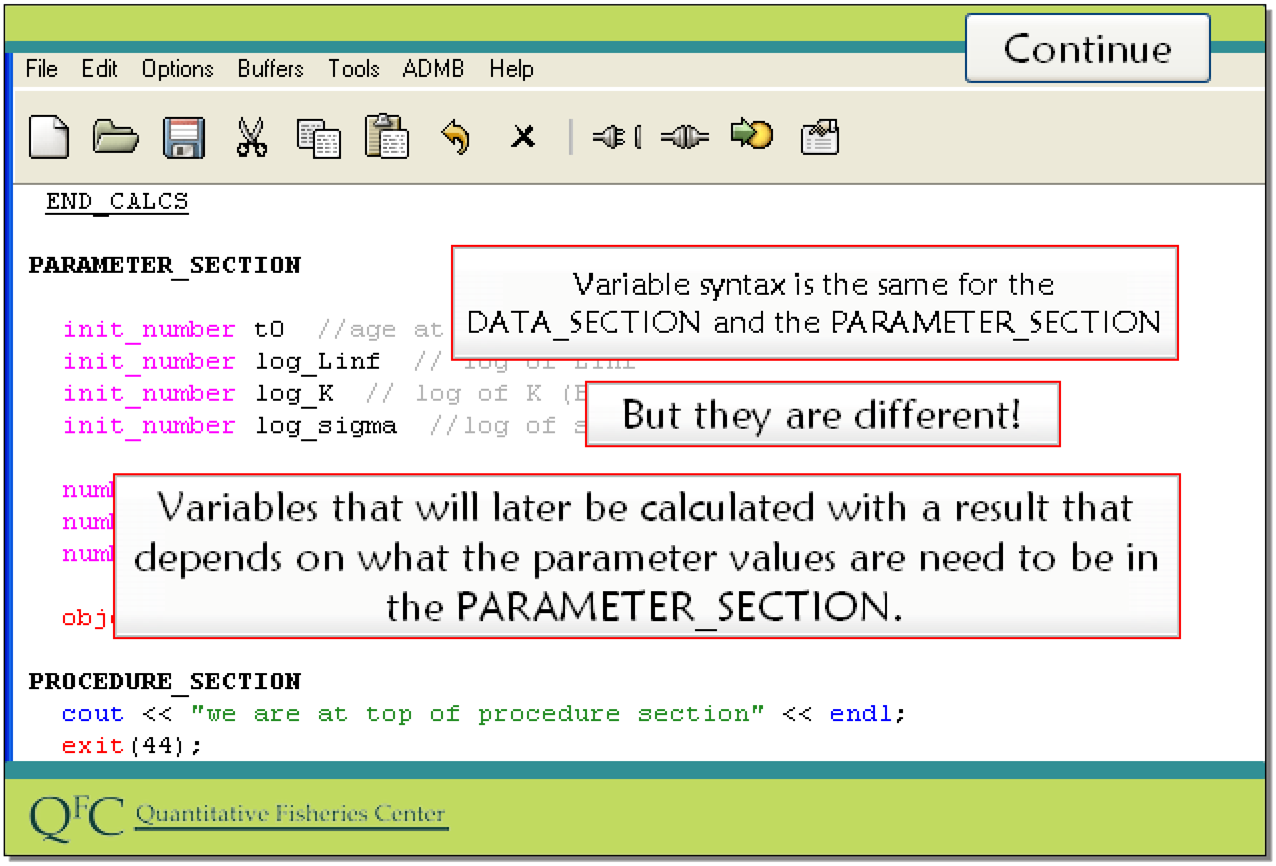



At this point we create variables to hold the back-transformed parameters we just created.

**Slide Code:**

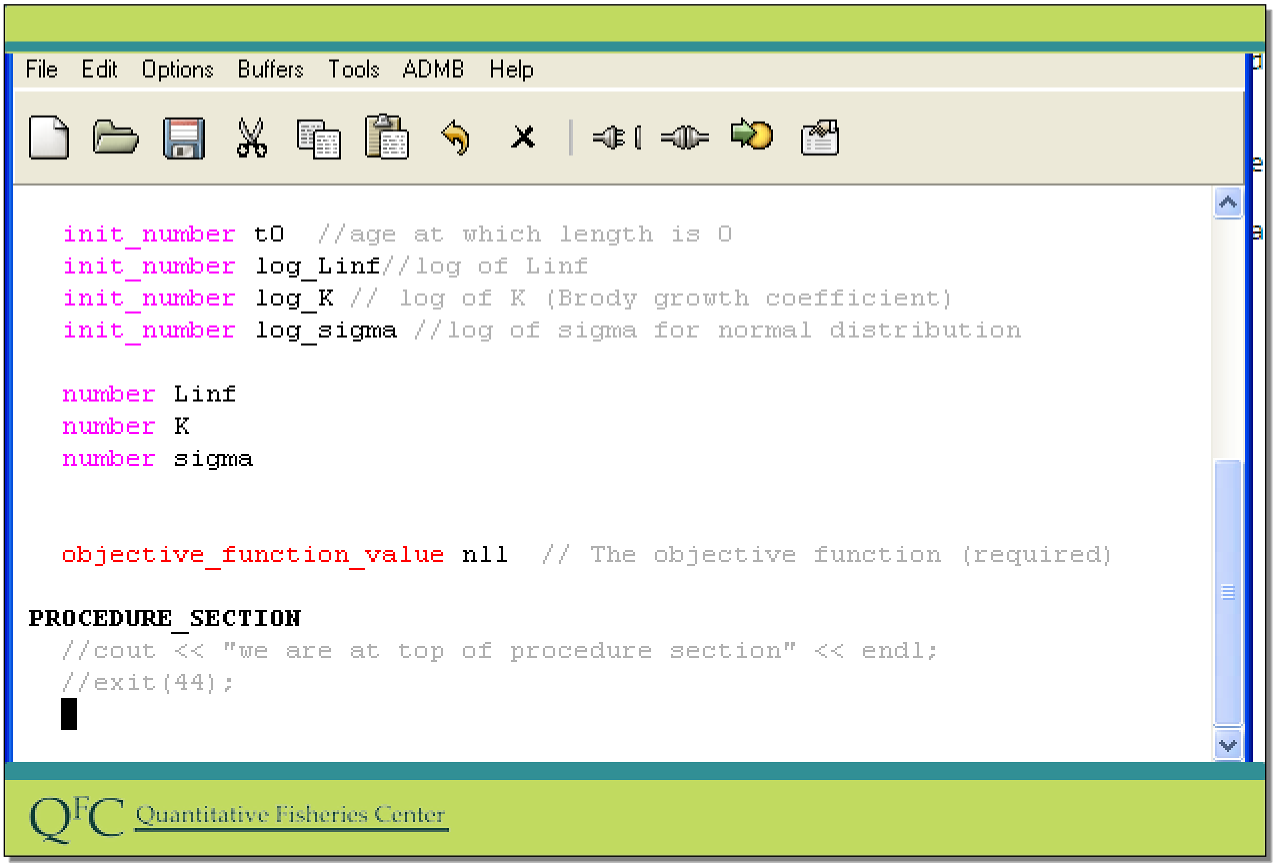
```
number  Linf
number K
number sigma
```


Review Area
(442 x 50)
:0)
Back
Clear
Submit
Match the following with regards to whether they should be defined in the parameter or data section.
Try again
Column 1
Column 2
A PARAMETER_SECTION
A) A variable that will later be calculated with a result that depends on what the parameter values are.
B DATA_SECTION
B) A variable with a value that does not depend on what the current value of the parameters are.
The variables have the same syntax, however, you must put variables that will later be calculated with a result that depends on what the parameter values are in the PARAMETER_SECTION.
QFC Quantitative Fisheries Center
Show me the answer
Rollover Area




A word of caution here. You could define variables with these names in the data section using the exact same syntax. However variables defined in the data section and the parameter section are actually of different types and this makes a difference as admb conducts its numerical search. The key thing to remember is that if you are defining a variable that will later be calculated with a result that depends on what the parameter values are, then it should be defined in the parameter section, not the data section. Only things that are going to stay constant should be defined in the data section.

# Procedure_Section

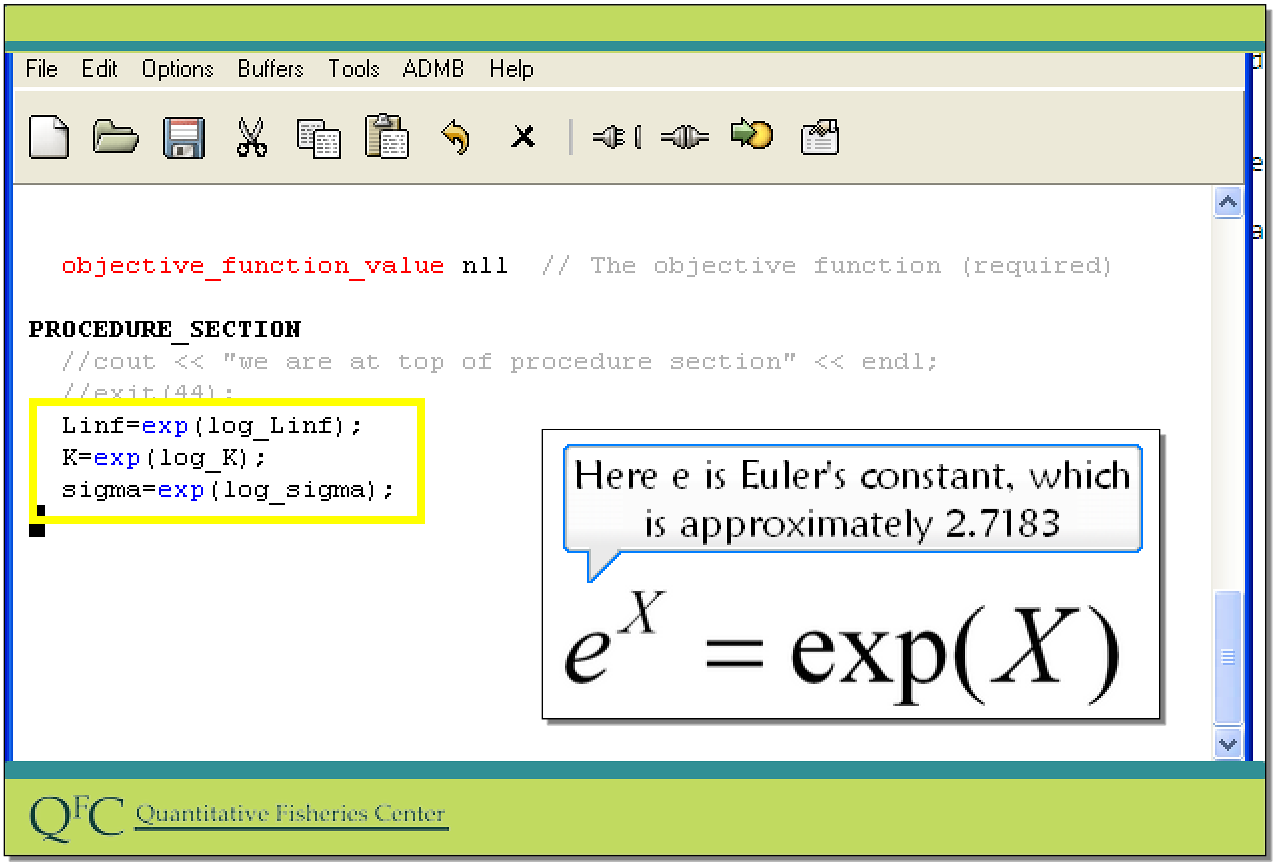



We now add some lines to the procedure section, but first we will comment out the old cout and exit lines.

**Slide Code:**

**// !!cout<<…………**

**// !!exit……………..**



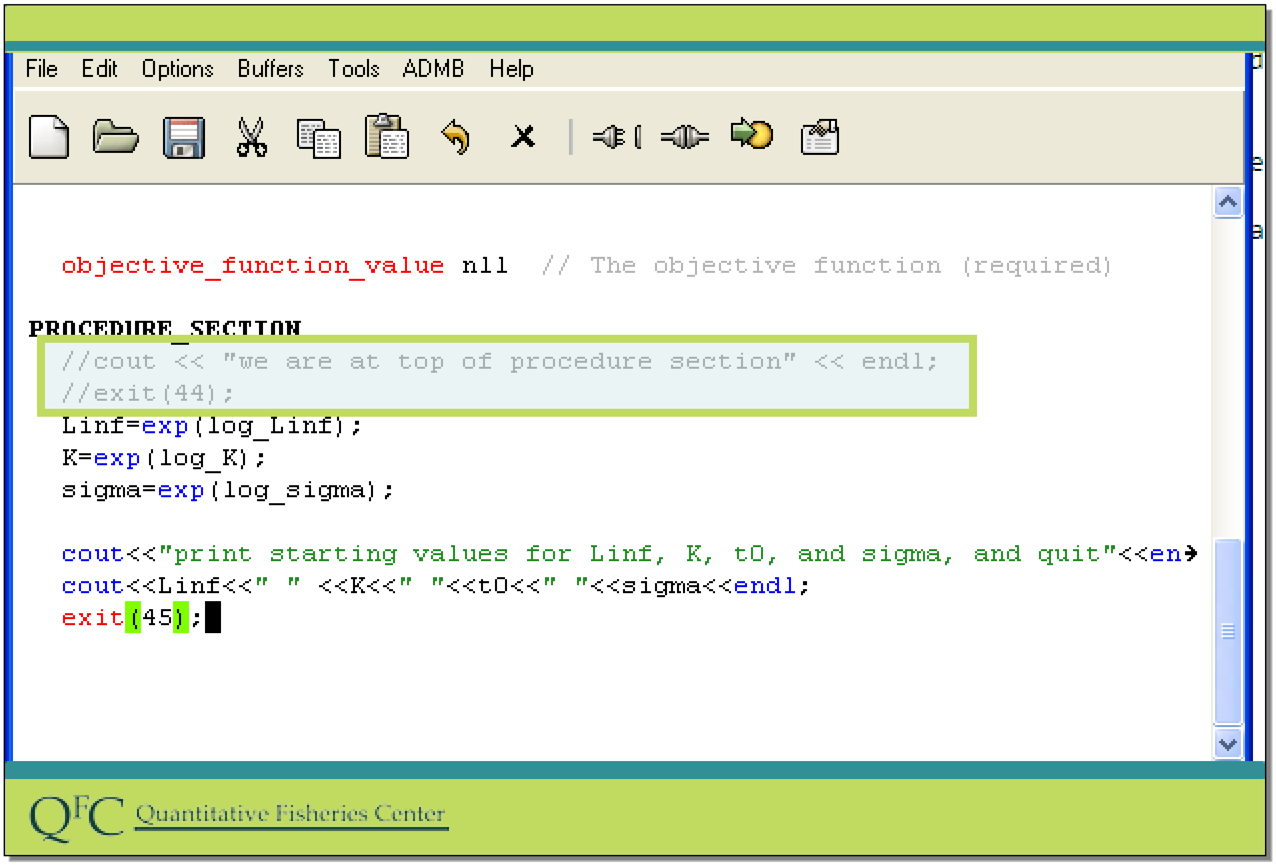

These lines are used to back-transform the parameters we just defined. Just as in many other software packages exp just means raise e to the power x.

**Slide Code:**

**Linf=exp (log_Linf);**

**K=exp (log_K);**

**sigma=exp (log_sigma);**

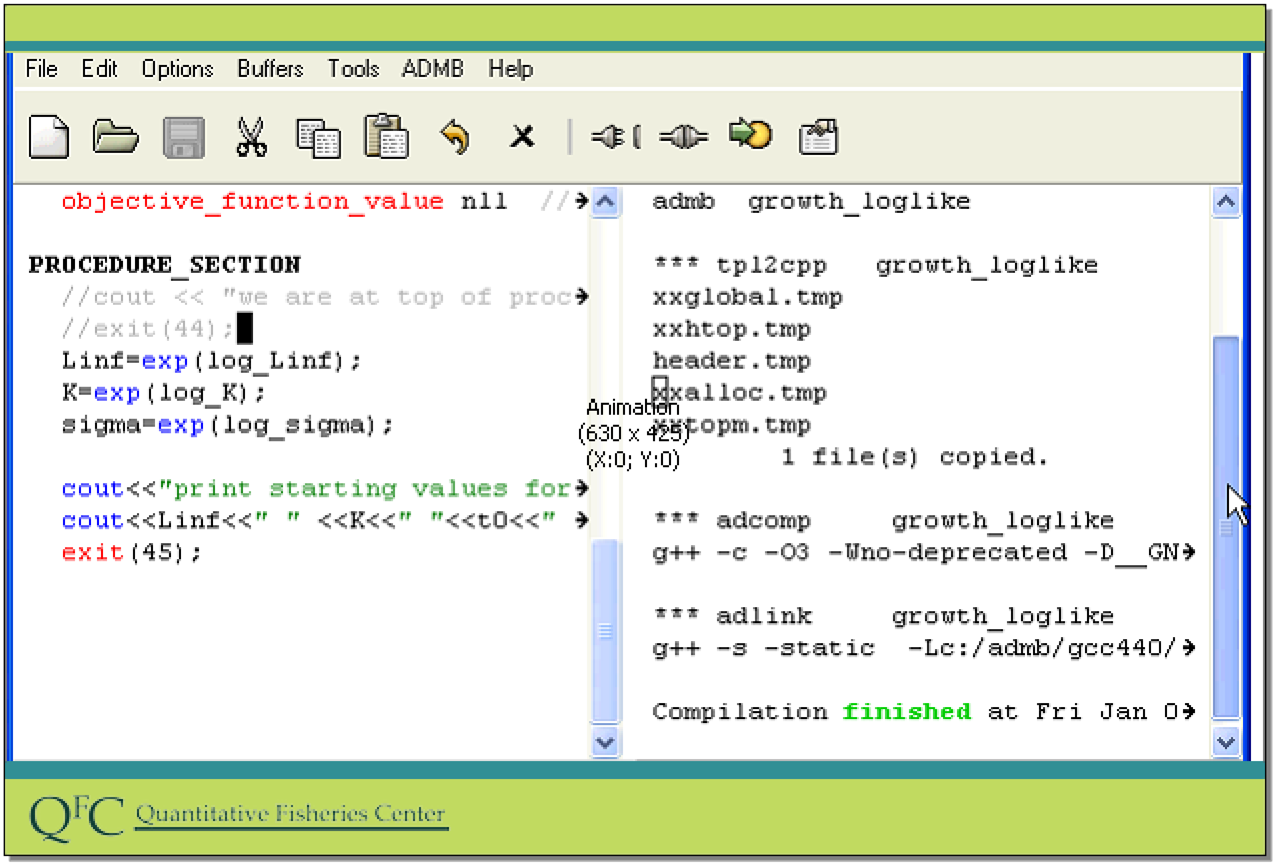



We are now ready to test whether we did this all correctly so we add some cout lines and an exit command.

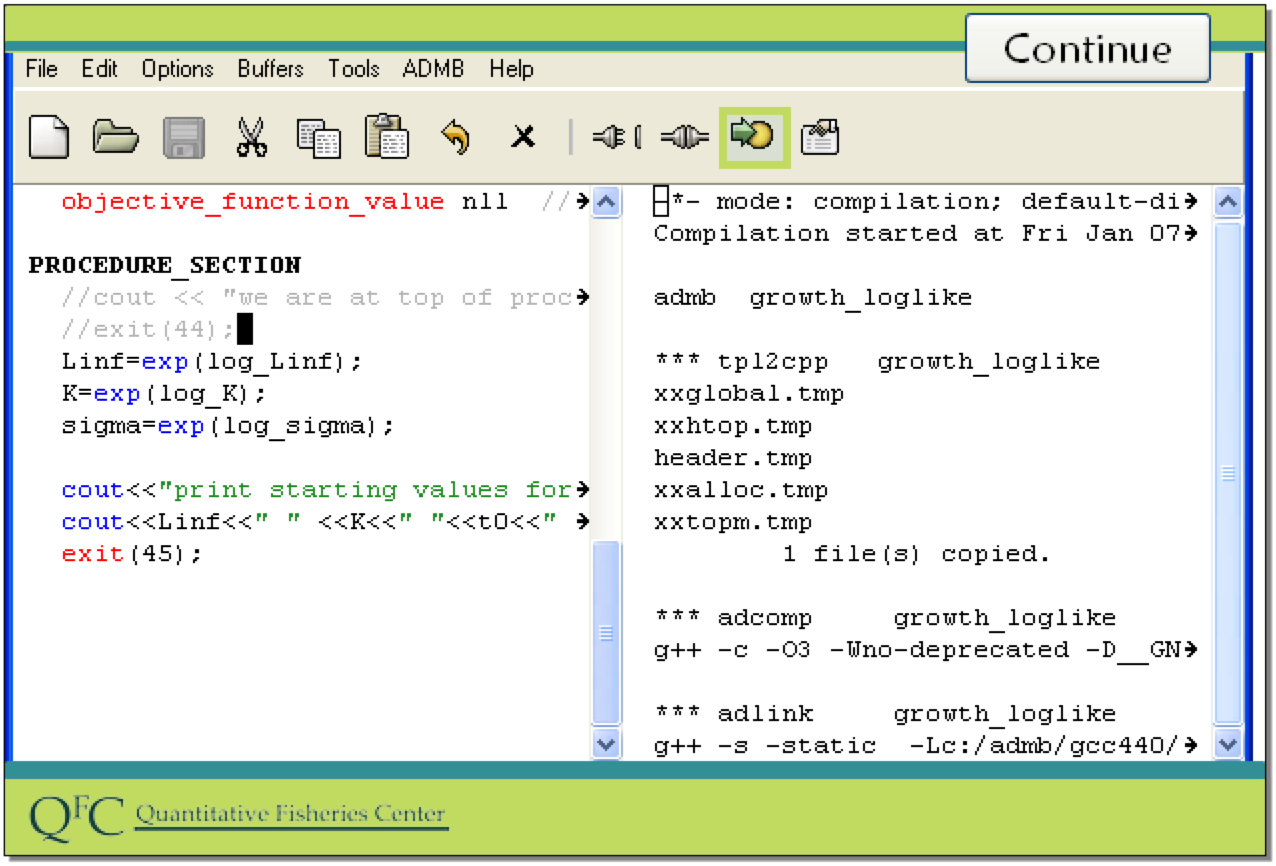
Our program would have never reached these commands if we hadn't commented out the earlier cout and exit commands. These have already served their function so they could have been deleted. By "commenting them out" they will not do anything, but it would be easy to turn them back on by removing the slashes. Commenting out is a very useful trick.

**Slide Code:**

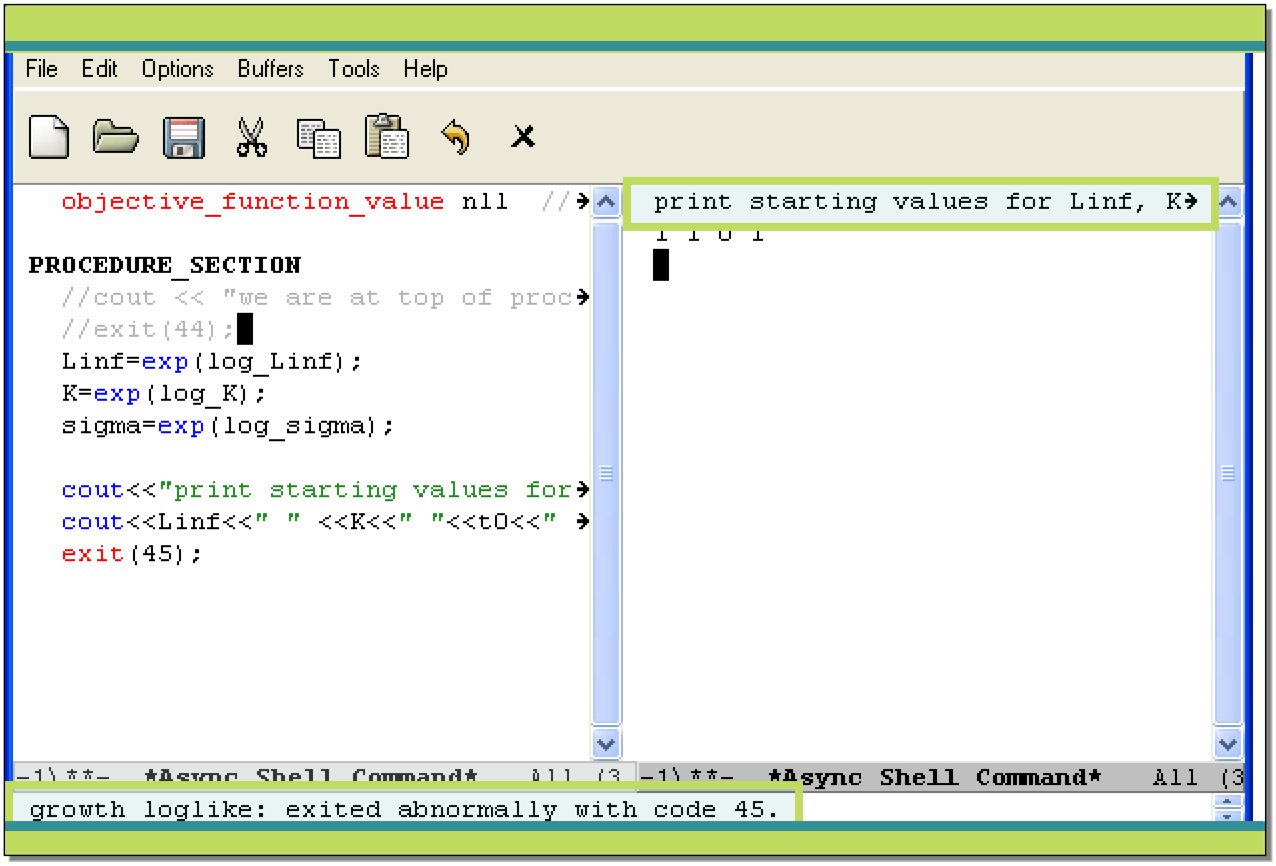
```
cout<<"print starting values for Linf, K, t0, and sigma, and quit"<<endl;   K=exp (log_K);
cout<<Linf<< “  “  <<K<< “  “<<t0<< “  “<<endl;
exit(45);
```



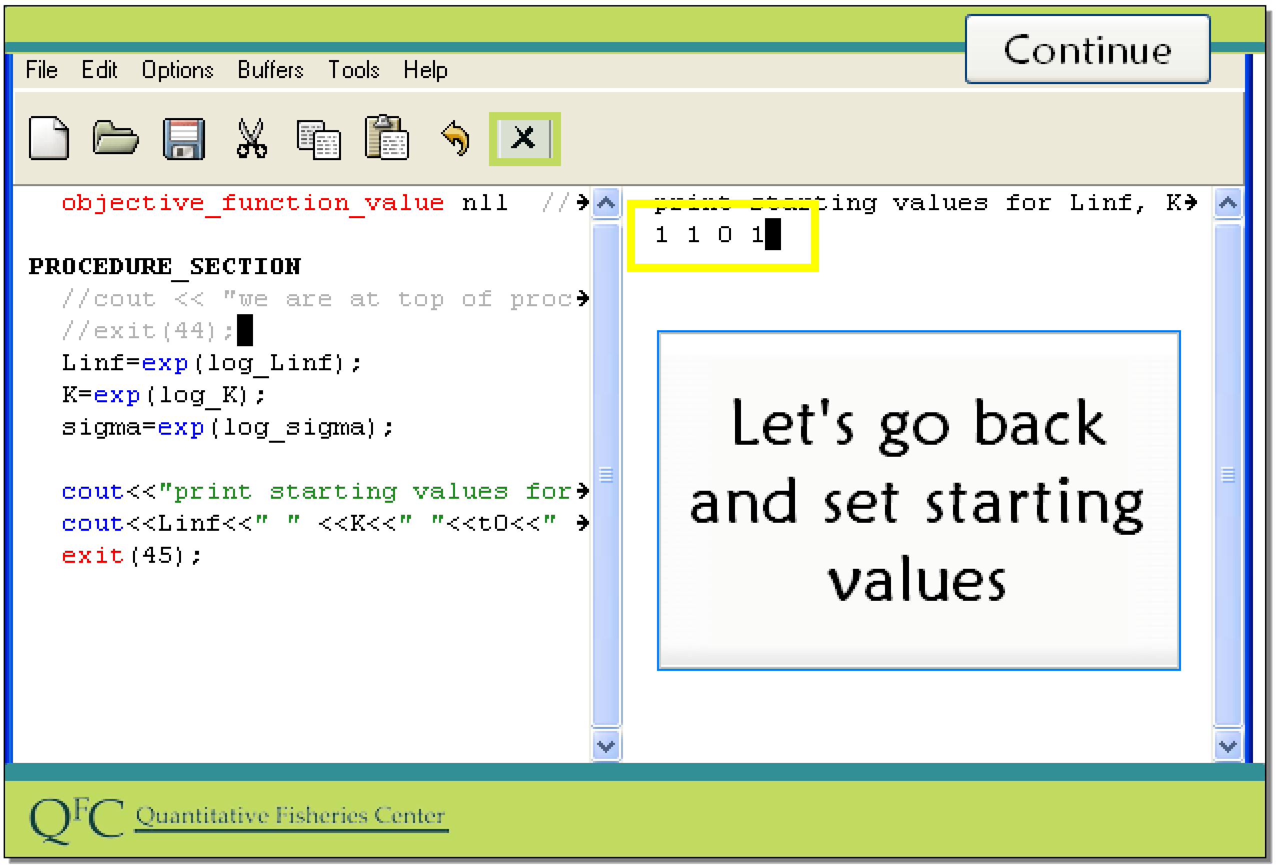

We are now ready to test the program. Again I build it.and check the output to make sure everything worked fine.


Continue
File Edit Options Buffers Tools ADMB Help
objective_function_value nll //
PROCEDURE_SECTION
//cout << "we are at top of proc
//exit(44);
Linf=exp(log_Linf);
K=exp(log_K);
sigma=exp(log_sigma);
cout<<"print starting values for
cout<<Linf<<" " <<K<<" "<<t0<<"
exit(45);
-*- mode: compilation; default-di
Compilation started at Fri Jan 07
admb growth_loglike
*** tpl2cpp growth_loglike
xxglobal.tmp
xxhtop.tmp
header.tmp
xxalloc.tmp
xxtopm.tmp
1 file(s) copied.
*** adcomp growth_loglike
g++ -c -O3 -Wno-deprecated -D__GN
*** adlink growth_loglike
g++ -s -static -Lc:/admb/gcc440/
QFC Quantitative Fisheries Center

Then I run it



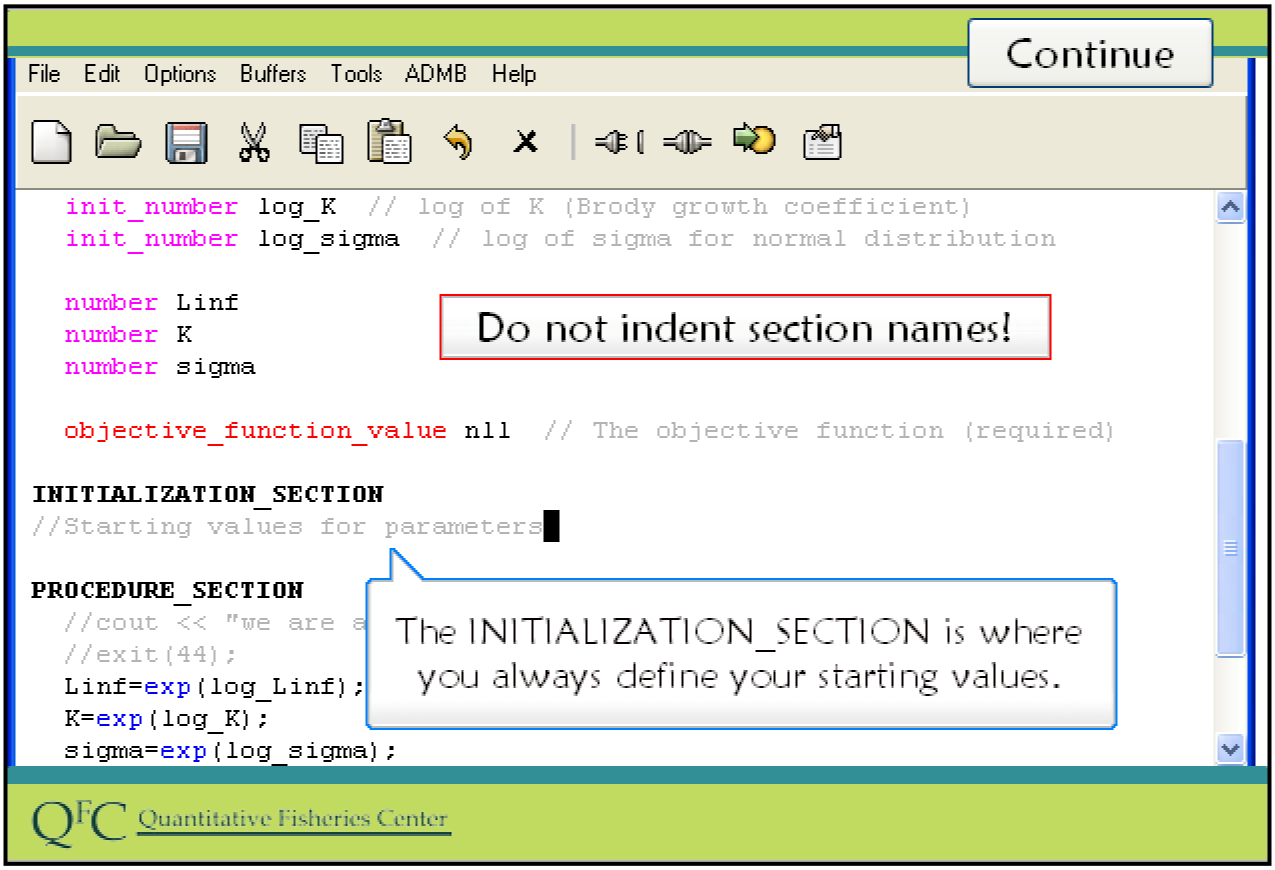

and indeed we get to the right place. Notice our cout message we previously wrote: "print starting values for Linf, K, t0 and sigma and quit."




But notice we have very odd values for Linf, K, t0 and sigma. Like many programs admb default starting values may not be very reasonable. For standard parameters without bounds admb uses zero, so t0 is zero and when we back transform zero for the other parameters we get 1.

So now we need to go back to our template and add some code to set starting values.

# Initialization_Section

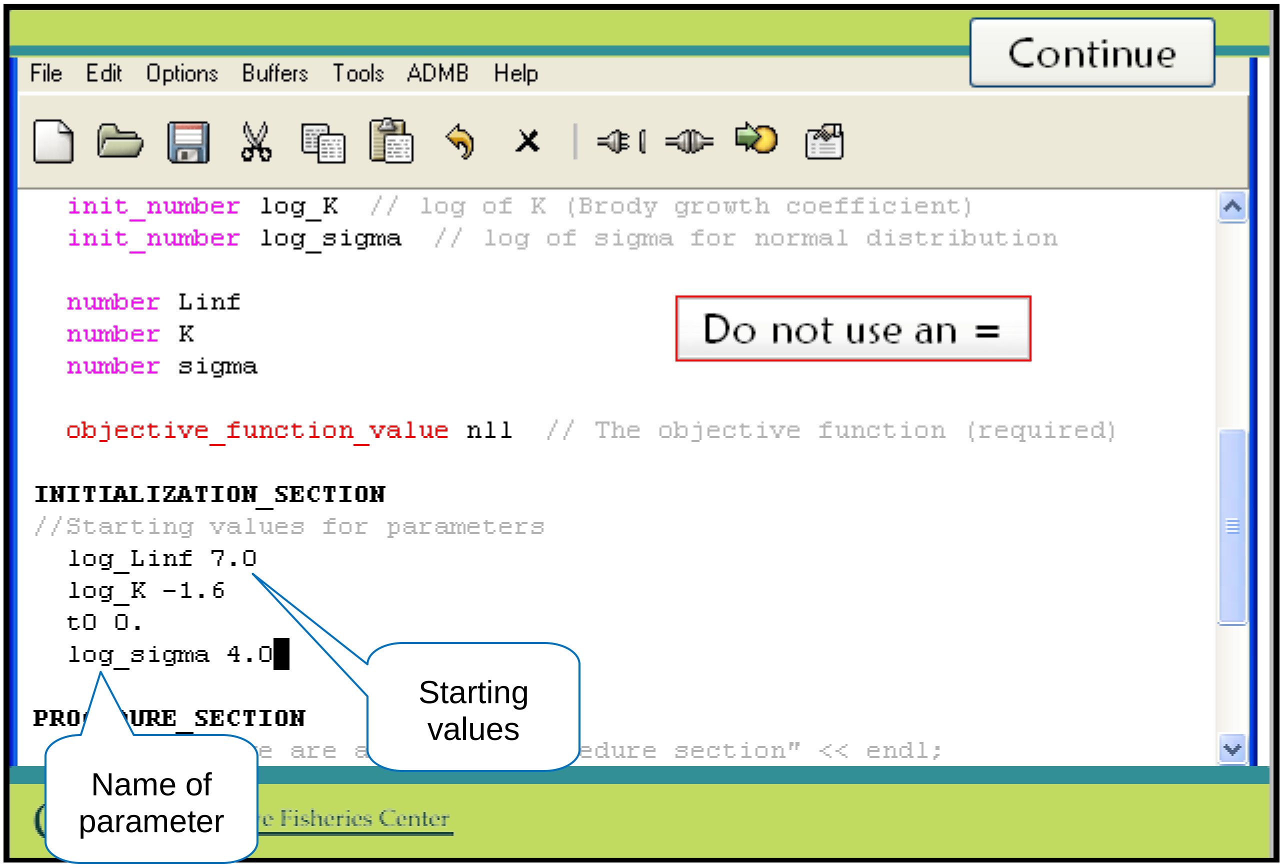



One simple way to set starting values in admb is by using a new initialization section for the starting values. Notice that it is not indented.

In this section, we will include starting values for parameters. We add a comment saying this but given this is what always goes in an initialization section once you get used to working with admb you probably will not include a comment like this.

**Slide Code:**

```
INITIALIZATION_SECTION
// Starting values for parameters
```

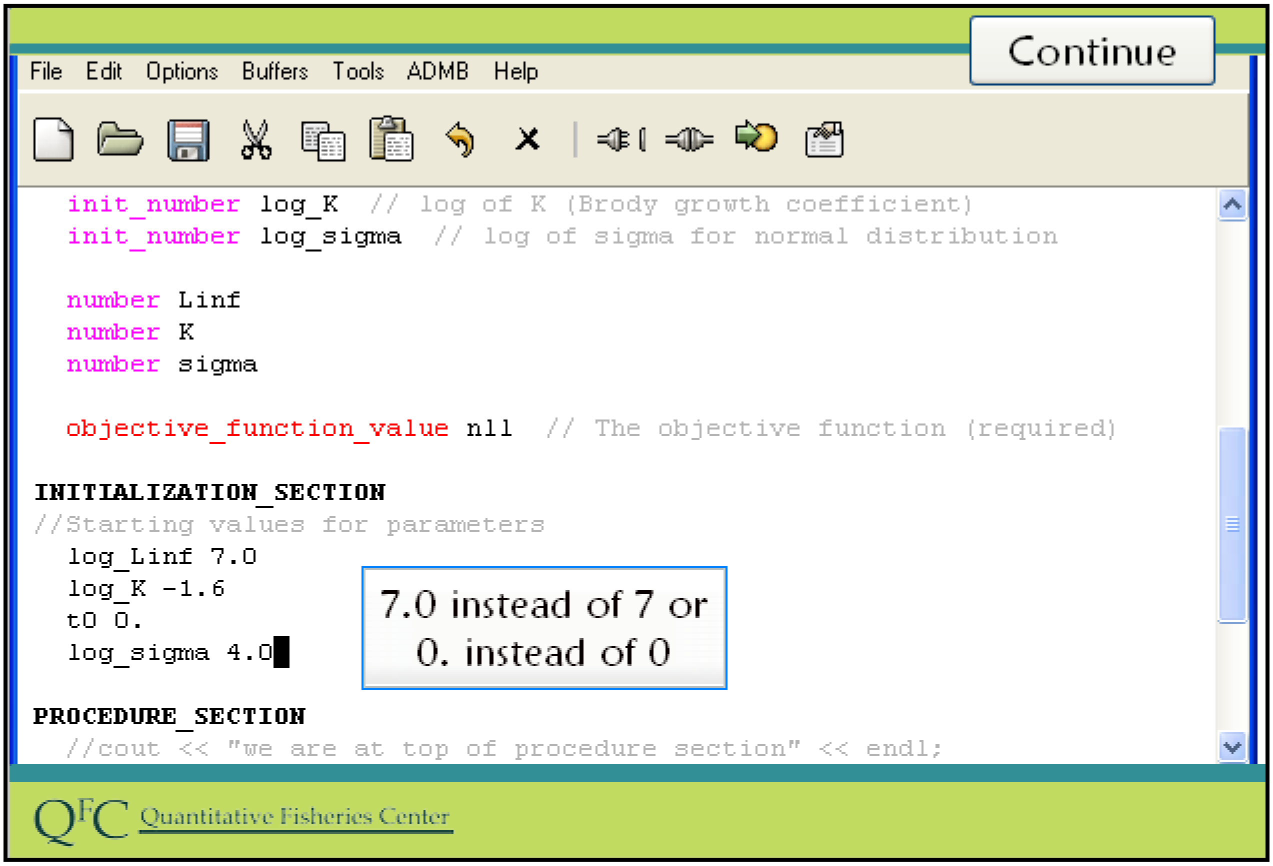



We now define starting values for each parameter in this section by giving the name of the parameter followed by its starting value.

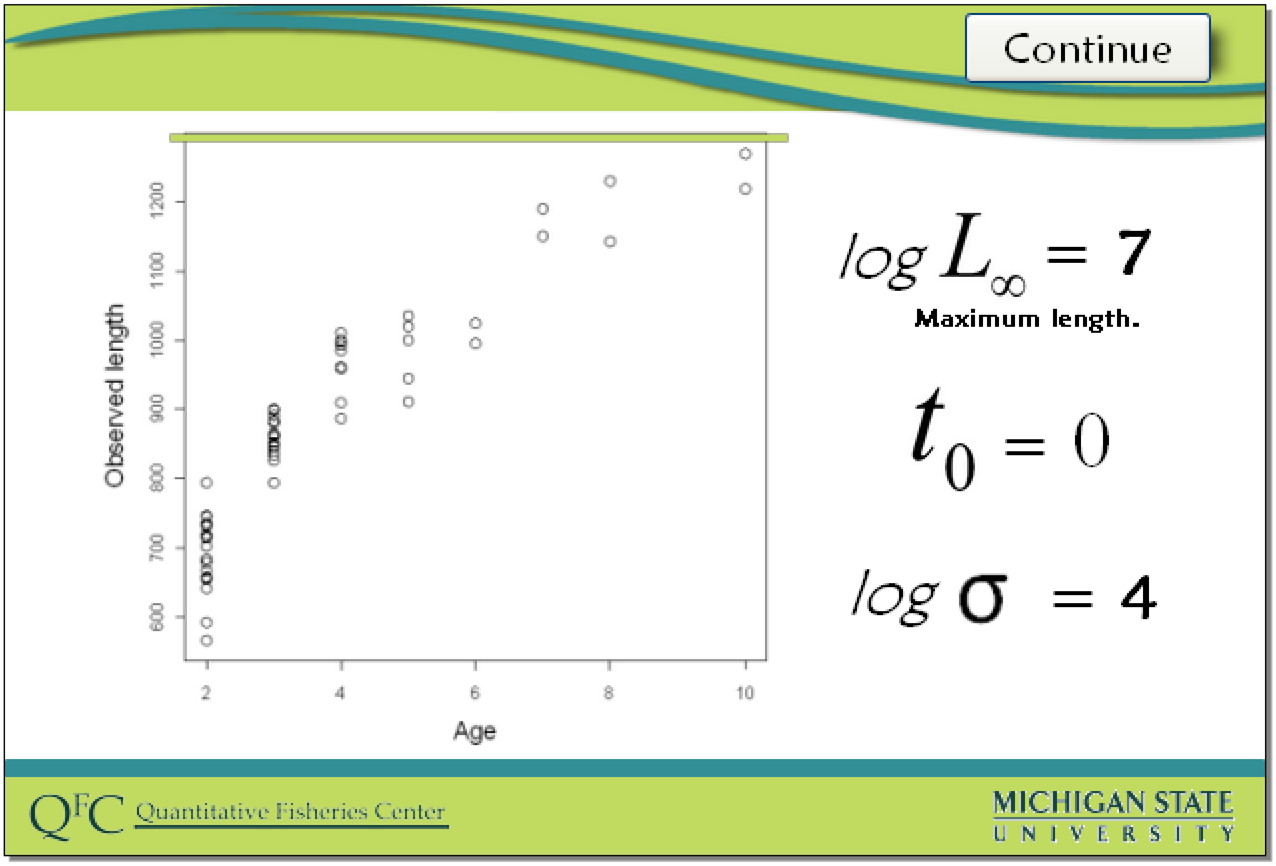
Notice that the parameter name is followed by a space and then the value with no equals sign. In just a moment we will discuss where these starting values came from. Notice that some of the starting values for parameters are negative. This is ok. As we said earlier, while it is not ok for L-infinity or K to be negative it is ok for the natural log of these parameters to be negative. Even if we started an estimated parameter as positive, during its search admb might try a negative value. By defining these estimated parameters as the natural log of L-infinity and K, whatever value they take during the search, when we later exponentiate them to get L-infinity and K the results will be positive.

**Slide Code:**

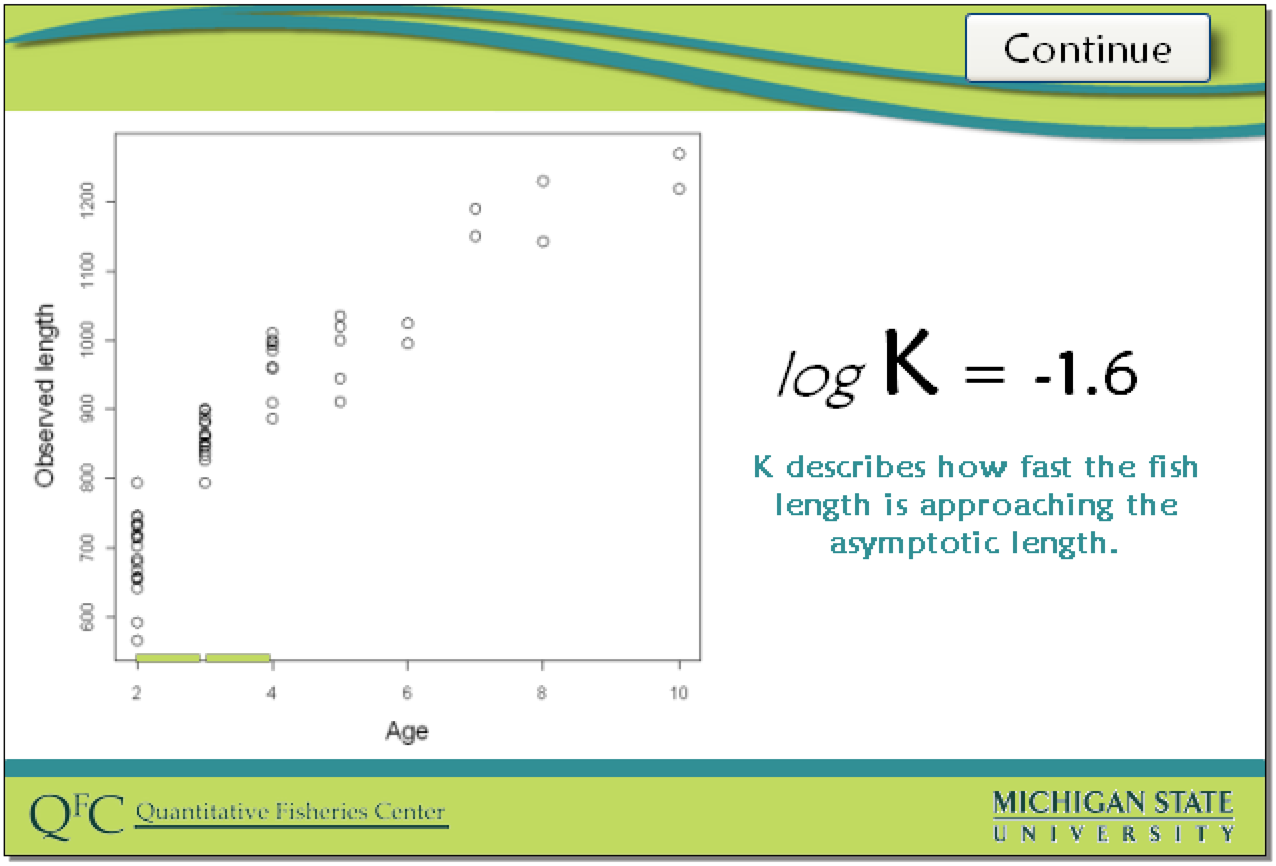
```
log_Linf   7.0
log_K  -1.6
t0   0.
log_sigma   4.0
```



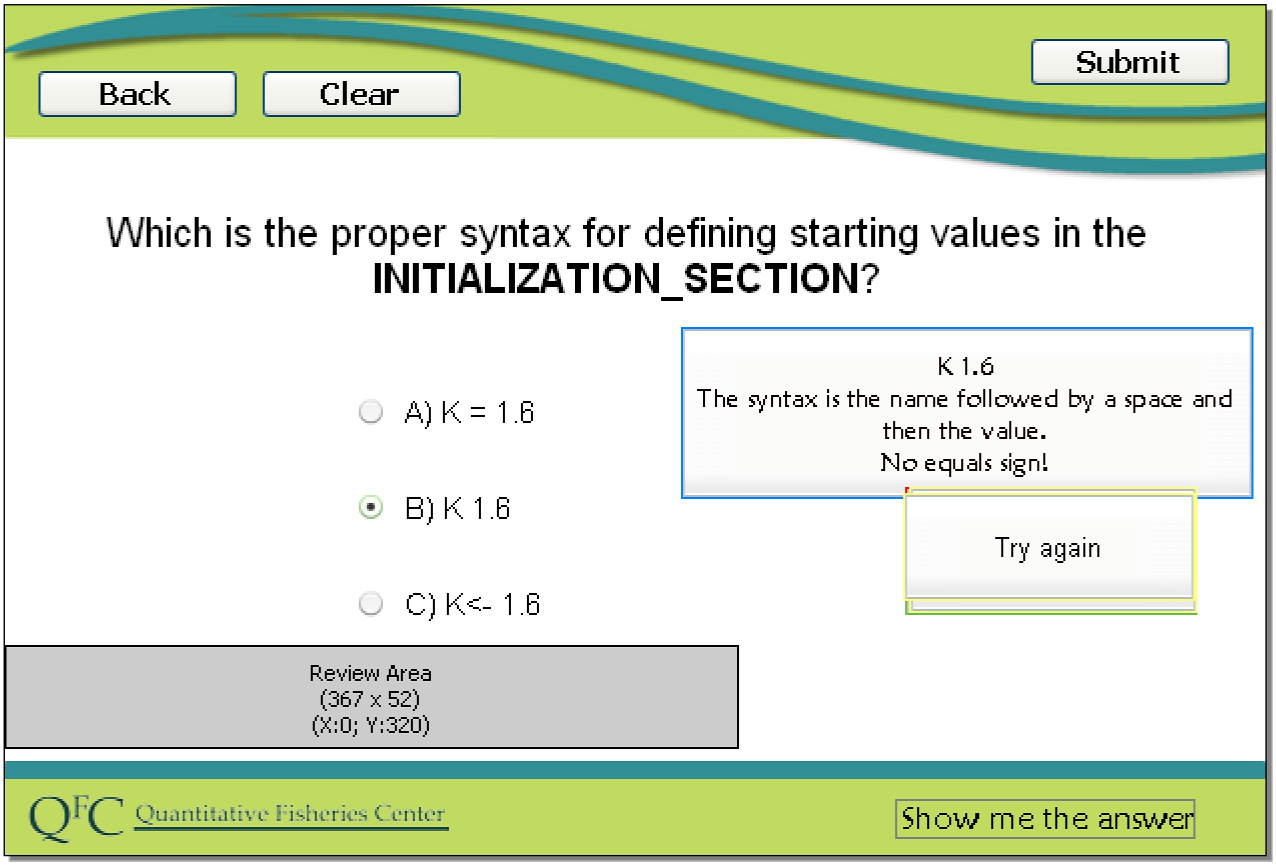

As a side note, notice that we use a decimal place with our numbers, e.g., 7.0 or 0. rather than 7 or 0. Here this just reflects a habit we have of generally trying to be very clear that we are working with a real number and not an integer. The code would work just without the decimal place. However using the decimal place can matter sometimes because the results of math with integers and math with real numbers can be different in C++, so its a good habit to use a decimal point with a number intended to be a real number.

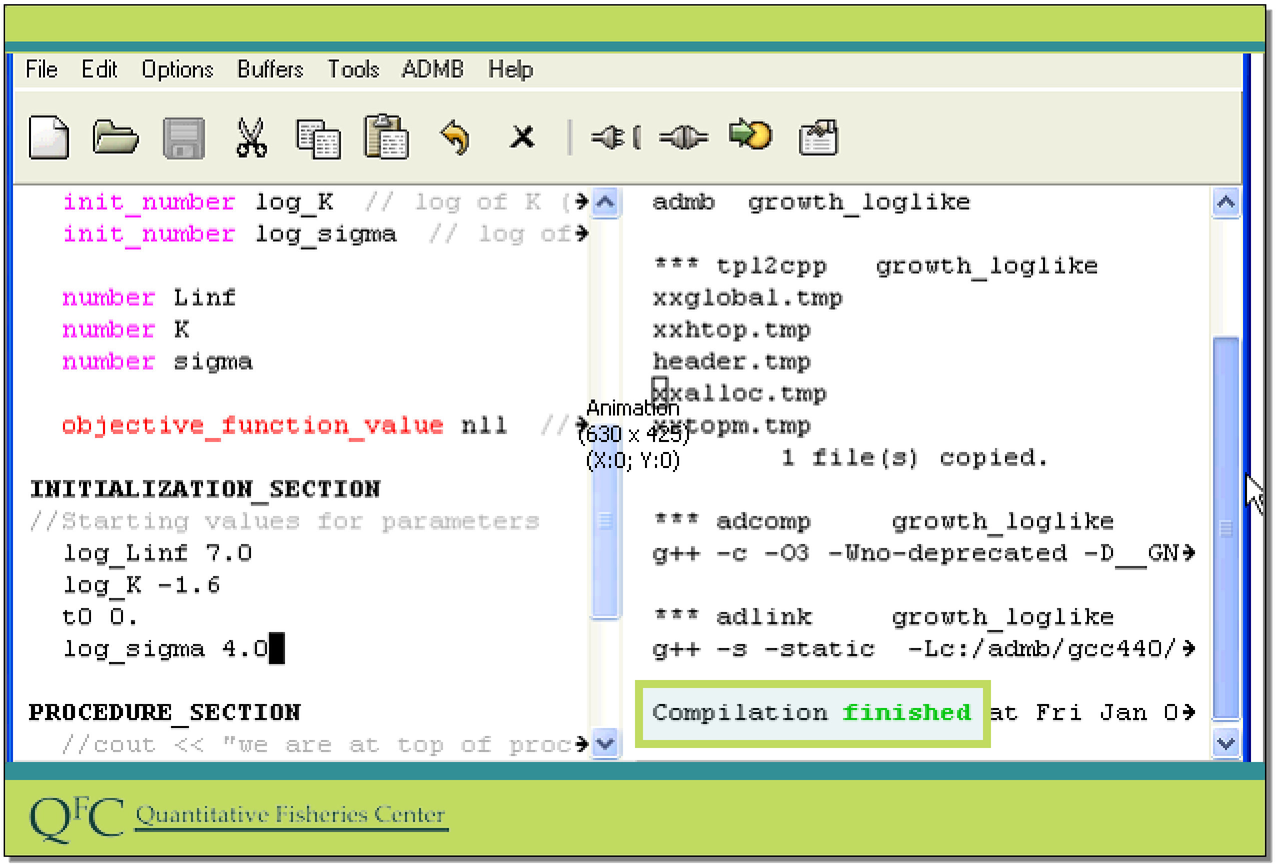



OK where did we get these starting values? Here we use a very simple rough and ready approach. If these videos were about the fitting the von Bertalanffy model in particular rather than about writing an admb program for the first time we might have gotten a little fancier. First L-infinity seems pretty easy to make a rough guess at. L infinity is the asymptotic length and it looks like length is heading toward about 1300. If we take the natural log of this and round we get a value of 7. While t0 really is a theoretical fitting constant and not the age we expect a fish to be length zero, using zero as a starting value is often reasonable. Recall that sigma-squared can be estimated as the average squared deviation about the expected value and thus sigma crudely would be around the magnitude of an average deviation. I just visually examined the graph and concluded that this would be something like 50. Taking the log and rounding off led to 4 as a starting value.

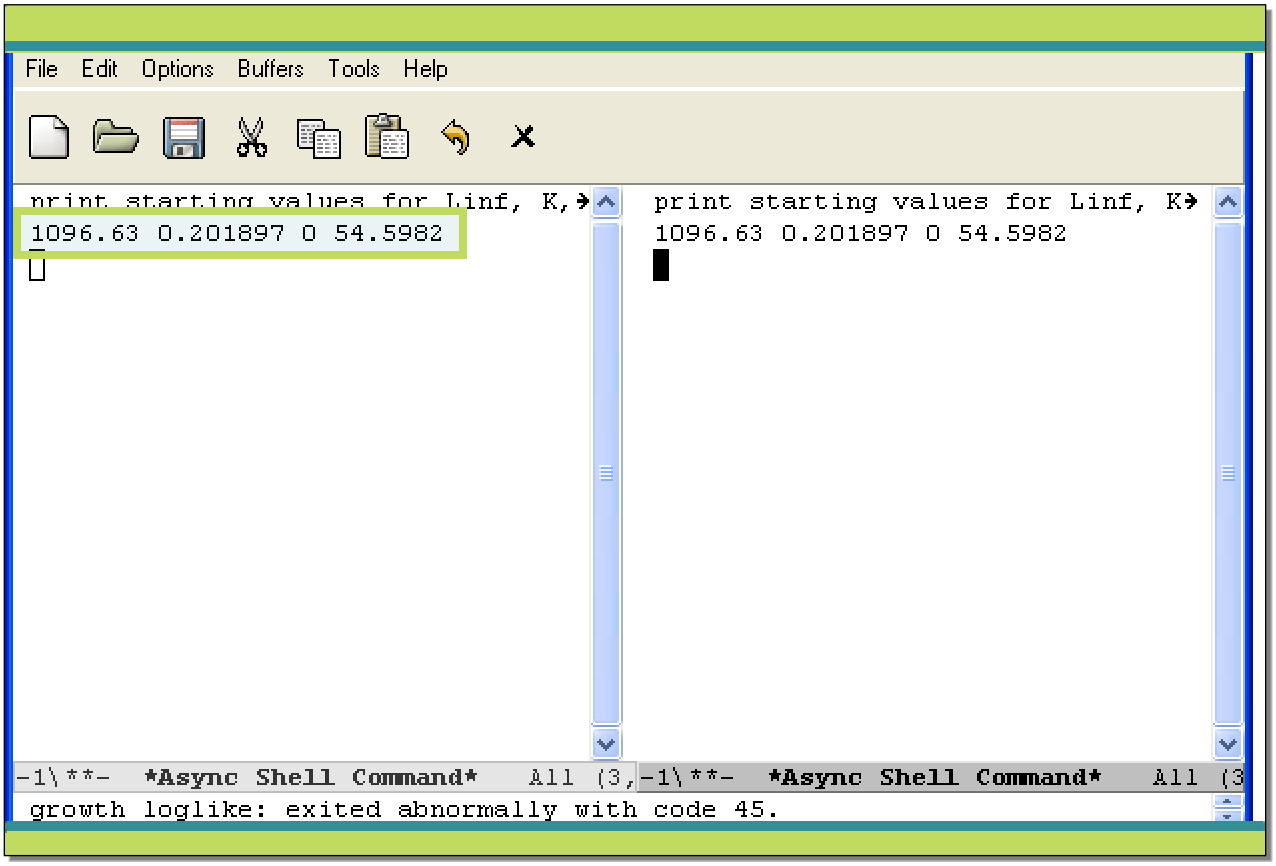



K is slightly more difficult to get a starting value for. A really simple way to get a starting value is to approximate increments for two adjacent pairs of ages say from ages 2 to 3 and from ages 3-4. I guesstimated 150 and 125 from the graph. Taking the ratio of the increment for the younger ages to the older ages and then taking the log of this provides a plausible starting value for K of 0.18. I rounded this to .2 and took the log to get -1.6. The procedure I used to get a starting value for K is based on various algebraic rearrangements of the von Bertalanffy model and an understanding of the meaning of the parameter of K. Here it is not important to understand the details of this particular model but it is worth keeping in mind that understanding the meaning of the parameters in your model and keeping these in mind can often help in specifying starting values.


Back
Clear
Submit
Which is the proper syntax for defining starting values in the INITIALIZATION_SECTION?
A) K = 1.6
B) K 1.6
C) K<- 1.6
K 1.6
The syntax is the name followed by a space and then the value.
No equals sign!
Try again
Review Area
(367 x 52)
(X:0; Y:320)
QFC Quantitative Fisheries Center
Show me the answer

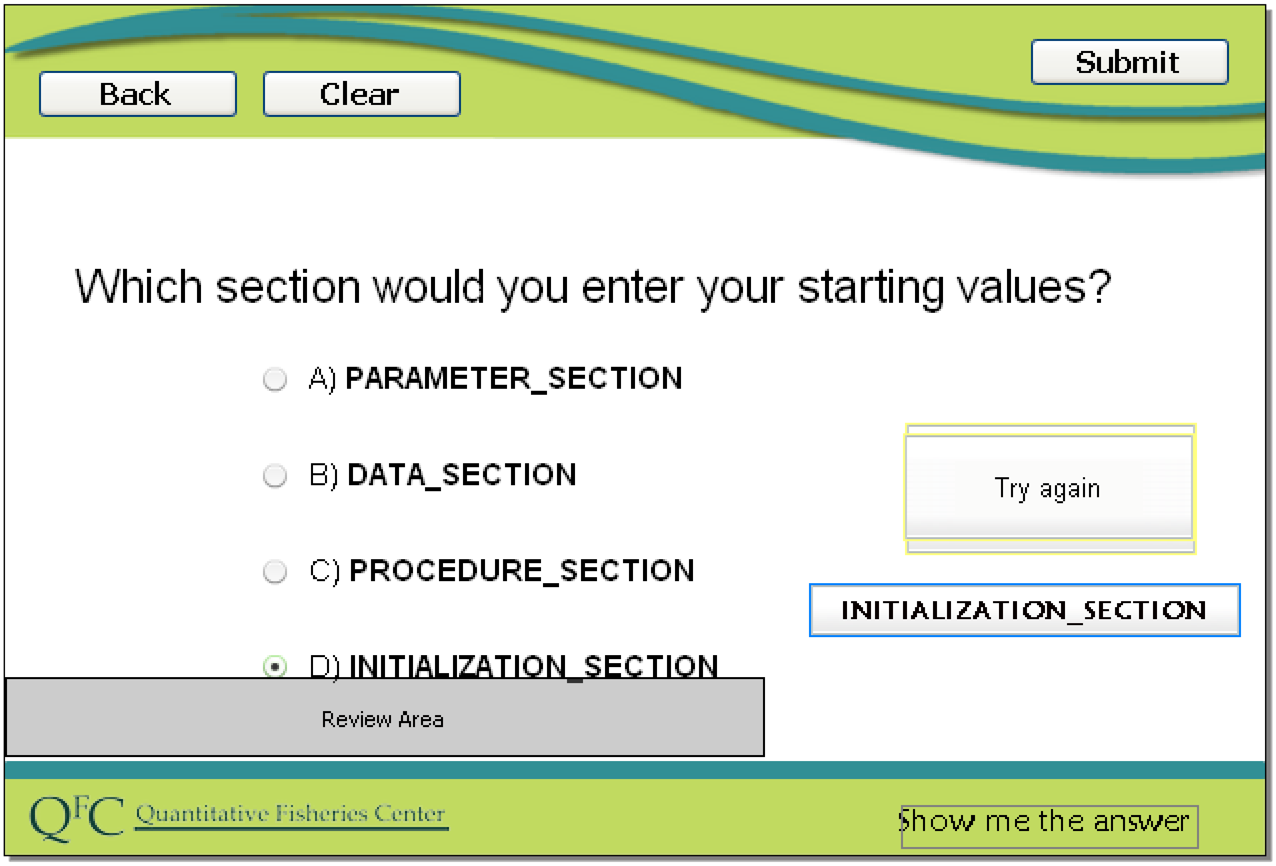



OK so we have starting values specified in a new section. Let's build and run our program again to test and see if things worked out right. It finished successfully.




Now let's run it to find our parameters.

Now that seems more reasonable for starting values. Notice that our rounding of starting values on a log-scale distorted them some and perhaps they are not the best starting values. However they are in the right ball park and we are hoping that the results are not sensitive to exactly what parameter values we started at.


Back
Clear
Submit
Which section would you enter your starting values?
A) PARAMETER_SECTION
B) DATA_SECTION
C) PROCEDURE_SECTION
D) INITIALIZATION_SECTION
Try again
INITIALIZATION_SECTION
Review Area
QFC Quantitative Fisheries Center
Show me the answer


This ends the third video. We now have completed a data section, defined all our parameters, given them reasonable starting values, and back transformed them at the start of the procedure section. We are now ready to actually write our procedure for generating predicted values and the negative log-likelihood in the next video.