



Next

# Safe Mode

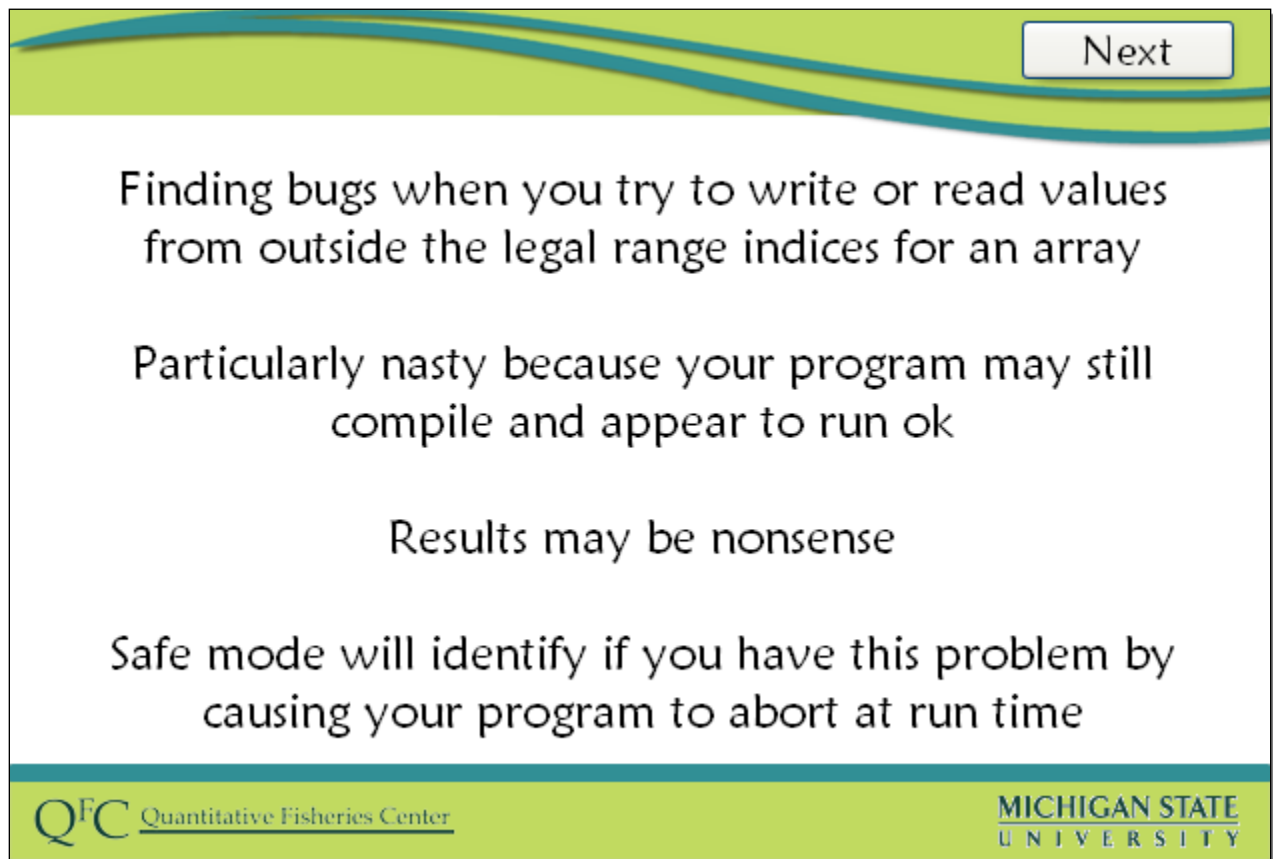


This video was created using ADMB-IDE release 4.5.0-1 (July 15, 2011)  
*You may notice some minor differences if using a different version.*

 Quantitative Fisheries Center

MICHIGAN STATE  
UNIVERSITY

In this video, we consider one particularly nasty type of bug and how to identify when it exists. The bug arises when you attempt use values outside the legal range of indices for an array.



Next

Finding bugs when you try to write or read values  
from outside the legal range indices for an array

Particularly nasty because your program may still  
compile and appear to run ok

Results may be nonsense

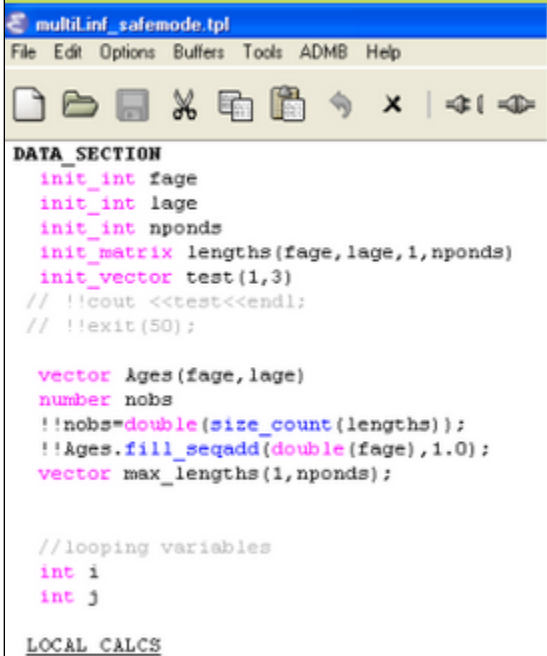
Safe mode will identify if you have this problem by  
causing your program to abort at run time

QFC Quantitative Fisheries Center

MICHIGAN STATE  
UNIVERSITY

C++ and admb by default do not do array checking. So for example, you can create a loop and assign a value of one vector equal to a value from another, but if your loop is constructed wrong so you either try to read from an element that does not exist or write to one that does not exist you are likely to get complete nonsense in the result. Your program ends up reading or writing to unintended areas of computer memory. Your program will compile and run but possibly produce complete nonsense. If you are lucky, it will be so nonsensical that you will know something is wrong.

Next



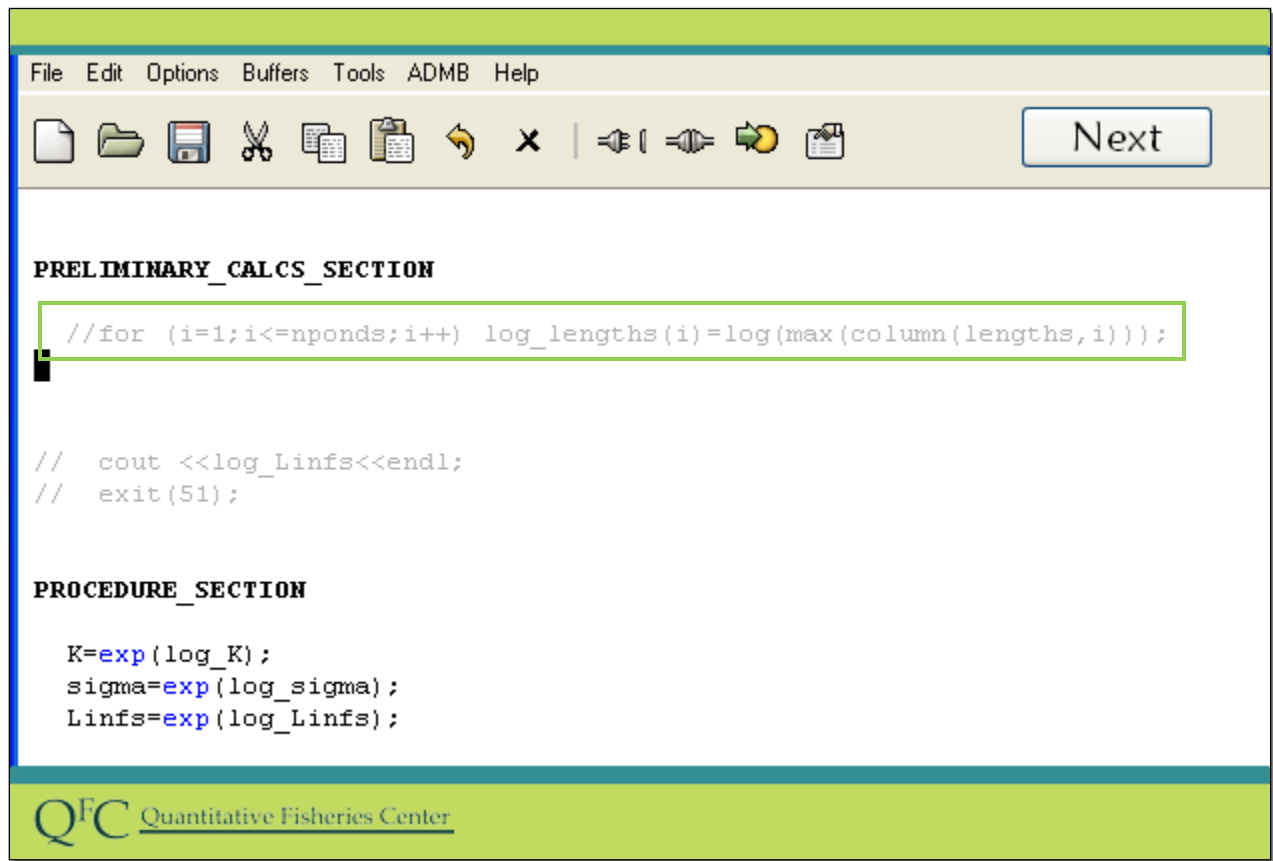
1. Save multiLinf\_safemode.tpl and multiLinf\_safemode.dat to your computer in a new folder.
2. Open multiLinf\_safemode.tpl before moving on.

We now will deliberately add a bug to a working tpl, show what can happen, and then show how you can use safe mode to avoid problems like this. For this demo we will use the working version of a growth model we previously used in a video to illustrate loops, and introduce a blatant bug simply for illustrative purposes.

If you want to follow along make sure you have the multiLinf\_safemode.tpl and multiLinf\_safemode.dat available from where you obtained this video. These files must be in the same directory.

**Slide Action:**

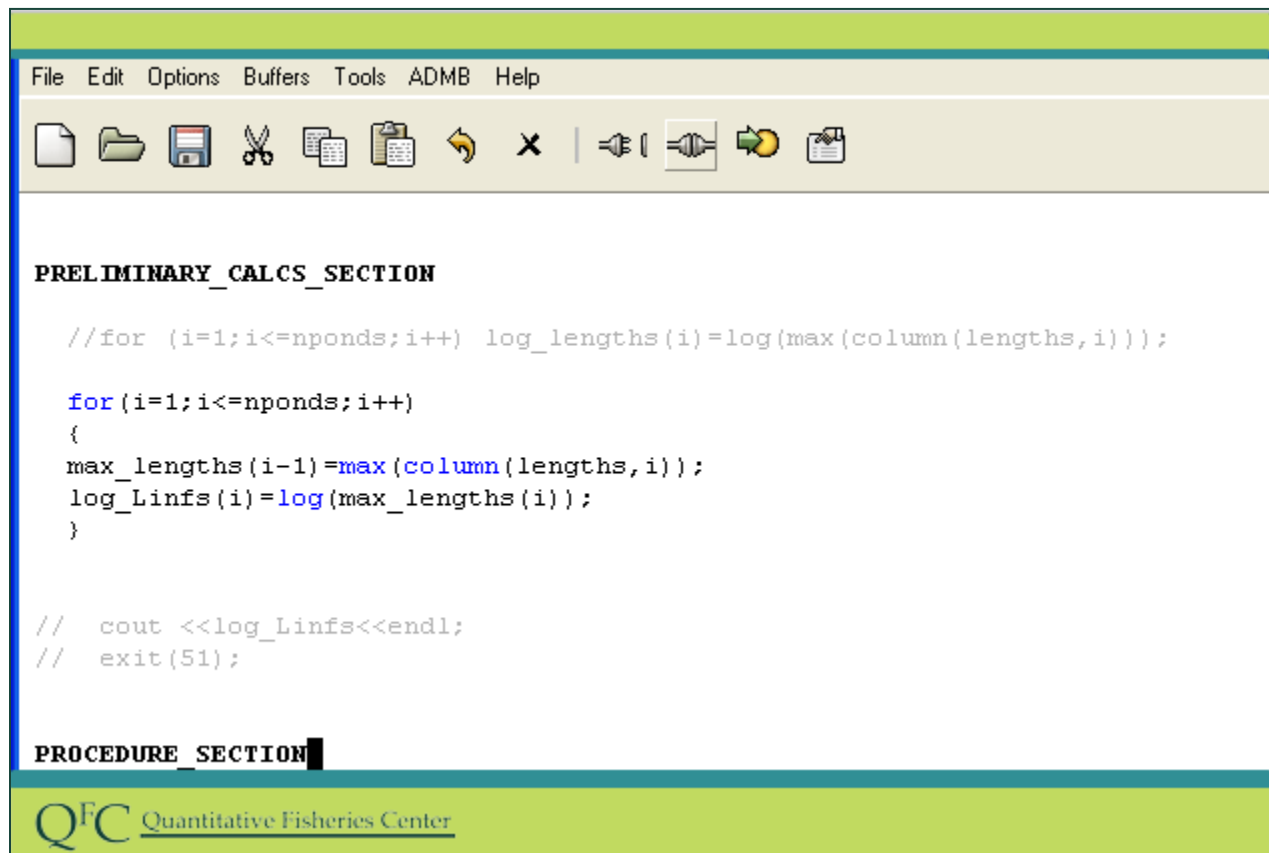
1. Save multiLinf\_safemode.tpl and multiLinf\_safemode.dat to your computer in a new folder.
2. Open multiLinf\_safemode.tpl before moving on.



Scroll down to the preliminary calcs section and comment out the active code.

**Slide Code:**

`// for (i=1;i<=nponds;i++).....`



```
PRELIMINARY_CALCS_SECTION

//for (i=1;i<=nponds;i++) log_lengths(i)=log(max(column(lengths,i)));

for(i=1;i<=nponds;i++)
{
max_lengths(i-1)=max(column(lengths,i));
log_Linfs(i)=log(max_lengths(i));
}

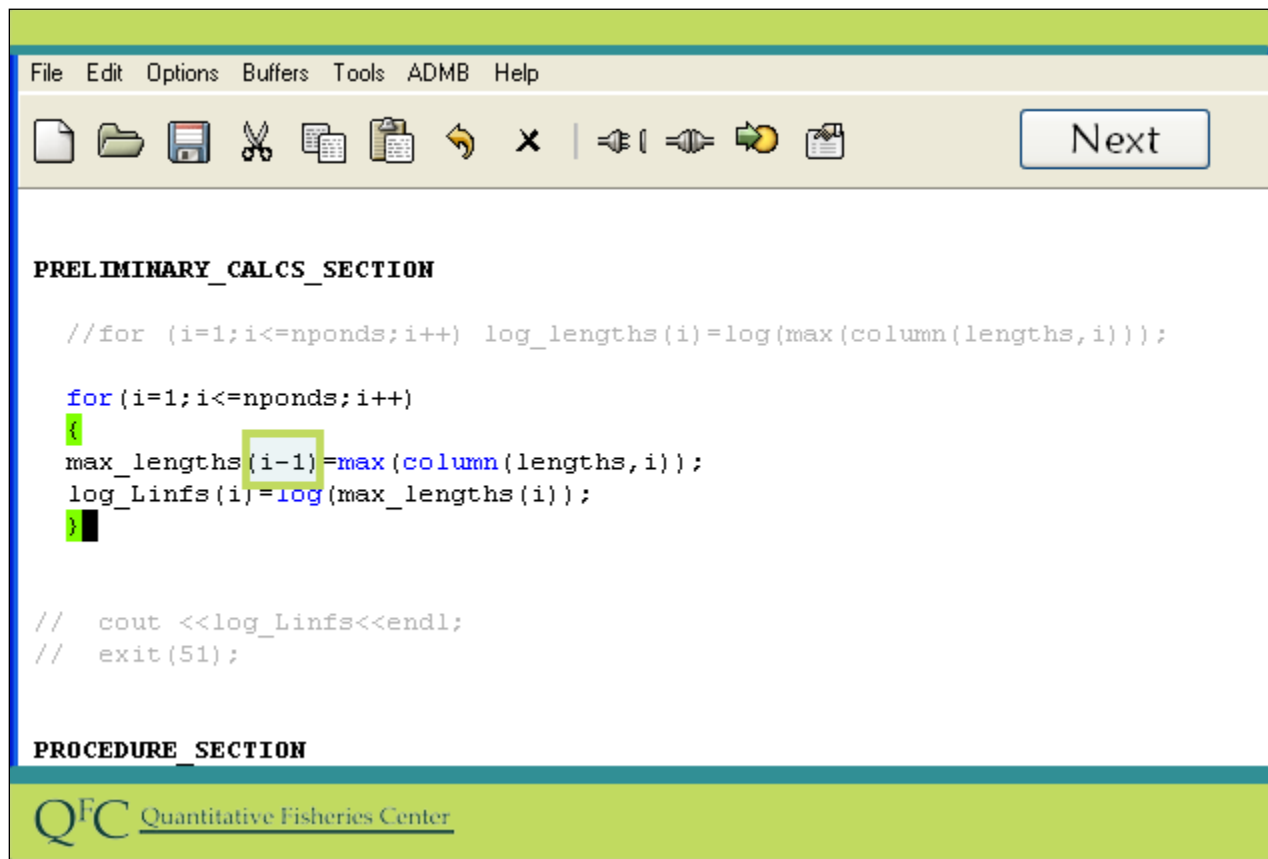
// cout <<log_Linfs<<endl;
// exit(51);

PROCEDURE_SECTION
```

We replace this version of the loop by a loop where the calculations are done in two separate lines. We first calculate the maximum value and then we take the log of it and set the value of the parameter we are estimating equal to this.

**Slide Code:**

```
for (i=1;i<=nponds;i++)
{
max_lengths (i-1)= max(column(lengths,i));
log_Linfs(i) = log(max_lengths(i));
}
```



```
File Edit Options Buffers Tools ADMB Help

//for (i=1;i<=nponds;i++) log_lengths(i)=log(max(column(lengths,i)));

for(i=1;i<=nponds;i++)
{
max_lengths(i-1)=max(column(lengths,i));
log_Linfs(i)=log(max_lengths(i));
}

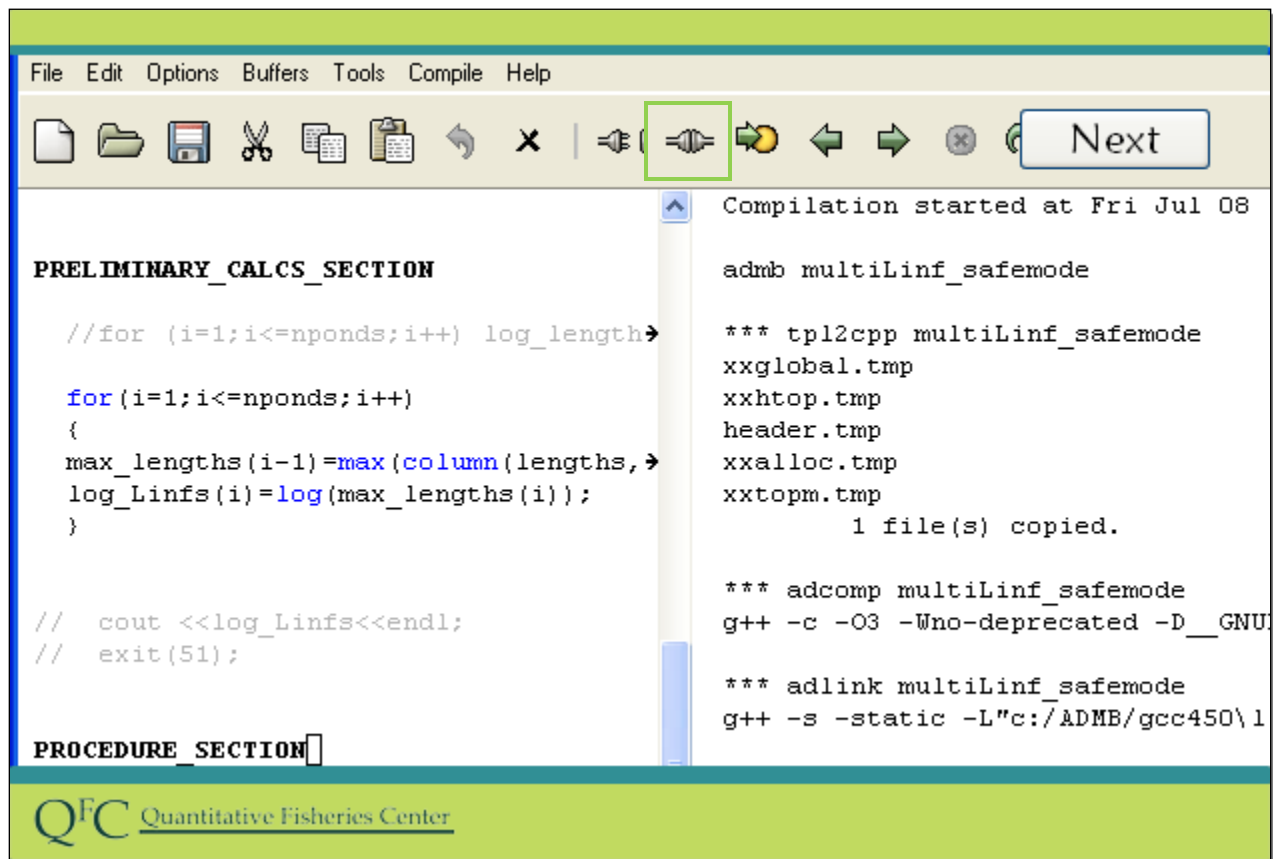
// cout <<log_Linfs<<endl;
// exit(51);

PRELIMINARY_CALCS_SECTION

PROCEDURE_SECTION

QFC Quantitative Fisheries Center
```

Notice that we did something else odd. In the first line of the code within the new loop we set the  $i-1$  value of `max_lengths` equal to the maximum of the  $i$ th column. We did this to create an error and it's a bit blatant here because it's pretty obvious that we should use the same index in both cases, but this kind of mistake can easily happen in more complex coding situations.



We now build the program. We see that things appear to be ok as the build works.

**Slide Action:**  
**Build program**

```

Function value  1.#INF000e+000; maximum gradient component mag  1.#INFe+000
Var  Value      Gradient |Var  Value      Gradient |Var  Value      Gradient
 1 -1.61000 1.#INF0e+000 | 2  3.90000 -1.#INF0e+000 | 3  0.00000 -1.#INF0e+
 4  -1.#J 0.00000e+000 | 5  -1.#J 0.00000e+000 | 6  -1.#J 0.00000e+00
 7  -1.#J 0.00000e+000 | 8 -722.2594 -3.24932e-314 | 9 -744.4401 -4.94066e-
10  -1.#J 0.00000e+000 |11 -723.0189 -1.45866e-314 |12  -1.#J 0.00000e+0
13 619.2987 1.#INF0e+000 |
ic > imax in fminim is answer attained ?
Var  Value      Gradient |Var  Value      Gradient |Var  Value      Gradient
 1 -1.61000 1.#INF0e+000 | 2  3.90000 -1.#INF0e+000 | 3  0.00000 -1.#INF0e+
 4  -1.#J 0.00000e+000 | 5  -1.#J 0.00000e+000 | 6  -1.#J 0.00000e+00
 7  -1.#J 0.00000e+000 | 8 -722.2594 -3.24932e-314 | 9 -744.4401 -4.94066e-
10  -1.#J 0.00000e+000 |11 -723.0189 -1.45866e-314 |12  -1.#J 0.00000e+0
13 619.2987 1.#INF0e+000 |
Function minimizer not making progress ... is minimum attained?
Minimprove criterion =  0.0000e+000

- final statistics:
13 variables; iteration 1; function evaluation 30
Function value  1.#QNBe+000; maximum gradient component mag  1.#INFe+000
Exit code = 1;  converg criter  1.0000e-004
Var  Value      Gradient |Var  Value      Gradient |Var  Value      Gradient
 1  1.61000 1.#INF0e+000 | 2  3.90000 -1.#INF0e+000 | 3  0.00000 -1.#INF0e+
 4  -1.#J 0.00000e+000 | 5  -1.#J 0.00000e+000 | 6  -1.#J 0.00000e+00
 7  -1.#J 0.00000e+000 | 8 -722.2594 -3.24932e-314 | 9 -744.4401 -4.94066e-
10  -1.#J 0.00000e+000 |11 -723.0189 -1.45866e-314 |12  -1.#J 0.00000e+0
13 619.2987 1.#INF0e+000 |

```

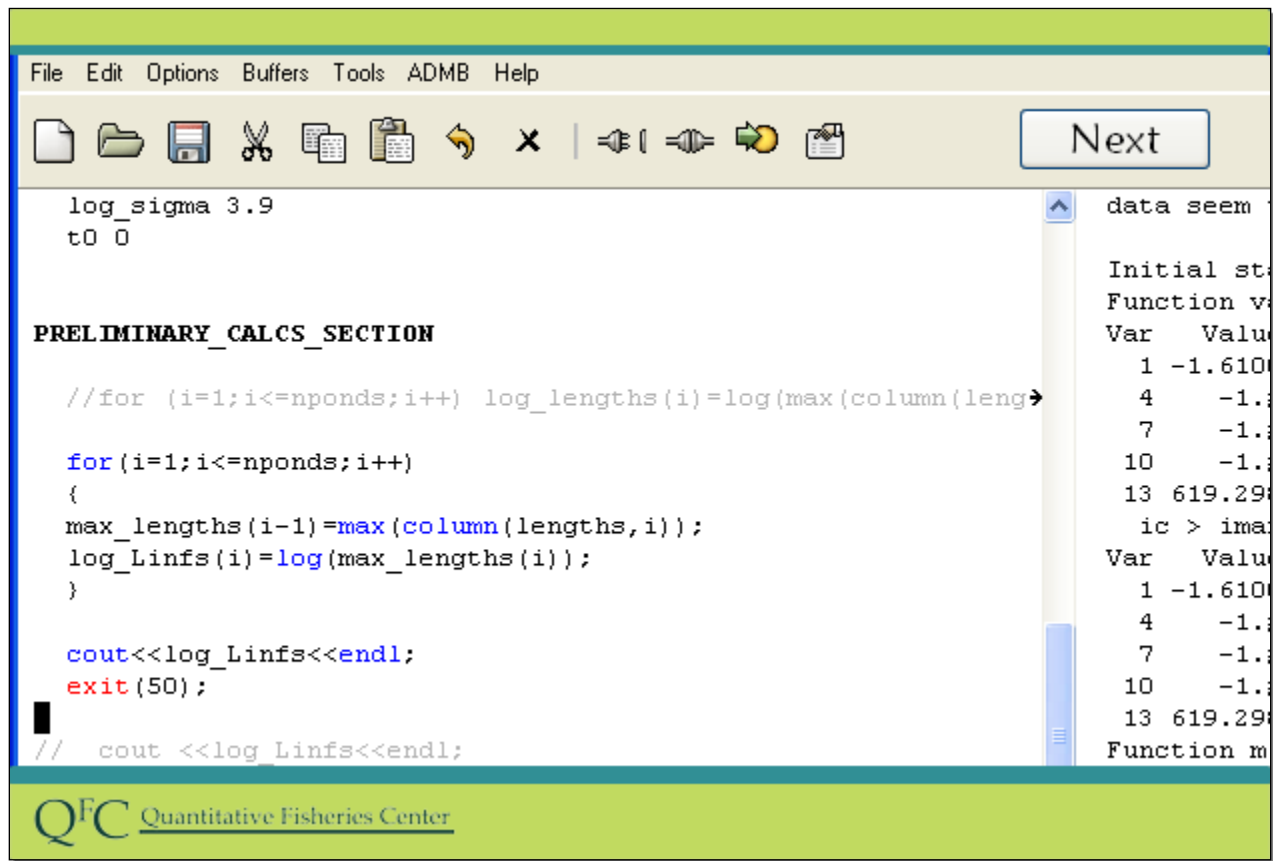
[Next](#)

Things get uglier when we run the program. When we look at the output produced on the screen we see ugly stuff.

We did not get convergence to the gradient condition and we see that some of our gradients are infinite and other things being written to the screen are not even numbers.

**Slide Action:**  
Run program





```

log_sigma 3.9
t0 0

PRELIMINARY_CALCS_SECTION

//for (i=1;i<=nponds;i++) log_lengths(i)=log(max(column(leng>

for(i=1;i<=nponds;i++)
{
max_lengths(i-1)=max(column(lengths,i));
log_Linfs(i)=log(max_lengths(i));
}

cout<<log_Linfs<<endl;
exit(50);

// cout <<log_Linfs<<endl;

```

data seem

Initial st:

Function v:

Var	Value
1	-1.6100
4	-1.6100
7	-1.6100
10	-1.6100
13	619.294

ic > ima

Var	Value
1	-1.6100
4	-1.6100
7	-1.6100
10	-1.6100
13	619.294

Function m

Q<sup>FC</sup> Quantitative Fisheries Center

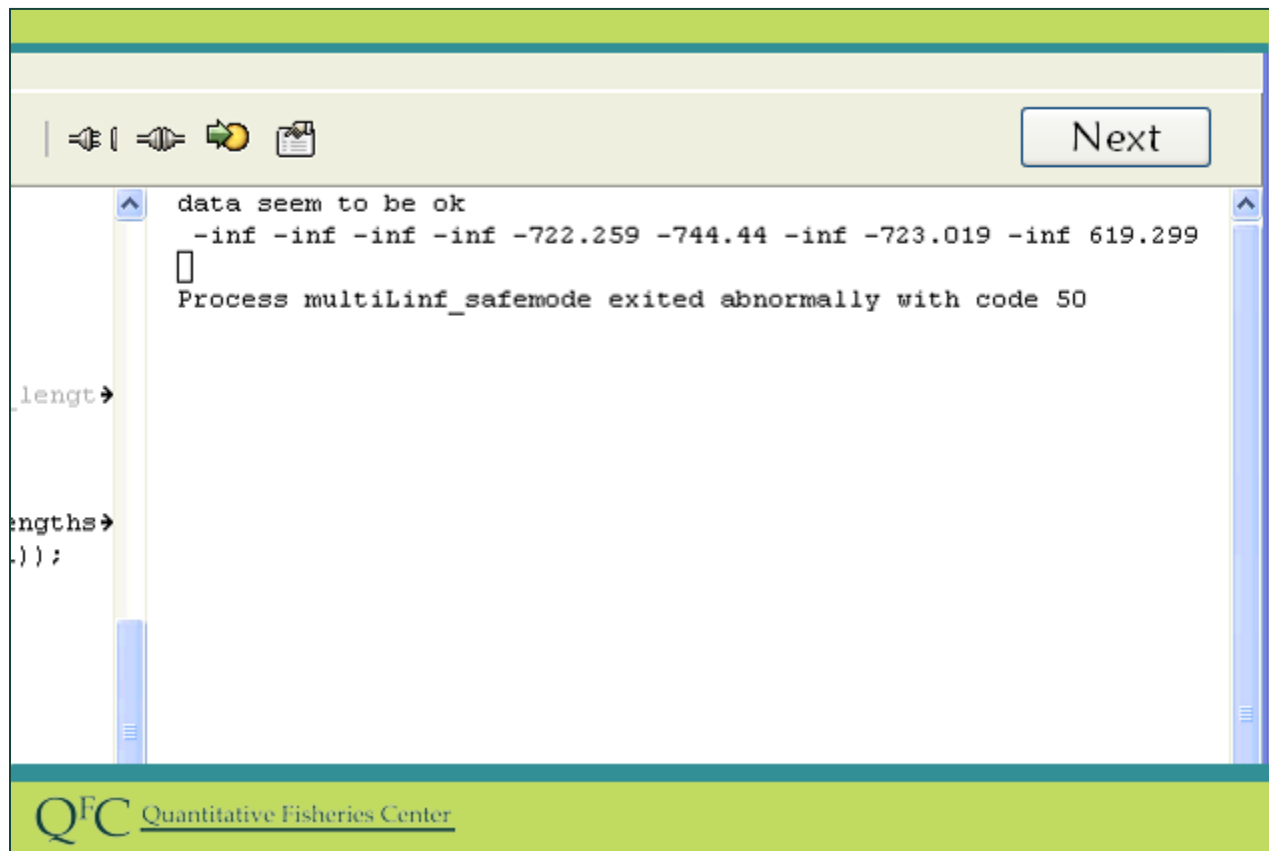
We can see the source of the problem by using cout and exit right after our bug to see what values of log L-infinity we were starting with.

### Slide Code:

```

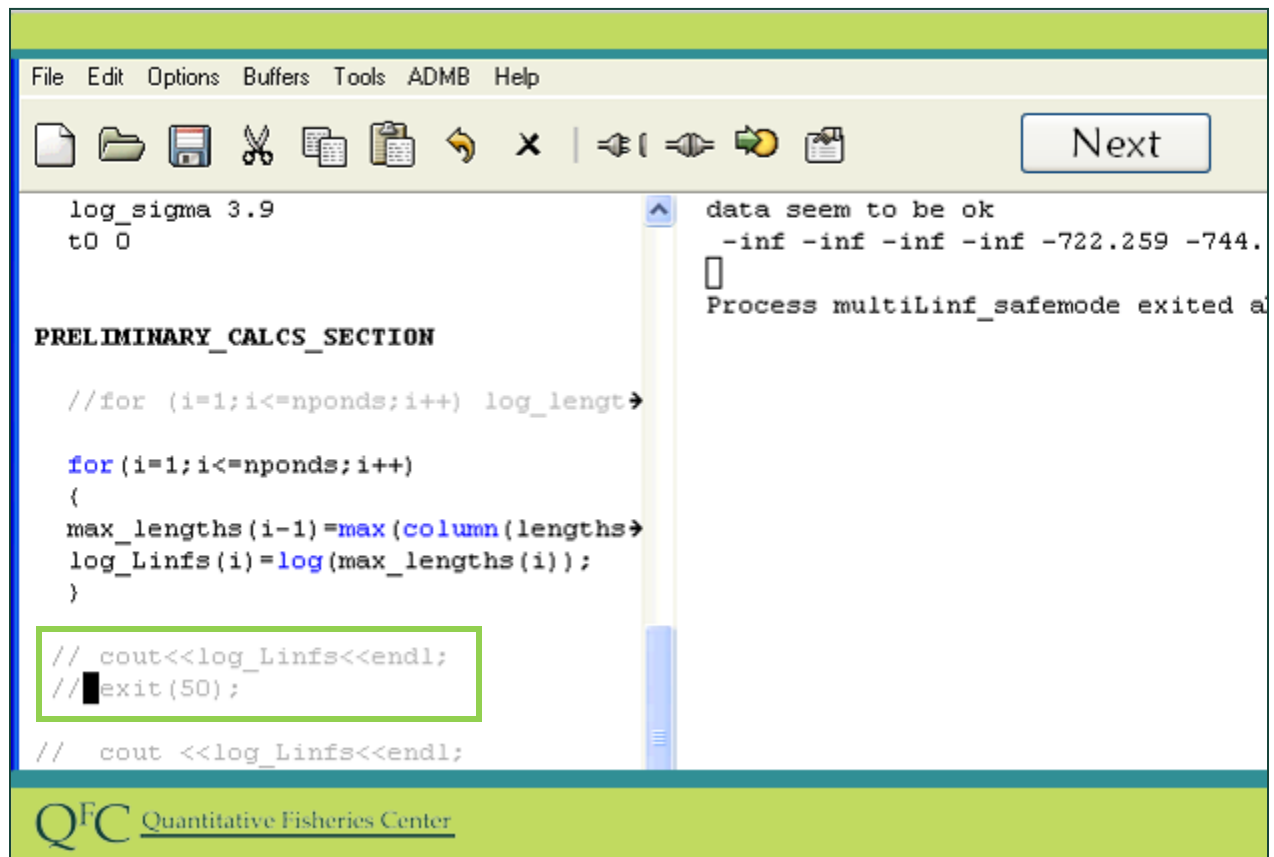
cout<<log_Linfs<<endl;
exit(50);

```



We now build and run our program. You can see that things started bad from the start with very bad values including some negative infinity values for log-Linfinity. In actual applications it is possible to introduce a bug like this and not know you did. Safe mode is a way to have your program automatically abort when there is an error when you try to read outside the legal lower or upper indices for an array. You do this by compiling your program in what is known as safe mode. Your program will still compile ok because the program does not know in advance – that is before it is run -- that the looping variable will take illegal values of  $i$  when an array is written to.

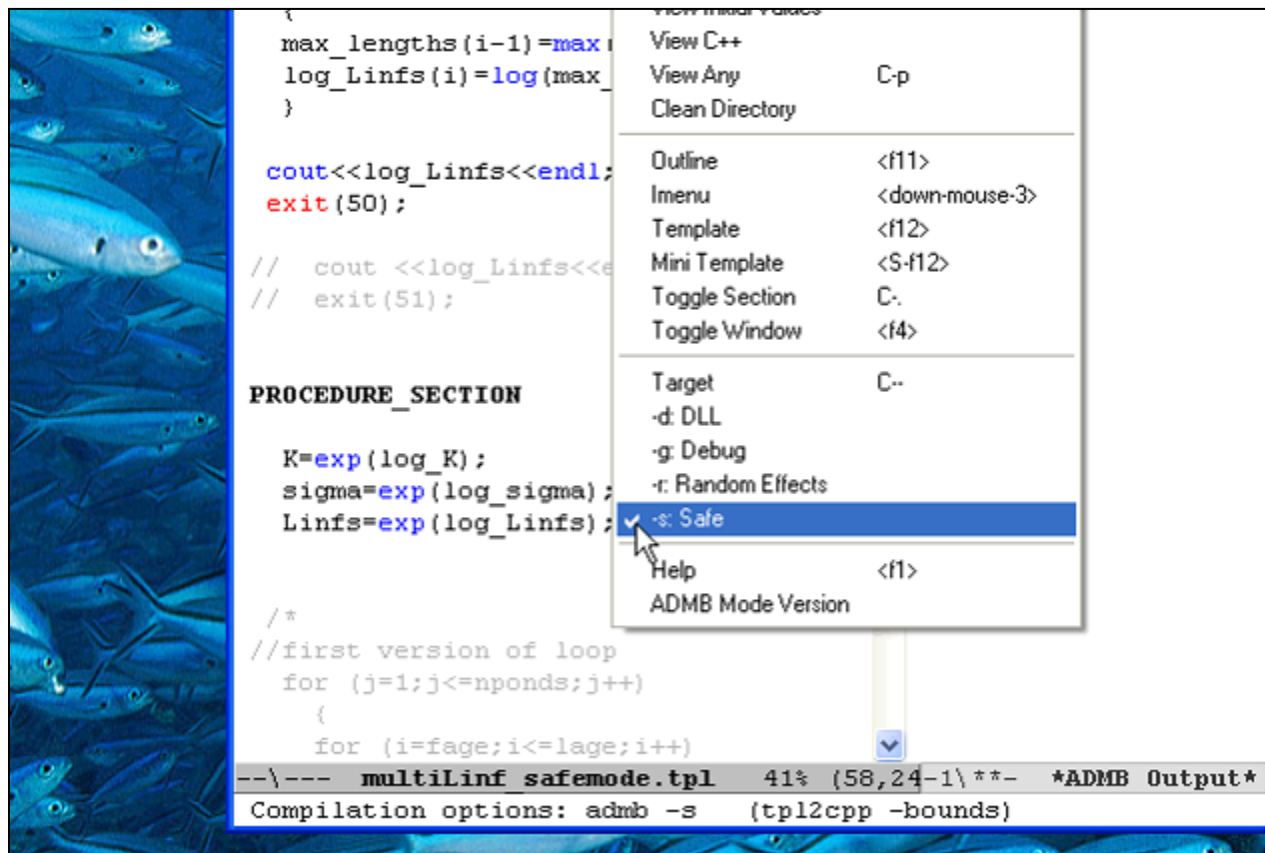
**Slide Action:**  
**Build program**  
**Run program**



To illustrate safe mode, first we comment out our cout and exit statements.

**Slide Code:**

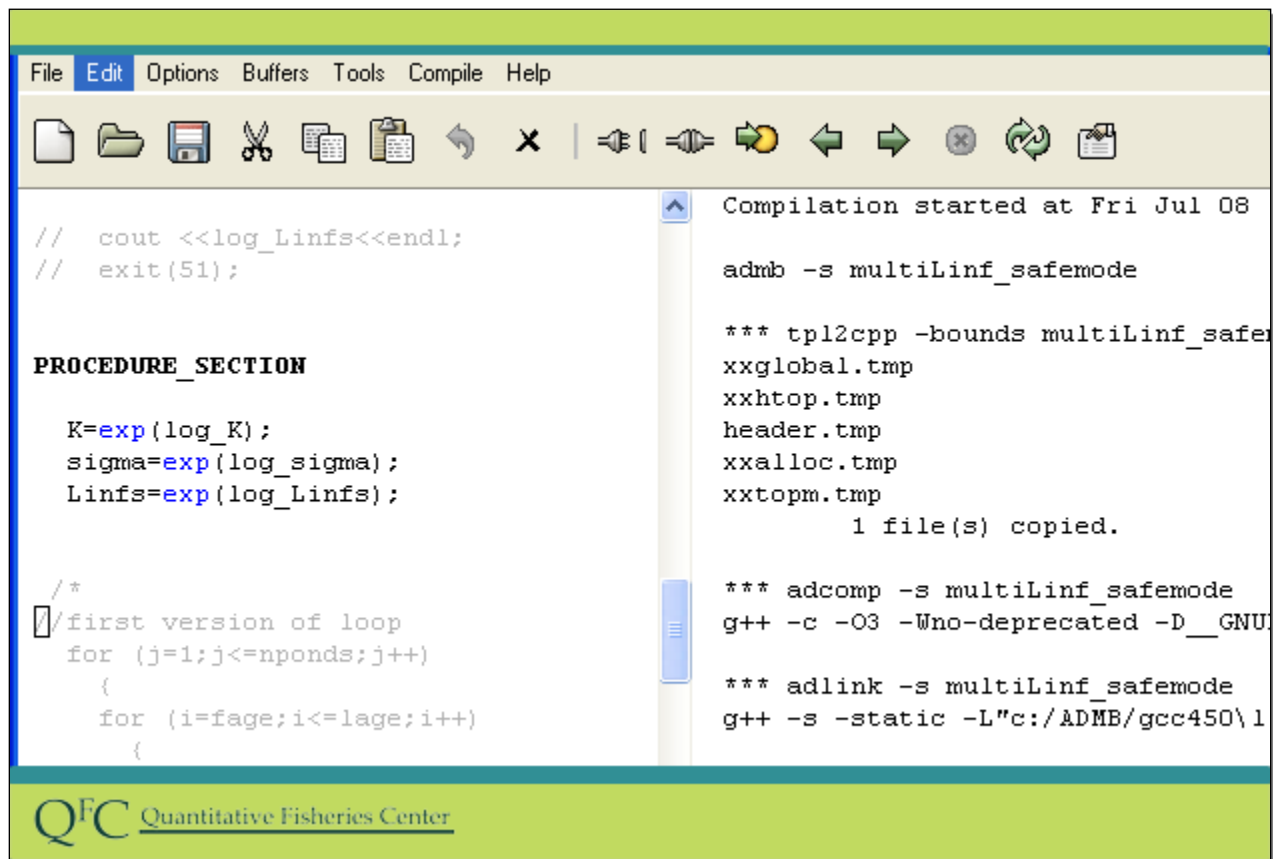
```
// cout<<log_Linfs<<endl;
// exit(50);
```



Now we go to the admb menu. Within the target section we select the dash s safe option (`-s: Safe`). You can look at the admb menu again to see that there is now a check by the safe mode.

### Slide Action:

Go to the ADMB menu and select `-s: Safe`. Make sure there is a check next to it.



The screenshot shows an IDE window with a menu bar (File, Edit, Options, Buffers, Tools, Compile, Help) and a toolbar. The main editor displays C++ code with the following content:

```
// cout <<log_Linfs<<endl;
// exit(51);

PROCEDURE_SECTION

K=exp(log_K);
sigma=exp(log_sigma);
Linfs=exp(log_Linfs);

/*
//first version of loop
for (j=1;j<=nponds;j++)
{
    for (i=fage;i<=lage;i++)
    {
```

On the right side, a terminal window shows the compilation process:

```
Compilation started at Fri Jul 08

admb -s multiLinf_safemode

*** tpl2cpp -bounds multiLinf_safemode
xxglobal.tmp
xxhtop.tmp
header.tmp
xxalloc.tmp
xxtopm.tmp
    1 file(s) copied.

*** adcomp -s multiLinf_safemode
g++ -c -O3 -Wno-deprecated -D__GNU__

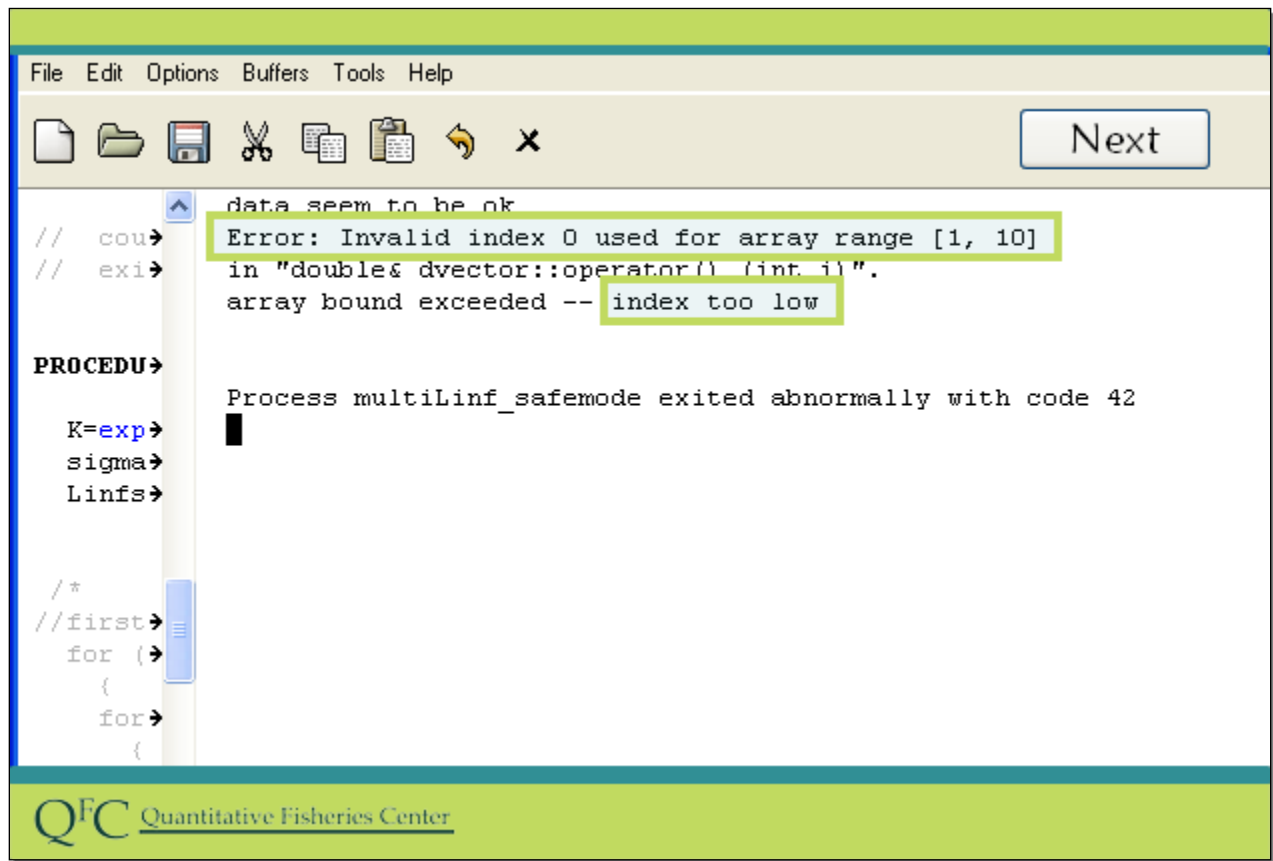
*** adlink -s multiLinf_safemode
g++ -s -static -L"c:/ADMB/gcc450\lib" -o multiLinf_safemode.exe
```

### Slide notes

Now we rebuild our program. Again it seems to build ok.

### Slide Action:

#### Build Program



```
File Edit Options Buffers Tools Help
[Icons] [Next]

data seem to be ok
// cout<
// exit<
Error: Invalid index 0 used for array range [1, 10]
in "double& dvector::operator() (int i)".
array bound exceeded -- index too low

PROCEDURE
K=exp
sigma
Linfo

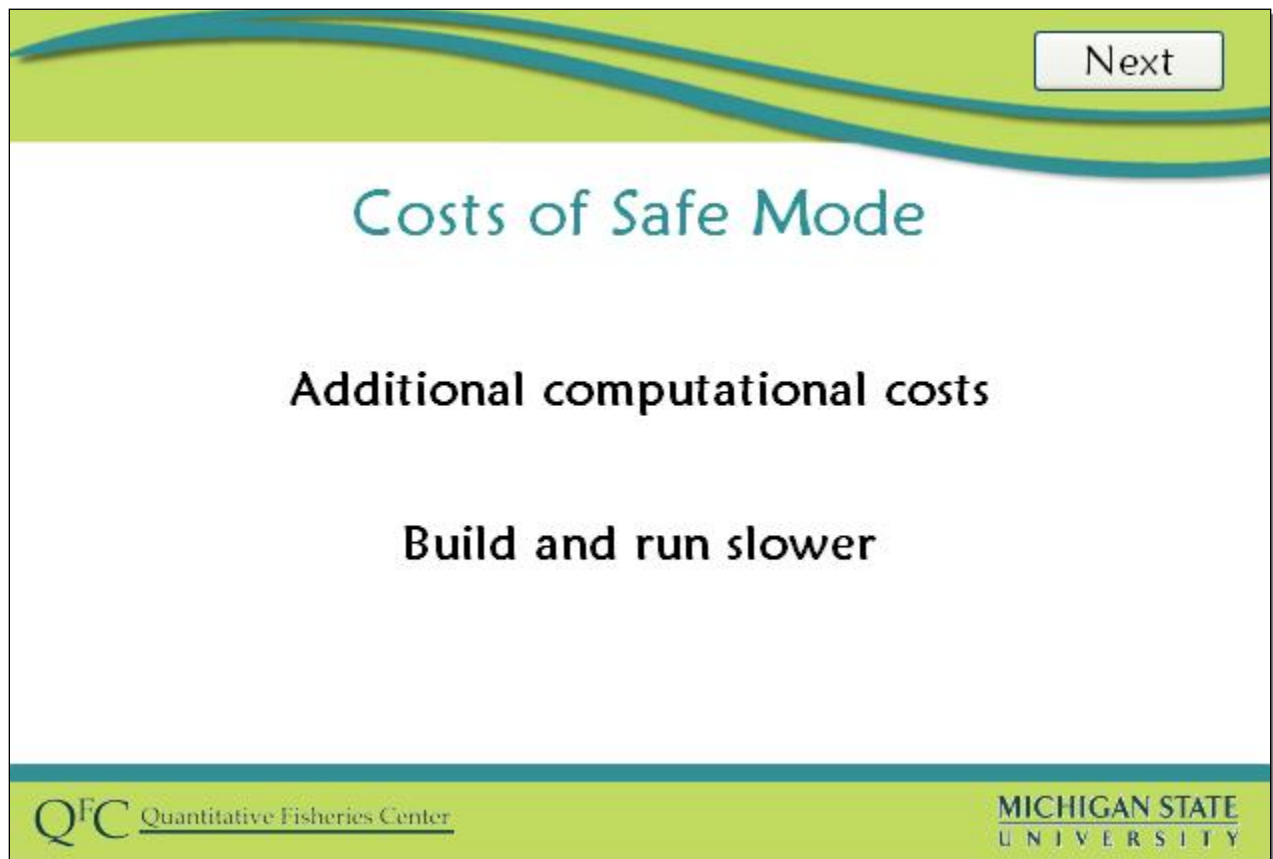
/*
//first
for (
{
for
{
```

Again we run it. But now instead of running and producing strange results it exits abnormally and gives us a runtime error that is more informative about what the problem is.

While this error is still a bit opaque in that it does not indicate which vector is really the problem, it does tell us we tried to access below the minimum index for an array. It also tells us we tried to use a zero index on an array that is legally defined from one to 10 so this gives us a place to start when looking at code or using cout and exit statements or a more formal debugging tool in search of the error.

### Slide Action:

### Run Program



Next

## Costs of Safe Mode

**Additional computational costs**

**Build and run slower**

QFC Quantitative Fisheries Center

MICHIGAN STATE UNIVERSITY

Safe mode is very handy. One might ask: why not use it all the time? In many cases just using it makes sense. However safe mode carries with it additional computational costs so when you use it programs will compile and run less quickly. In some cases that may matter, although using safe mode during debugging phases is usually a good idea.



You have now completed the video on safe mode.