Understanding EXC\_BAD\_ACCESS

<http://loufranco.com/blog/understanding-exc_bad_access>

What EXC\_BAD\_ACCESS is saying that you did something that caused a pointer to be dereferenced and that memory location isn`t inside one of the chunks assigned to your program.

This could be because:

1.The pointer used to point to that memory is ok, but its chunk was deallocated.

2.The pointer is corrupt.

\*The line of code your app crashes on is not the root cause of the problem.

\*The problem in #1 is whatever line of code caused the premature deallocation.

\*The problem in #2 is whatever line of code corrupted the pointer.

Solutions:

#1 is far easier to fine.It`s almost definitely because you didn`t use retain/release correctly and there where either too many releases or too few retains.

DO THIS:

1.Run Build and Analyze. Make sure you fix or understand every single error it flags. I personally have 0 Build and Analyze errors in every project I have and I go out of my way to keep it that way. If I ever get a false positive, I figure out how to make Build and Analyze understand what is going on, so that it doesn’t flag it.

2.Run scan-build with all checks on. This isn’t built in, so if you’re in a hurry, skip this for now. scan-build is the project that Build and Analyze is based on. It can be run with much more thorough settings.

3.Set up Xcode so that it never deallocates. Instead, it turns objects into Zombies that complain if they are used. See Tip #1 on this post for instructions.

\*\*\*If you have a clean Build and Analyze and no Zombies complain of being accessed, and you still get EXC\_BAD\_ACCESS, then it’s a good bet that you are not accessing deallocated memory. It’s not a sure bet, because the iPhone SDK gives you access to the C library which uses a different kind of allocation, which you could be using wrong.

#2 possible reasons:

1. The pointer could have never been initialized.

2. The pointer could have been accidentally written over because you overstepped the bounds of an array.

3.The pointer could be part of an object that was casted incorrectly,and then written to.

4.Any of the above could have corrupted a different pointer now points at or near this pointer,and using that one corrupts this one(and so on).

DO THIS:

1.Enable Guard Malloc (Tip #2) – this makes the datastructure that represents the allocations much more sensitive to corruption. You need to use the enhanced features of the debugger to get anything out of it (explained in the tip).

2.The line of code that triggers the crash is a clue to what pointer is corrupt. If you move it around it might be able to help you narrow down the point of corruption.

3.If all else fails and you are desperate, try Valgrind.

\*\*\*Once one of these methods gives you a different problem (either a warning or another EXC\_BAD\_ACCESS), don’t worry that it seems completely unrelated to your original problem — that’s a common problem with corruption.

\*\*\*Also, random changes to your program may make this problem “go away” — it’s not really fixed, though. Your corruption or early deallocation is still there, but it’s not triggering an EXC\_BAD\_ACCESS. Its effects could be far worse, however, so it’s a good idea to try to keep reproducing it until you are sure you addressed the problem.

\*\*\*\*THINGS TO REMEMBER:

1.Corrupting a pointer doesn’t immediately trigger EXC\_BAD\_ACCESS. Neither does using a corrupted pointer unless it’s specifically now pointing to memory that isn’t mapped to your application.

2.Deallocating an object doesn’t immediately release the memory to the operating system — only when the chunk is unused, can it be returned.

3.The line of code that is triggering the EXC\_BAD\_ACCESS might not be the problem. It can be a good clue, but don’t assume the problem is this code.