valgrind | Coursera

valgrind

valgrind

valgrind is particularly good at finding errors in your program that did not manifest simply because you got lucky when you ran it. For example, recall from from previous discussions that a variable does not have a value until you assign it one—we just draw a ? in its box to indicate the value is unknown. If your program has a variable which is used before it assigned any value—called an *uninitialized variable* —the variable will evaluate to whatever value was previously stored into the location that the variable resides in. You may get "lucky" on what value the uninitialized variable ends up holding. In fact, you may "lucky" thousands of times and not see any problem, but then get unlucky when it actually matters. This may seem highly improbable—after all, with billions of possible values that the variable could end up being, what are your chances of getting lucky? However, the value that you end up with is not random, it is whatever was in that storage location previously—for example, the value that some other variable had when it went out of scope.

When you run your program directly on the computer, it does not explicitly track whether a variable has been initialized—the compiler generates instructions which do exactly what your program specifies, and the computer does exactly what those instructions tell it to. When you run your program in *valgrind,* however, *valgrind* explicitly tracks which variables are initialized and which are not. If your program uses the uninitialized value in certain ways, *valgrind* will report an error to you (it does not do this in all cases for a variety of reasons beyond the scope of this discussion—however, suffice it to say that it will catch the cases where it really matters).

valgrind is useful for detecting many other problems with your program that you might not otherwise discover easily. We highly recommend running your program in *valgrind* whenever you are testing your program. We will talk about testing in much more detail later in the course, but for now, you will want to run your program to see if it works. When we learn how to make programs that work with various forms of input, you will want to test your program on a variety of inputs to become more and more confident that it works.