1 Terminology and techniques of testing

- A fault is the root problem in the software.
- An error is an incorrect program state b/c of the fault.
- A failure is an observable incorrect behaviour.
- Testing is done to find failures. We find what's observably wrong with the program.
- Debugging is done to fix faults. We fix our problem based on what failures we found.
- Boundary value testing is testing specific values (black box testing). Edges of our "bounds" are typically where errors will be found, so take care to test them. Testing bounds, and +- one or two of those bounds helps us check and potentially catch off-by-one errors and the like. Edge cases are on one boundary, corner cases are on more than one boundary. Note that testing ALL bounds may be unfeasible.
- Black box tests: We test based on having no knowledge of the code, but we may have access to program specifications.
  - Equivalence partitioning would basically be testing random values from between bounds just to check for general correctness.
  - Error guessing is basically just intuition - knowing what values will potentially break your program. Examples are 0, absurdly small and large numbers, negatives, positives, etc.
  - Error checking may reveal potential issues with documentation - potentially some cases that would never run are not specified so! For example, perhaps a payroll application might forget to check for invalid inputs and not specify that these inputs are invalid in documentation.
  - Rounding errors are another common thing to check, especially with floats/doubles.
  - If you don't have the code you're testing, you could do ”fuzz testing". Essentially, you write a program that tests your other program. This is very useful for testing either incorrect inputs IF you have a way of determining the answer another way, or if you just want to test if your program crashes.
- White box tests: We test every single branch of your code, as you have access to the code. This is useful for checking for dead code and obscure bugs.
- Assertion tests are kinda like white box tests, but a method of error checking nonetheless.
- Assertion tests include:
  - Pre-condition (test your arguments for validity)

  - Invarient (test stuff regarding your code in general)
  - Post-condition (test your final values for validity before returning)
  - Non-null (check for non-null values)
  - Unreachable (test for dead code, we can do this if we want to AVOID an else condition, and have specific if statements). For example:

```
if (a < b) {
    //...
}
else if (a > b) {
    //...
}
else if (a == b) {
    //...
}
else assert(false && ”How did I get here?”) //If it reaches this you
```

  - A trick we can do with assertions in C/C++ is that we can do assert(condition && ”ERROR MESSAGE”), as string literals are non-null constant char arrays/C++ strings, and as such, will always be true. We can use this to our advantage to print out failed asserts in addition to a message to help us know what went wrong.
  - In C++, we include the library ¡cassert¿. Note that you should never do any actual work with asserts, as you will likely disable asserts in your release program to increase speed. We disable assertions through a compiler flag (-DNDEBUG), or if you use “define NDEBUG”

- Regression testing: If your code worked before a change, and broke after, then you know where the issue is.

2 Automated testing

- We typically use a testing harness to do the testing for us.
- This allows us to easily push in inputs and return outputs, which we can check to see if it is correct.