How to Write a Test File

1 Intro

Test files are a way for you to be become reasonably confident that your code works the way you want it to. They are made up of multiple different test cases, in which you run the other code you have written and compare it to the answers you were expecting.

2 How do I start a test file?

First, create and open a new file using an editor of your choosing. A test file needs to have a main function that will be run when the code is executed, so at first the file should have the following code:

```
int main(){
    // tests will go here
    return 0;
}
```

3 Writing Test Cases

Important: For this section of the document we will be writing test cases for the following GCD function:

```
int GCD(int a, int b)
//@requires a > 0 && b > 0;
//@ensures \result >= 1;
//@ensures a % \result == 0 && b % \result == 0;
```

To start, we want to know that our code works on some small, simple tests, so we might want to test that `GCD(2, 5)` does what we want, as well as `GCD(19, 21)`. We want an assertion to fail if our code doesn't work as expected, so we will **`assert`** that the result of `GCD` on each call produces the expected output.

```
int main(){
    assert(GCD(2, 5) == 1);    // added
    assert(GCD(19, 21) == 1);  // added

    return 0;
}
```

Note: We don't use `//@assert` in test cases because we want the test cases to be run regardless of if contracts are enabled.

Furthermore, code used in an `//@assert` statement (like all contract) needs to be *pure*: executing it may not modify allocated memory. Therefore, tests that modify allocated memory would not even compile if written using `//@assert`.

Now we have written two test cases, but these cases only show us that our code probably works on inputs that have a common divisor of 1. To check that it works on some others, we can pick inputs that result in an output greater than 1.

```
int main(){
    assert(GCD(2, 5) == 1);
    assert(GCD(19, 21) == 1);

    assert(GCD(12, 20) == 4);    // added
    assert(GCD(6, 9) == 3);      // added

    return 0;
}
```

So now we have some basic test cases that test the simple functionality of our code, we should consider some trickier cases. So far, all of our test cases have the first input be less than the second input, but we can see from the `//@requires` that it should work either way.

```
int main(){
    assert(GCD(2, 5) == 1);
    assert(GCD(19, 21) == 1);

    assert(GCD(5, 2) == 1);    // added
    assert(GCD(21, 19) == 1);  // added

    assert(GCD(12, 20) == 4);
    assert(GCD(6, 9) == 3);

    assert(GCD(20, 12) == 4);  // added
    assert(GCD(9, 6) == 3);    // added

    return 0;
}
```

We also might want to consider what happens when the result is the same as one of the inputs, as well as what happens when the two inputs are the same.

```
int main(){
    assert(GCD(2, 5) == 1);
    assert(GCD(19, 21) == 1);

    assert(GCD(5, 2) == 1);
    assert(GCD(21, 19) == 1);

    assert(GCD(12, 20) == 4);
    assert(GCD(6, 9) == 3);
```

```
    assert(GCD(20, 12) == 4);
    assert(GCD(9, 6) == 3);

    assert(GCD(6, 18) == 6);   // added
    assert(GCD(18, 6) == 6);   // added
    assert(GCD(5, 20) == 5);   // added
    assert(GCD(20, 5) == 5);   // added

    return 0;
}
```

When it comes to thinking about edge cases, we usually try to see if our code works on the smallest inputs and largest inputs. In this case, there is a precondition that both of the inputs are positive, so the smallest they can be is `1` and the largest is `int_max()`. To be able to use `int_max()`, we have to `#use <util>`.

```
                                                    // added
#use <util>

int main(){
    assert(GCD(2, 5) == 1);
    assert(GCD(19, 21) == 1);

    assert(GCD(5, 2) == 1);
    assert(GCD(21, 19) == 1);

    assert(GCD(12, 20) == 4);
    assert(GCD(6, 9) == 3);

    assert(GCD(20, 12) == 4);
    assert(GCD(9, 6) == 3);

    assert(GCD(6, 18) == 6);
    assert(GCD(18, 6) == 6);
    assert(GCD(5, 20) == 5);
    assert(GCD(20, 5) == 5);

    assert(GCD(1, int_max()) == 1);                 // added
    assert(GCD(int_max(), 1) == 1);                 // added
    assert(GCD(int_max(),int_max()) == int_max());  // added
    assert(GCD(1, 1) == 1);                         // added

    return 0;
}
```

Important: It is impossible in C0 to write test cases that check that certain function calls fail a precondition, so all of the calls to our function that our test file makes has to satisfy the preconditions.

Let's say that we put this test code in a file called `test-gcd.c0`, and our gcd code was in a file called `gcd.c0`. To compile and run our tests with our gcd code we would execute the following command:

```
% cc0 gcd.c0 test-gcd.c0
% ./a.out
```

4 Edge Cases

Testing edge cases is super important. Code that works on edge cases often also works on tests that are not edge cases. Edge cases are test cases that deal with the edges of the range of valid inputs.

Some common edge cases are:

- extremely small or large integers
- empty arrays
- null pointers
- empty strings