– usage –

Your tests should be written as a single .c file separate from the body of text containing your functionality to be tested. A simple example might look something like this:

```
1  #include "simple-test.h"
2  #include "header_with_stuff_to_be_tested.h"
3
4  BEGIN_TEST
5
6  /* a simple test using only stack mem */
7  TEST("description of the first test")
8  {
9    int var1=2;
10   int var2=4;
11
12   /* add is a function included from our hypothetical
13    * header_with_stuff_to_be_tested */
14   EXPECT_INT("error message shown on failing",
15                   var1+var2, add(var1, var2));
16 }
17
18 /* this test uses heap memory, so things get a bit
19  * more complicated */
20 TEST("this is the second test")
21 {
22   /* first, ensure all your pointers which will
23    * point to heap mem are declared */
24   char *heap_string=NULL;
25
26   /* next, declare a list of statements to be
27    * called to clean up memory once the test
28    * is completed */
29   CLEANUP(
30       if(heap_string != NULL)
31         free(heap_string);
32         );
33
34   /* then, define the body of the test */
35
36   /* STATE can be used to report (with pretty
37    * formatting) the current state within the
38    * test, which may be useful in the case of
39    * a segfault */
40   STATE("grabbing heap string");
41
42   heap_string=get_heap_string_value();
43
44   EXPECT_STR("i suck at grabbing pointers!",
45                   "expected value", heap_string);
46
47   /* finally, call RETURN(); to run the
48    * cleanup code and continue */
49   RETURN();
50 }
51
52 END_TEST
```

If both tests above succeed, the output will look like this:

```
1 :: description of the first test
2 :: this is the second test
  :: grabbing heap string...
  :: success!
```

If the first test fails, it will look something like this:

```
1 :: description of the first test
  :: FAIL: error message shown on failing
  ::   expected:6
  ::     actual:0
```

– defined macros –

BEGIN_TEST: must appear before all tests and after all global variable declarations

END_TEST: must appear at the end of your test program

CLEANUP(statements): this defines a list of statements to run when the test exits, either successfully or on a failure. it isn't necessary for a test to run, but, if it does appear, it must be after the declaration of all variables to which it makes reference.

RETURN(): place at the end of a test which uses CLEANUP to ensure it is called before the test exits. i couldn't find any way around this without using more than just one header file, so i hope it isn't too annoying.

STATE(description): show a prettily-formatted description of the program's state during a test. takes printf-style arguments.

EXPECT_ZERO(summary, arg): fail if `arg` does not resolve to 0

EXPECT_ONE(summary, arg): fail if `arg` does not resolve to 1

EXPECT_GREATER_THAN_ZERO(summary, arg): fail if `arg` does not resolve to a value greater than 0. this will be replaced with more generic integer comparisons soon.

EXPECT_INT(summary, arg1, arg2): fail if `arg2` does not match the expected integer value `arg1`

EXPECT_EQUAL_INT(summary, arg1, arg2): fail if `arg1` and `arg2` are not equal

EXPECT_UNEQUAL_INT(summary, arg1, arg2): fail if `arg1` and `arg2` are equal

EXPECT_STR(summary, arg1, arg2): fail if string `arg2` does not match the expected string value `arg1`

EXPECT_EQUAL_STR(summary, arg1, arg2): fail if `arg1` and `arg2` are not equivalent strings

EXPECT_UNEQUAL_STR(summary, arg1, arg2): fail if `arg1` and `arg2` are equivalent strings