Unittesting

January 18, 2026

1 Unittest

- if a problem is broken into sub-problems, it can be solved using the bottom up design approach
- sub-problems can be solved by independent functions which can be designed, tested and debugged independently
- testing code at the functional level is called **unit testing**
- we'll learn the basics of unit testing by automatically testing functions
- C++ doesn't provide standard library for unit testing
- 3^{rd} party frameworks such as doctest, googletest, etc. can be used for comprehensive unit testing
- we can also do basic unit testing using **assert()** macro function provided in **<cassert>** library

1.1 External Resources

- YouTube Video - Unit Testing in C++ https://youtu.be/u6k04EkSXFI
- YouTube Podcast - Unit Testing in C++ https://youtu.be/R9_Cif-8sio
- NotebookLM learning resources - https://notebooklm.google.com/notebook/10ae29fb-db96-4b4f-891f-ae3a74f1c2c7

1.2 Debugging and unit testing with assert

- assert can be used for basic debugging and testing
- not useful for test-driven development and large scaling unit testing in software engineering

```
#include <cassert>

std::assert(<boolean expression>);
```

- boolean expression is created comparing two data values using e.g., == equality comparison operator
  - more on comparison operators covered in Conditionals chapter
- if expression evaluates to true (two values are indeed equal), assertion is correct!
  - otherwise, assertion fails and the program halts immidiately!
- more on assert: https://en.cppreference.com/w/cpp/error/assert
- **NOTE: assert( ) doesn't work in Jupyter notebook**; see the following demo programs

1.2.1 Debugging demo

- see `demos/unittest/assert/assertdebug.cpp`

1.2.2 Unit testing demos

- using the assert concept, we can automatically test if the returned results from functions are correct or not
- note floating point computation is accurate upto 7 digits (single precision) and double is accurate upto 15 digits (double precision)
- see unit testing fruitful function with assert here `demos/unittest/assert1/unittesting.cpp`
- see improved version 2.0 unit testing example here `demos/unittest/assert2/unittesting2.cpp`
- see unit testing void function with assert here `demos/unittest/assert3/unittesting3.cpp`

1.2.3 Demo program finding area and perimeter of a triangle using functions and unit testing

- solving problems using functions is demonstrated by this demo `demos/functions/rectangle/main.cpp`

1.3 Doctest Library

- https://github.com/doctest/doctest
- light weight, fast, clean and similar to Python unittest libarary
- single hearder file `doctest.h`
- `doctest.h` can be downloaded from https://github.com/doctest/doctest/tree/master/doctest
- no build system work needed — just `#include "doctest.h"`
- syntax is very simple
- compiles fast
- syntax demo:

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "doctest.h"

int add(int a, int b) { return a + b; }

TEST_CASE("testing the add function") {
    CHECK(add(2, 3) == 5);
    CHECK(add(-1, 1) == 0);
    CHECK(add(0, 0) == 0);
}
```

- TEST_CASE defines a test case with a name
- CHECK macro checks if the expression is true
- if any CHECK fails, the test case fails
- use doctest::Approx(y).epsilon(1e-6) for floating-point comparisons
- more on doctest:
  - supports assertions, exceptions, and logging
  - supports fixtures and parameterized tests
  - supports test suites and tags
- for complete documentation, see https://github.com/doctest/doctest/blob/master/README.md
- see unit testing demo using doctest here `demos/unittest/doctest/`
- use Makefile in the same folder to build and run the demo