Random Test Data Generation Research and Design Document

Started 20/01/2026
Maksym Redchenko C00302240
SETU

Question 1:

What is the most popular C unit testing format? Google Test? or what else:

Google Test [8], based on the research, is the most popular C/C++ unit testing framework for Ubuntu Linux [10] development. It dominates professional workflows with 95% market share among C++ projects due to its maturity, cross-platform support, and VS Code [9] integration.

Question 2:

Look at similar applications:

Commercial Solutions

1. Cantata AutoTest (QA Systems) [5]
Enterprise C/C++ test generator with boundary value analysis and MC/DC coverage. Windows/Linux standalone IDE with enterprise pricing (contact sales). No VS Code integration. Proves market demand for automated C test generation

2. Parasoft C/C++test [6]
Static analysis and unit test generation tool with Google Test integration. VS Code extension available for static analysis only (not test generation). Enterprise pricing (contact sales). Focuses on general coverage rather than boundary testing. Demonstrates C/C++ tooling market maturity

Open Source/Research

3. CUnit Test Generator (GitHub - yasinhajilou) [7]
AI-driven Python CLI tool using GPT/Hugging Face to generate CUnit framework tests. No IDE integration and uses CUnit instead of Google Test. Proves the feasibility of ML-powered C test generation

---

Gap: No free, VS Code-native tool generates Google Test code with boundary focus for Linux C developers.

Feature	Cantata	Parasoft	CUnit Gen	My Tool
Free/Open Source	✗	✗	✓	✓
VS Code Native	✗	✗(static analysis only)	✗	✓
Google Test Output	✓	✓	✗(CUnit)	✓
Linux-First	✓	✓	✓	✓
Boundary Focus	✓	✗(general coverage)	✗(AI-general)	✓
Price	Enterprise (contact sales)	Enterprise (contact sales)	Free	Free

Question 3:

Look at simple examples to start with

Simple Examples to Start With

Example 1: Basic Integer Function

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

Generated Tests:

- Zero values: `add(0, 0)`
- Positive boundaries: `add(INT_MAX, 0)`
- Negative boundaries: `add(INT_MIN, 0)`
- Overflow cases: `add(INT_MAX, 1)`

`INT_MAX` is the maximum value a signed 32-bit integer can hold: 2,147,483,647

When we do `add(INT_MAX, 1)`, we're trying to store 2,147,483,648, which exceeds the maximum. This causes integer overflow

Example 2: Float Function

c
```
float divide(float numerator, float denominator) {
   return numerator / denominator;
}
```

Generated Tests:

- Zero denominator: divide(1.0, 0.0)
- Normal values: divide(10.0, 2.0)
- Very small numbers: divide(0.0001, 0.0001)
- Infinity cases: divide(FLT_MAX, 0.1)

For `divide(FLT_MAX, 0.1)`, tool generates a Google Test that verifies the function returns infinity (inf) when float overflow occurs

```
TEST(DivideTest, InfinityCase) {
    EXPECT_TRUE(isinf(divide(FLT_MAX, 0.1f)));
}
```

Example 3: Array Function

c
```
int findMax(int arr[], int size) {
    // returns maximum value
}
```

Generated Tests:

- Empty array: findMax(arr, 0)
- Single element: findMax(arr, 1)
- All negative: findMax({-5, -2, -9}, 3)
- Boundary size: findMax(arr, INT_MAX)

Example 4: Simple Structure

c
```
struct Point {
    int x;
    int y;
};

int distance(struct Point p1, struct Point p2);
```

Generated Tests:

- Origin points: `distance({0,0}, {0,0})`
- Max coordinates: `distance({INT_MAX, INT_MAX}, {0,0})`
- Negative coordinates: `distance({-10,-10}, {10,10}`

Question 4:

Why not use only regex to parse C function signatures?

Basically, we can use regex just for simple tasks as it's not reliable for full C syntax because declarations can contain nested parentheses and complicated constructs that regular expressions(regex) won't be able to handle well. [1]

Question 5:

Why use boundary value testing (and why focus on INT_MAX/MIN,

etc.)?

Boundary Value Analysis targets values at the edges of valid/invalid ranges because defects/errors are more likely to occur at boundaries than in the range between(Wikipedia,2026) [2]

Question 6:

How can Google Test be used if the code under test is C?

It can be considered three main “golden rules” for testing C with Google Test:

- Bridging the Languages:
  To test C code with Google Test, we must manage Name Mangling.

  The Problem: C++ supports function overloading, so the compiler "mangles" names to include type info (e.g., *add* becomes *_Z3addii*).

  The Conflict: C compilers store names plainly (just add). Without intervention, the C++ test runner will look for the "fancy" mangled name and fail to find your C function, causing a linker error.

  The Fix: Using *extern "C"* forces the C++ compiler to use the plain C naming convention, allowing the two languages to communicate.

  The File Setup: source code remains *C,* but test files must be *.cpp.* This allows the use of C++ features like Google Test macros while still calling C functions. [3]
  You can see an example of how it is supposed to look in the C++ on figure 1.

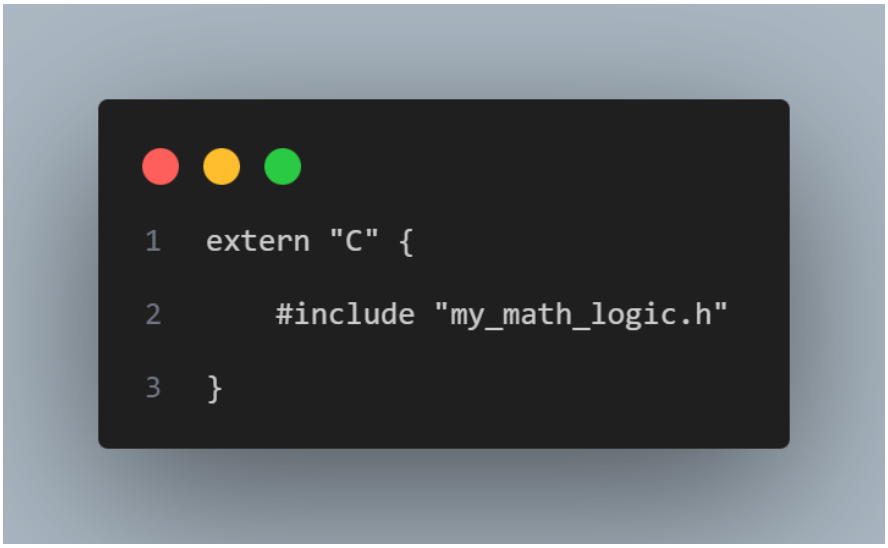


Figure 1 - Implementation Example

Question 7:

Why is Google Test version consistency important?

Current versions require a C++17 compatible environment to build successfully. If there is any mismatch between the expected code version and the installed one, linker errors could occur.

There can also be a “works on my machine” issue; documentation recommends incorporating Google Test directly into the project’s build. Based on that, it ensures that Google Test and the project’s code use the same compiler and linker settings, preventing many common errors. [4]

References:

[1] Stack Overflow (2009) Regex to pull out C function prototype declarations? Available at: https://stackoverflow.com/questions/476173/regex-to-pull-out-c-function-prototype-declarations [Accessed: 28 January 2026].

[2] Wikipedia (2026) ‘Boundary-value analysis’, *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Boundary-value_analysis [Accessed: 28 January 2026].

[3] Google C++ Testing Framework (2011) *Testing C API using gtest*. [Online discussion group] 26 January. Available at: https://groups.google.com/g/googletestframework/c/xWLOP4MY0Z4 [Accessed: 28 January 2026].

[4] Google (2025) *googletest/README.md at main*, *GitHub*. Available at: https://github.com/google/googletest/blob/main/googletest/README.md [Accessed: 28 January 2026].

[5] QA Systems (2025) Cantata AutoTest (Feature Brief). QA Systems. Available at: https://www.qa-systems.com/resource/cantata-feature-brief-autotest/ [Accessed: 29 January 2026]

[6] Parasoft (2026) Static Code Analysis Tool – Optimize C and C++. Parasoft. Available at: https://www.parasoft.com/products/parasoft-c-ctest/c-c-static-analysis/ [Accessed: 29 January 2026]

[7] Hajilou, Y. (2026) *CUnit Test Generator*, GitHub. Available at: https://github.com/yasinhajilou/cunit-test-generator [Accessed: 28 January 2026].

[8] Google (2026) GoogleTest User’s Guide. GoogleTest (GitHub Pages). Available at: http://google.github.io/googletest/ [Accessed: 29 January 2026]

[9] Microsoft (2026) Visual Studio Code. Available at: https://code.visualstudio.com/ [Accessed: 29 January 2026]

[10] Canonical (2026) *Enterprise Open Source and Linux | Ubuntu*. Ubuntu. Available at: https://ubuntu.com/ [Accessed: 29 January 2026]