The Day that GNU-C++ -Os Broke

We spent a good day of debugging a few of our C++ tests before a pattern emerged. We are working on a project to support *freestanding* C and C++ environments with the SuperTest™ test and validation suite for C and C++ compilers. A freestanding environment is defined to support only a subset of the C and C++ libraries. Its goal is to allow applications to run on targets that do not have an extensive run-time system like a proper OS. Instead, it allows programs to run on 'bare-metal'. This is useful for all kinds of embedded applications.

One of the goals in this project is to minimize the memory requirements of the SuperTest, so we use the compiler with the memory optimization option ***-Os***.

Several C++ tests, related to exception handling and inheritance, suddenly failed in the new freestanding configuration.

When tests fail like this, the first thing to suspect is our own changes. After all, the compiler used is the default g++ compiler on a fully up-to-date x86 Ubuntu environment. Many thousands (millions?) of mission-critical systems use GNU-C++ and Ubuntu. Space-X, just to name a high-profile example, uses C++ and Linux on x86 in their rockets too (according to https://www.rankred.com/what-hardware-software-does-spacex-use-to-power-its-rockets/).

Here is one such test. Yes, we know this is not *nice* C++, but we are not in the business of writing nice C++. Our job is to go near the edge of the language specification to verify the correct behavior of its implementation.

```
#include <cassert>

class A {
        virtual void f(){};
    public:
        int x;
        A(int in): x(in) {};
};

class B: public A {
    public:
        int y;
        B(int in):A(in-1), y(in) {};
};
```

```
    int test(void) {
        int res;
        B b(2);
        A* bp = &b;
        void* vp = dynamic_cast<void*>(bp);
        if (  ((A*)vp)->x == 1
           && ((B*)(A*)vp)->y == 2
           ) {
            return 1;   // PASS
        } else {
            return 0;   // FAIL
        }
    }

    int main (void) {
        assert (test ());
    }
```

The code is not complicated, but it has some unnecessary, though well-defined, type-casting. Class **A** is inherited by class **B** and both have their own instance variable, **x** and **y** respectively. The instantiation of an object of class **B**, in the second line of the function **test**, initializes these variables with the values **1** and **2**. After several type casts, the **if** statement verifies these values.

Let us first admire the beauty of the x86-64 code generated by **g++** for the function **test()** with the option **-O1**:

```
test():
  mov DWORD PTR [rsp-8], 1
  mov DWORD PTR [rsp-4], 2
  mov rdx, QWORD PTR vtable for B[rip]
  movabs rax, 8589934593
  cmp QWORD PTR [rsp-8+rdx], rax
  sete al
  movzx eax, al
  retA::f():
  rep ret
```

The compiler knows the layout of the object of type **B**. With the first two moves, it sets the values of the two variables **x** and **y**. Then it compares the values of object **b**'s fields with the 64-bit immediate **8589934593**. In hexadecimal, this value is easier to understand: **0x200000001**. The compiler does two comparisons of 32-bit values in one 64-bit comparison! That is a clever move. So far so good.

But now compile the code with **-Os**, for size optimization. The generated code becomes:

```
test():
  mov rdx, QWORD PTR vtable for B[rip]
  mov DWORD PTR [rsp-8], 1
  movabs rax, 8589934593
  cmp QWORD PTR [rsp-8+rdx], rax
  sete al
  movzx eax, al
  ret
```

Do you see what is missing? Somehow, the compiler has forgotten to initialize **b**'s variable **y** to **2**! And thus our test-program fails at the assert.

This is a serious error. Due to the optimization that turns two comparisons into one, the compiler ‘forgets’ that the **y** field of the object **b** is used. A *def-use* analysis after the optimization of the two comparisons then sees no use of **y**. Therefore the compiler concludes that the initialization of the field **y** is redundant, and removes it. Is this behavior limited to comparison optimization in combination with some liberal type-casting? Perhaps, but there is no guarantee for that.

It is not the only error. There are also errors in the generated exception handling code when **-Os** is used. The version of GNU-C shown here is 7.3.0 because that is the current version on Ubuntu. We tried different versions of GNU-C, also for ARM64 targets and not Ubuntu related, and they are all affected. We must conclude that it is not safe to use **g++** in combination with the **-Os** option for 64-bit targets.

Compiler developers run many tests to prevent that errors like these slip through. But compilers are complicated and they have so many configuration options that no compiler supplier can state upfront that your particular use case is verified.

If your application domain is mission-critical or even safety-critical, you need to set up compiler validation for the compiler and for the use cases that are specific to you. If that is beyond your scope, then at least verify that your compiler supplier uses SuperTest. SuperTest provides you with a better chance of staying ahead of compiler errors than any other method that we know of. Let's hope that Space-X does not use **g++** with the **-Os** option.