Can cout alter variables somehow?

Asked 16 years, 3 months ago Modified 12 years, 7 months ago Viewed 2k times

9

So I have a function that looks something like this:

```
float function(){
    float x = SomeValue;
    return x / SomeOtherValue;
}
```

At some point, this function overflows and returns a really large negative value. To try and track down exactly where this was happening, I added a cout statement so that the function looked like this:

```
float function(){
    float x = SomeValue;
    cout << x;
    return x / SomeOtherValue;
}
```

and it worked! Of course, I solved the problem altogether by using a double. But I'm curious as to why the function worked properly when I couted it. Is this typical, or could there be a bug somewhere else that I'm missing?

(If it's any help, the value stored in the float is just an integer value, and not a particularly big one. I just put it in a float to avoid casting.)

c++ floating-point cout cpu-registers floating-point-precision

Share
Improve this question
Follow

edited May 9, 2012 at 14:17

mskfisher
3,406 ● 4 ● 37 ● 49

asked Sep 8, 2008 at 3:02

Jason Baker
198k ● 138 ● 382 ● 520

5 Answers

Sorted by: Highest score (default)

19

Welcome to the wonderful world of floating point. The answer you get will likely depend on the floating point model you compiled the code with.

This happens because of the difference between the IEEE spec and the hardware the code is running on. Your CPU likely has 80 bit floating point registers that get use to

hold the 32-bit float value. This means that there is far more precision while the value stays in a register than when it is forced to a memory address (also known as 'homing' the register).

When you passed the value to cout the compiler had to write the floating point to memory, and this results in a lost of precision and interesting behaviour WRT overflow cases.

See the MSDN documentation on VC++ floating point switches. You could try compiling with /fp:strict and seeing what happens.

Share Improve this answer Follow

answered Sep 8, 2008 at 3:43

Rob Walker
47.4k ● 15 ● 100 ● 137

> There is also a GCC note for this at gcc.gnu.org/wiki/x87note Due to this wonderful behaviour, comparing floating point computations is also inherently broken, unless using pre-computed values. – hazzen Sep 8, 2008 at 19:57

3

Printing a value to cout should not change the value of the paramter in any way at all.

However, I have seen similar behaviour, adding debugging statements causes a change in the value. In those cases, and probably this one as well my guess was that the additional statements were causing the compiler's optimizer to behave differently, so generate different code for your function.

Adding the cout statement means that the vaue of x is used directly. Without it the optimizer could remove the variable, so changing the order of the calculation and therefore changing the answer.

Share Improve this answer Follow

answered Sep 8, 2008 at 3:20

David Dibben
18.7k ● 6 ● 43 ● 41

2

As an aside, it's always a good idea to declare immutable variables using `const`:

```
float function(){
    const float x = SomeValue;
    cout << x;
    return x / SomeOtherValue;
}
```

Among other things this will prevent you from unintentionally passing your variables to functions that may modify them via non-`const` references.

Share Improve this answer Follow

answered Sep 8, 2008 at 16:16

David Joyner

23.1k ● 4 ● 30 ● 33

1

cout causes a reference to the variable, which often will cause the compiler to force it to spill it to the stack.

Because it is a float, this likely causes its value to be truncated from the double or long double representation it would normally have.

Calling any function (non-inlined) that takes a pointer or reference to x should end up causing the same behavior, but if the compiler later gets smarter and learns to inline it, you'll be equally screwed :)

Share Improve this answer Follow

answered Aug 31, 2010 at 10:07

Joe

227 ● 1 ● 2

0

I dont think the cout has any effect on the variable, the problem would have to be somewhere else.

Share Improve this answer Follow

answered Sep 8, 2008 at 3:21

John Boker

83.7k ● 17 ● 100 ● 130