A noinline inline function? What sorcery is this?

devblogs.microsoft.com/oldnewthing/20200521-00

May 21, 2020

Raymond Chen

You can declare a noinline inline function.

```
void g();

// gcc
__attribute__((noinline)) inline void f()
{
    g();
}

// MSVC
__declspec(noinline) inline void f()
{
    g();
}

void tryme()
{
    f();
    f();
}
```

What sorcery is this, a function that is both inline and not-inline?

The two keywords are not contradictory because they describe different senses of the word "inline".

The C++ language keyword `inline` means "can be defined in multiple translation units without triggering an ODR violation." In other words, it lets you put the function definition in a header file that is included by multiple C++ files.

The function attribute/declaration specifier `noinline` means "do not inline this function during code generation." It is a directive to the optimizer not to perform inline substitution during code generation.

Historically, the `inline` C++ keyword was originally an optimizer hint, but optimizers were given permission to ignore it and make their own decisions about inline substitution during code generation. Nowadays, compilers pretty much ignore the optimization aspect of the `inline` keyword. The only thing that remains of the `inline` keyword is the ability to have multiple definitions without violating ODR.

You could say that the modern sense of the C++ keyword `inline` is "defined right here." It's a statement about the source code, not the object code.

In the above example, the function `f` is a noinline inline function. The `inline` keyword allows the definition of `f` to go into a header file that is consumed by multiple translation units. The noinline attribute/declaration specifier tells the optimizer to emit code for `f` and call it, rather than embedding the body of `f` into its call sites. The function `tryme` will call the function `f` twice, instead of optimizing out the call and just calling `g` twice.

Raymond Chen

Follow