In the function `fu` (above), the address of a dynamically allocated **`int`** is supplied to `fu` to be managed temporarily by a guard, `u`. An externally defined function, `g`, that may throw is then invoked. Provided `g` doesn't throw, the address of that supplied **`int`** object is returned; otherwise, it is deleted, and the exception is rethrown.

Under the covers, the compiler will implicitly generate a **`try`**/**`catch`** block to prepare for the case where `g` throws. For didactic purposes, we have rewritten the `fu` above as `fv`, stating all implicitly generated functionality explicitly:

```
int* fv(int* p)  // runtime optimizable if compiler knows g won't throw
{
    int* d_u = p;                    // #1 Initialize guard u.
    try  // implicitly generated try/catch block.
    {
        g();                         // #2 This call to g may throw.
    }
    catch (...)
    {
        delete d_u;  // Invoke guard u's destructor.
        throw;       // Rethrow currently in-flight exception.
    }
                                     // #3 Return the locally managed resource.
    int* retval = d_u;  // release(): Save value to be returned.
    d_u = 0;            // release(): Clear internal pointer value.
    delete d_u;         // Invoke guard u's destructor.
    return retval;      // Return allocated resource.
}
```

Let's now consider what sort of optimizations are possible along the hot path if the compiler knows that `g()` doesn't throw. For starters, the **`try`** and all code in the **`catch`** clause can be eliminated as there's no possibility of an exception propagating from the call to `g()`. Next, we can trace `p` through `d_u` and `retval` to see that it is always the value returned, so the return statement can be replaced with **`return`** `p`. The **`delete`** operator applied to `d_u` is always passed a **null pointer value**, which does nothing, and so that can be elided. Finally, while `d_u` is assigned to twice (first to `p` at the top of the function, then to `0` before the return), its value is never read, so both assignments and the local variable itself can be eliminated. That is, just by knowing that `g` doesn't throw the compiler can elide the catch block, both local variables, two assignments, and the invocation of the **`delete`** operator.

After the optimizer is done, a similarly didactic rendering of the equivalent code would be noticeably smaller: