WHAT DO WE MEAN WHEN WE SAY NOTHING AT ALL?

Kate Gregory
kate@gregcons.com
@gregcons
Who Do We Write For?

- The compiler?
  - A little
  - But compilers don’t care whether things are called Foo or UpdateOrders
- Ourselves right now?
  - Sure
- Ourselves later?
  - Definitely
- Others later?
  - Whether we like it or not
Writing Code is a Form of Communication

- We’re primarily communicating with the future
- With ourselves, and with some strangers
- We’re leaving what trail markers we can for those who follow us

- It’s not enough for the code to compile
- It’s not enough for the code to run without error
- It has to be understood, by humans
Reading Code

- What the heck does this do?
- Why is it doing that?
- Are we sure this actually works?
- What no-talent sad beginner wrote this?
  - Oh, right, me
- I bet that silly goose never considered ...
Well Written Code

- Has considered those questions and pre-answers them
- Expressive
- Transparent
- Communicative

/*
orders.cpp
Purpose: Calculates the total of all orders
Author: Jo Programmer
Last Modified: 4/10/06
*/
Roger Orr’s Favourite Code Snippet

{ }
Introducing
Saying Nothing Sometimes Means Nothing

class Holder
{
private:
    int number;
public:
    Holder(int i);
    Holder();
    void inc() { number++; }
    int getNumber() { return number; }
    std::string to_string();
};
Saying Nothing Sometimes Speaks Volumes

class Holder
{
    private:
        int number;
    public:
        explicit Holder(int i);
        Holder();
        void inc() { number++; }
        int getNumber() const { return number; }
        virtual std::string to_string() const;
};
Some things in C++ are paired with their opposites

- Operators
  - + -
  - */
  - * &
- Brackets
  - ()
  - {}
  - []
  - <>
- Keywords
  - if else
  - noexcept noexcept(false)
Most things don’t really have opposites

- break
- continue
- return
- UpdateBalance(x)
- while, for, switch
Let’s talk about these...

- virtual, override
- explicit
- const
  - mutable? Not always
- mutable
  - On a lambda
- public, private
  - In a struct vs in a class
- Ref-qualifiers on a function or on function parameters
- New C++ 17 attributes
Fallthrough

switch (i)
{
    case 1:
    case 2:
        msg += "case 1 or case 2. ";
    break;
    case 3:
        msg += "case 3 or ";
    case 4:
        msg += "case 4.";
    default:
        break;
}
Fallthrough

switch (i)
{
    case 1:
    case 2:
        msg += "case 1 or case 2. ";
    break;
    case 3:
        msg += "case 3 or ";
        //fallthrough
    case 4:
        msg += "case 4.";
    default:
        break;
}
Fallthrough

switch (i) {
    case 1:
    case 2:
        msg += "case 1 or case 2. ";
    break;
    case 3:
        msg += "case 3 or ";
        [[fallthrough]];
    case 4:
        msg += "case 4.";
    default:
        break;
}
Maybe Unused

```c
int j = FunctionWithSideEffects();
assert(j > 0);

[[maybe_unused]] int j = FunctionWithSideEffects();
assert(j > 0);
```
No Discard

```cpp
int getNumber() { return 42; }

auto num = getNumber();
getNumber();

[[nodiscard]] int getNumber() { return 42; }

auto num = getNumber();
getNumber();
```

discarding return value of function with 'nodiscard' attribute
How Can You Be Clearer About Intent?

- **Avoid defaults**
  - *In a class or struct, always include public: and private:*
    - Yes, even in a two-element struct like Point
  - *Add a return at the end of your void function*

- **Use those optional things**
  - *Mark overrides of virtual functions with override*
  - *Use noexcept(false) if you’ve thought about it*

- **Sure, they’re not needed, but using them carries meaning**
  - *Saves others guessing about whether you considered it*
How Can You Be Clearer About Intent?

- There is a limit to how verbose you can be
- We do not have these keywords
  - implicit
  - const(false)
  - nonvirtual
  - ByVal
- What should you do?

```cpp
// I know what I'm doing, don't change this
```

```cpp
// note: passing by value
```
Absence of a keyword means one of two things
- *I’ve thought about it and I don’t need keyword here*
- *I have never heard of keyword, or at least haven’t considered whether or not to use it here*

If you use it routinely and consistently throughout the codebase, readers can (possibly? With some certainty?) rule out that second option.

Comments?
- *Only for cases that deceive*

// I know this looks like it might be an override // of ship() but it’s actually a different signature
Optional Return Statements

```c
void Thimbule(int robbit) {
    robbit ++;
    if (robbit)
        return;
    robbit --;
    return;
}

void Sprial(int oob, int boo) {
    oob ++;
    while (true)
    {
        if (++oob > boo)
            return;
    }
```
Ranged For

for (auto emp : department)
{
    // ...
}

for (auto & emp : department)
{
    // ...
}

for (auto const & emp : department)
{
    // ...
}
Parameter Passing

- `Order createOrder(Customer c, OrderItem oi);`
  - Are you sure?
- `Order createOrder(Customer& c, OrderItem oi);`
  - Pass the order item by value, then move? Copy?
- `Order createOrder(Customer const & c, OrderItem oi);`
  - Oh, Customer objects don’t know their orders?
Omitting Parameter Names

- You can in the declaration
  - Compiler etc don’t care
  - Humans care, so don’t
- You also can in the definition
  - If it’s an unused parameter
  - (virtual function, api drift, whatever)
  - Suppresses compiler warning
  - Big signal to humans
- So, why not follow the same pattern in declaration?

```c
int DetermineTotalTaxes(int, int, int);
```
WHAT OTHER CHOICES CAN SPEAK VOLUMES?
Is A Raw Pointer Always A Non-owning Pointer?

- Does this code use smart pointers?
- Is there a lot of new and delete? Rule of 3 or 5?
  - Are there any destructors anywhere?
What Does & Mean? *?

- Is something passed by address or reference as non-const always changed?
- Is there any meta-meaning to passing by address vs by reference?
  - *Many style guides suggest pass-by-address to transfer ownership*
  - *This isn’t about what the compiler thinks*
  - *You have nothing in the code that mentions owning, yet maybe you’re speaking about owning anyway?*
Is A Traditional for Loop Always Doing Something Odd?

■ Why did I choose that loop?
  - *Does it touch every element?*
  - *Was there a reason not to use a ranged for?*

■ Isn’t there an algorithm for that?
  - *Is this something without a name we all know*
    ■ find, count, all_of, sort, ...
  - *If you use algorithms when you can, then your choice of a loop gets my attention*
Initializing

- If a constructor doesn’t set a member variable after : , perhaps:
  - There’s a nonstatic member initializer that does
  - It gets set in the body
    - Why?
    - It was forgotten when the member was added to the class
      - Bonus points: forgotten in only one of the 5 constructors

- What does it mean when I initialize something to its default value?
  - string s = "";
  - vector<Employee> department(0);
Could The Language Help Us?

- Should we add keywords or attributes? Would you use them?
  - `implicit  [or explicit(false)]`
  - `const(false)`
  - `nonvirtual`
  - `ByVal`

- Are you using `fallthrough`, `no_discard`, and `maybe_unused`?  
  - Why not?

April 2019, ACCU 2019
CALL TO ACTION
Communicate

- Clear code involves thinking about what you are telling the future reader
- Show them why you did this
- No puzzles or mysteries
- No chance to think you were foolish or ill-informed
What is Not in Your Code?

- Think about what you’re not including or doing
- The other ways you could have done this
- The other choices you could have made
- Can people learn from a seemingly arbitrary choice?
Nothingness

■ Can you express your choice without nothingness?
  - A little verbosity goes a long way

■ If the only way to express yourself is with nothingness, then fine, but make that nothingness speak
  - Context
  - Show your colours
Use Nothing In a Generous Way

- Give that future reader all they need
- Make sure your nothing speaks volumes
- Ensure that they will understand