Topics Discussed in Issue

Categorizing topics discussed.

Original Issue posted

- We have a **style rule** that **case blocks** always have **braces**:
  - Always use **braces** for **conditional**, **switch**, and **loop statements**, even when the body is a single statement.
  - but **we deviate** from this significantly in **explorer**: [list of files]
  - We should either **fix** all of these (ideally via tooling) and **add automated checks** (such as the above perl script) or **remove the style rule**.
- **explorer** label added
  - referenced merge. https://github.com/carbon-language/carbon-lang/pull/2250. This reference appears to show a recent example of a successful pull request that contains **explorer** files using case blocks without braces.

Issue Comments

- We have **style rules** that we **don't automatically check**
  - **braces** for **if** are an obvious and similar case, but there are also things like
    - **casing**
    - **comments on closing namespaces**
    - etc.
  - Hm, looking at the **style guide** again, I don't find evidence that we do have this **rule**. It says to always use **braces** for a **switch**, but for case labels it only says:
    - Within a **switch** statement, use **braces** after a **case** label when necessary to create a **scope** for a **variable**.
    - That doesn't match my memory -- my recollection was that the interpretation in this comment was accurate, that we had said we always wanted **braces** after **case** or default in C++ code -- but it looks like maybe we never made this **decision**?
      - comment. https://github.com/carbon-language/carbon-lang/pull/2250#pullrequestreview-1129128004
      - Looking again, yes, I agree. I can retract if you prefer to avoid this.
        - Note, this will mean we aren't really trying out the **syntax** we're going to require for **match**.
      - I'm confused why it mentions **switch** in the prior sentence; maybe remove that if that's the path? (there's no such thing as a **brace-less switch**, only **brace-less case**, right?)
        - **Braceless switch** is possible, for example in this version of Duff's device: [code block]

- Okay, I take it back, but I still don't see why we need to call that out in **style**. To me, it's not like adding **braces** would make me feel like that's an encouraged **code structure**, even if it's sometimes appropriate. To hearken back to the **style guide**, I'd think that's reminiscent of the **goto** commentary (and lack thereof).
    - A simple perspective from the carbon-lang **design** may help show why **style** could be strengthened for **case**. **case** is a more advanced form of a mathematical resolution, implying that **precedence** may apply.
        - If **'most developers'** are unable to habitually avoid **scoping** issues inside **switch**, then **style**, or **clang**, may need to give a reminder.
        - Is carbon-lang ready to claim that "it's reasonable to expect **most developers** to understand the **precedence** without [**braces**]?"
            - docs/design/expressions/README.md
                - "**Expressions** are **interpreted** based on a **partial precedence ordering**. **Expression components** which lack a relative ordering must be disambiguated by the developer, for example by adding **parentheses**; otherwise, the **expression** will be invalid due to ambiguity. **Precedence orderings** will only be added when it's reasonable to expect **most developers** to **understand** the **precedence** without **parentheses**."
        - We use **-Wimplicit-fallthrough** and zygoloid's example won't compile as-is. And in C++, **braces** actually don't do much regarding **fallthrough**, they're just a visual reminder.
            - Compile explorer. https://compiler-explorer.com/z/ePa1Gx6oo
        - In Carbon, although the early thoughts on **match** are thin, I think **braces** are required and **fallthrough** is **explicit**. I'm not sure how something like what zygoloid drew up would be written.
            - early thoughts on **match**. https://github.com/carbon-language/carbon-lang/tree/trunk/docs%2Fdesign
        - So if we're trying to make **C++ code mirror what we expect Carbon code to look like**, we **can't change fallthrough behavior**: **break will still be there**. As a consequence, ***braces*** *have less utility and are more just repetitive when there's no* ***variable*** *being* ***scoped***. I think that's where we're just encountering habit and familiarity with C++; **braces** may not be too helpful beyond making the code look incrementally more Carbon-like.
            - New Carbon developers **without C++ experience**, won't remark if **case** requires **braces** or not. **Style**

doesn't currently require **braces** for **case**. The **code inconsistencies** in **explorer** concerning **case** may be **irrelevant**.
        - Other points raised in this **issue** may need to be **categorized** and **resolved**.
        - **C++ developers** would like seeing Carbon handle the **example by zygoloid**.
        - Developers **not familiar with C++** will not have understanding of zygoloids elegant **recursion algorithm**. **Lists** should be part of Carbon. **Recursion** and **references** are handled generally by the Carbon design.
        - I am a relatively new contributor to Carbon. I can understand why my **pull requests** would be compared against current **style** docs, while **frequent contributors** may not be.
        - Identifying **style compliance** weak points in the pull request **review process** may be the easiest route to progress Carbon incrementally. There appears to be a divide between new and regular contributors.
        - **Styling** more consistently upon **review** should decrease this **contributor gap**. Making changes to the **review process** incrementally, may allow new **design and style issues**, to arise and **resolve** steadily.
            - **Modern and evolving**
                - Solid language foundations that are **easy to learn**, especially if you have used C++
- Why is **switch** different in requiring all cases be addressed with **automation** (or **removing the rule**), can you please elaborate?
    - The rule for **switch** is missed a lot in checked-in code, which I take as evidence that it isn't being caught consistently in **code review**. By contrast, I think our **casing rules** are broadly **being followed**, and **comments on closing namespaces** are enforced by our **clang-format checks**. If we don't have some kind of **automated** checking, I expect we'll continue to expand our set of **cases** without **braces**. So I think our realistic options are either that we have **automated checks** or that we accept that we'll **accumulate more** of these over time.
        - I think that **style rules** that are frequently **not followed** are usually a net negative -- they create **cost** in churn, they create friction in

code review, frustration ("why can't I write this this way when nearby code does the same?") and local inconsistency, a risk of **unequal treatment** (if **frequent contributors** don't have their patches checked for these issues but **new contributors** do), and they don't provide the gain in **global consistency** that we're looking for from a **style rule**. So I would prefer that we do **not retain style rules** that we think it's not worth consistently enforcing.

- In addition to @zygoloid 's approach, It won't **cost** a lot if the **fixing** process is ensured to come after the formatter. this is the idea: [code block]

Categorized Topics Meant for Resolution

- **style rule**
    - **braces**
        - **conditional**
            - **if**
        - **switch**
            - **case blocks**
                - **variable scope**
                    - **-Wimplicit-fallthrough**
        - **loop statements**
        - **casing**
            - **being followed**
        - **comments on closing namespaces**
            - **clang-format checks**
    - **we deviate**
        - **explorer**
            - **fix**
        - **not followed**
            - **accumulate more**
            - **cost**
            - **not retain style rules**
    - **add automated checks**
        - **clang**
    - **remove the style rule**
    - **don't automatically check**
        - *The **code inconsistencies** in **explorer** concerning **case** may be **irrelevant***
    - **leads decision**
    - **code structure**
    - **pull requests**
        - **frequent contributors**
        - **style compliance**
        - **review process**

            - **contributor gap**
                - **unequal treatment**
                - **new contributors**
                - **global consistency**
            - **Modern and evolving**
                - **easy to learn**
- **match**
    - **syntax**
    - **braces**
    - **fallthrough**
        - **explicit**
- **goto**
- **design**
    - **Expressions**
        - **interpreted**
            - **partial precedence ordering**
                - **Expression components**
                - **most developers understand**
                - **parentheses**
                    - **fallthrough**
- **C++ code mirror what we expect Carbon code to look like**
    - **can't change fallthrough behavior**
        - **break will still be there**
        - **example by zygoloid**
            - **Recursion**
            - **references**
    - ***braces** have less utility and are more just repetitive when there's no **variable** being **scoped***
- ***Carbon without C++ experience***
    - ***Lists***

Modern and evolving

Solid language foundations that are easy to learn, especially if you have used C++