Low-Level Arguments: Code Companion

Reference code for the Low-Level Arguments lecture. Sections correspond to the lecture document.

Section 1: The Anatomy of Low-Level Arguments

```rust
/// A collection of "low level" arguments.
///
/// The "low level" here is meant to constrain this type to be as close to the
/// actual CLI flags and arguments as possible.
#[derive(Debug, Default)]
pub(crate) struct LowArgs {
    // Essential arguments.
    pub(crate) special: Option<SpecialMode>,
    pub(crate) mode: Mode,
    pub(crate) positional: Vec<OsString>,
    pub(crate) patterns: Vec<PatternSource>,

    // Everything else, sorted lexicographically.
    pub(crate) binary: BinaryMode,
    pub(crate) boundary: Option<BoundaryMode>,
    pub(crate) buffer: BufferMode,
    pub(crate) byte_offset: bool,
    pub(crate) case: CaseMode,
    pub(crate) color: ColorChoice,
    pub(crate) colors: Vec<UserColorSpec>,
    pub(crate) column: Option<bool>,
    pub(crate) context: ContextMode,
    // ... many more fields follow
    pub(crate) encoding: EncodingMode,
    pub(crate) engine: EngineChoice,
    pub(crate) fixed_strings: bool,
    pub(crate) follow: bool,
    pub(crate) globs: Vec<String>,
    pub(crate) hidden: bool,
    pub(crate) max_count: Option<u64>,
    pub(crate) multiline: bool,
    pub(crate) threads: Option<usize>,
    pub(crate) type_changes: Vec<TypeChange>,
    pub(crate) unrestricted: usize,
    pub(crate) vimgrep: bool,
    pub(crate) with_filename: Option<bool>,
}
```

The `pub(crate)` visibility allows any module within the crate to read/write fields directly, while hiding them from external consumers. The `Default` derive enables incremental construction during parsing.

Section 2: Special Modes and Short-Circuiting

```rust
/// A "special" mode that supercedes everything else.
///
/// When one of these modes is present, it overrides everything else and causes
/// ripgrep to short-circuit.
#[derive(Clone, Copy, Debug, Eq, PartialEq)]
pub(crate) enum SpecialMode {
    /// Condensed help output (-h flag)
    HelpShort,
    /// Verbose help output (--help flag)
    HelpLong,
    /// Condensed version (ripgrep x.y.z)
    VersionShort,
    /// Verbose version with build features
    VersionLong,
    /// PCRE2's version information
    VersionPCRE2,
}
```

The enum is `Copy` because it's small and frequently passed around. The distinction between Short/Long variants enables Unix-style `-h` vs `--help` behavior.

Section 3: Operational Modes and Override Semantics

```rust
#[derive(Clone, Copy, Debug, Eq, PartialEq)]
pub(crate) enum Mode {
    /// ripgrep will execute a search of some kind.
    Search(SearchMode),
    /// Show files that *would* be searched without searching.
    Files,
    /// List all file type definitions configured.
    Types,
    /// Generate man pages, completions, etc.
    Generate(GenerateMode),
}

impl Default for Mode {
    fn default() -> Mode {
        Mode::Search(SearchMode::Standard)
    }
}

impl Mode {
    /// Update this mode while implementing override semantics.
    pub(crate) fn update(&mut self, new: Mode) {
        match *self {
            // Search mode can be overridden by anything
            Mode::Search(_) => *self = new,
            _ => {
                // Non-search modes can override each other,
                // but search modes cannot override non-search modes.
                // Example: `--files -l` stays as Mode::Files
                if !matches!(*self, Mode::Search(_)) {
                    *self = new;
                }
            }
        }
    }
}
```

The `update` method's logic ensures explicit mode flags (like `--files`) take precedence over implicit search mode flags from config files.

Section 4: Search Mode Variations

```rust
/// The kind of search that ripgrep is going to perform.
#[derive(Clone, Copy, Debug, Eq, PartialEq)]
pub(crate) enum SearchMode {
    /// Default mode - print matching lines
    /// No explicit flag; use --no-json to reset to this
    Standard,
    /// Show files containing at least one match (-l)
    FilesWithMatches,
    /// Show files that don't contain any matches (--files-without-match)
    FilesWithoutMatch,
    /// Show files with match count per file (-c)
    Count,
    /// Show files with total match count (--count-matches)
    CountMatches,
    /// Print matches in JSON lines format (--json)
    JSON,
}
```

`Count` vs `CountMatches`: a file with one line containing three matches contributes 1 to `Count` but 3 to `CountMatches`.

Section 5: Binary Data Handling

```rust
/// Indicates how ripgrep should treat binary data.
#[derive(Debug, Default, Eq, PartialEq)]
pub(crate) enum BinaryMode {
    /// Context-dependent: explicit files get SearchAndSuppress,
    /// discovered files get aggressive binary skipping.
    #[default]
    Auto,

    /// Search binary files but suppress matches and warn.
    /// NUL bytes replaced with line terminators to prevent
    /// memory exhaustion from long binary "lines".
    SearchAndSuppress,

    /// Treat all files as plain text. No skipping, no NUL replacement.
    AsText,
}
```

The `#[default]` attribute on `Auto` eliminates the need for a separate `Default` impl. The NUL-to-newline replacement in `SearchAndSuppress` is a crucial memory safety heuristic.

Section 6: Context Mode and Precedence

```rust
/// Line context options for output.
#[derive(Debug, Eq, PartialEq)]
pub(crate) enum ContextMode {
    /// Print all lines (unbounded context)
    Passthru,
    /// Show specific number of lines before/after matches
    Limited(ContextModeLimited),
}

/// Tracks before/after/both separately for correct precedence.
#[derive(Debug, Default, Eq, PartialEq)]
pub(crate) struct ContextModeLimited {
    before: Option<usize>,  // -B flag
    after: Option<usize>,   // -A flag
    both: Option<usize>,    // -C flag
}

impl ContextModeLimited {
    /// Returns (before, after) with correct precedence:
    /// -A/-B always override -C regardless of order.
    pub(crate) fn get(&self) -> (usize, usize) {
        // Start with -C value or (0, 0)
        let (mut before, mut after) =
            self.both.map(|lines| (lines, lines)).unwrap_or((0, 0));

        // -A and -B always win over -C
        if let Some(lines) = self.before {
            before = lines;
        }
        if let Some(lines) = self.after {
            after = lines;
        }
        (before, after)
    }
}
```

The three-field design with deferred resolution in `get()` enables order-independent flag processing where `-A`/`-B` always override `-C`.

Quick Reference

Mode Hierarchy

```
SpecialMode (highest priority - short-circuits everything)
    └── HelpShort, HelpLong, VersionShort, VersionLong, VersionPCRE2

Mode (operational mode)
    ├── Search(SearchMode) - default, can be overridden
    ├── Files - overrides search, can be overridden by other non-search
    ├── Types - overrides search
    └── Generate(GenerateMode) - overrides search
```

Key Types Summary

Type	Purpose	Default
`LowArgs`	All CLI arguments	`Default::default()`
`SpecialMode`	Short-circuit operations	`None`
`Mode`	What ripgrep does	`Search(Standard)`
`SearchMode`	How to report matches	`Standard`
`BinaryMode`	Binary file handling	`Auto`
`ContextMode`	Lines around matches	`Limited(0, 0)`
`ColorChoice`	Output coloring	`Auto`
`EncodingMode`	Text encoding	`Auto`
`EngineChoice`	Regex engine	`Default`

Visibility Pattern

```rust
pub(crate) struct LowArgs {
    pub(crate) field: Type,  // Crate-internal access only
}
```

All fields use `pub(crate)` - accessible within the crate, hidden from external consumers.