ripgrep crates/searcher/src/lib.rs: Code Companion

Reference code for the Searcher Overview lecture. Sections correspond to the lecture document.

Section 1: The Crate Documentation as Architecture Document

```
/*!
This crate provides an implementation of line oriented search, with optional
support for multi-line search.

# Brief overview

The principle type in this crate is a ['Searcher'], which can be configured
and built by a ['SearcherBuilder']. A 'Searcher' is responsible for reading
bytes from a source (e.g., a file), executing a search of those bytes using
a 'Matcher' (e.g., a regex) and then reporting the results of that search to
a ['Sink'] (e.g., stdout). The 'Searcher' itself is principally responsible
for managing the consumption of bytes from a source and applying a 'Matcher'
over those bytes in an efficient way. The 'Searcher' is also responsible for
inverting a search, counting lines, reporting contextual lines, detecting
binary data and even deciding whether or not to use memory maps.
*/
```

syntax creates links to types in generated documentation. The [Searcher]

Section 2: Public API Design Through Re-exports

The crate:: prefix refers to the current crate's root. Users import these types directly from grep_searcher:: without knowing the internal module structure.

Section 3: The Builder Pattern Ecosystem

```
pub use crate::{
    searcher::{
        // The builder creates configured Searcher instances
        SearcherBuilder,
        // The built type with all configuration applied
        Searcher,
    },
    // ...
};
```

The builder pattern pairs visible here: SearcherBuilder constructs Searcher. This mirrors RegexMatcherBuilder / RegexMatcher from the regex crate.

Section 4: The Sink Abstraction

The supporting types (SinkContext, SinkMatch, etc.) carry data to the Sink trait's callback methods. This is a push-based streaming API rather than a pull-based collection.

Section 5: Convenience Sinks and Closures

```
// From the documentation example - using the UTF8 convenience sink
use grep_searcher::sinks::UTF8;

// The closure signature: (line_number, line_content) -> Result<bool, Error>
// Return Ok(true) to continue, Ok(false) to stop early
Searcher::new().search_slice(&matcher, SHERLOCK, UTF8(|lnum, line| {
    // Process the match...
    Ok(true) // Continue searching
}))?;
```

The UTF8 sink wraps a closure, handling byte-to-string conversion. The bool return value provides early termination control—useful for "find first match" scenarios.

Section 6: The Documentation Example Dissected

Three crates compose: grep_matcher defines the interface, grep_regex provides the implementation, grep_searcher orchestrates the search. The nested matcher.find() call extracts precise match text from matching lines.

Section 7: Module Organization and Visibility

```
// Macro definitions available to subsequent modules
#[macro_use]
mod macros;

// Internal modules (no pub = private to crate)
mod line_buffer; // Buffered reading implementation
mod lines; // Line iteration utilities
mod searcher; // Core Searcher type and builder
mod sink; // Sink trait and implementations

// Test utilities only compiled in test builds
#[cfg(test)]
mod testutil;
```

The #[macro_use] attribute must appear before modules that use the macros. The #[cfg(test)] conditional compilation excludes test utilities from release builds.

Quick Reference

Architecture Flow

```
Source (file/stdin/bytes)

$\delta$
Searcher (orchestration)

$\delta$
Matcher (pattern matching)

$\delta$
Sink (result handling)
```

Key Types

Туре	
Searcher	
SearcherBuilder	Configures and constructs Searcher
Sink	
SinkMatch	
SinkContext	
UTF8	

Re-export Pattern

```
// Internal structure:
mod internal_module {
    pub struct SomeType;
}

// Public API (in lib.rs):
pub use crate::internal_module::SomeType;

// User sees:
use my_crate::SomeType; // Clean, flat path
```

Closure Sink Return Values

	Effect
Ok(true)	
Ok(false)	
Err(e)	