# Ripgrep search.rs: Code Companion

Reference code for the search.rs lecture. Sections correspond to the lecture document.

## Section 1: The Config Struct

#### Config field purposes

	Туре	
preprocessor	Option <pathbuf></pathbuf>	
preprocessor_globs	Override	Which files to preprocess

search_zip	bool	
binary_implicit	BinaryDetection	
binary_explicit	BinaryDetection	

Section 2: The Builder Pattern

#### The .then() pattern:

```
// Option<T> if condition is true, None otherwise
let maybe_builder = condition.then(|| expensive_construction());
```

### Section 3: The Builder Methods

}

#### **BinaryDetection modes:**

```
BinaryDetection::none() // No detection, search everything
BinaryDetection::quit() // Stop searching if binary detected
BinaryDetection::convert() // Replace NUL bytes with replacement char
```

### Section 4: SearchResult

```
/// The result of executing a search.
#[derive(Clone, Debug, Default)]
pub(crate) struct SearchResult {
    has_match: bool,
    stats: Option<grep::printer::Stats>,
}

impl SearchResult {
    /// Whether the search found a match or not.
    pub(crate) fn has_match(&self) -> bool {
        self.has_match
    }

    /// Return aggregate search statistics for a single search, if available.
    ///
    /// It can be expensive to compute statistics, so these are only present
    /// if explicitly enabled in the printer provided by the caller.
    pub(crate) fn stats(&self) -> Option<&grep::printer::Stats> {
        self.stats.as_ref()
    }
}
```

#### Exit code mapping:

```
// In main.rs
if search_result.has_match() {
    ExitCode::from(0) // Matches found
} else {
    ExitCode::from(1) // No matches
}
```

### Section 5: PatternMatcher Enum

```
/// The pattern matcher used by a search worker.
#[derive(Clone, Debug)]
pub(crate) enum PatternMatcher {
    RustRegex(grep::regex::RegexMatcher),
    #[cfg(feature = "pcre2")]
    PCRE2(grep::pcre2::RegexMatcher),
}
```

#### **Conditional compilation:**

```
// When pcre2 feature is disabled, this variant doesn't exist
#[cfg(feature = "pcre2")]
PCRE2(grep::pcre2::RegexMatcher),

// Usage in match expressions:
match self.matcher {
   RustRegex(ref m) => /* ... */,
   #[cfg(feature = "pcre2")]
   PCRE2(ref m) => /* ... */,
}
```

#### Why enum over trait object:

```
// Enum approach (used here):
match matcher { RustRegex(m) => m.search(), PCRE2(m) => m.search() }
// Pro: inlinable, no vtable
// Con: closed set of variants

// Trait object approach:
dyn Matcher
// Pro: open to extension
// Con: virtual dispatch overhead
```

### Section 6: Printer Fnum

#### Printer types by mode:

	Printer Type	
-c		Match counts
-1		
json	JSON	

Section 7: The SearchWorker Struct

```
/// A worker for executing searches.
///
/// It is intended for a single worker to execute many searches, and is
/// generally intended to be used from a single thread.
#[derive(Clone, Debug)]
pub(crate) struct SearchWorker<W> {
    config: Config,
    command_builder: grep::cli::CommandReaderBuilder,
    decomp_builder: Option<grep::cli::DecompressionReaderBuilder>,
    matcher: PatternMatcher,
    searcher: grep::searcher::Searcher,
    printer: Printer<W>,
}
```

#### **Component responsibilities**

Component	
matcher	
searcher	
printer	Output formatting
command_builder	
decomp_builder	

### Section 8: The Search Method

Routing priority: 1. stdin  $\rightarrow$  search\_reader (can't preprocess or mmap) 2. preprocessor match  $\rightarrow$  search\_preprocessor (user override) 3. decompression match  $\rightarrow$  search\_decompress (transparent) 4. otherwise  $\rightarrow$  search\_path (fast path with mmap)

## Section 9: Helper Predicates

```
impl<W: WriteColor> SearchWorker<W> {
    /// Returns true if and only if the given file path should be
    /// decompressed before searching.
    fn should_decompress(&self, path: &Path) -> bool {
        self.decomp_builder.as_ref().is_some_and(|decomp_builder| {
            decomp_builder.get_matcher().has_command(path)
        })
    }

/// Returns true if and only if the given file path should be run through
/// the preprocessor.
fn should_preprocess(&self, path: &Path) -> bool {
        // No preprocessor configured
        if !self.config.preprocessor.is_some() {
            return false;
        }
        // No globs = preprocess everything
        if self.config.preprocessor_globs.is_empty() {
            return true;
        }
        // Check if path matches globs (negation-aware)
        !self.config.preprocessor_globs.matched(path, false).is_ignore()
    }
}
```

#### The is\_some\_and pattern (Rust 1.70+):

```
// Old way
option.map(|x| predicate(x)).unwrap_or(false)

// New way
option.is_some_and(|x| predicate(x))
```

## Section 10: Preprocessor Search

#### Error handling pattern:

```
// Search might fail
let result = self.search_reader(path, &mut rdr);
// Close might fail (check after search completes)
let close_result = rdr.close();
// Return first error encountered
let search_result = result?;
close_result?;
Ok(search_result)
```

## Section 11: Decompression Search

```
/// Attempt to decompress the data at the given file path and search the
/// result.
fn search_decompress(&mut self, path: &Path) -> io::Result<SearchResult> {
    let Some(ref decomp_builder) = self.decomp_builder else {
        return self.search_path(path);
    };

    let mut rdr = decomp_builder.build(path)?;
    let result = self.search_reader(path, &mut rdr);
    let close_result = rdr.close();
    let search_result = result?;
    close_result?;
    Ok(search_result)
}
```

```
Supported formats (via grep-cli): - gzip (.gz) - bzip2 (.bz2) - xz (.xz) - lz4 (.lz4) - lzma (.lzma) - zstd (.zst)
```

### Section 12: Direct File Search

```
/// Search the contents of the given file path.
fn search_path(&mut self, path: &Path) -> io::Result<SearchResult> {
   use self::PatternMatcher::*;

   let (searcher, printer) = (&mut self.searcher, &mut self.printer);
   match self.matcher {
      RustRegex(ref m) => search_path(m, searcher, printer, path),
      #[cfg(feature = "pcre2")]
      PCRE2(ref m) => search_path(m, searcher, printer, path),
   }
}
```

Why search\_path is the fast path: - Memory mapping possible (avoids copying to userspace) - No child process overhead - No decompression overhead - Direct syscall-level optimizations available

## Section 13: Reader-Based Search

```
/// Executes a search on the given reader, which may or may not correspond
/// directly to the contents of the given file path.
///
/// Generally speaking, this method should only be used when there is no
/// other choice. Searching via `search_path` provides more opportunities
/// for optimizations (such as memory maps).
fn search_reader<R: io::Read>(
    &mut self,
    path: &Path,
    rdr: &mut R,
) -> io::Result<SearchResult> {
        use self::PatternMatcher::*;

    let (searcher, printer) = (&mut self.searcher, &mut self.printer);
    match self.matcher {
        RustRegex(ref m) => search_reader(m, searcher, printer, path, rdr),
        #[cfg(feature = "pcre2")]
        PCRE2(ref m) => search_reader(m, searcher, printer, path, rdr),
    }
}
```

When search\_reader is used: - stdin (can't be mmapped) - Preprocessor output (pipe, not file) - Decompressor output (pipe, not file)

Section 14: The Free Functions

#### The Sink pattern:

```
// Printer creates a sink that knows how to handle matches
let mut sink = printer.sink_with_path(&matcher, path);

// Searcher produces match events, sink handles them
searcher.search_path(&matcher, path, &mut sink)?;

// Sink accumulates results
sink.has_match() // Did anything match?
sink.stats() // Aggregate statistics
```

## Quick Reference: Key Types

```
// From this module
struct SearchWorkerBuilder { config. command_builder }
struct SearchWorker=\( \) { config. matcher, searcher, printer, ... }
struct SearchResult { has_match, stats }
enum PatternMatcher { RustRegex(...), PCRE2(...) }
enum Printer=\( \) { Standard(...), Summary(...), JSON(...) }

// From grep-searcher
struct Searcher; // File reading + line iteration
enum BinaryDetection { None, Quit, Convert }

// From grep-printer
struct Standard=\( \) // Line-by-line printer
struct Summary=\( \) // Aggregate printer
struct JSON=\( \) // Structured printer
struct Stats; // Match statistics

// From grep-matcher (trait)
trait Matcher { fn find_iter(...) }

// From grep-cli
struct CommandReaderBuilder; // Runs external commands
struct DecompressionReaderBuilder; // Runs decompressors
```

## Data Flow Summary