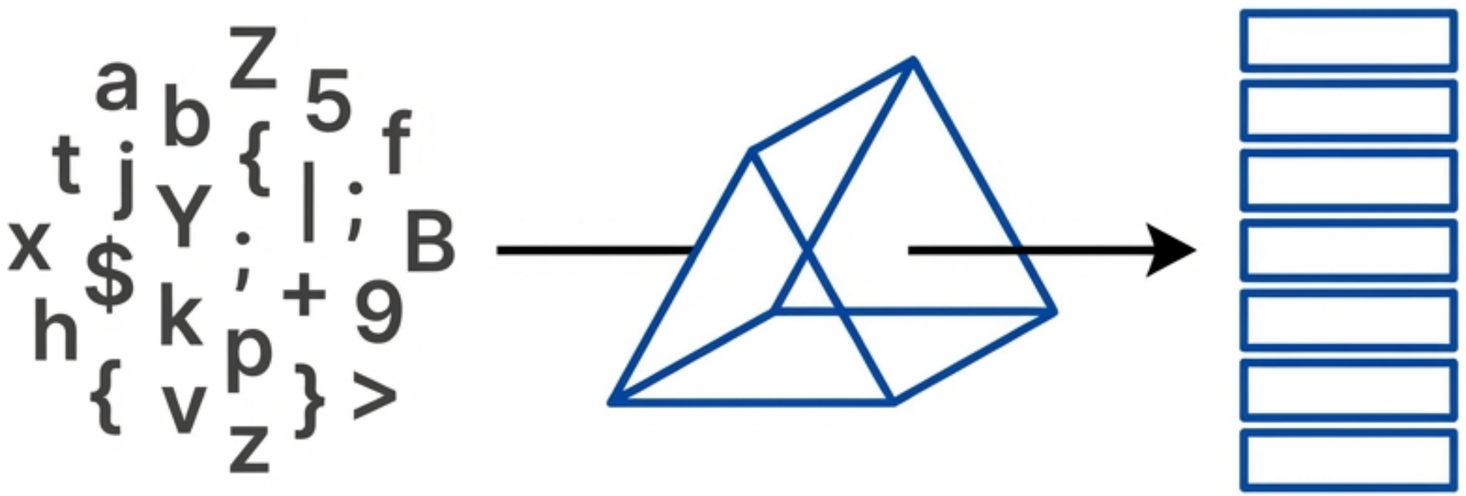
Building a C Lexical Analyzer

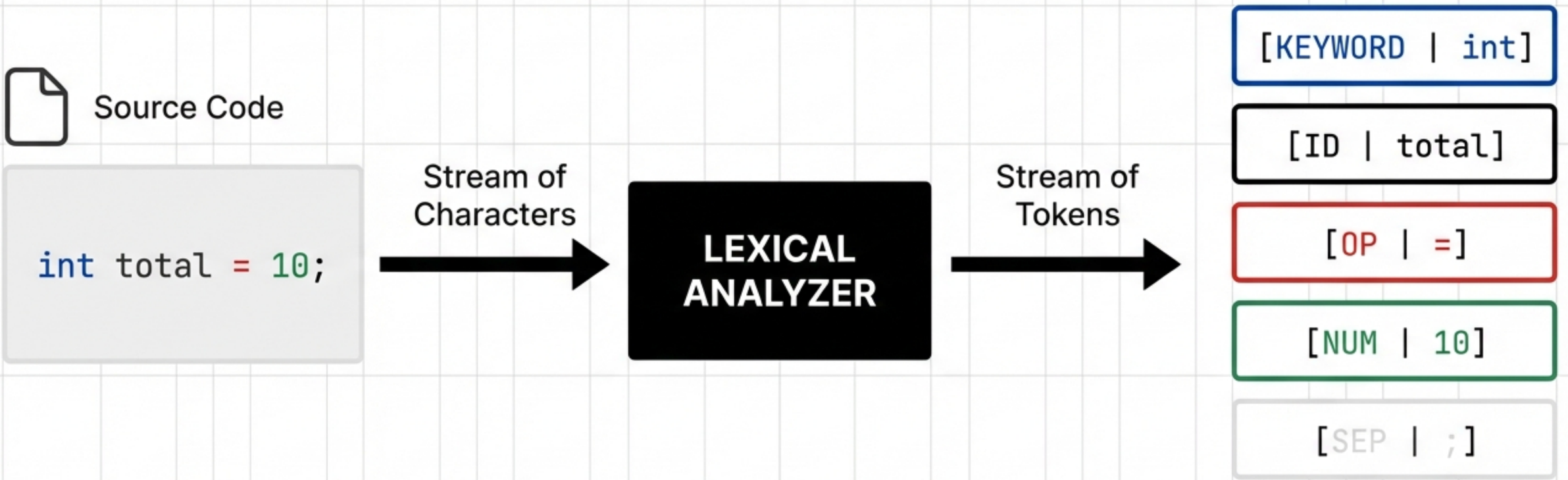
Theory, Implementation, and Execution



Sapandeep Singh Sandhu
ID: J4837
B.Tech Coursework

The Role: From Characters to Tokens

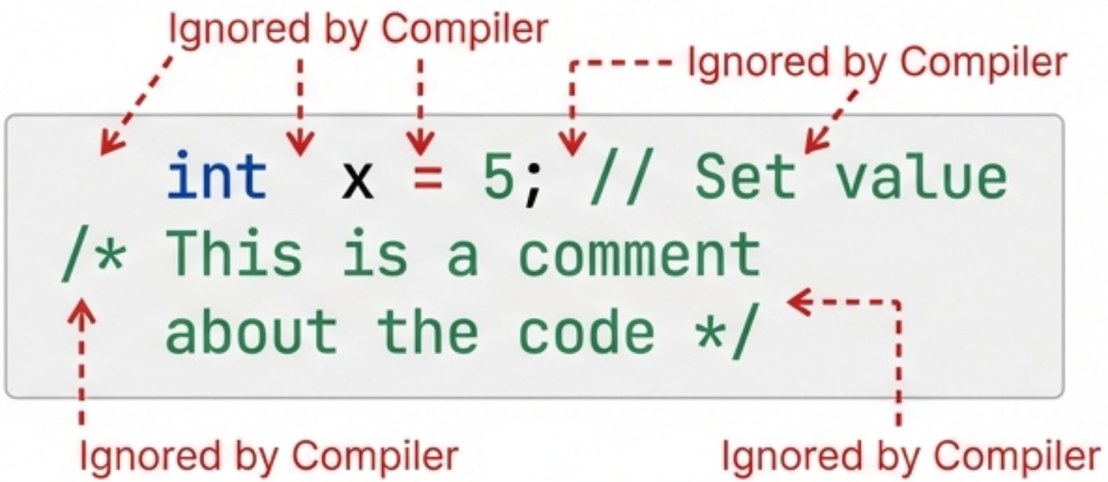
A Lexical Analyzer (Scanner) reads raw source code and groups characters into meaningful units called **Tokens**.



Why Do We Need Lexical Analysis?

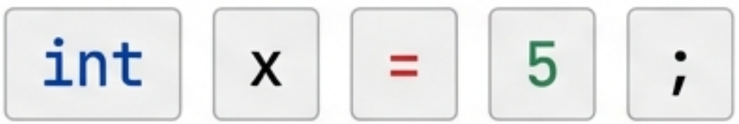
Filtering Noise to Simplify the Parser

The Problem (Noise)



Whitespace and comments are for humans. They do not affect program logic but clutter the input stream.

The Solution (Signal)



Efficiency & Simplification. The scanner acts as a filter, removing non-executable metadata so the parser only sees the logic.

The Toolkit: Standard C Libraries

Core dependencies used in implementation

<stdio.h>

The Eyes

Handles File I/O. Reads the source file one character at a time.

Key Functions:
`fgetc()`,
`ungetc()`,
`fopen()`
`#0047AB`

<ctype.h>

The Brain

Classifies characters. Determines if a char is a letter, digit, or symbol.

Key Functions:
`isalpha()`,
`isdigit()`,
`isspace()`
`#0047AB`

<string.h>

The Hands

Manipulates text buffers. Checks if a word matches a keyword.

Key Functions:
`strcmp()`,
`strncpy()`,
`strlen()`
`#0047AB`

<stdlib.h>

Utilities

General purpose definitions and memory management.

The Blueprint: Data Structures

Defining the Token Object

Enum: The Token Types

```
typedef enum {
    TOK_EOF, TOK_KEYWORD,
    TOK_IDENTIFIER, TOK_INT,
    TOK_FLOAT, TOK_STRING,
    TOK_OPERATOR, TOK_SEPARATOR
} TokenType;
```

Struct: The Token Object

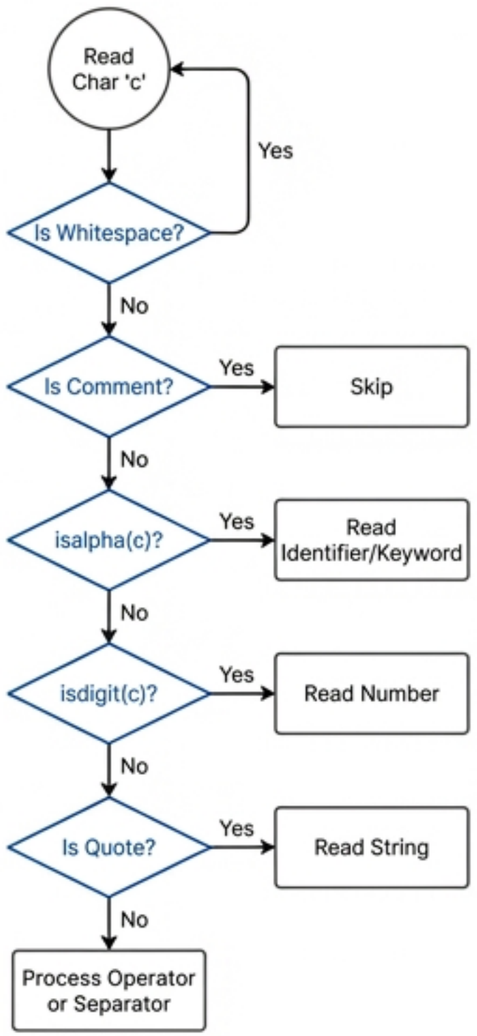
```
typedef struct {
    TokenType type;
    char lexeme[256];
    int line;
    int col;
} Token;
```

Holds the actual text content

Metadata for Error Tracking

Control Flow: The `next_token` Function

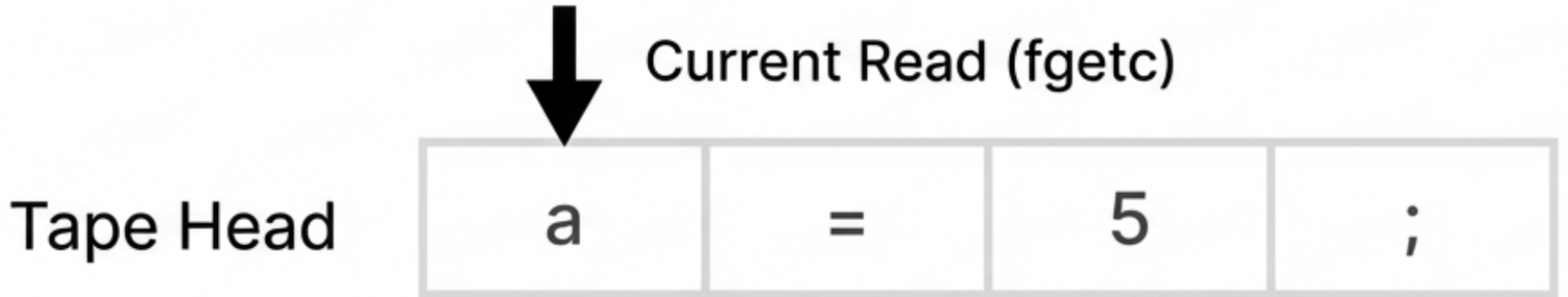
The decision tree for processing characters




```
// Simplified Loop
skip_whitespace();
c = read_char();
if (isalpha(c)) return read_identifier();
```

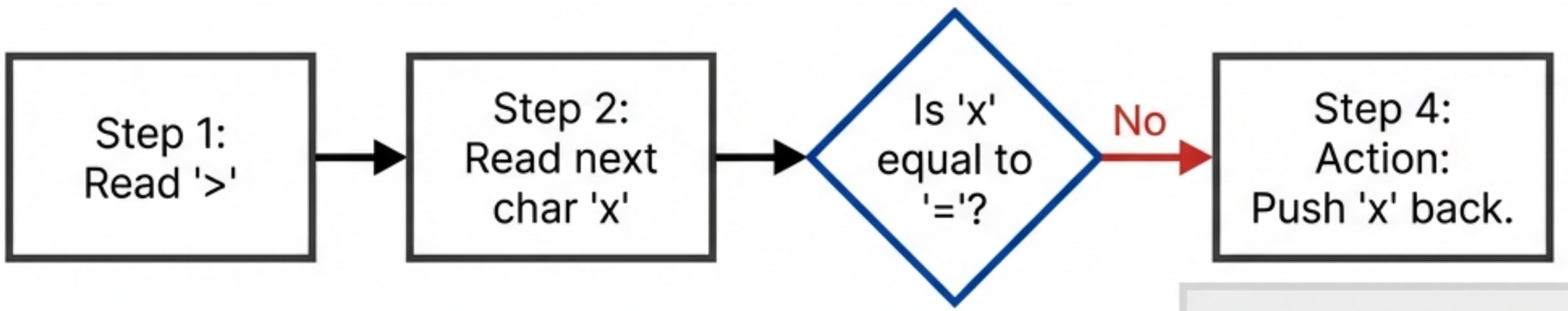
Reading Strategy: The Lookahead

Handling file pointers and character tracking



Global variables g_line and g_col track the cursor position. Newlines reset the column count.

“Unread” Concept




```
ungetc(c, fp);
```

We use ungetc() to push a character back into the stream if we read too far (e.g., distinguishing '>' from '>=').

Implementation: Skipping Noise

Filtering whitespace and comments

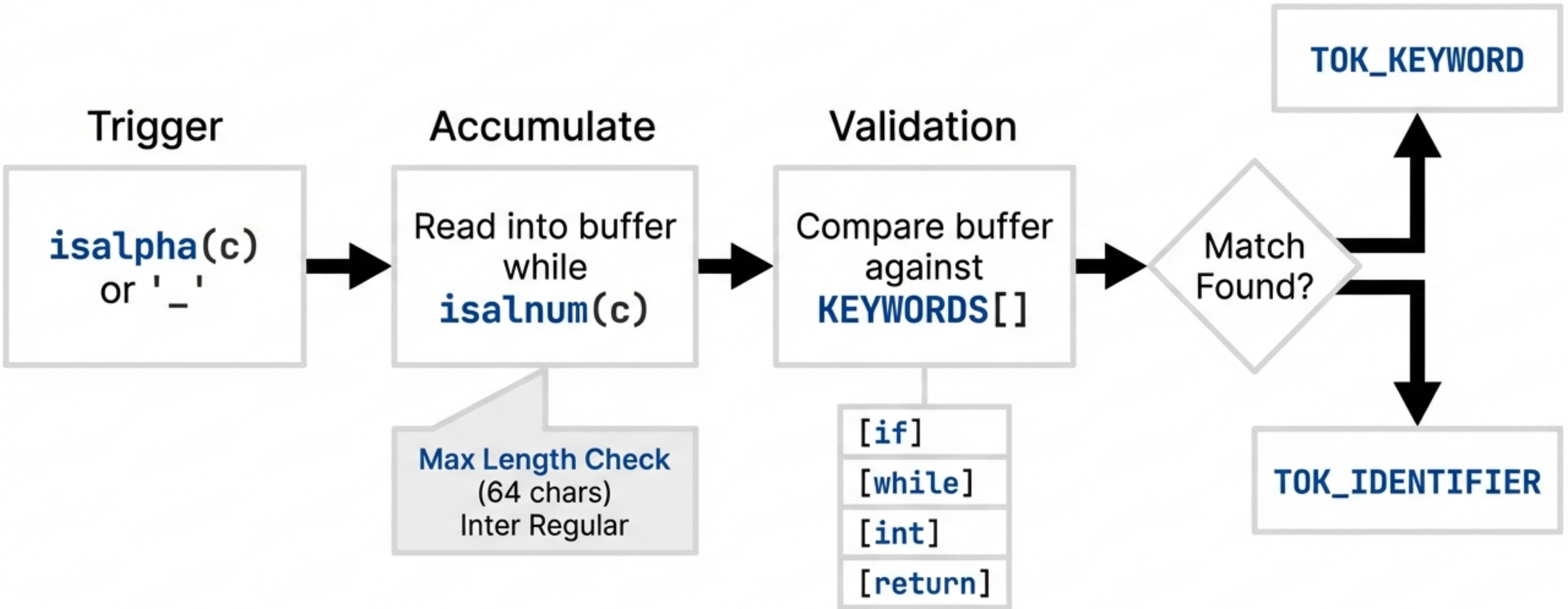
Conceptual Logic

This function runs BEFORE any token generation.

1. Read **char**.
2. If **Space/Tab/Newline** -> Loop.
3. If **'/'** -> Peek next.
4. If **'//'** -> Ignore until Newline.
5. If **'/*'** -> Ignore until **'*/'**.

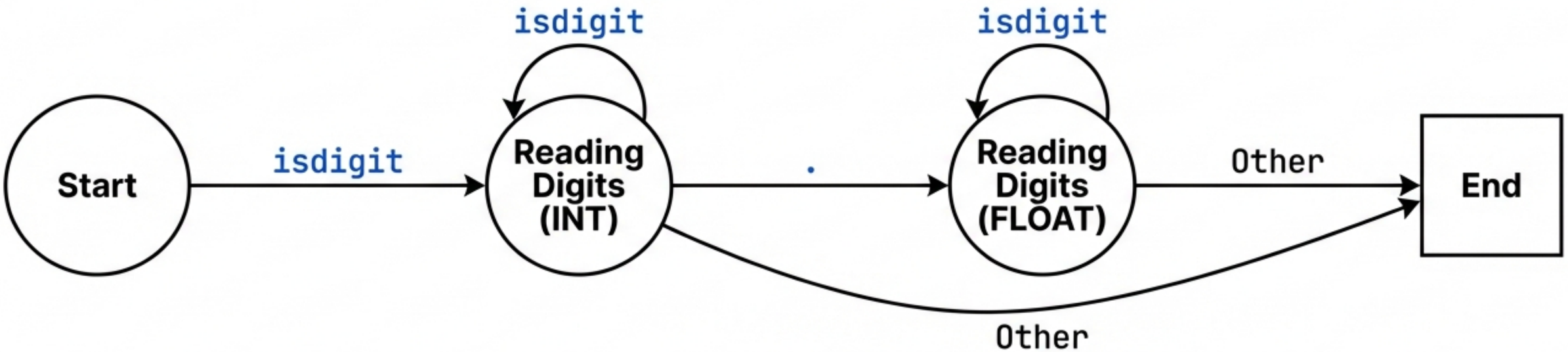
```c
static void skip_whitespace_and_comments(FILE *fp) {
    while (1) {
        int c = read_char(fp);
        if (isspace(c)) continue;
        if (c == '/') {
            // Check next char for '/' or '*'
            // If comment, consume it.
        }
        // ...
    }
}
```

Logic: Keywords vs. Identifiers



Logic: Parsing Numbers

Distinguishing Integers from Floats




```
while (isdigit(c)) { append(c); }
if (c == '.') {
    is_float = 1; append(c);
    while (isdigit(c)) { append(c); }
}
return is_float ? TOK_FLOAT : TOK_INT;
```

Logic: Strings and Escape Sequences

The Boundary

String starts at first quote and ends at the next unescaped quote.

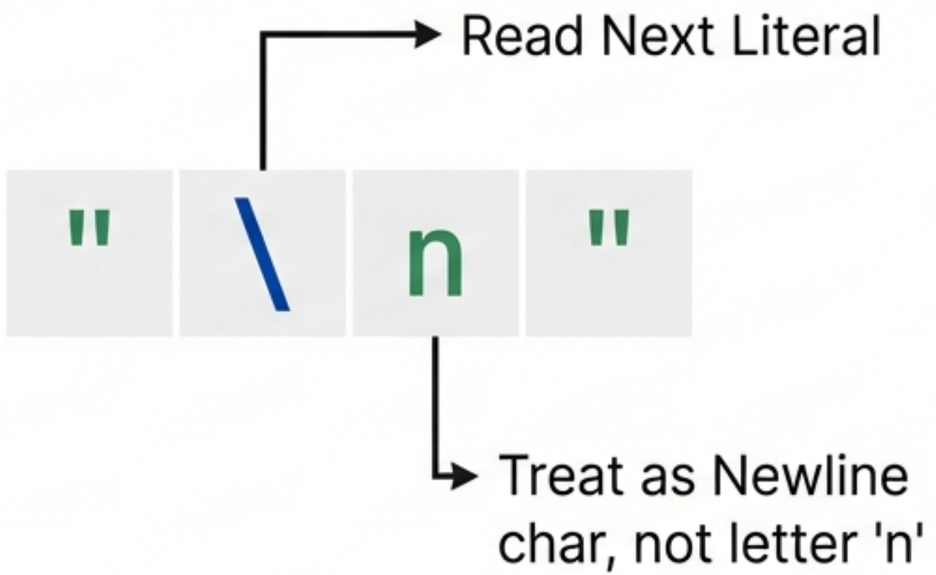
```
char str[] = "Hello World";
```

Error Case

If Newline or EOF is found before closing quote → TOK_ERROR

Escape Handling

The Backslash `\` escapes the next character.



Logic: Operators and Separators

Helvetica Now Display (Jet Black, #1A1A1A)

Separators

Simple single-character tokens.

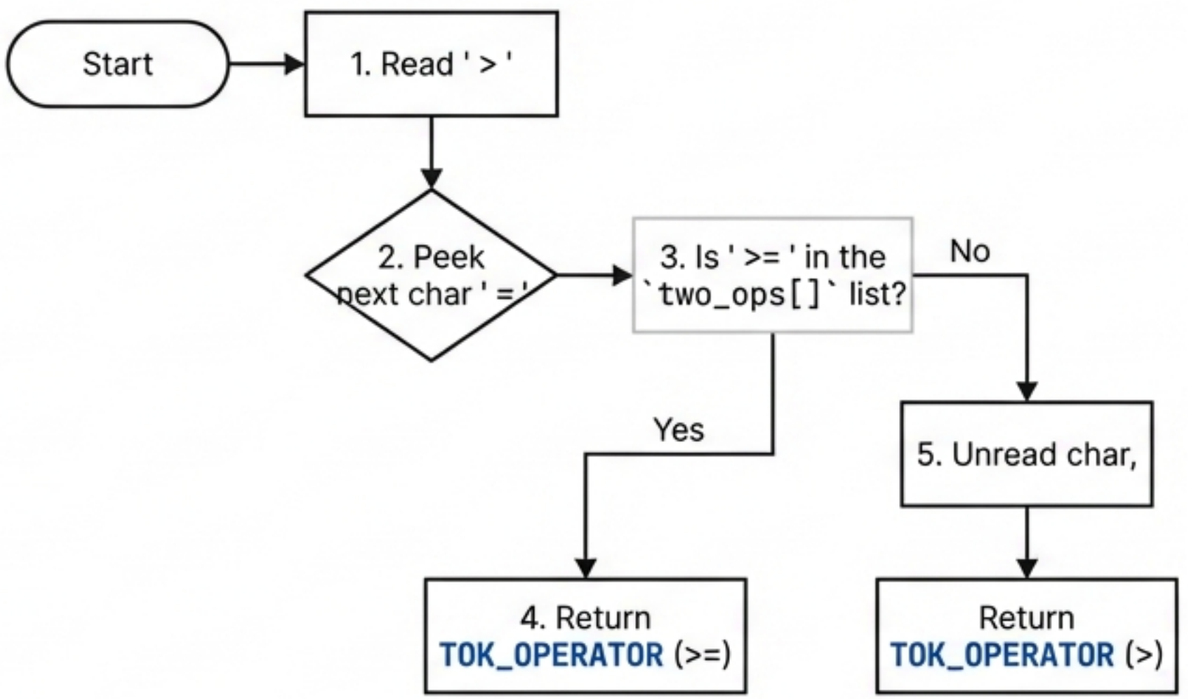
```
( ) { } ; ,
```

JetBrains Mono (Medium, #1A1A1A)

Return Type: **TOK_SEPARATOR**

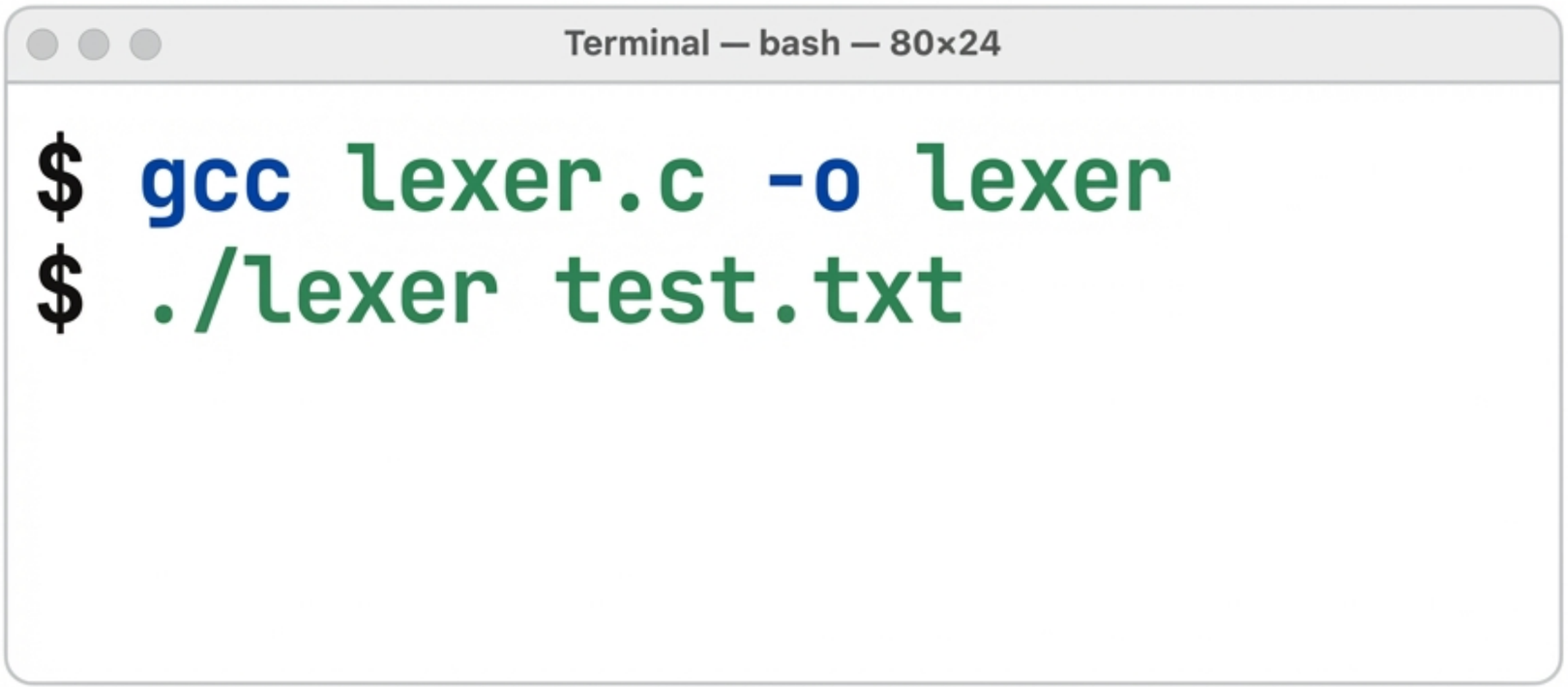
Multi-Character Operators

Greedy Matching Strategy



Build and Execution

Compiling via GCC



1. Compile the source code using standard GCC (C11).
2. Execute the binary, passing the input filename as an argument.

Verification: Input vs. Output

Input File (test.txt)

```
// Check value
if (total >= 30) {
    return 0; /* Done */
}
```

Terminal Output

```
[2:1]  KEYWORD    \"if"
[2:4]  SEPARATOR  "\("
[2:5]  IDENTIFIER \"total"
[2:11] OPERATOR   ">=""
[2:14] INT        \"30"
[2:16] SEPARATOR  "\)"
[2:18] SEPARATOR  "\{"
...
```

Comments Removed
Correct Token Types