Snippets of code? No. I don't want to focus on grammar. Focus on terminals.

Format: terminals

Keywords, ops, "$"
The Scanner

```java
for (; ; peek = next input character ) {
    if ( peek is a blank or a tab ) do nothing;
    else if ( peek is a newline ) line = line+1;
    else break;
}
```

Figure 2.29: Skipping white space

• What is the purpose of line?
• What is the purpose of peek?
Reading Ahead

• Read the next char, it is an “i”
• Could be int, if, or an identifier, so read next char, “f”
• Could be if, could still be an identifier, so read next char, “(”
• Oops, we’ve gone too far, push back “(”
Buffers

• Why is this important?

• Ways to implement:
  – Two pointers into buffer (start_char, look_ahead)
  – Push back buffer (peek)
The Lexical Analyzer

```plaintext
if ( peek holds a digit ) {
    v = 0;
    do {
        v = v * 10 + integer value of digit peek;
        peek = next input character;
    } while ( peek holds a digit );
    return token ⟨num, v⟩;
}
```

Figure 2.30: Grouping digits into integers

-23×X

What it is number
Keywords vs. Identifiers

- \( \text{count} = \text{count} + \text{increment}; \)
- \(<\text{id, "count"}>, <\text{id, "count"}>, <\text{id, "increment"}>, \) 

- How do we know count is an id vs. keyword?
- Why use a hash table?
- What is in the hash table?

1. One hash table w/ chained scopes
2. Multiple hash for scopes chained together
3. Multiple for types
How to distinguish words?

```java
if ( peek holds a letter ) {
    collect letters or digits into a buffer b;
    s = string formed from the characters in b;
    w = token returned by words.get(s);
    if ( w is not null ) return w;
    else {
        Enter the key-value pair (s, ⟨id, s⟩) into words
        return token ⟨id, s⟩;
    }
}
```

Figure 2.31: Distinguishing keywords from identifiers.
Tokens

```java
public class Token {
    public final int tag;
    public Token(int t) { tag = t; }
}
```

---

Figure 2.32: Class `Token` and subclasses `Num` and `Word`
Numbers vs. Words

1) package lexer; // File Num.java
2) public class Num extends Token {
3)     public final int value;
4)     public Num(int v) { super(Tag.NUM); value = v; }
5) }

1) package lexer; // File Word.java
2) public class Word extends Token {
3)     public final String lexeme;
4)     public Word(int t, String s) {
5)         super(t); lexeme = new String(s);
6)     }
7) }

Figure 2.33: Subclasses Num and Word of Token
Token Data Structures

struct token_t {
    int tag;
    union {
        int value;
        char *lexeme;
    } val;
};

struct token_t {
    int tag;
    union {
        int value;
        void *val;
    } val;
};