Recognition of Tokens

Lecture 3
Section 3.4

Robb T. Koether

Hampden-Sydney College

Mon, Jan 19, 2015

Outline

A Class of Tokens

- We will explore and demonstrate the concepts of a lexer by using a simple class of tokens.

$$
\begin{aligned}
\textit{digit} &\rightarrow \texttt{[0-9]} \\
\textit{digits} &\rightarrow \textit{digit}^{+} \\
\textit{number} &\rightarrow \textit{digits}\ (\texttt{.}\ \textit{digits})?\ (\text{E}\ [+-]?\ \textit{digits})? \\
\textit{letter} &\rightarrow \texttt{[A-Za-z]} \\
\textit{id} &\rightarrow \textit{letter}\ (\textit{letter} \mid \textit{digit})^{*} \\
\textit{if} &\rightarrow \texttt{if} \\
\textit{then} &\rightarrow \texttt{then} \\
\textit{else} &\rightarrow \texttt{else} \\
\textit{relop} &\rightarrow \texttt{<} \mid \texttt{>} \mid \texttt{<=} \mid \texttt{>=} \mid \texttt{=} \mid \texttt{<>}
\end{aligned}
$$

Whitespace

- In addition to recognizing tokens, the lexer must strip whitespace from the input.
- Whitespace is not a token, but it must be recognized by the lexer.

$$ws \rightarrow (\textbf{blank} \mid \textbf{tab} \mid \textbf{newline})^{+}$$

Outline

The Input Buffer

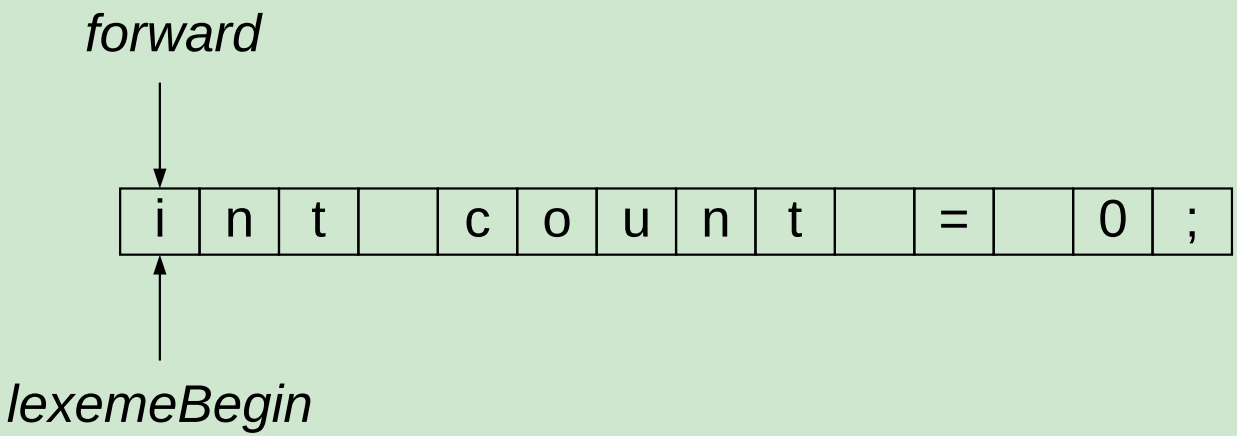
- The input to the lexer is a stream of characters.
- We may consider the characters to be residing in a buffer.
- We mark two positions in the buffer.
  - *lexemeBegin*
  - *forward*
- The pointer *lexemeBegin* holds the starting position of the current token.
- The pointer *forward* points to the current symbol.

The Input Buffer

- The lexer begins in the start state with the current symbol (pointed to by both *lexemeBegin* and *forward*).
- The process moves from state to state by following the transitions whose labels match the current symbol (*forward*).
- This continues until no further moves are possible.

The Input Buffer

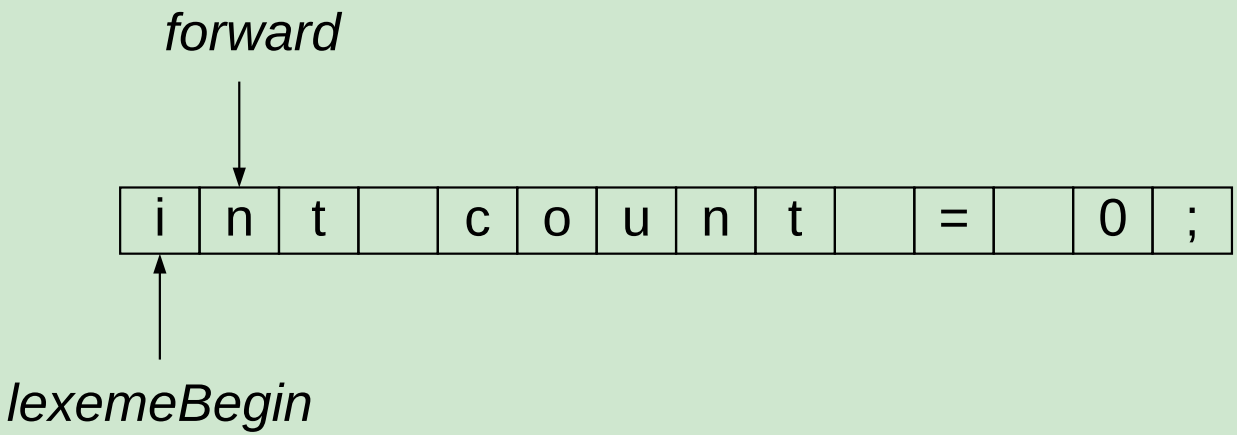
Example (Processing a Statement)



The input buffer

The Input Buffer

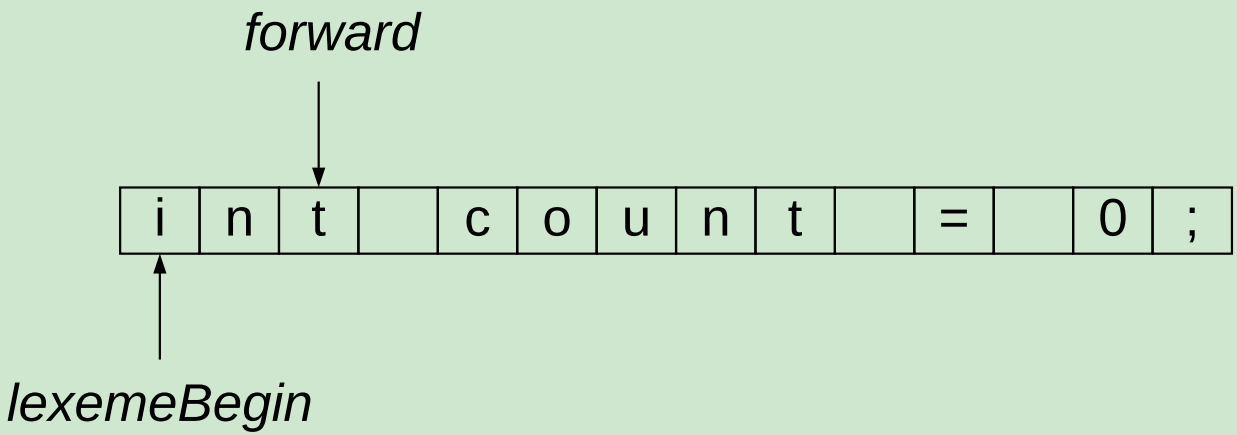
Example (Processing a Statement)



Advance one symbol

The Input Buffer

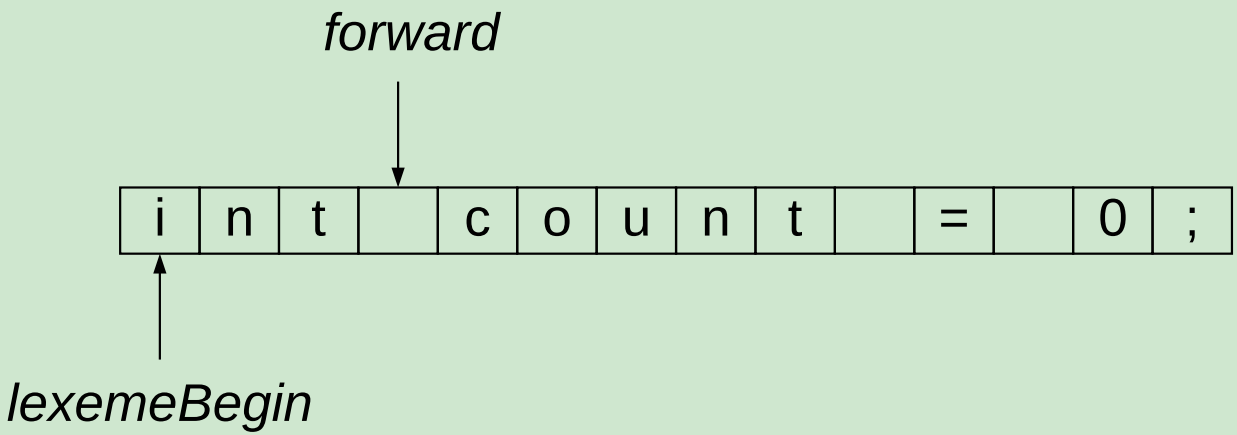
Example (Processing a Statement)



Could be an identifier; could be a keyword

The Input Buffer

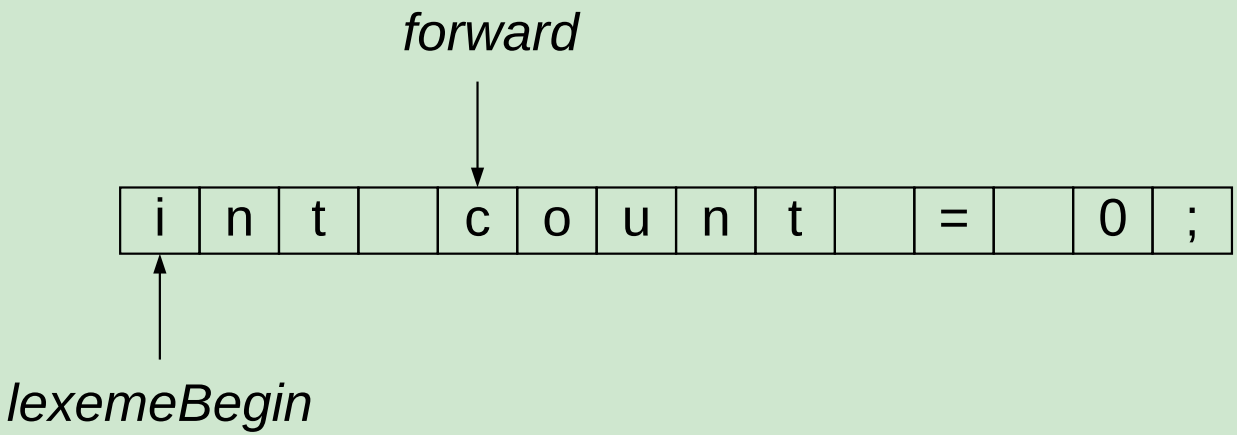
Example (Processing a Statement)



It is the keyword **int**

The Input Buffer

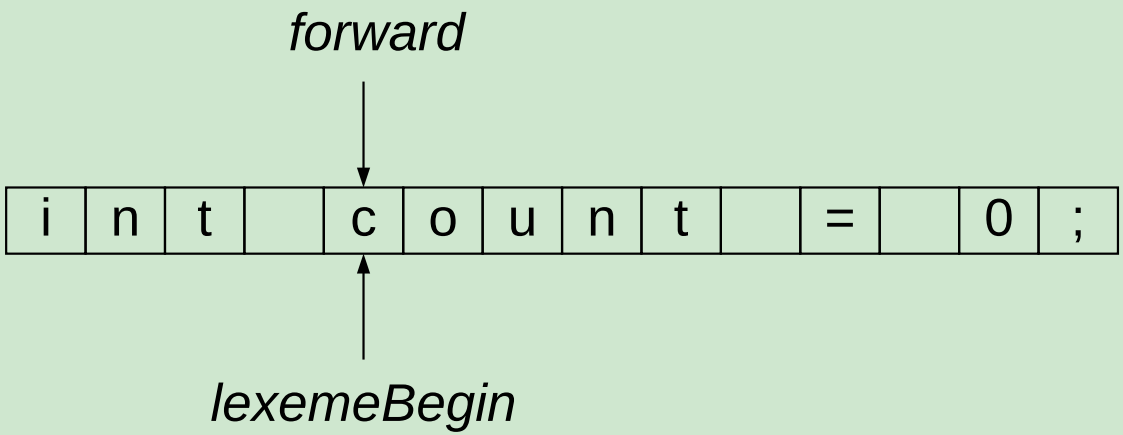
Example (Processing a Statement)



Skip whitespace

The Input Buffer

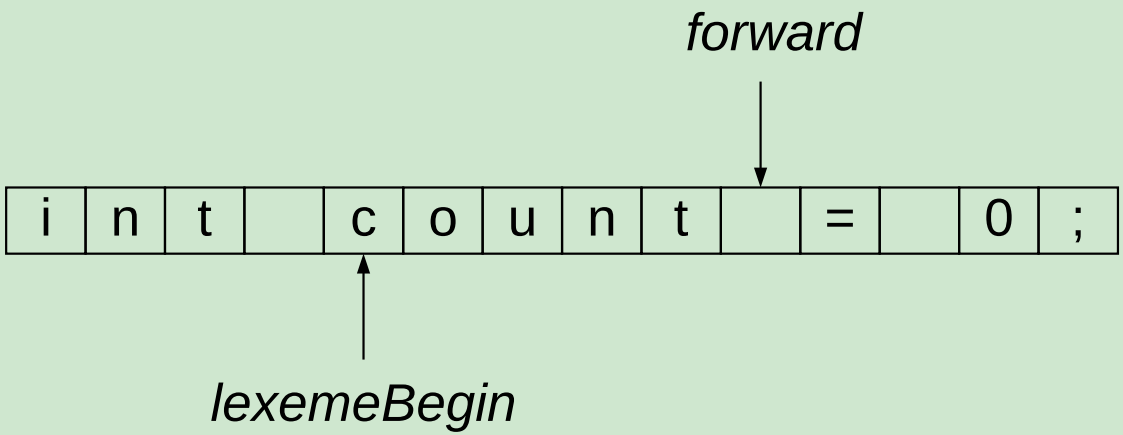
Example (Processing a Statement)



Could be an identifier; could be a keyword

The Input Buffer

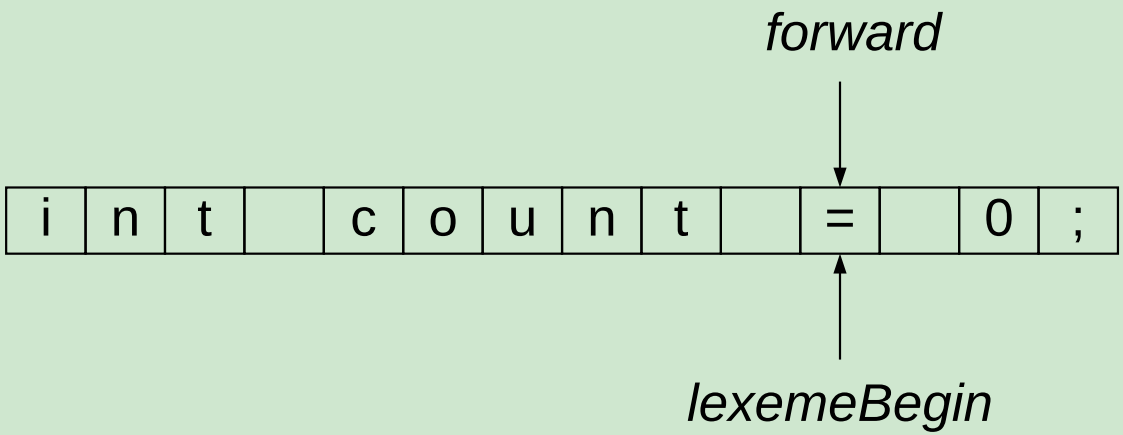
Example (Processing a Statement)



It is an identifier

The Input Buffer

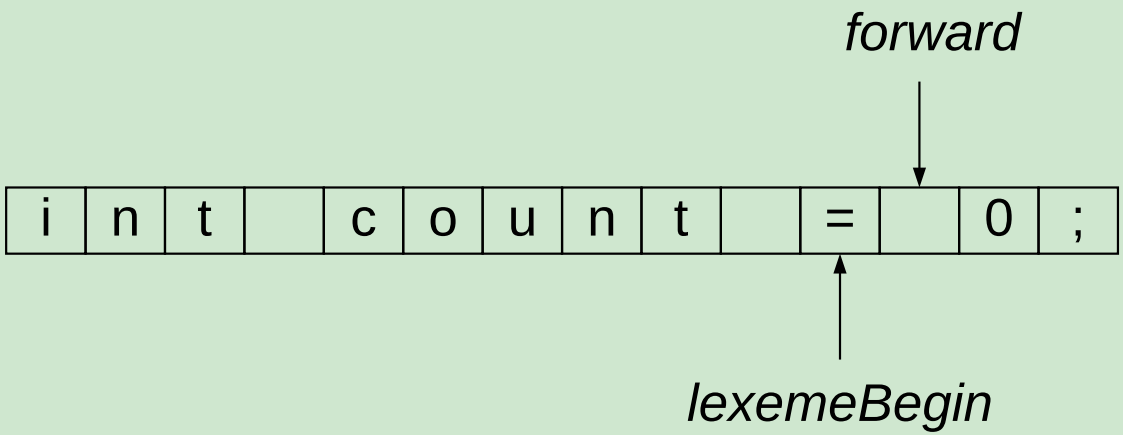
Example (Processing a Statement)



This is an operator, but which one?

The Input Buffer

Example (Processing a Statement)



It is the assignment operator

Outline

Transition Diagrams

Definition (Transition Diagram)

- A transition diagram is a directed graph.
- It consists of a finite set of nodes, called states.
- One state is designated the start state.
- The directed edges between states represent transitions.
- Each transition is labeled with a symbol (or possibly a regular expression).
- A subset of the set of states is designated the accepting states. The remaining states are rejecting states.

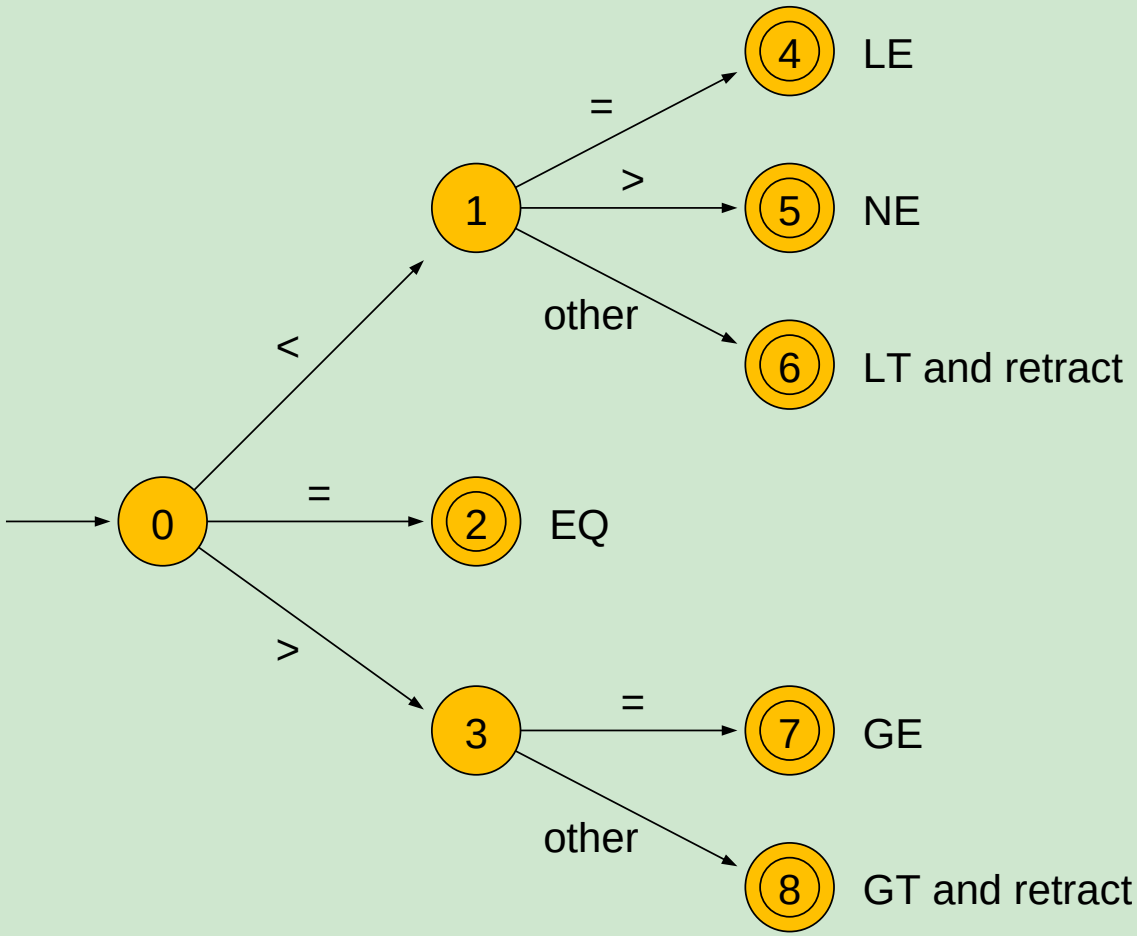
Transition Diagrams

Example (Relational Operators)

- Consider the relational operators <, >, <=, >=, =, and <>.
- The first symbol may be <, >, =, or something else.
- If the first symbol is <, then the next symbol may be =, >, or something else.
- If the first symbol is >, then the next symbol may be = or something else.
- If the first symbol is =, then the next symbol is something else.

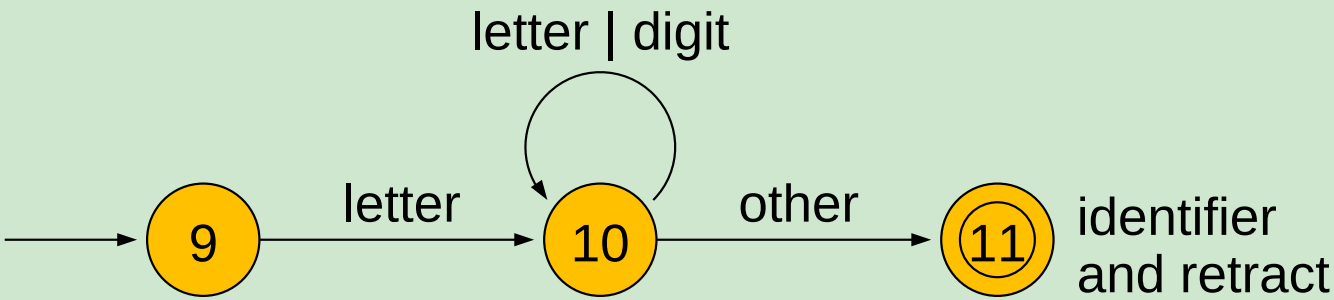
Transition Diagrams

Example (Relational Operators)



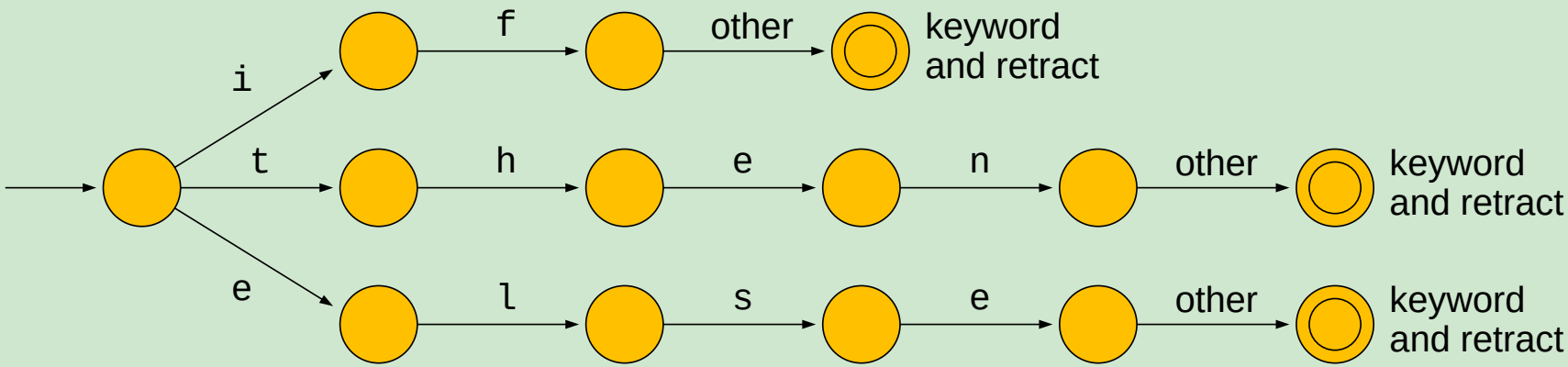
Transition Diagrams

Example (Identifiers)



Transition Diagrams

Example (Keywords)



Transition Diagrams

- Draw the transition diagram for numbers

$$
\begin{aligned}
\textit{digit} &\longrightarrow [0-9] \\
\textit{digits} &\longrightarrow \textit{digit}^{+} \\
\textit{number} &\longrightarrow \textit{digits}\ (.\ \textit{digits})?\ (\mathtt{E}\ [+-]?\ \textit{digits})?
\end{aligned}
$$

Outline

Writing the Lexer

- The lexer is the program that implements the transition diagram.
- We could use
  - A **`switch`** statement, and/or
  - An **`if-else`** structure.

Writing the Lexer

The Lexer for Relational Operators

```
Token getRelop()
{
    int state = 0;
    char c = get_next_symbol();
    while (c == '<' || c == '=' || c == '>')
    {
        switch (state)
        {
            case 0:
                if (c == '<') state = 1;
                else if (c == '=') state = 2;
                else if (c == '>') state = 3;
                else fail();
                break;
            ...
            case 8:
                retract();
                return Token(GT);
        }
    }
}
```

Outline

Assignment

Assignment

- Read Section 3.4.
- Exercises 1, 2(c)(i).