

# Recognition of Tokens

Lecture 2  
Section 3.3

ROKAN UDDIN FARUQUI

Associate Professor  
Dept of Computer Science and Engineering  
University of Chittagong, Bangladesh  
Email: *rokan@cu.ac.bd*

- ① A Class of Tokens
- ② The Input Buffer
- ③ Transition Diagrams
- ④ Writing the Lexer
- ⑤ Assignment



# Outline

- ① A Class of Tokens
- ② The Input Buffer
- ③ Transition Diagrams
- ④ Writing the Lexer
- ⑤ Assignment



# A Class of Tokens

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

*digit* → [0-9]

*digits* → *digit*<sup>+</sup>

*number* → *digits* ( . *digits*)? (E [+-]? *digits*)?

*letter* → [A-Za-z]

*id* → *letter* (*letter* | *digit*)\*

*if* → if

*then* → then

*else* → else

*relop* → < | > | <= | >= | = | <>



# 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.

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


# 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.

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


# Outline

- 1 A Class of Tokens
- 2 The Input Buffer
- 3 Transition Diagrams
- 4 Writing the Lexer
- 5 Assignment



# The Input Buffer

**Buffer Pairs** – a two-buffer scheme that handles large lookaheads safely

**Sentinels** – an improvement involving sentinels that saves time checking for the ends of buffers



# The Input Buffer: Buffer Pairs

- Input to the lexer — a stream of characters.
- Characters reside in a buffer.
- Mark two positions in the buffer.
  - *lexemeBegin*
  - *forward*
- *lexemeBegin* pointer – holds the starting position of the current token.
- *forward* pointer – points to the current symbol.



# The Input Buffer: Buffer Pairs

- Input to the lexer — a stream of characters.
- Characters reside in a buffer.
- Mark two positions in the buffer.
  - *lexemeBegin*
  - *forward*
- *lexemeBegin* pointer – holds the starting position of the current token.
- *forward* pointer – points to the current symbol.



# The Input Buffer: Buffer Pairs

- Input to the lexer — a stream of characters.
- Characters reside in a buffer.
- Mark two positions in the buffer.
  - *lexemeBegin*
  - *forward*
- *lexemeBegin* pointer – holds the starting position of the current token.
- *forward* pointer – points to the current symbol.



# The Input Buffer: Buffer Pairs

- Input to the lexer — a stream of characters.
- Characters reside in a buffer.
- Mark two positions in the buffer.
  - *lexemeBegin*
  - *forward*
- *lexemeBegin* pointer – holds the starting position of the current token.
- *forward* pointer – points to the current symbol.



# The Input Buffer: Buffer Pairs

- Input to the lexer — a stream of characters.
- Characters reside in a buffer.
- Mark two positions in the buffer.
  - *lexemeBegin*
  - *forward*
- *lexemeBegin* pointer – holds the starting position of the current token.
- *forward* pointer – points to the current symbol.



# The Input Buffer: Buffer Pairs

- Input to the lexer — a stream of characters.
- Characters reside in a buffer.
- Mark two positions in the buffer.
  - *lexemeBegin*
  - *forward*
- *lexemeBegin* pointer – holds the starting position of the current token.
- *forward* pointer – 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

- 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.



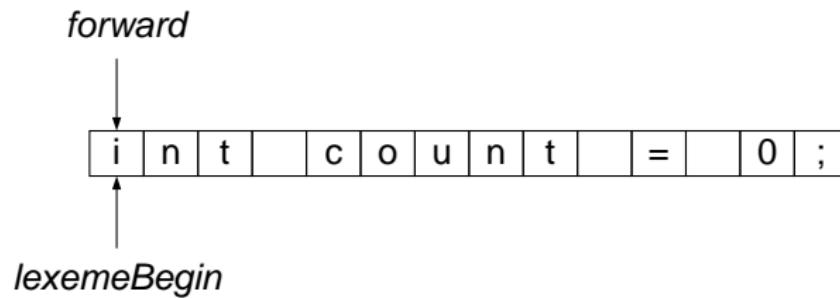
# The Input Buffer

- 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

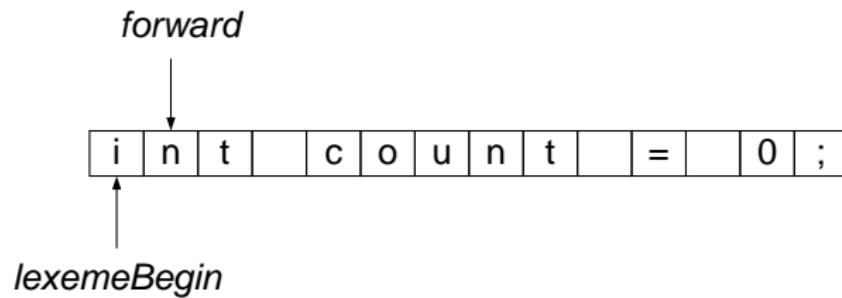
Example (Processing a Statement)



The input buffer

# The Input Buffer

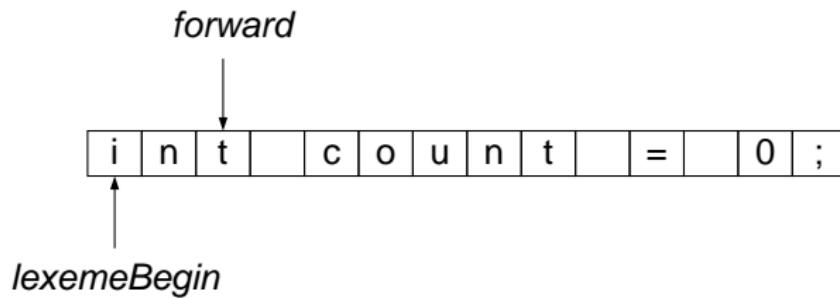
Example (Processing a Statement)



Advance one symbol

# The Input Buffer

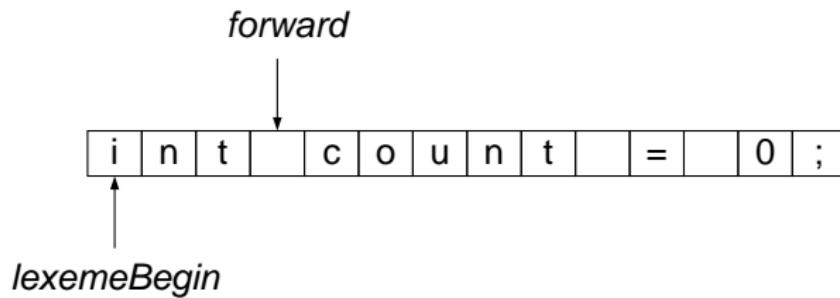
Example (Processing a Statement)



Could be an identifier; could be a keyword

# The Input Buffer

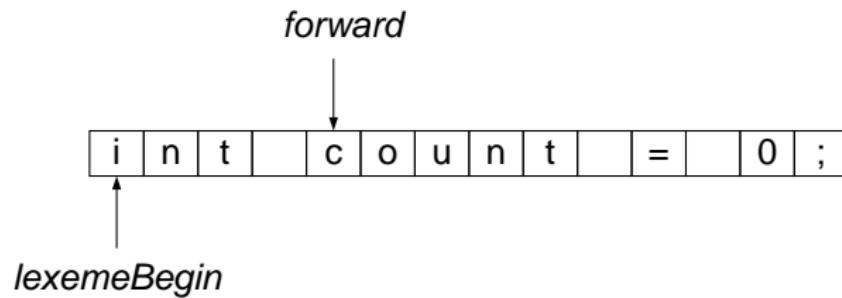
Example (Processing a Statement)



It is the keyword **int**

# The Input Buffer

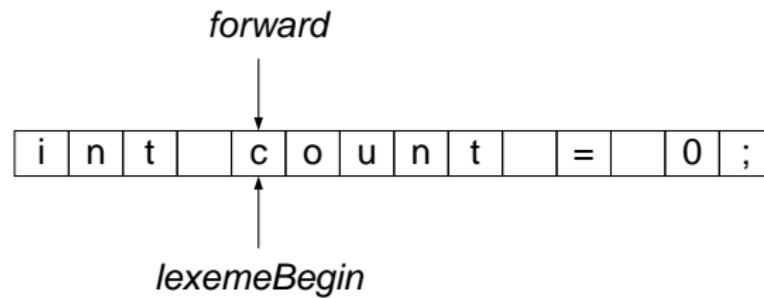
Example (Processing a Statement)



Skip whitespace

# The Input Buffer

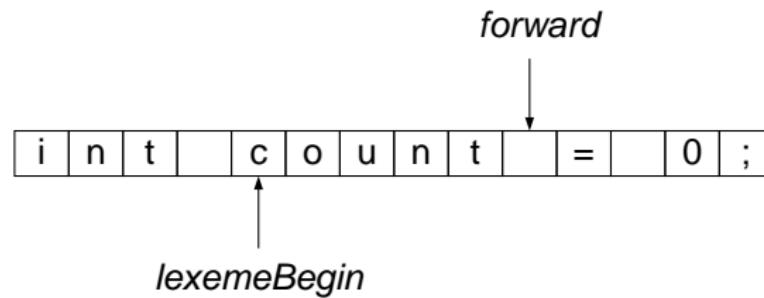
Example (Processing a Statement)



Could be an identifier; could be a keyword

# The Input Buffer

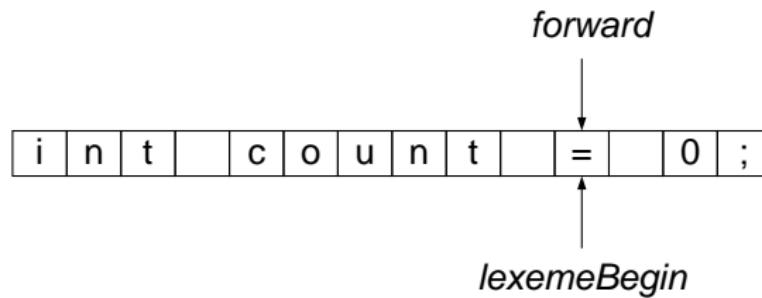
Example (Processing a Statement)



It is an identifier

# The Input Buffer

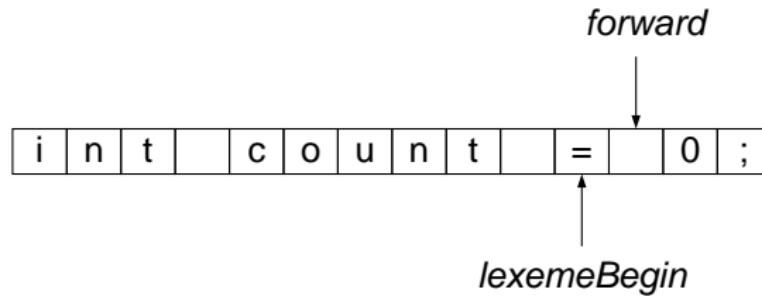
Example (Processing a Statement)



This is an operator, but which one?

# The Input Buffer

Example (Processing a Statement)



It is the assignment operator

# Outline

- 1 A Class of Tokens
- 2 The Input Buffer
- 3 Transition Diagrams
- 4 Writing the Lexer
- 5 Assignment



# 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

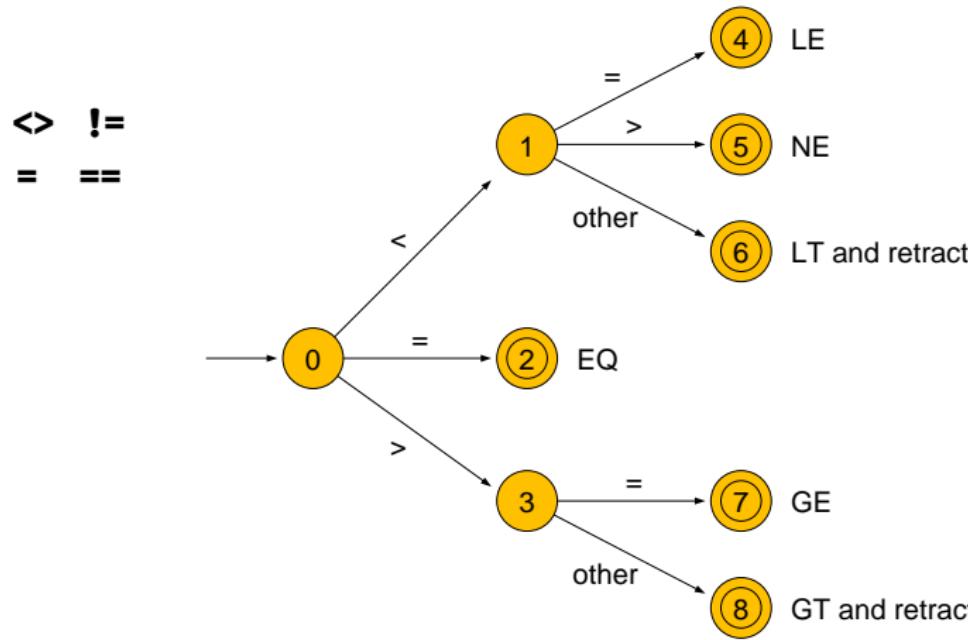
Example (Relational Operators)

- Consider the relational operators  $<$ ,  $>$ ,  $\leq$ ,  $\geq$ ,  $=$ , and  $\neq$ .
- 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.



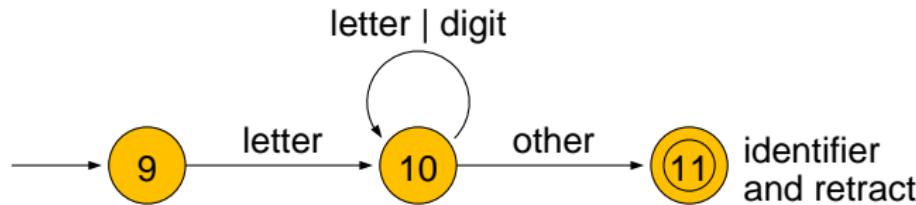
# Transition Diagrams

Example (Relational Operators)



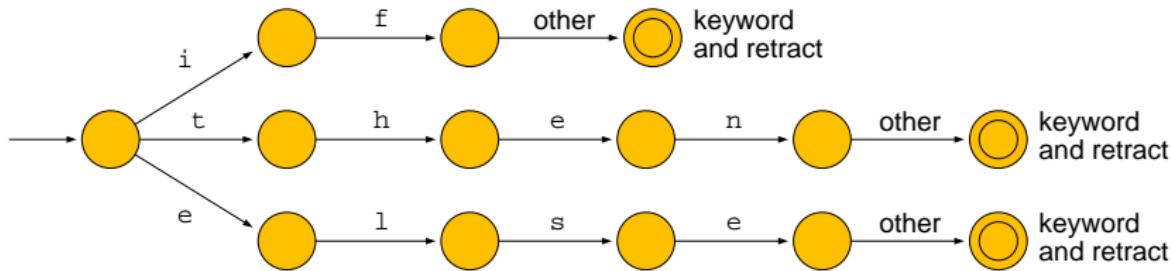
# Transition Diagrams

Example (Identifiers)



# Transition Diagrams

Example (Keywords)



# Transition Diagrams

- Draw the transition diagram for numbers

*digit* → [0-9]

*digits* → *digit*<sup>+</sup>

*number* → *digits* (. *digits*)? (E [+−]?) *digits*)?



# Outline

- 1 A Class of Tokens
- 2 The Input Buffer
- 3 Transition Diagrams
- 4 Writing the Lexer
- 5 Assignment



# 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 == '>')      flex/lex – regular
    {
        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

- 1 A Class of Tokens
- 2 The Input Buffer
- 3 Transition Diagrams
- 4 Writing the Lexer
- 5 Assignment



# Assignment

## Assignment

- Read Section 3.4.
- Exercises 3.4.1 - 3.4.2

