# Compilers

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

# Today's Lecture

- Scanner

- What are the parts of speech of the words in the following sentence?

The woman opened the door.

# Parts of Speech in an English Sentence

- What are the parts of speech of the words in the following sentence?

The woman opened the door.


- The – Specific instance of an article.
- Woman – Specific instance of a noun.
- Opened – Specific instance of a verb.
- Door – Specific instance of a noun.

# Parts of Speech in an English Sentence

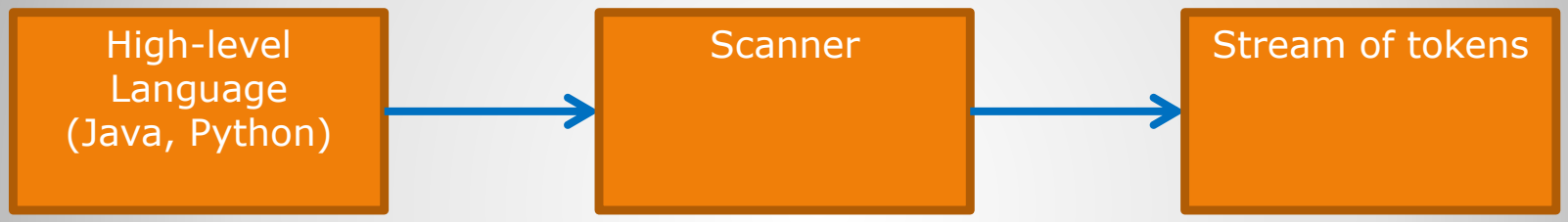
# English Language Scanner

- An English language scanner would be given an English language sentence and generate the parts of speech of each word.
- The output is a stream containing the parts of speech of each word.


- The scanner reads each word of the sentence and determines its part of speech.

# Program Scanner

- A program is given to the scanner as input.
- The scanner generates a stream of tokens as output
- Tokens are like the parts of speech in an English sentence.



# Scanner

- Within a compiler, the Scanner is responsible for lexical analysis.
- The scanner reads from an input stream and returns tokens.
- The scanner will have a method (say scan()) that reads from the input file and returns one token each time it is called.
- The scanner will keep track of where it is in the input stream.
- Subsequent calls to scan() will keep it moving through the input stream.

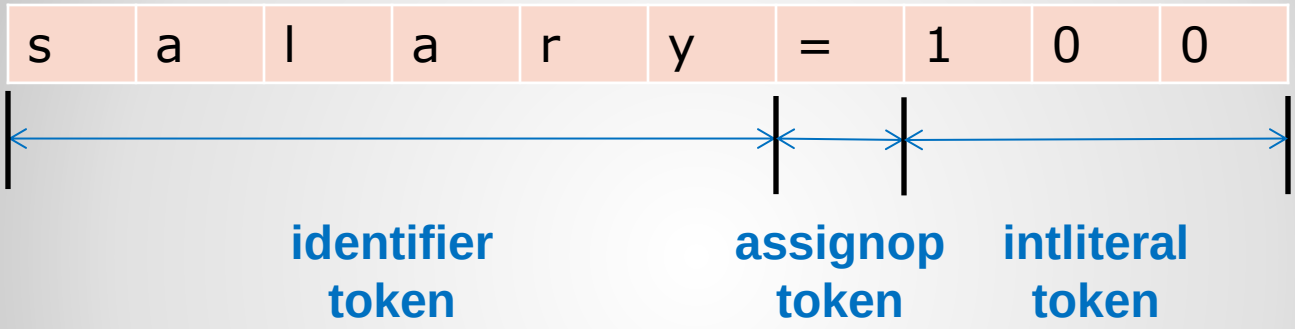
# Scanner Terminology

- **Token** – Represents a lexical unit. For example: id (an identifier or variable), if, intliteral, relationaloperator, assignop.
- **Lexeme** – A sequence of characters that match a token. For example: x (a variable or id), if (same as the token in this case), 77 (an int literal), < (a relational operator), = (assignop).
- A lexeme is like a specific instance of a token.
- For example:

| Token | Lexeme |
|---|---|
| id | x |
| if | if |
| intliteral | 77 |
| relationaloperator | < |
| assignop | = |
| identifier | salary |

# Scanner Processing

- The scanner takes a stream of characters as input and produces tokens.




- The lexeme for the identifier is "salary".
- The lexeme for the assignop is "=".
- The lexeme for the intliteral is "100".

- Regular expressions will be used to define the valid patterns of tokens.
- Valid ids (variables) will have a regular expression that defines their pattern.
- Integer literals will have a regular expression.
- Floating point literals will have a regular expression.

# Regular Expressions and Scanner

# Scan Method

- Here is pseudocode for basic structure of a scanner's Scan() method:

**Scan() return TOKEN**

```
Read c

While (c != EOF)
    Process c (will return tokens as appropriate)
    Read c
```

- Keep reading and processing one character at a time.
- The Process c statement will cause it to break out of the loop when it gets to the end of a token.
- One call to Scan should consume one token in the input stream.

# Checking for a Whitespace Character

- Here a method to check if a character is whitespace.
- It will check the ASCII code value to see if it corresponds to whitespace.

**IsWhiteSpace(int c) return boolean**

```
If (c == 32) or (c == 9) or (c == 10) or (c == 13)
   return true
Else
   return false
```

- 32 is a blank space
- 9 is a tab
- 10 is a linefeed (this means go to the next line)
- 13 is a carriage return (go to the start of the line)

# Ignoring Whitespace When Scanning

- Here is the main loop for a scanner that ignores whitespace.

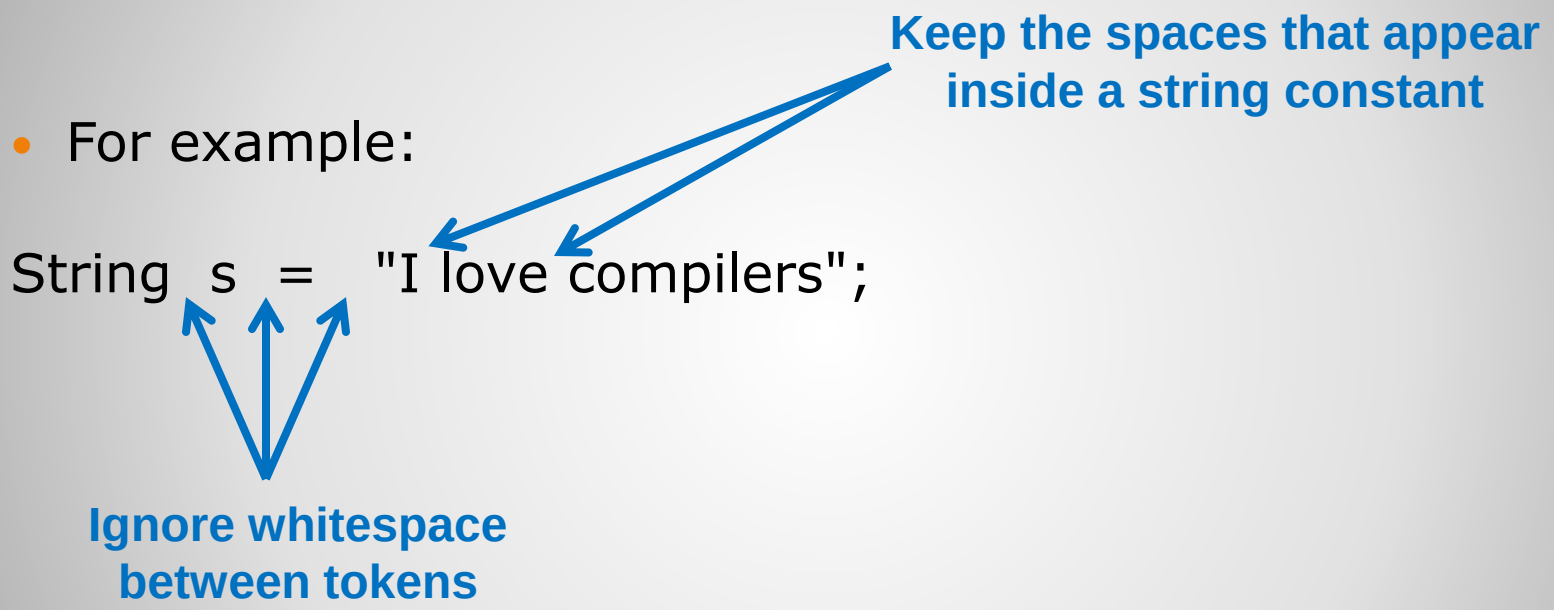
```
Read c
While (not end of file)
   if (isWhiteSpace(c))
      Read c
      continue

   // Other scanner code goes here…
EndWhile
```

**Check to see if the current character is whitespace at the start of the loop**

**If the character is whitespace, then read another character and go back to the start of the loop**

# When to Keep Spaces

- A scanner will generally keep spaces that appear inside of a string constant.
- Whitespace between tokens is generally removed.




- For example:

# Matching an Integer Constant (no buffering)

- Check for an integer constant (an integer literal).
- Once a digit is found it must keep reading characters until it reaches something that is not a digit (that is the end of the integer constant).
- The character that causes it to break out of the loop (the nondigit) must be put back into the input stream.

```
Read c
If (c is a digit)
    Read c
    While (c is a digit)
        Read c

    Unread c
    return TOKEN.INTLITERAL;
```

**Keep reading characters while they are digits**

**The last character that was read is not a digit so it must be put back into the input stream (that character might be part of the next token so should not be consumed)**

# Matching an Integer Constant (with character buffering)

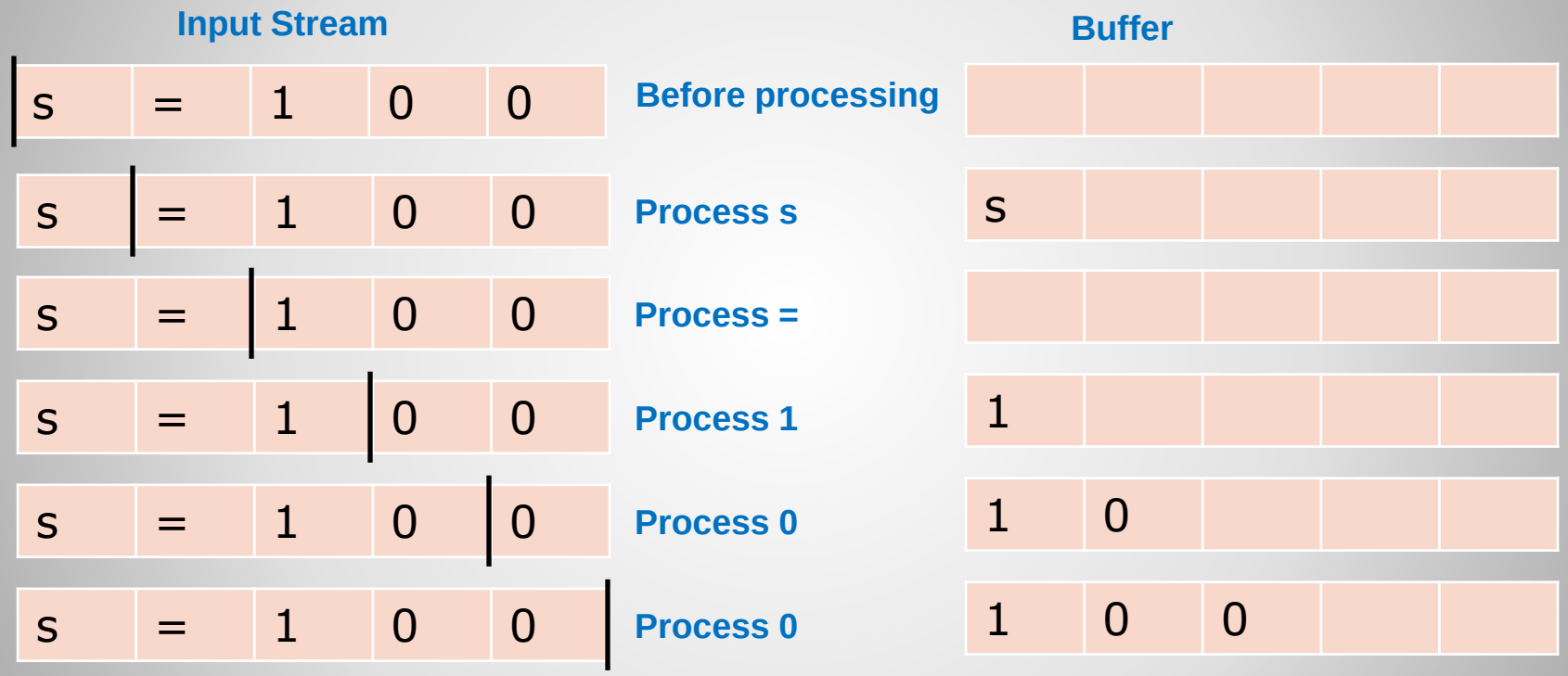
- The actual string should also be saved when reading an integer.
- Just add the character to a string buffer before reading the next one.
- The buffer should be cleared when starting a new token.
- For example:

```
Read c
If (c is a digit)
    Clear buffer
    Add c to character buffer
    Read c
    While (c is a digit)
        Add c to character buffer
        Read c

    Unread c
    return TOKEN.INTLITERAL;
```

Clear the buffer for the new token

Add each new character to the buffer. In Java, you can use a StringBuilder for the buffer.

- The identifier and intliteral characters should be buffered.
- When processing a new token clear the buffer.
- There is no need to buffer the assignop.

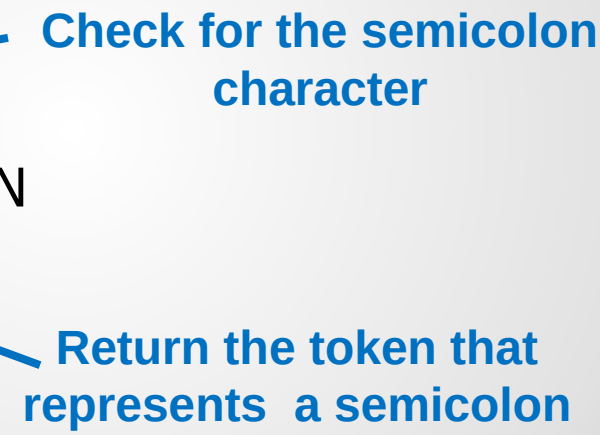



# Processing Characters and Buffering

# Matching a Specific Character

- To match a single character, you should use an if statement that tests for the character.
- The code below matches a semicolon.
- There should be a token value that represents the character.

```
Read c
If (c == ';')
   return TOKEN.SEMICOLON
```



- Java PushbackReader.
- Java input stream class that allows putting data back into the input stream after it is read.
- This can be used for the Unread pseudocode command.

```
PushbackReader pbr;
pbr = new PushbackReader(new FileReader(new File("in.txt")));

int inChar;

inChar = pbr.read();
System.out.println(inChar);

pbr.unread(inChar);

inChar = pbr.read();
System.out.println(inChar);
```

**Reads the next character from the input stream as an int (the ASCII code of the character)**

**The unread method will put the character back into the input stream**

**This read will get back whatever character the unread method put back on the input stream**

# Java PushbackReader

# Hands-on Exercise

- Write a program that scans unsigned integers and the + operator. Here is some sample data (put in a file):

1
+
2
+
3

1. Create a console application project.
2. Open the input file and connect it to a PushbackReader.
3. Scan all data until the end of file is reached. When it recognizes a token, it should print out the token name (integer or plus). It should ignore all whitespace.

EXTRA CHALLENGE – Use multiple digits in a number (111, 222, etc…) and save them in a character buffer. Print the contents of the buffer when a token is recognized.

- **End of Slides**

# End of Slides