CMPE 152: Compiler Design

September 12 Lab

Department of Computer Engineering
San Jose State University

Fall 2017
Instructor: Ron Mak
www.cs.sjsu.edu/~mak

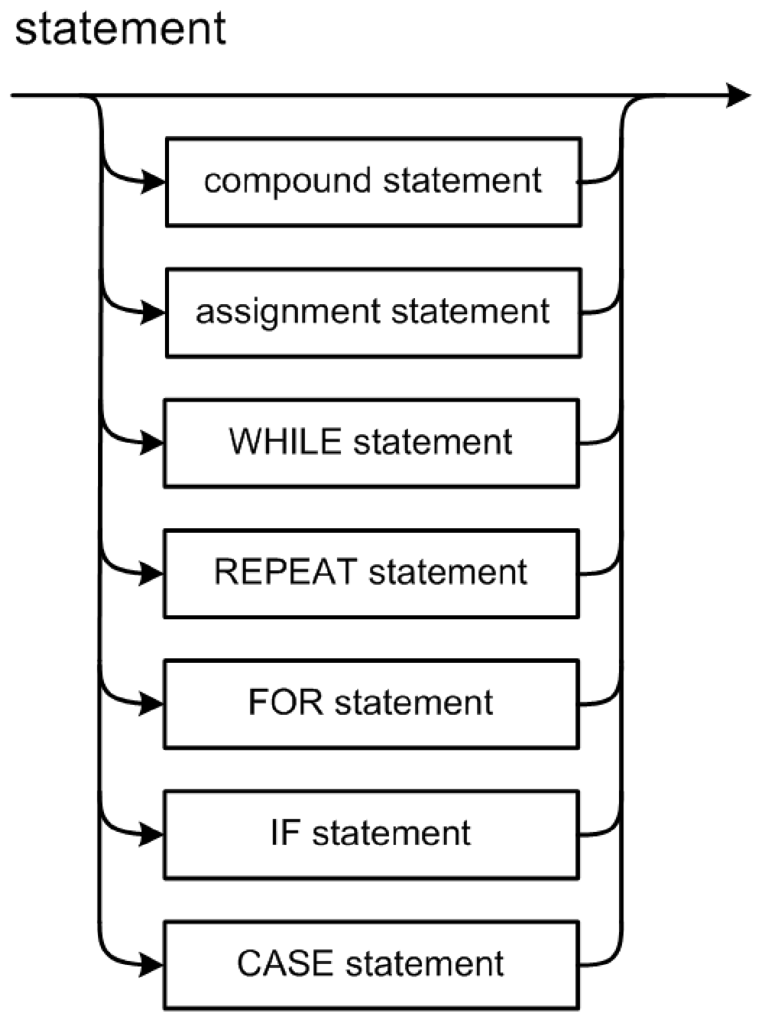
SILICON VALLEY'S
FIRST CHOICE
FOR NEW
ENGINEERING HIRES
— Silicon Valley Business Journal, 2013

San José State
UNIVERSITY

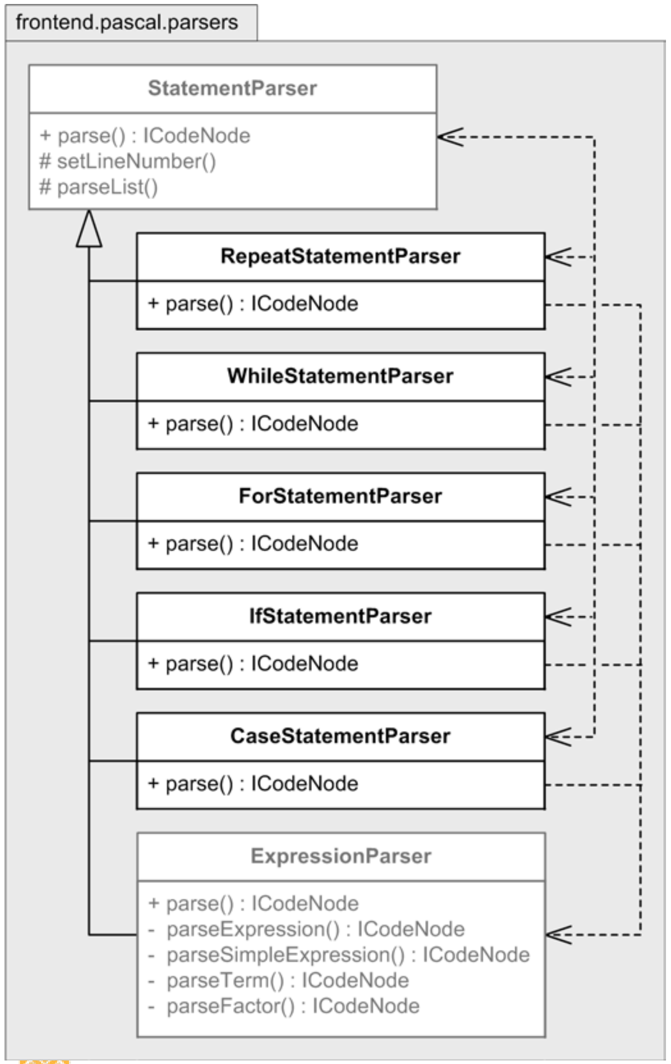
Pascal Control Statements

- Looping statements
  - `REPEAT UNTIL`
  - `WHILE DO`
  - `FOR TO`
  - `FOR DOWNTO`
- Conditional statements
  - `IF THEN`
  - `IF THEN ELSE`
  - `CASE`

Statement Syntax Diagram

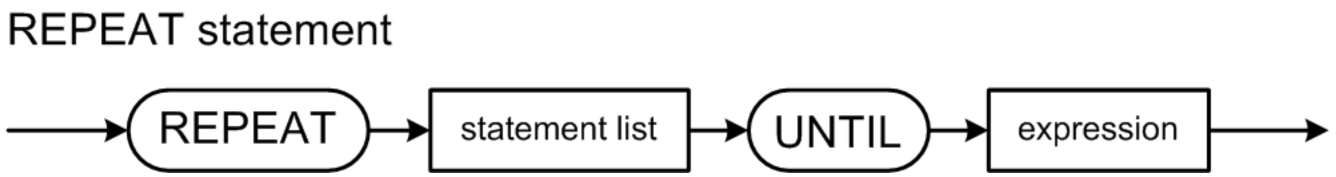


Pascal Statement Parsers



- New statement parser subclasses.
  - **`RepeatStatementParser`**
  - **`WhileStatementParser`**
  - **`ForStatementParser`**
  - **`IfStatementParser`**
  - **`CaseStatementParser`**
- Each **`parse()`** method builds a parse subtree and returns the root node.

REPEAT Statement

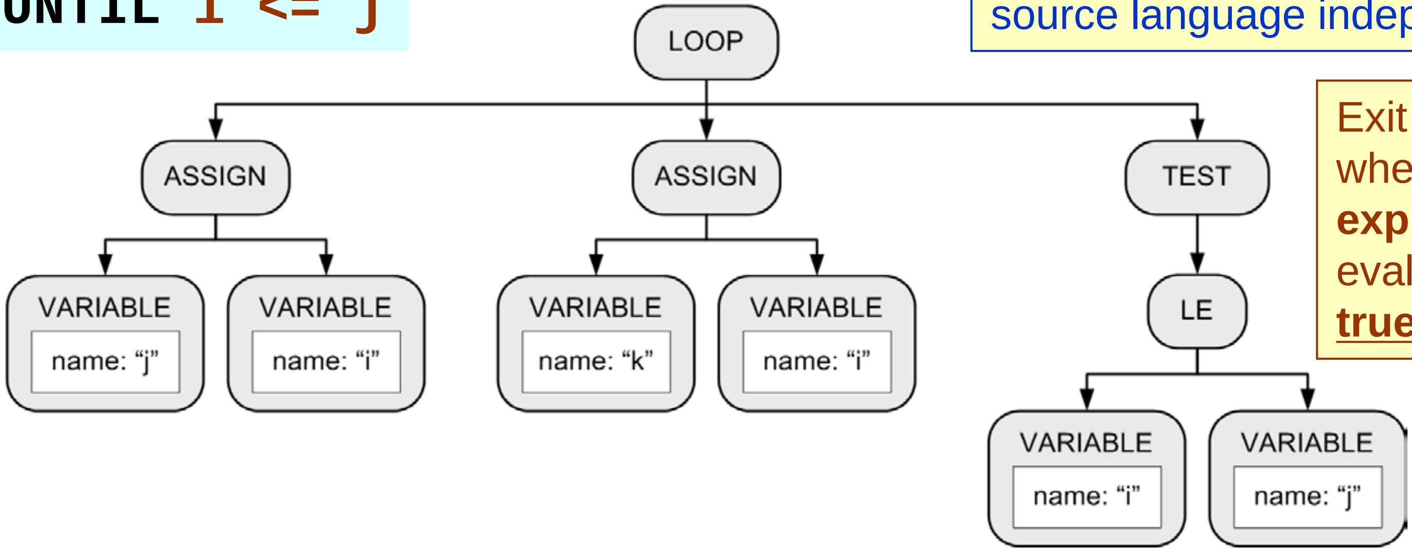


- Example:

```
REPEAT
      j := i;
      k := i
UNTIL i <= j
```

- Keep looping until the boolean expression becomes true.
  - Execute the loop at least once.

Use **LOOP** and **TEST** nodes for source language independence.



Exit the loop when the **test expression** evaluates to **true**.

Syntax Error Handling

- Recall that syntax error handling in the front end is a three-step process.

  1. Detect the error.
  2. Flag the error.
  3. Recover from the error.

- Good syntax error handling is important!

Options for Error Recovery

- Stop after the first error.
  - No error recovery at all.
  - Easiest for the compiler writer, annoying for the programmer.
  - Worse case: The compiler crashes or hangs.
- Become hopelessly lost.
  - Attempt to continue parsing the rest of the source program.
  - Spew out lots of irrelevant and meaningless error messages.
  - No error recovery here, either …
    - … but the compiler writer doesn’t admit it!

Options for Error Recovery, *cont'd*

- Skip tokens after the erroneous token until …
  - The parser finds a token it recognizes, and
  - It can safely resume syntax checking the rest of the source program.

Parser Synchronization

- Skipping tokens to reach a safe, recognizable place to resume parsing is known as synchronizing.
  - "Resynchronize the parser" after an error.

- Good error recovery with top-down parsers is more art than science.
  - How many tokens should the parser skip?
    - Skipping too many (the rest of the program?) can be considered "panic mode" recovery.
  - For this class, we'll take a rather simplistic approach to synchronization.

Function synchronize()

```
PascalToken *PascalParserTD::synchronize(
                              const set<PascalTokenType>& sync_set)
    throw (string)
{
    Token *token = current_token();

    if (sync_set.find((PascalTokenType) token->get_type())
            == sync_set.end())
    {
        error_handler.flag(token, UNEXPECTED_TOKEN, this);

        do
        {
            token = next_token(token);
        } while ((token != nullptr) &&
                 (sync_set.find((PascalTokenType) token->get_type())
                      == sync_set.end()));
    }

    return (PascalToken *) token;
}
```

Flag the **first** bad token.

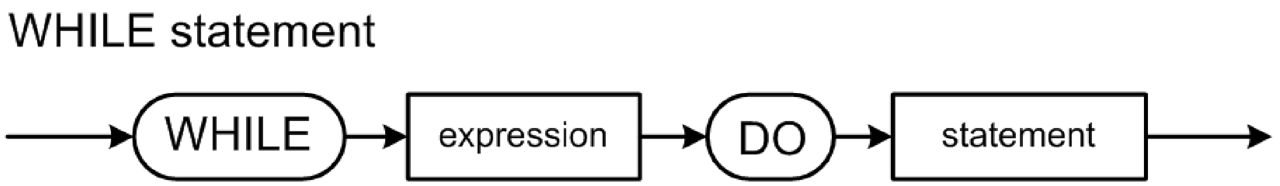
Recover by skipping tokens **not in** the synchronization set.

Resume parsing at this token! (It's the first token after the error that **is in** the synchronization set.

Pascal Syntax Checker II: **REPEAT**

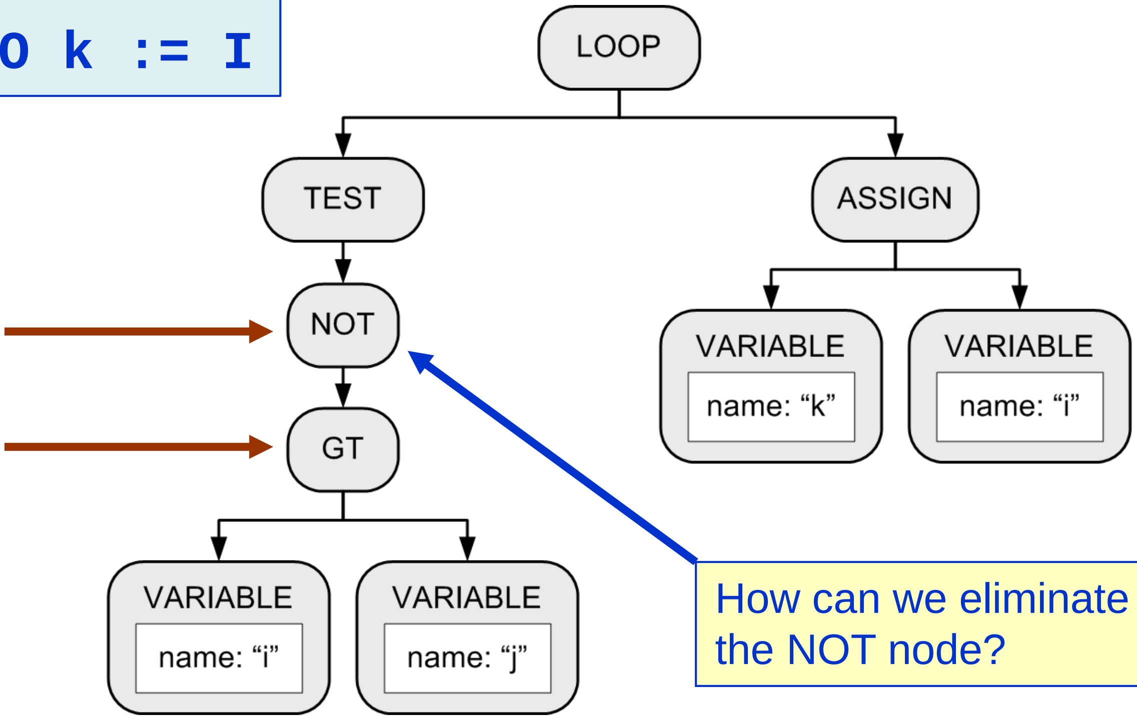
- Demo (Chapter 7)
  - `./Chapter7cpp compile -i repeat.txt`
  - `./Chapter7cpp compile –i repeaterrors.txt`

WHILE Statement



- Example

```
WHILE i > j DO k := I
```



Exit the loop when the **test expression** evaluates to **false**.

How can we eliminate the NOT node?

Class WhileStatementParser

- From parent class **StatementParser**:

```
set<PascalTokenType> StatementParser::STMT_START_SET =
{
    PT_BEGIN, PT_CASE, PT_FOR, PT_IF, PT_REPEAT, PT_WHILE,
    PT_IDENTIFIER, PT_SEMICOLON,
};

set<PascalTokenType> StatementParser::STMT_FOLLOW_SET =
{
    PT_SEMICOLON, PT_END, PT_ELSE, PT_UNTIL, PT_DOT,
};
```

Class **WhileStatementParser**, *cont'd*

- In class **WhileStatementParser**:

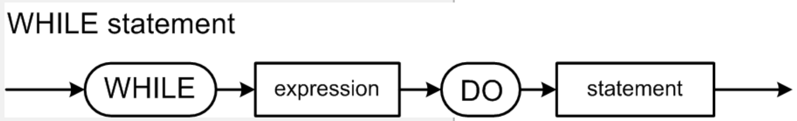
```
DO_SET = StatementParser::STMT_START_SET;
DO_SET.insert(PascalTokenType::DO);

set<PascalTokenType>::iterator it;
for (it  = StatementParser::STMT_FOLLOW_SET.begin();
     it != StatementParser::STMT_FOLLOW_SET.end();
     it++)
{
    DO_SET.insert(*it);
}
```

- **DO_SET** contains all the tokens that can start a statement or follow a statement, plus the **DO** token.

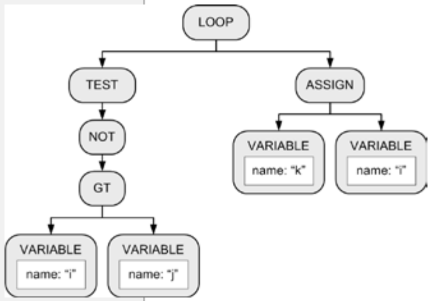
Class **WhileStatementParser**, *cont'd*

```
ICodeNode *WhileStatementParser::parse_statement(Token *token) throw (string)
{
    token = next_token(token);  // consume the WHILE

    ICodeNode *loop_node =
            ICodeFactory::create_icode_node((ICodeNodeType) NT_LOOP);
    ICodeNode *test_node =
            ICodeFactory::create_icode_node((ICodeNodeType) NT_TEST);
    ICodeNode *not_node =
            ICodeFactory::create_icode_node((ICodeNodeType) NT_NOT);

    loop_node->add_child(test_node);
    test_node->add_child(not_node);

    ExpressionParser expression_parser(this);
    not_node->add_child(expression_parser.parse_statement(token));

    token = synchronize(DO_SET);
    if (token->get_type() == (TokenType) PT_DO)
    {
        token = next_token(token);  // consume the DO
    }
    else {
        error_handler.flag(token, MISSING_DO, this);
    }

    StatementParser statement_parser(this);
    loop_node->add_child(statement_parser.parse_statement(token));

    return loop_node;
}
```

We're in this method because the parser has already seen **WHILE**.



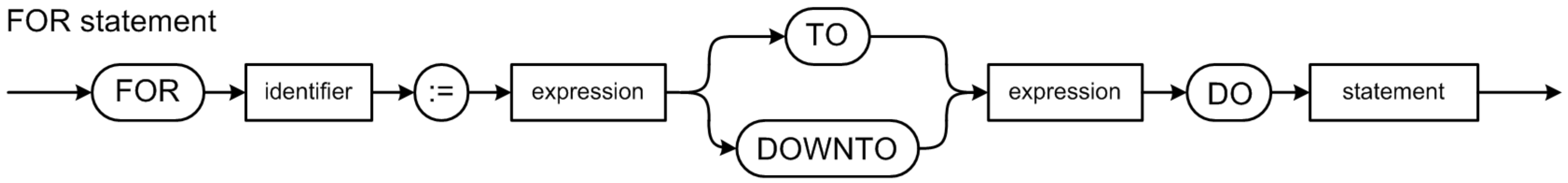
Synchronize the parser here! If the current token is not **DO**, then skip tokens until we find a token that is in **DO_SET**.



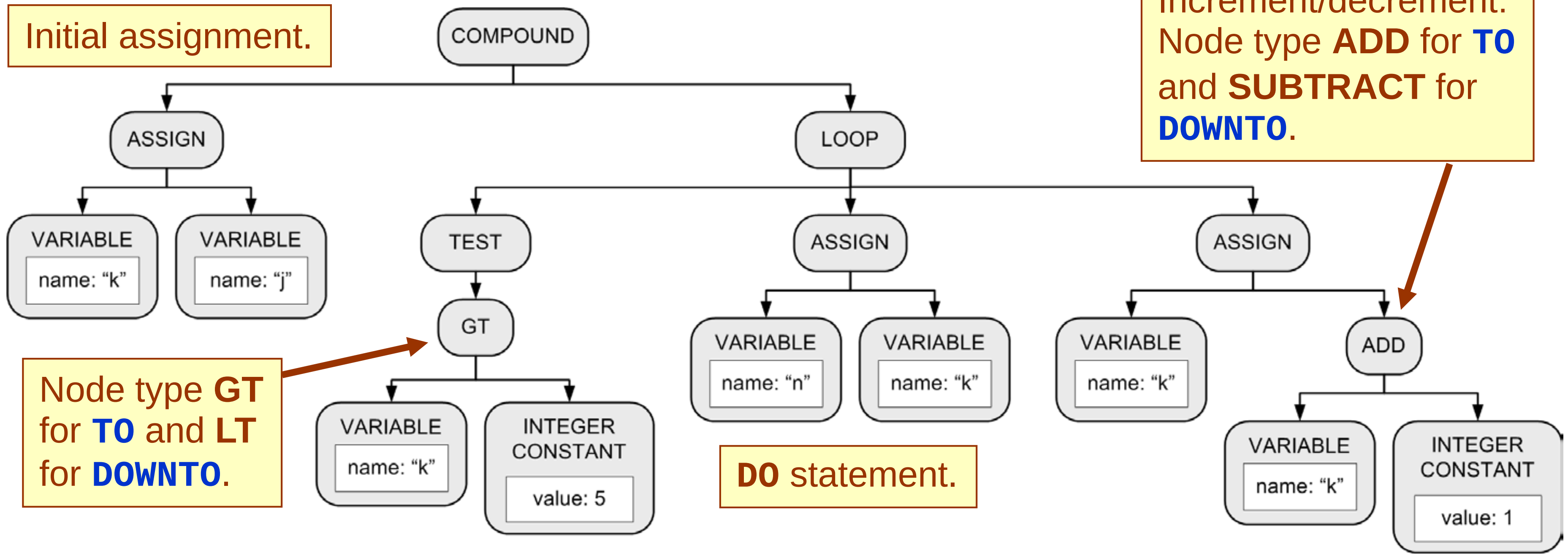
Pascal Syntax Checker II: **WHILE**

- We can recover (better) from syntax errors.
- Demo.
  - `./Chapter7cpp compile -i while.txt`
  - `./Chapter7cpp compile -i whileerrors.txt`

FOR Statement



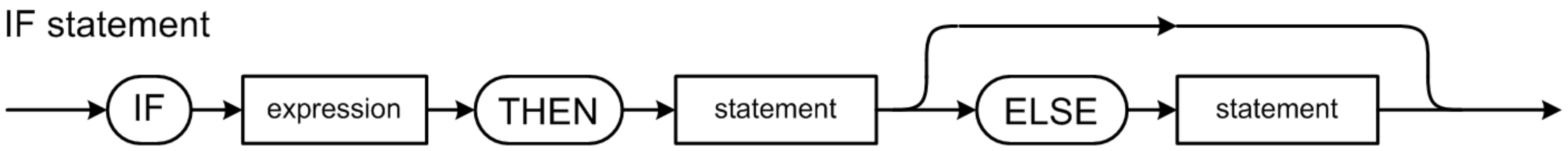
- Example: `FOR k := j TO 5 DO n := k`



Pascal Syntax Checker II: **FOR**

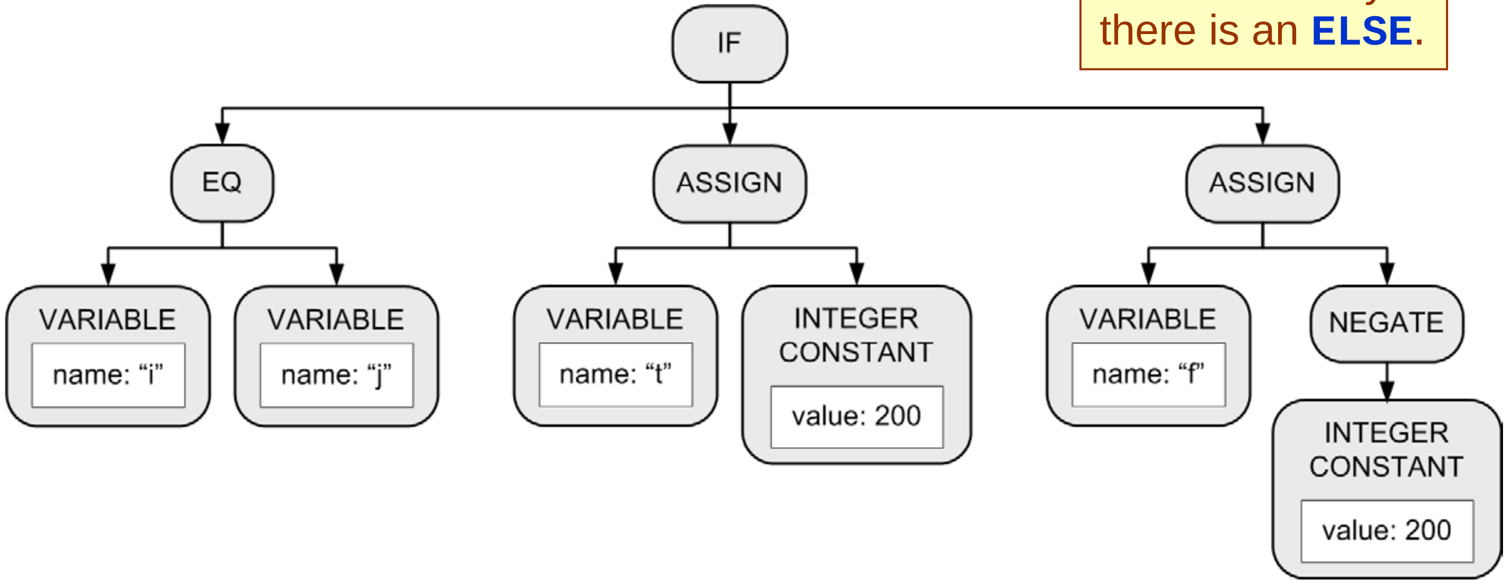
- Demo.
  - `./Chapter7cpp compile -i for.txt`
  - `./Chapter7cpp compile -i forerrors.txt`

IF Statement



- Example:

```
IF (i = j) THEN t := 200
           ELSE f := -200;
```



Third child only if there is an **ELSE**.

The “Dangling” **ELSE**

- Consider:

```
IF i = 3 THEN IF j = 2 THEN t := 500 ELSE f := -500
```

- Which **THEN** does the **ELSE** pair with?
  - Is it:

```
IF i = 3 THEN IF j = 2 THEN t := 500 ELSE f := -500
```

  - Or is it:

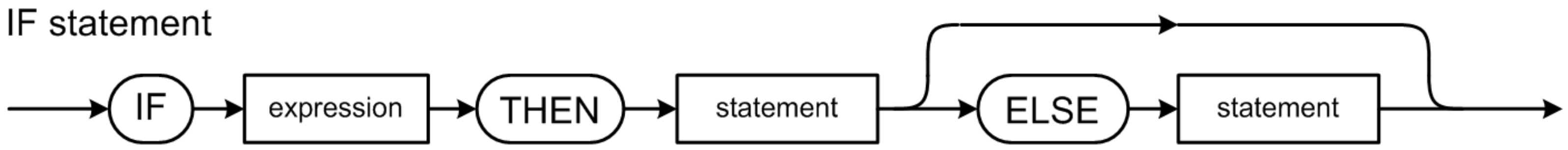
```
IF i = 3 THEN IF j = 2 THEN t := 500 ELSE f := -500
```

The "Dangling" **ELSE**, *cont'd*

- According to Pascal syntax, the nested **IF** statement is the **THEN** statement of the outer **IF** statement

```
IF i = 3 THEN IF j = 2 THEN t := 500 ELSE f := -500
```

- Therefore, the **ELSE** pairs with the closest (i.e., the second) **THEN**.



Scanner and Parser Rules of Thumb

- Scanner
  - At any point in the source file, extract the longest possible token.
    "maximum munch"
  - Example:
    - **<<=** is one shift-left-assign token
    - Not a shift-left token followed by an assign token

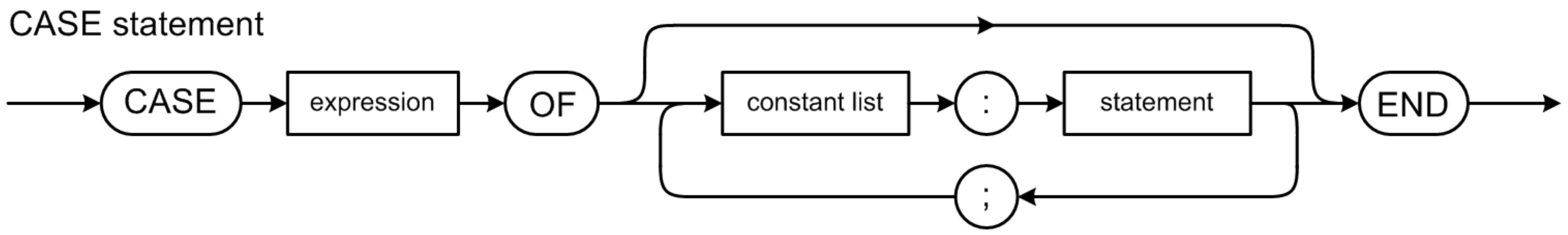
- Parser
  - At any point in the source file, parse the longest possible statement.
  - Example:

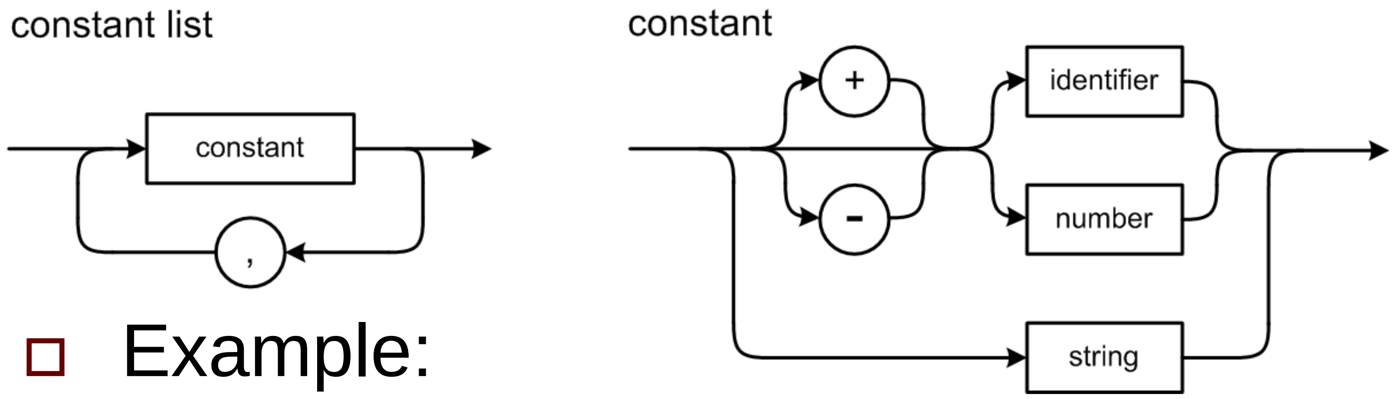
```
IF i = 3 THEN IF j = 2 THEN t := 500 ELSE f := -500
```

Pascal Syntax Checker II: **IF**

- Demo.
  - `java -classpath classes Pascal compile -i if.txt`
  - `java -classpath classes Pascal compile -i iftest.txt`

CASE Statement





- Example:

```
CASE i+1 OF
    1:        j := i;
    4:        j := 4*i;
    5, 2, 3: j := 523*i;
END
```

Note that Pascal's **CASE** statement does not use **BREAK** statements.

CASE Statement, *cont'd*

- Example:

  -
    ```
    CASE i+1 OF


    END
    ```

Pascal Syntax Checker II: **CASE**

- Demo.
  - `./Chapter7cpp compile -i case.txt`
  - `./Chapter7cpp compile -i caseerrors.txt`

Install GNU C++ on Windows

- Cygwin: https://www.cygwin.com/
- Install bash and the GNU g++ compiler.