CMPE 152: Compiler Design

October 5 Class Meeting

Department of Computer Engineering
San Jose State University

Fall 2017
Instructor: Ron Mak

www.cs.sjsu.edu/~mak

Midterm Review: Question 1

- The parser builds data structures consisting of symbol table entry objects (STEO) and type specification objects (TSO).

  a. How are these structures used at compile time?

     - STEO: Store information about locally-declared identifiers.
     - TSO: Represent type information.
     - Both: Assign types to variables.
     - Both: Do type checking in statements and expressions.

Midterm Review: Question 1, *cont'd*

b. How does the interpreter use these structures at run time?

- STEO: Create the memory map for each activation record that's pushed onto the runtime stack.
- STEO: Get the values of defined and enumeration constants.
- TSO: Do runtime range checking for subranges.
- TSO: Get the data types of array indexes and elements.

Midterm Review: Question 2

- How does the symbol table stack …

a. … allow a variable declared in an enclosing scope to be redeclared in the local scope?

- Push a new symbol table onto the symbol table stack for the local scope. Enter the variable’s name into the local table.

b. … prevent a variable already declared in the local scope from being redeclared in the local scope?

a.

- First check the local symbol table to see whether the variable’s name is already entered into the table.

Midterm Review: Question 2, *cont'd*

- How does the symbol table stack …
  - c. … allow two record types defined in the same procedure to declare fields with the same names?
    - Push a separate symbol table for each record type onto the symbol table stack when the parser is parsing the record type specification. Pop it off when the parser is done parsing the specification.

Midterm Review: Question 3

- Explain the role of synchronization sets during parsing.
  - Each synchronization set tells the parser what tokens to look for at a particular point. Some members of the set are tokens that are syntactically valid at that point, and other members are there to allow the parser to recover from syntax errors.

Midterm Review: Question 4

- Some programming languages use parentheses to enclose array subscripts and to enclose arguments to function calls. Suppose Pascal did the same. Consider the assignment statement:

  ```
  x := arr(i, 2*j)
  ```

  Describe how the Pascal parser could (or could not) determine whether it is parsing a call to function **`arr`** or an access to an element of array **`arr`**.

Midterm Review: Question 4, *cont'd*

```
x := arr(i, 2*j)
```

- By the time the parser is parsing the assignment statement, it would have already parsed the declaration of **`arr`** as either an array or a function.

  Therefore, when the parser encounters **`arr`** in the assignment statement, it looks in the symbol table to see how **`arr`** was declared.

© R. Mak

Midterm Review: Question 5

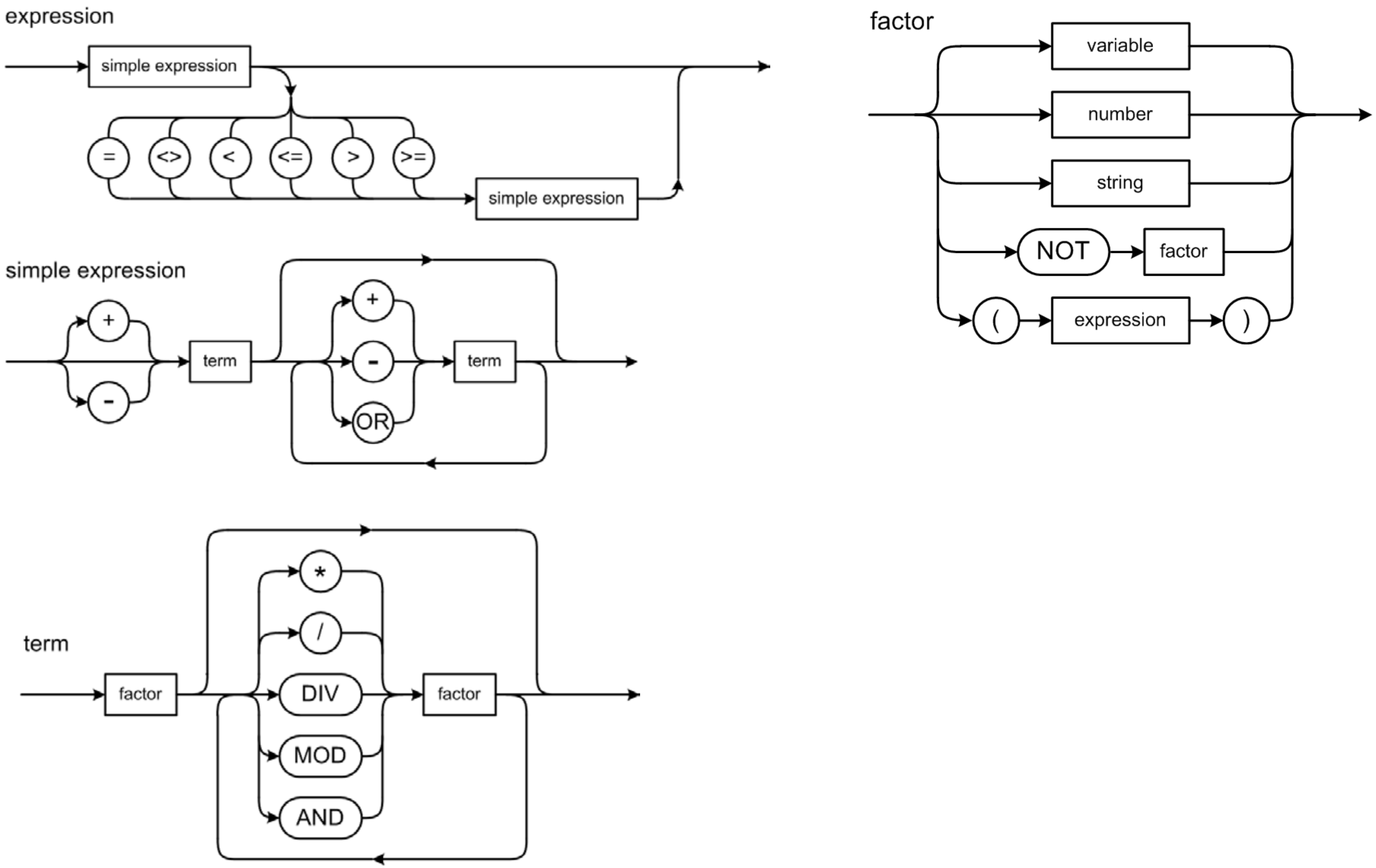
- Suppose Pascal has an exponentiation operator `**`, so that `n**4` represents the mathematical expression n^4.

  Exponentiation has a higher operator precedence level than the multiplicative operators. Therefore, `n**i*j` is evaluated as `(n**i)*j`. Furthermore, exponentiation is right-associative, so that `n**i**j**k` is evaluated as `n**(i**(j**k))`.

Midterm Review: Question 5

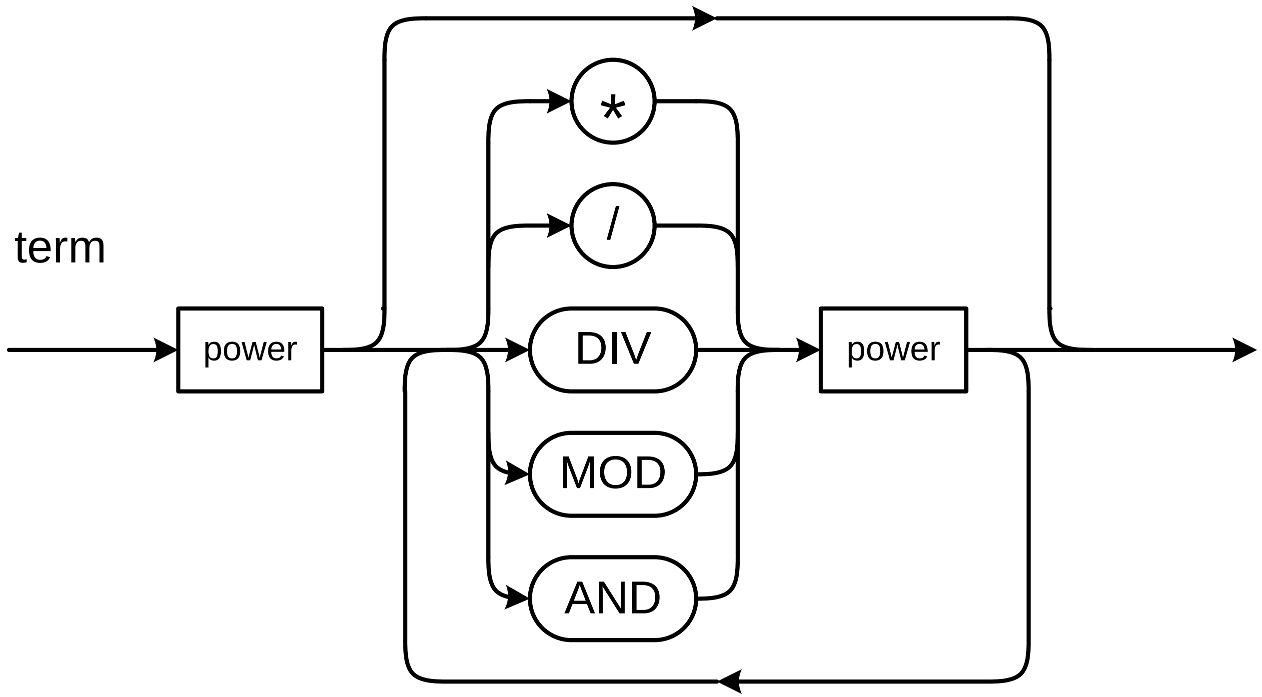
a. What modifications to the expression, simple expression, term, and factor syntax diagrams are required to accommodate the ** operator? Redraw only the diagrams that change. (Not all the diagrams may need to change, and you may need to add new diagrams.)

Midterm Review: Question 5, *cont'd*

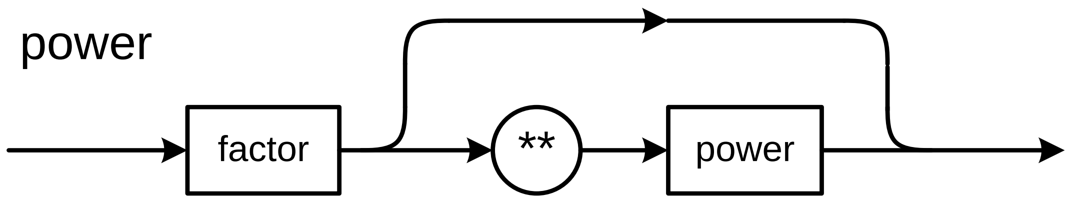


Midterm Review: Question 5, *cont'd*

term



power



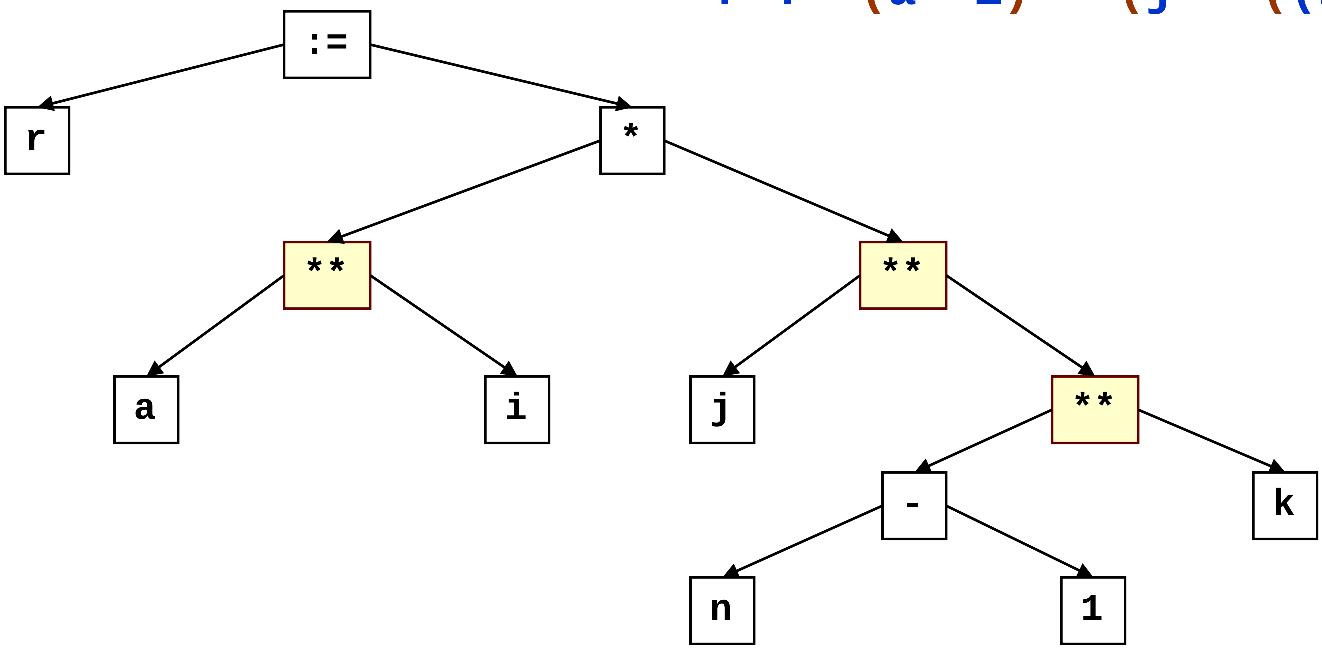
Midterm Review: Question 5, *cont'd*

b. Draw the parse tree for the assignment statement

```
r := a**i*j**(n-1)**k
```

You may assume the existence of a node type for the `**` operator.

```
r := (a**i) * (j** ((n-1)**k))
```



Midterm Review: Question 6

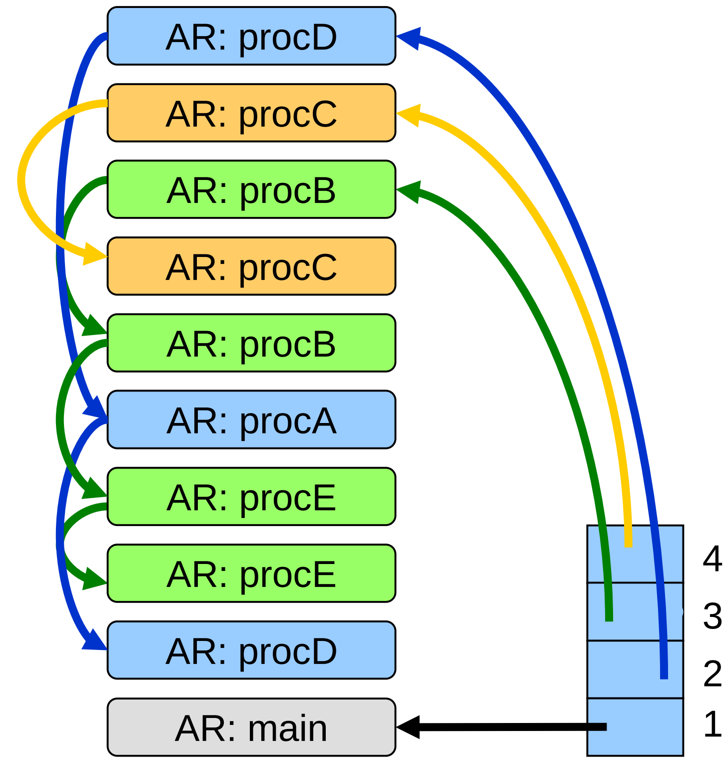
- Given the Pascal program structure and call chain:

```
PROGRAM main;
    PROCEDURE procA;
        PROCEDURE procB;
            PROCEDURE procC;

    PROCEDURE procD;
        PROCEDURE procE;
```

main → procD → procE → procE → procA → procB → procC → procB → procC → procD

Draw the runtime stack, activation records, and runtime display at the end of the call chain. Show the display pointers and the activation record links.



RUNTIME STACK

RUNTIME DISPLAY

Midterm Review: Question 7

- Suppose Pascal had a **LOOP-AGAIN** statement.

  For example:

```
LOOP
    k := k + 1;
    WHEN k > 10 LEAVE;
    j := 2*k
AGAIN
```

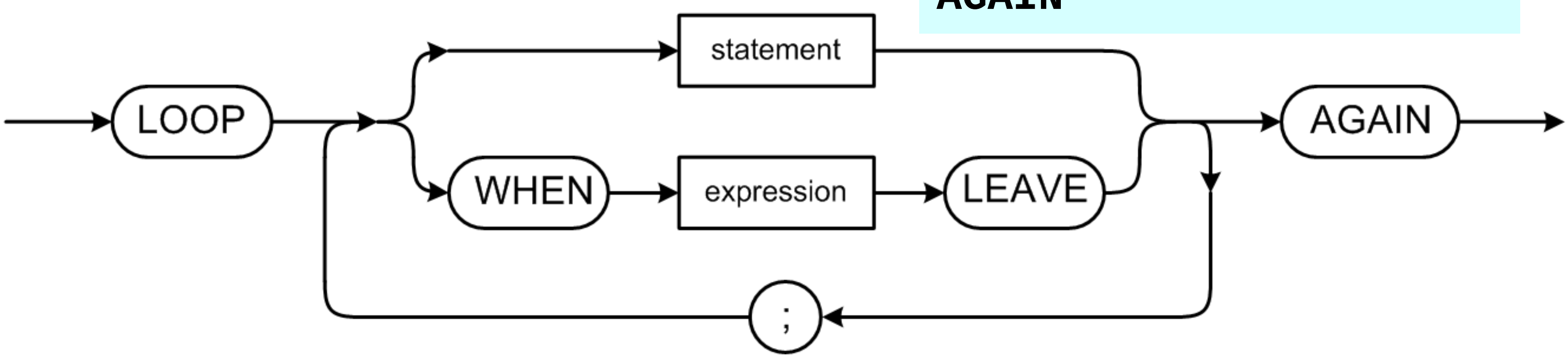
  Like the **REPEAT-UNTIL** statement, the **LOOP-AGAIN** statement can contain any number of statements that will be repeatedly executed. Any one or more (or none) of the contained statements can be a **WHEN-LEAVE** statement which will cause a break out of the loop if its Boolean expression evaluates to true. A **WHEN-LEAVE** statement can ***only*** appear inside of a **LOOP-AGAIN** statement.

Midterm Review: Question 7, *cont'd*

a. Draw syntax diagrams for the **LOOP-AGAIN** and **WHEN-LEAVE** statements. You may assume that diagrams already exist for "statement" and "expression" and that **LOOP**, **AGAIN**, **WHEN**, and **LEAVE** are reserved words.

```
LOOP
    k := k + 1;
    WHEN k > 10 LEAVE;
    j := 2*k
AGAIN
```

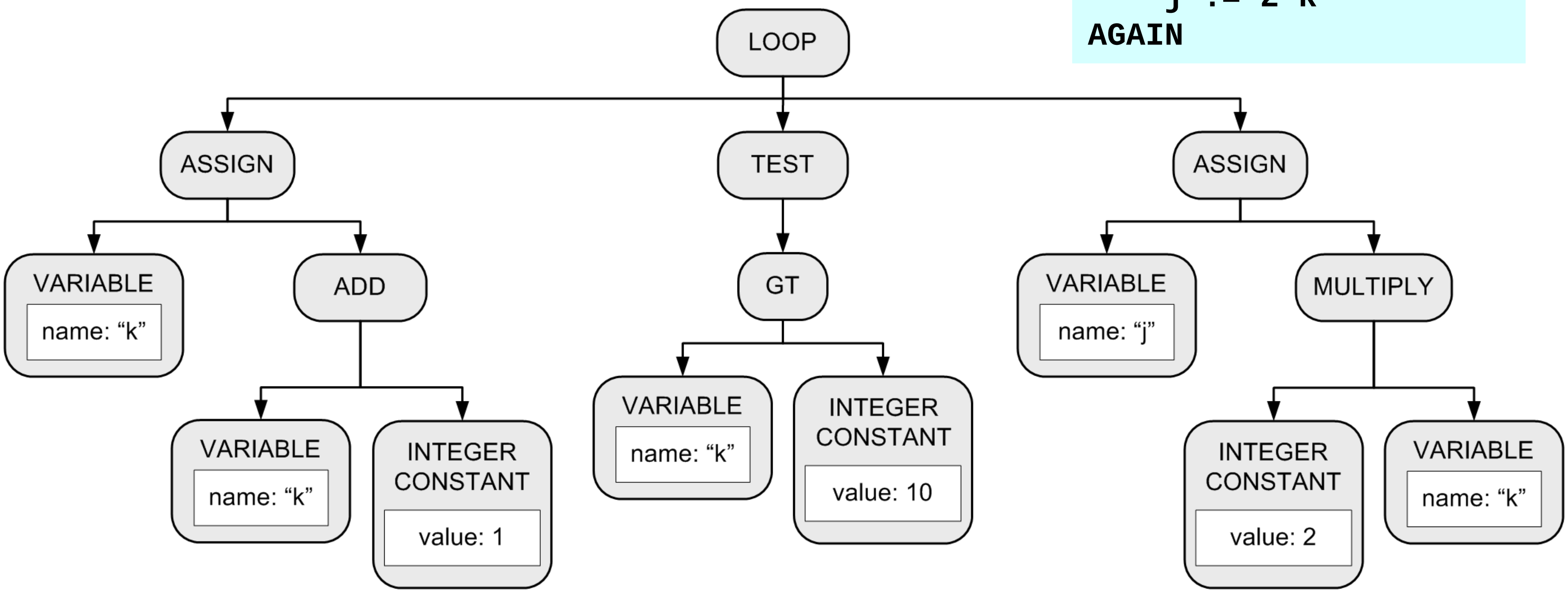
LOOP statement



Midterm Review: Question 7, *cont'd*

b. Using only the node types defined in class **ICodeNodeTypeImpl**, draw the parse tree for the **LOOP-AGAIN** statement shown on the previous page.

```
LOOP
    k := k + 1;
    WHEN k > 10 LEAVE;
    j := 2*k
AGAIN
```



Midterm Review: Question 8

- Suppose you wanted to modify the Pascal interpreter to execute **`GOTO`** statements.

Midterm Review: Question 8, *cont'd*

a. As it is currently designed, what problems would the executor in the back end encounter with **`GOTO`** statements?

- The executor recursively visits the nodes of the parse tree in order to execute the program. There is no easy way to jump from one node of the tree to a target node and have the executor continue walking the tree from the target node.

- If the **`GOTO`** is from one scope to a more outer scope, the executor needs to know how to pop off the right number of activation records from the runtime stack.

Midterm Review: Question 8, *cont'd*

b. Describe at the conceptual level a possible design change to the interpreter that would allow it to execute **GOTO** statements more easily.

- Before the executor starts, make pass over the parse tree of a routine by walking it to "linearize" the nodes into an array list.
- Add implicit GOTO nodes to branch back for loops and to branch around parts of IF THEN ELSE statements, etc.
- The executor simply executes the nodes sequentially from the start of the array list.
- If there is a jump to another node, the executor resumes sequential execution from the target node.

Midterm Review: Question 8, *cont'd*

- Each parse tree node must record its nesting level. If a jump is from one nesting level to another, the executor must pop off the correct number of activation records from the runtime stack.