

# CS 153: Concepts of Compiler Design

## 2008 Midterm and Solution

---

Department of Computer Science  
San Jose State University

Fall 2009

Instructor: Ron Mak

[www.cs.sjsu.edu/~mak](http://www.cs.sjsu.edu/~mak)

# Question 1

---

- Some of the early compilers and interpreters simply allocated a fixed-size block of memory for each procedure or function to hold that routine's runtime values.
  - a. [5 points] For what types of programming languages would this be appropriate?
    - Any language that does not allow recursion or dynamic data (malloc/free, new/dispose, etc.)
  - b. [5 points] Why wouldn't this work for the Pascal language?
    - Pascal supports both recursion and dynamic data.

## Question 2

---

- The symbol table stack has two lookup methods, `lookup()` to search the entire stack, and `lookupLocal()` to search only the symbol table at the top of the stack.
  - a. [10 points] When is it appropriate to use method `lookup()`?
    - Whenever the parser must search the entire stack to look for the declarations of identifiers, such as while parsing expressions. The identifiers can be declared either in the local scope or in an enclosing scope.
  - b. [10 points] When is it appropriate to use method `lookupLocal()`?
    - While parsing declarations, when the parser needs to verify that an identifier is not already declared in the current scope. If `lookupLocal()` returns a symbol table entry (instead of null), then the identifier is being redeclared in the same scope, which is an error.

## Question 3

---

- Suppose Pascal had a **LOOP-AGAIN** statement such as this example:

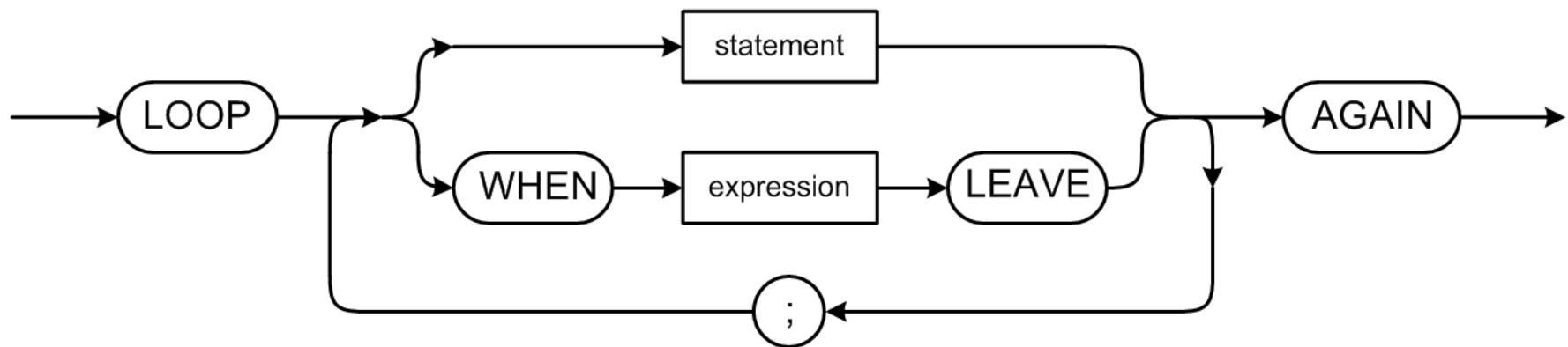
```
LOOP
    k := k + 1;
    WHEN k > 10 LEAVE;
    j := 2*k;
AGAIN
```

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.

## Question 3 (cont'd)

- a. [15 points] 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.

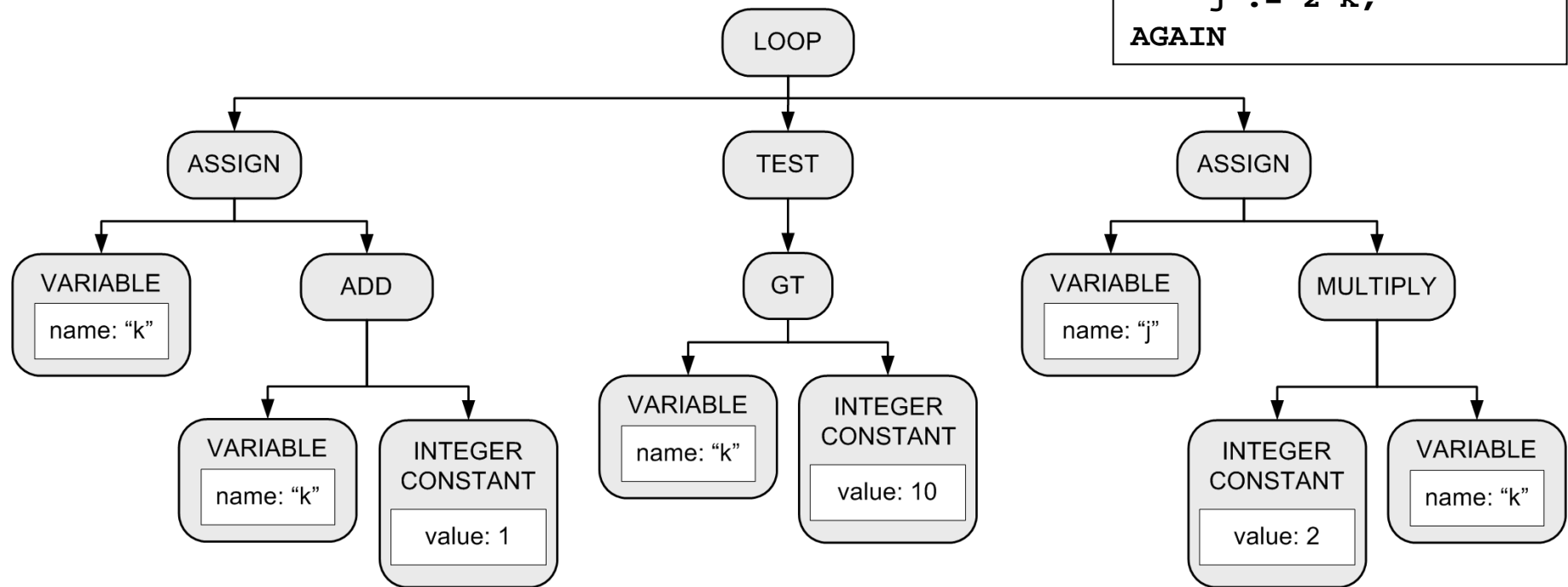
LOOP statement



## Question 3 (cont'd)

- b. [15 points] 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
```



## Question 4

---

- [10 points] Describe how some of the software engineering techniques we employed during development helped us overcome the size and complexity of the Pascal interpreter.
  - **Partitioning** into front end, middle tier, and back end.
    - Divide and conquer
    - Support parallel development
  - **Use of interfaces**
    - Contract between developers
    - Support parallel development
  - **Code to the interfaces**
    - Don't depend on the implementation of a component, only on the interface contract.
    - Support parallel development

## Question 4 (*cont'd*)

---

- **Loose coupling** among components.
  - Changes in one component have fewer effects on other components
- Use of **design patterns**
  - Proven techniques for solving common architectural problems.
- **Framework** development
  - Early integration of all components
  - Early end-to-end thread of execution
  - Separate language-independent framework from language-dependent components
- **Always build on code that's already working**
  - No “big bang” integration effort at the final deadline
- Use **IDE tools** such as Eclipse
  - Set breakpoints, examine at the call stack, etc.

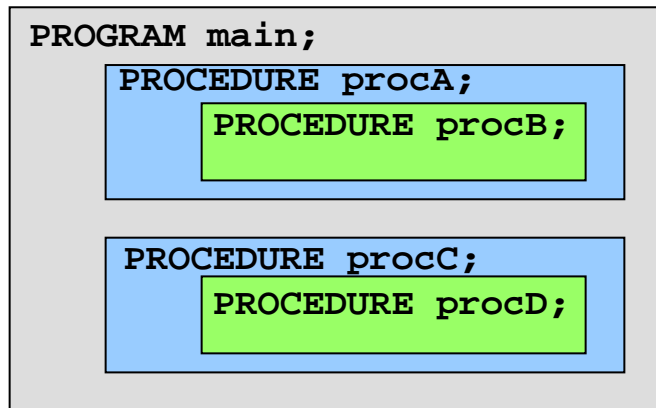
## Question 5

---

- When parsing a Pascal **WITH** statement that has a single record variable:
  - a. [3 points] What is pushed onto a stack?
    - The symbol table of the record variable's type
  - b. [2 points] Which stack?
    - The symbol table stack
  - c. [5 points] For what purpose?
    - To enable the parser to recognize fields of the record type while it is parsing the **DO** statement

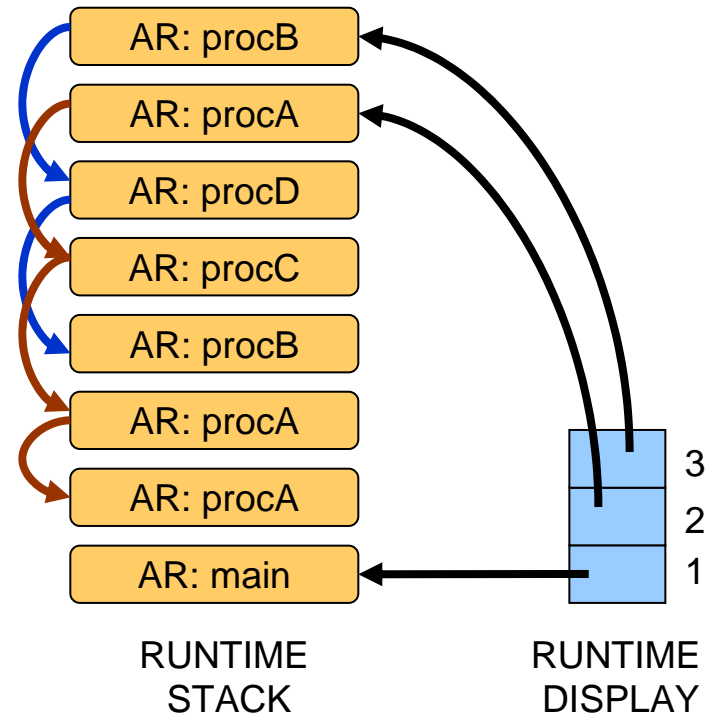
# Question 6

- Given the following Pascal program structure:

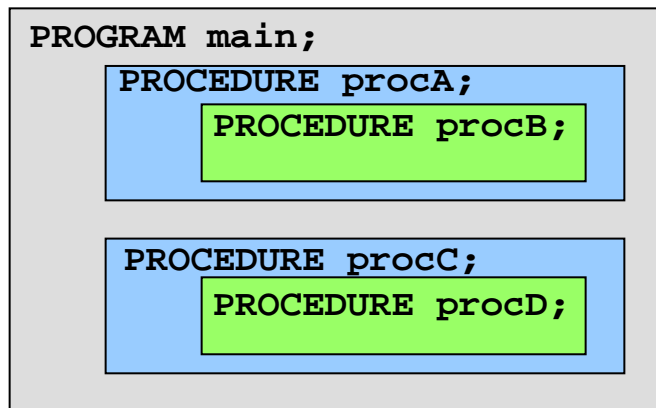


main → procA → procA → procB → procC → procD → procA → procB

- a. [10 points] Draw the runtime stack, activation records, and runtime display at the end of the following call chain. Show display pointers and activation record links.



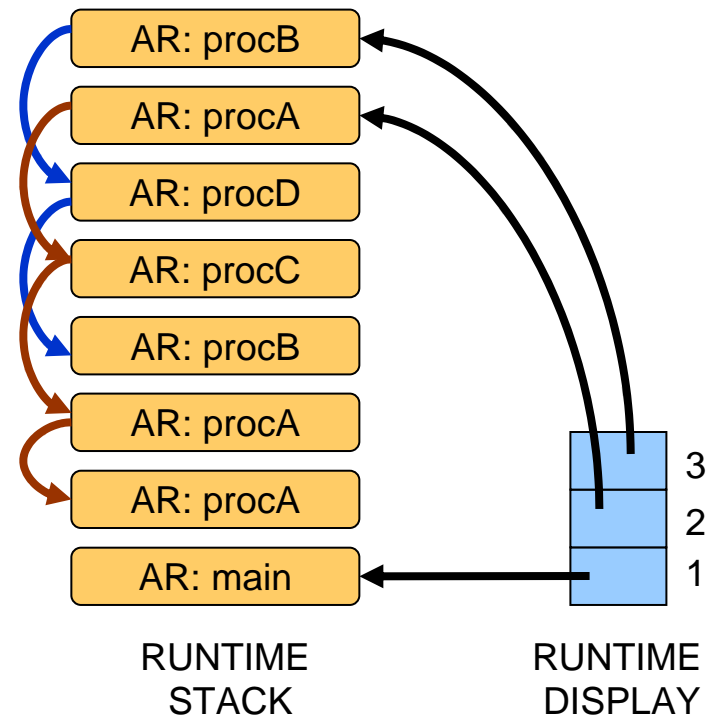
## Question 6 (cont'd)



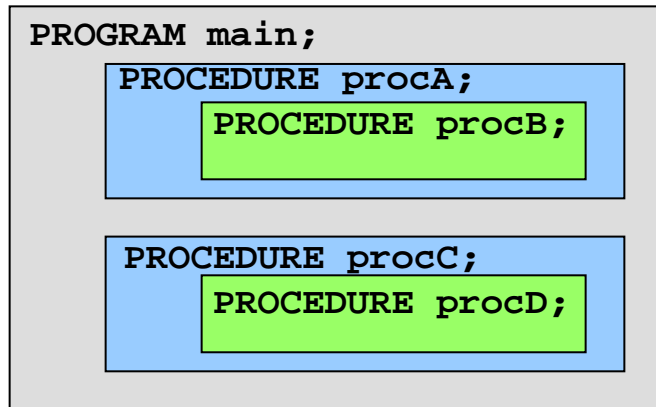
main → procA → procA → procB → procC → procD → procA → procB

- b. [5 points] At run time, how does a statement in **procB** access variables declared in **procA** and in **main**?

- For variables declared in **procA**: Use runtime display element 2 to access **AR:procA**. For variables declared in **main**: Use runtime display element 1 to access **AR:main**.



## Question 6 (cont'd)



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

- c. [5 points] At run time, what prevents a statement in **procB** from accessing a variable declared in **procC**?
- If the front end parser had done its job right, it would not have allowed this access because of the scope rules. Therefore, the back end executor would never see this access in the parse tree.

