CÓRDOBA UNIVERSITY

SUPERIOR POLYTECHNIC SCHOOL

DEPARTMENT OF
COMPUTER SCIENCE AND NUMERICAL ANALYSIS

DECLARATIVE PROGRAMMING

COMPUTER ENGINEERING

COMPUTATION ESPECIALITY

FOURTH YEAR

FIRST FOUR-MONTH PERIOD

Subject 1.- Introduction to Scheme language

DECLARATIVE PROGRAMMING — PROGRAM

First part: Scheme

- Subject 1.- Introduction to Scheme language
- Subject 2.- Expressions and Functions
- Subject 3.- Conditional Predicates and Sentences
- Subject 4.- Iteration and Recursion
- Subject 5.- Compound Data Types
- Subject 6.- Data Abstraction
- Subject 7.- Reading and Writing

Second part: Prolog

- Subject 8.- Introduction to Prolog language
- Subject 9.- Basic Elements of Prolog
- Subject 10.- Lists
- Subject 11.- Re-evaluation and the "cut"
- Subject 12.- Input and Output

2

DECLARATIVE PROGRAMMING — PROGRAM

First part: Scheme

- Subject 1.- Introduction to Scheme language
- Subject 2.- Expressions and Functions
- Subject 3.- Conditional Predicates and Sentences
- Subject 4.- Iteration and Recursion
- Subject 5.- Compound Data Types
- Subject 6.- Data Abstraction
- Subject 7.- Reading and Writing

3

Contents

Contents

1. **Fundamental Characteristics of Functional Programming**
   - ✓ **Functional** Programming is a **subtype** of **Declarative** Programming

1. **Fundamental Characteristics of Functional Programming**

- ✓ **Declarative Programming (1 / 2)**
  - ➢ **Objective**: **Problem description**

    "What" problem must be resolved?

    - **Notice**:
      - It does **not** mind "**how**" the problem is resolved
      - It **avoids** the implementation features.

7

1. **Fundamental Characteristics of Functional Programming**

- ✓ **Declarative Programming (2 / 2)**
  - ➢ **Features**
    - Expressivity
    - Extensible: 10% - 90% rule
    - Protection
    - Mathematic Elegance
  - ➢ **Types**:
    - **Functional** *or* Applicative Programming:
      - Lisp, **Scheme**, Haskell, ...
    - **Logic** Programming: **Prolog**

8

1. **Fundamental Characteristics of Functional Programming**

- ✓ **Principle** of the **"Pure"** **Functional** Programming

  *"The **expression value** only **depends on** its **sub-expressions** values, if such sub-expressions exist ".*
- ✓ **Non collateral effects**

  The value of "a + b" **only** depends on "a" and "b".
- ✓ The **function** term is used in its **mathematical** sense.
- ✓ **No instructions**: programming **without** assignments
  - ➢ The **impure** Functional programming **allows** the

    "assignment instruction"

9

1. **Fundamental Characteristics of Functional Programming**
   - ✓ **Program structure** in **Functional** Programming
     - ➢ **The program is a function composed of simpler functions**
     - ➢ **Function execution:**
       - ▪ **Receives the input data**: functions arguments or parameters
       - ▪ **Evaluates the expressions**
       - ▪ **Returns the Result**: computed value of the function

10

1. **Fundamental Characteristics of Functional Programming**
   - ✓ **Type of Functional Languages**
     - ➢ **Most** of them are **interpreted** languages
     - ➢ Some of them have **compiled** versions
   - ✓ **Memory management**
     - ➢ **Implicit memory management**:
       - ▪ Memory management is a task of the interpreter.
       - ▪ The programmer must **not** worry about memory management.
     - ➢ **Garbage collection**: task of the interpreter.

In short: the programmer must only worry about the **Problem description**

11

Contents

12

2. **Historic Summary of Scheme**
   - ✓ LISP
   - ✓ Compilation versus Interpretation
   - ✓ Lexical (or static) versus dynamical scope
   - ✓ Origin of Scheme

13

2. **Historic Summary of Scheme**
   - ✓ **LISP**
   - ✓ Compilation versus Interpretation
   - ✓ Lexical (or static) versus dynamical scope
   - ✓ Origin of Scheme

14

2. **Historic Summary of Scheme**
   - ✓ **LISP**
     - ➢ **John McCarthy** (MIT)
     - ➢ **"Advice Taker"** program:
       - Theoretical basis: Logic Mathematics
       - Objective: Deduction and Inferences
     - ➢ **LISP**: **LIS**t **P**rocessing (1956 – 1958)
       - Second historic language of **Artificial Intelligence** (after IPL)
       - At present time, second historic language **in use** (after Fortran)
       - LISP is based on Lambda Calculus (**Alonzo Church**)
     - ➢ **Scheme** is a **dialect** of **LISP**

15

2. **Historic Summary of Scheme**

- ✓ **LISP**
  - ➢ **Functional Programming Characteristics**
    - **Recursion**
    - **Lists**
    - **Implicit** memory management
    - Interactive and **interpreted** programs
    - **Symbolic** Programming
    - **Dynamically scoped** for **non** local variables

16

2. **Historic Summary of Scheme**

- ✓ **LISP**
  - ➢ LISP's **contributions**:
    - **Built - in** functions
    - **Garbage collection**
    - **Definition Formal Language**: **LISP itself**

17

2. **Historic Summary of Scheme**

- ✓ **LISP**
  - ➢ **Applications**: **Artificial Intelligence** Programs
    - Theorem verification and testing
    - Symbolic differentiation and integration
    - Search Problems
    - Natural Language Processing
    - Computer Vision
    - Robotics
    - Knowledge Representation Systems
    - Expert Systems
    - And so on

18

2. **Historic Summary of Scheme**

- ✓ **LISP**
  - ➢ **Dialects (1 /2)**
    - **Mac LISP** (Man and computer or Machine – aided cognition): **East** Coast Version
    - **Inter LISP** (Interactive LISP): **West** Coast Version
      - Bolt, Beranek y Newman Company (BBN)
      - Research Center of Xerox at Palo Alto (Texas)
      - **LISP Machine**

19

2. **Historic Summary of Scheme**

- ✓ **LISP**
  - ➢ **Dialects** (2 / 2)
    - **Mac LISP** (Man and computer or Machine – aided cognition): East Coast Version
      - C-LISP: Massachusetts University
      - Franz LISP: California University (Berkeley). **Compiled version**.
      - NIL (New implementation of LISP): MIT.
      - PSL (Portable Standard LISP): Utah University
      - **Scheme**: MIT.
      - T (True):Yale University.
      - Common LISP

20

2. **Historic Summary of Scheme**

- ✓ LISP
- ✓ **Compilation versus Interpretation**
- ✓ Lexical (or static) versus dynamical scope
- ✓ Origin of Scheme

21

2. **Historic Summary of Scheme**

- ✓ **Compilation versus interpretation**
  - ➢ **Compilation**:
    - The **source code** (**high level**) is **transformed** into **executable code** (**low level**), which can be independently run.

22

2. **Historic Summary of Scheme**

- ✓ **Compilation versus interpretation**
  - ➢ **Compilation**

Source code → **Compiler**

23

2. **Historic Summary of Scheme**

- ✓ **Compilation versus interpretation**
  - ➢ **Compilation**

Source code → **Compiler**

↓

Compilation errors

24

2. Historic Summary of Scheme

- ✓ Compilation versus interpretation
  - ➢ Compilation

Source code → Compiler → Executable code

25

2. Historic Summary of Scheme

- ✓ Compilation versus interpretation
  - ➢ Compilation

Input data
↓
Source code → Compiler → Executable code

26

2. Historic Summary of Scheme

- ✓ Compilation versus interpretation
  - ➢ Compilation

Input data
↓
Source code → Compiler → Executable code
↙ ↓
Execution errors
Output

27

2. **Historic Summary of Scheme**

- ✓ **Compilation versus interpretation**
  - ➢ **Compilation**

Input data

↓

Source code → **Compiler** → **Executable code**

↓

Output

28

2. **Historic Summary of Scheme**

- ✓ **Compilation versus interpretation**
  - ➢ **Interpretation**

29

2. **Historic Summary of Scheme**

- ✓ **Compilation versus interpretation**
  - ➢ **Interpretation** or **simulation**: consists of a cycle of three stages

30

2. **Historic Summary of Scheme**

- ✓ **Compilation versus interpretation**
  - ➢ **Interpretation** or simulation: consists of a cycle of three stages
    1. **Analysis**: the source code is analysed to determine the following correct sentence to be run.

31

2. **Historic Summary of Scheme**

- ✓ **Compilation versus interpretation**
  - ➢ **Interpretation** or simulation: consists of a cycle of three stages
    1. **Analysis**: the source code is analysed to determine the following correct sentence to be run.
    2. **Generation**: the sentence is transformed into executable code.

32

2. **Historic Summary of Scheme**

- ✓ **Compilation versus interpretation**
  - ➢ **Interpretation** or simulation: consists of a cycle of **three** stages
    1. **Analysis**: the source code is analysed to determine the following correct sentence to be run.
    2. **Generation**: the sentence is transformed into executable code.
    3. **Execution**: the executable code is run.

33

2. Historic Summary of Scheme

- ✓ Compilation versus interpretation
  - ➢ Interpretation

Source code → Interpreter

34

2. Historic Summary of Scheme

- ✓ Compilation versus interpretation
  - ➢ Interpretation

Input data ↓

Source code → Interpreter ↗ Interpretation errors

35

2. Historic Summary of Scheme

- ✓ Compilation versus interpretation
  - ➢ Interpretation

Input data ↓

Source code → Interpreter ↘ Execution errors

↓ Output

36

2. **Historic Summary of Scheme**

- ✓ **Compilation versus interpretation**
  - ➢ **Interpretation**

Input data

↓

Source code → **Interpreter**

↓

Output

37

2. **Historic Summary of Scheme**

- ✓ **Compilation versus interpretation**

▪ **Compilation**

- Independent
- Memory necessities
- Efficient
- Global
- No interaction
- Closed code during execution

▪ **Interpretation**

- Dependent
- No memory necessities
- Less efficient
- Local
- Interaction
- Open code during execution

38

2. **Historic Summary of Scheme**

- ✓ LISP
- ✓ Compilation versus Interpretation
- ✓ **Lexical (or static) versus dynamical scope**
- ✓ Origin of Scheme

39

2. **Historic Summary of Scheme**

- ✓ **Lexical (or static) versus dynamical scope**
  - ➢ The **scope rules** determine the **declaration** of **non** local identifiers
  - ➢ **Non** local identifiers:
    - **Variables** or **functions** which can be **used** in a function or procedure but are **not** declared in that function or procedure
  - ➢ **Two types**
    - **Lexical or static scope**
      - **With** "blocks structure": Pascal, **Scheme**
      - **Without** "blocks structure": C, Fortran
    - **Dynamical scope:**
      - **Always with** "blocks structure": Lisp, SNOBOL, APL

40

2. **Historic Summary of Scheme**

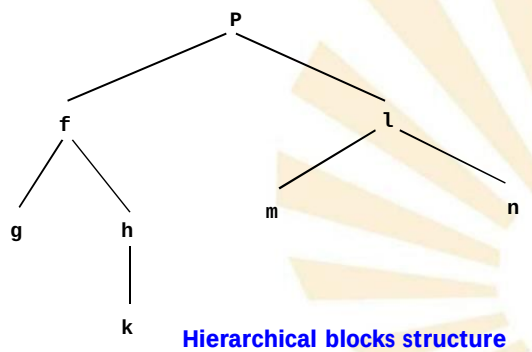
- ✓ **Lexical (or static) versus dynamical scope**
  - ➢ **Block structure**
    - A procedure or function can **call**
      - Itself
      - Its children (but **not** its grandchildren...)
      - Its brothers (but **not** its nephews)
      - Its father, grandfather, great-grandfather, ...
      - The brothers of its father, grandfather, ...
    - A procedure or function can **be called** by
      - Itself
      - Its father (but **not** by its grandfather, ...)
      - Its children, grandchildren, great-grandchildren, ...
      - Its brothers and their children, grandchildren, ...

41

Example of blocks structure

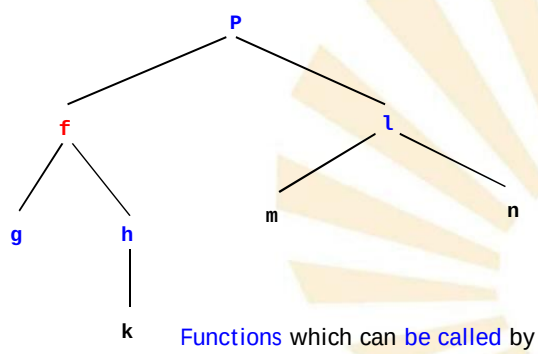
```
P
  Declaration of procedure f
    Declaration of procedure g
    Declaration of procedure h
      Declaration of procedure k
  Declaration of procedure l
    Declaration of procedure m
    Declaration of procedure n
```

42



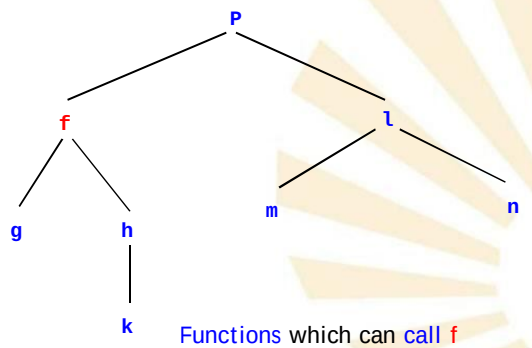
Hierarchical blocks structure

43



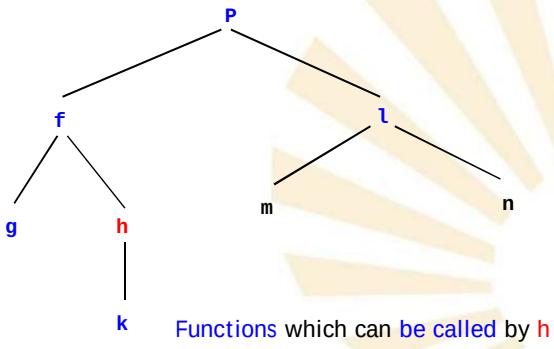
Functions which can be called by **f**

44



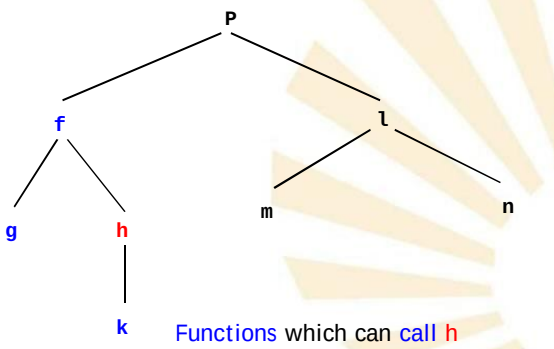
Functions which can call f

45



Functions which can be called by h

46



Functions which can call h

47

2. **Historic Summary of Scheme**
   - ✓ **Lexical (or static) versus dynamical scope**
     - ➢ **Lexical or static scope**
       - The **declaration** of a **non** local identifier depends on the **closest lexical context**
       - **The closest nesting rules**

48

2. **Historic Summary of Scheme**

✓ **Lexical (or static) versus dynamical scope**

➢ **Lexical or static scope**

- The **declaration** of a **non** local identifier depends on the **closest lexical context:**

  You only have to **read** the program

  to determine the declaration of an identifier.

49

2. **Historic Summary of Scheme**

✓ **Lexical (or static) versus dynamical scope**

➢ **Lexical or static scope**

- **The closest nesting rules:**
  - The **scope** of a procedure (*) **f** includes the procedure **f**.
  - If a **non** local identifier **x** is used in **f** then the declaration of **x** must be found in the **closest** procedure **g** which includes **f**
  - **Notice** (*) : procedure, function or block

50

Example.

Lexical scope

with

"block structure"

```
Declaration of procedure h
 Declaration of variable x   (x1)
 Declaration of variable y   (y1)
 Declaration of variable z   (z1)

 Declaration of procedure g
  Declaration of variable x (x2)
  Declaration of variable y (y2)

  Declaration of procedure f
    Declaration of variable x (x3)

    Use of x   (→ x3)
    Use of y   (→ y2)
    Use of z   (→ z1)

    Use of x   (→ x2)
    Use of y   (→ y2)
    Use of z   (→ z1)
    Call to f

    Use of x   (→ x1)
    Use of y   (→ y1)
    Use of z   (→ z1)
    Call to g
```

51

2. **Historic Summary of Scheme**

- ✓ **Lexical (or static) versus dynamical scope**
  - ➢ **Lexical or static scope**
    - **Without** block structure:
      - If **x** is **not** local for a **specific** function then it is **not** local for **all** functions

52

Example in C:

without

"block structure"

```
int x;   /* x1 */
int y;   /* y1 */
int z;   /* z1 */

main()
        {
   int x;   /* x2 */
   int y;   /* y2 */

   /* Use of x → x2 */
   /* Use of y → y2 */
   /* Use of z → z1 */
   /* Call to f */
   f ();
  }

 f()
  {
   int x;   /* x3 */
   /* Use of x → x3 */
   /* Use of y → y1 */
   /* Use of z → z1 */
   }
```

Global variables are not recommended

53

2. **Historic Summary of Scheme**

- ✓ **Lexical (or static) versus dynamical scope**
  - ➢ **Dynamical scope:**
    - The **declaration** of an **identifier** depends on the **execution of the program**
    - **The closest activation rules**

54

2. **Historic Summary of Scheme**

✓ **Lexical (or static) versus dynamical scope**

➢ **Dynamical scope:**

- The **declaration** of an **identifier** depends on the **execution of the program**

  You have to **run** the program

  to determine the declaration of an identifier

55

2. **Historic Summary of Scheme**

✓ **Lexical (or static) versus dynamical scope**

➢ **Dynamical scope:**

- **The closest activation rules:**
  - The **scope** of a procedure (*) **f** includes the procedure **f**.
  - If a **non** local identifier **x** is used in the **activation** of **f** then the declaration of **x** must be found in the **closest active** procedure **g** with a declaration of x
  - **Notice** (*) : procedure, function or block

56

2. **Historic Summary of Scheme**

✓ **Lexical (or static) versus dynamical scope**

➢ **Notice**:

- The **dynamical scope** allows that an **identifier** can be associated to **different declarations** during the program execution

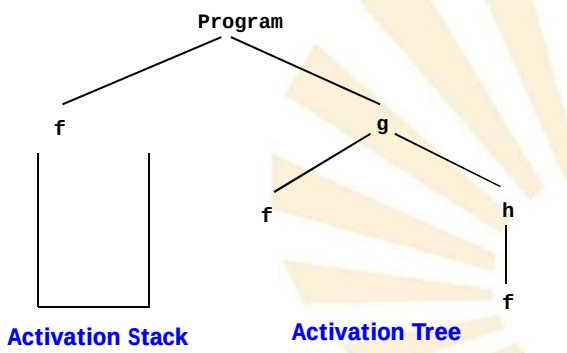
57

Example:

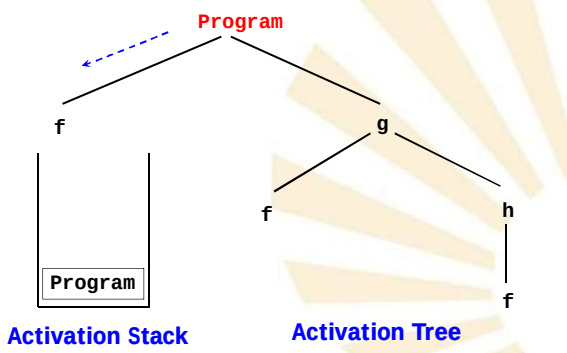
Lexical versus Dynamical scope

```
Program
    Declaration of variable x

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

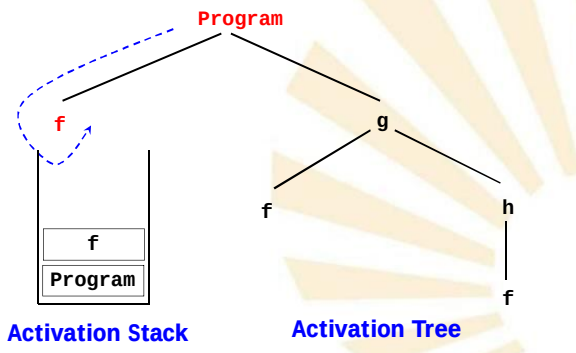
58



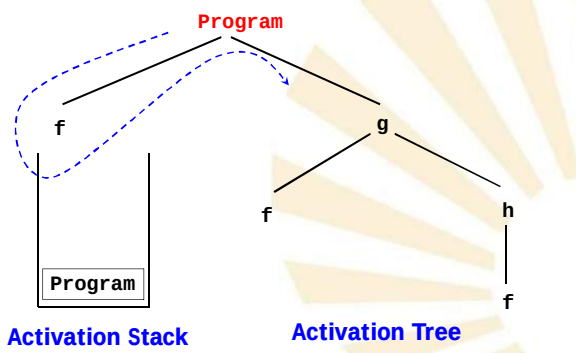
59



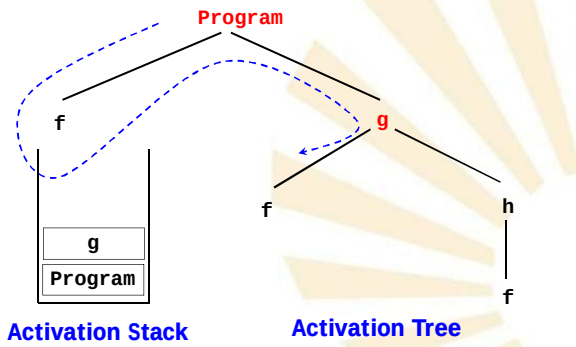
60


Program
f
g
f
h
f
f
Program
Activation Stack
Activation Tree

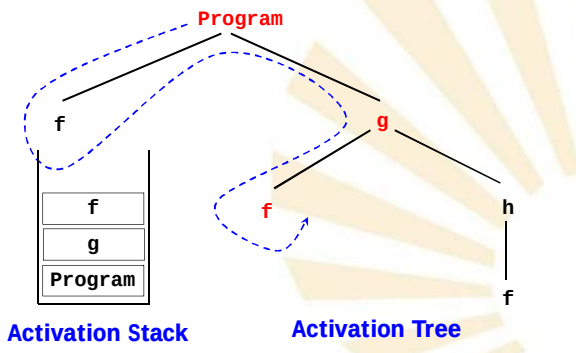
61


Program
f
g
f
h
f
Program
Activation Stack
Activation Tree

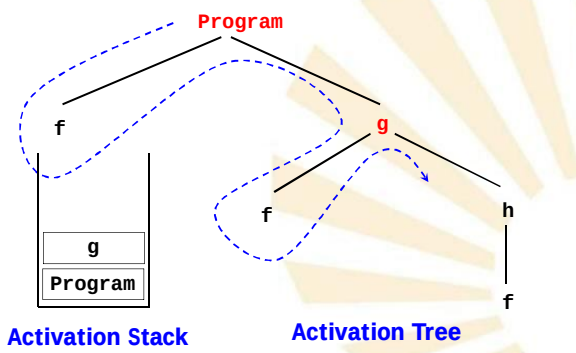
62


Program
f
g
f
h
f
g
Program
Activation Stack
Activation Tree

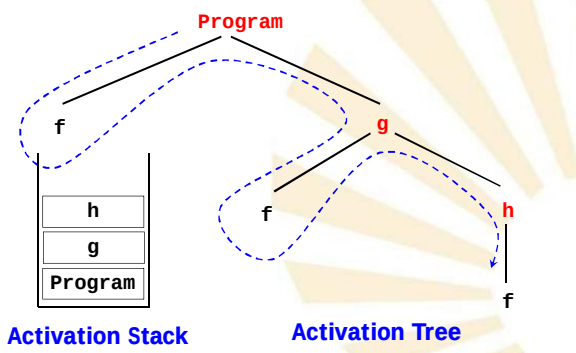
63


Program
f
g
f
h
f
f
g
Program
Activation Stack
Activation Tree

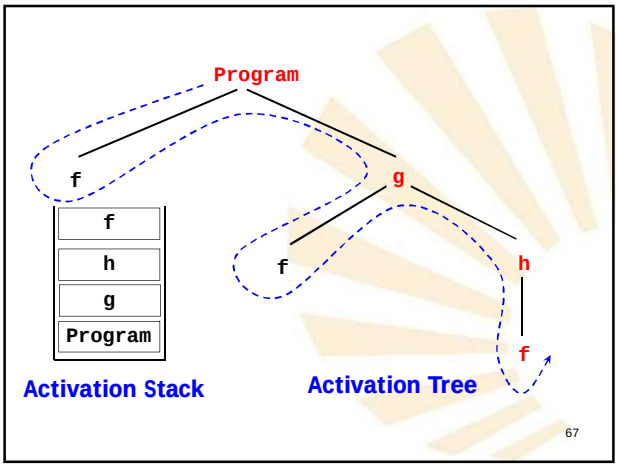
64

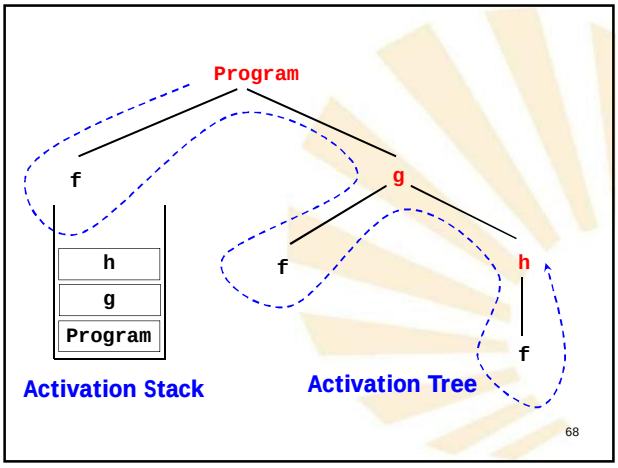

Program
f
g
f
h
f
g
Program
Activation Stack
Activation Tree

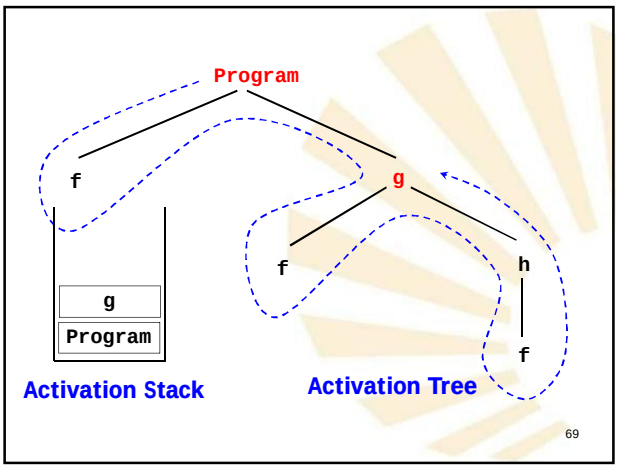
65

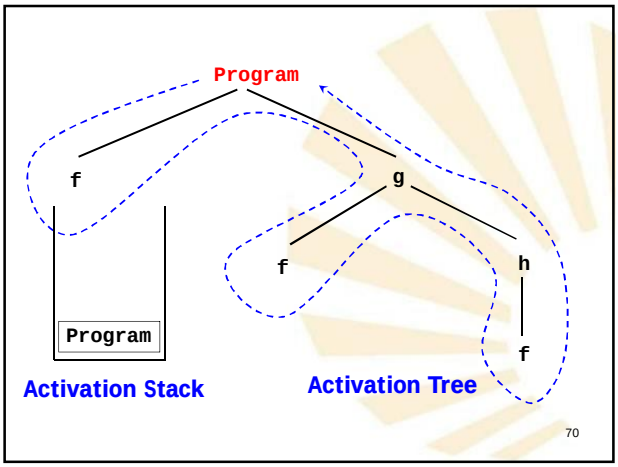

Program
f
g
f
h
f
h
g
Program
Activation Stack
Activation Tree

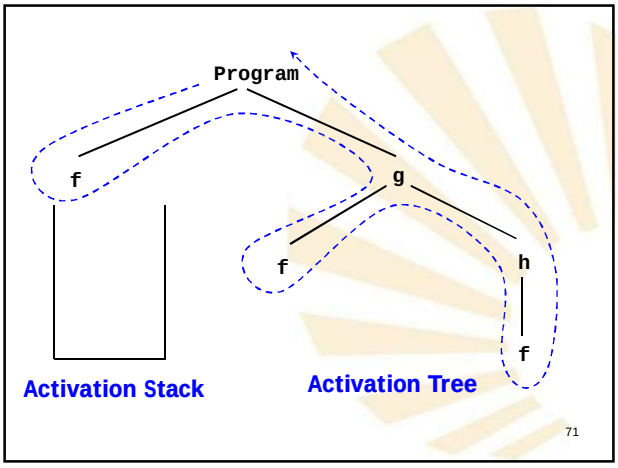
66

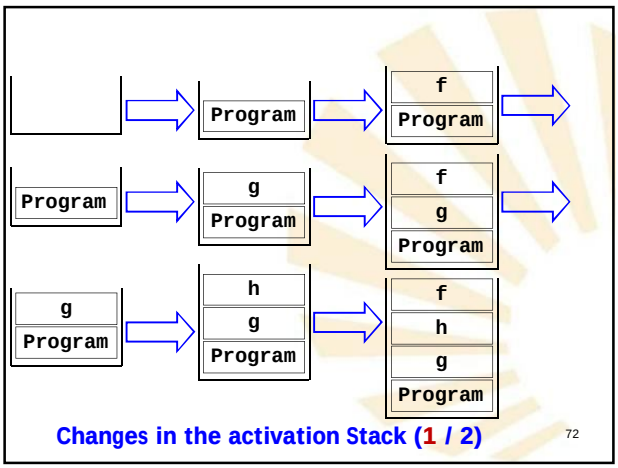

Program
f
g
f
h
g
Program
f
h
f
Activation Stack
Activation Tree
67

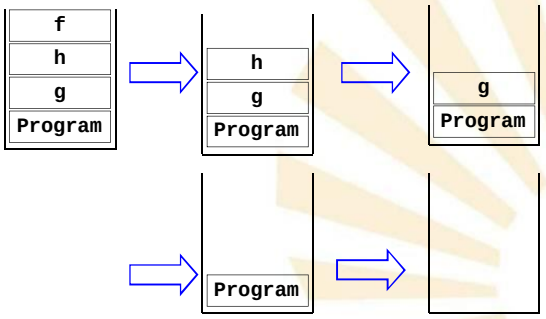

Program
f
g
h
g
Program
f
h
f
Activation Stack
Activation Tree
68


Program
f
g
g
Program
f
h
f
Activation Stack
Activation Tree
69


Program
f
g
f
h
f
Program
Activation Stack
Activation Tree
70


Program
f
g
f
h
f
Activation Stack
Activation Tree
71


Program
Program
g
Program
g
Program
h
g
Program
f
Program
f
g
Program
f
h
g
Program
Changes in the activation Stack (1 / 2)
72



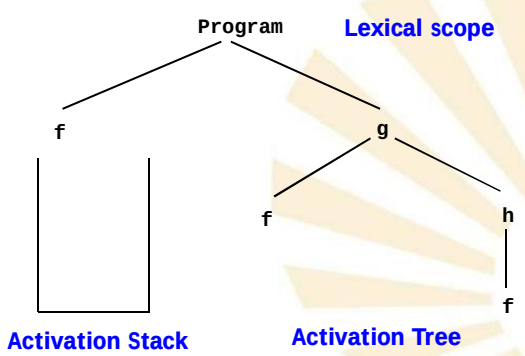
Changes in the activation Stack (2 / 2)

73

Run with lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

74

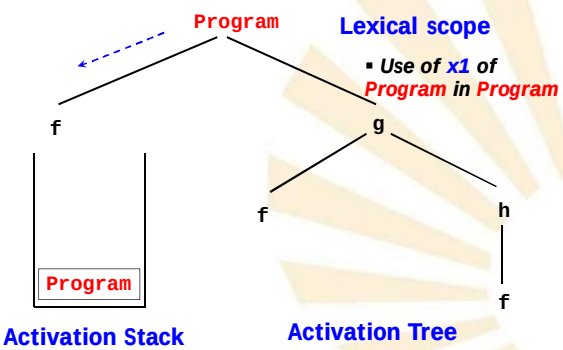


75

Run with lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x1
     Call to f
     Call to g
```

76



Lexical scope

- *Use of x1 of Program in Program*

77

Run with lexical scope

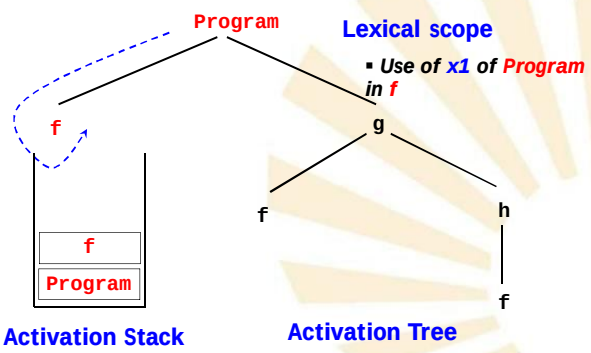
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

78

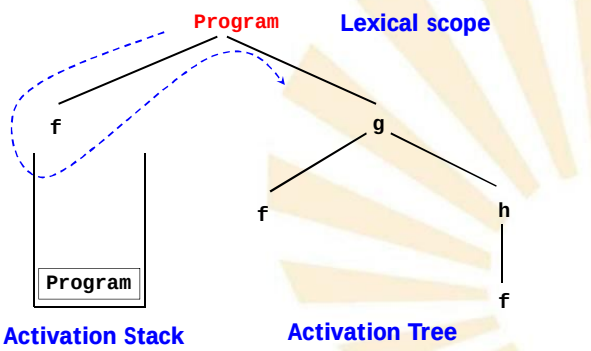
Run with lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x1

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

79



80

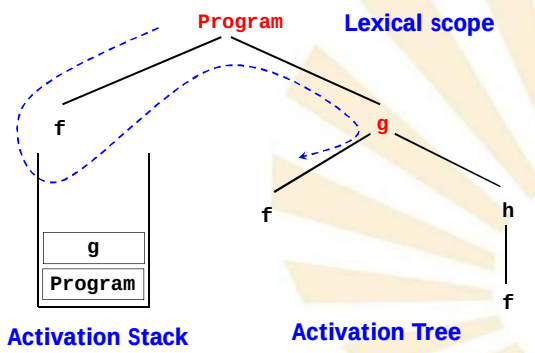


81

Run with lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

82



83

Run with lexical scope

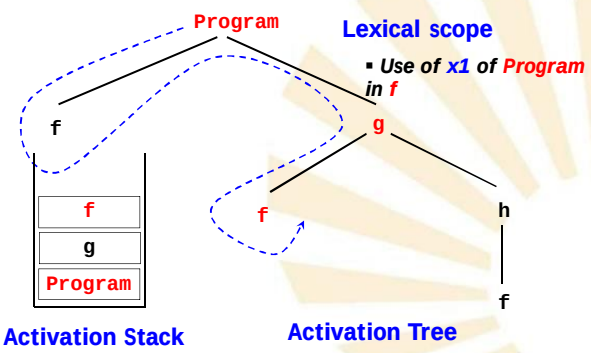
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

84

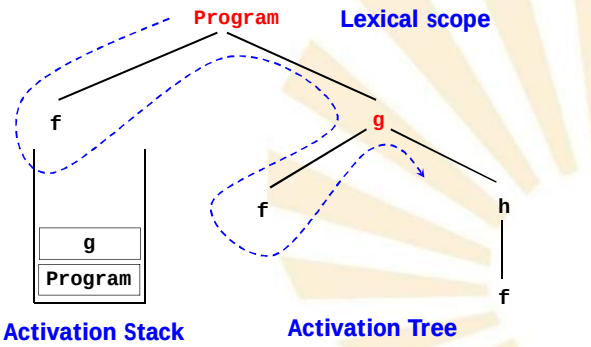
Run with lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x1

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

85



86



87

Run with lexical scope

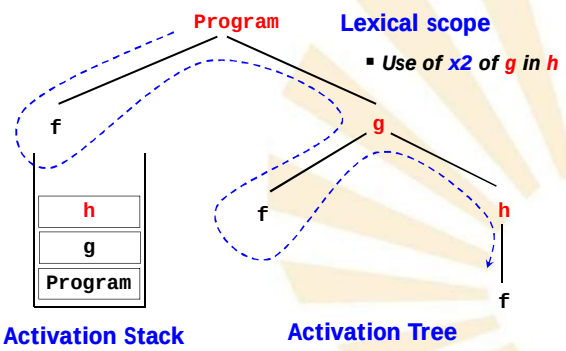
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

88

Run with lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x2
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

89



Lexical scope

- *Use of x2 of g in h*

90

Run with lexical scope

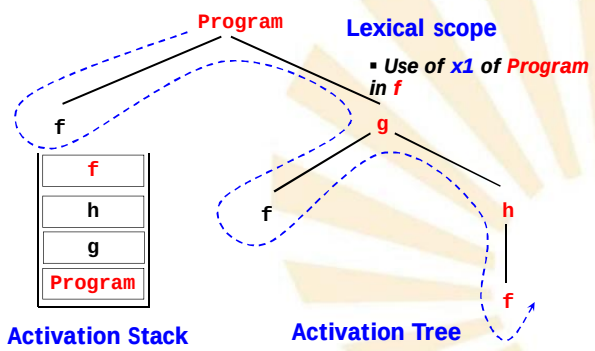
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

91

Run with lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x1

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

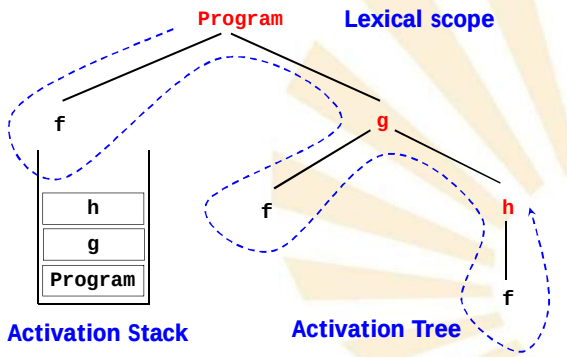
92



Lexical scope

- *Use of x1 of Program in f*

93

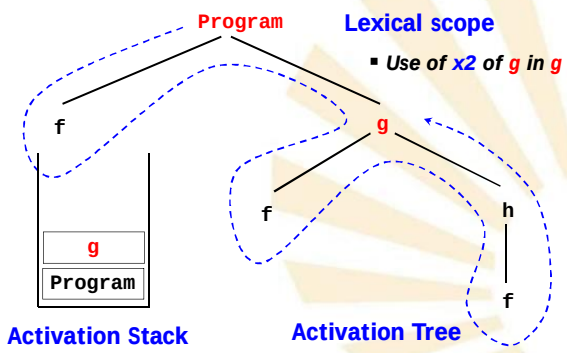


94

Run with lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x2

    Declaration of procedure g
      Declaration of variable x  (x2)

      Declaration of procedure h
       Use of x
       Call to f

       Call to f
       Call to h
       if condition = true then Call to g
       else   Use of x2

      Use of x
      Call to f
      Call to g
```

95

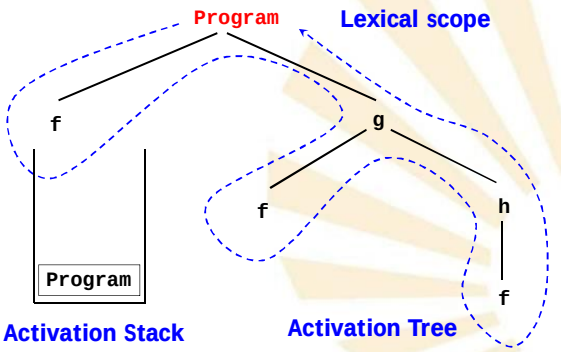


96

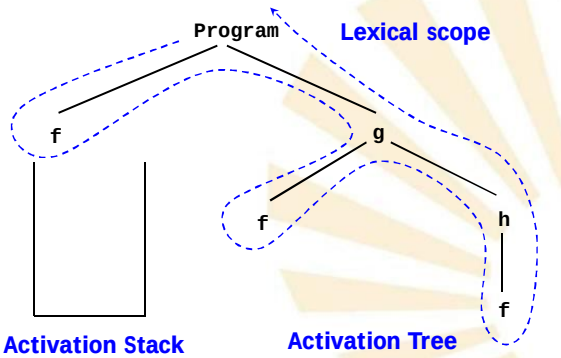
Run with lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x2

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x2

    Use of x
    Call to f
    Call to g
```

97



98



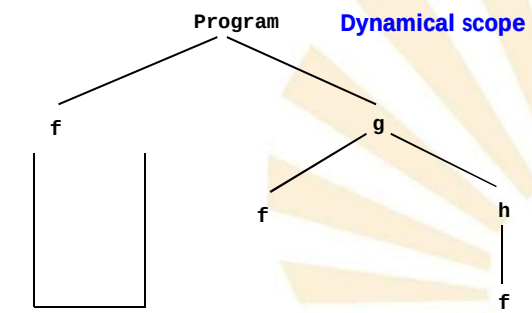
99

Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

100

Dynamical scope



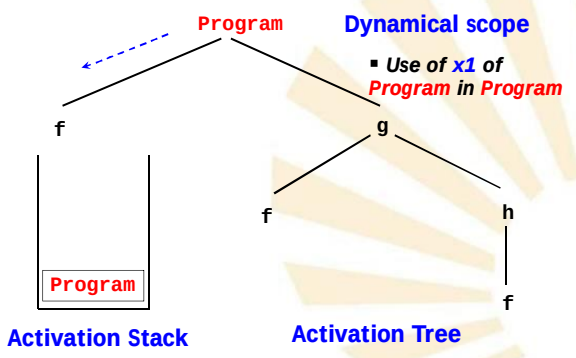
Activation Stack **Activation Tree**

101

Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x1
     Call to f
     Call to g
```

102



103

Run with dynamical scope

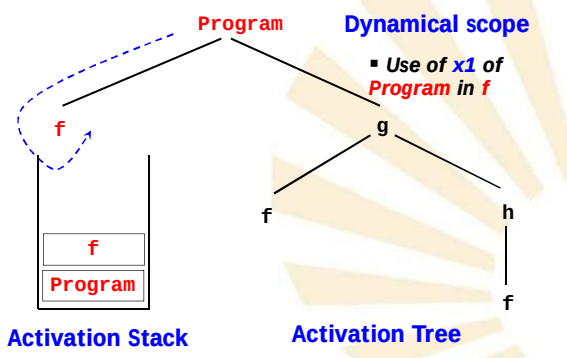
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

104

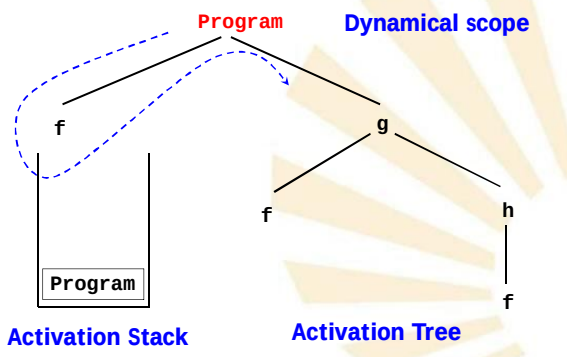
Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x1

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

105



106

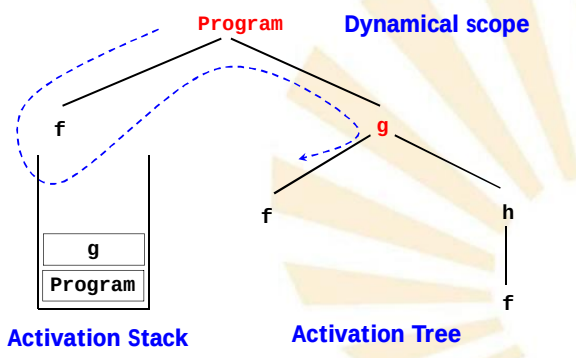


107

Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

      Declaration of procedure h
       Use of x
       Call to f

       Call to f
       Call to h
       if condition = true then Call to g
       else   Use of x

     Use of x
     Call to f
     Call to g
```

108



109

Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

110

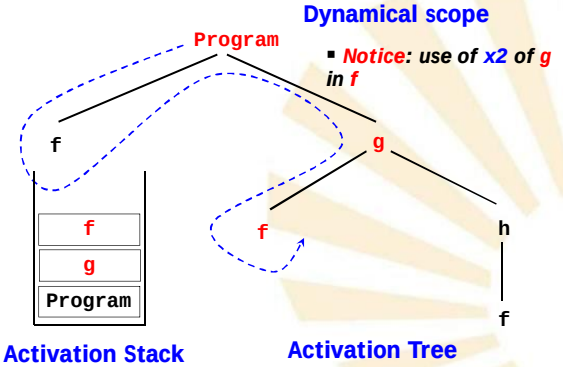
Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x2

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

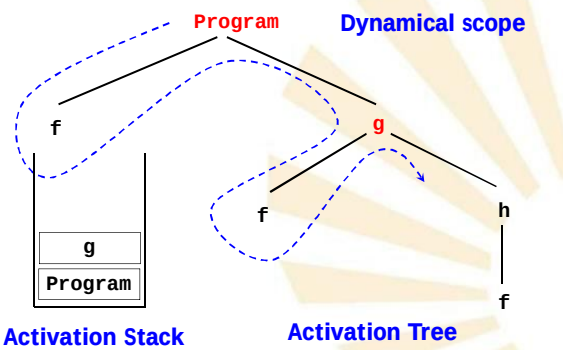
111

Dynamical scope

- *Notice: use of x2 of g in f*



112



113

Run with dynamical scope

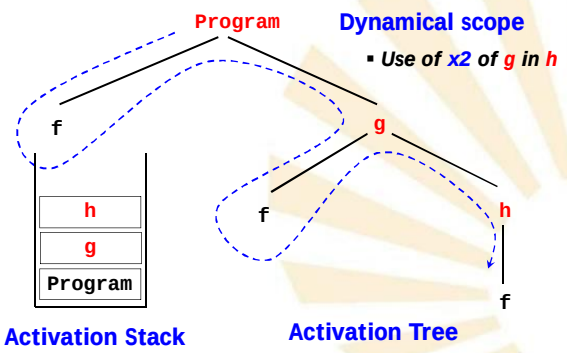
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

      Declaration of procedure h
       Use of x
       Call to f

       Call to f
       Call to h
       if condition = true then Call to g
       else   Use of x

      Use of x
      Call to f
      Call to g
```

114

Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x2
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

115



116

Run with dynamical scope

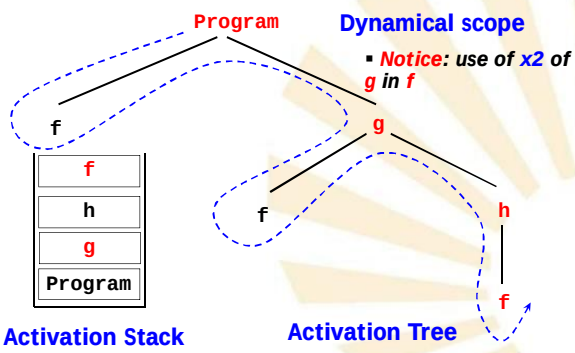
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

117

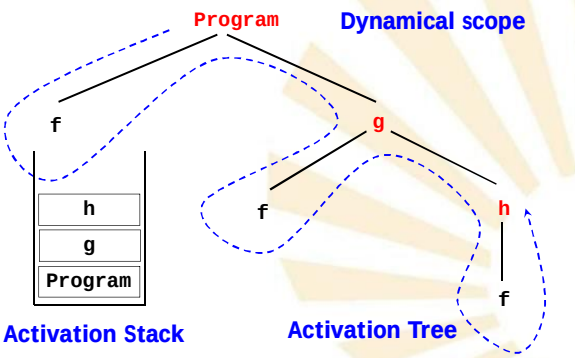
Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x2

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

118



119

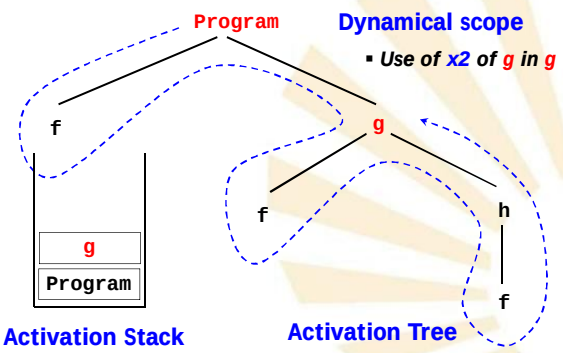


120

Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x2

     Use of x
     Call to f
     Call to g
```

121

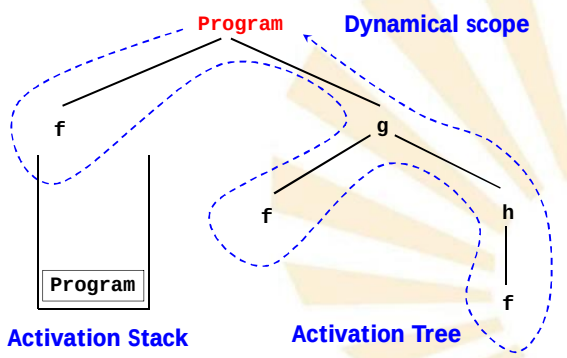


122

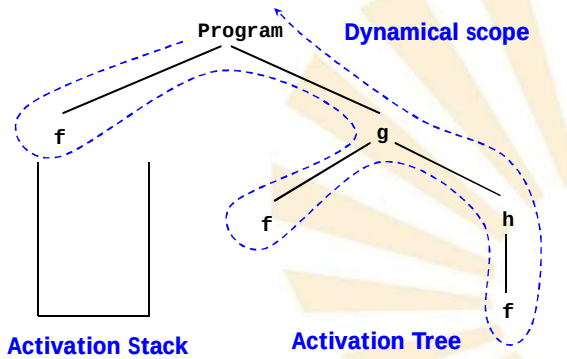
Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x2

     Use of x
     Call to f
     Call to g
```

123



124



125

2. **Historic Summary of Scheme**
   - ✓ LISP
   - ✓ Compilation versus Interpretation
   - ✓ Dynamically versus statically scope
   - ✓ **Origin of Scheme**

126

2. **Historic Summary of Scheme**

- ✓ **Origin of Scheme**:
  - Gerald Jay **Sussman** (MIT) and Guy Lewis **Steele** Jr.
  - **Question**:

    How would **LISP** be with **lexical** or **static scope** rules?
  - **Answer**: new language → **Scheme**
    - More **efficient** implementation of **recursion**
    - **First class functions**.
    - Rigorous **semantic** rules
  - **Influence** on Common LISP: lexical scope rules
  - ***Revised 5 Report on the Algorithmic Language Scheme***

127

2. **Historic Summary of Scheme**

- ✓ **Scheme**:
  - **Structure of scheme programs**
    - Sequence of
      - **definitions** of functions and variables
      - and **expressions**

128

CÓRDOBA UNIVERSITY

SUPERIOR POLYTECHNIC SCHOOL

DEPARTMENT OF
COMPUTER SCIENCE AND NUMERICAL ANALYSIS

DECLARATIVE PROGRAMMING

COMPUTER ENGINEERING

COMPUTATION ESPECIALITY

FOURTH YEAR

FIRST FOUR-MONTH PERIOD

Subject 1.- Introduction to Scheme language