INFORMATION and REMINDERS

• **Midterm - PLEASE see our class webpage**

  1. Regular exam: Wednesday, March 8, 9:40 - 11:00am Please check your section assignment to either LCH AUD or BE AUD.

  2. Make-up 1: Wednesday, March 8, 8:40 - 10:00am, TIL-232

  3. Make-up 2: Wednesday, March 8, 8:00 - 9:20am, TIL-226

• Make-up exams: you should received an email from me if you have a conflict

• Don’t forget to bring your **Rutgers ID**. If we don’t know who you are, you cannot take the exam.
Lexical Scoping Example

scope of a declaration: Portion of program to which the declaration applies

Program
x, y: integer // declarations of x and y
begin
Procedure B // declaration of B
  y, z: real // declaration of y and z
  begin
    ...
    y = x + z // occurrences of y, x, and z
    if (...) call B // occurrence of B
  end
Procedure C // declaration of C
  x: real // declaration of x
  begin
    ...
    call B // occurrence of B
  end

... call C // occurrence of C
    call B // occurrence of B
end
Lexical Scoping Example

Calling chain: MAIN ⇒ C ⇒ B ⇒ B
Scoping and the Run-time Stack

Access links and control links may be used to look for non-local variable references.

Static Scope:

Access link points to stack frame of the most recently activated lexically enclosing procedure

⇒ Non-local name binding is determined at compile time, and implemented at run-time

Dynamic Scope:

Control link points to stack frame of caller

⇒ Non-local name binding is determined and implemented at run-time
Lexical scoping (de Bruijn notation)

Symbol table matches declarations and occurrences.
⇒ Each variable name can be represented as a pair (nesting_level, local_index).

Program
(1,1), (1,2): integer // declarations of x and y
begin

Procedure B // declaration of B
(2,1), (2,2): real // declaration of y and z
begin

... // occurrences of y, x, and z

(2,1) = (1,1) + (2,2)

if (...) call B // occurrence of B
end

Procedure C // declaration of C
(2,1): real // declaration of x
begin

... call B // occurrence of B
end

...

call C // occurrence of C
call B // occurrence of B
end
Access to non-local data

How does the code find non-local data at run-time?

Real globals

- visible everywhere
- translated into an address at compile time

Lexical scoping

- view variables as \((level, offset)\) pairs
  (compile-time symbol table)
- look-up of \((level, offset)\) pair uses chains of access links (at run-time)
- optimization to reduce access cost: display

Dynamic scoping

- variable names must be preserved
- look-up of variable name uses chains of control links (at run-time)
- optimization to reduce access cost: reference table
Access to non-local data (lexical scoping)

What code (ILOC) do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

What do we know?

1. The nesting level of the statement is level 2.
2. Register \(r_0\) contains the current FP (frame pointer).
3. \((2,1)\) and \((2,2)\) are local variables, so they are allocated in the activation record that current FP points to; \((1,1)\) is a non-local variable.
4. A new instruction:

\[
\text{load } R_x \Rightarrow R_y \quad \text{means } R_y \leftarrow MEM(R_x)
\]

Compiler-generated code for the statements in a procedure must work for all possible, valid runtime stacks/environments
Access to non-local data (lexical scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

\[
\begin{array}{ll}
(1,1) & | \text{loadI -4 } \Rightarrow r1 \quad \text{// offset of access link} \\
& | \quad \text{// in frame (bytes)} \\
& | \text{add r0, r1 } \Rightarrow r2 \quad \text{// address of access link in frame} \\
& | \text{load r2 } \Rightarrow r3 \quad \text{// get access link; r3 now} \\
& | \quad \text{// contains ‘‘one-level-up’’ FP} \\
& | \text{loadAI r3, 4 } \Rightarrow r4 \quad \text{// get content of first local variable} \\
& | \quad \text{// in ‘‘one-level-up’’ frame (bytes)} \\

(2,2) & | \text{loadAI r0, 8 } \Rightarrow r5 \quad \text{// content of second local variable} \\
& | \quad \text{// current frame (bytes)} \\
& + | \text{Add r4, r5 } \Rightarrow r6 \quad \text{// (1,1) + (2,2)} \\

(2,1) & | \text{storeAI r6 } \Rightarrow r0, 4 \quad \text{// store value into first local variable} \\
& = | \quad \text{// in current frame (bytes)}
\end{array}
\]
Access to non-local data (lexical scoping)

Two important problems arise

1. *How do we map a name into a (level, offset) pair?*

   We use a block structured symbol table
   (compile-time)
   
   • when we look up a name, we want to get the
     most recent declaration for the name
   • the declaration may be found in the current
     procedure or in any nested procedure

2. *Given a (level, offset) pair, what’s the address?*

   Two classic approaches
   (run-time)
   
   ⇒ access links
   ⇒ displays
Managing non-local data (lexical scoping)

To find the value specified by \((l, o)\)

- need current procedure level, \(k\)
- if \(k = l\), is a local value
- if \(k > l\), must find \(l\)'s activation record
  \(\Rightarrow\) follow \(k - l\) access links
- \(k < l\) cannot occur

Maintaining access links:

If procedure \(p\) is nested immediately within procedure \(q\), the access link for \(p\) points to the activation record of the most recent activation of \(q\).

- calling level \(k + 1\) procedure
  1. pass my FP as access link
  2. my backward chain will work for lower levels
- calling procedure at level \(l \leq k\)
  1. find my link to level \(l - 1\) and pass it
  2. its access link will work for lower levels
The display

To improve run-time access costs, use a display.

- table of access links for lower levels
- lookup is index from known offset
- takes slight amount of time at call
- a single display or one per frame

Access with the display

*assume a value described by \((l, o)\)*

- find slot as \(DP[l]\) in display pointer array
- add offset to pointer from slot

“setting up the activation frame” now includes display manipulation.
Display management

Single global display:  

*simple method*

on entry to a procedure at level $l$

- save the level $l$ display value
- push FP into level $l$ display slot

on return

- restore the level $l$ display value
Next Lectures Roadmap

• Parameter passing styles: Read Scott Chapter 8.3
• Introduction to functional languages; read Scott Chapter 10
• Lambda calculus