Homework 4 extension until Saturday, March 4, at 11:59pm.

Sample solutions for HW2 and HW3 have been posted. Sample solution for HW4 will come out on Sunday, March 4.

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. IF NOT, please let me know no later than Sunday, March 5.
Names, Bindings, and Memory

Scott: Chap. 3.1 - 3.4; ALSU Chap. 7.1 - 7.3

**Binding** – association of a name with an attribute (e.g., a name and a memory location, a function name and its “meaning”, a name and a value)

- **Compile time** – during compilation process – static (e.g.: macro expansion, type definitions)
- **Link time** – separately compiled modules/files are joined together by the linker (e.g., adding the standard library routines for I/O (stdio.h), external variables)
- **Run time** – when program executes – dynamic

Compiler needs bindings to know meaning of names during translation and execution.
Binding Times - Choices

- **Early binding** times – more efficient (faster) than at runtime

- **Late binding** times – more flexible (postpone binding decision until more “information” is available)

- Examples of static binding (early):
  - functions in C
  - types in C

- Examples of dynamic binding (late):
  - method calls in Java or virtual function calls in C++
  - actual types of objects pointed to by a Java reference variable (class hierarchy)
  - dynamic typing in Scheme

Note: **dynamic linking** is somewhat inbetween static and dynamic binding; the function signature has to be known (static), but the implementation is linked and loaded at run time (dynamic).
How to Maintain Bindings

• **symbol table**: maintained by compiler during compilation
  names $\Rightarrow$ attributes

• **environment**: maintained by compiler generated code during program execution
  names $\Rightarrow$ memory locations

• **memory**: maps memory locations to values
  memory locations $\Rightarrow$ values

Questions

• How long do bindings for a name hold in a program?
• What initiates a binding?
• What ends a binding?
Scope Example

Block Structures Programming Languages

program L;
  var n: char; {n declared in L}

  procedure W;
  begin
    write(n); {n referenced in W}
  end;

  procedure D;
    var n: char; {n declared in D}
  begin
    n:= 'D'; {n referenced in D}
    W
  end;

begin
  n:= 'L'; {n referenced in L}
  W;
  D
end.

CS 314
Lexical Scope

- Non-local variables are associated with declarations at compile time
- Find the smallest block syntactically enclosing the reference and containing a declaration of the variable

- Example:
  - The reference to \( n \) in \( W \) is associated with the declaration of \( n \) in \( L \)
  - The output is?
Dynamic Scope

- Non-local variables are associated with declarations at run time
- Find the most recent, currently active run-time stack frame containing a declaration of the variable

Example:
- The reference to \( n \) in \( W \) is associated with two different declarations at two different times
- The output is?
Stack Frame, Activation Record

- Run-time stack contains frames for main program and each active procedure.

- Each stack frame includes:
  1. Pointer to stack frame of caller (control link for stack maintainance and dynamic scoping)
  2. Return address (within calling procedure)
  3. Mechanism to find non-local variables (access link for lexical scoping)
  4. Storage for parameters, local variables, and final values
Context of Procedures

Two contexts:

- static placement in source code (same for each invocation)
- dynamic run-time stack context (different for each invocation)

Scope Rules

Each variable reference must be associated with a single declaration (ie, an offset within a stack frame).

Two choices:

1. Use static and dynamic context: lexical scope
2. Use dynamic context: dynamic scope

• Easy for variables declared locally, and same for lexical and dynamic scoping
• Harder for variables not declared locally, and not same for lexical and dynamic scoping
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: \( \text{MAIN} \Rightarrow C \Rightarrow B \Rightarrow B \)
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
Next Lectures Roadmap

- More on static / lexical scoping
- Parameter passing styles: Read Scott Chapter 8.3
- Introduction to functional languages; read Scott Chapter 10
- Lambda calculus