Class Information

- Midterm exam: Friday, March 14, in class, closed book, closed notes.
- Fifth homework will be posted by tomorrow. Last homework before the exam.
- Will start posting homework sample solutions over the weekend.

C Programming Project

Parse table construction.

Review: How to Maintain Bindings

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)

- symbol table: maintained by compiler during compilation
  names ⇒ attributes
- environment: maintained by compiler generated code during program execution
  names ⇒ memory locations
- memory: maps memory locations to values
  memory locations ⇒ values

Questions

- What initiates a binding?
- What ends a binding?
- How long do bindings for a name hold in a program?

Block structured programming languages

Binding: variable **x** to memory locations

```
program main;
   var x: int
   procedure foo;
       procedure bar;
           var x: int;
           begin ...
              if (...) foo() else bar();
           end;
       procedure blah;
           var z, y, x: float;
           begin ...
              if (...) bar() else foo();
              x = x + 1; (*)
           end
       begin ...
          if (...) bar() else blah();
          x = x + 1; (**)
       end;
begin
  foo();
end.
```

How to generate code for statements (*) and (**)?

How much do we know about the binding at compile time?

Review: Lexical / Dynamic Scope

lexical

- 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

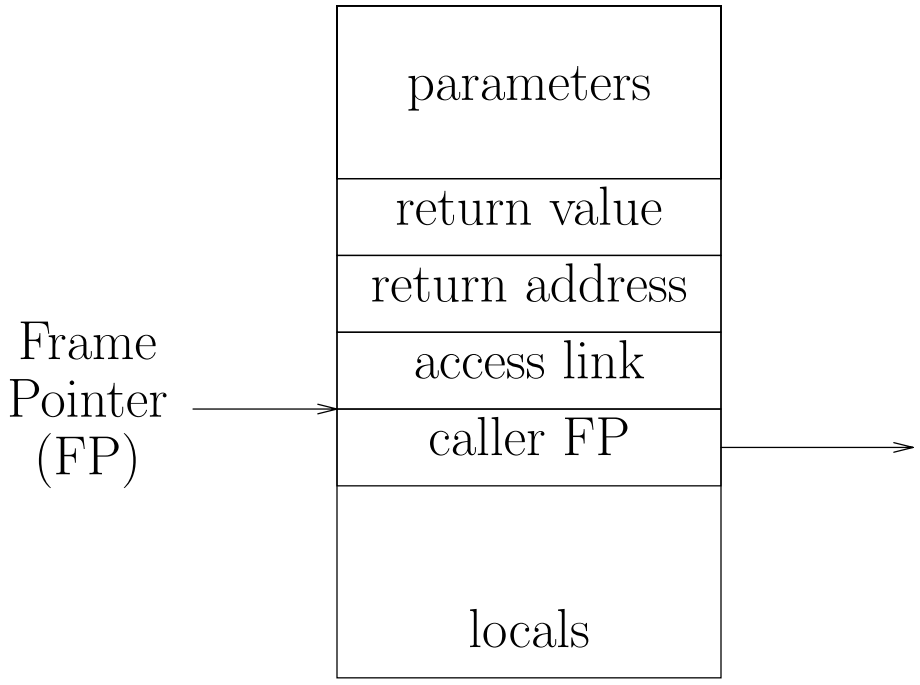
dynamic

- 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

Stack Frame, Activation Record

Scott: Chap. 8.1 - 8.2; ALSU Chap. 7.1 - 7.3

- 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

Next Lecture

Things to do:

Continue working on the project. Due Friday March 7!

Read Scott: Chap. 8.3 ;

Next time:

- More on dynamic runtime environments
- How to use access links and displays
- Parameter passing styles and their implementation.