

Announcements

- **Start project 1 if you haven't yet**
- **Midterm is Friday, March 5th, 2:50-4:10**
 - last chance to let us know you can't make that time is 1 week before: Friday, Feb 27th

1

Semantics - Memory

- **Identifiers - attributes, binding times**
- **Declaration scope**
 - Lexical
 - Dynamic
- **Symbol table**
- **Kinds of storage**
 - Stack
 - Heap - lifetimes, environment

2

Names and Bindings

- ***Name***
 - Denotes a PL construct (e.g., procedure)
 - Has associated *attributes* (e.g., type, variable id, memory location)
- ***Binding*** - process of associating a name with an attribute
 - Can happen at different times during translation and execution

3

Binding Times

- ***Static***
 - *Compile time* - often layout of data in memory is chosen at this time
 - *Link time* - separately compiled modules of a program are joined together by linker (e.g., adding in standard library routines for I/O)
 - *Load time* - time when program is actually loaded into memory to run.
- ***Dynamic***
 - *Run time* - when program executes
- **Compiler needs bindings to know meaning of names during translation and execution**

4

Binding Times - Choices

- **Earlier binding times -- more efficient**
- **Later binding times -- more flexibility in PL**
- **Examples of static binding:**
 - **Function signature types in C**
 - **Declared types in Pascal, Algol, Java, C++**
- **Examples of dynamic binding:**
 - **Methods called in Java or virtual calls in C++**
 - **Actual types of objects pointed to by a Java reference variable**
 - **Types of Prolog or Scheme variables**

5

Bindings as Compiler Maps

- **Symbol table: names \Rightarrow attributes**
- **Environment: names \Rightarrow locations**
- **Memory: locations \Rightarrow values**
- **Q: How long do these bindings hold in a program?**
 - What initiates a binding?**
 - What terminates a binding?**
 - answers are PL definition choices.**
 - *Scope of a binding:* region of program over which binding is maintained

6

Lexical Scoping

- **Block structured PLs**
 - Allow for local variable declaration
 - Inherit global variables from enclosing blocks
 - Local declarations take precedence over inherited ones
 - Hole in scope
 - Lookup for non-local variables proceeds from inner to enclosing blocks in inner to outer order.
 - Used in Algol, Pascal, Scheme (with *let*), C++, C, Java
 - C is a flat language for procedure declarations, disallows nested procedures
 - Declaration grouping mechanisms include Java packages, C *extern*'s, C *static*'s

7

Example - Block Structured PL

```
main
  a, b, c: integer;   main.a, main.b, main.c  main.p(), main.r()
  procedure p
    c: integer;      main.a, main.b, p.c  main.p(), p.s(), main.r()
    procedure s
      c, d: integer;  main.a, main.b, s.c, s.d  main.p(), s.r(), p.s()
      procedure r
        ...          main.a, main.b, s.c, s.d, main.p(), s.r(), p.s()
        end r;
      r;
    end s;
    r;
    s;
  end p;
  procedure r
    a: integer;      r.a, main.b, main.c  main.r(), main.p()
    ...
  end r;
  ...; p; ...
end main
```

nested block structure allows locally defined variables and functions

8

Example - Scheme

```
((lambda (x)
  (lambda (y)
    (lambda (z)
      (+ x y)
      5)
    4)
  3) evaluates to 12
```

```
(let ((x 3))
  (let ((y 4))
    (let ((z 5))
      (+ x y))))
also evaluates to 12
```

**(LET ((x 2)) (+ x ...)) is just an abbreviation for
((LAMBDA (x) (+ x ...)) 2)**

9

Symbol Table

- **Must support insertion, deletion, lookup of names**
- **For lexical scoping, need to use a stack for storing attributes of a variable (to handle hole-in-scope)**
- **Need to update as enter and leave a block at compile time (during translation)**

10

Example

