



Names, Bindings, and Scopes

Charles R. Hardnett

Fall/Spring 2003
Copyright © 2003 Spelman College

Department of Computer and
Information Sciences (CIS)

CIS346



Lecture Topics

- Binding
 - Type Binding
 - Storage Binding
- Scope
 - Static Scope
 - Dynamic Scope

Assignments:

1. Read Chapter 5
2. Homework Problems (see website)

Fall/Spring 2003
Copyright © 2003 Spelman College

Department of Computer and
Information Sciences (CIS)

CIS346
(Lecture #5 Slide #2)



Binding

- What is binding?
 - The association between an attribute and an entity or between an operation and a symbol
- What is binding time?
 - The time at which the binding takes place
 - Static binding => Compile-Time
 - Remains unchanged during execution
 - Dynamic binding => Run-Time
- Binding example: Binding identifiers to memory
 - Global variables are statically bound
 - Dynamically allocated variables are dynamically bound
 - New operation



Full Binding Example

```
int Count;
```

```
...
```

```
Count = Count + 1;
```

- **Count** is bound to **int** at compile-time
- **Count** is bound to **a value** at run-time
- The **+** symbol is bound to **integer addition** at compile-time
- The **memory location** is bound to the **identifier** at compile-time or run-time

Language Design time bindings

- The possible types for an identifier
- The possible values of a type
- The size of the type
- The internal representation of the literal 1

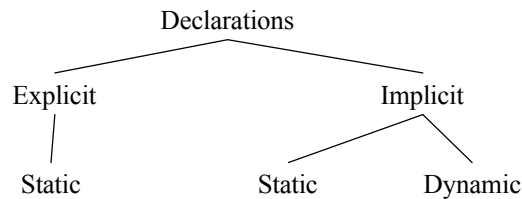


Type Bindings

- Explicit variable declaration
 - Identifier names are listed with legal type name
 - Most contemporary languages
 - C, C++, Ada, Pascal, Java, etc.
 - Statically bound
- Implicit variable declaration
 - The context of a reference to the identifier determines the type
 - Perl, ML, Basic, PL/I, APL, FORTRAN
 - Statically or Dynamically bound
- In Perl (static)
 - Identifiers are `<symbol><name>`
 - `<symbol>` is @, %, or \$
 - \$List is a scalar
 - %List is an array
 - @List is a map
- In FORTRAN (static)
 - If undeclared the identifier name begins with I, J, K, L, M, or N then its an INTEGER type
 - Otherwise, it's a REAL
- In APL (dynamic)
 - If A is integer and C is float, and
 - A = C
 - Now A is float



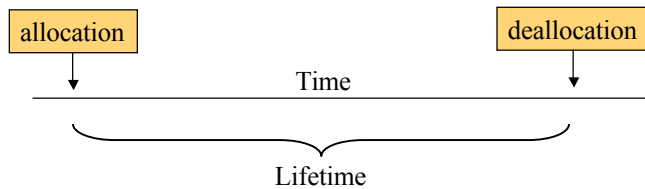
Evaluation: Explicit vs Implicit Declarations



- Readability can be hindered by implicit declarations
- Writability can be improved by implicit declarations
- Reliability can be hindered by implicit declarations
 - New identifiers introduced unknowingly
- Reliability can be hindered by dynamic declarations



Storage Bindings



- Lifetime is the time during which an identifier is bound to a memory location
 - **Static storage binding (see more later)**
 - All global variables and other explicitly declared variables
 - Allocation during compile-time
 - **Stack-dynamic storage binding (see more later)**
 - Local variables and parameters of subroutines
 - Allocation at invocation time and Deallocation at end of routine
 - **Heap-dynamic storage binding**
 - Explicit
 - Implicit



Heap-Dynamic Bindings

- Explicit bindings
 - **Explicit allocation operation**
`new (C++), malloc (C)`
 - **Explicit deallocation operation**
`delete (C++), free (C)`
- Implicit bindings
 - As the data is added and removed from the data structure, allocation and deallocation happens in the background
 - **Example: Perl**

```
my $myarray = ("one", 1, "two", 2, "three", 3);  
my $myarray{"four"} = 4;
```
 - **Implicit deallocation is called "Garbage Collection" (more later)**
 - A the correct point in time a background process deallocates unused storage
 - Java, Scheme, Lisp, APL



Evaluation: Storage Binding Strategies

- Static bindings allow compilers to analyze storage needs and increase reliability
- Statically bound variables are accessed more efficiently
 - Dynamically bound variables are indirectly accessed
- Dynamically bound variables enable flexible data structures
- Dynamic storage bindings use memory more efficiently
 - Static arrays vs Linked Lists
- Reliability is adversely affected by dynamic binding
 - Programming errors with pointers lead to core dumps!
 - Increasing reliability requires additional overhead



Type Inference

- Type inference is implicit type binding where the type is inferred from the context of the expression
- Perl is a language with exclusive use of type inference
 - Types are inferred by the operator usage
 - `($name ne $num) ## $name and $num are inferred to be string`
 - `($val < $max) ## $val and $max are inferred to be numeric`
 - Types are inferred in assignment
 - `$name = "Erin"; ## type of $name is inferred to be string`
 - `$name = 1024; ## type of $name is inferred to be integer`
 - `$name = 55.4; ## type of $name is inferred to be real`



Scope

- The scope of a variable is the range of statements over which it is visible
 - The non-local variables of a program unit are those that are visible but not declared there
 - The scope rules of a language determine how references to names are associated with variables
- Static Scope
 - Based on program text, which is known at compile-time
 - To connect a name reference to a variable, you (or the compiler) must find the declaration
- Dynamic Scope
 - Based on calling sequences of program units during execution
- Static Scope vs Dynamic Scope
 - Classic dual of temporal(time dependent) versus spatial(layout)



Static Scope

- Name Search Algorithm
 - Goal: search declarations for a given identifier name
 - 1. Search locally
 - 2. If name is found stop searching, this is the scope for that name
 - 3. Otherwise, move to the next larger enclosing scope and goto step 1.
- A. Where is a visible?
B. Where is c visible?
C. Where is b visible?
D. Where is z visible?
E. Is stmt1 legal?
F. Is stmt2 legal?

```
int foo() {  
    int z;  
    for.. { //loop A  
        int a;  
        for.. { // loop B  
            int c;  
        }  
        if.. { // if A  
            int b;  
            b = c; // stmt 2  
        }  
        a = z; // stmt 1  
    }  
}
```



Static Scoping: Function Blocks

- C defines most functions at the global scope
 - **main(), foo(), and bar() definitions can be global**
- C++ defines most functions at the global scope and class scope
 - **Functions at the class scope are member functions**
 - **Methods are private (class scope) and public (global scope)**
- Pascal and Ada allow functions at the local scope
 - **Functions are defined local to other functions**



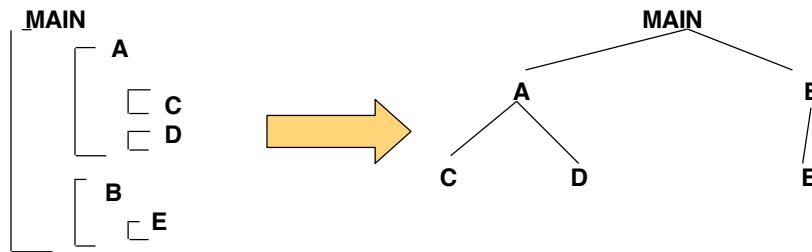
Defining subroutines locally?

- The example shows local definitions of subroutines
 - **The scope of these definitions is within the enclosing subroutine**
 - **function cube() and procedure found() are only visible in procedure wrapper().**
- Which variable is being accessed in each stmt?
- Which y is being accessed by stmt4, cube's y or wrapper's y?

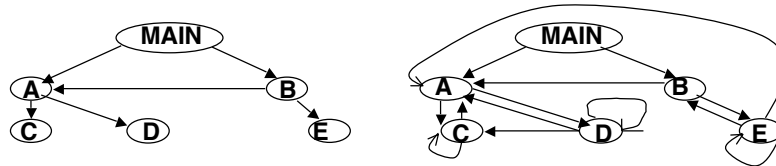
```
procedure wrapper(...);  
  var a,b,c:integer;  
  var x, y: real;  
  function cube(...): real;  
  var y,z:integer;  
  begin  
    writeln(x); // stmt3  
    writeln(y); // stmt4  
  end;  
  procedure found(...);  
  begin  
    writeln(a); // stmt5  
    writeln(b); // stmt6  
    writeln(y); // stmt7  
  end;  
begin  
  writeln(b); // stmt1  
  writeln(y); // stmt2  
end;
```



More Static Scope Examples



Some Legal Subroutine Invocation Possibilities



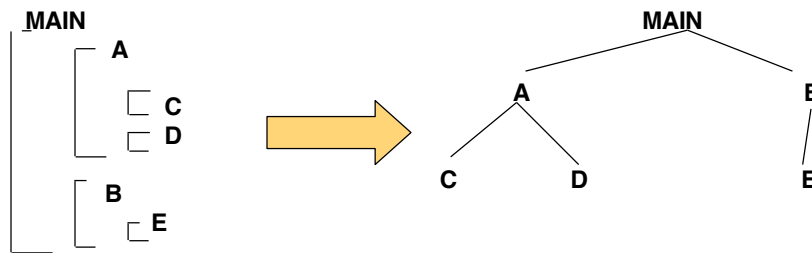
Fall/Spring 2003
Copyright © 2003 Spelman College

Department of Computer and
Information Sciences (CIS)

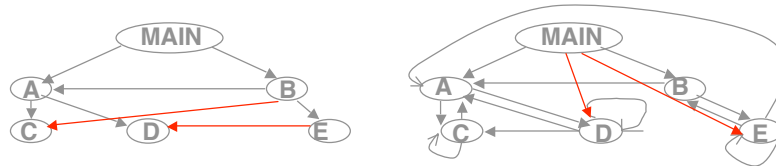
CIS346
(Lecture #5 Slide #15)



More Static Scope Examples



Some **Illegal** Subroutine Invocation Possibilities



Fall/Spring 2003
Copyright © 2003 Spelman College

Department of Computer and
Information Sciences (CIS)

CIS346
(Lecture #5 Slide #16)



A Static Scope Problem

- Problem Description
 - Suppose the spec is changed so that D must now access some data in B
- Possible Static Solutions
 1. Put D in B
(but then C can no longer call it and D cannot access A's variables)
 2. Move the data from B that D needs to MAIN (Globals)
(but then all procedures can access them)
 3. **BONUS QUESTION: What is the solution that C++ takes to solve this problem, and how is it safer than globals? (10 HW Points)**
- Conclusion
 - Static scoping often encourages many globals

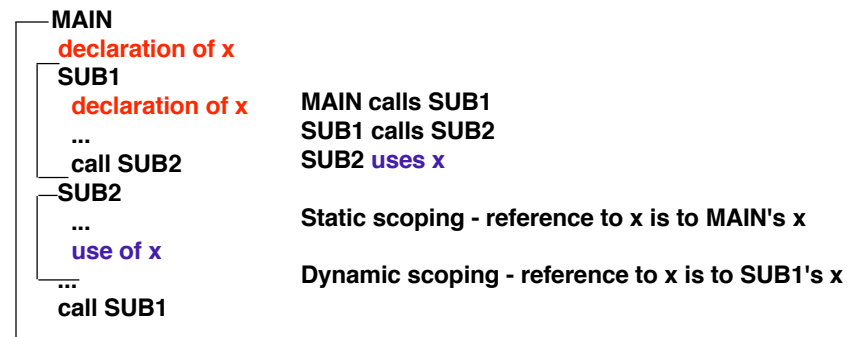


Dynamic Scope

- Can be used to solve the previous static scope problem
- Based on calling sequences of program units, not their textual layout
- References to variables are connected to declarations by searching back through the current chain of subroutine calls to this point
- Dynamic scope is found in Perl, APL, and early LISP



Dynamic Scope Example



- Evaluation of Dynamic Scoping

Advantages

convenience
Enables flexible access to data
(fixes static scope problem)

Disadvantage

poor readability