CS 461

Programming Language Concepts

Gang Tan
Computer Science and Engineering
Penn State University

1

Ch3 Names, Scopes, and Bindings

*Some slides adapted from the ones by Michael Scott

2

2

Naming plays a fundamental role in PLs

- Use names for
  - variables
  - functions
  - types
  - Modules (packages)

3

3

Syntactic Issues for Naming

- Lexical rules for names
  - most languages: a letter followed by a series of letters or digits
  - some languages allow special characters
    - Cobol: allow the hyphen character
    - C-like language: allow the underscore character
  - some early languages has length restrictions
    - Fortran 77: 6 chars

4

Syntactic Issues for Naming

- Collection of reserved words or keywords.
  - Cannot be used as identifiers (e.g., if, while, do, …)
  - Predefined identifiers: e.g., library routines
- Case sensitivity
  - C-like languages: yes
  - Early languages (Pascal, Ada): no

5

Variable Names

Language	Name limits	Connectors	Case Sensitivity	Notes
Fortran 77	6 chars.	none	No	only letters and digits
COBOL	30 chars.	hyphen	No	
Ada	no limit	underscore	No	
C89	none, 31 chars. significant	underscore	Yes	
C99	none, 63 chars significant	underscore	Yes	
C++	implementation specific[1]	underscore	Yes	
Java	no limit	underscore	Yes	also allows Unicode currency symbols

[1]C++ has no limit on name length; the number of significant characters is implementation specific

6

Variable Naming Convention

- Hungarian notation
  - Each variable name begins with one or more lowercase characters identifying the data type

Prefix	Data Type	Examples
b	Bool	// bCondition
c	Char	
l	LONG	
n	int	// nCount
p	pointer	// pNextNode
w	WORD	

7

7

Binding

- Binding is an association between a program entity (such as a variable) and a property (such as its value, scope, type, …)
  - The scope of a binding is the part of the program in which the binding is active
- A binding is **static** if the association occurs at compile time.
- A binding is **dynamic** if the association occurs at run-time.
  - AKA late binding

8

Static vs. Dynamic Binding

- In general, early binding times are associated with greater efficiency
  - Compiled languages tend to have early binding times
  - E.g., static type checking
- Later binding times are associated with greater flexibility
  - Interpreted languages tend to have later binding times
  - E.g., dynamic type checking

9

Variables' Bindings

- Storage location (e.g., memory address)
- Value
- Type
- Scope
- Lifetime

10

L-values and R-values

- In C-like languages, "x = x + 1"
  - The same x refers to different bindings depending on whether it appears on the left of or the right of the assignment
- L-value - use of a variable name to denote its storage location.
  - Ex: x = …
- R-value - use of a variable name to denotes its value.
  - Ex: … = … x …
- Some languages support/require explicit dereferencing (e.g., ML)
  - Ex: x := !x + 1

11

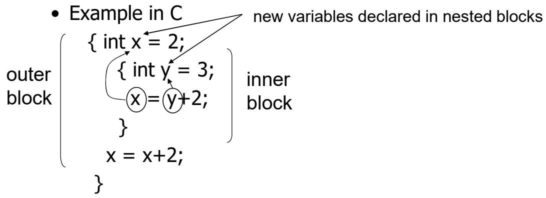
Scope

12

12

Block-Structured Languages

- Nested blocks, local variables
  - Example in C



  - Storage management
    - Enter block: allocate space for variables
    - Exits block: some or all space may be deallocated

13

Examples

- Blocks in common languages
  - C/C++/Java { … }
  - Algol begin … end
  - ML let … in … end
    - let x = 3 in let y =3 in x + y

14

Forms of Scope

- Inlined blocks
- Scope associated with functions or procedures
- A for-loop in Java/C++ can introduce a scope

15

Java/C++ for-loop: can introduce a scope

```
for (int i = 0; i < 10; i++) {
   System.out.println(i);
   ...
}

... i ... // invalid reference to i
```

- Not for C though

16

Scoping in typical languages

	Algol	C	Java	Ada
Block	nested	nested	nested	nested
For Loop	no	no	yes	yes
Function	nested	yes	yes	nested
Class	n/a	n/a	nested	yes
Package	n/a	n/a	yes	yes

17

17

Scope Vs. Lifetime

- Scope
  - Region of program text where a variable is visible
- Lifetime
  - Period of time when the storage for the variable is allocated

```
{ int x = … ;
   {  int y = … ;
      {  int x = … ;
         ….
      };
   };
};
```

- Nested scopes
- Inner declaration of x hides outer one.
  - outer one not visible
  - Note: Java does not support redeclaration of variables
- Called "hole in scope"
- Lifetime of outer x includes time when inner block is executed
- Lifetime ≠ scope
- Lines indicate "contour model" of scope.

18

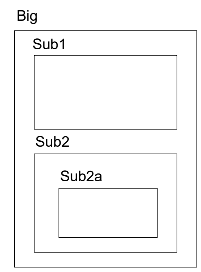
Static Scoping

- In static scoping, a name is visible to a collection of statements according to its lexical position in the source program.
- Most modern languages use static scoping
  - Java, C, Scheme, Ada

19

Example of static scoping (Ada)

```
procedure Big is
  X : Integer;
  procedure Sub1 is
  begin -- of Sub1
  ... X ...
  end; -- of Sub1
  procedure Sub2 is
    X : Integer;
    procedure Sub2a is
    begin -- of Sub2a
      ... X ...
    end -- of Sub2a
  begin -- of Sub2
  ... X ...; ... Sub2a ...
  end; -- of Sub2
begin -- of Big
... X ...
end; -- of Big
```



20

20

Implementing the scope: Symbol Tables

- A *symbol table* is a data structure kept by a translator that allows it to track declared names and their bindings.
- Assume for now that each name is unique within its local scope.
- The data structure is usually a stack of dictionaries
  - Which are maps from keys to values; keys: names; values: bindings for names

21

Pseudo-algorithm for scoping

1. For each scope, build a dictionary, which records name-binding pairs for all names declared in the scope
2. Build the stack of dictionaries
   - The rules are different between static scoping and dynamic scoping
3. Given a name use, to find its bindings
   a) Search the dictionary on top of the stack; if found, return the bindings.
   b) Otherwise, repeat the process on the next dictionary down the stack.
   c) If the name is not found in any dictionary, report an error.

22

Static scoping

- The stack of dictionaries is built based on the lexical position of where a name appears

23

23

Dynamic Scoping

- In dynamic scoping, a name is bound to its most recent declaration based on the program's call stack
  - Used by Lisp, APL, Snobol, Perl.
- Stack of dictionaries corresponds to the call stack
- Dictionary for each scope built at compile time, but managed at run time.

24