Names, Scopes, Bindings

CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

Names in Programs

Scope: visibility of names

Storage: memory space associated with names

Lifetime: the time interval a variable is allocated with memory

Binding: a mapping between a name and its property

Scope

The visibility or availability of names

- Which code region is a name visible?

When is it determined?

- Static or Dynamic?
- Lexical or Control-Flow?

How & when does it change?

Static (Lexical) Scoping

Scopes can be determined by the compiler

All bindings for identifiers can be resolved by examining the program

Typically, use the closet binding

Used in most compiled languages (C, Java)

Scope Rules in Scheme

```
(let ((x 0))
    (+ x x))
```

Scope of x

```
(let ((x 0))
    (let ((y (+ x 1)))
      (+ x y)))
```

Scope of x

```
(define (f x)
   (g x x))
```

Scope of x

Scope of z

```
(let ((x 17))
    (lambda (z) (+ z x)))
```

(let*) Scope of x

```
(let* ((x 0) (y x))
    (+ x y))
```

Scope of inner y

```
(let ((x 0) (y 7))
    (let ((y (+ x 1))
          (x (+ y 1)))
      (+ x y)))
```

which y?

```
(define (f x)
   (let ((g *))
    (g x x)))
```

Scope of g

```
(define (f x)
   (let ((* +))
    (* x x)))
```

Scope of *

Scope Rules in Scheme

Three “let”s in Scheme

- let

```
(let ((x 0) (y 1))
     (+ x y))
```

Scope of x y

- let*

```
(let* ((x 0) (y x))
     (+ x y))
```

Scope of x

- letrec

```
(letrec ((isEven ((lambda n)
                  (if (zero? n) #t
                      (isOdd (- n 1)))))
        ((isOdd  ((lambda n)
                  (if (zero? n) #f
                      (isEven (- n 1))))))
```

Scope of isEven and isOdd

Nested Scope

In Scheme, each (let …) defines one scope

```
(let ((x 0))
     (let ((y 1)) (+ x y)))
```

In Pascal, ALGOL, function can be nested

```
function E (int x) {
  function F (int y) …
  return F(1)+x; }
// F is not visible here
```

In C, block scopes can be nested

```
function E (int x) {
  { int y=3; x=x+y;}
  // y is not visible here
  return x; }
```

How are scope rules being checked?

Scope Check

Symbol table: a table of names and their bindings

Single scope: a dictionary or hash map

Nested scope: a tree of symbol tables

- Each scope has one symbol table
- Each symbol table may have a parent

Symbol Table

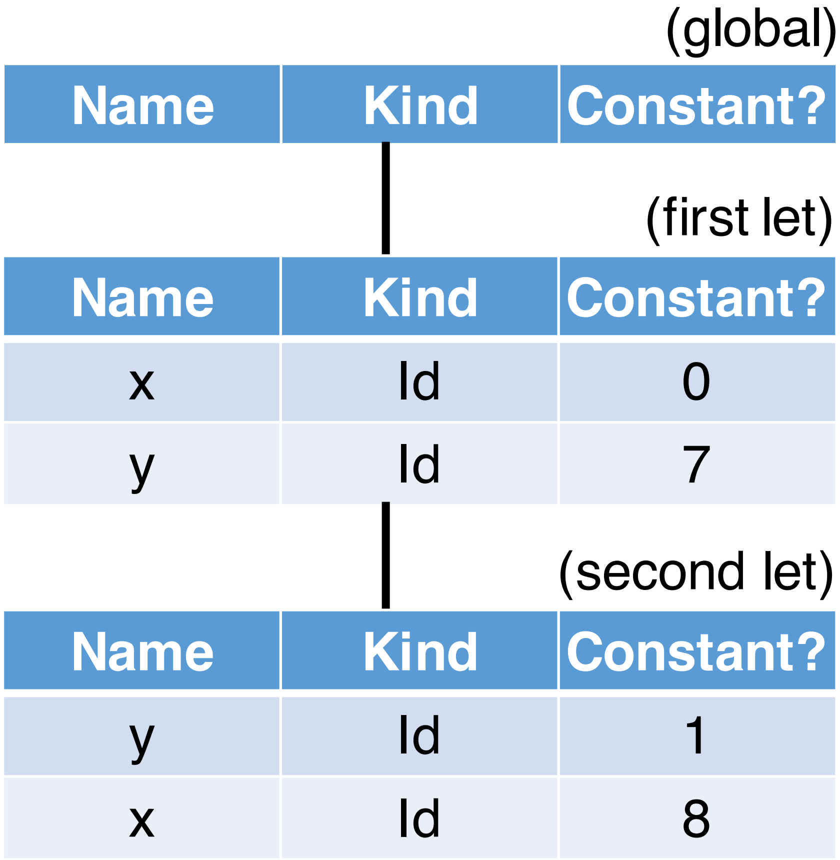
Each entry in the symbol table contains

- The name of an identifier
- Additional information: its kind, its type, if it is constant and so on

Name	Kind	Constant?
x	Id	0

Scope Check Example

```
(let ((x 0) (y 7))
   (let ((y (+ x 1))
         (x (+ y 1)))
     (+ x y)))
```



(global)

Name	Kind	Constant?

(first let)

Name	Kind	Constant?
x	Id	0
y	Id	7

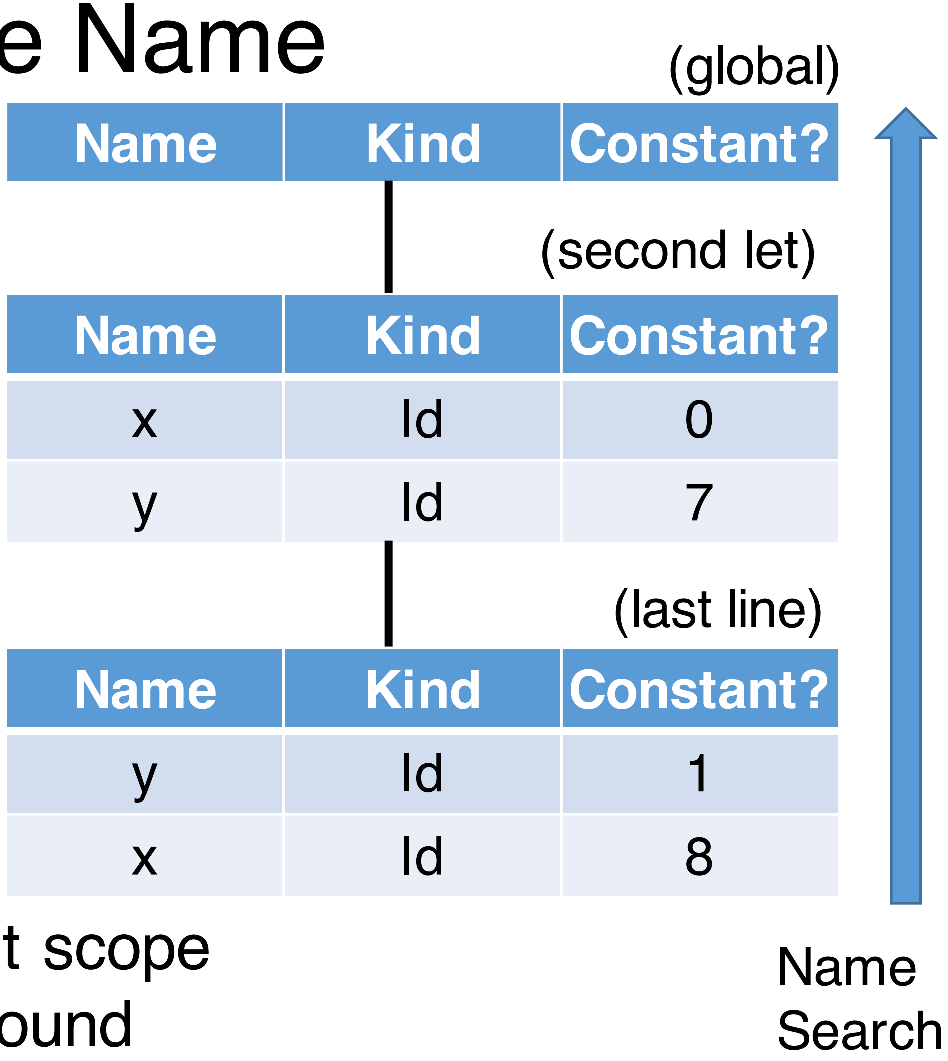
(second let)

Name	Kind	Constant?
y	Id	1
x	Id	8

Name Collision:
Identifier with Same Name

```
(let ((x 0) (y 7))
   (let ((y (+ x 1))
         (x (+ y 1)))
     (+ x y)))
```

which y?



(global)

Name	Kind	Constant?

(second let)

Name	Kind	Constant?
x	Id	0
y	Id	7

(last line)

Name	Kind	Constant?
y	Id	1
x	Id	8

Name Search

To find a binding:

1. Start from table of current scope
2. Go up until the name is found
3. Fail if no table contains the name

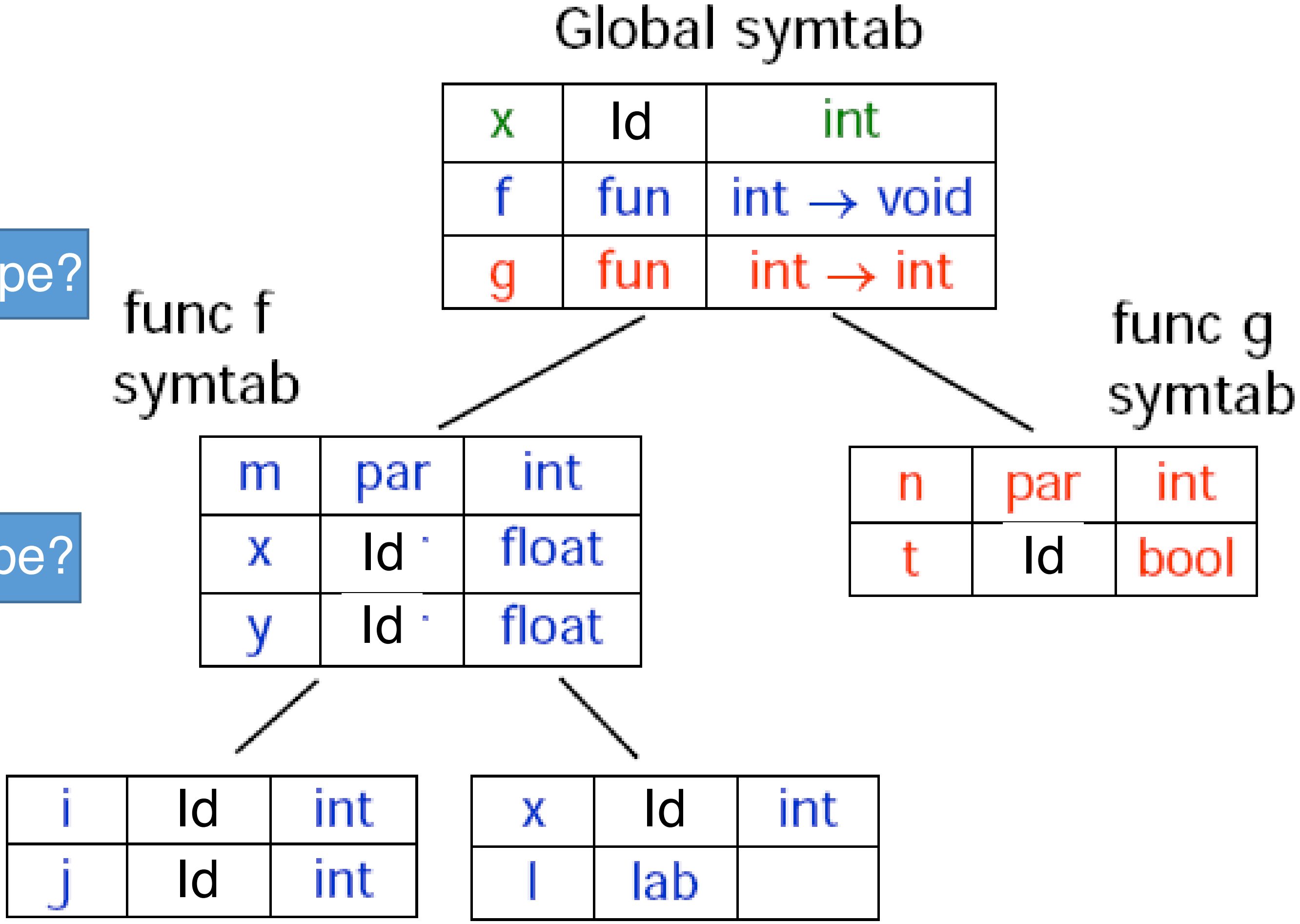
Scope Check Example

```
int x;

void f(int m) {
    float x, y;
    ...
  { int i, j; x = 1; }
  { int x; l: x = 2; }
}

int g(int n) {
    bool t;
    x = 3;
}
```

In scope?

In scope?

In scope?



Global symtab

x	Id	int
f	fun	int → void
g	fun	int → int

func f symtab

m	par	int
x	Id	float
y	Id	float

func g symtab

n	par	int
t	Id	bool

i	Id	int
j	Id	int

x	Id	int
l	lab	

Dynamic Scoping

Static Scoping

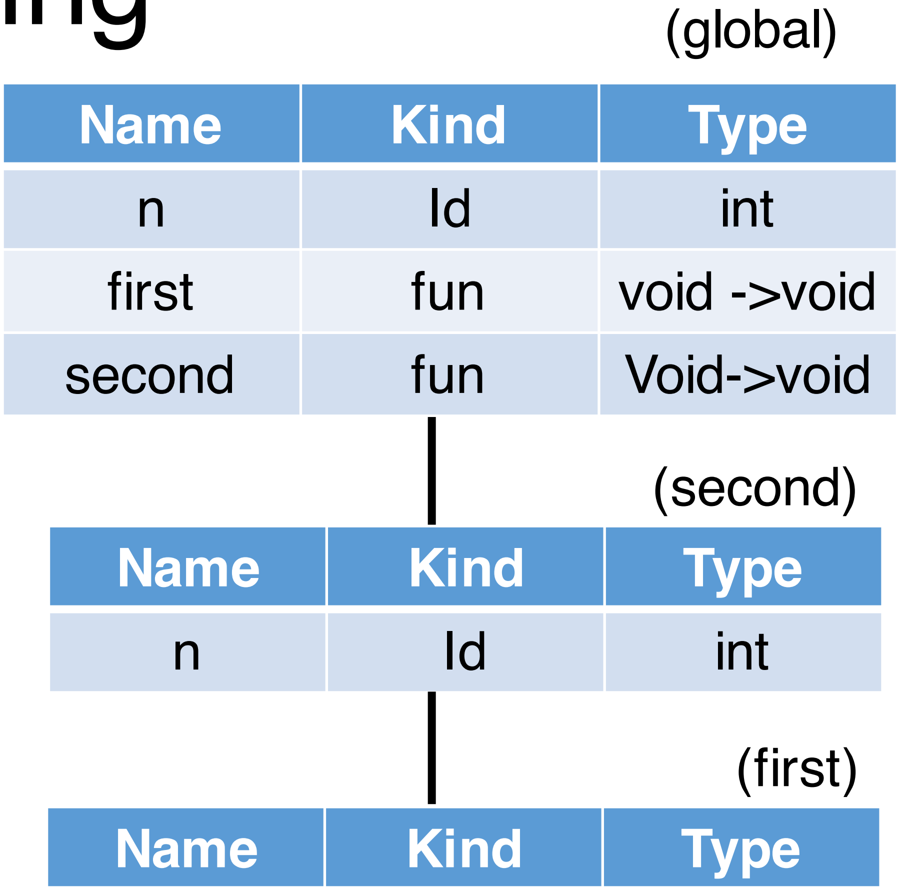
- Bindings determined by lexical structure
- Local renaming principle

Dynamic Scoping

- Bindings depend on flow of control
- Always use most recent, active binding
- Names important!
- Meaning of a variable can change

Dynamic Scoping

```
int n=2;

void first() {
 n = 1
}

void second() {
 int n=0;
 first();
}

first();
second();
```



(global)

Name	Kind	Type
n	Id	int
first	fun	void ->void
second	fun	Void->void

(second)

Name	Kind	Type
n	Id	int

(first)

Name	Kind	Type

Symbol tables changes at run time!
Always use most recent, active binding