5 Names & Scope

Why Names?

Video clip

5 Names & Scope

Scope & Blocks

Activation Records & Runtime Stack

Scope of Functions and Parameters

Static vs. Dynamic Scoping

Implementation of Static Scoping

Implementation of Recursion

Meaning of Names

Oxford English Dictionary | The definitive record of the English language

trondhjemite, *n*.

Pronunciation: /ˈtrɒndheɪmaɪt/

Etymology: < German *trondhjemit* (V. M. Goldschmidt 1916, in ...

Geol.

Any leucocratic tonalite, esp. one in which the plagioclase is oligoclase.

Jill likes oranges. Jane likes apples.
She enjoys eating them.

Scope of Symbols

Scope of a symbol:
All *locations* in a program where the symbol is visible

that particular definition

Things to know about scope

- Blocks (limited scope)
- Nested blocks (shadowing)
- Runtime stack & activation records
- Non-local variables
- Static vs. dynamic scoping

Blocks

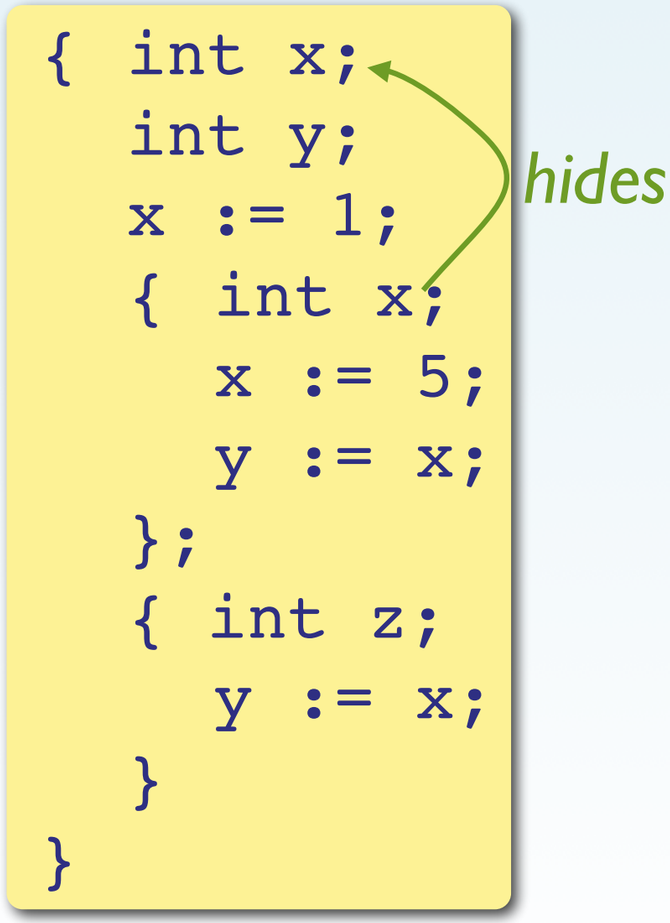
```
{  int x;❶
   int y;❷
  ❶x := 1;
   {  int x;❸
     ❸x := 5;
     ❷y := ❸x;
   };
   {  int z;❹
     ❷y := ❶x;
   }
}
```

A *block* consists of a group of declarations and
(a) a sequence of statements (in imperative languages)
(b) an expression (in functional languages)

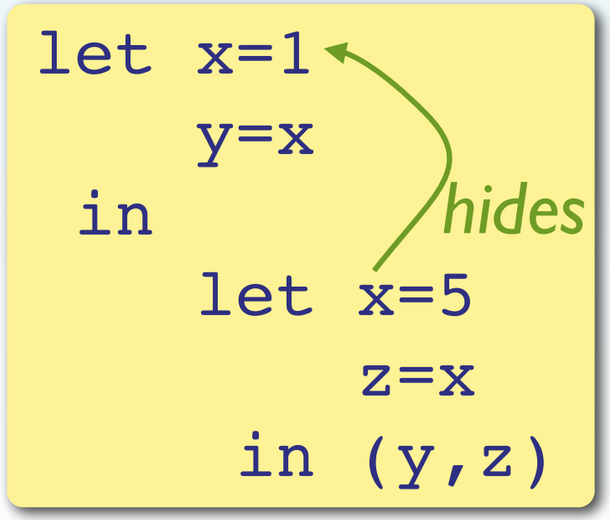
```
let ❶x=1
    ❷y=❶x
 in
    let ❸x=5
        ❹z=❸x
     in (❷y,❹z)
```

Observe references to
local and *non-local* variables

Nested Blocks: Shadowing



Declarations in inner blocks can temporarily hide declarations in enclosing blocks



Homonyms & Synonyms

A name is a homonym if it has more than one meaning.

$$C \neq C' \Rightarrow \texttt{sem}\ C\ x \neq \texttt{sem}\ C'\ x$$

context is needed for disambiguation

Two names x and y are synonyms if they have the same meaning.

$$\texttt{sem}\ x = \texttt{sem}\ y$$

Activation Records

Local variables are kept in memory blocks, called *activation records*, on the *runtime stack*

Enter/leave block: push/pop activation record on/off the runtime stack

```
{  int x;
   int y;
   x := 1;
   {  int x;
      x := 5;
      y := x;
   };
   {  int z;
      y := x;
   }
}
```

[]	
[⟨x:?, y:?⟩]	push
[⟨x:1, y:?⟩]	
[⟨x:?⟩, ⟨x:1, y:?⟩]	push
[⟨x:5⟩, ⟨x:1, y:?⟩]	
[⟨x:5⟩, ⟨x:1, y:5⟩]	
[⟨x:1, y:5⟩]	pop
[⟨z:?⟩, ⟨x:1, y:5⟩]	push
[⟨z:?⟩, ⟨x:1, y:1⟩]	
[⟨x:1, y:1⟩]	pop
[]	pop

A Simplified Model

A declaration of a group of variables is equivalent to a corresponding group of nested blocks for each variable

```
{  int x;
   int y;
   int z;
   x := 1;
   y := x;
}
```

≡

```
{  int x;
   {  int y;
      {  int z;
         x := 1;
         y := x;
      }
   }
}
```

```
let x=1
    y=2
 in x+y
```

≡

```
let x=1
 in  let y=2
      in x+y
```

... we can use activation records of single variables

Simplified Activation Records & Stacks

Enter/leave block:
push/pop activation record
on/off the runtime stack

```
let x=1
 in let y=2
     in x+y
```

[]	
[x:1]	push
[y:2, x:1]	push
[x:1]	pop
[]	pop

Exercise

What is the value of the following expression?

```
let x=1 in (let x=2 in x,x)
```

Example ...

Var.hs
(Variables and Definitions)

Scope of Functions and Parameters

```
{int x;
 {int f(int y){return y+1};
  x := f(1);

 }
}
```

[]	
[x:?]	push
[f:{}, x:?]	push
[y:1, f:{}, x:?]	push
[f:{}, x:2]	pop
[x:2]	pop
[]	pop

Dynamic Scoping

non-local variable

```
{int x;
 x := 1;
 {int f(int y){return y+x};
  {int x;
   x := 2;
   x := f(3);
  }
 }
}
```

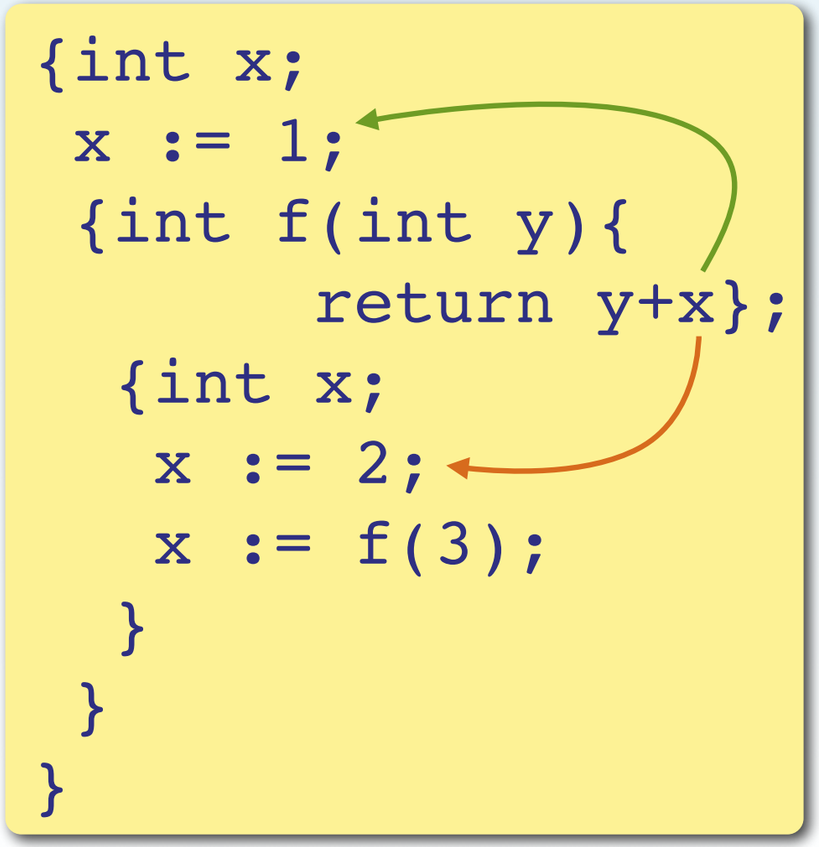
[]	
[x:?]	push
[x:1]	
[f:{}, x:1]	push
[x:?, f:{}, x:1]	push
[x:2, f:{}, x:1]	
[y:3, x:2, f:{}, x:1]	push
[x:5, f:{}, x:1]	pop
[f:{}, x:1]	pop
[x:1]	pop
[]	pop

Dynamic Scoping

Example

FunDynScope.hs
(Functions)

Static vs. Dynamic Scoping



Static scoping: A non-local name refers to the variable that is visible (= in scope) at the *definition* of a function

Dynamic scoping: A non-local name refers to the variable that is visible (= in scope) at the *use* of a function

Static Scoping

non-local variable

```
{int x;
 x := 1;
 {int f(int y){return y+x};
  {int x;
   x := 2;
   x := f(3);
  }
 }
}
```

[]	
[x:?]	push
[x:1]	
[f:{}, x:1]	push
[x:?, f:{}, x:1]	push
[x:2, f:{}, x:1]	
[y:3, x:2, f:{}, x:1]	push
[x:4, f:{}, x:1]	pop
[f:{}, x:1]	pop
[x:1]	pop
[]	pop

Static Scoping

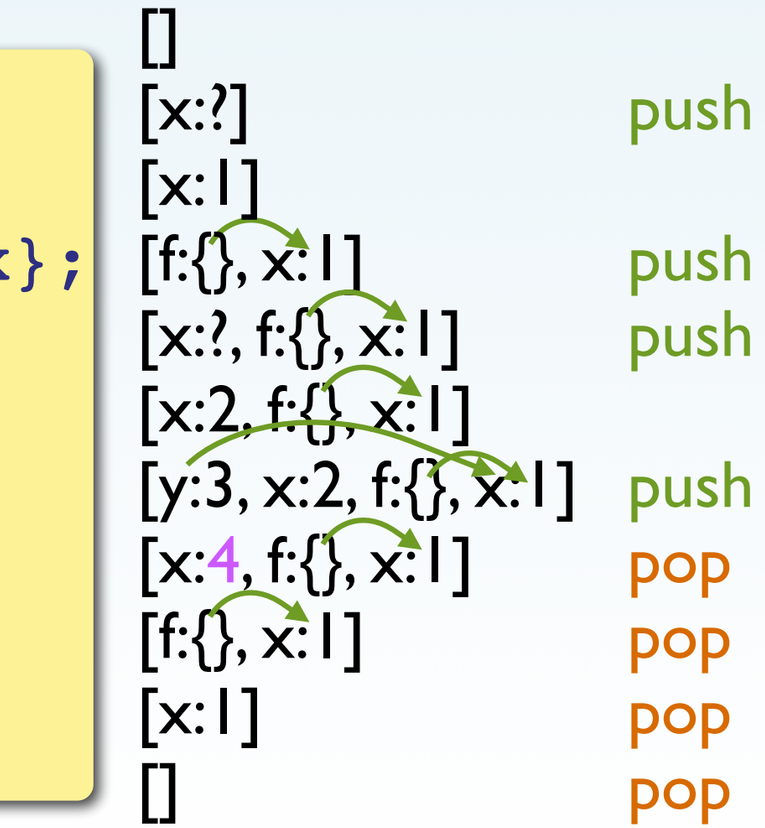
Implementation of Static Scoping

access link

How? Store a *pointer* to the previous activation record in the runtime stack with function definition

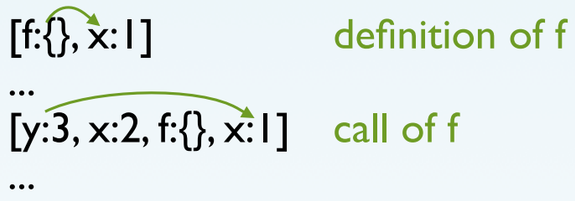
Goal: remember earlier definitions together with function definition

```
{int x;
 x := 1;
 {int f(int y){return y+x};
  {int x;
   x := 2;
   x := f(3);
  }
 }
}
```

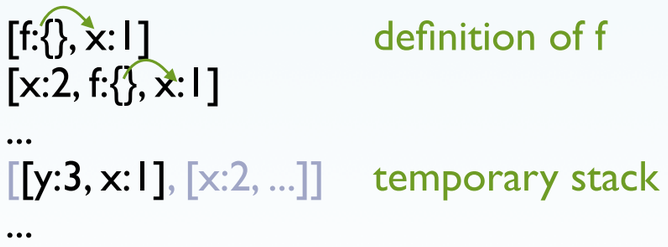


Two Interpretations of Access Links

When a function `f` (with parameter `y`) is called:



(a) Push activation record for `f` onto the runtime stack. *Follow access links* when searching for variables.



(b) Push activation record for `f` onto a temporary stack (the remainder of the runtime stack pointed to by the access link). *Evaluate `f` on temporary stack.*

Example

FunStatScope.hs
(Closures)

Dynamic vs. Static Scope: Runtime Stack

```
data Val = ...
         | F Name Expr

eval s (Fun x e)  = F x e
eval s (App f e') = case eval s f of
                      F x e → eval ((x,eval s e'):s) e
                      _     → Error
```

```
data Expr = ...
          | Fun Name Expr
```

```
data Val = ...
         | C Name Expr Stack

eval s (Fun x e)  = C x e s
eval s (App f e') = case eval s f of
                      C x e s' → eval ((x,eval s e'):s') e
                      _        → Error
```

Exercise

Show the development of the runtime stack under *static* and *dynamic* scoping for the execution of the following code.

```
{int y := 1;
 {int z := 0;
  {int f(int x){return y+x};
   {int g(int y){return f(2)};
    z := g(3);


   }
  ...
```

Exercise

Show the development of the runtime stack under *static* and *dynamic* scoping for the execution of the following code.

```
{int z := 0;
 {int f(int x){return x+1};
  {int g(int y){return f(y)};
   {int f(int x){return x-1};
    z := g(3);


   }
  ...
```

Implementation of Recursion

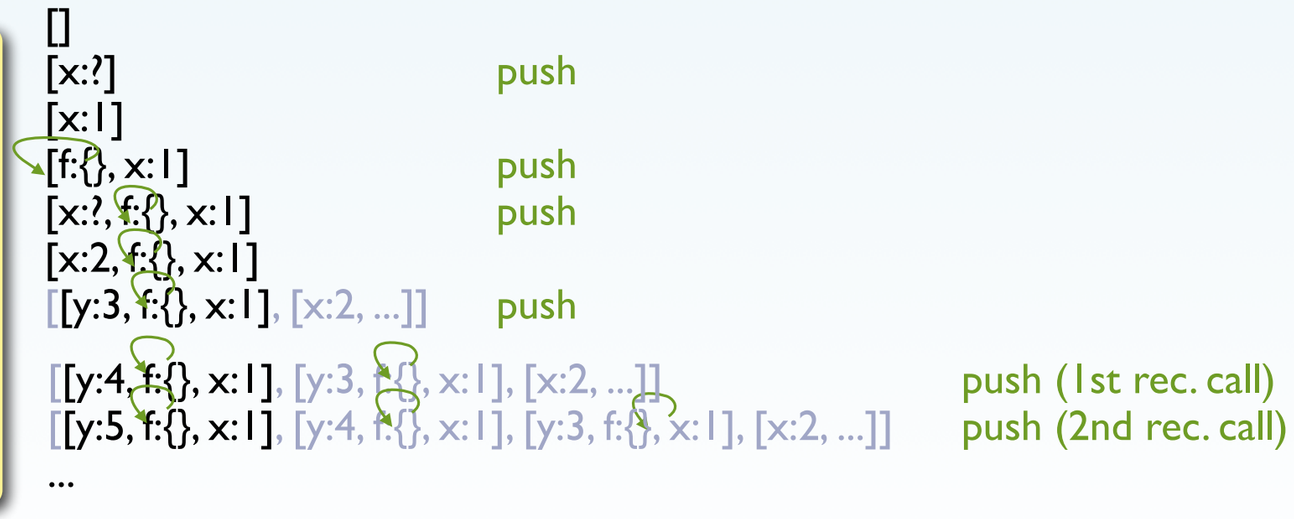
Problem: Need access to function definition when evaluating the function body

works for the 2nd interpretation of access links

Solution: Let *access link* point to the *very same* activation record in the runtime stack containing the function definition

```
{int x;
 x := 1;
 {int f(int y){return f(x+y)};
  {int x;
   x := 2;
   x := f(3);

 }
}
```



Example

FunRec.hs