What's In a Name? (Memory) location, location, location
Symbol Table Construction

Name Analysis

Kristoffer H. Rose
krisrose@cs.nyu.edu

Compiler Construction
CSCI-GA.2130-001/Fall 2013
NYU Courant Institute

October 3, 2013
1 What’s In a Name?

2 (Memory) location, location, location

3 Symbol Table Construction

Name Analysis

Kristoffer H. Rose krisrose@cs.nyu.edu
Spoiler Alert!

It is all about Names!
It is all about *Names*!
Suggestions?

- Variables.
- Parameters.
- Fields.
- Methods.
- Functions.
- Procedures.
- Modules.
- Types.
- . . .
Names Everywhere…

Suggestions?

▶ Variables.
▶ Parameters.
▶ Fields.
▶ Methods.
▶ Functions.
▶ Procedures.
▶ Modules.
▶ Types.
▶ …
Clarifications

- **Identifier** just string unit of characters.
- **Variable** denotes a location.
- **Name** possibly composite (System.out.println).
- Procedure callable unit.
- Function—returns value.
- Method—has special self parameter.
- Declaration specifies type and other properties.
- Definition specifies value.
Clarifications

- **Identifier** just string unit of characters.
- **Variable** denotes a location.
- **Name** possibly composite (System.out.println).
- **Procedure** callable unit.
- **Function**—returns value.
- **Method**—has special `self` parameter.

- Declaration specifies type and other properties.
- Definition specifies value.
Clarifications

- **Identifier** just string unit of characters.
- **Variable** denotes a location.
- **Name** possibly composite (System.out.println).
- **Procedure** callable unit.
- **Function**—returns value.
- **Method**—has special *self* parameter.
- **Declaration** specifies type and other properties.
- **Definition** specifies value.
Compiler issues with names:

- Where is it visible?
- How is it defined?
- What code is needed where it is used?

For example:

- Procedures have call.
- Variables have load and store.
- —so can a parameter only be loaded while the function is being called?
Compiler issues with names:

- Where is it visible?
- How is it defined?
- What code is needed where it is used?

For example:

- Procedures have call.
- Variables have load and store.
- —so can a parameter only be loaded while the function is being called?
1. What’s In a Name?

2. (Memory) location, location, location

3. Symbol Table Construction
Name = Memory Location

- environment
  - name
  - location
  - value

state
Location Issues

- How is the memory address determined?
- When is the memory address known?
- What does the memory address depend on?
**Static vs. Dynamic**

**Definition (Static)**
Completely defined/determined at *compile-time*.

**Definition (Dynamic)**
Not static (!)

*Compiler writers like static.*
**Static vs. Dynamic**

**Definition (Static)**
Completely defined/determined at *compile-time*.

**Definition (Dynamic)**
Not static (!)

*Compiler writers like static.*
Static vs. Dynamic

**Definition (Static)**
Completely defined/determined at *compile-time*.

**Definition (Dynamic)**
Not static (!)

*Compiler writers like static.*
A Program

function $F() = \text{let } x := 1 \text{ in } G() \text{ end}$

function $G() = x \leftarrow \text{Bad!}$

$\text{print}(F())$
A Program

function $F() = \text{let } x := 1 \text{ in } G() \text{ end}$

function $G() = x \leftarrow \text{Bad!}$

print($F()$)
Another Program

function $F(x: \text{int}) = G()$

function $G() = x \leftarrow \text{Bad!}$

print($F(5)$)
Another Program

\[
\text{function } F(x:int) = G()
\]

\[
\text{function } G() = x \leftarrow \text{Bad!}
\]

\[
\text{print}(F(5))
\]
function $F() = G()$

function $G() = x \leftarrow \text{Bad!}$

```{.lang-plaintext}
var $x : int := 5$

print($F()$)
```
Yet Another Program

function $F() = G()$

function $G() = x \leftarrow \text{Bad!}$

var $x : \text{int} := 5$

print($F()$)
Yet Another Another Program

\[
\begin{align*}
\text{var } x &: \text{ int } := 5 \\
\text{function } F(x: \text{int}) &= G() \\
\text{function } G() &= x \\
\text{print}(F(3)) &\leftarrow \text{prints } 5
\end{align*}
\]
var x : int := 5

function F(x:int) = G()

function G() = x

print(F(3)) ⇐ prints 5
Static Scoping

```
var a : int := 1  
var b : int := 1 
let var b : int := 2 
in  let var a : int := 3; 
  in  print(a); print(b)  ⇐ prints 3 2 
end; 
let var b : int := 4; 
in  print(a); print(b)  ⇐ prints 1 4 
end 
end; 
print(a); print(b)  ⇐ prints 1 1 
```
What's In a Name? (Memory) location, location, location
Symbol Table Construction

Static Scoping

```plaintext
var a : int := 1
var b : int := 1
let var b : int := 2
in let var a : int := 3;
    in print(a); print(b)  \<\> prints 3 2
end;
let var b : int := 4;
in  print(a); print(b)  \<\> prints 1 4
end
end;
print(a); print(b)  \<\> prints 1 1
```

Name Analysis

Kristoffer H. Rose  krisrose@cs.nyu.edu
What's In a Name? (Memory) location, location, location
Symbol Table Construction

Static Scoping

```
var a : int := 1
var b : int := 1
let var b : int := 2
in let var a : int := 3;
in print(a); print(b)  \leftarrow \text{prints 3 2}
end;
let var b : int := 4;
in print(a); print(b)  \leftarrow \text{prints 1 4}
end;
print(a); print(b)  \leftarrow \text{prints 1 1}
```
What's In a Name? (Memory) location, location, location

Symbol Table Construction

Static Scoping

```plaintext
var a : int := 1
var b : int := 1
let var b : int := 2
in let var a : int := 3;
    in print(a); print(b) ⇐ prints 3 2
end;
let var b : int := 4;
    in print(a); print(b) ⇐ prints 1 4
end
end;
print(a); print(b) ⇐ prints 1 1
```
Parameters

- **Formal** parameter—the one in the procedure declaration.
- **Actual** parameter—the one in the procedure call.

```plaintext
function f(a:int, b:int): int = (a := a+1; a+b)
var c : int := 1
print(f(c,c))  ⇐ prints 3 (CbV) or 4 (CbR)
```

- **Call-by-Value**—the actual parameter is first evaluated to a value, then copied into the formal parameter location.
- **Call-by-Reference**—the actual parameter location is used as the formal parameter location—creates aliasing issues.
Parameters

- **Formal** parameter—the one in the procedure declaration.
- **Actual** parameter—the one in the procedure call.

```plaintext
function f(a:int, b:int): int = (a := a + 1; a + b)
var c : int := 1
print(f(c, c))  \iff prints 3 (CbV) or 4 (CbR)
```

- Call-by-Value—the actual parameter is first evaluated to a value, then copied into the formal parameter location.
- Call-by-Reference—the actual parameter location is used as the formal parameter location—creates aliasing issues.
Parameters

- **Formal** parameter—the one in the procedure declaration.
- **Actual** parameter—the one in the procedure call.

```plaintext
function f(a:int, b:int): int = (a := a + 1; a + b)
var c : int := 1
print(f(c, c))← prints 3 (CbV) or 4 (CbR)
```

- **Call-by-Value**—the actual parameter is first evaluated to a value, then copied into the formal parameter location.
- **Call-by-Reference**—the actual parameter location is used as the formal parameter location—creates aliasing issues.
What's In a Name? (Memory) location, location, location

Symbol Table Construction

Parameters

- **Formal** parameter—the one in the procedure declaration.
- **Actual** parameter—the one in the procedure call.

```python
function f(a:int, b:int): int = (a := a + 1; a + b)
var c : int := 1
print(f(c,c))  # prints 3 (CbV) or 4 (CbR)
```

- **Call-by-Value**—the actual parameter is first evaluated to a value, then copied into the formal parameter location.
- **Call-by-Reference**—the actual parameter location is used as the formal parameter location—creates aliasing issues.
Parameters

- **Formal** parameter—the one in the procedure declaration.
- **Actual** parameter—the one in the procedure call.

```plaintext
function f(a:int, b:int): int = (a := a + 1; a + b)

var c : int := 1
print(f(c, c)) \leftarrow prints 3 (CbV) or 4 (CbR)
```

- **Call-by-Value**—the actual parameter is first evaluated to a value, then copied into the formal parameter location.
- **Call-by-Reference**—the actual parameter location is used as the formal parameter location—creates aliasing issues.
Runtime Memory

Keep track of static vs dynamic.

“Weakest link:” location is typically sum of many components.

Scoping means static/dynamic may vary over the compilation.
Keep track of **static vs dynamic**.

“**Weakest link:**” location is typically sum of many components.

Scoping means static/dynamic may vary over the compilation.
Variable on stack

frame pointer

y

2

y
Array member

frame

pointer

\[ x[y] \]

\[ 2 \]

\[ y \]
Object/record member

What's In a Name?

(Memory) location, location, location

Symbol Table Construction

Name Analysis

Kristofer H. Rose  krisrose@cs.nyu.edu
1. **What’s In a Name?**

2. **(Memory) location, location, location**

3. **Symbol Table Construction**
Symbol Tables

- Holds information about source program constructs.
- One per scope (but they can share).
Some Scopes (Ex. 2.15)

```c
{ int x1; int y1;
  { int w2; bool y2; int z2;
    ...w2...; ...x1...; ...y2...; ...z2...;
  }
  ...w0...; ...x1...; ...y1...;
}
```
Some Scopes (Ex. 2.15)

```c
1 {  int x1; int y1;
2   {  int w2; bool y2; int z2;
3      ... w2 ...;  ... x1 ...;  ... y2 ...;  ... z2 ...;
4   }
5     ... w0 ...;  ... x1 ...;  ... y1 ...;
6  }
```
Some Scopes (Ex. 2.15)

{  int x1; int y1;
  {  int w2; bool y2; int z2;
      ...w2...;  ...x1 ...;  ...y2 ...;  ...z2 ...;
  }
  ...
  ...
  ...
  }
  ...
  ...
  ...
  }

Name Analysis  
Kristoffer H. Rose  krisrose@cs.nyu.edu
Some Scopes (Ex. 2.15)

```c
{ int x1; int y1;
  { int w2; bool y2; int z2;
    ...w2...; ...x1 ...; ...y2 ...; ...z2 ...;
  }
  ...w0...; ...x1 ...; ...y1 ...;
}
```
1–2–3–Hack!
Summary

1. What’s In a Name?
2. (Memory) location, location, location
3. Symbol Table Construction

Questions?