CS 314 Principles of Programming Languages

Lecture 12: Names, Scopes and Bindings

Zheng (Eddy) Zhang
Rutgers University
February 26, 2018
Class Information

- HW4 posted, due this Friday.
- Reminder: Project 1 due in about one week.
What’s in a name? — Each name “means” something!

• Denotes a programming language construct

• Has associated “attributes”
  Examples: type, memory location, read/write permission, storage class, access restrictions.

• Has a meaning
  Examples: represents a semantic object, a type description, an integer value, a function value, a memory address.
Names, Bindings and Memory

**Bindings** – association of a name with the thing it “names” (e.g., a name and a memory location, a function name and its “meaning”, a name and a value)

- **Compile time**: during compilation process - static (e.g.: macro expansion, type definition)
- **Link time**: separately compiled modules/files are joined together by the linker (e.g: adding the standard library routines for I/O (stdio.h), external variables)
- **Run time**: when program executes - dynamic

Compiler needs bindings to know meaning of names during translation (and execution).
• **Early binding** times — more efficient (faster) at run time
• **Late binding** times — more flexible (postpone binding decision until more “information” is available)

• Examples of static binding (early):
  - functions in C
  - types in C

• Examples of dynamic binding (late):
  - virtual methods in Java
  - dynamic typing in Javascript, Scheme

Note: dynamic linking is somewhat in between static and dynamic binding; the function signature has to be known (static), but the implementation is linked and loaded at run time (dynamic).
How to Maintain Bindings

• Symbol table: maintained by compiler during compilation
  \(\text{names} \Rightarrow \text{attributes}\)

• Environment: maintained by compiler-generated-code during program execution
  \(\text{names} \Rightarrow \text{memory locations}\)

Question:

• How long do bindings last for a name hold in a program?
• What initiates a binding?
• What ends a binding?
program L;
    var n: char;            {n declared in L}
procedure W;
begin
    write (n);    {n referenced in W}
end;
procedure D;
begin
    var n: char; {n declared in D}
    n := ‘D’;     {n referenced in D}
    W
end;
begin
    n := ‘L’;                 {n referenced in L}
    W;
    D
end
Scope Example

Nested Subroutines (Algol 60, Ada, ML, Common Lisp, Python, ….)

```plaintext
program L;
    var n: char; {n declared in L}

procedure W;
    begin
        write (n); {n referenced in W}
    end;

procedure D;
    var n: char; {n declared in D}
    begin
        n := 'D'; {n referenced in D}
        W
    end;
    begin
        n := 'L'; {n referenced in L}
        W; D
    end
```
Scope Example

Nested Subroutines (Algol 60, Ada, ML, Common Lisp, Python, ….)

program L;
    var n: char; {n declared in L}
    procedure W;
    begin
        write (n); {n referenced in W}
    end;
    procedure D;
        var n: char; {n declared in D}
        begin
            n := 'D'; {n referenced in D}
            W
        end;
    begin
        n := 'L'; {n referenced in L}
        W;
        D
    end
Scope Example

Nested Subroutines (Algol 60, Ada, ML, Common Lisp, Python, ….)

program L;
  var n: char; {n declared in L}
procedure W;
begin
  write (n); {n referenced in W}
end;
procedure D;
begin
  n := ‘D’; {n referenced in D}
  W
end;
begin
  n := ‘L’; {n referenced in L}
  W;
  D
end
Scope Example

Nested Subroutines (Algol 60, Ada, ML, Common Lisp, Python, ….)

```plaintext
program L;
    var n: char;            {n declared in L}
procedure W;
    begin
        write (n);    {n referenced in W}
    end;
procedure D;
    var n: char; {n declared in D}
    begin
        n := 'D';     {n referenced in D}
        W
    end;
begin
    n := 'L';                 {n referenced in L}
    W;
    D
end
```
Lexical Scope

- Non-local variables are associated with declarations at *compile* time
- Find the smallest block syntactically enclosing the reference and containing a declaration of the variable

**Example:**
- The reference to \( n \) in \( W \) is associated with two different declarations at two different times
- The output is?

**Calling Chain:**

\[
L \Rightarrow W \\
L \Rightarrow D \Rightarrow W
\]
Lexical Scope

- Non-local variables are associated with declarations at compile time
- Find the smallest block syntactically enclosing the reference and containing a declaration of the variable

Example:
- The reference to n in W is associated with two different declarations at two different times
- The output is?

Calling Chain:

\[
\begin{align*}
L & \Rightarrow W \\
L & \Rightarrow D \Rightarrow W
\end{align*}
\]
Dynamic Scope

- Non-local variables are associated with declarations at run time
- Find the most recent, currently active run-time stack frame containing a declaration of the variable

Example:
- The reference to n in W is associated with two different declarations at two different times
- The output is ?

    program L;
    var n: char; {n declared in L}
    procedure W;
    begin
        write (n); {n referenced in W}
    end;
    procedure D;
    var n: char; {n declared in D}
    begin
        n := ‘D’; {n referenced in D}
        W
    end;
    begin 
        n := ‘L’; {n referenced in L}
        W;
        D
    end
Review: Program Memory Layout

- Static objects are given an absolute address that is retained throughout the execution of the program
- Stack objects are allocated and deallocated in last-in, first-out order, usually in conjunction with subroutine calls and returns
- Heap objects are allocated and deallocated at any arbitrary time
Procedure Activations

- Begins when control enters activation (call)
- Ends when control returns from call

Example:

```
procedure C:
  D
procedure B:
  if...then B else C
procedure A:
  B
main program:
  A
```

Calling chain: A ⇒ B ⇒ B ⇒ C ⇒ D

Diagram:

- **procedure C:** D
- **procedure B:** if...then B else C
- **procedure A:** B
- **main program:** A

Direction of stack growth (usually lower addresses):

- Parameter
- Return value
- Return address
- Access link
- Caller FP
- Local variables
Procedure Activations

- Run-time stack contains frames from main program & active procedure
- Each stack frame includes:
  1. Pointer to stack frame of caller (**control link** for stack maintenance and dynamic scoping)
  2. Return address (within calling procedure)
  3. Mechanism to find non-local variables (**access link** for lexical scoping)
  4. Storage for parameters, local variables and final values

<table>
<thead>
<tr>
<th>parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>return value</td>
</tr>
<tr>
<td>return address</td>
</tr>
<tr>
<td>access link</td>
</tr>
<tr>
<td>caller FP</td>
</tr>
<tr>
<td>local variables</td>
</tr>
</tbody>
</table>
How do we look for non-local variables?

Program

x, y: integer   // declarations of x and y
begin
  Procedure B   // declaration of B
    y, z: real  // declaration of y and z
    begin
      ...
      y = x + z // occurrences of y, x, and z
    if (...) call B // occurrence of B
    end
end

Procedure C   // declaration of C
  x: real
  begin
      ...
      call B // occurrence of B
  end

... call C  // occurrence of C
call B   // occurrence of B
end
Lexical Scoping and Dynamic Scoping Example

How do we look for non-local variables?

Program

\begin{verbatim}
x, y: integer  // declarations of x and y
begin
  Procedure B  // declaration of B
    y, z: real  // declaration of y and z
    begin
      ... 
      y = x + z // occurrences of y, x, and z
      if (...) call B // occurrence of B
    end
  Procedure C  // declaration of C
    x: real
    begin
      ...
      call B // occurrence of B
    end
  ...
end
\end{verbatim}
Lexical Scoping and Dynamic Scoping Example

How do we look for non-local variables?

Program

\[ x, y: \text{integer} \quad // \text{declarations of } x \text{ and } y \]
begin

Procedure B \quad // \text{declaration of } B
\[ y, z: \text{real} \quad // \text{declaration of } y \text{ and } z \]
begin
\[ ... \]
\[ y = x + z \quad // \text{occurrences of } y, x, \text{ and } z \]
\[ \text{if } (...) \text{ call } B \quad // \text{occurrence of } B \]
end

Procedure C \quad // \text{declaration of } C
\[ x: \text{real} \]
begin
\[ ... \]
\[ \text{call } B \quad // \text{occurrence of } B \]
end
\[ ... \]
\[ \text{call } C \quad // \text{occurrence of } C \]
\[ \text{call } B \quad // \text{occurrence of } B \]
end
Lexical Scoping and Dynamic Scoping Example

How do we look for non-local variables?

Program

x, y: integer  // declarations of x and y
begin

Procedure B  // declaration of B
  y, z: real  // declaration of y and z
  begin
    ...
    y = x + z // occurrences of y, x, and z
    if (...) call B // occurrence of B
  end

Procedure C  // declaration of C
  x: real
  begin
    ...
    call B // occurrence of B
  end

...  // occurrence of C

end
How do we look for non-local variables?

Program

\[
x, y: \text{integer} \quad // \text{declarations of } x \text{ and } y
\]
begin

Procedure B \quad // \text{declaration of } B
\[
y, z: \text{real} \quad // \text{declaration of } y \text{ and } z
\]
begin

\[
... \\
y = x + z \quad // \text{occurrences of } y, x, \text{ and } z
\]
if (...) call B \quad // \text{occurrence of } B
end

Procedure C \quad // \text{declaration of } C
\[
x: \text{real}
\]
begin

\[
... \\
call B \quad // \text{occurrence of } B
\]
end

\[
... \\
call C \quad // \text{occurrence of } C \\
call B \quad // \text{occurrence of } B
\]
Lexical Scoping and Dynamic Scoping Example

Calling chain: MAIN ⇒ C ⇒ B ⇒ B

Program
  x, y: integer   // declarations of x and y
  begin
    Procedure B   // declaration of B
      y, z: real  // declaration of y and z
      begin
        ...
        y = x + z // occurrences of y, x, and z
        if (...) call B // occurrence of B
        end
    Procedure C   // declaration of C
      x: real
      begin
        ...
        call B // occurrence of B
        end
    ...
    call C       // occurrence of C
    call B       // occurrence of B
  end
Access links and control links are used to look for non-local variable references.

**Static Scope:**

*Access link points to the stack frame of the most recently activated lexically enclosing procedure*

⇒ Non-local name binding is determined at compile time, and implemented at run-time

**Dynamic Scope:**

*Control link points to the stack frame of caller*

⇒ Non-local name binding is determined and implemented at run-time
Access to Non-Local Data

How does the code find non-local data at run-time?

Real globals:

• visible everywhere
• translated into an address at compile time

Lexical scoping:

• view variables as (level, offset) pairs, (compile-time symbol table)
• use (level, offset) pair to get address by using chains of access link (at run-time)

Dynamic scoping:

• variable names are preserved
• look-up of variable name uses chains of control links (at run-time)
Lexical Scoping

Symbol table generated at compile time matches declarations and occurrences.
⇒ Each name can be represented as a pair (nesting_level, local_index).

Program
x, y: integer  // declarations of x and y
begin
  Procedure B  // declaration of B
    y, z: real  // declaration of y and z
    begin
      ...  
      y = x + z // occurrences of y, x, and z
      if (...) call B // occurrence of B
    end
  Procedure C  // declaration of C
    x: real
    begin
      ...
      call B // occurrence of B
    end
  ...
  call C  // occurrence of C
  call B  // occurrence of B
end

Program
(1,1), (1,2): integer  // declarations of x and y
begin
  Procedure (1,3)  // declaration of B
    (2,1), (2,2): real  // declaration of y and z
    begin
      ...
      (2,1) = (1,1) + (2,2) // occurrences of y, x, and z
      if (...) call (1,3) // occurrence of B
    end
  Procedure (1,4)  // declaration of C
    (2,1): real
    begin
      ...
      call (1,3) // occurrence of B
    end
  ...
  call (1,4)  // occurrence of C
  call (1,3)  // occurrence of B
end
Lexical Scoping

Symbol table generated at compile time matches declarations and occurrences. ⇒ Each name can be represented as a pair (nesting_level, local_index).

Program

\[
\text{x, y: integer} \quad // \text{declarations of x and y}\\
\begin{aligned}
\text{begin}\\
\text{Procedure B} \quad // \text{declaration of B}\\
\text{y, z: real} \quad // \text{declaration of y and z}\\
\text{begin}\\
\text{...}\\
\text{y = x + z} \quad // \text{occurrences of y, x, and z}\\
\text{if (\ldots) call B} \quad // \text{occurrence of B}\\
\text{end}\\
\text{Procedure C} \quad // \text{declaration of C}\\
\text{x: real}\\
\text{begin}\\
\text{...}\\
\text{call B} \quad // \text{occurrence of B}\\
\text{end}\\
\text{...}\\
\text{call C} \quad // \text{occurrence of C}\\
\text{call B} \quad // \text{occurrence of B}\\
\text{end}
\end{aligned}
\]

Program

\[
\text{(1,1), (1,2): integer} \quad // \text{declarations of x and y}\\
\begin{aligned}
\text{begin}\\
\text{Procedure (1,3) \quad // declaration of B}\\
\text{(2,1), (2,2): real} \quad // \text{declaration of y and z}\\
\text{begin}\\
\text{...}\\
\text{(2,1) = (1,1) + (2,2)} \quad // \text{occurrences of y, x, and z}\\
\text{if (\ldots) call (1,3)} \quad // \text{occurrence of B}\\
\text{end}\\
\text{Procedure (1,4) \quad // declaration of C}\\
\text{(2,1): real}\\
\text{begin}\\
\text{...}\\
\text{call (1,3)} \quad // \text{occurrence of B}\\
\text{end}\\
\text{...}\\
\text{call (1,4)} \quad // \text{occurrence of C}\\
\text{call (1,3)} \quad // \text{occurrence of B}\\
\text{end}
\end{aligned}
\]
Lexical Scoping

Symbol table generated at compile time matches declarations and occurrences. Each name can be represented as a pair (nesting_level, local_index).

Program
x, y: integer  // declarations of x and y
begin
  Procedure B  // declaration of B
  y, z: real   // declaration of y and z
  begin
    ... y = x + z // occurrences of y, x, and z
    if (...) call B // occurrence of B
  end
Procedure C  // declaration of C
x: real
begin
  ...
  call B // occurrence of B
end
...
call C  // occurrence of C
call B  // occurrence of B
end

Program
(1,1), (1,2): integer  // declarations of x and y
begin
  Procedure (1,3)  // declaration of B
  (2,1), (2,2): real  // declaration of y and z
  begin
    ... (2,1) = (1,1) + (2,2) // occurrences of y, x, and z
    if (...) call (1,3) // occurrence of B
  end
Procedure (1,4)  // declaration of C
(2,1): real
begin
  ...
  call (1,3) // occurrence of B
end
...
call (1,4)  // occurrence of C
call (1,3)  // occurrence of B
end
Lexical Scoping

Symbol table generated at compile time matches declarations and occurrences.
⇒ Each name can be represented as a pair (nesting_level, local_index).

Program

```
x, y: integer   // declarations of x and y
begin
  Procedure B   // declaration of B
    y, z: real  // declaration of y and z
    begin
      ...   
      y = x + z // occurrences of y, x, and z
      if (...) call B // occurrence of B
    end
  Procedure C   // declaration of C
    x: real
      begin
        ...
        call B // occurrence of B
      end
    ...
    call C  // occurrence of C
    call B  // occurrence of B
end
```

Program

```
(1,1), (1,2): integer   // declarations of x and y
begin
  Procedure (1,3) // declaration of B
    (2,1), (2,2): real  // declaration of y and z
      begin
        ...
        (2,1) = (1,1) + (2,2) // occurrences of y, x, and z
        if (...) call (1,3) // occurrence of B
      end
  Procedure (1,4) // declaration of C
    (2,1): real
      begin
        ...
        call (1,3) // occurrence of B
      end
    ...
    call (1,4) // occurrence of C
    call (1,3) // occurrence of B
end
```
Lexical Scoping

Symbol table generated at compile time matches declarations and occurrences.
⇒ Each name can be represented as a pair (nesting_level, local_index).

Program
  x, y: integer   // declarations of x and y
  begin
    Procedure B   // declaration of B
      y, z: real  // declaration of y and z
      begin
        ... 
        y = x + z // occurrences of y, x, and z
        if (...) call B // occurrence of B
      end
    Procedure C   // declaration of C
      x: real
      begin
        ... 
        call B // occurrence of B
      end
  end

Program
  (1,1), (1,2): integer   // declarations of x and y
  begin
    Procedure (1,3)   // declaration of B
      (2,1), (2,2): real  // declaration of y and z
      begin
        (2,1) = (1,1) + (2,2) // occurrences of y, x, and z
        if (...) call (1,3) // occurrence of B
      end
    Procedure (1,4)   // declaration of C
      (2,1): real
      begin
        ... 
        call (1,3) // occurrence of B
      end
  end
Lexical Scoping

Symbol table generated at compile time matches declarations and occurrences. ⇒ Each name can be represented as a pair (nesting_level, local_index).

Program

```
x, y: integer   // declarations of x and y
begin
  Procedure B   // declaration of B
  y, z: real    // declaration of y and z
  begin
    ...        // occurrences of y, x, and z
    y = x + z   // occurrences of y, x, and z
    if (...) call B   // occurrence of B
  end
  Procedure C   // declaration of C
  x: real
  begin
    ...
    call B   // occurrence of B
  end
  ...        // occurrences of y, x, and z
  call C   // occurrence of C
  call B   // occurrence of B
end
```

Program

```
(1,1), (1,2): integer   // declarations of x and y
begin
  Procedure (1,3)   // declaration of B
  (2,1), (2,2): real // declaration of y and z
  begin
    ...        // occurrences of y, x, and z
    (2,1) = (1,1) + (2,2)   // occurrences of y, x, and z
    if (...) call (1,3)   // occurrence of B
  end
  Procedure (1,4)   // declaration of C
  (2,1): real
  begin
    ...
    call (1,3)   // occurrence of B
  end
  ...        // occurrences of y, x, and z
  call (1,4)   // occurrence of C
  call (1,3)   // occurrence of B
end
```
Lexical Scoping

Symbol table generated at compile time matches declarations and occurrences. ⇒ Each name can be represented as a pair (nesting_level, local_index).

Program

\( x, y: \text{integer} \quad // \text{declarations of } x \text{ and } y \)
begin

\text{Procedure } B \quad // \text{declaration of } B
\( y, z: \text{real} \quad // \text{declaration of } y \text{ and } z \)
begin
  ...
  y = x + z \quad // \text{occurrences of } y, x, \text{ and } z
  if (...) call B \quad // \text{occurrence of } B
end

\text{Procedure } C \quad // \text{declaration of } C
\( x: \text{real} \)
begin
  ...
  call B \quad // \text{occurrence of } B
end

...

call C \quad // \text{occurrence of } C
call B \quad // \text{occurrence of } B
end

Program

\( (1,1), (1,2): \text{integer} \quad // \text{declarations of } x \text{ and } y \)
begin

\text{Procedure } (1,3) \quad // \text{declaration of } B
\( (2,1), (2,2): \text{real} \quad // \text{declaration of } y \text{ and } z \)
begin
  ...
  (2,1) = (1,1) + (2,2) \quad // \text{occurrences of } y, x, \text{ and } z
  if (...) call (1,3) \quad // \text{occurrence of } B
end

\text{Procedure } (1,4) \quad // \text{declaration of } C
\( (2,1): \text{real} \)
begin
  ...
  call (1,3) \quad // \text{occurrence of } B
end

...

call (1,4) \quad // \text{occurrence of } C
call (1,3) \quad // \text{occurrence of } B
end
Lexical Scoping

Symbol table generated at compile time matches declarations and occurrences.
⇒ Each name can be represented as a pair (nesting_level, local_index).

Program

```
x, y: integer   // declarations of x and y
begin
  Procedure B   // declaration of B
    y, z: real  // declaration of y and z
    begin
      y = x + z // occurrences of y, x, and z
      if (...) call B // occurrence of B
    end
  Procedure C   // declaration of C
    x: real
    begin
      call B // occurrence of B
    end
  ... 
  call C   // occurrence of C
  call B   // occurrence of B
end
```

Program

```
(1,1), (1,2): integer   // declarations of x and y
begin
  Procedure (1,3)   // declaration of B
    (2,1), (2,2): real  // declaration of y and z
    begin
      (2,1) = (1,1) + (2,2) // occurrences of y, x, and z
      if (...) call (1,3) // occurrence of B
    end
  Procedure (1,4)   // declaration of C
    (2,1): real
    begin
      call (1,3) // occurrence of B
    end
  ... 
  call (1,4)   // occurrence of C
  call (1,3)   // occurrence of B
end
```
Lexical Scoping

Symbol table generated at compile time matches declarations and occurrences.
⇒ Each name can be represented as a pair (nesting_level, local_index).

Program
x, y: integer  // declarations of x and y
begin
  Procedure B    // declaration of B
    y, z: real  // declaration of y and z
    begin
      ...
      y = x + z // occurrences of y, x, and z
      if (...) call B // occurrence of B
    end
  Procedure C    // declaration of C
    x: real
    begin
      ...
      call B // occurrence of B
    end
  ...
  call C    // occurrence of C
  call B    // occurrence of B
end

Program
(1,1), (1,2): integer  // declarations of x and y
begin
  Procedure (1,3)    // declaration of B
    (2,1), (2,2): real  // declaration of y and z
    begin
      ...
      (2,1) = (1,1) + (2,2) // occurrences of y, x, and z
      if (...) call (1,3) // occurrence of B
    end
  Procedure (1,4)    // declaration of C
    (2,1): real
    begin
      ...
      call (1,3) // occurrence of B
    end
  ...
  call (1,4)    // occurrence of C
  call (1,3)    // occurrence of B
end
Lexical Scoping

Symbol table generated at compile time matches declarations and occurrences.
⇒ Each name can be represented as a pair (nesting_level, local_index).

Program

\[
\begin{align*}
x, y &: \text{integer} \\
\text{begin} & \\
\text{Procedure } B & \quad \text{declaration of } B \\
\quad y, z &: \text{real} \\
\quad \text{begin} & \\
\quad \quad y = x + z & \quad \text{occurrences of } y, x, \text{ and } z \\
\quad \quad \text{if } (...) \text{ call } B & \quad \text{occurrence of } B \\
\quad \text{end} & \\
\text{Procedure } C & \quad \text{declaration of } C \\
\quad x &: \text{real} \\
\quad \text{begin} & \\
\quad \quad \text{call } B & \quad \text{occurrence of } B \\
\quad \text{end} & \\
\end{align*}
\]

Program

\[
\begin{align*}
(1,1), (1,2) &: \text{integer} \\
\text{begin} & \\
\text{Procedure } (1,3) & \quad \text{declaration of } B \\
\quad (2,1), (2,2) &: \text{real} \\
\quad \text{begin} & \\
\quad \quad (2,1) &= (1,1) + (2,2) & \quad \text{occurrences of } y, x, \text{ and } z \\
\quad \quad \text{if } (...) \text{ call } (1,3) & \quad \text{occurrence of } B \\
\quad \text{end} & \\
\text{Procedure } (1,4) & \quad \text{declaration of } C \\
\quad (2,1) &: \text{real} \\
\quad \text{begin} & \\
\quad \quad \text{call } (1,3) & \quad \text{occurrence of } B \\
\quad \text{end} & \\
\end{align*}
\]

\[
\begin{align*}
\text{...} & \\
\text{call } (1,4) & \quad \text{occurrence of } C \\
\text{call } (1,3) & \quad \text{occurrence of } B \\
\text{end} & \\
\end{align*}
\]
Lexical Scoping

Symbol table generated at compile time matches declarations and occurrences. ⇒ Each name can be represented as a pair (nesting_level, local_index).

Program

```
x, y: integer    // declarations of x and y
begin
    Procedure B // declaration of B
        y, z: real  // declaration of y and z
        begin
            ...  // occurrences of y, x, and z
            y = x + z // occurrences of y, x, and z
            if (...) call B // occurrence of B
        end
    Procedure C // declaration of C
        x: real
        begin
            ...  // occurrences of y, x, and z
            call B // occurrence of B
        end
    ...  // occurrences of y, x, and z
    call C   // occurrence of C
    call B   // occurrence of B
end
```

Program

```
(1,1), (1,2): integer    // declarations of x and y
begin
    Procedure (1,3) // declaration of B
        (2,1), (2,2): real  // declaration of y and z
        begin
            ...  // occurrences of y, x, and z
            (2,1) = (1,1) + (2,2) // occurrences of y, x, and z
            if (...) call (1,3) // occurrence of B
        end
    Procedure (1,4) // declaration of C
        (2,1): real
        begin
            ...  // occurrences of y, x, and z
            call (1,3) // occurrence of B
        end
    ...  // occurrences of y, x, and z
    call (1,4)   // occurrence of C
    call (1,3)   // occurrence of B
end
```
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for this statement:

\[(2, 1) = (1, 1) + (2, 2)\]

What do we know?

- Assume the nesting level of the statement is **level 2**
- Register \( r_0 \) contains the current FP (frame pointer)
- \( (2, 1) \) and \( (2, 2) \) are local variables, so they are allocated in the activation record that current FP points to.
- \( (1, 1) \) is an non-local variable.
- Two new instructions:

  - **LOAD \( R_x, R_y \)** means \( R_x \leftarrow \text{MEM}(R_y) \)
  - **STORE \( R_x, R_y \)** means \( \text{MEM}(R_x) \leftarrow R_y \)
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

<table>
<thead>
<tr>
<th>Direction of stack growth (usually lower addresses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Pointer (FP)</td>
</tr>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>Return value</td>
</tr>
<tr>
<td>Return address</td>
</tr>
<tr>
<td>Access link</td>
</tr>
<tr>
<td>Caller FP</td>
</tr>
<tr>
<td>Local variables</td>
</tr>
</tbody>
</table>

(1,1): LOADI r1, #4  // offset of local variable (1,1) in frame
LOADI r2, #-4  // offset of access link in frame (bytes)
ADD r3 r0 r2  // address of access link
LOAD r4 r3  // get access link
ADD r5 r4 r1  // address of local variable (1,1) in frame
LOAD r6 r5  // get content of variable (1,1)
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

```
 (1,1)  LOADI r1, #4  // offset of local variable (1,1) in frame
            LOADI r2, #-4  // offset of access link in frame (bytes)
            ADD r3 r0 r2   // address of access link
            LOAD r4 r3     // get access link
            ADD r5 r4 r1    // address of local variable (1,1) in frame
            LOAD r6 r5     // get content of variable (1,1)
```
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

```
(1,1)  LOADI r1, #4  // offset of local variable (1,1) in frame
       LOADI r2, #-4  // offset of access link in frame (bytes)
       ADD r3 r0 r2  // address of access link
LOAD r4 r3  // get access link
ADD r5 r4 r1  // address of local variable (1,1) in frame
LOAD r6 r5  // get content of variable (1,1)
```
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

- \((1,1)\) LOADI r1, #4 // offset of local variable (1,1) in frame
- LOADI r2, #-4 // offset of access link in frame (bytes)
- ADD r3 r0 r2 // address of access link
- \(\text{LOAD} r4 r3\) // get access link
- ADD r5 r4 r1 // address of local variable (1,1) in frame
- LOAD r6 r5 // get content of variable (1,1)
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

<table>
<thead>
<tr>
<th>Direction of stack growth (usually lower addresses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Pointer FP</td>
</tr>
<tr>
<td>parameter</td>
</tr>
<tr>
<td>return value</td>
</tr>
<tr>
<td>return address</td>
</tr>
<tr>
<td>access link</td>
</tr>
<tr>
<td>caller FP</td>
</tr>
<tr>
<td>local variables</td>
</tr>
</tbody>
</table>

(1,1) LOADI r1, #4  // offset of local variable (1,1) in frame
LOADI r2, #-4  // offset of access link in frame (bytes)
ADD r3 r0 r2  // address of access link
LOAD r4 r3  // get access link
ADD r5 r4 r1  // address of local variable (1,1) in frame
LOAD r6 r5  // get content of variable (1,1)
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

```plaintext
(1,1)  LOADI r1, #4  // offset of local variable (1,1) in frame
LOADI r2, #-4  // offset of access link in frame (bytes)
ADD r3 r0 r2  // address of access link
LOAD r4 r3  // get access link
ADD r5 r4 r1  // address of local variable (1,1) in frame
LOAD r6 r5  // get content of variable (1,1)
```

Parameter

Return value

Return address

Access link

Caller FP

Local variables

Frame Pointer (FP)

Direction of stack growth (usually lower addresses)
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

<table>
<thead>
<tr>
<th>Direction of stack growth (usually lower addresses)</th>
<th>parameter</th>
<th>return value</th>
<th>return address</th>
<th>access link</th>
<th>caller FP</th>
<th>local variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Pointer FP</td>
<td>(1,1) LOADI r1, #4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOADI r2, #4-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADD r3 r0 r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOAD r4 r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADD r5 r4 r1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOAD r6 r5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2,2) LOADI r7 #8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADD r8 r0 r7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOAD r9 r8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

// offset of local variable (1,1) in frame
// offset of access link in frame (bytes)
// address of access link
// get access link
// address of local variable (1,1) in frame
// get content of variable (1,1)
// offset of local variable (2,2) in frame
// address of local variable (2,2)
// get content of variable (2, 2)
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

(1,1) \:
- `LOADI r1, #4` \hspace{0.5cm} \text{offset of local variable (1,1) in frame}
- `LOADI r2, #\text{-}4` \hspace{0.5cm} \text{offset of access link in frame (bytes)}
- `ADD r3 r0 r2` \hspace{0.5cm} \text{address of access link}
- `LOAD r4 r3` \hspace{0.5cm} \text{get access link}
- `ADD r5 r4 r1` \hspace{0.5cm} \text{address of local variable (1,1) in frame}
- `LOAD r6 r5` \hspace{0.5cm} \text{get content of variable (1,1)}

(2,2) \:
- `LOADI r7, #8` \hspace{0.5cm} \text{offset of local variable (2,2) in frame}
- `ADD r8 r0 r7` \hspace{0.5cm} \text{address of local variable (2,2)}
- `LOAD r9 r8` \hspace{0.5cm} \text{get content of variable (2,2)}
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

<table>
<thead>
<tr>
<th>Frame Pointer FP</th>
<th>Direction of stack growth (usually lower addresses)</th>
</tr>
</thead>
</table>
| parameter        | \begin{align*}(1,1) & \text{ LOADI r1, #4} & \text{// offset of local variable (1,1) in frame} \\
|                  | \text{ LOADI r2, #-4} & \text{// offset of access link in frame (bytes)} \\
|                  | \text{ ADD r3 r0 r2} & \text{// address of access link} \\
| return value     | \text{ LOAD r4 r3} & \text{// get access link} \\
| return address   | \text{ ADD r5 r4 r1} & \text{// address of local variable (1,1) in frame} \\
| access link      | \text{ LOAD r6 r5} & \text{// get content of variable (1,1)} \\
| caller FP        | \begin{align*}(2,2) & \text{ LOADI r7 #8} & \text{// offset of local variable (2,2) in frame} \\
|                  | \text{ ADD r8 r0 r7} & \text{// address of local variable (2,2)} \\
| local variables  | \text{ LOAD r9 r8} & \text{// get content of variable (2,2)} \end{align*} |
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

<table>
<thead>
<tr>
<th>Access link</th>
<th>local variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>(1,1) LOADI r1, #4 // offset of local variable (1,1) in frame</td>
</tr>
<tr>
<td></td>
<td>LOADI r2, #-4 // offset of access link in frame (bytes)</td>
</tr>
<tr>
<td></td>
<td>ADD r3 r0 r2 // address of access link</td>
</tr>
<tr>
<td></td>
<td>LOAD r4 r3 // get access link</td>
</tr>
<tr>
<td></td>
<td>ADD r5 r4 r1 // address of local variable (1,1) in frame</td>
</tr>
<tr>
<td></td>
<td>LOAD r6 r5 // get content of variable (1,1)</td>
</tr>
<tr>
<td></td>
<td>(2,2) LOADI r7 #8 // offset of local variable (2,2) in frame</td>
</tr>
<tr>
<td></td>
<td>ADD r8 r0 r7 // address of local variable (2,2)</td>
</tr>
<tr>
<td></td>
<td>LOAD r9 r8 // get content of variable (2,2)</td>
</tr>
</tbody>
</table>

\[\text{add} + \text{ADD r10 r6 r9} \] // (1,1) + (2,2)
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

\[
\begin{align*}
\text{parameter} & \quad \text{LOADI r1, #4} & \quad \text{(1,1) offset of local variable (1,1) in frame} \\
\text{return value} & \quad \text{LOADI r2, #-4} & \quad \text{(1,1) offset of access link in frame (bytes)} \\
\text{return address} & \quad \text{ADD r3 r0 r2} & \quad \text{address of access link} \\
\text{access link} & \quad \text{LOAD r4 r3} & \quad \text{get access link} \\
\text{caller FP} & \quad \text{ADD r5 r4 r1} & \quad \text{(1,1) address of local variable (1,1) in frame} \\
\text{local variables} & \quad \text{LOAD r6 r5} & \quad \text{(1,1) get content of variable (1,1)} \\
(2,2) & \quad \text{LOADI r7 #8} & \quad \text{(2,2) offset of local variable (2,2) in frame} \\
& \quad \text{ADD r8 r0 r7} & \quad \text{address of local variable (2,2)} \\
& \quad \text{LOAD r9 r8} & \quad \text{get content of variable (2,2)} \\
+ & \quad \text{ADD r10 r6 r9} & \quad \text{(1,1) + (2,2)} \\
(2,1) & \quad \text{LOADI r11 #4} & \quad \text{(2,1) offset of local variable (2,1) in frame} \\
& \quad \text{ADD r12 r0 r11} & \quad \text{(2,1) address of local variable (2,1)}
\end{align*}
\]
Access to Non-Local Data (Lexical Scoping)

What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,1)</td>
<td>LOADI r1, #4 // offset of local variable (1,1) in frame</td>
</tr>
<tr>
<td></td>
<td>LOADI r2, #-4 // offset of access link in frame (bytes)</td>
</tr>
<tr>
<td></td>
<td>ADD r3 r0 r2  // address of access link</td>
</tr>
<tr>
<td></td>
<td>LOAD r4 r3    // get access link</td>
</tr>
<tr>
<td></td>
<td>ADD r5 r4 r1  // address of local variable (1,1) in frame</td>
</tr>
<tr>
<td></td>
<td>LOAD r6 r5    // get content of variable (1,1)</td>
</tr>
<tr>
<td>(2,2)</td>
<td>LOADI r7 #8   // offset of local variable (2,2) in frame</td>
</tr>
<tr>
<td></td>
<td>ADD r8 r0 r7  // address of local variable (2,2)</td>
</tr>
<tr>
<td></td>
<td>LOAD r9 r8    // get content of variable (2, 2)</td>
</tr>
<tr>
<td>+</td>
<td>ADD r10 r6 r9 // (1,1) + (2,2)</td>
</tr>
<tr>
<td>(2,1)</td>
<td>LOADI r11 #4  // offset of local variable (2,1) in frame</td>
</tr>
<tr>
<td></td>
<td>ADD r12 r0 r11 // address of local variable (2,1)</td>
</tr>
</tbody>
</table>
What code do we need to generate for statement (*)?

\[(2,1) = (1,1) + (2,2)\]

<table>
<thead>
<tr>
<th>parameter</th>
<th>(1,1) LOADI r1, #4 (\text{offset of local variable (1,1) in frame})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LOADI r2, #-4 (\text{offset of access link in frame (bytes)})</td>
</tr>
<tr>
<td></td>
<td>ADD r3 r0 r2 (\text{address of access link})</td>
</tr>
<tr>
<td></td>
<td>LOAD r4 r3 (\text{get access link})</td>
</tr>
<tr>
<td></td>
<td>ADD r5 r4 r1 (\text{address of local variable (1,1) in frame})</td>
</tr>
<tr>
<td></td>
<td>LOAD r6 r5 (\text{get content of variable (1,1)})</td>
</tr>
<tr>
<td>(2,2)</td>
<td>LOADI r7 #8 (\text{offset of local variable (2,2) in frame})</td>
</tr>
<tr>
<td></td>
<td>ADD r8 r0 r7 (\text{address of local variable (2,2)})</td>
</tr>
<tr>
<td></td>
<td>LOAD r9 r8 (\text{get content of variable (2,2)})</td>
</tr>
<tr>
<td></td>
<td>ADD r10 r6 r9 (\text{(1,1) + (2,2)})</td>
</tr>
<tr>
<td>(2,1)</td>
<td>LOADI r11 #4 (\text{offset of local variable (2,1) in frame})</td>
</tr>
<tr>
<td></td>
<td>ADD r12 r0 r11 (\text{address of local variable (2,1)})</td>
</tr>
<tr>
<td></td>
<td>STORE r12 r10 (\text{(2,1) = (1,1) + (2,2)})</td>
</tr>
</tbody>
</table>
Things to do:

• Read ALSU, Chapter 7.1 - 7.3 (2nd Edition).