Figure 1.6, page 5 of text
Reminder:
Friday is a workshop day

- Come ready to work
- Ensure all team members are present
- Bring along a list of questions
What intermediate code do you come up with for this example?
As before, remember the structure...

Examples:

\[ f(x+1), \quad f(x+1,2*y) \]

\[ t1 = x + 1, \quad t2 = 2*y \]

\[ \text{param } t1, \quad \text{param } t2 \]

\[ \text{call}(f,1) \]

\[ f(g(3*z),h(a+b,a*b)) \]
examples

\begin{align*}
\text{t1} &= x + 1 \\
\text{param t1} \\
\text{call(f,1)}
\end{align*}

\begin{align*}
\text{f(x+1)} \\
\text{f(x+1,2*y)} \\
\text{t1} &= x + 1 \\
\text{param t1} \\
\text{t2} &= 2 * y \\
\text{param t2} \\
\text{call(f,2)}
\end{align*}

\begin{align*}
\text{f(g(3*z),h(a+b,a*b))} \\
\text{t1} &= 3 * z
\end{align*}

…but not just the top-level structure!
This translation will happen automatically due to the recursive structure of the function call for f...
examples

\[ f(x+1) \]
\[ \quad t_1 = x + 1 \quad \text{param } t_1 \quad \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ \quad t_1 = x + 1 \quad t_2 = 2 * y \quad \text{param } t_1 \quad \text{param } t_2 \quad \text{call}(f,2) \quad \text{call}(f,2) \]

\[ g(3*z) \]
\[ \quad t_1 = 3 * z \quad \text{param } t_1 \quad t_2 = \text{call}(g,1) \]

...view this as a function call in isolation.
examples

```
f(x+1)  f(x+1,2*y)
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ \text{call}(f,1) \]
\[ t2 = 2*y \]
\[ \text{param } t2 \]
\[ \text{call}(f,2) \]
\[ f(g(3*z),h(a+b,a*b)) \]
\[ t1 = 3*z \]
\[ \text{param } t1 \]
\[ t2 = \text{call}(g,1) \]
\[ \text{param } t2 \]
```

Mark the result as a parameter.
examples

\[ f(x+1) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t_1 = x + 1 \]
\[ t_2 = 2 * y \]
\[ \text{param } t_1 \text{ param } t_2 \]
\[ \text{call}(f,2) \]

\[ f(g(3*z),h(a+b,a*b)) \]
\[ t_1 = 3 * z \]
\[ t_2 = \text{call}(g,1) \]
\[ \text{param } t_1 \text{ param } t_2 \]
examples

\[ f(x+1) \]
param \( t1 = x + 1 \)
call(f,1)

\[ f(x+1,2*y) \]
param \( t1 = x + 1 \)
param \( t2 = 2 * y \)
call(f,2)

f\left( g\left( 3*z \right), h\left( a+b, a*b \right) \right) 
param \( t1 = 3 * z \)
param \( t1 = x + 1 \)
param \( t2 = 2 * y \)
param \( t2 = \text{call}(g,1) \)
param \( t2 = \text{call}(f,2) \)
param \( t2 \)
t3 = a + b

expression
examples

\[
\begin{align*}
\text{f}(x+1) & \quad \text{f}(x+1,2*y) \\
& \quad \text{t1} = x + 1 \\
& \quad \text{param } \text{t1} \\
& \quad \text{call(f,1)} \\
\text{t2} = 2 * y & \quad \text{t2} = 2 * y \\
& \quad \text{param } \text{t2} \\
& \quad \text{call(f,2)} \\
\text{f}(g(3*z),h(a+b,a*b)) & \\
& \quad \text{t1} = 3 * z \\
& \quad \text{param } \text{t1} \\
& \quad \text{t2} = \text{call(g,1)} \\
& \quad \text{param } \text{t2} \\
& \quad \text{t3} = a + b \\
& \quad \text{param } \text{t3}
\end{align*}
\]
examples

\[ f(x+1) \]
\[ f(x+1,2\times y) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ \text{call}(f,1) \]

\[ f(g(3\times z), h(a+b,a\times b)) \]
\[ t1 = 3 \times z \]
\[ \text{param } t1 \]
\[ t2 = \text{call}(g,1) \]
\[ \text{param } t2 \]
\[ t3 = a + b \]
\[ \text{param } t3 \]
\[ t4 = a \times b \]
examples

\[
\begin{align*}
\text{f(x+1)} & \quad \text{f(x+1,2*y)} \\
\text{t1 = x + 1} & \quad \text{t1 = x + 1} & \quad \text{t1 = x + 1} \\
\text{param t1} & \quad \text{param t1} & \quad \text{param t1} \\
\text{call(f,1)} & \quad \text{t2 = 2 * y} & \quad \text{param t2} \\
\text{param t2} & \quad \text{param t2} & \quad \text{call(f,2)} \\
\text{call(f,2)} & \quad \text{call(f,2)}
\end{align*}
\]

\[
\begin{align*}
\text{f(g(3*z),h(a+b,a*b))} \\
\text{t1 = 3 * z} & \quad \text{param t1} \\
\text{param t1} & \quad \text{t2 = call(g,1)} \\
\text{param t2} & \quad \text{param t2} \\
\text{t3 = a + b} & \quad \text{call(f,2)} \\
\text{param t3} & \quad \text{param t4} \\
\text{t4 = a * b} & \quad \text{call(h,2)}
\end{align*}
\]

parameter marking and call
examples

\[ f(x+1) \]
\[ f(x+1,2*y) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ \text{call}(f,1) \]

\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ t2 = 2 * y \]
\[ \text{param } t2 \]
\[ \text{call}(f,2) \]

\[ f(g(3*z),h(a+b,a*b)) \]
\[ t1 = 3 * z \]
\[ \text{param } t1 \]
\[ t2 = \text{call}(g,1) \]
\[ \text{param } t2 \]
\[ t3 = a + b \]
\[ \text{param } t3 \]
\[ t4 = a * b \]
\[ \text{param } t4 \]
\[ t5 = \text{call}(h,2) \]
\[ \text{param } t5 \]
\[ t6 = \text{call}(f,2) \]

parameter marking and call

© 2022 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
Examples

\[
f(x+1) \quad f(x+1,2y) \quad f(g(3z),h(a+b,a*b))
\]

\[
t1 = x + 1 \quad t1 = x + 1 \quad t1 = x + 1 \\
param t1 \quad param t1 \quad param t1 \\
call(f,1) \quad call(f,1) \quad call(f,2)
\]

Alternate translation gathering 'param' instructions together with call to function.
It may make sense to always save the value of a call to a temporary.
Memory Organization
Memory organization

code
static
heap
free memory
stack
Memory organization

- code
- static
- heap
- free memory
- stack

machine language instructions of the program

© 2022 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
Memory organization

- code
- static
- heap
- free memory
- stack

statically allocated memory (e.g. constants, string literals)
Memory organization

- code
- static
- heap
- free memory
- stack

Dynamically allocated memory (e.g., records, arrays)
Memory organization

- code
- static
- heap
- free memory
- stack

Heap grows towards stack
Memory organization

- Code
- Static
- Heap
- Free memory
- Stack

'Free memory' denotes the unallocated memory between heap and stack.
Memory organization

- code
- static
- heap
- free memory

- stack

- stack is used for function invocation records ("stack frames")
Memory organization

- Code
- Static
- Heap
- Free memory
- Stack

Stack grows towards heap
Memory organization

The size, layout and contents of both the code and static regions are determined at compile time.
Memory organization

- code
- static
- heap
- free memory
- stack

These regions are handled dynamically (i.e. at runtime)
Memory organization

- code
- static
- heap
- free memory
- stack

Heap allocation:
- reserve
- release
Memory organization

- code
- static
- heap
- free memory

Stack allocation: function call
# Stack frame organization

<table>
<thead>
<tr>
<th>Stack Frame Components</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>actual parameters</strong></td>
</tr>
<tr>
<td>(arguments)</td>
</tr>
<tr>
<td><strong>returned value</strong></td>
</tr>
<tr>
<td><strong>control link</strong></td>
</tr>
<tr>
<td>(dynamic link)</td>
</tr>
<tr>
<td><strong>access link</strong></td>
</tr>
<tr>
<td>(static link)</td>
</tr>
<tr>
<td><strong>saved machine status</strong></td>
</tr>
<tr>
<td>(return address)</td>
</tr>
<tr>
<td><strong>local data</strong></td>
</tr>
<tr>
<td><strong>temporaries</strong></td>
</tr>
</tbody>
</table>
Stack frame organization

- actual parameters (arguments)
- returned value
- control link (dynamic link)
- access link (static link)
- saved machine status (return address)
- local data
- temporaries

Initialized by caller, used by callee.
May be in CPU registers.
# Stack frame organization

<table>
<thead>
<tr>
<th>Stack Frame Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>actual parameters</code></td>
<td>(arguments)</td>
</tr>
<tr>
<td><code>returned value</code></td>
<td></td>
</tr>
<tr>
<td><code>control link</code></td>
<td>(dynamic link)</td>
</tr>
<tr>
<td><code>access link</code></td>
<td>(static link)</td>
</tr>
<tr>
<td><code>saved machine status</code></td>
<td>(return address)</td>
</tr>
<tr>
<td><code>local data</code></td>
<td></td>
</tr>
<tr>
<td><code>temporaries</code></td>
<td></td>
</tr>
</tbody>
</table>

- **Initialized by callee, read by caller.**
- **May be in a CPU register.**

© 2022 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
Stack frame organization

- **actual parameters** (arguments)
- **returned value**
- **control link** (dynamic link)
- **access link** (static link)
- **saved machine status** (return address)
- **local data**
- **temporaries**

The address of the caller’s invocation record (stack frame).
# Stack frame organization

<table>
<thead>
<tr>
<th>Stack Frame Components</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual parameters (arguments)</td>
<td>Used to achieve static scope for nested function definitions. Our language does not use this. Scheme/ML do.</td>
</tr>
<tr>
<td>returned value</td>
<td></td>
</tr>
<tr>
<td>control link (dynamic link)</td>
<td></td>
</tr>
<tr>
<td>access link (static link)</td>
<td></td>
</tr>
<tr>
<td>saved machine status (return address)</td>
<td></td>
</tr>
<tr>
<td>local data</td>
<td></td>
</tr>
<tr>
<td>temporaries</td>
<td></td>
</tr>
</tbody>
</table>
# Stack frame organization

<table>
<thead>
<tr>
<th>Stack Frame Components</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>actual parameters</strong> (arguments)</td>
</tr>
<tr>
<td><strong>returned value</strong></td>
</tr>
<tr>
<td><strong>control link</strong> (dynamic link)</td>
</tr>
<tr>
<td><strong>access link</strong> (static link)</td>
</tr>
<tr>
<td><strong>saved machine status</strong> (return address)</td>
</tr>
<tr>
<td><strong>local data</strong></td>
</tr>
<tr>
<td><strong>temporaries</strong></td>
</tr>
</tbody>
</table>

Information needed to restore machine to state at function call, including the return address (the value of the Program Counter at the time of the call).
### Stack frame organization

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>actual parameters</strong></td>
<td><strong>returned value</strong></td>
<td></td>
</tr>
<tr>
<td><em>(arguments)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>control link</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>(dynamic link)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>access link</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>(static link)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>saved machine status</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>(return address)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>local data</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>temporaries</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Space for local variables.**
Stack frame organization

- actual parameters (arguments)
- returned value
- control link (dynamic link)
- access link (static link)
- saved machine status (return address)
- local data
- temporaries

Space for temporary variables, and variable-length local data.

Temporaries may be in CPU registers.
7.2.3 Calling Sequence

What happens during a function call?
caller's invocation record

top_sp

top

Prior to function call.
7.2.3 Calling Sequence

"Procedure calls are implemented by what are known as calling sequences, which consist of code that allocates an activation record on the stack and enters information into its fields."

[p. 436]
During function call.
7.2.3 Calling Sequence

"A return sequence is similar code to restore the state of the machine so the calling procedure can continue its execution after the call."

[p. 436]
After function call.
"In general, if a procedure is called from \( n \) different points, then the portion of the calling sequence assigned to the caller is generated \( n \) times. However, the portion assigned to the callee is generated only once."

[p. 436]
Typical calling sequence [p. 437]

"1. The caller evaluates the actual parameters."

Recall:

formal parameter == parameter
actual parameter == argument
<table>
<thead>
<tr>
<th>caller's invocation record</th>
</tr>
</thead>
<tbody>
<tr>
<td>top_sp</td>
</tr>
<tr>
<td>top</td>
</tr>
</tbody>
</table>

![Diagram with red box showing:
- actual parameters
- returned value
- control link
- access link
- saved machine status
- local data
- temporaries

Prior to function call.](image)
caller's invocation record

top_sp

top

actual parameters
returned value
control link
access link
saved machine status
local data
temporaries
actual parameters

Caller writes arguments (actual parameters) past the end of its own invocation record.
Typical calling sequence [p. 437]

"2. The caller stores a return address and the old value of top_sp into the callee's activation record. The caller then increments top_sp [...] top_sp is moved past the caller's local data and temporaries and the callee's parameters and status fields."
caller's invocation record

top_sp

top

Caller knows the offset of the eventual returned value. When callee returns the caller will look at this location for the returned value.
Typical calling sequence [p. 437]

"2. The caller stores a return address and the old value of top_sp into the callee's activation record. ... "
The caller stores its stack pointer here.
The caller stores its stack pointer here. When the callee finishes the stack pointer's value will be reset to this value, thereby restoring the caller's invocation record as the active one (the one on top of the stack).
Typical calling sequence [p. 437]

"2. The caller stores a return address and the old value of top_sp into the callee's activation record. The caller then increments top_sp [...] top_sp is moved past the caller's local data and temporaries and the callee's parameters and status fields."
caller's invocation record

- actual parameters
- returned value
- control link
- access link
- saved machine status
- local data
- temporaries

- actual parameters
- returned value
- top_sp
- access link
- saved machine status
- ...
- ...

Move top_sp

top_sp
Typical calling sequence [p. 437]

"3. The callee saves the register values and other status information."
Write the return address, the current value of the Program Counter (PC), into the saved machine status. When the callee finishes execution will resume with the address pointed to by this saved address.
When control transfers to the callee, the top_sp and top are updated.

Callee writes local data and temporaries into its invocation record.
If the number of arguments can vary from call to call (e.g. printf) then the caller writes the arguments to the "actual parameters" area, as well as information about the number of arguments to the status area.
If the callee has variable length local data (e.g., local arrays whose size is determined by the value of a parameter) then the arrays are allocated space at the end of the invocation record, and pointers to those arrays are stored in the "locals" block.