Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
Exercise Review
exercise 1

How will you modify your grammar rules to generate intermediate code for switch statements?

Exercise 1: add the necessary actions to the switch productions from your grammar; here's my rule set for the switch statement - your formulation may be different. Where do the actions go, and what are the actions?

compound_statement :
  SWITCH L_PARENTHESIS expression R_PARENTHESIS case_list OTHERWISE COLON sblock

case_list :
  case_statement | case_statement case_list ;

case_statement :
  CASE constant COLON sblock ;

I won't show a solution here since this is part of your task for PR03, but I happy to answer questions/give advice during team meetings.
exercise 3

How will you modify your grammar rules to generate intermediate code for function calls?

Assume that type checking and argument list length checking has already been accounted for in the semantic actions attached to productions:

- type checking of each argument with corresponding parameter declaration (remembering that there is no coercion allowed in either an explicit or an implicit assignment)

- checking that the number of arguments and the number of parameters is the same
Basic approach teams took was to gather up information about argument expressions in an expression list, and generate the 'param' instructions at the end of the 'assignable ablock' rule, but only if assignable is a function (as opposed to an array). After the param instructions have been generated the 'call' instruction is generated, including the arity of the function (which is determined either by looking it up in the symbol table or by counting the number of arguments supplied).
exercise 2

What intermediate code do you come up with for this example?

\[ f(g(3z), h(a+b, a*b)) \]
As before, remember the structure...

\begin{align*}
\text{f}(x+1) & \quad \text{f}(x+1,2*y) \\
n & \quad n \\
\text{param } n & \quad \text{param } n \\
\text{call(f,1)} & \quad \text{call(f,2)} \\
\end{align*}
examples

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

...but not just the top-level structure!

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

© 2020 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
This translation will happen automatically due to the recursive structure of the function call for \( f \)...
examples

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

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

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

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

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

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

Mark the result as a parameter.
### examples

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x+1)$</td>
<td>$f(x+1,2*y)$</td>
<td>$f(g(3<em>z),h(a+b,a</em>b))$</td>
</tr>
<tr>
<td>$t1 = x + 1$</td>
<td>$t1 = x + 1$</td>
<td>$t1 = x + 1$</td>
</tr>
<tr>
<td>param $t1$</td>
<td>param $t1$</td>
<td>param $t1$</td>
</tr>
<tr>
<td>call($f,1$)</td>
<td>$t2 = 2 * y$</td>
<td>param $t2$</td>
</tr>
<tr>
<td></td>
<td>param $t2$</td>
<td>param $t2$</td>
</tr>
<tr>
<td></td>
<td>call($f,2$)</td>
<td>call($f,2$)</td>
</tr>
</tbody>
</table>

More structure!
examples

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

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

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

expression
examples

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

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

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

\[
f(x+1)
\]

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

\[
t_1 = x + 1
\]

param \ t_1

\[
call(f, 1)
\]

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

\[
t_1 = 3 \cdot z
\]

param \ t_1

\[
t_2 = \text{call}(g, 1)
\]

param \ t_2

\[
t_3 = a + b
\]

param \ t_3

\[
t_4 = a \cdot b
\]

expression

© 2020 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
examples

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

\[ f(g(3*z),h(a+b,a*b)) \]
\[ t_1 = 3 * z \]  param \ t_1  t_2 = \text{call}(g,1) \]
\[ t_2 = 2 * y \]  param \ t_2  \]
\[ t_3 = a + b \]  param \ t_3  \]
\[ t_4 = a * b \]  param \ t_4  \]
\[ t_5 = \text{call}(h,2) \]

parameter marking and call
### Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Parameter Marking and Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x+1)$</td>
<td>$t_1 = x + 1$ param $t_1$ call($f$,1)</td>
</tr>
<tr>
<td>$f(x+1,2*y)$</td>
<td>$t_1 = x + 1$ param $t_1$ $t_2 = 2 * y$ param $t_2$ call($f$,2)</td>
</tr>
<tr>
<td>$f(g(3<em>z),h(a+b,a</em>b))$</td>
<td>$t_1 = 3 * z$ param $t_1$ $t_2 = \text{call}(g,1)$ param $t_2$ $t_3 = a + b$ param $t_3$ $t_4 = a * b$ param $t_4$ $t_5 = \text{call}(h,2)$ param $t_5$ call($f$,2)</td>
</tr>
</tbody>
</table>
Examples:

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

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

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

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

<table>
<thead>
<tr>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
</tr>
<tr>
<td>Heap</td>
</tr>
<tr>
<td>Free memory</td>
</tr>
<tr>
<td>Stack</td>
</tr>
</tbody>
</table>

- **Machine language**
- **Instructions of the program**
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 is used for function invocation records ("stack frames")
Memory organization

```
free memory

stack grows towards heap
```

- code
- static
- heap
- stack

© 2020 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
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

© 2020 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
# Stack frame organization

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

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

- **Initialized by caller, used by callee.**
- **May be in CPU registers.**

© 2020 Carl Alphonce - Reproduction of this material is prohibited without the author’s consent
## Stack frame organization

<table>
<thead>
<tr>
<th>Stack Frame Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>actual parameters</strong></td>
<td>(arguments) initialized by callee, read by caller.</td>
</tr>
<tr>
<td><strong>returned value</strong></td>
<td></td>
</tr>
<tr>
<td><strong>control link</strong></td>
<td>(dynamic link)</td>
</tr>
<tr>
<td><strong>access link</strong></td>
<td>(static link)</td>
</tr>
<tr>
<td><strong>saved machine status</strong></td>
<td>(return address)</td>
</tr>
<tr>
<td><strong>local data</strong></td>
<td></td>
</tr>
<tr>
<td><strong>temporaries</strong></td>
<td></td>
</tr>
</tbody>
</table>

- **Initialized by callee, read by caller.**
- **May be in a CPU register.**
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 Organization</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>

- **Used to achieve static scope for nested function definitions.**
- **Our language does not use this.**
- **Scheme/ML do.**
### Stack frame organization

<table>
<thead>
<tr>
<th><strong>actual parameters</strong> (arguments)</th>
</tr>
</thead>
<tbody>
<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

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