Processing control structures

Statement lists

- So far we have discussed generating code for one assignment statement
- Generating code for multiple statements is easy

\[
\text{stmt_list} \rightarrow \text{stmt} \text{ stmt_list} | \lambda
\]

- Keep appending (or prepending) the code generated by a single statement to the code generated by the rest of the statement list
- What if statement is not an assignment?

If statements

\[
\text{if} \ \langle \text{bool_expr_1} \rangle \ \text{then} \\
\langle \text{stmt_list_1} \rangle \\
\text{elseif} \ \langle \text{bool_expr_2} \rangle \ \text{then} \\
\langle \text{stmt_list_2} \rangle \\
\text{else} \\
\langle \text{stmt_list_3} \rangle \\
\text{endif}
\]

Generating code for ifs

```
if <bool_expr_1> then
    <stmt_list_1>
else if <bool_expr_2> then
    <stmt_list_2>
else
    <stmt_list_3>
endif
```

Notes on code generation

- The \langle op \rangle in \langle op \rangle is dependent on the type of comparison you are doing in <bool_expr>
- When you generate JUMP instructions, you should also generate the appropriate LABELs
- But you may not put the LABEL into the code immediately
  - e.g., the OUT label (when should you create this? When should you put this in code?)
- Instead, generate the labels when you first process the if statement (i.e., before you process the children) so that it's available when necessary
- Remember: labels have to be unique!

Create the out label when you process the beginning of an if statement
Put it in code when you are done processing the if statement
Processing Loops

While loops

while <bool_expr> do
  <stmt_list>
end

---

Generating code for do-while loops

do
  <stmt_list>
while <bool_expr>;
  LOOP:
    <stmt_list>
    <bool_expr> J<op> LOOP
    OUT:
---

For loops

for (<init_stmt>;<bool_expr>;<incr_stmt>)
  <stmt_list>
end

---

Generating code: for loops

for (<init_stmt>;<bool_expr>;<incr_stmt>)
  <stmt_list>
end

---

continue and break statements

for (<init_stmt>;<bool_expr>;<incr_stmt>)
  <stmt_list>
end

---
Switch statements

```
switch (<expr>)
  case <const_list>: <stmt_list>
  case <const_list>: <stmt_list>
  ...
  default: <stmt_list>
end
```

### Deciding where to jump

- **Problem:** do not know which label to jump to until switch expression is evaluated
- Use a jump table: an array indexed by case values, contains address to jump to
  - If table is not full (i.e., some possible values are skipped), can point to a default clause
  - If default clause does not exist, this can point to error code
- **Problems**
  - If table is sparse, wastes a lot of space
  - If many choices, table will be very large

### Jump table example

Consider the code:

```
Case x is
(0010) When 0: stmts0
(0017) When 1: stmts1
(0192) When 2: stmts2
(0198) When 3: stmts3
(1000) When 5: stmts5
(1050) Else stmts5
```

**Jump table has 6 entries:**

<table>
<thead>
<tr>
<th>Case</th>
<th>Jump to</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>JUMP 0010</td>
</tr>
<tr>
<td>1</td>
<td>JUMP 0017</td>
</tr>
<tr>
<td>2</td>
<td>JUMP 0192</td>
</tr>
<tr>
<td>3</td>
<td>JUMP 0198</td>
</tr>
<tr>
<td>4</td>
<td>JUMP 1050</td>
</tr>
<tr>
<td>986</td>
<td>JUMP 1050</td>
</tr>
<tr>
<td>987</td>
<td>JUMP 1000</td>
</tr>
</tbody>
</table>

Table only has 983 unnecessary rows.
Doesn’t appear to be the right thing to do!

**Note:** table size is proportional to range of choice clauses, not number of clauses!

### Do a binary search

Consider the code:

```
Case x is
(0010) When 0: stmts0
(0017) When 1: stmts1
(0192) When 2: stmts2
(0198) When 3: stmts3
(1000) When 5: stmts5
(1050) Else stmts5
```

**Jump table has 6 entries:**

<table>
<thead>
<tr>
<th>Case</th>
<th>Jump to</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>JUMP 0010</td>
</tr>
<tr>
<td>1</td>
<td>JUMP 0017</td>
</tr>
<tr>
<td>2</td>
<td>JUMP 0192</td>
</tr>
<tr>
<td>3</td>
<td>JUMP 0198</td>
</tr>
<tr>
<td>986</td>
<td>JUMP 1050</td>
</tr>
<tr>
<td>987</td>
<td>JUMP 1000</td>
</tr>
</tbody>
</table>

Perform a binary search on the table. If the entry is found, then jump to that offset.
If the entry isn’t found, jump to others clause. **O(log n)** time, n is the size of the table, for each jump.
Linear search example

Consider the code:

If there are a small number of choices, then do an in-line linear search. A straightforward way to do this is generate code analogous to an IF THEN ELSE.

Case x is
0: stmts
1: stmts
2: stmts
others: stmts;

\( O(n) \) time, \( n \) is the size of the table, for each jump.

Dealing with jump tables

switch (<expr>)
  case <const_list>: <stmt_list>
  case <const_list>: <stmt_list>
  ...
  default: <stmt_list>
end

• Generate labels, code, then build jump table
• Put jump table after generated code
• Why do we need the OUT label?
• In case of break statements