Switch Statements
Switch statements

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

- Generated code for `<expr>` then check all the cases to see which matches the result

- Key issues:
  - Where to jump?
  - Multiple cases lead to the same code
  - Many different cases --- potentially dozens or hundreds

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

- 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

```c
switch (<expr>)
  case <const_list>: <stmt_list>
  case <const_list>: <stmt_list>
  ...  
  default: <stmt_list>
end
```
Jump table example

Consider the code:
((xxxx) is address of code)

Case x is
(0010) When 0: stmts
(0017) When 1: stmts
(0192) When 2: stmts
(0198) When 3 stmts;
(1000) When 5 stmts;
(1050) Else stmts;

Jump table has 6 entries:

<table>
<thead>
<tr>
<th></th>
<th>JUMP 0010</th>
<th>JUMP 0017</th>
<th>JUMP 0192</th>
<th>JUMP 0198</th>
<th>JUMP 1050</th>
<th>JUMP 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table only has one Unnecessary row (for choice 4)
**Jump table example**

Consider the code:

((xxxx) Is address of code)

Case x is

(0010) When 0: stmts0
(0017) When 1: stmts1
(0192) When 2: stmts2
(0198) When 3 stmts3
(1000) When 987 stmts4
(1050) When others stmts5

<table>
<thead>
<tr>
<th>Address</th>
<th>Case</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>...</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 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:
((xxxx) Is address of code)

Case x is
(0010) When 0: stmts0
(0017) When 1: stmts1
(0192) When 2: stmts2
(0198) When 3: stmts3
(1000) When 987: stmts4
(1050) When others: stmts5

Jump table has 5 entries:

<table>
<thead>
<tr>
<th>Jump Target</th>
<th>Address</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>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.
Consider the code:
((xxxx) Is address of code)

Case x is
(0010) When 0: stmts1
(0017) When 1: stmts2
(0192) When 2: stmts3
(1050) When others stmts4

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.

If (x == 0) then stmts1;
Elseif (x = 1) then stmts2;
Elseif (x = 2) then stmts3;
Else stmts4;

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

<expr>
<code for jump table>
LABEL0:
  <stmt_list>
LABEL1:
  <stmt_list>
...
DEFAULT:
  <stmt_list>
OUT:

• 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