If statements

\[
\text{if } <\text{bool_expr}_1> \text{ then } \\
<\text{stmt_list}_1> \text{ elseif } <\text{bool_expr}_2> \text{ then } \\
<\text{stmt_list}_2> \ldots \text{ else } \\
<\text{stmt_list}_3> \text{ endif}
\]

Notes on code generation

- The \texttt{<op>} in \texttt{j<!op>} is dependent on the type of comparison you are doing in \texttt{<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!

Directly generating binary code

- Recall difference between assembly code and machine code
- Assembly code must be processed by assembler; machine code directly executable
- One job of assembler: decide actual addresses to jump to instead of labels
- So what happens if we generate binary directly?
- Need to insert JMP instructions before knowing where the label will be
- Solution: backpatching
  - Store offset of JMP instruction in semantic record
  - When label is created, access JMP instruction and "patch up" jump target
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:

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

- Continue statements: skip past rest of block, perform incr_stmt and restart loop
- Break statements: jump out of loop (do not execute incr_stmt)
- Caveats:
  - Code for stmt_list is generated earlier—where do we jump?
  - Keep track of “loop depth” as you descend through AST
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 stmts
(1000) When 5 stmts;
(1050) Else stmts;

Jump table has 6 entries:

Table only has one
Unnecessary row
(for choice 4)

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 stmts
(1000) When 5 stmts;
(1050) Else stmts;

Jump table has 6 entries:

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:

\[
\text{(xxxx) is offset of local Code start from the Jump instruction}
\]

Case x is

- (0010) When 0: stmts
- (0017) When 1: stmts
- (0192) When 2: stmts
- (1050) When others stmts;

\[O(n) \text{ time, } n \text{ is the size of the table, for each jump.}\]

Dealing with jump tables

switch (\texttt{<expr>})

\begin{align*}
\text{case } & \texttt{<const_list>: <stmt_list>} \\
& \text{case } \texttt{<const_list>: <stmt_list>} \\
& \quad \ldots \\
& \text{default: <stmt_list>}
\end{align*}

\text{end}

\texttt{<expr>}

\begin{align*}
\text{<code for jump table>}
\text{LABEL0:}
\text{<stmt_list>}
\text{LABEL1:}
\text{<stmt_list>}
\quad \ldots
\text{DEFAULT:}
\text{<stmt_list>}
\text{OUT:}
\end{align*}

- 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

Case statements

- As in LITTLE
- What makes them different from switch statements?
  - Arbitrary expressions in each CASE
- How should you generate code for this?