Statement-Level Control

- Three types:
  - sequencing (not much to be said about this)
  - selection
  - repetition

Selection

- Issues:
  - How is selection controlled?
  - How many choices? Can none of the choices be selected

- Single-way selectors

  ![Flowchart of single-way selector]

  if <bool> then <stmt>

  - Without compound statements, need GOTO
Two-Way Selection

- Two-way selectors

Dangling Elses

- Recall potential ambiguity with nested 2-way selectors:
  
  \[
  \text{if } b_1 \text{ then if } b_2 \text{ then } s_1 \text{ else } s_2
  \]

  - Several possible remedies...
Avoiding Ambiguity

- **Disallow nested conditionals**

  \[
  \begin{align*}
  \langle \text{stmt} \rangle & \rightarrow \langle \text{a_stmt} \rangle \mid \langle \text{if_stmt} \rangle \mid \langle \text{c_stmt} \rangle \\
  \langle \text{c_stmt} \rangle & \rightarrow \text{begin} \ \langle \text{stmt_list} \rangle \ \text{end} \\
  \langle \text{if_stmt} \rangle & \rightarrow \text{if} \ \langle \text{bool} \rangle \ \text{then} \ \langle \text{s_stmt} \rangle \\
  & \quad \mid \ \text{if} \ \langle \text{bool} \rangle \ \text{then} \ \langle \text{s_stmt} \rangle \ \text{else} \ \langle \text{stmt} \rangle \\
  \langle \text{s_stmt} \rangle & \rightarrow \langle \text{a_stmt} \rangle \mid \langle \text{c_stmt} \rangle
  \end{align*}
  \]

  if b1 then
  
  begin if b2 then s1 else s2 end

  if b1 then
  
  begin if b2 then s1 end
  
  else s2

Multiple-Way Selection

- **Multiple-way Selection**

  - **New issues:**
    
    How to handle unrepresented selector values?
    Should selectable segments be followed by implicit branches out of construct?

  - **Lots of forms of multiple selectors . . .**
Multiple-Way Selection -- Early Approaches

- Three-way selectors (Fortran)
  
  ```fortran
  IF (<arith_exp>) L1, L2, L3
  IF (...) 10, 20, 30
  10 ... GO TO 40
  20 ... GO TO 40
  30 ... 40 ...
  ```

- Computed GO TO (Fortran)
  
  ```fortran
  GO TO (L1, L2, ..., Ln), exp
  ```

  - goes to first label if exp=1, second if exp=2, etc.
  - if exp < 1 or exp > n, no effect
  - must still jump out of segment

Multiple-Way Selection -- Modern Approaches

- Pascal example
  
  ```pascal
  case <exp> of
    <const_list> : <stmt>
    ...
    <const_list> : <stmt>
  end
  ```

  - <exp> of ordinal type
  - implicit branch to end after each segment
  - "otherwise" option
  - selector value unrepresented => error
More Modern Approaches

- **C example**
  
  ```c
  switch (<exp>)
  {
    case <const_exp> : <stmt>
    ...
    case <const_exp> : <stmt>
    [default : <stmt>]
  }
  
  - <exp> and <const_exp> yield integers
  - No implicit branches -- use "break" to leave switch
    Is this a good idea?
  
- **Elsif (Ada)**
  
  - Like LISP cond
  - Cuts down on indenting
  - More flexible than case statement

Repetition

- **Iteration (statement-level)**

- **Recursion (unit-level) -- later**

- **Issues:**
  
  - How is iteration controlled?
  - Where in loop is control mechanism? 
    top
    bottom
    anywhere
Counter-Controlled Loops

- Issues:
  - Type and scope of loop variable
  - Value of loop variable at loop term
  - Change loop parameters in body? Effect?
  - Branch into loop?
  - Test at top or bottom?
  - Loop parameters evaluated when?

FORTRAN loops

- DO <label> <var> = <init>,<term>
  - Can’t change loop variable/parameters inside loop
  - Loop variable undefined after normal termination; last value if jump out
  - Can jump out of, back into loop
    DO 100 I = 1,10
    ...  
    100 CONTINUE
  - Note!
    DO 100 I = 1.10
More Loop Examples

- **ALGOL 60**
  - Very complicated! see book.

- **ALGOL 68**
  
  ```
  for i from j by k to m
  while b do . . . od
  ```

- **C**
  
  ```
  for (<exp>; <exp>; <exp>) <stmt>
   \[\begin{array}{c}
   \text{init} \\
   \text{term} \\
   \text{incr}
   \end{array}\]
  
  for (i = 0; i <= 10; i++)
  sum = sum + a[i];
  ```
  - No special loop variables.
  - Can have multiple statements for each <exp>

Logically Controlled Loops

- **Issues:**
  - Pretest or posttest?
  - Jump into loop?

- **Pascal constructs**
  
  ```
  while <exp> do <stmt>
  repeat <stmt> until <exp>
  ```
More Statement-Level Control Constructs

- **Explicit loop exits**
  - Basic model: (Modula)
    ```
    loop
    ...
    if ... then EXIT
    ...
    end
    ```

- **Multi-level exits (Ada)**
  - named loops
  - conditional EXIT
    ```
    exit [<loop name>] [when <condition>]
    ```

Ada Example

```ada
Outer_loop:
  for i in 1..10 loop
    Inner_loop:
      for j in 1..20 loop;
      ...
      EXIT Outer_loop when <cond>;
      ...
    end loop Inner_loop;
    ...
  end loop Outer_loop;
```
Iterators (Clu)

- Lets user specify range of values over which a loop iterates.
  
  ```
  for atom:node in list(x) do
    ...
    list = iter(z: linked_list) yields (node)
    ...
    yield (n)
    ...
  end list
  ```

  - list produces elements of type node one at a time

Guarded Commands

```latex
\textbf{Guarded Commands}

\textbf{Semantics}

- Evaluate all $\langle \text{bool} \rangle$-s
- If $\geq 1$ is true, choose one nondeterministically and execute its $\langle \text{stmt} \rangle$
- If none true, error
```

```latex
\begin{align*}
\text{if} & \quad \langle \text{bool} \rangle \rightarrow \langle \text{stmt} \rangle \\
\text{...} & \quad \langle \text{bool} \rangle \rightarrow \langle \text{stmt} \rangle \\
\text{fi}
\end{align*}
```
Guarded Commands (continued)

```
do <bool> -> <stmt>  
  <bool> -> <stmt>  
  ...  
  <bool> -> <stmt>  
  od
```

- **Semantics**
  - Evaluate all `<bool>`s
  - If $\geq 1$ is true, choose one nondeterministically and execute its `<stmt>`
  - Repeat until none true, then normal termination