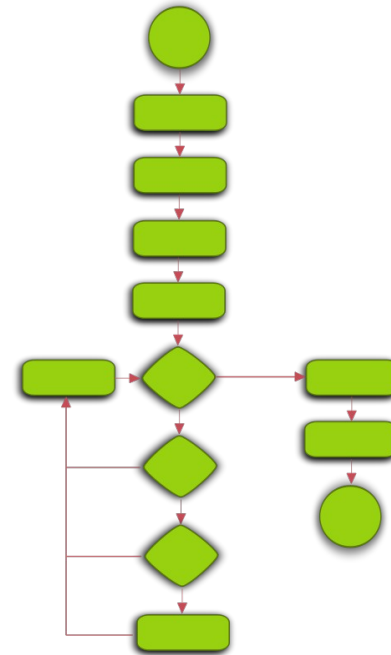# CS 430
Spring 2019

Mike Lam, Professor

Control Structures

Control Structures

- **Control flow path**: sequence order of executed instructions
- **Control structure**: control statement and its associated flow path
- **Selection** statements (e.g., `if/then/else`, `switch/case`)
  - Choose between alternative control flow paths
- **Iteration** statements (e.g., `do`, `while`, `for`, `until`)
  - Repeatedly execute a control flow path
- How many kinds of control statements?
  - Many: higher expressivity
  - Few: higher readability, learnability, and orthogonality

Selection Structures

- **Two-way selection** (`if/then`)
  - Inclusion of "else" clause
  - Blocks often delimited by braces, keywords (e.g., "begin", "end") or indentation
  - Nesting issues
- **Multiple selection** (`switch/case`)
  - Form ("`if/elseif/else`" vs. "`switch/case`")
  - Case value types
  - Multiple execution
  - Fallthrough
  - Default values
  - Efficient implementation using jump tables

Iteration Structures

- Control form: logic vs. counter vs. user-controlled vs. iterator-based
  - Counter loop parameters: loop variables, initial/terminal values, step sizes
  - Counter variable in scope outside loop? (no, starting with Ada)
- Control location: pre-test vs. post-test vs. user-defined
- Examples:
  - While loop: logic pre-test
  - Do-while loop: logic post-test
  - For loop: counter pre-test
- Functional languages: recursion instead of iteration

Language Design

- Can iteration structures have multiple entries?
  - General answer: no!
  - Increase in flexibility/expressiveness is small relative to decrease in readability
- Can iteration structures have multiple exits?
  - For most procedural languages: yes
  - Same as "should `goto` or `break` be included?"

Minimally-Sufficiency Constructs

- Böhm and Jacopini (1966)
  - "Structured program theorem"
  - Strictly necessary: 1) sequencing, 2) two-way logical selection, and 3) logical iteration
    - Can implement ALL flowchart-representable programs
    - Alternatively: a selectable `goto` statement
      - E.g., "if (E) goto L1" goto code from CS 261!
  - Facilitates automated translation of block-structured code
    - Use "templates" to guide translation

Minimally-Sufficient Constructs

if statement: if (E) B1

```
        << E code >>
        if E goto l1
        goto l2
    l1:
        << B1 code >>
    l2:
```

Minimally-Sufficient Constructs

if statement: if (E) B1 else B2

```
        << E code >>
        if E goto l1
        goto l2
    l1:
        << B1 code >>
        goto l3
    l2:
        << B2 code >>
    l3:
```

Minimally-Sufficient Constructs

while loop: while (E) B

```
    l1:                             ; CONTINUE target

        << E code >>

        if E goto l2

        goto l3

    l2:

        << B code >>

        goto l1

    l3:                             ; BREAK target
```

Minimally-Sufficient Constructs

for loop: for V in E1, E2 B

```
        << E1 code >>
        << E2 code >>
        V = E1
    l1:
        if (V >= E2) goto l2
        << B code >>
        V = V + 1                    ; CONTINUE target
        goto l1
    l2:                              ; BREAK target
```

Minimally-Sufficient Constructs

- Use only the following constructs:
  - S → S; S
  - S → **if** (E) { S } **else** { S }
  - S → **while** (E) { S }
  - S → *<assignment statement>*
  - E → *<boolean expression>*

- Rewrite the following Ruby code:

```
until a >= b
  a += 5
end
```

```
1.upto(10) do |i|
  y = y + i
end
```

```
3.times do
  x = x * 2
end
```

```
case (n % 3)
when 0
  d = 1
when 1
  d = 2
when 2
  d = 3
end
```

```
if x > 90 then
  g = 'A'
elsif x > 80 then
  g = 'B'
elsif x > 70 then
  g = 'C'
else
  g = 'D'
end
```

Greatest Argument in PL History

- "Should languages provide a `goto` statement?"
  - Pro: extremely powerful construct – high expressiveness and writability
  - Against: without restrictions, can make programs very difficult to understand – low readability and maintainability
- Classic 1968 CACM letter by Edsger Dijkstra: "*Go To Statement Considered Harmful*"
  - Widely misunderstood
  - Original title: "A Case Against the Goto Statement"
  - Criticized **excessive** use of `goto`
  - Consensus: structured control flow is safer
    - Use control structures, exceptions, or tail recursion instead
    - Only C descendants tend to have `goto` statements these days

Guarded Commands

- Dijkstra (1975): guarded selection and iteration statements: `if/fi` and `do/od`
  - More than one boolean condition may be true
  - Control flow path is chosen **non-deterministically** out of the available true conditions
  - Pro: sometimes more elegant and easily proven correct
  - Con: greatly-increased complexity and lowered readability

Guarded Commands

- Maximum of (x,y):

  - `if   x >= y   →   max := x`
  - `[]   y >= x   →   max := y`
  - `fi`

- Sorting four integers (q1, q2, q3, q4):

  - `do   q1 > q2   →   temp := q1; q1 = q2; q2 := temp;`
  - `[]   q2 > q3   →   temp := q2; q2 = q3; q3 := temp;`
  - `[]   q3 > q4   →   temp := q3; q3 = q4; q4 := temp;`
  - `od`