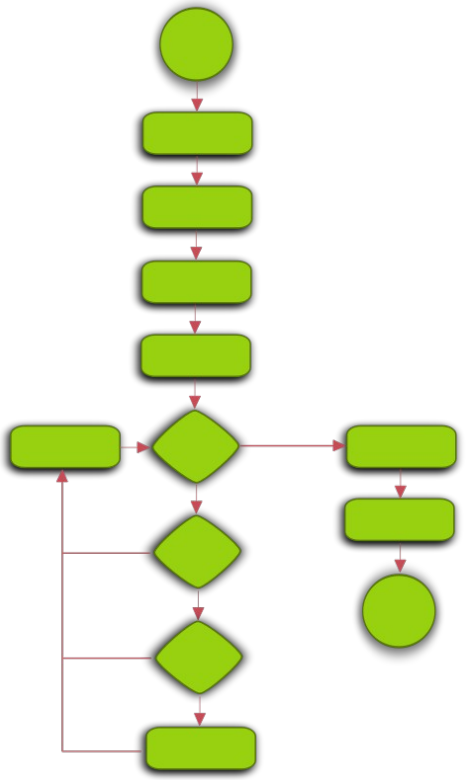
CS 430 Spring 2015

Mike Lam, Professor

```
x = a+b+c

y = sin(x)

E = m*(c**2)
```



Expressions and Control Structures

Warm-up Exercises

- What are the values of a, b, and c after each Java statement?

```
1) a = 2*-3;
2) b = 4 + 3 * 2;
3) c = b-3 * a;
4) a = 3;
5) b -= 5 - a++;
6) c = a+++4-(--b);
```

Expressions

- Expression: specification of computation
  - Form/syntax expressed using BNF grammars
  - Operations
  - Operands
  - Parentheses
  - Function calls

Expressions

- Operators
  - Unary vs. binary vs. ternary
  - Infix vs. prefix vs. postfix
  - Precedence
  - Associativity (left or right)
  - Overloading
  - Arithmetic vs. boolean
  - Short-circuit boolean operators

Expressions

- Operands
  - Evaluation order
  - Type conversions
    - Implicit vs. explicit
    - Narrowing vs. widening
  - Errors
    - Overflow and underflow
    - Division by zero
    - Floating-point issues (e.g., NaN, subnormal)

Expressions

- Parentheses
  - Explicit precedence and associativity
  - Tuple creation
- Function calls
  - Side effects and referential transparency

Assignment Statements

- Symbol and ambiguity with equality operator
  - "=" vs. ":=" vs. "==" vs. "←"
  - Assignments as expressions; good idea?
- Simple assignments
- Conditional targets (ternary LHS)
  - `(n > 5 ? a : b) = n*2`
- Compound assignments
  - Shortened forms of an assignment: "+=" and "++"
- Multiple assignments
  - `a,b = c/2,c%2` `a,b = b,a`

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

Minimally-Sufficiency Constructs

- Böhm and Jacopini (1966)
  - “Structured program theorem”
  - 1) Sequencing, 2) two-way logical selection, and 3) logical iteration
  - Can implement ALL flowchart-representable programs
  - Alternatively: a selectable `goto` statement

Selection Structures

- Two-way selection (`if/then`)
  - Inclusion of "else" clause
  - Blocks 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

- Basic questions:
  - How the iteration is controlled: logic vs. counter
    - Counter loop parameters: loop variables, initial/terminal values, step sizes
    - Counter variable in scope outside loop? (no, starting with Ada)
  - Where the control mechanism appears in the loop statement: pre vs. post vs. user-defined
- Examples:
  - While loop: logic pre-test
  - Until loop: logic post-test
  - For loop: counter
  - Iterator-based loops: variant of counters
- Functional languages: recursion instead of iteration

Minimally-Sufficient Constructs

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

- Rewrite the following Ruby code:

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

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

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

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:
        t1 = V >= E2
        if t1 goto l2
        << B code >>
        V = V + 1                   ; CONTINUE target
        goto l1
    l2:                             ; BREAK target
```

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`

Language Design

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

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: some constructs are 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`