Assignment Statements in Programming Languages

A Programming Language – Universal: All Solvable Computations

- integer values and arithmetic operators (arithmetic expressions)
- variables
- assignment statement
- selection statement
- loop statement/go to statement

Assignment Statements

Simple Assignment Statements

- Simple Assignments
  - `<target_variable> <assignment_operator> <exp>`
- The assignment operator symbol:
  - `:=` ALGOLs, Pascal, Modula-2, Ada
  - `=` FORTRAN, BASIC, PL/I, C, C++, Java
    - = can be bad if it is overloaded for the relational operator for equality.

Assignment Statements

- Multiple targets
  - `A, B = 10`
- Conditional targets
  - C, C++, and Java
  - `(first = true) ? total : subtotal = 0`

Compound Assignment Statements

- Compound assignment operators:
  - The target variable (LHS) is the first operand in the expression (RHS).
  - Abbreviated assignment
  - C, C++, and Java
  - `sum = sum + next;`
  - `sum += next;`

Unary Assignment Statements

- Unary assignment operators:
  - Abbreviated assignment combined with increment and decrement operators.
  - Prefix or postfix
  - C, C++, and Java

```
sum = ++ count;

count = count + 1;
sum = count;
```

```
sum = count ++;

sum = count;
count = count + 1;
```


Assignment as an Expression

- The assignment statement produces a result, which is the same as the value assigned to the target variable.
  - C, C++, and Java
- Can be used as an operand in an expression.
  - `while ((ch = getchar()) != EOF) { ... }`
- Disadvantage:
  - Another kind of expression side effect.

Mixed-Mode Assignment

- The types of LHS and RHS of an assignment statement are different.
- Design issues:
  - Does the type of the expression have to be the same as the type of the variable being assigned?
  - Can type coercion be used?

Mixed-Mode Assignment

- FORTRAN, C, and C++:
  - Use coercion rules.
- Pascal:
  - integers can be assigned to reals, but reals cannot be assigned to integers.
- Java:
  - Only widening assignment coercions are done.
- Ada:
  - No assignment coercion.

Statement-Level Control Structures in Programming Languages

A Programming Language – Universal: All Solvable Computations

- integer values and arithmetic operators (arithmetic expressions)
- variables
- assignment statement
- selection statement
- loop statement/go to statement

Levels of Control Flow

- Flow of control (execution sequence) in a program:
  - Within expressions
  - Among statements
  - Among program units

Compound Statements as Blocks

- A collection of statements:
  - Introduced by ALGOL 60: `begin...end`
- Variable declarations in a compound statement.
  - Can define a new scope (with local variables).
  - A **block**.

Control Statements

- Control statements:
  - **Selection**
  - **Iteration (Loop)**
  - **Unconditional branch (Goto)**
- It was proven that all computations can be coded with only two-way **selection** and **pretest logical loops**.
- Overall Design Question:
  - What control statements should a language have beyond selection and pretest logical loops?

Control Statements

1. **Selection**
   - Two-way selection
   - Multi-way selection
2. **Iteration (Loop)**
   - Counter-controlled loops
   - Logically-controlled loops
3. **Unconditional Branching (Goto)**

1. Selection Statements

- **Two-way** selection statements
- **Multi-way** selection statements

Two-Way Selection Statements

- Design Issues:
  - What is the form and type of the control expression?
  - What is the selectable segment form (single statement, statement sequence, compound statement)?
  - How should the meaning of nested selectors be specified?

Single-Way Selector

- A subform of a two-way selector.
- FORTRAN IV: A logical IF
  - `IF (boolean_expr) a_single_statement`
  - Can select only a single statement.
  - To select more, a goto must be used.

```
    IF (FLAG .NOT. 1) GOTO 20
     I = 1
     J = 2
20 CONTINUE
```


Two-Way Selector

- ALGOL 60:

```
if (boolean_expr) then
    begin
     ...
    end

if (boolean_expr)then
       statement
    else
       statement
```


Nested Selectors

- Pascal:

```
if ... then
        if ... then
                ...
        else ...
```

- Which **`then`** gets the **`else`**?
  - Pascal's rule: **`else`** goes with the nearest **`then`**

Nested Selectors

- ALGOL 60's solution:
  - Disallow direct nesting. (Only in a compound statement)

```
if ... then
    begin
      if ... then
         ...
      else ...
    end
```

```
if ... then
     begin
       if ... then …
     end
else ...
```


Nested Selectors

- Algol 68, FORTRAN 77/90, Ada, Modula-2:
  - Closing special word (**`end if`**) adds readability.
  - Ada:

```
if ... then
   if ... then
      ...
   else
      ...
   end if
end if
```

```
if ... then
   if ... then
      ...
   end if
else
   ...
end if
```


Selection Closure

- Modula-2:
  - Uses the same closing special word for all control structures (**`END`**).
  - Less readable

Multiple (N-Way) Selection Statements

- Design Issues:
  - What is the form and type of the control expression?
  - What segments are selectable (single, compound, sequential)?
  - Is the entire construct encapsulated?
  - Is execution flow through the structure restricted to include just a single selectable segment?
  - What is done about unrepresented expression values?

Early Multiple Selectors - FORTRAN

- FORTRAN:
  - Arithmetic IF (a three-way selector):
    - `IF (arithmetic expression) N1, N2, N3`
  - Computed GOTO
    - `GO TO (label_1, label_2, …, label_n) exp`
  - Assigned GOTO
- Problems:
  - Lack of encapsulation
  - Multiple entries

Modern Multiple Selectors

- Pascal:
  - **`case`**

```
case expression of
    constant_list_1 : statement_1;
    ...
    constant_list_n : statement_n
end
```

  - Encapsulation of the selectable segments.
  - Implicit branch to the single exit point.

Modern Multiple Selectors

```
case index of
   1, 3: begin
           odd := odd + 1;
           sumodd := sumodd + index
         end;
   2, 4: begin
           even := even + 1;
           sumeven := sumeven + index
         end
   else  writeln("Error in case, index =', index)
end
```


Modern Multiple Selectors

- C and C++:
  - **`switch`**

```
switch (expression)  {
case constant_expression_1 :statement_1;
        ...
case  constant_expression_n : statement_n;
[default: statement_n+1]
}
```


Modern Multiple Selectors

```
switch (index) {
  case 1:
  case 3: odd += 1;
          sumodd += index;
  case 2:
  case 4: even += 1;
          sumeven += index;
  default:
printf ("Error in switch, index = %d\n", index);
}
```

Modern Multiple Selectors

```
switch (index) {
  case 1:
  case 3: odd += 1;
          sumodd += index;
          break;
  case 2:
  case 4: even += 1;
          sumeven += index;
          break:
  default:
printf ("Error in switch, index = %d\n", index);
}
```


Modern Multiple Selectors

- Using else-if clauses
  - ALGOL 68, FORTRAN 77, Modula-2, Ada

```
if ...
  then ...
elsif ...
  then ...
elsif ...
  then ...
  else ...
end if
```

  - More readable than deeply nested if's.

2. Iteration (Loop) Statements

- Execute a statement or compound statement zero or more times.
- Design Issues:
  - How is iteration controlled?
    - Counting
    - Logical
    - A combination
  - Where is the control mechanism in the loop?
    - Top
    - Bottom
    - User-defined control
  - Pretest vs. Posttest?

Iteration (Loop) Statements

1. **Counter-controlled loops**
2. **Logically-controlled loops**

(1) Counter-Controlled Loops

- Design Issues:
  - What is the type and scope of the loop var?
  - What is the value of the loop var at loop termination?
  - Should it be legal for the loop var or loop parameters to be changed in the loop body, and if so, does the change affect loop control?
  - Should the loop parameters be evaluated only once, or once for every iteration?

Counter-Controlled Loops - Example

- FORTRAN 77 and 90:
  - `DO label var = start, finish [,stepsize]`
  - See its operational semantics (p. 331)!
- FORTRAN 90's Other DO:

```
[name:] DO variable = initial, terminal [, stepsize]
        . . .
        . . .                    …
        END DO [name]
```

Counter-Controlled Loops - Example

- ALGOL 60:
  - `for var := <list_of_stuff> do statement`

    where `<list_of_stuff>` is a list of
    - expression
    - expression **`step`** expression **`until`** expression
    - expression **`while`** boolean_expression

Counter-Controlled Loops - Example

```
for index : = 1, 4, 13, 41 step 2 until 47,
                3*index while index < 1000,
                34, 2, -24 do
   sum := sum + index
```

1, 4, 13, 41, 43, 45, 47, 141, 423, 34, 2, -24

Counter-Controlled Loops - Example

- Pascal:
  - `for variable := initial (to | downto) final do statement`
- Ada:
  - `for var in [reverse] discrete_range loop ... end loop`
  - See the operational semantics (p. 330)!
- C:
  - `for (expr_1; expr_2; expr_3) statement`

Counter-Controlled Loops - Example

- C++:
  - The control expression can also be Boolean.
  - The initial expression can include variable definitions. (scope is from the definition to the end of the function in which it is defined)
- Java:
  - Control expression must be Boolean.
  - Scope of variables defined in the initial expression is only the loop body.

(2) Logically-Controlled Loops

- Design Issues:
  - Pretest or posttest?
  - Should this be a special case of the counting loop statement (or a separate statement)?

Logically-Controlled Loops - Example

- Pascal:
  - **`while-do`** and **`repeat-until`**
- C and C++:
  - **`while`** and **`do-while`**
- Java:
  - Like C, except the control expression must be Boolean.
- Ada:
  - Has a pretest version, but no posttest.
- FORTRAN 77 and 90:
  - Have neither.

User-Located Loop Control Mechanisms

- Design issues:
  - Should the conditional be part of the exit?
  - Should the mechanism be allowed in an already controlled loop?
  - Should control be transferable out of more than one loop?

User-Located Loop Control Mechanisms - Example

- Ada:
  - `exit [loop_label] [when condition]`

```
for ... loop
  ...
  exit when ...
  ...
end loop
```

```
LOOP1:
    for  …  loop
    ...
LOOP2:
    for ... loop
      ...
      exit LOOP1 when
..
      ...
    end loop LOOP2;
    ...
end loop LOOP1;
```


User-Located Loop Control Mechanisms - Example

- C , C++, and Java:
  - **`break`**
    - Unconditional and unlabeled exit.
- C , C++:
  - **`continue`**
    - Skips the remainder of this iteration, but does not exit the loop.
- Java and FORTRAN:
  - **`break EXIT`**
    - Unconditional and labeled exit.

User-Located Loop Control Mechanisms - Example

```
while (sum < 1000) {
   getnext(value);
   if (value < 0) break;
   sum += value;
}
```

```
while (sum < 1000) {
   getnext(value);
   if (value < 0) continue;
   sum += value;
}
```


3. Unconditional Branching (GOTO)

- Problems:
  - Readability!
- Some languages do not have them.
  - Modula-2 and Java.
- Should remain, but restricted use!

Statement-Level Control Structures - Summary

- Choice of control statements beyond selection and logical pretest loops?
  - Which control statements?
  - How about goto?
- A trade-off between language size and writability!