Chapter 8

Statement-Level Control Structures

Topics

- Introduction
- Selection Statements
- Iterative Statements
- Unconditional Branching
- Guarded Commands
- Conclusions

Levels of Control Flow

- Within expressions (Chapter 7)
  - governed by operator associativity and precedence rules
- Among program units (Chapter 9)
  - subprograms
- Among program statements (Chapter 8)

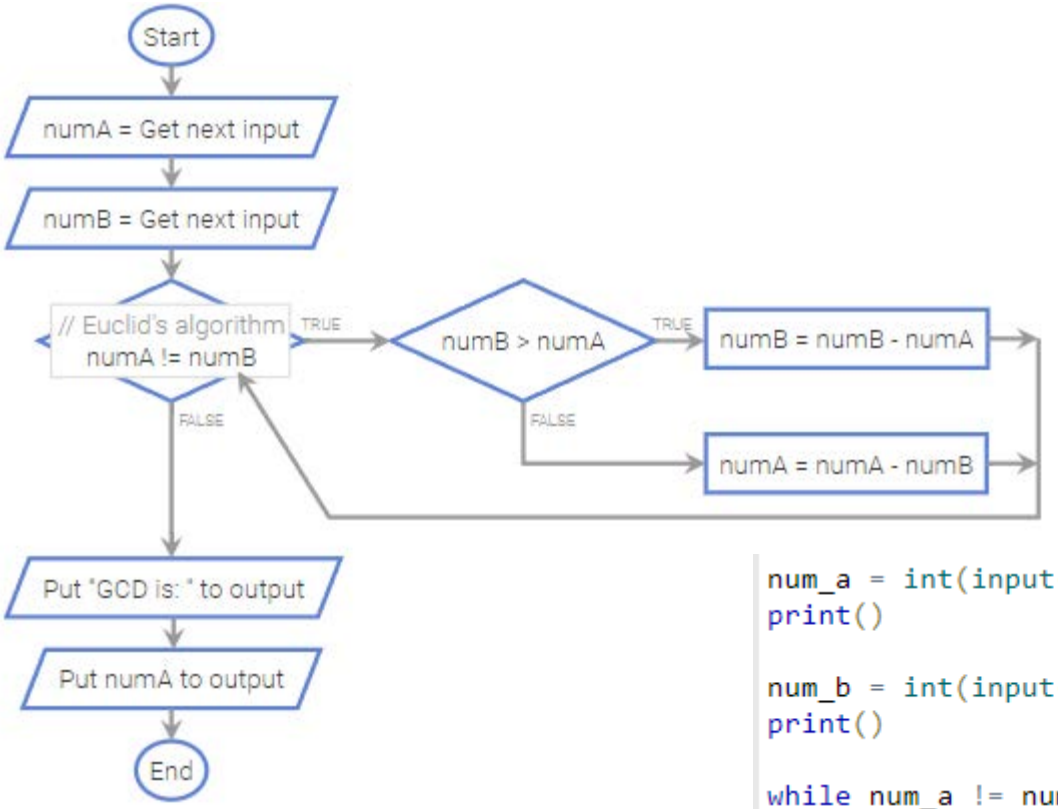
Control Statements: Evolution

- Some means of
  - *selecting* among alternative control flow paths (of statement execution) -- selection
  - those causing the *repeated execution* of (sequences of) statements. -- iteration
- Control statements in FORTRAN I: based directly on IBM 704 hardware. Cf) GOTO statement for control in Basic.
- Regardless many issues:

  It was proven that all algorithms represented by flowcharts can be coded with only a *two-way selection* and *pretest logical loops*.

Control Statements: Evolution (cont.)

- Example: Computing GCD (greatest common divisor)




```
num_a = int(input('Enter first positive integer: '))
print()

num_b = int(input('Enter second positive integer: '))
print()

while num_a != num_b:
    if num_a > num_b:
        num_a = num_a - num_b
    else:
        num_b = num_b - num_a

print(f'GCD is {num_a}')
```

Control Structure

- A *control structure* is a control statement and
  the collection of statements whose execution it controls.
  - Selection construct
  - Iteration construct

- Design question
  - Should a control structure have *multiple entries*?
    i.e. whether the execution of the code segments that are controlled by selection/iteration constructs always
    begin with the *first statement in the segment*?

Selection Statements

- A *selection statement* provides the means of choosing between two or more paths of execution.
- Two general categories:
  - Two-way selectors
  - Multiple-way selectors

Two-Way Selection Statements

- General form:

  `if` *control_expression*

  then-clause

  else-cause

- Design Issues:
  - What is the form and type of the *control expression*?
  - How are the **`then`** and **`else`** clauses specified?
  - How should the meaning of *nested selectors* be specified?

2-way selection: Control Expression

- If the ***then*** reserved word or some other syntactic marker is *not used* to introduce the then clause, the control expression is placed in *parentheses* ( ).

E.g.)
```
if (sum == 0)
    if (count == 0)
        result = 0;
    else result = 1;
```

- In C89, C99, Python, and C++:

  the control expression can be *arithmetic*.

- In most other languages:

  the control expression must be *Boolean*.

2-way selection: then/else Clause Form

- In many contemporary languages,
  the then and else clauses can be single statements
  or compound statements (i.e. block).
- In Perl: all clauses must be delimited by *braces* { }
  (they must be compound)
- In Python and Ruby: clauses are statement sequences.
- Python uses *indentation* to define clauses

```
if x > y :
    x = y
    print "x was greater than y"
```

2-way selection: Nesting Selectors

- ambiguous grammar:
  - <if_stmt> → **if** (<logic_expr>) <stmt>
    | **if** (<logic_expr>) **then** <stmt> **else** <stmt>
- Java example

```
if (sum == 0)
  if (count == 0)
      result = 0;
 else result = 1;
```

- Which **`if`** gets the **`else`**?

2-way selection: Nesting Selectors (cont.)

- unambiguous grammar: (chap. 3-#24)

<stmt> → <matched> | <unmatched>
<matched> → **if** (<logic_expr>) <matched> **else** <matched>
| any non-if statement
<unmatched> → **if** (<logic_expr>) <stmt>
| **if** (<logic_expr>) <matched> **else** <unmatched>

- Java example

```
if (sum == 0)
  if (count == 0)
      result = 0;
 else result = 1;
```

- Java's static semantics rule: **else** matches with the *nearest* previous **if**

2-way selection: Nesting Selectors (cont.)

- To force alternative semantics, compound statements may be used: Java, C, C++, and C#

```
if (sum == 0) {
  if (count == 0)
     result = 0;
}
else result = 1;
```

- In Perl: all then/else clauses be compound.

```
if (sum == 0) {
  if (count == 0){
     result = 0;
  }
} else {
     result = 1;
}
```

OR ?

```
if (sum == 0) {
   if (count == 0){
      result = 0;
   }
   else {
     result = 1;
   }
}
```

2-way selection: Nesting Selectors (cont.)

- Use of special keywords to resolve the semantic questions and to add readability.
- Statement sequences as clauses in Ruby:

eg1)

```
if sum == 0 then
   if count == 0 then
     result = 0
   else
     result = 1
   end
 end
```

eg2)

```
if a > b then sum = sum + a
  acount = acount + 1
else sum = sum + b
  bcount = bcount + 1
end
```

- Python

```
if sum == 0 :
   if count == 0 :
     result = 0
   else :
     result = 1
```

2-way selection: Selector Expressions

- In ML, F#, and Lisp, the selector is an expression, not a statement.
- In F#:

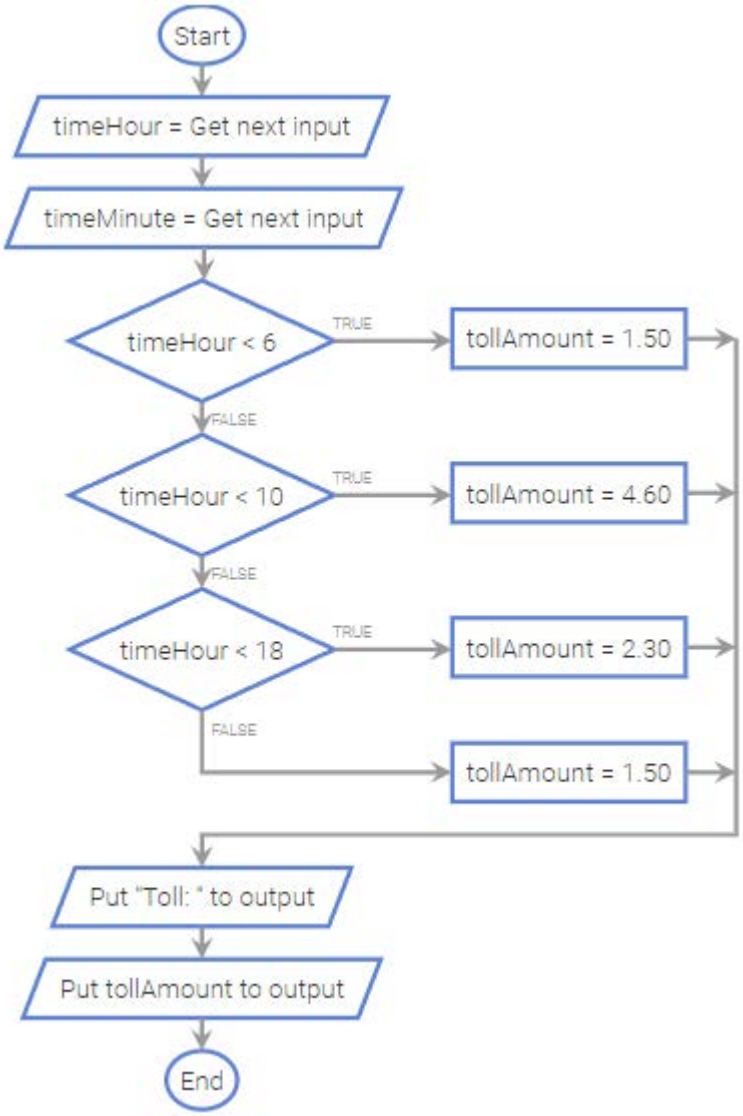
```
let y =
    if x > 0 then x
    else 2 * x;;
```

  - If the **`if`** expression *returns a value*, there must be an *else* clause.
  - The types of the returned values must be same for *then* and *else* clauses.

Multiple-Way Selection Statements

- Allow the selection of *one* of any number of statements or statement groups
- Design Issues:
  1. What is the form and type of the control expression?
  2. How are the selectable segments specified?
  3. Is execution flow through the structure restricted to include just a single selectable segment?
  4. How are case values specified?
  5. What is done about unrepresented expression values?

Multiple-Way Selection Statements: Example




```
timeHour = int(input('Enter an Hour of driving: '))
timeMinute = int(input('Enter a Minute of driving: '))

if timeHour < 6:            # Hour 6 and under
   tallAmount = 1.50

elif timeHour < 10:          # Hour 6 - 9
   tallAmount = 4.60

elif timeHour < 18:           # Hour 10 - 17
   tallAmount = 2.30

else:                        # Hour 17 and up
   tallAmount = 1.50

print('Tall: $',tallAmount)
```

Multiple-Way Selection: Examples

- C, C++, Java, and JavaScript

```
switch (expression) {
  case const_expr1: stmt1;
  …
  case const_exprn: stmtn;
  [default: stmtn+1]
}
```

- Example:

```
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);
}
```

Multiple-Way Selection: Examples

- The general form of the C switch statement:

```
switch (expression) {
 case constant_expression1: statement1;
  break;
 . . .
 case constantn: statementn;
  break;
 [default: statementn+1]
}
```

- translation:

```
Code to evaluate expression into t
goto branches
label1: code for statement1
            goto out
. . .
labeln: code for statementn
            goto out
default: code for statementn+1
            goto out
branches: if t = constant_expression1 goto label1
            . . .
            if t = constant_expressionn goto labeln
            goto default
out:
```

Multiple-Way Selection: Grammar from HW1

```
switch(expression) {
    case constant-expression:
        statement(s);
        break;          /* optional */

    /* you can have any number of case statements */

    default:            /* optional */
    statements(s);
}
```

<switch_stmt> specifies the switch statement abstraction
<expr> specifies an expression,
<literal> specifies a constant-expression,
<stmt_list> specifies a list of statements.
switch, case, break, default: keywords in C.

<switch_stmt> → **switch** (<expr>) {

case <literal> : <stmt_list> [**break**] - required

{ **case** <literal> : <stmt_list> [**break**] } - repeated any number of times >=0

[**default**: <stmt_list>] }

Multiple-Way Selection: Examples

```
switch (expression) {
  case const_expr1: stmt1;
  ...
  case const_exprn: stmtn;
  [default: stmtn+1]
}
```

- Design choices for C's **`switch`** statement
  1. Control expression can be only an *integer* type.
  2. Selectable segments can be statement sequences, blocks, or compound statements.
  3. *Any number of segments can be executed* in one execution of the construct (*there is no implicit branch at the end of selectable segments*)
  4. **`default`** clause is for unrepresented values.

     if there is no **`default`**, the whole statement does nothing.

Multiple-Way Selection: Examples

- Design choices for C's **`switch`** statement (cont.)

```
int index = 2;
int sumodd, odd = 0;
int sumeven, even = 0;

switch (index) {
    case 1:
    case 3: odd += 1;
            sumodd += index;
            printf("index = %d\n", index);
            printf("sumodd = %d\n", sumodd);

    case 2:
    case 4: even += index;
            sumeven += index;
            printf("index = %d\n", index);
            printf("sumeven = %d\n", sumeven);

    default: printf("Error in switch, \n");
             printf("index = %d\n", index);
             printf("sumeven = %d\n", sumeven);
             printf("sumodd = %d\n", sumodd);
}
```

```
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 = 2
sumeven = 2
Error in switch,
index = 2
sumeven = 2
sumodd = 0
```

It prints the error message on every execution.

The code for the 2 and 4 constants is executed every time the code at the 1 or 3 constants is executed.

→ Need to separate the segments logically – use of 'break'

Multiple-Way Selection: Examples

- C#'s switch statement
  - It disallows the implicit execution of more than one segment.
  - Each selectable segment must end with an unconditional branch (**`goto`** or **`break`**)
  - The control expression and the case constants can be *strings.*
  - Example:

```
switch (value) {
   case -1:
      Negatives++;
      break;
   case 0:
      Zeros++;
      goto case 1;
   case 1:
      Positives++;
   default:
      Console.WriteLine("Error in switch \n");
}
```

Multiple-Way Selection: Examples (cont.)

- Ruby: two forms of case statements

1.
```
case
    when Boolean_expression then expression
    ….
    when Boolean_expression then expression
    [else expression]
 end
```

similar to a list of nested if statements

2.
```
case expression
    when expression [, expression ..] then stmt
    …
    [else stmt ]
end
```

similar to switch statement of Java

Multiple-Way Selection: Examples (cont.)

1.
```
leap = case
           when year % 400 == 0 then true
           when year % 100 == 0 then false
           else year % 4 == 0
       end
```

2.
```
$age =  5
 case $age
    when 0 .. 2 then
       puts "baby"
    when 3 .. 6 then
       puts "little child“
    when 7 .. 12 then
       puts "child"
    when 13 .. 18 then
       puts "youth"
    else
       puts "adult"
 end
```

Implementing Multiple Selectors

- Approaches:
  - Multiple conditional branches.
  - Store case values in a table and use a *linear/binary search* of the table.
  - When there are more than 10 cases, a *hash table* of case values can be used – approximately equal/short time to choose any of the selectable segments.
  - If the number of cases is small and more than half of the whole range of case values are represented, an array whose *indices* are the case values and whose values are the *case labels* can be used.

Multiple-Way Selection Using `if`

- Multiple Selectors can appear as direct extensions to two-way selectors, using else-if clauses.
- in Python:

```
if count < 10 :
    bag1 = True
elif count < 100 :
    bag2 = True
elif count < 1000 :
    bag3 = True
else bag4 = True
```

≡

```
if count < 10 :
  bag1 = True
else :
  if count < 100 :
    bag2 = True
  else :
    if count < 1000 :
      bag3 = True
    else :
      bag4 = True
```

- Operational semantics description of a general selector statement with else-if clause:

```
    if E1 goto 1
    if E2 goto 2
    …
1:  S1
    goto out
2:  S2
    goto out
 ….
out:  ….
```

```
if E1:
      S1
elif E2:
      S2
```

Multiple-Way Selection Using **`if`**

- Multiple Selectors can appear as direct extensions to two-way selectors, using else-if clauses.

- in Ruby: it's written as a **`case`**.

```
case
   when count < 10   then bag1 = true
   when count < 100  then bag2 = true
   when count < 1000 then bag3 = true
   else bag4 = true
end
```

Scheme's Multiple Selector: Scheme

- General form of a call to `COND`:

```
(COND
  (predicate1 expression1)
   …
  (predicaten expressionn)
  [(ELSE expressionn+1)]
 )
```

```
(COND
  ((> x y) "x is greater than y")
  ((< x y) "y is greater than x")
  (ELSE "x and y are equal")
)
```

  - The `ELSE` clause is optional; `ELSE` is a synonym for true.
  - Each predicate-expression pair is a parameter.
  - Semantics: The value of the evaluation of `COND` is the value of the expression associated with the first predicate expression that is true.

Unconditional Branching

- Transfers execution control to a specified place in the program: goto
- One of the most heated debates in 1960's and 1970's.
  - Major concern: Readability
- Some languages do not support **`goto`** statement:
  - Java
- C# offers **`goto`** statement
  - **`goto`** can be used in **`switch`** statements
- All of the loop exit statements are actually disguised **`goto`** statements.

Conclusions

- Variety of statement-level structures.
- Choice of control statements beyond selection and logical pretest loops is a trade-off between language size and writability.
- Functional and logic programming languages use quite different control structures.