Statement-Level Control Structures

- selection statements
  - if
  - case
- iterative
  - while
  - repeat – until
  - for
- unconditional branching
  - goto
- guarded command control structures
  - Dijkstra

8.1. Introduction

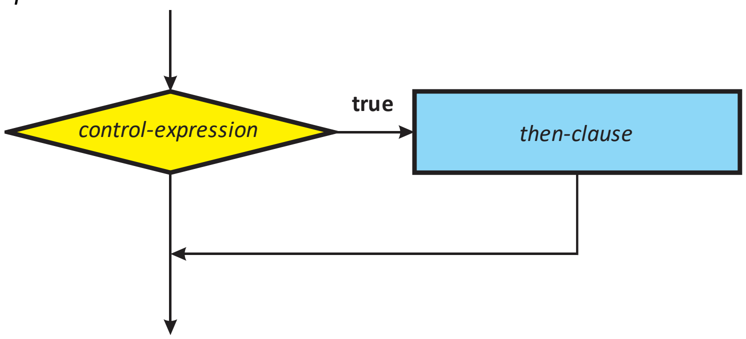
- while and if are sufficient – Böhm and Jacopini 1966

8.2. Selection Statements

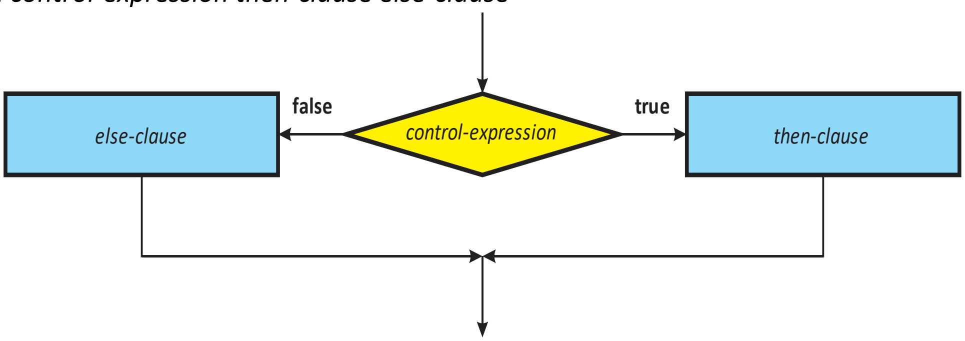
- A **selection statement** provides the means of choosing between two or more execution paths in a program.

8.2.1. Two-Way Selection Statements

- One-armed if
  if *control-expression then-clause*



- Two-armed if
  if *control-expression then-clause else-clause*



8.2.1.1. Design Issues

- What are the form and type of the expression that controls the selection?
- How are the then and else clauses specified?
- How should the meaning of nested selectors be specified?

8.2.1.2. The Control Expression

- Syntactic markers are required to distinguish the *control-expression*. One of two alternatives are generally chosen as exemplified by Pascal and C++.
  - Pascal

```
if control-expression then then-clause
if i<0 then i:=-i;              {one-armed if}
if control-expression then then-clause else else-clause
if (p>0) and (p>q)              {two-armed if}
    then begin p:=p-q; q:=p+q end
    else begin p:=p+q; q:=p-q end;
```

  - C++

```
if (control-expression) then-clause
if (i<0) i=-i;                  //one-armed if
if (control-expression) then-clause else else-clause
if (p>0 && p>q)  {              //two-armed if
    p=p-q; q=p+q;
} else {
    p=p+q; q=p-q;
}
```

  - Ada

```
if i<0 then i:=-i; end if;      --one-armed if
if p>0 and p>q              --two-armed if
    then p:=p-q; q:=p+q;
    else begin p:=p+q; q:=p-q;
end if;
```

  - Ruby

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

8.2.1.3. Clause Form

- Issue: single or compound statement
- Perl – all then and else clauses must be compound statements
- C-Based languages, JavaScript, Perl enclose compound statements in curly braces.
- Fortran 95, Ada, Python, and Ruby – then and else clauses are statement sequences. The complete selection construct is terminated with a reserve word.

- Python uses indentation to specify compound statements, For example,
  ```
  if x>y :
      x = y
      print "case 1"
  ```

8.2.1.4. Nesting Selectors

- Issue: which if-statement does an else-clause belong to when it is nested?
  ```
  if (sum==0)
      if (count==0)
          result=0;
      else                 //Does this else belong to if (sum==0)
          result=1;        //Or does this else belong to if (count==0)
  ```
- Normally, the static semantics of the language specify that the *else-clause* is always paired with the nearest previous unpaired *then-clause*.
- Compound-statements can force the issue
  ```
  if (sum==0) {
      if (count==0)result=0;
  } else result=1;
  ```

8.2.2. Multiple-Selection Statements

- The **multiple-selection** construct allows the selection of one of any number of statements or statement groups.

8.2.2.1. Design Issues

- What is the form and type of the expression that controls the selection?
- How are the selectable segments specified?
- Is execution flow though the structure restricted to include just a single selectable segment?
- How are the case values specified?
- How should unrepresented selector expression values be handled, if at all?

8.2.2.2. Examples of Multiple Selectors

- C, C++, Java
  ```
  switch (index) {
      case 1:
      case 3:  odd+=1;
               sumodd+=index;
               break;
      case 2:
      case 4:  even+=1;
               sumeven+=index;
               break;
      default:cout << "Error in switch, index = " << index; break;
  }
  ```

  Notes:
  1. Without the *break-statement* control continues to the next alternative.

- C#

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

Notes:

1. Every selectable segment must end with an explicit unconditional branch statement: either a **break**, which transfers control out of the switch construct, or a **goto**, which can transfer control to one of the selectable segments (or virtually anywhere else).

- Pascal

```
type direction=(North,East,South,West);
var x:direction;
…
case x of
      North:  x:=East;
      East:   x:=South;
      South:  x:=West;
      West:   x:=North
end;
```

- Ada

```
case x of
      when North    =>  x:=East;
      when East     =>  x:=South;
      when South    =>  x:=West;
      when West     =>  x:=North;
end;
```

- Ruby

```
case
when Boolean-expression then expression
…
when Boolean-expression then expression
[else expression]
end
```

8.2.2.3. Implementing Multiple Selection Structures

- A multiple selection construct is essentially an n-way branch to segments of code, where n is the number of selectable segments.
- Implementing such a construct must be done with multiple conditional branch instructions.

```
//The following code fragment in C++ is implemented below
switch (expression) {
      case     constant-expression-1: statement-1;
               break;
      …
      case     constant-expression-n: statement-n;
               break;
      [default: statement-n+1]
 }
//The foregoing code fragment in C++ is implemented below
t=expression;
goto branches
label-1:  code for statement-1
          goto out;
…
label-n:  code for statement-n
          goto out;
default:  code for statement-n+1
          goto out;

branches:    if t==constant-expression-1 goto label-1
             …
             if t==constant-expressionn-n goto label-n
             goto default
out:
```

8.2.2.4. Multiple Selection Using if

- Python

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

Implemented as

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

8.3. Iterative Statements

- The repeated execution of a statement or compound statement is accomplished either by iteration or recursion
- General design issues for iteration control statements:
  - How is iteration controlled?
  - Where is the control mechanism in the loop?
  - Entrances and exits

8.3.1. Counter-Controlled Loops

- A counting iterative statement has a loop variable, and a means of specifying the *initial* and *terminal*, and *stepsize* values

8.3.1.1. Design Issues

- What are the type and scope of the loop variable?
- Should it be legal for the loop variable 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?

8.3.1.2. The Do Statements of Fortran 95

- FORTRAN 95 syntax
  DO label var = start, finish [, stepsize]
- Stepsize can be any value but zero
- Parameters can be expressions
- Design choices:
  - o Loop variable must be INTEGER
  - o The loop variable cannot be changed in the loop, but the parameters can; because they are evaluated only once, it does not affect loop control
  - o Loop parameters are evaluated only once
  - o FORTRAN 95 : a second form:
    [name:] Do variable = initial, terminal [,stepsize]
    ...
    End Do [name]
    - Cannot branch into either of Fortran's Do statements

8.3.1.3. The Ada For Statement

- Ada

for var in [reverse] discrete_range loop ...
end loop

- Design choices:
  - Type of the loop variable is that of the discrete range (A discrete range is a sub-range of an integer or enumeration type).
  - Loop variable does not exist outside the loop
  - The loop variable cannot be changed in the loop, but the discrete range can; it does not affect loop control
  - The discrete range is evaluated just once
- Cannot branch into the loop body

8.3.1.4. The For Statement of the C-Based Language

- C-based languages
  for ([expr_1] ; [expr_2] ; [expr_3]) statement
  – The expressions can be whole statements, or even statement sequences, with the statements separated by commas
  – The value of a multiple-statement expression is the value of the last statement in the expression
  – If the second expression is absent, it is an infinite loop
- Design choices:
  – There is no explicit loop variable
  – Everything can be changed in the loop

- The first expression is evaluated once, but the other two are evaluated with each iteration

- C++ differs from C in two ways:
  1. The control expression can also be Boolean
  2. The initial expression can include variable definitions (scope is from the definition to the end of the loop body)
- Java and C#
  1. Differs from C++ in that the control expression must be Boolean

8.3.1.5. The For Statement of Python

- Python

```
for loop_variable in object:
 - loop body
[else:
 - else clause]
```

  - The object is often a range, which is either a list of values in brackets ([2, 4, 6]), or a call to the range function (range(5), which returns 0, 1, 2, 3, 4
  - The loop variable takes on the values specified in the given range, one for each iteration
  - The else clause, which is optional, is executed if the loop terminates normally

8.3.2. Logically Controlled Loops

- Repetition control is based on a Boolean expression

8.3.2.1. Design Issues

- Pretest or posttest?
- Should the logically controlled loop be a special case of the counting loop statement or a separate statement?

8.3.2.2. Example

- C and C++ have both pretest and posttest forms, in which the control expression can be arithmetic:

```
while (ctrl_expr)          do
    loop body                  loop body
                           while (ctrl_expr)
```

- Java is like C and C++, except the control expression must be Boolean (and the body can only be entered at the beginning -- Java has no goto
- Ada has a pretest version, but no posttest
- FORTRAN 95 has neither

- Perl and Ruby have two pretest logical loops, while and until. Perl also has two posttest loops

8.3.3. User-Located Loop Control Mechanisms

- Sometimes it is convenient for the programmers to decide a location for loop control (other than top or bottom of the loop)
- Simple design for single loops (e.g., break)
- Design issues for nested loops
    1. Should the conditional be part of the exit?
    2. Should control be transferable out of more than one loop?
- C , C++, Python, Ruby, and C# have unconditional unlabeled exits (**break)**
- Java and Perl have unconditional labeled exits (**break** in Java, **last** in Perl)
- C, C++, and Python have an unlabeled control statement, **continue**, that skips the remainder of the current iteration, but does not exit the loop
- Java and Perl have labeled versions of **continue**

8.3.4. Iteration Based on Data Structures

- Number of elements of in a data structure control loop iteration
- Control mechanism is a call to an *iterator* function that returns the next element in some chosen order, if there is one; else loop is terminate
- C's for can be used to build a user-defined iterator:

```
for (p=root; p==NULL; traverse(p)){
}
```

PHP
- current points at one element of the array
- next moves current to the next element
- reset moves current to the first element

- Java

- For any collection that implements the Iterator interface
- next moves the pointer into the collection
- hasNext is a predicate
- remove deletes an element

- Perl has a built-in iterator for arrays and hashes, foreach
- Java 5.0 (uses for, although it is called foreach)

- For arrays and any other class that implements Iterable interface, e.g., ArrayList

```
for (String myElement : myList) { … }
```

- C#'s foreach statement iterates on the elements of arrays and other collections:

```
Strings[] = strList = {"Bob", "Carol", "Ted"};
foreach (Strings name in strList)
    Console.WriteLine ("Name: {0}", name);
```

- The notation {0} indicates the position in the string to be displayed

- Lua

- Lua has two forms of its iterative statement, one like Fortran's Do, and a more general form:

```
for variable_1 [, variable_2] in iterator(table) do
  …
end
```

  - The most commonly used iterators are pairs and ipairs

8.4. Unconditional Branching

- Transfers execution control to a specified place in the program
- Represented one of the most heated debates in 1960's and 1970's
- Major concern: Readability
- Some languages do not support goto statement (e.g., Java)
- C# offers goto statement (can be used in switch statements)
- Loop exit statements are restricted and somewhat camouflaged goto's

8.5. Guarded Commands

- Designed by Dijkstra
- Purpose: to support a new programming methodology that supported verification (correctness) during development
- Basis for two linguistic mechanisms for concurrent programming (in CSP and Ada)
- Basic Idea: if the order of evaluation is not important, the program should not specify one
- Form

```
if <Boolean exp> -> <statement>
[] <Boolean exp> -> <statement>
 ...
[] <Boolean exp> -> <statement>
fi
```

- Semantics: when construct is reached,
  - Evaluate all Boolean expressions
  - If more than one are true, choose one non-deterministically
  - If none are true, it is a runtime error
- Connection between control statements and program verification is intimate
- Verification is impossible with goto statements
- Verification is possible with only selection and logical pretest loops
- Verification is relatively simple with only guarded commands

8.6. 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 are quite different control structures