A Book Chapter

By Narendra Kumar Chahar

on **Conditional Statements**

Index:

Chapter 5

Conditional Statements

In computer science, conditional statements, conditional expressions and conditional constructs are features of a programming language which perform different computations or actions depending on whether a programmer-specified Boolean condition evaluates to true or false. Apart from the case of branch predication, this is always achieved by selectively altering the control flow based on some condition.

In imperative programming languages, the term "conditional statement" is usually used, whereas in functional programming, the terms "conditional expression" or "conditional construct" are preferred, because these terms all have distinct meanings.

Although dynamic dispatch is not usually classified as a conditional construct, it is another way to select between alternatives at runtime.

If then (else)

The if–then construct (sometimes called if–then–else) is common across many programming languages. Although the syntax varies quite a bit from language to language, the basic structure (in pseudo code form) looks like this: (The example is actually perfectly valid Visual Basic or Quick BASIC syntax.)

```
IF (boolean condition)
  (consequent)
ELSE
  (alternative)
END IF
```

When an interpreter finds an If, it expects a Boolean condition – for example, x > 0, which means "the variable x contains a number that is greater than zero" – and evaluates that condition. If the condition is true, the statements following the then are executed. Otherwise, the execution continues in the following branch – either in the else block (which is usually optional), or if there is no else branch, then after the end If.

After either branch has been executed, control returns to the point after the end If.

In early programming languages, especially some dialects of BASIC in the 1980s home computers, an if–then statement could only contain GOTO statements. This led to a hard-to-read style of programming known as spaghetti programming, with programs in this style called spaghetti code. As a result, so called structured programming which allows (virtually) arbitrary statements to be put in statement blocks inside an if statement, gained in popularity, until it became the norm even in most BASIC programming circles. Such mechanisms and principles were based on the older but more advanced ALGOL family of languages, and ALGOL-like languages such as Pascal and Modula-2 influenced modern BASIC variants for many years. While it is possible while using only GOTO statements in if-then statements to write programs that are not spaghetti code and are just as well structured and readable as programs written in a structured programming language, so called structured programming makes this easier and enforces it. Structured if-then-else statements like the example above are one of the key elements of structured programming, and they are present in most popular high-level programming languages includingC, its derivatives (including C++ and C#), Java, JavaScript and Visual Basic (all versions including .NET).

Else if

By using else if, it is possible to combine several conditions. Only the statements following the first condition that is found to be true will be executed. All other statements will be skipped. The statements of

```
if condition then
   -- statements;
elsif condition then
   -- more statements
elsif condition then
   -- more statements;
...
else
   -- other statements;
end if;
```

elsif, in Ada, is simply syntactic sugar for else followed by if. In Ada, the difference is that only one end if is needed, if one uses elsif instead of else followed by if.

In some other languages, such as C and Java, else if literally just means else followed by if, and syntactic sugar is neither needed nor provided. This works because in these languages, an else followed by one statement (in this case the if) does not need braces. This is not true in Perl, which provides the keyword elsif to avoid the large number of braces that would be required by

multiple if and else statements. Similarly, Python uses the special keyword elif because structure is denoted by indentation rather than braces, so a repeated use of else and if would require increased indentation after every condition.

C and Java implementations, however, do have a disadvantage that each else if branch is, effectively, adding an extra nesting level, which will complicate compiler's writer job if arbitrarily long else if chains are to be supported.

LOOPS:

A loop is a sequence of statements which is specified once but which may be carried out several times in succession. The code "inside" the loop (the body of the loop, shown below as xxx) is obeyed a specified number of times, or once for each of a collection of items, or until some condition is met, or indefinitely.

In functional programming languages, such as Haskell and Scheme, loops can be expressed by using recursion or fixed point iteration rather than explicit looping constructs. Tail recursion is a special case of recursion which can be easily transformed to iteration.

Count-controlled loops

Most programming languages have constructions for repeating a loop a certain number of times. Note that if N is less than 1 in these examples then the language may specify that the body is skipped completely, or that the body is executed just once with N = 1. In most cases counting can go downwards instead of upwards and step sizes other than 1 can be used.

```
FOR I = 1 TO N              for I := 1 to N do begin
    xxx                         xxx
NEXT I                      end;

DO I = 1,N                  for ( I=1; I<=N; ++I ) {
    xxx                         xxx
END DO                      }
```

In many programming languages, only integers can be reliably used in a count-controlled loop. Floating-point numbers are represented imprecisely due to hardware constraints, so a loop such as

```
for X := 0.1 step 0.1 to 1.0 do
```

might be repeated 9 or 10 times, depending on rounding errors and/or the hardware and/or the compiler version. Furthermore, if the increment of X occurs by repeated

addition, accumulated rounding errors may mean that the value of X in each iteration can differ quite significantly from the expected sequence 0.1, 0.2, 0.3, ..., 1.0.

Condition-controlled loops

Most programming languages have constructions for repeating a loop until some condition changes. Note that some variations place the test at the start of the loop, while others have the test at the end of the loop. In the former case the body may be skipped completely, while in the latter case the body is always executed at least once.

```
DO WHILE (test)          repeat
   xxx                   xxx
LOOP                     until test;

while (test) {           do
   xxx                   xxx
}                      while (test);
```

A control break is a value change detection method used within ordinary loops to trigger processing for groups of values. A key changeable value or values are monitored within the loop and a change diverts program flow to the handling of the group event associated with the changeable value.

```
DO UNTIL (End-of-File)
  IF new-zipcode <> current-zipcode
    display_tally(current-zipcode, zipcount)

    current-zipcode = new-zipcode
    zipcount = 0
  ENDIF

  zipcount++
LOOP
```

Collection-controlled loops

Several programming languages
(e.g. Ada, D, Smalltalk, PHP, Perl, Object,Pascal, Java, C#, Mythryl, Visual,Basic, R uby, Python, JavaScript, Fortran 95 and later) have special constructs which allow

implicitly looping through all elements of an array, or all members of a set or collection.

```
someCollection do: [:eachElement |xxx].

for Item in Collection do begin xxx end;

foreach (item; myCollection) { xxx }

foreach someArray { xxx }

foreach ($someArray as $k => $v) { xxx }

Collection<String> coll; for (String s : coll) {}

foreach (string s in myStringCollection) { xxx }

$someCollection | ForEach-Object { $_ }

forall ( index = first:last:step... )
```

General iteration

General iteration constructs such as C's **for** statement and Common Lisp's **do** form can be used to express any of the above sorts of loops, as well as others—such as looping over a number of collections in parallel. Where a more specific looping construct can be used, it is usually preferred over the general iteration construct, since it often makes the purpose of the expression more clear.

Infinite loops

Infinite loops are used to assure a program segment loops forever or until an exceptional condition arises, such as an error. For instance, an event-driven program (such as a server) should loop forever handling events as they occur, only stopping when the process is terminated by an operator.

Often, an infinite loop is unintentionally created by a programming error in a condition-controlled loop, wherein the loop condition uses variables that never change within the loop.

Continuation with next iteration

Sometimes within the body of a loop there is a desire to skip the remainder of the loop body and continue with the next iteration of the loop. Some languages provide a statement such as continue (most languages), skip, or next (Perl and Ruby), which

will do this. The effect is to prematurely terminate the innermost loop body and then resume as normal with the next iteration. If the iteration is the last one in the loop, the effect is to terminate the entire loop early.

Redo current iteration

Some languages, like Perl and Ruby, have a redo statement that restarts the current iteration from the beginning.

Restart loop

Ruby has a retry statement that restarts the entire loop from the initial iteration.

Early exit from loops

When using a count-controlled loop to search through a table, it might be desirable to stop searching as soon as the required item is found. Some programming languages provide a statement such as break (most languages), exit, or last (Perl), whose effect is to terminate the current loop immediately and transfer control to the statement immediately following that loop. One can also return out of a subroutine executing the looped statements, breaking out of both the nested loop and the subroutine. Things can get a bit messy if searching a multi-dimensional table using nested loops (see #Proposed control structures below).

The following example is done in Ada which supports both early exit from loops and loops with test in the middle. Both features are very similar and comparing both code snippets will show the difference: early exit needs to be combined with an **if** statement while a condition in the middle is a self contained construct.

```
with Ada.Text IO;
with Ada.Integer Text IO;

procedure Print_Squares is
   X : Integer;
begin
   Read_Data : loop
      Ada.Integer Text IO.Get(X);
   exit Read_Data when X = 0;
      Ada.Text IO.Put (X * X);
      Ada.Text IO.New_Line;
   end loop Read_Data;
end Print_Squares;
```

Python supports conditional execution of code depending on whether a loop was exited early (with a break statement) or not by using a else-clause with the loop. For example,

```
for n in set_of_numbers:
   if isprime(n):
      print "Set contains a prime number"
      break
else:
   print "Set did not contain any prime numbers"
```

Note that the else clause in the above example is attached to the for statement, and not the inner if statement. Both Python's for and while loops support such an else clause, which is executed only if early exit of the loop has not occurred.

Loop variants and invariants

Loop variants and loop invariants are used to express correctness of loops.

In practical terms, a loop variant is an integer expression which has an initial non-negative value. The variant's value must decrease during each loop iteration but must never become negative during the correct execution of the loop. Loop variants are used to guarantee that loops will terminate.

A loop invariant is an assertion which must be true before the first loop iteration and remain true after each iteration. This implies that when a loop terminates correctly, both the exit condition and the loop invariant are satisfied. Loop invariants are used to monitor specific properties of a loop during successive iterations.

Some programming languages, such as Eiffel contain native support for loop variants and invariants. In other cases, support is an add-on, such as the Java Modeling Language's specification for loop statements in Java.

References:

[1]. Vessey, I., & Weber, R. (1984). Conditional statements and program coding: an experimental evaluation. *International Journal of Man-Machine Studies*, *21*(2), 161-190.

[2]. Wu, Bo, and Craig A. Knoblock. "Iteratively learning conditional statements in transforming data by example." *2014 IEEE International Conference on Data Mining Workshop*. IEEE, 2014.

[3]. Green, T. R. G. "Conditional program statements and their comprehensibility to professional programmers." *Journal of Occupational Psychology* 50.2 (1977): 93-109.

[4]. Hoc, Jean-Michel. "Do we really have conditional statements in our brains?." *Readings on Cognitive Ergonomics—Mind and Computers*. Springer, Berlin, Heidelberg, 1984. 92-101.

[5]. Rogalski, Janine, and Renan Samurçay. "Acquisition of programming knowledge and skills." *Psychology of programming*. Academic Press, 1990. 157-174.

[6]. Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008, March). Programming by choice: urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education* (pp. 367-371).