Chapter 15. Using Conditionals

Contents

A conditional is a statement that controls the execution of other statements; execution of the other statements is "conditioned" on statements such as if, else, case, and switch.

15.1 if Statements

Depending on the language you're using, you might be able to use any of several kinds of if statements.

- The simplest is the plain if or if-then statement.
- The if-then-else is a little more complex, and chains of if-then-else-if are the most complex.

Plain if-then Statements

Follow these guidelines when writing if statements:

Write the nominal path through the code first; then write the unusual cases.

- Write your code so that the normal path through the code is clear.
- Make sure that the rare cases don't obscure the normal path of execution. This is important for both readability and performance.

Put the normal case after the if rather than after the else.

- Put the case you normally expect to process first. This is in line with the general principle of putting code that results from a decision as close as possible to the decision.

Example 15-1. Visual Basic Example of Code That Processes a Lot of Errors

```
                 OpenFile( inputFile, status )
                 If ( status = Status_Error ) Then
error case    |     errorType = FileOpenError
                 Else
nominal case  |     ReadFile( inputFile, fileData, status )
                    If ( status = Status_Success ) Then
nominal case  |        SummarizeFileData( fileData, summaryData, status )
                       If ( status = Status_Error ) Then
error case    |           errorType = ErrorType_DataSummaryError
                       Else
nominal case  |           PrintSummary( summaryData )
                          SaveSummaryData( summaryData, status )
                          If ( status = Status_Error ) Then
error case    |              errorType = ErrorType_SummarySaveError
                          Else
nominal case  |              UpdateAllAccounts()
                             EraseUndoFile()
                             errorType = ErrorType_None
                          End If
                       End If
                    Else
                       errorType = ErrorType_FileReadError
                    End If
                 End If
```

Errors:

- the nominal cases and the error cases are all mixed together.
- It's hard to find the path that is normally taken through the code.
- In addition, because the error conditions are sometimes processed in the if clause rather than the else clause, it's hard to figure out which if test the normal case goes with.

For a complete list of available approaches, see "Summary of Techniques for Reducing Deep Nesting" in Taming Dangerously Deep Nesting.

Follow the if clause with a meaningful statement.

Example: Java Example of a Null if Clause

```
if ( SomeTest )
      ;
else {
      // do something
      ...
}
```

Cross-Reference
One key to constructing an effective if statement is writing the right boolean expression to control it.

Example 15-4. Java Example of a Converted Null

```
if ( ! SomeTest ) {
      // do something
      ...
}
```

- Consider the else clause. If you think you need a plain if statement, consider whether you don't actually need an if-then-else statement.
- One option is to code the else clause—with a null statement if necessary—to show that the else case has been considered.
- Use comments to explain why the else clause isn't necessary,

Example 15-5. Java Example of a Helpful, Commented else Clause

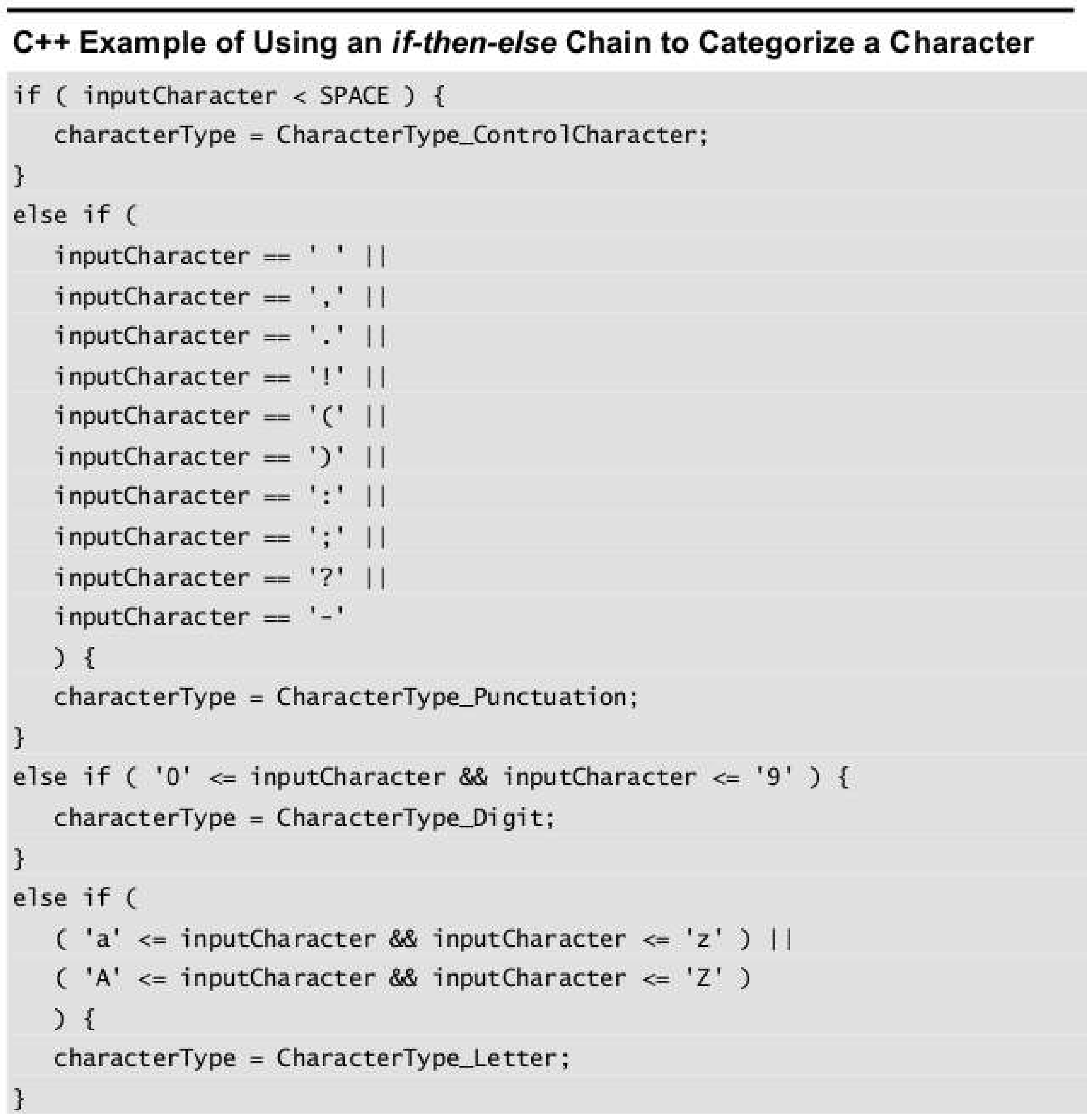
```
// if color is valid

if ( COLOR_MIN <= color && color <= COLOR_MAX ) {
      // do something
      ...
}
else {
      // else color is invalid
      // screen not written to -- safely ignore command
      en Python debemos escribir pass
}
```

15.2 Chains of if-then-else Statements

In languages that don't support case statements—or that support them only partially— you'll often find yourself writing chains of if-then-else tests.

Simplify complicated tests with boolean function calls.

Example 15-7. C++ Example of an if-then-else Chain That Uses Boolean Function Calls



C++ Example of Using an *if-then-else* Chain to Categorize a Character

```
if ( inputCharacter < SPACE ) {
   characterType = CharacterType_ControlCharacter;
}
else if (
   inputCharacter == ' ' ||
   inputCharacter == ',' ||
   inputCharacter == '.' ||
   inputCharacter == '!' ||
   inputCharacter == '(' ||
   inputCharacter == ')' ||
   inputCharacter == ':' ||
   inputCharacter == ';' ||
   inputCharacter == '?' ||
   inputCharacter == '-'
   ) {
   characterType = CharacterType_Punctuation;
}
else if ( '0' <= inputCharacter && inputCharacter <= '9' ) {
   characterType = CharacterType_Digit;
}
else if (
   ( 'a' <= inputCharacter && inputCharacter <= 'z' ) ||
   ( 'A' <= inputCharacter && inputCharacter <= 'Z' )
   ) {
   characterType = CharacterType_Letter;
}
```

C++ Example of an if-then-else Chain That Uses Boolean Function Calls

```
if ( IsControl( inputCharacter ) ) {
      characterType = CharacterType_ControlCharacter;
}
else if ( IsPunctuation( inputCharacter ) ) {
characterType = CharacterType_Punctuation;
}
else if ( IsDigit( inputCharacter ) ) {
characterType = CharacterType_Digit;
}
else if ( IsLetter( inputCharacter ) ) {
      characterType = CharacterType_Letter;
}
```

Put the most common cases first.

By putting the most common cases first:

- you minimize the amount of exception-case handling code someone has to read to find the usual cases.
- You improve efficiency because you minimize the number of tests the code does to find the most common cases.

C++ Example of Testing the Most Common Case First

```
if ( IsLetter( inputCharacter ) ) {
      characterType = CharacterType_Letter;
}
else if ( IsPunctuation( inputCharacter ) ) {
      characterType = CharacterType_Punctuation;
}
else if ( IsDigit( inputCharacter ) ) {
      characterType = CharacterType_Digit;
}
else if ( IsControl( inputCharacter ) ) {
      characterType = CharacterType_ControlCharacter;
}
```

Make sure that all cases are covered.

Code a final else clause with an error message or assertion to catch cases you didn't plan for. This error message is intended for you rather than for the user, so word it appropriately.

Example 15-9. C++ Example of Using the Default Case to Trap Errors

```
if ( IsLetter( inputCharacter ) ) {
      characterType = CharacterType_Letter;
}
else if ( IsPunctuation( inputCharacter ) ) {
      characterType = CharacterType_Punctuation;
}
else if ( IsDigit( inputCharacter ) ) {
      characterType = CharacterType_Digit;
}
else if ( IsControl( inputCharacter ) ) {
      characterType = CharacterType_ControlCharacter;
}
else {
      DisplayInternalError( "Unexpected type of character
detected." );
}
```

Replace if-then-else chains with other constructs if your language supports them.

15.2 Case Statements

Choosing the Most Effective Ordering of Cases

Order cases alphabetically or numerically

- If cases are equally important, putting them in A-B-C order improves readability.
- A specific case is easy to pick out of the group.

Put the normal case first

- If you have one normal case and several exceptions, put the normal case first.
- Indicate with comments that it's the normal case and that the others are unusual.

Order cases by frequency

Put the most frequently executed cases first and the least frequently executed last.
This approach has two advantage:

- Human readers can find the most common cases easily. Readers scanning the list for a specific case are likely to be interested in one of the most common cases.
- Putting the common ones at the top of the code makes the search quicker.

Tips for Using case Statements

Keep the actions of each case simple

Keep the code associated with each case short.
Short code following each case helps make the structure of the case statement clear.
If the actions performed for a case are complicated, write a routine and call the routine from the case rather than putting the code into the case itself.

Don't make up phony variables in order to be able to use the case statement

A case statement should be used for simple data that's easily categorized. If your data isn't simple, use chains of if-then-elses instead.

Use the default clause only to detect legitimate defaults

You might sometimes have only one case remaining and decide to code that case as the default clause. Though sometimes tempting, that's dumb. You lose the automatic documentation provided by case-statement labels, and you lose the ability to detect errors with the default clause.

Use the default clause to detect errors

If the default clause in a case statement isn't being used for other processing and isn't supposed to occur, put a diagnostic message in it.
Messages like this are useful in both debugging and production code.

Java Example of Using the Default Case to Detect Errors—Good Practice

```
switch ( commandShortcutLetter ) {
      case 'a':
            PrintAnnualReport();
            break;
      case 'p':
            // no action required, but case was considered
            break;
      case 'q':
            PrintQuarterlyReport();
            break;
      case 's':
            PrintSummaryReport();
            break;
      default:
            DisplayInternalError( "Internal Error 905: Call customer
      support." );
      }
```

In C++ and Java, avoid dropping through the end of a case statement

C-like languages (C, C++, and Java) don't automatically break out of each case. Instead, you have to code the end of each case explicitly. If you don't code the end of a case, the program drops through the end and executes the code for the next case.

Comparing Switches and if-then Statements

Del libro Beginning Java Programming

Just like for and while loops are similar structures, switches and if-then statements are also easy to compare.

When you are using a switch, you read it the same way as an if-then statement: if the value matches the case, then do something. So how do you know when to use each one?

As with the other control structures, there will be situations when either one is appropriate and you
can choose according to your own preference. As you continue coding, your experience will tell you if a problem would be better solved with a switch or not.

In general, you might consider the following criteria:

➤➤ If you have a single variable that can take multiple values, a switch might be suitable.
➤➤ If you have multiple variables or conditions to consider, you will probably need an if-then statement.
➤➤ If the value you are considering can have a finite number of values, consider using a switch.
➤➤ If the variable can take any value within a continuous range of numbers, consider an if-then statement.

CHECKLIST: Using Conditionals

if-then Statements

❏ Is the nominal path through the code clear?
❏ Do if-then tests branch correctly on equality?
❏ Is the else clause present and documented?
❏ Is the else clause correct?
❏ Are the if and else clauses used correctly—not reversed?
❏ Does the normal case follow the if rather than the else?

if-then-else-if Chains

❏ Are complicated tests encapsulated in boolean function calls?
❏ Are the most common cases tested first?
❏ Are all cases covered?
❏ Is the if-then-else-if chain the best implementation—better than a case statement?

KEY POINTS

- For simple if-else statements, pay attention to the order of the if and else clauses, especially if they process a lot of errors. Make sure the nominal case is clear.
- For if-then-else chains and case statements, choose an order that maximizes readability.
- To trap errors, use the default clause in a case statement or the last else in a chain of if-then-else statements.
- All control constructs are not created equal. Choose the control construct that's most appropriate for each section of code.