The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

6.8.1 Labeled statements

labeled statements syntax

1722

```
labeled-statement:
                identifier : statement
                case constant-expression : statement
                default : statement
```

Commentary

Rationale

Case ranges of the form, *lo .. hi*, were seriously considered, but ultimately not adopted in the Standard on the grounds that it added no new capability, just a problematic coding convenience. The construct seems to promise more than it could be mandated to deliver:

- A great deal of code or jump table space might be generated for an innocent-looking case range such as `0 .. 65535`.
- The range `'A' .. 'Z'` would specify all the integers between the character code for "upper-case-A" and that for "upper-case-Z". In some common character sets this range would include non-alphabetic characters, and in others it might not include all the alphabetic characters, especially in non-English character sets.

No serious consideration was given to making switch more structured, as in Pascal, out of fear of invalidating working code.

When source code oriented symbolic debuggers (i.e., the ability to refer to lines in the source) are not available labels are sometimes used as a means of associating a symbol with a known point in the code.

Other Languages

Some languages (e.g., Fortran and Pascal) require that labels be digit, rather than alphabetic, character sequences.

Languages in the Pascal family do not classify **`case`**, or any equivalent to **`default`**, as a label. They are usually defined as part of the syntax of the selection statement in which they appear. Ada intentionally[1] differentiates visually between statement labels and case labels. An identifier that is a statement label is required to appear between the tokens `<<` and `>>`. Some languages use **`else`** or **`otherwise`** rather than **`default`** in their selection statements.

Common Implementations

Some implementations (e.g., `gcc` and Code Warrior[2]) support case ranges as an extension.

Coding Guidelines

Many coding guideline documents recommend against the use of **`goto`** statements. However, in the case of C language, prohibiting labels (except **`case`** and **`default`**) would be more encompassing. For instance, a mistyped label in a **`switch`** statement may turn it into another kind of label (e.g., misspelling **`default`**, `defolt` is often seen for non-English speakers, creates a different kind of label). The issues associated with the use of jump statements are discussed elsewhere.

jump statement syntax

Usage

In the translated form of this book's benchmark programs 2% of labels were not the destination of any **`goto`** statement. Usage information on **`goto`** statements is given elsewhere.

jump statement causes jump to

Table 1722.1: Percentage of function definitions containing a given number of labeled statements (other than a **case** or **default** label). Based on the visible form of the `.c` files.

Labels	% Functions	Labels	% Functions
1	3.5	3	0.3
2	0.9	4	0.1

Constraints

1723 A **case** or **default** label shall appear only in a **switch** statement.

case label appear within switch

Commentary

This constraint is necessary because these constructs are not distinguished syntactically from other kinds of labels.

Other Languages

In other languages these kinds of labels are usually distinguished in the language syntax.

Coding Guidelines

This constraint does not prohibit **case** or **default** labels occurring in a variety of potentially surprising, to a reader, contexts within a **switch** statement. For instance, within a nested compound statement.

```
extern int glob;

void f(int valu)
{
switch (valu)
   {
   case 0: if (glob == 3)
              {
              case 2: value--;
              }
           break;
   }
}
```

In practice this usage is rare. Some practical applications of this usage are discussed elsewhere.

Duff's Device

1724 Further constraints on such labels are discussed under the **switch** statement.

Commentary

This is essentially a forward reference.

forward reference

1725 Label names shall be unique within a function.

label name unique

Commentary

Label names have function scope, which means they are visible anywhere within the function definition that contains them.

scope function

C90

This wording occurred in Clause 6.1.2.1 Scope of identifiers in the C90 Standard. As such a program that contained non unique labels exhibited undefined behavior.

A function definition containing a non-unique label name that was accepted by some C90 translator (a very rare situation) will cause a C99 translator to generate a diagnostic.

Other Languages

This requirement is common to most languages, although a few (e.g., Algol 68) give labels block scope. Some assembly languages supported duplicate label names. A jump instruction usually being defined, in these cases, to jump to the closest label with a given name.

Common Implementations

Although this was not a requirement that appeared within a Constraints clause in C90, all implementations known to your author issued a diagnostic if a label name was not unique within a function definition.

Coding Guidelines

redundant code

Label names that are not referenced might be classified as redundant code. However, labels have uses other than as the destination of **goto** statements. They may also be used as breakpoints when stepping through code using a symbolic debugger. Automatically generated code may also contain labels that are not jumped to. Given that labels are relatively uncommon, and the only nontrivial cost involved in a redundant label name is likely to be a small increase in reader comprehension time, a recommendation that all defined labels be jumped to from somewhere does not appear to have a worthwhile benefit.

label naming conventions

The issues relating to the spelling of label names are discussed elsewhere.

Semantics

Any statement may be preceded by a prefix that declares an identifier as a label name. 1726

Commentary

It is not possible to prefix a declaration with a label (although a preceding null statement could be labeled).

C does not provide any mechanism for declaring labels before they appear as a prefix on a statement. Labels may be referenced before they are declared. That is it is possible to **goto** a label that has not yet been seen within the current function.

Other Languages

Some languages (e.g., those in the Pascal family) provide declaration syntax for label names. Their appearance as a statement prefix is simply a use of a name that has been previously defined. Some languages (e.g., Fortran) allow all lines to be prefixed by a line number. These line numbers can also be used as the destination in **goto** statements.

---

case fall through

Labels in themselves do not alter the flow of control, which continues unimpeded across them. 1727

Commentary

EXAMPLE case fall through

An example of this is given elsewhere.

C++

The C++ Standard only explicitly states this behavior for **case** and **default** labels (6.4.2p6). It does not specify any alternative semantics for 'ordinary' labels.

Other Languages

In some languages certain kinds of labels alter the flow of control. For instance, in Pascal encountering a **case** label cases flow of control to jump to the end of the **switch** statement (the following using Pascal syntax and keywords).

```
case expr of
   1: begin
      x:=88;
      y:=99;
      end;

   2: z:=77;
   end;
```

Common Implementations

Labels may not alter the flow of control but they are usually the destination of a change of flow of control. As such they are bad news for code optimization, because accumulated information on the values of objects either has to be reset to nothing (worst-case assumption made by a simple optimizer), or merged with information obtained along other paths (more sophisticated optimizer).

Coding Guidelines

Most C coding guideline documents recommend against *falling through* to a statement prefixed by a **case** or **default** label. For instance, in the following fragment it is likely that the author had forgotten to place a **break** statement after the assignment to y.

```
case 1: x=88;
        y=99;

case 2: z=77;
```

There are occasions where falling through to a statement prefixed by a **case** label is the intended behavior. A convention sometimes adopted is to place a comment containing the words FALL THROUGH on the line before the statement that is fallen into (some static analysis tools recognize this usage and don't issue a diagnostic for the fall through that it comments). Duff's Device

Experience suggests that code containing a statement prefixed by a **case** label whose flow of control *falls through* to another statement prefixed by a **case** label causes readers to spend significant effort in verifying whether the usage is intended or a fault in the code. Commenting such usage has a cost that is likely to be significantly smaller than the benefit.

Cg 1727.1

If the flow of control can reach a statement prefixed by one or more **case** labels, other than by a direct jump from the controlling expression of a **switch** statement, the source line immediately before that statement shall contain the words "FALL THROUGH" in a comment.

Table 1727.1: Common token pairs involving a **case** or **default** label. Based on the visible form of the .c files. Almost all of the sequences { case occur immediately after the controlling expression of the **switch** statement.

Token Sequence	% Occurrence First Token	% Occurrence of Second Token
; default	0.4	81.4
; case	2.1	52.1
: case	15.5	22.1
{ case	2.6	15.0
} case	1.3	7.3
: default	0.5	5.7
#endif default	0.8	4.4

1728 **Forward references:** the **goto** statement (6.8.6.1), the **switch** statement (6.8.4.2).

References

1. J. Ichbiah, J. Barnes, R. Firth, and M. Woodger. *Rationale for the Design of the Ada Programming Language*. Cambridge University Press, 1991.
2. Metroworks. *CodeWarrior C Compilers Reference*. Metroworks Corp., Aug. 2001.