P2324R1
Title: Labels at the end of compound statements (C compatibility)
Author: Martin Uecker, Graz University of Technology
Date: 2021-10-14

Changes in revision 1

2021-10-14:
- Implementation experience
- Update wording to be consistent with latest draft
- Add change for adding the implied null statement similar to C.

2021-10-12:
- Removed wording alternatives
- WG21 poll

Introduction

WG14 adopted a change for C2X that allows placement of labels everywhere inside a compound statement (N2508). While this improves compatibility with C++ which previously diverged from C by allowing labels in front of declarations, there is still a remaining incompatibility: C now does allow labels at the end of a compound statement, while C++ does not. It is proposed to change the C++ grammar to remove this remaining difference.

Example:

```
void foo(void)
{
first:      // allowed in C++, now also allowed in C
      int x;

second:     // allowed in both C++ and C
      x = 1;

last:       // not allowed in C++, but now allowed in C
}
```

The underlying reason for this difference is that the structure of the grammar is different.

In C declarations and statements are separate production rules which can both appear as block-items inside compound statements. The simplest change for C was to also allow labels as independent block-items in addition to statements and declarations. This change then also allowed placing labels at the end of a compound statement which was seen as useful feature.

In C++ declarations are statements and compound statements can only have statements as block-items. Thus, labels can already be attached to all statements, i.e. including declarations, but can not be placed at the end of compound statements. Another difference is that in C++ (but not in C) it is possible to use declarations as sub-statements of a control statements. The later seems to be an unintended side effect of making declarations be statements and now requires a rewrite rule to place this declaration into a new scope.

Example:

```
void bar(void)
{
      if (1)
      here: int x;  // declaration allowed in C++ (not in C)
}
```

is rewritten to:

```
void bar(void)
{
      if (1) {
      here:
            int x;
      }
}
```

The paper was discussed in the C++/C compatibility study group with the recommendation to pass it on to EWG:

Does SG22 recommend forwarding P2324R0 to EWG?

SF	F	N	A	SA	
1	4	0	0	0	WG14
0	6	2	0	0	WG21

Implementation Experience

GCC implements the new syntax for C since version 11 in default mode with warnings for pre C2X versions only with -Wpedantic. TCC accepts the code in development versions. ICC, SDCC, and released versions of TCC accept it with warnings. ICC also accepts the code for C++ with warning.

Backward Compatibility

There are no backward compatibility concerns. Compilers seem to either accept the syntax and then implement the obvious semantics or emit an error. For C there is a possible subtle change in the interpretation of blocks (N2663), but this does not seem to affect C++. regarding A feature testing macro seems unnecessary as it would be far simpler for code that relies on older compilers to add continue to add a semicolon as a null statement.

Required Wording Changes

The wording proposed here is a minimal self-contained change that adds an explicit rule to have labels at the end of compound-statement. The disadvantage is that such labels are treated specially and the formal grammar does not reflect the full symmetry of the situation. (P2324R0 lists alternatives). Additionally, we insert an implied null statement after the label if it appears at the end of a compound statement, because control flow is defined in terms of statements. The changes are against the latest draft available on https://eel.is/c++draft/ generated on 2021-08-26.

[stmt.label] 8.2 Label ~~Labeled statement~~

~~A statement can be labeled.~~ A label can be added to a statement or used anywhere in a compound-statement.

~~labeled-statement:~~
label:
attribute-specifier-seq$_{opt}$ identifier : ~~statement~~
attribute-specifier-seq$_{opt}$ case constant-expression : ~~statement~~
attribute-specifier-seq$_{opt}$ default : ~~statement~~

labeled-statement:
label statement

The optional attribute-specifier-seq appertains to the label. The only use of an identifier label is as the target of a goto. No two labels in a function shall have the same *identifier*. A label can be used in a goto statement before its introduction.~~by a *labeled-statement.*~~

Case labels and default labels shall occur only in switch statements.

[stmt.expr] 8.3 Expression statement

Expression statements have the form

expression-statement:
expression$_{opt}$;

The expression is a discarded-value expression. All side effects from an expression statement are completed before the next statement is executed. An expression statement with the expression missing is called a null statement.

[Note: Most statements are expression statements — usually assignments or function calls. A null statement is useful ~~to carry a label just before the } of a compound statement and~~ to supply a null body to an iteration statement such as a while statement ([stmt.while]). — end note]

[stmt.block] 8.4 Compound statement or block

A compound-statement (also known as a block) groups a sequence of statements into a single statement.

compound-statement:
{ statement-seq$_{opt}$ label-seq$_{opt}$ }

statement-seq:
statement
statement-seq statement

label-seq:
label
label-seq label

A label at the end of a compound statement shall be translated as if it were followed by a null statement. [Note 1: A compound statement defines a block scope ([basic.scope]). A declaration is a statement ([stmt.dcl]). — end note]