Title: Labels at the end of compound statements (C compatibility)
Author: Martin Uecker, University of Göttingen, Germany
Date: 2020-10-26

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. On the other hand, in C++ (but not in C) is it 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;
      }
}
```

Wording Changes

We list three alternative wording changes.

Alternative 1 is a minimal self-contained change that adds a labels an explicit rule to have labels at the end of compound-statement. The disadvantage is that such labels are treated specially and formal grammar does not reflect the full symmetry of the situation.

Alternative 2 is still a simple change, which treats all labels in a compound-statement equally. The change makes the grammar also more similar to the C language, which could be seen as an advantage. It preserves the treatment of declarations as statement in C++ which is different to C.

Alternative 3 is a more complex change. It is based on the observation that regular statements are used as substatements of selection statements and iteration statements only. But these are also exactly the exceptional cases that in C++ need to be rewritten into a compound-statements to introduce a scope in case they contain declarations (see the example above). But after such rewriting, declarations, labels, and statements then only appear inside compound-statements where they can be mixed freely (according to the proposed rules). This suggests refactoring of the grammar by introducing the concept of a controlled-statement that can be used substatement of selection statements and iteration statements. These controlled-statements are then always rewritten to compound-statements, taken care of all special cases using a single rule. A regular statement can then be defined in a simpler and more natural way by excluding both declaration-statements and labels. Statements, declaration-statements, and labels are all treated equally inside compound-statements. With these changes, the C++ grammar would again follow the structure of the C grammar closely, become more symmetric, and the exceptional rewrite rules are consolidated to a single one.

Common Wording

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. An identifier label declares the identifier. The only use of an identifier label is as the target of a goto. The scope of a label is the function in which it appears. Labels shall not be redeclared within a function. A label can be used in a gotoIntr statement before its declaration. Labels have their own name space and do not interfere with other identifiers. [Note: A label may have the same name as another declaration in the same scope or a template-parameter from an enclosing scope. Unqualified name lookup ignores labels. — end note]

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

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. — end note]

Wording Alternative 1

8.4 Compound statement or block

So that several statements can be and used where one is expected, the compound statement (also, and equivalently, called “block”) is provided.

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

statement-seq:
statement
statement-seq statement

label-seq:
label
label-seq label

A compound statement defines a block scope. [Note: A declaration is a statement ([stmt.dcl]). — end note]

Wording Alternative 2

8. Statements
8.1 Except as indicated, statements are executed in sequence.

statement:
labeled-statement
unlabeled-statement

unlabeled-statement:
~~labeled-statement~~
attribute-specifier-seq$_{opt}$ expression-statement
attribute-specifier-seq$_{opt}$ compound-statement
attribute-specifier-seq$_{opt}$ selection-tatement
attribute-specifier-seq$_{opt}$ iteration-statement
attribute-specifier-seq$_{opt}$ jump-statement
declaration-statement
attribute-specifier-seq$_{opt}$ try-block

8.4 Compound statement or block

So that several statements and labels can be mixed freely and used where ~~one~~ a single statement is expected, the compound statement (also, and equivalently, called “block”) is provided.

compound-statement:
{ ~~statement-seq$_{opt}$~~ block-item-seq$_{opt}$ }

~~statement-seq:~~
~~statement~~
~~statement-seq statement~~

block-item-seq:
block-item
block-item-seq block-item

block-item:
label
unlabeled-statement

A compound statement defines a block scope. [Note: A declaration is a statement ([stmt.dcl]). — end note]

8.5

The substatement in a selection-statement (each substatement, in the else form of the if statement) implicitly defines a block scope ([basic.scope]). If the substatement in a selection-statement is a single statement and not a compound-statement, it is as if it was rewritten to be a compound-statement containing the original substatement including all labels which become independent block items. [Example:
if (x)
  here: int i;

can be equivalently rewritten as
```
if (x) {
  here:
  int i;
}
```
Thus after the if statement, i is no longer in scope. — end example]

8.6.

The substatement in an iteration-statement implicitly defines a block scope which is entered and exited each time through the loop. If the substatement in an iteration-statement is a single statement and not a compound-statement, it is as if it was rewritten to be a compound-statement containing the original statement including all labels which become independent block items. [Example:
```
while (--x >= 0)
  here: int i;
```
can be equivalently rewritten as
```
while (--x >= 0) {
  here:
  int i;
}
```
Thus after the while statement, i is no longer in scope. — end example]

Wording Alternative 3

8. Statements
8.1 Except as indicated, statements are executed in sequence.

statement:
- ~~labeled-statement~~
- attribute-specifier-seq$_{opt}$ expression-statement
- attribute-specifier-seq$_{opt}$ compound-statement
- attribute-specifier-seq$_{opt}$ selection-tatement
- attribute-specifier-seq$_{opt}$ iteration-statement
- attribute-specifier-seq$_{opt}$ jump-statement
- ~~declaration-statement~~
- attribute-specifier-seq$_{opt}$ try-block

controlled-statement:
label controlled-statement
statement
declaration-statement

A controlled-statement implicitly defines a block scope ([basic.scope]). A controlled-statement is rewritten to be a compound-statement containing the original statement or declaration-statement and including all labels which become independent block items. [Note: Controlled statements are used in a selection statements and iteration statements – end node]
[Example:
if (x)
here: int i;
is rewritten to
if (x) {
here:
int i;
} – end example]

A substatement of a statement is one of the following:
(2.1) for a labeled-statement, its contained statement or declaration-statement,
(2.2) for a compound-statement, any statement or declaration-statement of its statement-seq,
(2.3) for a selection-statement, ~~any of its statements (but not its init-statement), or,~~ any statement or declaration statement contained in any of its controlled-statements
(2.4) for an iteration-statement, ~~its contained statement (but not an init-statement),~~ the statement or declaration-statement contained in its controlled-statement
(2.5) for a controlled-statement, its contained statement or declaration-statement.

8.4 Compound statement or block

So that several statements, declarations, and labels can be mixed freely and used where ~~one~~ a single statement is expected, the compound statement (also, and equivalently, called “block”) is provided.

compound-statement:
{ ~~statement-seq~~$_{opt}$ block-item-seq$_{opt}$ }

~~statement-seq:~~
~~statement~~
~~statement-seq statement~~

block-item-seq:
block-item
block-item-seq block-item

block-item:
label
statement
declaration-statement

A compound statement defines a block scope. ~~[Note: A declaration is a statement ([stmt.dcl]). — end note]~~

8.5

Selection statements choose one of several flows of control.

selection-statement:
if constexpr$_{opt}$ (init-statement$_{opt}$ condition) controlled-statement
if constexpr$_{opt}$ (init-statement$_{opt}$ condition) controlled-statement else controlled-statement
switch (init-statement$_{opt}$ condition) controlled-statement

~~The substatement in a selection-statement (each substatement, in the else form of the if statement) implicitly defines a block scope ([basic.scope]). If the substatement in a selection-statement is a single statement and not a compound-statement, it is as if it was rewritten to be a compound-statement containing the original substatement. [Example:~~
~~if (x)~~
~~int i;~~
~~can be equivalently rewritten as~~
~~if (x) {~~
~~int i;~~
~~}~~
~~Thus after the if statement, i is no longer in scope. — end example]~~

An if statement of the form

if constexpr$_{opt}$ (init-statement condition) controlled-statement

is equivalent to

{
init-statement
if constexpr$_{opt}$ (condition) controlled-statement
}

and an if statement of the form

if constexpr$_{opt}$ (init-statement condition) controlled-statement else controlled-statement

is equivalent to

{
init-statement
if constexpropt (condition) controlled-statement else controlled-statement
}

except that names declared in the init-statement are in the same declarative region as those declared in the condition.

A switch statement of the form

switch (init-statement condition) controlled-statement

is equivalent to

```
{
        init-statement
        switch ( condition ) controlled-statement
}
```

except that names declared in the init-statement are in the same declarative region as those declared in the condition.

8.6 Iteration statements [stmt.iter]

Iteration statements specify looping.

```
iteration-statement:
    while ( condition ) controlled-statement
    do controlled-statement while ( expression ) ;
    for ( init-statement conditionopt ; expressionopt ) controlled-statement
    for ( init-statementopt for-range-declaration : for-range-initializer ) controlled-statement
```

~~The substatement in an iteration-statement implicitly defines a block scope which is entered and exited each time through the loop. If the substatement in an iteration-statement is a single statement and not a compound-statement, it is as if it was rewritten to be a compound-statement containing the original statement. [Example:~~

```
~~while (--x >= 0)~~
~~  here: int i;~~
~~can be equivalently rewritten as~~
~~while (--x >= 0) {~~
~~  here:~~
~~  int i;~~
~~}~~
```

~~Thus after the while statement, i is no longer in scope. — end example]~~

8.6.1 The while statement

When the condition of a *while* statement is a declaration, the scope of the variable that is declared extends from its point of declaration ([basic.scope.pdecl]) to the end of the *while* controlled-statement. A *while* statement is equivalent to

```
label :
{
        if ( condition ) {
                controlled-statement
                goto label ;
        }
}
```

8.6.3 The for statement

The for statement

for (init-statement condition$_{opt}$; expression$_{opt}$) controlled-statement

is equivalent to

```
{
    init-statement
    while ( condition ) {
        controlled-statement
        expression ;
    }
}
```

except that names declared in the init-statement are in the same declarative region as those declared in the condition, and except that a continue in controlled-statement (not enclosed in another iteration statement) will execute expression before re-evaluating condition. [Note: Thus the first statement specifies initialization for the loop; the condition ([stmt.select]) specifies a test, sequenced before each iteration, such that the loop is exited when the condition becomes false; the expression often specifies incrementing that is sequenced after each iteration. — end note

8.6.4 The range-based for statement

The range-based for statement

for (init-statement$_{opt}$ for-range-declaration : for-range-initializer) controlled-statement

Is equivalent to

```
{
    init-statement_opt
    auto &&range = for-range-initializer ;
    auto begin = begin-expr ;
    auto end = end-expr ;
    for ( ; begin != end; ++begin ) {
        for-range-declaration = * begin ;
        controlled-statement
    }
}
```

where