Proposal for C2y
WG14 N3377

Title: Named Loops Should Name Their Loops: An Improved Syntax For N3355
Author, Affiliation: Erich Keane, NVIDIA
Author, Affiliation: Aaron Ballman, Intel
Proposal Category: Existing Feature
Target Audience: Compiler Implementers, users

Abstract: N3355 introduced a feature for ‘named loops’, which provides for a mechanism to `break`/`continue` to an arbitrary loop or `switch` statement in the hierarchy of the current context. This proposal suggests an alternative syntax that no longer inherits the issues with traditional ‘label’ declarations.

Named Loops Should Name Their Loops: An Alternative Syntax for N3355

Reply-to: Erich Keane (ekeane@nvidia.com) Aaron Ballman (aaron@aaronballman.com)
Document No: N3377
Date: 2024-11-05

Summary of Changes

N3377

- Original Proposal

Introduction and Rationale

At the Minneapolis 2024 meeting of WG14, the committee approved N3355 for C2y, which proposes the useful feature of being able to `continue`/`break` to an arbitrary loop or `switch` in the current hierarchy of scopes. During discussions of this paper the use of the traditional label syntax, historically only used as targets for `goto`, was a contentious point of discussion. While the use of traditional labels is convenient, these labels come with significant historical baggage which forces semantics that don't match other languages and are awkward/cumbersome. The authors of this paper believe we can provide a syntax that maintains the attractive nature of the original proposal with a less confusing syntax for the naming of loops.

Labels are their own Declarations

The first issue with N3355's syntax is that while labels have historically applied to the next statement, this is a historical artifact of the implementation, and not because there is any practical or implicit association between the two before this paper. In fact, the committee has recently made such association no longer necessary in C23, as the null-statement is no longer necessary after a label.

The proposal adds a level of association between the label and the statement that comes immediately after it in a way that does not exist for the common mental model of a label. At least one major implementation doesn't bother to store any such relationship, as it is historically irrelevant.

Doesn't properly imply the target

Traditionally, labels are only targeted by a `goto` statement, and does an excellent job of being clear where execution will resume textually: at the location immediately after the label. However, the N3355 proposal requires that execution resumes in the middle of the next line in the case of

`continue` and `for` loops (since the initializer isn't re-executed), or at the end of the associated statement of whatever construct it is associated with. This is a surprising deviation from its historical use of jumping to the label.

Scoping Issues

As labels are declarations that are typically owned by the function in which they are declared. As a result, there can only be one of each name in a function. This ends up being quite limiting to the feature proposed in N3355 for a few patterns that are natural uses of N3355.

The first is the use in a macro. A common use of macros is for finding/altering a data structure which heavily benefits from early-exit. Traditionally, this early-exit needs to be implemented with some awkward machinations of loops, but with the feature proposed by N3355 these early exits can be implemented in a much more natural and function-like manner. However, the use of labels causes this use to be hamstrung, as a macro containing a naive use of N3355's `break`/`continue` statements cannot be called multiple times in the same function, as they would use the same name. While there are difficult applications of `__LINE__`, `__COUNTER__`, etc that could be used to get around this, it needlessly complicates the definition of what should be a pretty simple macro. Additionally, while there are proposals for scoped/local labels this does not improve the awkwardness in many cases. For example:

C/C++

```
#define UPDATE_MAP(MAP, ELEM, NEWVAL)\
do { \
   MapGroup *BucketItr = MAP.Lists;\
   OUTER: while (BucketItr) {\
       if (BucketItr->Hash == HASH(ELEM)) {\
           MapNode *NodeItr = BucketItr->Nodes;\
           while (NodeItr) {\
               if (NodeItr->Key == ELEM) {\
                   NodeItr->Value = NEWVAL;\
                   break OUTER;\
               }\
               NodeItr = NodeItr->Next;\
           }\
           BucketItr = BucketItr->Next;\
       }\
```

```
   }\
} while(0)
```

That macro, intended to update a value of an element in a contrived hash map, very much benefits from the named loops implementation, as it allows it to early-exit once the element is completed. However, this macro cannot be called more than once in a function, as it would declare `OUTER` twice. While unique names could potentially be generated using token-pasting solutions and `__COUNTER__` (`__LINE__` is also sometimes mentioned here, but it alone cannot solve this, as multiple invocations on the same line are possible), it is very much worth making the 'simple' implementation the right way in this case. With this proposal, using the same name for another loop that is perfectly acceptable (as long as it doesn't shadow), as the names are scoped inside the loop.

A second useful application of loop/`switch` names is to name and jump to a loop based on its depth in the loop, such as `outer` or `inner`. For example, you may wish to use `ROW_LOOP` and `COL_LOOP` when iterating over a 2 dimensional structure. However, without this proposal, the programmer is forced to come up with new names for each loop in a function, despite there being an obvious and natural hierarchy that the compiler needs to enforce for the purposes of checking anyway. The original proposal uses `outer` and `inner` as the name of loops, but with said proposal, those would not be particularly useful names, as they couldn't be used more than once in the same function.

Other Languages

One point the original proposal (N3355) makes is that its syntax for naming loops is used in other languages. However, these languages all lack the historical common use of labels for the purposes of `goto`, and thus can have different semantics for the label. In fact, all of the languages in the original proposal work very much like the proposed syntax here: they are associated directly with the loop or `switch` statement they apply to, and can be repeated in a function. In Rust, the same name can even shadow a loop name of a previous loop in the hierarchy (though this is diagnosed with a warning, and this is not proposed here).

It is clear that the semantics that the author of N3355 was trying to emulate from other languages is not something that can be done with the current syntax, so the syntax proposed in this paper is, though lexically different, semantically the same.

Proposal

This paper proposes to add to the grammar of `for`, `while` and `switch`, an optional identifier to name the loop before the parens. For example: `for OUTER (int i = 0; i < 5;`

`++i)`. We believe that this better associates the name of the loop with the loop, as it is directly part of the statement for the loop or `switch`. This does provide for a mild implementation consideration, as the check for the existence of a loop-label in a `do` loop needs to be deferred until the end of the loop, however this is behavior that existsf or goto labels today. Additionally, this allows us to provide for the semantics of Rust/Java/Javascript (the languages used as examples in N3355). Namely:

The scope of a loop name is the loop itself. This means that the same name can be reused as one would expect multiple times in the same function. Shadowing the name of a loop higher in the current scope is to be ill-formed.

The scope is obviously lexically related to the loop itself. C historically doesn't make this association, and other declarations/statements in the way that may not have semantics or effects (such as pragmas) don't have to interfere.

These names cannot be used as a `goto` target. This reduces the cognitive burden on the programmer to determine whether a label is a jump target from an arbitrary goto or is naming a loop, thus providing for less programmer errors.

After N3355	After N3377 (this paper)

C/C++
`OUTER:`
`for(unsigned x = 0; x < DIM1;`
`++x){`
`  INNER:`
`  for (unsigned y = 0; y < DIM2;`
`++y) {`
`  break OUTER;`
`  }`
`}`
	C/C++
	`for OUTER(unsigned x = 0; x <`
	`DIM1; ++x){`
	`  for INNER(unsigned y = 0; y <`
	`DIM2; ++y) {`
	`  break OUTER;`
	`  }`
	`}`

C/C++
`#define FIND_ELEM(ARRAY2D,`
`ARRAYDIM1, ARRAYDIM2, ELEMID,`
`OUTVAR) while(0) {\`
`OUTER:\`
`for (unsigned x = 0; x <`
`ARRAYDIM1;++x){\`
	C/C++
	`#define FIND_ELEM(ARRAY2D,`
	`ARRAYDIM1, ARRAYDIM2, ELEMID,`
	`OUTVAR) while(0) {\`
	`for OUTER (unsigned x = 0; x <`
	`ARRAYDIM1;++x){\`

```
INNER:\
for(unsigned y = 0; y <
ARRAYDIM2;++y){\
if (ARRAY2D[x][y].id == ELEMID){\
  OUTVAR = &ARRAY2D[x][y];
  break OUTER;
}}}}

...
struct MyType *First, *Second;
FIND_ELEM(MyArray, MyDim1, MyDim2,
FirstId, First);
FIND_ELEM(MyArray, MyDim1, MyDim2,
SecondId, Second);
// ABOVE IS AN ERROR without
__LINE__ & __COLUMN__ magic.
```

```
for INNER (unsigned y = 0; y <
ARRAYDIM2;++y){\
if (ARRAY2D[x][y].id == ELEMID){\
  OUTVAR = &ARRAY2D[x][y];
  break OUTER;
}}}}

...
struct MyType *First, *Second;
FIND_ELEM(MyArray, MyDim1, MyDim2,
FirstId, First);
FIND_ELEM(MyArray, MyDim1, MyDim2,
SecondId, Second);
// ABOVE perfectly fine.
```

Alternate Spellings

This paper is heavily influenced by a Zoom comment on the original discussion which proposed the same lexical location, but with colons on each side of the name (such as `while :OUTER: (condition)`). Additional tokens are perhaps preferential, as it could leave the design space available for alternative tokens in this space, such as a type of `for` loop that provides for automatic parallelism. In addition to the colon-based-delimiting mentioned above, an alternative would be to require the names to start with a single quote, as it does in Rust. If the committee deems it preferential, the below wording can be trivially modified to include a modified grammar for an additional annotation.

Proposed Straw Polls

Preference Polls for any Alternate Spellings that include Token annotations.
Apply N3377 to the working draft for C2y, instructing the Project Editor to apply it after N3355.

Proposed Wording

The proposed changes are based on the latest public draft of C2y, N3301, with the addition of the wording as applied by N3355.

~~Delete the two new paragraphs in 6.8.2 added by N3355~~

~~Delete the new paragraph in 6.8.3 added by N3355~~

Modify 6.8.5p1:

block-name:
identifier

selection-statement:
if (expression) secondary-block
if (expression) secondary-block else secondary-block
switch block-name$_{opt}$ (expression) secondary-block

Insert new paragraph in Semantics before 6.8.5.3p4:
A switch statement with a block name is named by the block name identifier.

Modify 6.8.6p1:
iteration-statement:
while block-name$_{opt}$ (expression) secondary-block
do secondary-block while block-name$_{opt}$ (expression) ;
for block-name$_{opt}$ (expression$_{opt}$; expression$_{opt}$; expression$_{opt}$) secondary-block
for block-name$_{opt}$ (declaration expression$_{opt}$; expression$_{opt}$) secondary-block

Insert new paragraph in Semantics before 6.8.6p3:
An iteration statement with a block name is named by the block name identifier.

Modify 6.8.7.3 (altering the new para from N3355):
A continue statement with an identifier operand shall appear within an iteration statement named by the block name~~label~~ with the corresponding identifier.

Modify 6.8.7.3 (altering the new para from N3355):
If the continue statement has an identifier operand, the jump is to the loop-continuation of the iteration statement named by the block name~~label~~ with the corresponding identifier. Otherwise, the jump is to the loop-continuation of the innermost enclosing iteration statement.

Modify 6.8.7.3 (altering the new example from N3355):

```
~~outer:~~
for outer (int i = 0; i < IK; ++ i) {
  for (int j = 0; j < JK; ++ j) {

    continue;       // jumps to CONT1

    continue outer; // jumps to CONT2

    // CONT1
  }
  // CONT2
}
```

Modify 6.8.7.4 (altering the new para from N3355):
A break statement with an identifier operand shall appear within a switch or iteration statement named by the block name~~label~~ with the corresponding identifier.

Modify 6.8.7.4 (altering the new para from N3355):
If the break statement has an identifier operand, the jump exits the switch or iteration statement named by the block name~~label~~ with the corresponding identifier. Otherwise, the jump exits the innermost enclosing switch or iteration statement.

Modify 6.8.7.4 (altering the new example from N3355):

```
~~outer:~~
for outer (int i = 0; i < IK; ++ i) {
  switch (i) {
      case 1:
      break;     // jumps to CONT1
      case 2:
      break outer; // jumps to CONT2
  }
  // CONT1
}
// CONT2
```

Acknowledgements

We would like to recognize the following people for their help in this work: Niall Douglas.

References

[N3355]
Named Loops, v3: https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3355.htm
[N3301]
C2y Public Draft: https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3301.pdf