

Document number: **N3587**
Date: 2013-03-17
Project: Programming Language C++
Reference: N3485
Reply to: **Alan Talbot**
cpp@alantalbot.com

For Loop Exit Strategies

Abstract

This proposal suggests an enhancement to **for** iteration statements to allow the programmer to specify separate blocks of code that execute on completion of a **for** loop; one for normal termination (when the loop condition is no longer met) and the other for early termination (when the loop is exited with a **break**). This feature would be especially useful in range-based **for** statements.

The Problem

I often find myself writing code that looks something like this:

```
auto i = c.begin();           // Unfortunate that i is required here.
for (; i != c.end(); ++i)
{
    if (some_condition(*i)) break;
    do_something(*i);
}
if (i == c.end())           // Extra test here.
{
    do_stuff();
}
else
{
    do_something_else(*i);
}
```

This is rather a nuisance, involves an unnecessary test, and hoists `i` out into the surrounding scope. This is all rather annoying, but the problem gets much worse with range-based **for** loops. There it is not legal to declare the loop variable outside the **for** statement, so the best I can do would be something like this:

```

something_t last;                // Extra construction here.
for (auto& i : c)
{
    if (some_condition(i))
    {
        last = i;                // Extra copy here
        goto EARLY;
    }
    do_something(i);
}
do_stuff();
goto DONE;
EARLY:
do_something_else(last);
DONE:

```

This is pretty awful. Note the extra construction in the outer scope that requires stating the type. That might not even be possible (if the type isn't default constructible or copyable). This is clearly not an improvement over the conventional **for** version, so the point of range-based **for** has been lost, and in fact I suspect that no one would write code like this. I would simply use the conventional version.

(Of course, in some cases I could eliminate the `last` variable and call `do_something_else` from inside the loop, but that becomes impractical if there are a number of early exit points and `do_something_else` is actually several lines of code rather than a simple function call. And I would still need the **goto**.)

The Solution

What I'd really like to do is have the language provide me a way to catch the two cases, normal and early termination. If the language had a **then** keyword the clearest syntax would probably be:

```

for ( . . . )
{
    // Regular for loop body.
}
then
{
    // Normal termination: the loop condition failed.
}
else
{
    // Early termination: a break was encountered.
}

```

It is probably not practical to add a new keyword, especially **then**, but there *is* a syntax that I think would be fairly natural and would be a pure extension to the language. My proposal is to introduce an **if for** statement:

```

if for ( . . . )
{
    // Regular for loop body.
}
{
    // Normal termination: the loop condition failed.
}
else
{
    // Early termination: a break was encountered.
}

```

The semantics of the **if for** retain the exact sense of the **if** statement: the declared variable remains in scope in both the normal termination and early termination (**else**) substatements, and only one of the termination substatements is executed. Control transfers to the normal termination substatement if and when the loop condition is no longer met (even if the loop substatement is never executed), and to the early termination substatement if the loop exits with a **break**.

One interesting question is whether the normal termination block braces should be required. If they were not, then this example would be legal:

```

if for (int i = 0; i < 10; ++i)
    if (foo(i)) break;           // Loop body statement.
    cout << "no foo";           // Normal termination statement!
else
    cout << "foo at " << i;     // Early termination statement.

```

This seems too confusing for the human reader to parse, so I suggest that braces be required:

```

if for (int i = 0; i < 10; ++i)
    if (foo(i)) break;           // Loop body.
{
    cout << "no foo";           // Normal termination statement,
                                // block required.
}
else
    cout << "foo at " << i;     // Early termination statement.

```

The loop body braces are of course not required, so this scenario is legal and quite reasonable:

```

if for (int i = 0; i < 10; ++i)
    if (foo(i)) break;           // Loop body statement,
    else bar(i);                 // happens to have an else clause.
else
    cout << "foo at " << i; ;    // Early termination statement.

```

Examples

Here is our original range-based example:

```
if for (auto& i : c)
{
    if (some_condition(i)) break;
    do_something(i);
}
{
    do_stuff();
}
else
{
    do_something_else(i);
}
```

The **if for** statement also provides for a graceful multiple break. Suppose I want to iterate over a three-dimensional table and choose a particular cell. Today I would probably do something like this:

```
vector<vector<vector<...>>> table = ...;
for (auto& x : table)
    for (auto& y : x)
        for (auto& z : y)
            if (some_condition(z))
            {
                do_something(z);
                goto DONE;
            }
DONE:
```

This is a little ugly, and gets even worse if you have different exit situations. (I could solve this particular problem by writing a function that returns from the inner loop, but not all such constructions are easily put into a function.) With **if for** you can do this:

```
for (auto& x : table)
    if for (auto& y : x)
        if for (auto& z : y)
        {
            if (some_condition(z))
            {
                do_something(z);
                break;
            }
        }
    else break;
else break;
```

This scales well to more complicated cases since you can either continue or break on either termination condition.

Specifics

I am proposing to add a new **if for** iteration statement to section 6.5. This is not meant to be formal wording. I will provide formal wording in a revision of this proposal.

Conventional if for

```
if for ( for-init-statement conditionopt; expressionopt ) statement
compound-statement
```

```
if for ( for-init-statement conditionopt; expressionopt ) statement
compound-statementopt else statement
```

If and only if the early termination (**else**) substatement is present, then the normal termination substatement may be omitted. If the *for-init-statement* is a declaration, the scope of the name(s) declared includes the normal termination substatement and the early termination substatement.

Example

```
if for (int i = 0; i < 10; ++i)
{
    // Regular for loop substatement.
}
{
    // Normal termination substatement.
    // May be omitted entirely if the else is present.
    // Braces are required.
    // i is in scope and equal to 10.
}
else
{
    // Early termination substatement.
    // May be omitted entirely if the normal termination block is present.
    // Braces are not required.
    // i is in scope and has a value between 0 and 9.
}
```

Range-based if for

```
if for ( for-range-declaration : for-range-initializer ) statement
compound-statement
```

```
if for ( for-range-declaration : for-range-initializer ) statement
compound-statementopt else statement
```

If and only if the early termination (**else**) substatement is present, then the normal termination substatement may be omitted. The scope of the name declared in the *for-range-declaration* includes the normal termination substatement and the early termination substatement. In the normal termination substatement its value is undefined. (It is in scope simply for consistency with conventional **if for** statements.)

Example

```
if for (auto i : v)
{
    // Regular for loop substatement.
}
{
    // Normal termination substatement.
    // May be omitted entirely if the else is present.
    // Braces are required.
    // i is in scope but its value is undefined.
}
else
{
    // Early termination substatement.
    // May be omitted entirely if the normal termination block is present.
    // Braces are not required.
    // i is in scope and has the value it had when the break occurred.
}
```

Acknowledgements

Beman Dawes reviewed an early draft of this proposal and suggested several excellent clarifications. Clark Nelson reviewed the final draft and caught several mistakes.