Null Coalescing Conditional Operator

2014-07-02 N4120

Alexander Bock *alex@gimpel.com*

1 Introduction

Checking if a value is null before use to substitute another value in its place is a very common programming pattern. This can be achieved with the current conditional operator only through duplicating the expression in question or using a temporary variable to prevent side effects from being evaluated twice. Many languages with the conditional operator have implemented a null coalescing operator, most notably C#; it is also already implemented in gcc and clang as a GNU extension. This proposal essentialy entails standardizing the null coalescing conditional operator as implemented in gcc and clang, with the syntax `a ? : b`.

2 Rationale and Implementation

Expressions equivalent to `x ? x : y` are common, and when x is a non-trivial expression, lead to code duplication or require a temporary variable. The standard method of using a temporary variable to prevent multiple evaluations is similar to the proposed operational definition of the new null coalescing conditional syntax.

In effect, the implementation defines

```
a ? : b;
```

to be equivalent to

```
auto&& temp = a;
temp ? temp : b;
```

except that each of the two uses of temp is regarded as having the same value category as a.

Note: Despite being used in **over.built**, `?:` does not appear as a single operator token in **lex.operators**, and it is intended that `a ?: b` also be permittable syntax for this operation.

3 Proposed Standard Wording

To the grammar definition in **expr.cond**, make the second *expression* optional:

conditional-expression:
    logical-or-expression
    logical-or-expression ? expression$_{opt}$ *: assignment-expression*

In **expr.cond**, insert the following as **5.16.7**:

If the second expression is not present, a binary conditional is formed, and the first expression shall be evaluated without performing contextual conversion to `bool`. The result of this evaluation shall then used as both the first and second expressions of a ternary conditional. The third expression of this ternary conditional shall be the unevaluated original third expression. The result the original binary conditional expression shall be the result of evaluating this ternary conditional.