The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

6.5.3.1 Prefix increment and decrement operators

Constraints

postfix operator operand

The operand of the prefix increment or decrement operator shall have qualified or unqualified real or pointer type and shall be a modifiable lvalue. 1081

Commentary

postfix 1081 operator operand

This constraint mirrors that for the postfix forms of these operators.

C++

The use of an operand of type **bool** with the prefix ++ operator is deprecated (5.3.2p1); there is no corresponding entry in Annex D, but the proposed response to C++ DR #145 inserted one. In the case of the decrement operator:

5.3.2p1 *The operand shall not be of type **bool**.*

A C source file containing an instance of the prefix -- operator applied to an operand having type **_Bool** is likely to result in a C++ translator issuing a diagnostic.

Coding Guidelines

symbolic name enumeration set of named constants

Enumerated types are usually thought about in symbolic rather than arithmetic terms. The increment and decrement operators can also be given a symbolic interpretation. They are sometimes thought about in terms of moving on to the next symbolic name in a list. This *move to next* operation relies on the enumeration constants being represented by successive numeric values. While this usage is making use of representation information, there is often a need to step through a series of symbolic names (and C provides no other built-in mechanism), for instance, iterating over the named constants defined by an enumerated type.

relational operators real operands

Dev ??

The operand of a prefix increment or decrement operator may have an enumerated type, provided the enumeration constants defined by that type have successive numeric values.

Semantics

prefix ++ incremented

The value of the operand of the prefix ++ operator is incremented. 1082

Commentary

postfix ++ result

The ordering of this and the following C sentence is the reverse of that specified for the postfix ++ operator.

Common Implementations

The implementation of this operator is usually very straight-forward. A value is loaded into a register, incremented, and then stored back into the original object, leaving the result in the register. Some CISC processors contain instructions that increment the contents of storage directly. Processors that have a stack-based architecture either need to contain store instructions that leave the value on the stack, or be willing to pay the penalty of another load from storage.

Coding Guidelines

Translators have now progressed to the point where the optimizations many of them perform are much more sophisticated than those needed to detect the more verbose sequence of operations equivalent to the prefix ++ operator. The writers of optimizers study existing source code to find out what constructs occur frequently (they don't want to waste time and money implementing optimizations for constructs that rarely occur). However, in existing code it is rare to see an object being incremented (or decremented) without one of these operators being used. Consequently optimizers are unlikely to attempt to transform the C source `i=i+1` into

`++i` (which they might have to do for say Pascal, which has no increment operators requiring optimizers to analyze an expression looking for operations that are effectively increment object). So the assertion that `++i` can be written as `i=i+1` and that it will be optimized by the translator is not guaranteed, even for a highly optimizing translator. However, this is rarely an important issue anyway; the difference in quality of generated machine code rarely has any impact on program performance.

From the coding guidelines perspective, uses of these operators can be grouped into three categories:

1. The only operator in an expression statement. In this context the result returned by the operation is ignored. The statement simply increments/decrements its operand. Use of the prefix, rather than the postfix, form does not follow the pattern seen at the start of most visible source code statement lines—an identifier followed by an operator (see Figure **??**). A reader's scanning of the source looking for objects that are modified will be disrupted by the initial operator. For this reason, use of the postfix form is recommended.

   postfix operator constraint

2. One of the operators in a full expression that contains other operators. It is possible to write the code so that a prefix operator does not occur in the same expression as other operators. The evaluation can be moved back before the containing expression (see the postfix operators for a fuller discussion of this point).

   full expression

   postfix operator constraint

   ```
   ...++i...
   ```

   becomes the equivalent form:

   ```
   i++;
   ...i...
   ```

   The total cognitive effort needed to comprehend the equivalent form may be less than the prefix form, and the peak effort is likely to be less (because the operations may have been split into smaller chunks in serial rather than nested form).

3. The third point is the same as for the postfix operators.

   postfix operator constraint

Cg 1082.1 The prefix operators shall not appear in an expression statement.

1083 The result is the new value of the operand after incrementation.

prefix ++ result

Other Languages

Pascal contains the **succ** operator. This returns the successor value (i.e., it adds one to its operand), but it does not modify the value of an object appearing as its operand.

1084 The expression `++E` is equivalent to `(E+=1)`.

Commentary

The expression `++E` need not be equivalent to `E=E+1` (e.g., the expression `E` may contain a side effect).

C++

C++ lists an exception (5.3.2p1) for the case when `E` has type **`bool`**. This is needed because C++ does not define its boolean type in the same way as C. The behavior of this operator on operands is defined as a special case in C++. The final result is the same as in C.

_Bool large enough to store 0 and 1

1085 See the discussions of additive operators and compound assignment for information on constraints, types, side effects, and conversions and the effects of operations on pointers.

prefix operators see also

Commentary

postfix operators see also

The same references are given for the postfix operators.

C++

5.3.2p1 *[Note: see the discussions of addition (5.7) and assignment operators (5.17) for information on conversions.]*

There is no mention that the conditions described in these clauses also apply to this operator.

The prefix `--` operator is analogous to the prefix `++` operator, except that the value of the operand is decremented. 1086

Commentary

prefix ++ 1082 incremented

postfix -- analogous to ++

The same Commentary and Coding Guidelines' issues also apply. See the discussion elsewhere for cases where the affects are not analogous.

C++

The prefix `--` operator is not analogous to the prefix `++` operator in that its operand may not have type **`bool`**.

Other Languages

Pascal contains the **`pred`** reserved identifier. This returns the predecessor value, but does not modify the value of its operand.

Coding Guidelines

prefix 1082.1 in expression statement

The guideline recommendation for the prefix `++` operator has been worded to apply to either operator.

Forward references: additive operators (6.5.6), compound assignment (6.5.16.2). 1087

References