The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

6.5.11 Bitwise exclusive OR operator

```
exclusive-OR-expression:
                  AND-expression
                  exclusive-OR-expression ^ AND-expression
```

1240

Commentary

The ^ operator can be replaced by a sequence of equivalent bitwise operators; for instance, `x ^ y` can be written `(x & (~y)) | ((~x) & y)`. However, most processors contain a bitwise exclusive-OR instruction and thus this operator is included in C. Being able to represent a bitwise operator using an equivalent sequence of other operators is not necessarily an argument against that operator being included in C. It is possible to represent any boolean expression using a sequence of NAND (*not and*, e.g., `!(x & y)`) operators; for instance, `!x` becomes `x NAND x`, `x & y` becomes `(x NAND y) NAND (x NAND y)`, and so on. Although this equivalence may be of practical use in hardware design (where use of mass-produced circuits performing a single boolean operation can reduce costs), it is not applicable to software.

Other Languages

Languages that support bitwise operations usually support an exclusive-OR operator. The keyword **`xor`** is sometimes used to denote this operator. The ^ character is used as a token to indicate pointer indirection in Pascal and Ada.

Coding Guidelines

logical-OR-expression syntax

The exclusive-OR operator is not encountered in everyday life. There is no English word or phrase that expresses its semantics. The everyday usage discussed in the subclause on the logical-OR operator is based on selecting between two alternatives (e.g., *either one or the other* which only deals with two of the four possible combinations of operand values). Because of the lack of everyday usage, and because it does not occur often within C source (compared with bitwise-AND and bitwise-OR (see Table **??**)) it is to be expected that developers will have to expend more effort in comprehending the consequences of this operator when it is encountered in source code.

The greater cognitive cost associated with use of the exclusive-OR operator is not sufficient to recommend against its use. Developers need to be able to make use of an alternative that has a lower cost. While the behavior of the exclusive-OR operator can be obtained by various combinations of other bitwise operators, it seems unlikely that the cost of comprehending any of these sequences of operators will be less than that of the operator they replace. If the exclusive-OR operator appears within a more complicated boolean expression it may be possible to rewrite that expression in an alternative form. Rewriting an expression can increase the cognitive effort needed for readers to map between source code and the application domain (i.e., if the conditions in the application domain are naturally expressed in terms of a sequence of operators that include exclusive-OR). The cost/benefit of performing such a rewrite needs to be considered on a case by case basis.

boolean ?? expression minimize effort

Example

The ^ operator is equivalent to the binary + operator if its operands do not have any set bits in common. This property can be used to perform simultaneous addition on eight digits represented in packed BCD[1] (i.e., four bits per digit).

```
int BCD_add(a, b)
{
int t1 = a + 0x06666666,
    t2 = t1 + b,
    t3 = t1 ^ b,
    t4 = t2 ^ t3,
    t5 = ~t4 & 0x11111110,
    t6 = (t5 >> 2) | (t5 >> 3);
return t2 - t6;
}
```

Usage

The ^ operator represents 1.2% of all occurrences of bitwise operators in the visible source of the .c files.

Constraints

1241 Each of the operands shall have integer type.

Commentary

The discussion for the various subsections is the same as those for the bitwise AND operator.

& binary operand type

Semantics

1242 The usual arithmetic conversions are performed on the operands.

^ operands converted

Commentary

The discussion in the various subsections is the same as that for the bitwise AND operator.

& binary operands converted

1243 The result of the ^ operator is the bitwise exclusive OR of the operands (that is, each bit in the result is set if and only if exactly one of the corresponding bits in the converted operands is set).

Commentary

This information is usually expressed in tabular form.

	0	1
0	0	1
1	1	0

Common Implementations

The Unisys A Series[2] uses signed magnitude representation. If the operands have an unsigned type, the sign bit is not affected by the bitwise complement operator. If the operands have a signed type, the sign bit does take part in the bitwise complement operation.

The bitwise exclusive-OR instruction is sometimes generated, by optimizers, to swap the contents of two registers, without using a temporary register (as shown in the Example below).

Coding Guidelines

Although the result of the bitwise exclusive-OR operator is the common type derived from the usual arithmetic conversions, for the purpose of these guideline recommendations its role is the same as that of its operands.

usual arithmetic conversions
object role

Example

```
#define SWAP(x, y) (x=(x ^ y), y=(x ^ y), x=(x ^ y))
#define UNDEFINED_SWAP(x, y) (x ^= y ^= x ^= y) /* Requires right to left evaluation. */
```

References

1. D. W. Jones. BCD arithmetic, a tutorial. www.cs.uiowa.edu/~jones, July 1999.

2. Unisys Corporation. *C Programming Reference Manual, Volume 1: Basic Implementation*. Unisys Corporation, 8600 2268-203 edition, 1998.