The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

6.5.3 Unary operators

unary-expression syntax


```
unary-expression:
                postfix-expression
                ++ unary-expression
                -- unary-expression
                unary-operator cast-expression
                sizeof unary-expression
                sizeof ( type-name )
unary-operator: one of
                &  *  +  -  ~  !
```

Commentary

cast-expression syntax

Note that the operand of unary-operator is a *cast-expression*, not a *unary-expression*. A unary operator usually refers to an operator that takes a single argument. Technically all of the operators listed here, plus the postfix increment and decrement operators, could be considered as being unary operators.

Rationale

Unary plus was adopted by the C89 Committee from several implementations, for symmetry with unary minus.

Other Languages

multiplicative-expression syntax

Some languages (i.e., Ada and Pascal) specify the unary operators to have lower precedence than the multiplicative operators; for instance, `-x/y` is equivalent to `-(x/y)` in Ada, but `(-x)/y` in C. Most languages call all operators that take a single-operand unary operators.

Languages that support the unary `+` operator include Ada, Fortran, and Pascal. Some languages use the keyword **NOT** rather than `!`. In the case of Cobol this keyword can also appear to the left of an operator, indicating negation of the operator (i.e., **NOT** `<` meaning not less than).

Coding Guidelines

Coding guidelines need to be careful in their use of the term *unary operator*. Its meaning, as developers understand it, may be different from its actual definition in C. The operators in a *unary-expression* occur to the left of the operand. The only situation where a developer's incorrect assumption about precedence relationships might lead to a difference between predicted and actual behavior is when a postfix operator occurs immediately to the right of the *unary-expression*.

Dev ??

Except when *sizeof (type-name)* is immediately followed visually by a token having the lexical form of an additive operator, if a *unary-expression* is not immediately followed by a postfix operator it need not be parenthesized.

macro definition ?? expression

Although the expression `sizeof (int)-1` may not occur in the visible source code, it could easily occur as the result of macro replacement of the operand of the **sizeof** operator. This is one of the reasons behind the guideline recommendation specifying the parenthesizing of macro bodies (without parentheses the expression is equivalent to `(sizeof(int))-1`).

Example

```
1  struct s {
2            int x;
3            };
4  struct s *a;
5  int x;
6
```

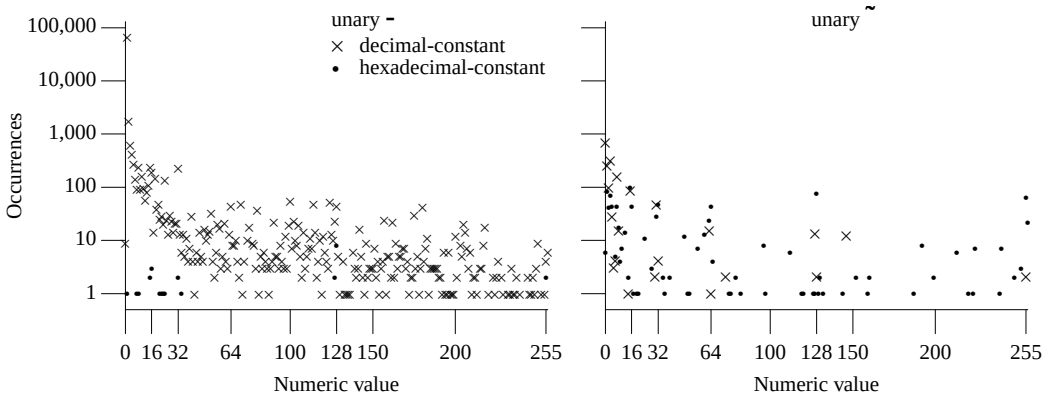

Figure 1080.1: Number of *integer-constant*s having a given value appearing as the operand of the unary minus and unary ~ operators. Based on the visible form of the `.c` files.

```
7   void f(void)
8   {
9   x<-a->x;
10  x<--a->x;
11  x<- --a->x;
12  x<- - --a->x;
13
14  sizeof(long)-3; /* Could be mistaken for sizeof a cast-expression. */
15  (sizeof(long))-3;
16  sizeof((long)-3);
17  }
```

Usage

See the Usage section of *postfix-expression* for ++ and -- digraph percentages.

postfix-expression syntax

Table 1080.1: Common token pairs involving **sizeof**, *unary-operator*, prefix ++, or prefix -- (as a percentage of all occurrences of each token). Based on the visible form of the `.c` files.

Token Sequence	% Occurrence of First Token	% Occurrence of Second Token	Token Sequence	% Occurrence of First Token	% Occurrence of Second Token
! defined	2.0	16.7	! (	14.5	0.5
*v --v	0.3	7.8	-v identifier	30.2	0.4
-v *floating-constant*	0.3	6.7	*v (	9.0	0.4
*v ++v	0.5	6.3	~ *integer-constant*	20.1	0.2
! --v	0.2	4.8	++v identifier	97.3	0.1
-v *integer-constant*	69.0	4.1	~ identifier	56.3	0.1
&v identifier	96.1	1.9	~ (	23.4	0.1
sizeof (	97.5	1.8	+v *integer-constant*	49.0	0.0
*v identifier	86.8	1.0	--v identifier	97.1	0.0
! identifier	81.9	0.8			

Table 1080.2: Occurrence of the *unary-operator*s, prefix `++`, and prefix `--` having particular operand types (as a percentage of all occurrences of the particular operator; an _ prefix indicates a literal operand). Based on the translated form of this book's benchmark programs.

Operator	Type	%	Operator	Type	%	Operator	Type	%
`-v`	`_int`	96.0	`~`	`unsigned long`	6.8	`!`	`_long`	2.7
`*v`	ptr-to	95.3	`&v`	`int`	6.2	`~`	`unsigned char`	2.5
`+v`	`_int`	72.2	`~`	`unsigned int`	6.0	`&v`	`unsigned char`	2.4
`--v`	`int`	54.7	`+v`	`unsigned long`	5.6	`!`	`unsigned long`	2.1
`!`	`int`	50.0	`+v`	`long`	5.6	`~`	`long`	2.0
`~`	`_int`	49.3	`+v`	`float`	5.6	`++v`	`unsigned char`	1.9
`&v`	other-types	45.1	`!`	other-types	5.6	`~`	`_unsigned long`	1.7
`++v`	`int`	43.8	`++v`	`unsigned long`	5.2	`~`	`_unsigned int`	1.7
`++v`	ptr-to	33.3	`&v`	`struct *`	4.9	`!`	`unsigned char`	1.6
`~`	`int`	28.5	`--v`	`unsigned long`	4.7	`~`	other-types	1.6
`--v`	`unsigned int`	22.1	`!`	`unsigned int`	4.7	`-v`	`_double`	1.4
`!`	ptr-to	20.1	`*v`	fnptr-to	4.1	`-v`	other-types	1.3
`--v`	ptr-to	14.6	`&v`	`unsigned long`	4.0	`++v`	`long`	1.2
`&v`	`struct`	13.9	`--v`	other-types	4.0	`-v`	`int`	1.2
`&v`	`char`	13.1	`&v`	`long`	3.4	`!`	`_int`	1.2
`++v`	`unsigned int`	12.6	`&v`	`unsigned int`	3.0	`++v`	`unsigned short`	1.1
`+v`	`int`	11.1	`&v`	`unsigned short`	2.9	`&v`	`char *`	1.1
`!`	`char`	9.2	`!`	`enum`	2.9			

References