The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

6.5.2 Postfix operators

postfix-expression syntax


```
postfix-expression:
                primary-expression
                postfix-expression [ expression ]
                postfix-expression ( argument-expression-list_opt )
                postfix-expression .  identifier
                postfix-expression -> identifier
                postfix-expression ++
                postfix-expression --
                ( type-name ) { initializer-list }
                ( type-name ) { initializer-list , }
argument-expression-list:
                assignment-expression
                argument-expression-list , assignment-expression
```

Commentary

The token pairs `[ ]` and `( )` are not commonly thought of as being operators.

C90

Support for the forms (compound literals):

```
( type-name ) { initializer-list }
( type-name ) { initializer-list , }
```

is new in C99.

C++

Support for the forms (compound literals):

```
( type-name ) { initializer-list }
( type-name ) { initializer-list , }
```

is new in C99 and is not specified in the C++ Standard.

Other Languages

Many languages do not treat the array subscript (`[]`), structure and union member accesses (`.` and `->`), or function calls (`()`) as operators. They are often included as part of the syntax (as punctuators) for primary expressions. The syntax used in C for these operators is identical or very similar to that often used by other languages containing the same construct.

Many languages support the use of comma-separated expressions within an array index expression. Each expression is used to indicate the element of a different dimension— for instance, the C form `a[i][j]` can be written as `a[i, j]`.

Some languages use parentheses, `()`, to indicate an array subscript. The rationale given for using parentheses in Ada[1] is based on the principle of *uniform referents*— a change in the method (i.e., a function call or array index) of evaluating an operand does not require a change of syntax.

Cobol uses the keyword `OF` to indicate member selection. Fortran 95 uses the `%` symbol to represent the `->` operator.

Perl allows the parentheses around the arguments in a function call to be omitted if there is a declaration of that function visible.

Common Implementations

The question of whether using the postfix or prefix form of the ++ and -- operators results in the more efficient machine code crops up regularly in developer discussions. The answer can depend on the processor instruction set, the translator being used, and the context in which the expression occurs. It is outside the scope of this book to give minor efficiency advice, or to list all the permutations of possible code sequences that could be generated for specific operators.

Coding Guidelines

There is one set of cases where developers sometime confuse the order in which prefix operators and *unary-operators* are applied to their operand. The expression `*p++` is sometimes assumed to be equivalent to `(*p)++` rather than `*(p++)`. A similar assumption can also be seen for the postfix operator `--`. The guideline recommendation dealing with the use of parenthesis is applicable here.

unary-expression syntax

Dev ??

A postfix-expression denoting an array subscript, function call, member access, or compound literal need not be parenthesized.

Dev ??

Provided the result of a postfix-expression, denoting a postfix increment or postfix decrement operation, is not operated on by a unary operator it need not be parenthesized.

Example

```
1  struct s {
2           struct s *x;
3           };
4  struct s *a,
5           *x;
6
7  void f(void)
8  {
9  a>x;
10 a->x;
11 a-->x;
12 a--->x;
13 }
```

Table 985.1: Occurrence of postfix operators having particular operand types (as a percentage of all occurrences of each operator, with **[** denoting array subscripting). Based on the translated form of this book's benchmark programs.

Operator	Type	%	Operator	Type	%
v++	**int**	54.0	**[**	**unsigned char**	5.1
v--	**int**	52.5	**[**	other-types	4.7
[	*****	38.0	**[**	**int**	4.1
v++	*****	25.7	**v++**	**unsigned long**	3.1
v--	**long**	15.9	**v--**	**unsigned short**	2.7
[	**struct**	14.5	**v--**	**unsigned char**	2.6
v++	**unsigned int**	13.3	**[**	**const char**	2.4
[	**float**	12.0	**[**	**unsigned long**	1.2
v--	**unsigned int**	11.5	**v++**	**long**	1.1
[	**union**	10.2	**[**	**unsigned int**	1.1
v--	*****	7.1	**v++**	**unsigned short**	1.0
[	**char**	6.8	**v++**	**unsigned char**	1.0
v--	**unsigned long**	6.1	**v--**	**short**	1.0

Table 985.2: Common token pairs involving `.`, `->`, `++`, or `--` (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
identifier `->`	9.8	97.5	`v++ )`	41.4	1.4
identifier `v++`	0.9	96.9	`v++ ;`	39.9	1.4
identifier `v--`	0.1	96.1	`v++ ]`	4.6	1.3
identifier `.`	3.6	83.8	`v++ =`	7.6	0.7
`] .`	20.3	15.4	`v-- ;`	58.4	0.3
`->` identifier	100.0	10.1	`v-- )`	29.1	0.1
`.` identifier	100.0	4.2			

footnote 77

77) Thus, an undeclared identifier is a violation of the syntax. 986

Commentary

This fact was not explicitly pointed out in the C90 Standard, which led some people to believe that the behavior was undefined. The response to DR #163 specified the order in which requirements, given in the standard, needed to be read (to deduce the intended behavior).

C++

The C++ Standard does not explicitly point out this consequence.

Other Languages

Some languages regard a reference to an undeclared identifier as a violation of the language semantics that is required to be diagnosed by an implementation.

Common Implementations

Most implementations treat undeclared identifiers as primary expressions. It is during the subsequent semantic processing where the diagnostic associated with this violation is generated. The diagnostic often points out that the identifier has not been declared rather than saying anything about syntax.

References

1. J. Ichbiah, J. Barnes, R. Firth, and M. Woodger. *Rationale for the Design of the Ada Programming Language*. Cambridge University Press, 1991.