PRE02-C. Macro replacement lists should be parenthesized

Macro replacement lists should be parenthesized to protect any lower-precedence operators from the surrounding expression. See also PRE00-C. Prefer inline or static functions to function-like macros and PRE01-C. Use parentheses within macros around parameter names.

Noncompliant Code Example

This `CUBE()` macro definition is noncompliant because it fails to parenthesize the replacement list:

```
#define CUBE(X) (X) * (X) * (X)
int i = 3;
int a = 81 / CUBE(i);
```

As a result, the invocation

```
int a = 81 / CUBE(i);
```

expands to

```
int a = 81 / i * i * i;
```

which evaluates as

```
int a = ((81 / i) * i) * i);  /* Evaluates to 243 */
```

which is not the desired behavior.

Compliant Solution

With its replacement list parenthesized, the `CUBE()` macro expands correctly for this type of invocation.

```
#define CUBE(X) ((X) * (X) * (X))
int i = 3;
int a = 81 / CUBE(i);
```

This compliant solution violates PRE00-C. Prefer inline or static functions to function-like macros. Consequently, this solution would be better implemented as an inline function.

Noncompliant Code Example

In this noncompliant code example, `END_OF_FILE` is defined as `-1`. The macro replacement list consists of a unary negation operator followed by an integer literal 1:

```
#define END_OF_FILE -1
/* ... */
if (getchar() END_OF_FILE) {
   /* ... */
}
```

In this example, the programmer has mistakenly omitted the comparison operator from the conditional statement, which should be `getchar() != END_OF_FILE`. (See void MSC02-C. Avoid errors of omission.) After macro expansion, the conditional expression is incorrectly evaluated as a binary operation: `getchar()-1`. This statement is syntactically correct, even though it is certainly not what the programmer intended. Note that this example also violates DCL00-C. Const-qualify immutable objects.

Parenthesizing the `-1` in the declaration of `END_OF_FILE` ensures that the macro expansion is evaluated correctly:

```
#define END_OF_FILE (-1)
```

Once this modification is made, the noncompliant code example no longer compiles because the macro expansion results in the conditional expression `get char() (-1)`, which is no longer syntactically valid. Note that there must be a space after `END_OF_FILE` because, otherwise, it becomes a function-like macro (and one that is incorrectly formed because 1 cannot be a formal parameter).

Compliant Solution

In this compliant solution, the macro definition is replaced with an enumeration constant in compliance with DCL00-C. Const-qualify immutable objects. In addition, because `EOF` is a reserved macro defined in the `<stdio.h>` header, the compliant solution must also use a different indentifier in order to comply with DCL37-C. Do not declare or define a reserved identifier.

```
enum { END_OF_FILE = -1 };
/* ... */
if (getchar() != END_OF_FILE) {
   /* ... */
}
```

Exceptions

PRE02-C-EX1: A macro that expands to a single identifier or function call is not affected by the precedence of any operators in the surrounding expression, so its replacement list need not be parenthesized.

```
#define MY_PID getpid()
```

PRE02-C-EX2: A macro that expands to an array reference using the array-subscript operator `[]`, or an expression designating a member of a structure or union object using either the member-access `.` or `->` operators is not affected by the precedence of any operators in the surrounding expression, so its replacement list need not be parenthesized.

```
#define NEXT_FREE block->next_free
#define CID customer_record.account.cid
#define TOOFAR array[MAX_ARRAY_SIZE]
```

Risk Assessment

Failing to parenthesize macro replacement lists can cause unexpected results.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
PRE02-C	Medium	Probable	Low	P12	L1

Automated Detection

Tool	Version	Checker	Description
Axivion Bauhaus Suite	6.9.0	**CertC-PRE02**	
CodeSonar	5.2p0	**LANG.PREPROC.MACROEND**	Macro Does Not End With) or }
		LANG.PREPROC.MACROSTART	Macro Does Not Start With (or {
ECLAIR	1.2	**CC2.PRE02**	Fully implemented
Klocwork	2018	**MISRA.DEFINE.BADEXP**	
LDRA tool suite	9.7.1	**77 S**	Fully implemented
Parasoft C/C++test	10.4.2	**CERT_C-PRE02-a**	Enclose in parentheses whole definition of a function-like macro
PRQA QA-C	9.7	**3409**	Fully implemented

Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the CERT website.

Related Guidelines

SEI CERT C++ Coding Standard	VOID PRE02-CPP. Macro replacement lists should be parenthesized
ISO/IEC TR 24772:2013	Operator Precedence/Order of Evaluation [JCW] Pre-processor Directives [NMP]

Bibliography

[Plum 1985]	Rule 1-1
[Summit 2005]	Question 10.1