PRE10-C. Wrap multistatement macros in a do-while loop

Macros are often used to execute a sequence of multiple statements as a group.

Inline functions are, in general, more suitable for this task (see PRE00-C. Prefer inline or static functions to function-like macros). Occasionally, however, they are not feasible (when macros are expected to operate on variables of different types, for example).

When multiple statements are used in a macro, they should be bound together in a `do-while` loop syntactically, so the macro can appear safely inside `if` clauses or other places that expect a single statement or a statement block. (Alternatively, when an `if`, `for`, or `while` statement uses braces even for a single body statement, then multiple statements in a macro will expand correctly even without a `do-while` loop (see EXP19-C. Use braces for the body of an if, for, or while statement).

Noncompliant Code Example

This noncompliant code example contains multiple, unbound statements:

```
/*
 * Swaps two values and requires
 * tmp variable to be defined.
 */
#define SWAP(x, y) \
  tmp = x; \
  x = y; \
  y = tmp
```

This macro expands correctly in a normal sequence of statements but not as the `then` clause in an `if` statement:

```
int x, y, z, tmp;
if (z == 0)
  SWAP(x, y);
```

It expands to the following, which is certainly not what the programmer intended:

```
int x, y, z, tmp;
if (z == 0)
  tmp = x;
x = y;
y = tmp;
```

Furthermore, this macro violates PRE02-C. Macro replacement lists should be parenthesized.

Noncompliant Code Example

This noncompliant code example parenthesizes its macro arguments, but inadequately bounds multiple statements:

```
/*
 * Swaps two values and requires
 * tmp variable to be defined.
 */
#define SWAP(x, y) { tmp = (x); (x) = (y); (y) = tmp; }
```

This macro fails to expand correctly in some case, such as the following example, which is meant to be an `if` statement with two branches:

```
if (x > y)
  SWAP(x, y);          /* Branch 1 */
else
  do_something();     /* Branch 2 */
```

Following macro expansion, however, this code is interpreted as an `if` statement with only one branch:

```
if (x > y) { /* Single-branch if-statement!!! */

  tmp = x;   /* The one and only branch consists */
  x = y;     /* of the block. */
  y = tmp;
}
;            /* Empty statement */
else         /* ERROR!!! "parse error before else" */
  do_something();
```

The problem is the semicolon (`;`) following the block.

Compliant Solution

Wrapping the macro inside a `do-while` loop mitigates the problem:

```
/*
 * Swaps two values and requires
 * tmp variable to be defined.
 */
#define SWAP(x, y) \
  do { \
    tmp = (x); \
    (x) = (y); \
    (y) = tmp; } \
  while (0)
```

The `do-while` loop will always be executed exactly once.

This macro still violates the recommendation PRE12-C. Do not define unsafe macros, because both macro arguments are evaluated twice. It is expected that the arguments are simple lvalues.

Risk Assessment

Improperly wrapped statement macros can result in unexpected and difficult to diagnose behavior.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
PRE10-C	Medium	Probable	Low	P12	L1

Automated Detection

Tool	Version	Checker	Description
Axivion Bauhaus Suite	6.9.0	**CertC-PRE10**	
Klocwork	2018	**MISRA.DEFINE.BADEXP**	
LDRA tool suite	9.7.1	**79 S**	Enhanced enforcement
PRQA QA-C	9.7	**3412, 3458**	Fully implemented

Related Vulnerabilities

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

Related Guidelines

ISO/IEC TR 24772:2013	Pre-processor Directives [NMP]

Bibliography

Linux Kernel Newbies FAQ	FAQ/DoWhile0