RULES FOR SECURE C LANGUAGE SOFTWARE DEVELOPMENT

ANSSI GUIDELINES

ANSSI-PA-073
24/03/2022

TARGETED AUDIENCE:

Developers " E N J O J T*US STBFUDPVSSTJUZ N B O BHTFSST

Information

8 B S O J O H

This document, written by ANSSI, the French National Information Security Agency, presents the **"Rules for secure C language software development"**. It is freely available at r r r X b b B X ; Q m p.X 7 ` f 2 M f

It is an original creation from ANSSI and it is placed under the "Open Licence v2.0" published by the Etalab mission [ETALAB].

According to the Open Licence v2.0, this guide can be freely reused, subject to mentionning its paternity (source and date of last update). Reuse means the right to communicate, distribute, redistribute, publish, transmit, reproduce, copy, adapt, modify, extract, transform and use, including for commercial purposes

These recommendations are provided as is and are related to threats known at the publication time. Considering the information systems diversity, ANSSI cannot guarantee direct application of these recommendations on targeted information systems. Applying the following recommendations shall be, at first, validated by IT administrators and/or IT security managers.

This document is a courtesy translation of the initial French document **"Règles de programmation pour le développement sécurisé de logiciels en langage C"**, available at r r r X b b B X ; Q. m p.X 7 `. In case of conflicts between these two documents, the latter is considered as the only reference.

% P D V N F O U D I B O H F M P H

7 & 3 4 * 0	% " 5 &	$) " / (&
1.4	24/03/2022	First English version

Contents

1
Introduction

Œ ‰ • … ’ ‹ ™ … ‹ ‰ “ ® ‰ – — ‹ – ‰ … ˜ Š – ‰ ‰ ˆ “ ‘ ˜ “ ˆ ‰ š ‰ • “ ” ‰ – —
“ ™ — “ – – • — • • ‡ “ ’ — ˜ – ™ ‡ ˜ • “ ’ — ˜ Œ … ˜ ‘ … • • ’ ˜ – “ ˆ ™ ‡ ‰ ‰ – – “
• … ’ ‹ ™ … ‹ ‰ — ˜ … ’ ˆ … – ˆ ˆ “ ‰ — ’ “ ˜ — ” ‰ ‡ • Š • … • • ˜ Œ ‰ ˆ ‰ —
— “ ‘ ‰ – ‰ ‘ … • ’ ™ ’ ˆ ‰ ± ’ ‰ ˆ [1]“À ™‰ š ‰‰ “ ‡ ‰ ‰ ˆ “ Š ‡ “ ‘ ” • • ‰ – — ½
“ ” ‰ – … ˜ • ’ ‹ — • — ˜ ‰ ‘ — … – ‰ ˜ Œ ‰ – ‰ Š “ – ‰ Š – ‰ ‰ ˜ “ ‘ … • ‰ ˜ Œ

‰ — ˜ – • ‡ ˜ • “ ’ — ‘ ™ — ˜ ˜ Œ ‰ – ‰ Š “ – ‰ † ‰ — ‰ ˜ “ ’ ˜ Œ ‰ ™ — ‰ “ Š
š … – • “ ™ — – • — • • “ – ’ “ ’ Â ” “ – ˜ … † • ‰ ‡ “ ’ — ˜ – ™ ‡ ˜ • “ ’ — … ’ ˆ ˜ “ • •
Œ ‰ – ‰ — ˜ – • ‡ ˜ • “ ’ — ˆ ‰ ± ’ ‰ ˆ • ’ ˜ Œ • — ‹ ™ • ˆ ‰ … – ‰ • ’ ˜ ‰ ’ ˆ ‰ ˆ
‘ “ – ‰ — ‰ ‡ ™ – ‰ ½ — … Š ‰ – ½ ‘ “ – ‰ – “ † ™ — ˜ — “ [1] › … – ‰ ½ … ’ ˆ
Š – “ ‘ “ ’ ‰ — • — ˜ ‰ ‘ ˜ “ … ’ “ ˜ Œ ‰ – ½ › Œ ‰ ˜ Œ ‰ – ˜ • ” ‰ “ – ‰ ‘ † ‰

This guide defines a set of rules, recommendations and good practices dedicated to secure C language development. ’ ˜ Œ • — ˆ “ ‡ ™ ‘ ‰ ’ ˜ ½ › ‰ ‡ ™ – – ‰ ’ ˜ • • – ‰ — ˜ – • ‡ ˜ [2] ™ … – ’ ˆ — ‰ • š ‰ —
½ › Œ • ‡ Œ … – ‰ ˜ Œ ‰ ‘ “ — ˜ › • ˆ ‰ • • ™ — ‰ ˆ À

When a rule is directly associated with one specific standard, this is clearly indicated to avoid any confusion. If not specified, both standards are concerned.

3 V M F

A – ™ must … • › be respected; no exceptions are tolerated.

3 F D P N N F O E B U J P O

A – ‰ ‡ “ ‘ ‘ ‰ ’ ˆ should be respected except in certain exceptional cases, which implies a clear and precise justification from the developer. Recommendations are abbreviated to “RECO”.

This guide also contains good practices. These are often somewhat more subjective points like coding conventions, such as the indentation of the code, for example.

1. These concepts are defined on page 9.
2. N.B. the C90 standard is also referred to as C89 by the C community.

(PPE QSBDUJDF

The ‹ “ “ ^ ” – … ‡ defined in this guide are highly recommended, but they can be replaced by those already in place in the developer’s organisation or development team if equivalent rules exist.

This guide has various objectives:

- increasing the security, quality and reliability of the source code produced, by identifying bad or dangerous programming practices;
- facilitating the analysis of the source code during peer review or using static analysis tools;
- establishing a level of confidence in the security, reliability and robustness of a development;
- making software maintenance easier, as well as the addition of features.

The idea of this guide is not to reinvent the wheel, but rather to use existing documents (methodological guides, language standard references, ‰ ƴ ‡ extract, modify and specify a set of recommendations for the secure development of the C language. The reference documents used are as follows:

- MISRA-C: 2012 Guidelines for the use of the C language in critical systems [Misra2012],
- C ANSI 90 [AnsiC90],
- C ANSI 99 [AnsiC99],
- GCC: Reference Documentation [GccRef],
- CLANG’S Documentation [ClangRef],
- SEI CERT C Coding Standard [Cert],
- ISO 17961 C Secure Coding Rules [IsoSecu],
- CWE MITRE Common Weakness Enumeration [Cwe].

This guide is not aimed at any particular application field and is not intended to replace the development constraints imposed by any normative context (automotive, aeronautics, critical systems, ‰ ƴ. ‡ ıà aim is to address precisely those secure C developments not covered by these normative constraints.

2
Coding convention

Above all, any development project of any kind must follow a clear, precise and documented development convention. This development convention must be known to all developers and applied systematically.

Each developer has his or hers own programming habits, code layout and variable naming. However, when producing software, these different programming habits between developers result in a heterogeneous set of source files, which are more difficult to audit and maintain.

36-&

36-&±"QQMJDBUJPOPGDMFBSBOEF

Coding conventions must be defined and documented at the software project level. These conventions must define at least the following points: encoding of source files, code layout and indentation, standard types to be used, naming (libraries, files, functions, types, variables, ‰ʃ, ‡ocumentation format.
These conventions must be followed by each developer.

This rule regarding development conventions is certainly obvious and the aim here is not to impose, for example, a choice of variable naming (such as snake-case versus camel-case), but to ensure that a choice has indeed been made at the beginning of the development project and that it is explicit.

Appendix E provides examples of coding conventions that can be used or adapted as required.

[Misra2012] Dir. 4.5 Identifiers in the same name space with overlapping visibility should be typographically unambiguous.
[Cert] Rec. DCL02-C Use visually distinct identifiers.
[IsoSecu] Using Identifiers that are reserved for the implementation [resident].

3
Undefined and unspecified behaviours

As the remainder of this guide will frequently make use of the concepts of ™ ' ˆ ‰ ± ' ‰ ˆ † ‰ and … š • “ ™ – ™ ' — " ‰ ‡ • ± ‰ ˆ, we recall them here below.

6 O E F ¾ O F E C F I B W J P V S

An ™ ' ˆ ‰ ± ' ‰ ˆ † ‰ is a behaviour for which nothing is imposed by the C standard, and which follows an error in the construction of the program, a non-portable construction or an incorrect use of data. An example is a — • ‹ ' ‰ ˆ • ' ˜ ‰ ‹ ‰ – “ š ‰ – · “ ›

6 O T Q F D J ¾ F E C F I B W J P V S

An ™ ' — " ‰ ‡ • ± ‰ ˆ is a behaviour for which the C standard provides at least two alternative behaviours that are accepted, but none of which are imposed. An example is the order of evaluation of the operations # and ## during the substitution of a macro.

* O G P S N B U J P O

The exhaustive list of all the unspecified behaviours and all the undefined behaviours is available in appendices G and J of standards C90 [AnsiC90] and C99 [AnsiC99].

This guide only considers a C coding environment compliant with the C90 or C99 standards.

* O G P S N B U J P O

Concurrent programming is not covered in this version of the guide, but will be covered in a later version.

3 6 - &

3 6 - & ± 0 O M Z $ D P E J O H J O B D D P S E B O

No violation of C constraints and syntax as defined in the C90 or C99 standards is authorised.

3 F G F S F O D F T

[Misra2012] Rule 1.1 The program shall contain no violations of the standard C syntax and ‡ “ ' Â — ˜ – . and shall not exceed the implementation translation limits.
[Misra2012] Rule 1.2 Language extensions should not be used.
[Misra2012] Rule 1.3 There shall be no occurrence of undefined or critical unspecified behaviour.
[Cwe] CWE-710 Improper Adherence to coding standard.

[Cwe] CWE-758 Reliance on undefined, unspecified or implementation-defined behavior.

4
Preprocessor and macros

*ODMVTJPO PG UIF OFD

Only the necessary header files need to be included, but some additional rules need to be observed. When a header file itself includes other header files, these declarations will be propagated to all source or header files that include this first file. This C language mechanism results in the cascaded inclusion of header files and declarations.

If the inclusion of header files is not minimised, this generates unnecessary dependencies, increases compilation time, and makes the subsequent analysis of the code more complex (whether manual or tool-based). In order to reduce dependencies and unnecessary propagation of declarations, header file inclusions should be made in a ".c" file and not in a ".h" header file. However, in some cases, such as typical type definition, the inclusion of header files from the standard library (such as bi//27 and bi/BMJI2) in another header file is justifiable.

3&$0..&/%"5*0/ ± -JNJU BOE KVTUJG IFBEFS ¾MF

Header files should be included as needed during development and not "automatically" by the developer.

36-& ± 0OMZ UIF OFDFTTBSZ IFBEFS

In addition, the header file inclusion mechanism can result in multiple inclusions of the same header file, making proofreading of the code difficult at best. Defining a specific symbol for each header file using the preprocessor directive (O/27 and checking that this symbol has not already been defined (OB7M) avoids repeated inclusion of a header file. This is referred to as a ' ™ • ˜ • " • ‰ • ' ‡ • ™ ^ ‰ ‹ ™ … be sure to define a unique symbol for each file. The name of this symbol can be constructed by taking the file name and substituting the "." with a "_".

36-& ± 6TF NVMUJQMF JODMVEF HVB

A guard macro against multiple inclusions of a file should be used to prevent the content of a header file from being included more than once:

```
ff bi `iQ7?2 /2`7BH2
```

```
OB 7 M / 2>71  . 1 _ n >
O/ 2 7 B M>21  . 1 _ n >
f  7 B H 2 Q M i 2 Mfi
O2 M / B 7
ff 2 M /Q 7? 2  / 2 `7 B H 2
```

8BSOJOH

The use of the O T ` ; K QlM+2ve is widespread but is not standardized. This solution is therefore not recognised in this guide, although it is supported by most compilers. Its use can be problematic since this directive is specific to each compiler (especially when managing header files that are duplicated in multiple physical sources or mount points).

Finally, for reasons of readability, the location of header file inclusions must comply with certain specific rules.

36-&

36-& ±)FBEFS ¾MF JODMVTJPOT BSF

All header file inclusions must be grouped at the beginning of the file or just after preprocessor comments or directives, but always before the definition of global variables or functions.

3&$0

3&$0..&/%"5*0/ ± 4ZTUFN IFBEFS ¾MF IFBEFS ¾MF JODMVTJPOT

(00% 13"$5

(00% 13"$5*$& ± 6TF BMQIBCFUJDBM IFBEFS ¾MFT

To avoid any redundancy in system or user header file inclusions, the developer can use the alphabetical order, which offers a deterministic inclusion order and facilitates the code review.

*OGPSNBUJPO

Although these last three rules, recommendations and good practices address a problem of readability and maintainability, and not directly a security problem in the strict sense of the term, these first two aspects remain essential for any type of development.

When the inclusion of a header file is omitted, the compiler may provide a warning about the use of an implicitly declared function.

*OGPSNBUJPO

Implicit function declarations are detected with GCC and CLANG using the option @ q B K T H B + B i @ 7 m M + i B .Q M @ / 2 + H ` i B Q M

#BE FYBNQMF

In the code below, the inclusion of bi`BM;X? serves no purpose in 7BH2X? since the only declaration of bi`BM;X? used by 7BH2X? is the K2K+ function. The inclusion of bi`BM;X? must therefore be moved to 7BH2X? with a multiple include guard in 7BH2X?

```
f  ?2 /2X?  f
OBM+Hmbi`BM;X?=b?QmHQ/MH#v2BM+HmbB2M7BHX2 f

pQB7QUQmBMi3npiHmBMijkHi2M;V?c

f  7BHX2 f
OBM+Hmbi`BM;X?=
OBM+Hmbi/BMiX?=
OBM+Hmbi/HB#X?=
OBM+Hmj?2 /2X?]

O/27BM"i166 1_nG1L 3I

pQB7QUQmBMi3npiHmBMijkHi2M;V?&
  mBMi3#im77(2'i166 1_nG1L
  B7ULIGG54pHV &
    K2K+U#vm77-2`pHKBUWi166 1_nG1HL2M;V?Vc
    XXX
  '
'
```

(PPE FYBNQMF

The example below includes in the header file only the necessary definitions, and an include guard is present. Be careful however, when using the include guard, not to use an already reserved identifier as a name in the macro, which is a classic error when using the include guard.

```
f  ?2 /2X?  f
OB7M/2>71 .1_nf> BM+Hm/m2`/iQ pQB/mHiBTBBM+HmbBBQM
O/27BM>21 .1_n>

pQB7QUQmBMi3npiHmBMijkHi2M;V?c

O2M/Bf7 >1 .1_nf>

f  7BHX2 f
OBM+Hmbi`BM;X?=
OBM+Hmbi/BMiX?=
OBM+Hmbi/HB#X?=
OBM+Hmj?2 /2X?]

O/27BM"i166 1_nG1L 3I

pQB7QUQmBMi3npiHmBMijkHi2M;V?&
  mBMi3#im77(2'i166 1_nG1L

  B7ULIGG54pHV &
    K2K+U#vm77-2`pHKBUWi166 1_nG1HL2M;V?Vc
    XXX
  '
'
```

3FGFSFODFT

[Misra2012] Dir. 4.10. Precautions shall be taken in order to prevent the contents of a header file being included more than once.
[Misra2012] Rule 20.1. O B M + kirectives should only be preceded by preprocessor directives or comments.
[Cert] Rec. PRE06-C Enclose header files in an include guard.

/PO JODMVTJPO PG TPV

The inclusion of a source file in another source file can generate problems with link editing (multiple definitions of global variables or identical functions) or duplication of binary code (in the event that the included elements have been declared with the keyword b i iB).-If a source file requires the use of functions from another source file, a corresponding header file must be declared and included in the source file that needs it. The code must be broken down into independent modules (".c" files).

If the purpose of including one module within another is to take advantage of the compiler's interprocedural optimizations (• ' • • ' •, constant propagation, ‰), it is preferable to rely on Link Time Optimization (LTO), for instance with GCC by using the @ 7 H flag when compiling **and** at link time:
;+ + @Q #BM `v @7HiQ 7B.H2RX+ 7BH2kX+

3 6 - &

3 6 - & ± %P OPU JODMVEF B TPVSDF ¾M

Only the inclusion of header files is authorised in a source file.

#BE FYBNQMF

In the following example, a source file inclusion is performed, which is prohibited.

```
f   7BHX+R f
OB M+Hmb1/BMiX?=

pQB7QUQmBMiRepnl+V &
  XXX
'


f   7BHX+k f
OB M+Hmj/72BHX+R} f   T`Q?B#Bi2/

pQB# `UV &
  7QUQJ :A*no  OVlc1
'
```

(PPE FYBNQMF

The example below correctly breaks down the code into different modules.

```
f   ?2  /2X?  f
OB7M/2>71  .1_n>
O/27BM>21  .1_n>

pQB7QUQmBMiRepnl+Vc

O2M/B7
```

```
f   7 B H 2X+R  f
p Q B 7 QUQm B M i R epn i+V  &
  X X X
'

f   7 B H 2X+k  f
OB M + H m l b2 / B M i X ? =
OB M + H m j /? 2   / 2X?]

O/ 2 7 B MJ 2 : A * n o  G l 1  9 k l

p Q B #  `U V  &
  7 QUQJ  : A * n o  GVlc1
'
```

'PSNBU PG B ¾MF JODM

Different file systems do not behave in the same way: some file systems are case sensitive, and the separator of the constituents of a path may vary.

When an operating system-specific path separator is used, this prevents the portability of the source code. When an OBM+Hlnclude directive includes a path to the header file to be included, the separating character for the path components must be the slash "/", not the backslash "\", to ensure portability of the source. In addition, the character "\", as well as the following characters or sequences of characters: ^;], f and ff located between opening and closing chevrons (l and ⇉) or between double quotes (• À ¾ead to undefined behaviour.

Directory and file name case must also be preserved.

36-&

36-& ± 'JMF QBUIT NVTU CF QPSUBCM

File paths, whether for an OBM+Hlnclude directive or not, must be portable while respecting the case of directory names.

#BE FYBNQMF

In the following example, portability is not assured and leads to undefined behaviour.

```
OB M + H m l b2 v b $ b i  i X ? =
OB M + H m j /2 Q / m H 2anm # n J Q / m$H 2 n/ 2X?]
```

(PPE FYBNQMF

The example below uses a correct format for including header files.

```
OB M + H m l b2 v b f b i  i X ? =
OB M + H m j K2 Q / m H f2bnm # n K Q / m f H 2 n/ 2X?]
```

In addition, certain specific rules must be respected in the header file name to avoid undefined behaviour.

36-&

36-& ± 5IF OBNF PG B IFBEFS ¾MF NVT TFRVFODFT PG DIBSBDUFST

The name of a header file must contain none of the following characters and character sequences: ^;], $ f and f f.

3FGFSFODFT

[Misra2012] Rule 20.2: The ”,” or ”\” characters and the ”/*” and ”//” character sequences shall not occur in a header file name.
[Misra2012] Rule 20.3: The OBM + Hdirective shall be followed by either a I7BH2MorK2=]7BH2Msequence.
[AnsiC99] Sections 6.4.7 and 6.10.2.

$PNNFOU BOE EF¾OJUJ

The compilation directives OB7OB7/O7B7M/O2H2HandO2M/form blocks. These may be long and impossible to display on a single screen. They may also contain nested blocks. It can then be very difficult to determine the interdependencies between the different directives. These directives must therefore be commented on carefully to explain the cases dealt with and the sequence of the different directives.

3&$0

3&$0..&/%"5*0/ ± 1SFQSPDFTTPS CMF

Preprocessor block directives must be commented on in order to clarify the cases being dealt with and, in the case of “intermediate” and “closing” directives, these must also be associated with the corresponding “opening” directive by means of a comment.

For reasons of readability, double negation in preprocessor directive expressions should be avoided, typically by using OB7Mand a “non-mode” (• À%mode defined via the negation of another mode such as L.1").:

(00%
13"$5

(00% 13"$5*$& ± %PVCMF OFHBUJPO CMPDL DPOEJUJPOT TIPVME CF BWPJ

Finally, it is essential that all the directives associated with a preprocessor block (• À%opening”, “intermediate” and “closing” directives) are present in the same file. Furthermore, it should only be possible to evaluate the control conditions used in these directives to 1 or 0.

36-&

36-& ± %F¾OJUJPO PG B QSFQSPDFTT

For a preprocessor block, all associated directives must be found in the same file.

3 & $ 0

3 & $ 0 . . & / % " 5 * 0 / ± 1 S F Q S P D F T T P S E J D P S S F D U M Z G P S N F E

Control expressions must be evaluated only to 0 or 1 and should only use defined identifiers (via O / 2 7 B M 2

B E F Y B N Q M F

The following code should be modified to add comments and not use double negation for the O 2 H a b 2 O 2 M / B 7 directives.

```
f  7 B H 2 X + R  f
O B 7 / 2 7  f  M Q O 2 M / B 7 f
O B M + H m j / + 2 Q X ? ]

f  H Q X ? f

O 2 M / B f 7  O 2 M / B Q 7 7 B H 2 X + R  f

O B 7 M / 2 G 7 P : n >
O / 2 7 B M G 2 P : n >

i v T 2 / 2 2 M m & K
  . 1 " l : 4  y -
  q  _ L
  A L 6 - P
  1 _ _ P _
  6  h  G
'  G Q ; G 2 p 2 H n h

p Q B H Q ; n K b ; G Q ; G 2 p 2 H 2 p 2 H + Q M b m M b B ; M + 2 / ` b G Q ; J 2 b V c ; 2

O B 7 M / 2 L 7 . 1 " l :  f  / Q m # H M 2 2 ; i B Q M
O / 2 7 B M G 2 P : n . 1 " l : U K b ; V  H Q ; n K b ; U . 1 " l : - U K b ; V V
O / 2 7 B M G 2 P : n q  _ L U K b ; V  H Q ; n K b ; U q  _ L - U K b ; V V
O / 2 7 B M G 2 P : n A L 6 P U K b ; V  H Q ; n K b ; U A L 6 P - U K b ; V V
O / 2 7 B M G 2 P : n 1 _ _ P _ U K b ; V  H Q ; n K b ; U 1 _ _ P _ - U K b ; V V
O / 2 7 B M G 2 P : n 6  h  G U K b ; V  H Q ; n K b ; U 6  h  G - U K b ; V V
O 2 H b 2
O / 2 7 B M G 2 P : n . 1 " l : U K b ; V
O / 2 7 B M G 2 P : n q  _ L U K b ; V
O / 2 7 B M G 2 P : n A L 6 P U K b ; V
O / 2 7 B M G 2 P : n 1 _ _ P _ U K b ; V
O / 2 7 B M G 2 P : n 6  h  G U K b ; V
O 2 M / B 7
O 2 M / B 7
```

(P P E F Y B N Q M F

In the following example, the directives are correctly commented on and the associated directives are in the same file.

```
f  7 B H 2 X + R  f
O B 7 / 2 7
O B M + H m j / + 2 Q X ? ]
O 2 M / B 7

f  H Q X ? f
O B 7 M / 2 G 7 P : n >
O / 2 7 B M G 2 P : n >

i v T 2 / 2 2 M m & K
  . 1 " l : 4  y -
  q  _ L
  A L 6 - P
```

```
  1 _ _ P _
  6 h G
' G Q ; G 2 p 2 H n h

p Q B H Q ; n K U G ; Q ; G 2 p 2 H 2 p 2 H + Q M b m M b B ; M + 2 / ` b G Q ; J 2 b V c ; 2

O B 7 / 2 7 L . 1 " l : f / Q m # H M 2 2 ; i B Q M K Q p 2 f
f L Q i / 2 # m K Q / 2 f
O / 2 7 B M G 2 P : n . 1 " l : U K b ; V
O / 2 7 B M G 2 P : n q _ L U K b ; V
O / 2 7 B M G 2 P : n A L 6 P U K b ; V
O / 2 7 B M G 2 P : n 1 _ _ P _ U K b ; V
O / 2 7 B M G 2 P : n 6 h G U K b ; V
O 2 H b 2 O B 7 / 2 7 L . 1 " l : f
f . 2 # m K Q / 2 f
O / 2 7 B M G 2 P : n . 1 " l : U K b ; V H Q ; n K b ; U . 1 " l : - U K b ; V V
O / 2 7 B M G 2 P : n q _ L U K b ; V H Q ; n K b ; U q _ L - U K b ; V V
O / 2 7 B M G 2 P : n A L 6 P U K b ; V H Q ; n K b ; U A L 6 P - U K b ; V V
O / 2 7 B M G 2 P : n 1 _ _ P _ U K b ; V H Q ; n K b ; U 1 _ _ P _ - U K b ; V V
O / 2 7 B M G 2 P : n 6 h G U K b ; V H Q ; n K b ; U 6 h G - U K b ; V V
O 2 M / B f 7 O B 7 / 2 7 L . 1 " l : f
O 2 M / B f 7 O B 7 M / 2 G 7 P : n > f
```

3FGFSFODFT

[Misra2012] Rule 20.8 The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1.
[Misra2012] Rule 20.9 All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #defined before evaluation.
[Misra2012] Rule 20.14 All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related.

6TJOH UIF QSFQSPDFTT

The evaluation order of several # (— ˜ – • ' ‹ • ± or character string conversion operator) or ## (concatenation operator) or the mixture of these two operators is not specified.

36-& ± %P OPU VTF NPSF UIBO POF PG JO UIF TBNF FYQSFTTJPO

It is also important, with these two operators, to have a good understanding of how it works, • À ‰ À the steps resulting from the macro replacement.

36-& ± 6OEFSTUBOE UIF NBDSP SFQM PQFSBUPST BOE

#BE FYBNQMF

```
OB M + H mlb2 / B Q X ? =
X X X
O/ 2 7 B MJ2u S _ A L h U b V  T ` B M i 7 U O b V
O/ 2 7 B Mh2q P  k

B M iK  BUp Q BV/
&
  J u S _ AUhqV cf  T ` B M ijhq P  f
  ` 2 i m ` R c
'
```

(PPE FYBNQMF

```
OB M + H mlb2 / B Q X ? =
X X X
O/ 2 7 B MJ2u S _ A L h k U b V  S _fA L h/UBbiVB Q MBHM / B ` 2 + i BQ2Mt T  M /
                                  ]hq P  f
O/ 2 7 B MS2_ A L h U b V  T ` B M i 7 U O b V
O/ 2 7 B Mh2q P  k

B M iK  BUp Q BV/
&
  J u S _ AUhqV cf  T ` B M ijbk ]  f
  ` 2 i m ` R c
'
```

3FGFSFODFT

[Misra2012] Rule 20.10 The # and ## preprocessor operators should not be used.
[Misra2012] Rule 20.11 A macro parameter immediately following a # operator shall not immediately be followed by a ## operator.
[Misra2012] Rule 20.12 A macro operator used as an operand to the # or ## operators which is itself subject to further macro replacement, shall only be used as an operand to these operators.
[Cert] PRE05-C Understand macro replacement when concatenating tokens or performing stringification.
[AnsiC90] Section 6.8.3.
[AnsiC99] Section 6.10.3.

4QFDJ¾D OBNJOH PG N

It is not always easy to distinguish the use of a preprocessor macro in the source code. The use of some macros may resemble function calls.

Furthermore, when a macro is not named in upper case, there is a risk that the name corresponds

36-& ± .BDSPT NVTU CF TQFDJ¾DBMM

36-&

To easily differentiate macros from functions and not use a reserved name from another C macro, preprocessor macros must be uppercase. In addition, the words making up the name must be separated by the underscore character "_", but without starting them with the underscore character, as this is a convention for reserved names in the C language.

#BE FYBNQMF

In the following example, the macro naming rule is not followed.

```
O/27BM+2i2 ytd1fl HQr2+ b2f
O/27BMn2K vnb[m `2/U V UU fV HQv2+ b2M/bi `ibrBi?n f
```

(PPE FYBNQMF

The following example shows suitable naming of the preprocessor macros.

```
O/27BM*2h1 ytd1l
O/27BMJ2unaZl _1.U V UU V U VV
```

3FGFSFODFT

[Misra2012] Rule 5.4 Macro identifiers shall be distinct.
[Misra2012] Rule 5.5 Identifiers shall be distinct from macro names.
[Misra2012] Rule 21.1: A #define or #undef shall not be used on a reserved identifier or reserved macro name.
[Misra2012] Rule 21.2: A reserved identifier or macro name shall not be declared.
[Misra2012] Rule 20.4: A macro shall not be defined with the same name as a keyword.
[IsoSecu] Using Identifiers that are reserved for the implementation [resident].
[Cert] Rule DCL37-C Do not declare or define a reserved identifier.
[Misra2012] Dir. 4.5 Identifiers in the same name space with overlapping visibility should be typographically unambiguous.
[Cert] Rec. DCL02-C Use visually distinct identifiers.

" NBDSP NVTU OPU FOE X

Macros are used to make it easier to read the code and avoid repeating the same code pattern several times. When expanding a macro, if the macro definition contains a semicolon, this one is also expanded, which can cause a complete and unexpected change in the control flow.

36-& ± %P OPU FOE B NBDSP XJUI B TFN

36-&

The final semicolon should be omitted at the end of the definition of a macro.

#BE FYBNQMF

The macro is not defined by applying the rule and ends with a semicolon.

```
O/27BMa2ZI _1.UMV UMV 4 UMV MQbM2VKB+QHiQiM222M/Q7 K +`Qf
XXX
B7Ut =4 yV
  aZI _1UtVcf +QM/BiBQrMBiPQm#i` +2f
2Hb2
  t 4 @c
XXX
```

on expansion, we have:

```
O/27BMa2ZI _1.UMV UMV 4 UMV UMVc
XXX
B7Ut =4 yV
  t4t tc
cf 2KTivbi i2K2Mf
2Hb2f T `bBM2;``Q`#27Q`i2222Hb2f
  t 4 @c
XXX
```

(PPE FYBNQMF

The macro is corrected:

```
O/27BMa2ZI _1.UMV UMV 4 UMV UMV
f XXXf
B7Ut =4 yV
&
  aZI _1UtVc
'
2Hb2
&
  t 4 @c
'
```

on expansion, we have:

```
 O/27BMa2ZI _1UMV UMV 4 MV UMV
f XXXf
B7Ut =4 yV
&
   t 4 t tc
  '
2Hb2
&
  t 4 @c
'
```

3FGFSFODFT

[Cert] Rec. PRE11-C Do not conclude macro definitions with a semicolon.

(JWF QbS$ Fi B JFOS M OODF GV P C
²GVODUJPO UZQF³ NBDS

'••f%actions have been available since the C99 version of the C language.

*OGPSNBUJPO

As mentioned in section 13.5, •'••fu%%ctions must also be declared as bi iB+

The use of a bi iB+ BMfuBtiv2 to replace these “function type” macros prevents errors in the order of operator evaluation when •'••'•macros, and makes the code easier to read.

*OGPSNBUJPO

It is important to note that unused bi iB+ BMfuBtiv2s may cause warnings to be issued on compilation for some compiler versions. In particular, some versions of the CLANG compiler issue warnings in this case, unlike the GCC compiler. This difference in behaviour between compilers occurs when the @qmMmb2/@otionis iBQM enabled at compile-time, either explicitly or via other options such as @q.When the associated code really cannot be deleted (in a library for example), the developer may have to use compiler extensions to silence these warnings, but these additions must be clearly commented on and justified.

3&$0..&/%"5b0/iB+6TBFMHGBVO2DUJPOT JONVMUJ TUBUFNFOU NBDSPT

In addition to the above recommendations and rules, it is important to add that macros whose replacement defines functions in the code should not be used. The associated risk of error is too great and the readability of the code can suffer from this kind of practice.

36-&±5IF SFQMBDFNFOU PG B EFWFM GVODUJPO

3FGFSFODFT

[Misra2012] Dir. 4.9: A function should be used in preference to a Š™'‡˜•“'Â••where.they– “ are interchangeable.
[Cert] Rec. PRE00-C Prefer inline or static functions to function-like macros.

.VMUJ TUBUFNFOU NBD

The use of macros with multiple statements can lead to unexpected behaviour. Indeed, when defining a macro with several statements, the character “\” must be used to indicate to the preprocessor that a line break must be inserted. This makes the defined macro difficult to read and can also be a source of errors.

The grouping of statements in a /Q &' r?BH2 loop limits the possibilities of unexpected behaviour. A /Q &' r?BH2 loop is always executed exactly once and prevents changing the control flow of the function calling the macro by grouping all its statements in a loop.

36-& ± .BDSPT DPOUBJOJOH NVMU /Q &' r?BH2 MVMUJQMF TUBUFNFOU NBDSPT JO B EP XIJMF MPPQ

#BE FYBNQMF

The macro does not implement the rule.

```
O/27BM>2 G6nalJU -#-+-/V $
  UV 4 U+U Y /UVV fkc $
  U#V 4 U+U @/UVV fk
f  H2 /biQ  /B772`2M2? pBQiQ`i?2QM22[mB`2/Bi?  + HHBM  +QM/BiBQM H
    bi i2K2M+Bi?Qm#i` +2bf

  BTU+ = /V
    > G6naUJ #- +- /Vc
  2Hb2
    f  XXXf
```

Even if the macro were defined between braces:

```
O/27BM>2 G6nalJU -#-+-/V & U V 4 UU+V Y U/VV fkc U#V 4 UU+V @ U/VV f
```

the replacement of the macro in the same conditional is always problematic because of the lack of braces in the conditional and the “ c” following the call of the macro:

```
BTU+=/V
   & UV4U+U Y /UVV fkc#U 4 U+U @/UVV fk 'c
2Hb2
   f  XXXf
```

(PPE FYBNQMF

In the following example the macro is correctly defined using a ^“ Â › Œ • k%pat ruc- ture.

```
O/27BM>2 G6nalJU - #- +- /V $
   /Q &                  $
       UV 4 U+U Y /UVV fkc $
       U#V 4 U+U @/UVV fkc $
   ' r?BHU2y V
```

3FGFSFODFT

[Cert] Rec. PRE10-C Wrap multistatement macros in a do-while loop.

"SHVNFOUT BOE QBSB

During the replacement of a macro by the preprocessor, side effects not expected by the developer may occur if the macro parameters are not protected. Parentheses must systematically be added

around parameters in the definition of a macro.

36-& ± .BOEBUPSZ QBSFOUIFTFT BSP PG B NBDSP

The parameters of a macro must always be enclosed in parentheses when used, in order to preserve the desired order of evaluation of the expressions.

In general, it is best to avoid arguments of a macro resulting in an operation in the broadest sense. Apart from the side effects, even if the operation performed as an argument is constant for a given input, the performance of the code is not optimal.

3&$0..&/%"5*0/ ± "SHVNFOUT PG B NB TIPVME CF BWPJEFE

Moreover, if the operation carried out by the arguments of a macro leads to a side effect in the sense of compilation, this can also lead to unexpected behaviors such as multiple evaluations of the arguments of the macro or even to no evaluation at all.

36-& ± "SHVNFOUT JO B NBDSP NVTU O

Macro arguments with side effects can lead to unwanted multiple evaluations.

Finally, the use of preprocessor directives (O / 2 7 B M 2 7 /.2) in macro arguments leads to undefined behaviour and is therefore to be avoided.

36-& ± %P OPU VTF QSFQSPDFTTPS EJ

#BE FYBNQMF

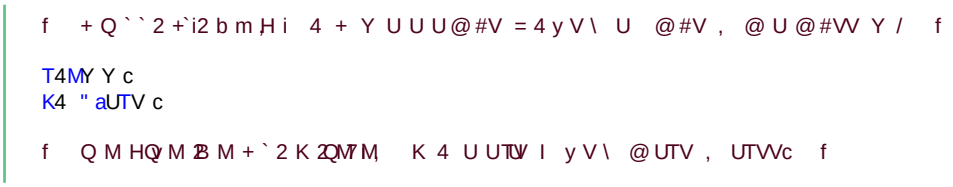
In the following example, the result will not be the one expected upon execution.

```
O/27BM27aUtV Ut =4 y \ t , @tV

  4 + Y  "aU  @#V  Y/c
f  `2bm;Hi  4 + Y U  @# =4y \   @# , @  @#V Y/  f

K4 "aUMY Y Vc
f   //BiBQMBHM+`2K2QM7M,  K 4 UUMY YI yV\ @MY Y, MY Y Vf
```

(PPE FYBNQMF

The following code defines a macro with parentheses correctly placed around its argument.

```
O/27BM27aUtV UUUtV =4 yV \ UtV , @UtVV

  4 + Y  "aU  @#V  Y/c
```



3FGFSFODFT

[Misra2012] Rule 20.7: Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses.
[Cert] Rec. PRE01-C Use parenthesis within macros around parameters names.
[Cert] Rec. PRE02-C Macro replacement lists should be parenthesized.
[Cert] Rule EXP30-C Do not depend on the order of evaluation for side effects.
[Cert] Rule. PRE31-C Avoid side effects in arguments to unsafe macros.
[Cert] Rule. PRE32-C Do not use preprocessor directives in invocations of function-like macros.
[Cert] Rec. PRE12-C Do not define unsafe macros.

6TOOmHMUE2JFS F D U J W F

Use of the Om Mdirective frequently leads to confusion. Inadvertently, its use can lead to partial code deletion if the inclusion of the code is in fact controlled by the symbol whose definition is deleted. It must never be necessary to delete the definition of a preprocessor symbol. If the purpose of deleting the symbol is to limit its scope, it is preferable to check why the scope of the symbol needs to be limited.

The use of Om Mmay result from the risk of a clash in the name chosen for a preprocessor symbol. The symbol name must then be changed to prevent this clash.

36-& ±O5mHMUE2JFS F D U J W F TIPVMEOPUC

36-&

3FGFSFODFT

[Misra2012] Rule 20.5 #undef should not be used.

5SJHSBQI BOE EPVCM

Two successive question marks in C mark the beginning of a sequence associated with a trigraph. For example, the trigraph “??-” represents the character “~”. All trigraphs will be replaced prior to preprocessor directives, regardless of the location of the trigraph. They must not therefore be used.

36-&±%POPUVTFUSJHSBQIT

In addition, to avoid confusion with a trigraph, all comments, strings and other literals should not contain two successive question marks.

3&$0..&/%"5*0/±4VDDFTTJWFRVFTU

This rule applies to all parts of the code, but also to comments.

*OGPSNBUJPO

The @qi`B;` option issues an alert when a trigraph is detected.

*OGPSNBUJPO

By default, trigraphs are disabled in GCC.

3FGFSFODFT

[Misra2012] Rule 4.2 Trigraphs should not be used.
[Cert] Rec. PRE07-C Avoid using repeated question marks.

5
Compilation

Compilation is an important step when developing software since it makes the connection between the code written by the developer and the code that will actually execute on the end user's machine. Mastering it is thus primordial. Moreover, the compiler is a precious ally to detect programming errors or dangerous uses of the language, and it also generally provides hardening features capable of strongly improving the security of the produced software.

8 B S O J O H

In this guide and … Š " –this"chapter, two compilers are frequently used for illustration purposes: GCC [GccRef] and CLANG [ClangRef]. This choice is largely motivated by the popularity of these compilers, which on top of that are open source. It doesn't mean in any way that this guide only recommends using one of these two compilers. Any alternative may be proposed but the developper shall transpose the various options presented in this guide himself.

. B T U F S Z P G U I F D P N Q J

Compilers offer different warning levels meant to inform the developper of the use of risky constructions or the presence of programming errors. The default level is usually rather low and reports few bad practices. It is thus insufficient and must be increased, which requires making used compilation options explicit. Furthermore, for the same version of the C standard, some default behaviors may vary from one compiler to another. Even warnings emitted when compiling are directly tied to the compiler version. It is thus of primary importance to precisely know the compiler in use, its version as well as all enabled options, ideally backed with justifications.

3 6 - &

3 6 - & ± 1 S F D J T F M Z E F ¾ O F D P N Q J M B U

Options used for compiling must be precisely defined for the whole software sources. These options should in particular accurately establish:

- the C standard version in use (for instance C99 or C90);
- the name and version of the compiler in use;
- the warning level (for example @ q 2 t for GCC);
- preprocessor symbols definitions (for instance defining L . 1 " lwhen compiling in release mode).

Moreover, any developer enabling compiler or linker options must be fully aware of the consequences regarding security of the generated executable or library.

3&$0

3&$0..&/%"5*0/±.BTUFS BDUJPOT QF MJOLFS

The developer must know and document all actions stemming from enabled compiler and linker options, including when such options have to do with code optimization.

8BSOJOH

In particular, the use of compilation options such as @7MQ@bi`B+i@Qp2`7HQ @7r`,Tp @7r` Tp@T,QBMi@27MQ@/2H2i2@MmHH@TQrBMi2`@+? @7MQ@bi`B+i@s mBt bTBe; time indicative of a risky usage of the C language.

Using a build automation software like ‘…•,‰…•%‰ ‰—facilitates management of compilation options. The latter may be defined globally and applied to all source files to compile.

(00% 13"$5

(00%13"$5*$&±.BLF VTF PG CVJME B

3FGFSFODFT

[Misra2012] Subsection 4.2. Understanding the compiler.
[Cert] MSC06-C Beware of compiler optimizations.
[Cert] PRE13-C Use the standard predefined macros to test for versions and features.
[Cwe] CWE-14 Compiler Removal of Code to Clear Buffers.

$PNQJMBUJPO XJUIPVU

Making sure that the code compiles without any error nor warning is an excellent way to decrease the risk of programming errors or risky constructions remaining in the code base. Obviously, the idea is not to lower the stringency of compilation options in order to achieve this objective, but to actually fix all issues reported by the compiler. By default, compilation options shall be as strict as possible with the aim of increasing the stringency of the compiler to the fullest.

36-&

36-&±$PNQJMF UIF DPEF XJUIPVU BO TUSJDU DPNQJMBUJPO PQUJPOT

High warning and error levels of the compiler and the linker must be enabled to ensure, as much as possible, the absence of potential issues related to incorrect use of the programming language.
All warnings and all errors reported by the compiler and the linker must be dealt with. It is incidentally very much advised, if using GCC or CLANG, to use the @q2``Q` option in order to turn any warning into a compilation error, hence not running the

risk of ignoring it.

The relevance and accuracy of warnings emitted by the compiler directly depends on the precision of analyzes it conducts, which in turn rely on the various optimizations the compiler is able to perform. Specifying a reasonably high optimization level is thus beneficial.

3 6 - & ± & O B C M F B S F B T P O B C M Z I J H I P

For GCC and CLANG, the optimization level must be at least @ PaRd ideally @ Pk @ P b

8 B S O J O H

It is the responsibility of the developer to make sure that a high optimization level does not suppress defensive code or manually added software countermeasures.

* O G P S N B U J P O

For example, the minimal command line for compiling with GCC or CLANG is: ; + + f + H M ; @ P R @ qq2 H H@ q T 2 / M i @ q 2 ` ` Q ` @ b i / 4 + N H 2 X y + @ Q 7 B H 2 X 2 t 2

3 & $ 0 . . & / % " 5 * 0 / ± 6 T F U I F T U S J D U F T

If a compilation option proves to be too strict for a given development and a choice is made to disable it, a justification shall be provided to explain it.

* O G P S N B U J P O

Appendix B.2 draws up a non-exhaustive list of extra warnings for GCC and CLANG, which can serve as a starting point to the developer.

In order to suppress errors and warnings, the first thing to do is of course to fix the source code, while imperatively commenting on any resulting code edit. However, if it appears to be a false positive, several methods exist. Firstly, the complexity of a code snippet may occasionally suffice to mislead the compiler analysis, and it is then beneficial in general to simply rewrite this hunk in a more intelligible form, while obviously making sure it is semantically equivalent. Then, if it turns out that a warning cannot be eliminated by fixing the source code, and if the compiler allows for it (via a O T ` ; directive for example), this warning may be disabled locally.

…Enables all the warnings about risky language constructions that are easy to avoid. See the GCC and CLANG compilers manuals for a complete listing.

† Enables some extra warning flags that are not enabled by @ q . See the GCC and CLANG compilers manuals for a complete listing.

‡ Enables all the warnings demanded by the C standard; disables all compiler extensions, including the ones that do not conflict with the standard.

ˆ. Specifies the C standard used by the compiler. See appendix B.1.

WARNING

Using `OT` ;K to suppress warnings can be quite dangerous and must therefore be perfectly understood in order not to disable one or more warnings in the whole code base by mistake. Moreover, as the use of `OT` ;K not standard, the developer must keep in mind that this is specific to each compiler (implementation-defined) and thus risky.

In case the developer opts for warning suppression, a clear justification needs to be provided with a comment:

```
OT` ;K :** /B ;MQbiB+ Tmb?
OT` ;K :** /B ;MQbiB+ B;MQ`2M mQ2/B #H2
f +Q/2f
f bQK2 `B #H2b2 MQimb2/BMi?2M2rH;Q`Bi?KvKQ#2mi`2F2Ti7Q` SA+QKT iB#BHfBiv
f r`MBM;HQmbim+?MMmbp2/`B #H2b2i?mb/Bb #H2B/Mi?2+Q/27QHHQrB?M2;T` ;K f
OT` ;K :** /B ;MQbiB+ TQT
```

BIBLIOGRAPHY

[Misra2012] Dir. 2.1: All sources files shall compile without any compilation errors.
[Misra2012] Dir. 4.1: Run-time failures shall be minimized.
[Cert] MSC00-C: Compile cleanly at high warning levels.
[Cwe] CWE-563 Unused variable.
[Cwe] CWE-570 Expression is always false.
[Cwe] CWE-571 Expression is always true.

TEMPORARY FIX

Modern compilers offer various options that make it possible to improve the robustness and defensiveness of the final executable. It can come to prevent a vulnerability from surfacing or to reduce its security impact, but also to harden the program against vulnerability exploitation attempts.

36-& ± .BLF VTF PG TFDVSJUZ GFBUVS

Developers must, as much as possible, take advantage of compilation options that allow for improving the security of the final software product.

WARNING

Throughout this section, when GCC or CLANG options are given as examples, it is necessary to keep in mind that:

- these options apply to GCC 11 and CLANG 13 respectively;
- some of them are already enabled by default, sometimes partially, depending on the compiler and its version, but it is preferable to specify them anyway;
- the accuracy of warnings emitted by the compiler may depend upon the selected optimization level;

- the impact on performances, if mentioned, is given for information purposes only and may vary a lot according to use cases; it is therefore the developer's job to ensure compliance with his needs.

8BSOJOHT PSJFOUFE UPXBSET

A number of warnings offered by compilers focus very especially on detecting potential security issues. Enabling these warnings and dealing with any issue they report are thus a first step in reducing the risk of software vulnerabilities being introduced in the code base.

36-&

36-& ± &OBCMF XBSOJOHT UIBU GPD XJUI BOZ SFQPSUFE JTTVF

For instance, the @q7Q`K ic4k and CLANG compilation option must be enabled and any reported issue needs to be methodically dealt with.
GCC options @q7Q`K i@Qp2`7@Q74Qk`K i@i`m Mmayiabs Qbd 4sk.

*OGPSNBUJPO

The GCC and CLANG compilers nowadays embed static source code analyzers capable of running deeper and more precise — but more expensive — analyses on programs in order to detect more programming errors, especially potential security vulnerabilities.
Even though these embedded analyzers are still rather basic and in the end yet quite experimental compared to standalone tools that are dedicated to static source code analysis, it may be convenient and interesting to use them. The interested developer may refer to the documentation [GccRef] of GCC option @7 M Has well as the section from the CLANG documentation [ClangRef].

*OTUSVNFOUBUJPO PG TPNF QB

The use of functions that handle memory or character strings is a large source of programming errors that frequently lead to vulnerabilities such as buffer overflows. In addition to choosing less dangerous versions of such functions when available (‡ Šuc…s from section 17.4), some compilers are capable of automatically enhancing them with simple checks, performed at compile or execution time, to detect potential buffer overflows.

36-&

36-& ± &OBCMF UIF VTF PG IBSEFOFE

For instance, when using GCC to compile a program intended for a GNU/Linux system running the glibc, the n6P_hA6unaPro_m1 must be defined. The optimization level must be greater or equal to @ for the added checks to be effective.
Run-time checks terminate the program as soon as an overflow is detected.

…This option adds additional compile-time checks on format strings, as well as on function calls that take them as arguments.

*OGPSNBUJPO

Option @qbi`BM;QT@Qpf2GC7HaQrs M for performing similar checks at compile-time only. Their stringency depends on the value of M

*OJUJBMJ[BUJPO PG BVUPNBUJD

The use of uninitialized automatic variables[3] is another common error and source of vulnerabilities (see section 6.10 as well). A number of compilers however are capable of detecting some of these uses, as long as the appropriate warnings are enabled.

36-&

36-& ± &OBCMF DPNQJMFS XBSOJOHT BCMFT

In particular, options @qmMBMBiB, HB@2q/BMBi@b2nH7 @qKv#2@mMBMmBusBbeHBab2d when GCC is used.
As for CLANG, options @qmMBMBiB2nHB@xq2+/QM/BiBQM H@mmBMBiBHBx be enabled.

8BSOJOH

It should be noted that these warnings do not detect all instances of uninitialized automatic variable uses, especially when such variables are passed to other functions by reference.

Furthermore, some compilers support automatic initialization of such variables. In practice, this forced initialization can use either the value zero or another particular value, called " … ˜˜.%Au-' tomatic initialization to zero should be chosen for –‰•%builds (‰Šnext section 5.4) because it usually limits the exploitability of this type of bug. In contrast, during development, testing and debugging phases, pattern initialization is preferable since it is more likely to uncover certain bugs. Selecting the right pattern is then essential: for example, for pointer-type variables, it may be a non-canonical address so that any memory access through an uninitialized pointer systematically faults.

36-&

36-& ± &OBCMF GPSDFE JOJUJBMJ[BU

CLANG supports automatic initialization with the two aforementioned approaches:

- @7i`BpB H@ miQ@p `@BMBi4T2èM and debugging;
- @7i`BpB H@ miQ@p `@BMBi4x2%Q•‰…–%ds, an option that currently necessitates appending option @2M #H2@i`BpB H@ miQ@p `@BMBi@x2`Q@FMQrBM;@Bi@

3. An automatic variable is a variable defined within a function, without the bii Bstorage class specifier. Its storage is allocated and deallocated automatically on the call stack.

…Automatically enabled by @q HH
† Automatically enabled by @q HH
‡ Automatically enabled by @q HH

8BSOJOH

Even when all automatic variables are forcibly initialized by the compiler, their use without prior explicit initialization by the developer is still a programming error that absolutely has to be fixed.
Automatic initialization by the compiler is thus a hardening only intended to limit the security impact of this type of bugs, and as a consequence the developer should never rely on this behavior.

*OUFHFS PWFS¿PXT

Signed integer overflows are not defined by the C standard and are thus particularly dangerous. For instance, depending on hardware architectures and compilers, a variable of type B Mreaching the value A L h n Jars › – ..after another increment, that is become A L h n ,Which may prove to be quite problematic especially in the case of a variable that represents a reference counter for a memory allocation. The compiler may be able to detect certain kinds of signed integer overflows.

3 & $ 0

3&$0..&/%"5*0/ ± &OBCMF DPNQJMF TJHOFE JOUFHFS PWFS¿PXT

In particular, GCC and CLANG both support option @ 7 i ` , Which makes the compiler instrument the source code in such a way as to emit a trap during program execution for any signed integer overflow on addition, subtraction or multiplication operations.

8BSOJOH

Even though **unsigned** integer overflows are actually a well-defined behavior of the C language, they are not any less dangerous and may just as well lead to the introduction of bugs and vulnerabilities in a piece of software. The developer should thus remain particularly careful when performing operations prone to overflows, even with unsigned operands.

*OGPSNBUJPO

GCC and CLANG support many other options that are useful for detecting integer overflows, but they are part of the UBSan ··— … ' •, the ž %e- of which as well as of other sanitizers is not addressed by this guide.

$BMM TUBDL IBSEFOJOHT

In order to make it harder to exploit certain vulnerabilities, the memory area corresponding to the program stack should not be executable. Modern toolchains generally endeavor to enforce this rule by default.

..Undefined Behavior Sanitizer

Nonetheless, the use of nested functions, as supported by GCC, imposes an executable stack[4] and is thus prohibited[5]. Moreover, supporting this feature has complicated the way GNU linkers — namely BFD and gold — choose to mark an executable or a library as requiring executable stack; it is thus necessary to be very cautious when using them.

36-&

36-& ± %P OPU VTF FYFD VUBCMF TUBD

In particular, GCC nested functions should not be used. Besides, for software intended for GNU/Linux or FreeBSD environments:

- option @x 2t2+bdf BFD, gold and lld linkers should **not** be used;
- option @x MQ;M mfld should **not** be used;
- option @x MQ2t2 ofBFD and gold **must** be used.

Stack buffer overflows certainly rank among the oldest and most common memory corruptions. Positioning guard variables with a random value, usually called ‡ … ' …[6], makes it possible for example to detect some linear overflow attempts that try to overwrite the value of a return address saved on the stack. If need be, program execution is automatically terminated.

36-&

36-& ± &OBCMF TUBDL DBOBSJFT

With GCC and CLANG, compilation option @7bi +F@T`Qi2+iQ must be enabled.

Section 19.2 also tackles manual implementation of a canary mechanism.

Depending on hardware architectures, platforms and compilers, it is also possible to use a guard variable whose value is different for each thread within the same process. This allows for reducing the severity of a potential leak of the canary value.

3&$0

3&$0../%"5*0/ ± 6TF QFS UISFBE DB

In particular, for x86 architectures, GCC and CLANG support option @Kbi +F@T`Qi2+iQ, relying on a glibc Œ – ‰ … ˆ “ ‡ … • ˜ “ – … ‹ ‰

%ZOBNJD MPBEJOH

Randomization is a probabilistic security defense tactics aiming to make software vulnerability exploitation less reliable. In particular, common toolchains are able to produce executables that can be loaded at a random memory address, in order to make the most of address space layout randomization (ASLR) as implemented by the operating system.

4. More precisely, it is the use of ‡ • “ —, implemented by GCC by means of such nested functions and of trampolines located on the stack, that is problematic.
5. It is a language extension anyway, offered by a compiler, yet, as a reminder, only C code compliant with the C90 or C99 standards is allowed by the present guide.
6. Alternatively known as ‡ “ “ • • ‰ —

36-& ± 1SPEVDF QPTJUJPO JOEFQFO

36-&

With GCC and CLANG, compilation option @ 7 SmAust be enabled, as well as option @ TBf 2BFD, gold and lld linkers.

To allow for relocation of shared libraries and executables, the dynamic loader must be able to modify some of their sections. If the corresponding memory mappings consequently remain writable during program execution, they may prove useful to an attacker trying to exploit a software vulnerability. However, it is possible at link-time to mark said sections so that the dynamic loader makes the underlying memory mappings read-only as soon as possible. This is referred to as ` 2 H ` Q or " … – ˜`•2.H•`mQode.

36-& ±`02THFNQP E F P G M J O L F S T

36-&

For instance, with BFD and gold linkers, option @ x ` 2 H`mùsQ be used. lld enforces ` 2 H `mQode by default thus no extra option is required.

Nevertheless, since function symbols resolution usually happens in the course of program execution (• … ž • † •),ˆa• number of sections within shared libraries or executables remain writable regardless of ` 2 H `mQode. Is is then possible to force the dynamic loader to resolve all symbols when the program is started [7] so that it can go on to make the underlying memory mappings read-only. This is referred to as Š ™ 2•H `oQ" A L . n LmPode.

3&$0..&/%"5*0/ ± %P OPU VTF MB[Z C

3&$0

For instance, option @ x MoQBFD, gold and lld causes these linkers to mark generated executables and libraries so as to tell the dynamic loader it needs to resolve all symbols at program startup.

*OGPSNBUJPO

When ` 2 H `mQode is used and • … ž • † • ïs disabled, some linkers reorder sections inside generated binaries in order to prevent overflows of data located in one section from overwriting the contents of sensitive sections.
This is notably the case with BFD, gold and lld.

3FQSPEVDJCMF CVJMET

Among other things, reproducible builds allow users of a particular software to verify independently that a binary they are provided with is indeed the unaltered product of a precise state of its sources, by rebuilding it and comparing the result. This requires a fully deterministic build process, which is nontrivial and goes beyond the scope of this guide, but taking this problematic into consideration as soon as possible in a project can greatly facilitate the implementation of such a feature the day that it becomes an objective.

7. Disabling • … ž • † • 'mi›t thus slow down a large software's **startup**. It should also be noted that this becomes of little importance in practice for a ˆ … ‰ ‹ " '

(00%
13"$5

(00%13"$5*$&±&OTVSF SFQSPEVDJ

For example, compilation options such as @q/ i2@fBK2cc and CLANG or @7` M/QK@om2dc4 may prove useful to limit the introduction of nondeterminism at compile-time.

%FCVH BOE SFMFBTF NP

Build modes ˆ‰†TM and –‰•‰ are usually both available with any compiler and are very useful for software development because of the significant changes they can induce on build products.

%FCVH BOE SFMFBTF NPEFTT

In ˆ‰†TMode, which is mainly designed for debugging while developing, most optimizations are disabled and debugging information for all symbols is preserved, making it easier in particular to set breakpoints. Compilation is faster and uses less memory, but the generated code will generally be larger and slower to execute. In –‰•‰…—‰ mode, which corresponds to the final mode suitable for client delivery or deploying to production, optimizations are enabled and information that does not add value to program execution, such as symbols, is removed. Compilation takes more time then and consumes more memory, but allows for generating more complex machine code that will thereby be faster and more compact.

‰†TMode enables the developer to better understand how a program works and fix reported errors whereas –‰•‰mode is required for delivery for performance or program size reasons. For instance, in ˆ‰†TMode, some compilers make sure that all variables are automatically initialized to y, while in –‰•‰mode this only applies to global variables, as per the standard.

*OGPSNBUJPO

If L.1"lis defined as a macro name at the point in the source file where the standard library header bb2`iX2included, then the bb2`macro is redefined to be disabled.

36-&

36-&±"MM QSPEVDUJPO SFBEZ DPEF

Compiling in –‰•‰mode is mandatory when putting software into production.

This can seem redundant with rules and recommandations from sections 5.1 and 5.2 but it is definitely a rather common mistake in software engineering. In addition to different behaviors regarding memory management and code optimizations, ˆ‰†TMode may even sometimes increase software attack surface. It is thus very important that the developer make use of these modes in full knowledge of the cause.

3&$0..&/%"5*0/±1BZ TQFDJBM BUUI XIFO CVJMEJOH B QSPKFDU

The use of ˆ ‰ † and – ‰ • ‰ modes at compile-time must be done while making sure that all induced changes regarding memory management and optimization are well known. Every difference between these two modes must be documented exhaustively.

6
Declaration, definition and initialisation

%F¾OJUJPO WFSTVT VTF PG WBSJBC

Defining a variable means assigning a value to it (• Àwriting to the variable's memory address), while using a variable means reading the value of the variable (• Àreading the value stored at the associated memory address).

.VMUJQMF WBSJBCMF E

The C language allows the sumultaneous declaration of several variables of the same type. In multiple variable declarations, each variable is separated with a comma. These multiple declarations are used to associate a given type with a group of variables or to group together related variables. However, this type of multiple declaration should only be used on simple variables (no pointers or structured variables) of the same type.

3&$0

3&$0..&/%"5*0/±0OMZ NVMUJQMF EF TBNF UZQF BSF BVUIPSJTFE

In order to also avoid errors in the initialisation of variables, initialisations coupled with a multiple declaration are to be prohibited. Indeed, in the case of a single initialisation at the end of the multiple declaration, only the last variable declared is actually initialised.

36-&

36-&±%P OPU NBLF NVMUJQMF WBSJ JOJUJBMJTBUJPO

Initialisations associated (• Àconsecutive and in the same statement) with a multiple declaration are prohibited.

#BE FYBNQMF

In the code below, several variables are declared in the same statement, but only the last variable is initialised.

```
mBMijkn#b Q`/4 ycf + miBQM?2p `B #H2#bBbMQib2iiQx2`Q?2`25 f
mBMijkni #cf iQ#2T`Q?B#BiK/BtQ7bBKTHp2`B #H2M/TQBMi2`
    /2+H ` iBQfM
bi`m+#HQ#ni ?(j8)cf iQ#2T`Q?B#BiK/BtQ7bBKTHp2`B #H2TQBMi2M/
    `` v/2+H ` iBQfM
```

(PPE FYBNQMF

```
mBMijkni-
mBMijkn#cf  b2T ` iBQM7i?2bBKTHp2`B #H2M/i?2TQBMi2f`
bi`m+#HQ#nic
bi`m+#HQ#n?(j8)cf  b #Qp2Q`b2T ` iBQM7i?2 `` v M/bBKTHp2`B #H2
    f
mBMijkni#b- Q`cf  DQBM2+H ` iBQM7irQ7mM+iBQM@2HHvi2/p `B #H2fb
 #b4 ycf  bbB;MK2QM7i?2irQp `B #H2fb
Q`/4 yc
```

3FGFSFODFT

[Cert] Rec. DCL04-C Do not declare more than one variable per declaration.

'SFF EFDMBSBUJPO PG

Since C99, variables can be declared anywhere in the code. This feature seems practical, but its abuse can make reading the code significantly more complex and may lead to possible redefinition of variables.

3&$0

3&$0..&/%"5*0/±(SPVQWBSJBCMF E CMPDL JO XIJDI UIFZ BSF VTFE

For reasons of readability and to avoid redefinition, variable declarations have to be grouped at the beginning of the file, of the function or of the block statement according to their scope.

*OGPSNBUJPO

This recommendation is not strictly related to security but has an impact on the readability, portability and/or maintainability of the code, and concerns all types of development.
The @q/2+H ` iBQM@ 7i2`@cbandI2KA2M compiler option can help with enforcing this recommendation.

A very common practice for loop counters is to declare them directly in the associated loop. This “on the fly” declaration is accepted, but a special care must be taken to ensure that the associated variable does not mask one of the other variables used in the body of the loop.

#BE FYBNQMF

In the following code, the variables are declared “on the fly” and not in a grouped and structured manner. This type of practice makes it more complex to identify all the variables declared, thus increasing the risk of variable shadowing.

```
OBM+Hmb2/BMiX?=
mBMi3nHQ#npc f  ;HQ# pH `B #H2
mBMi37iQMUp-QBV/
&
```

```
    mBMi3pi`Rc f   HQ+ pH `B  #H2
    B7U;HQ#np=`4yV
    &
        f   XXXf
    '
    2Hb2
        p `R4;HQ#npc `
    mBMi3pi`kc f   Qi?2`HQ+ pH `B  #H/22+H  `2BMi?2KB//HQ27  #HQ+ff
    f   XXXf
'
mBMi3nHQ#npc`fk  Qi?2`;HQ# pH `B  #H/22+H  `2#/2ir22MQ7mM+iBQfMb
pQBK BUpQBV/
&
    mBMi3ni4 7QMU+Vc
    f   XXXf
'
```

(PPEFYBNQMF

The variables are declared in a grouped and structured way at the beginning of the blocks, which makes them easier to read.

```
OBM+Hmb2/BMiX?=

mBMi3nHQ#npc f  ;HQ# pH `B  #H/22+H  `2i/Q;2i?2`f
mBMi3nHQ#npc`k

mBMi3i7QMU+QBV/
&
    mBMi3pi`Rc f   HQ+ pH `B  #H22+H  `2i/Q;2i?2`i i?2#2;BMMBQMi;?2
        7mM+iBQM
    mBMi3pi`kc
    B7U;HQ#np=`4 yV
    &
        f   XXXf
    '
    2Hb2
    &
        p `R4 ;HQ#npc `
    '
    f   XXXf
'
pQBK BUpQBV/
&
    mBMi3ni4 7QMU+Vc
    f   XXXf
'
```

%FDMBSBUJPO PG DPOT

The direct use of numerical values (or characters and constant or ••˜%) makes the source code difficult to maintain. If a value is changed, one has to remember to change all the statements in which the value is used.

36-&

36-&±%POPUVTF IBSE DPEFE WBMV

The values used in the code must be declared as constants.

The constant declaration rule is also to be applied for all types of values appearing several times in the source code. Therefore, if a character or string is repeated several times, it must also be defined using a global +QMvariable or, failing that, using a preprocessor macro.

Centralising the declaration of constants ensures that the change in their value is applied across the entire implementation.

(00% 13"$5

(00%13"$5*$&±$FOUSBMJTF UIF EF PG UIF ¾MF

To make it easier to read, the constants are declared together at the beginning of the file.

To identify these constants, several rules must be respected.

36-&

36-&±%FDMBSF DPOTUBOUT JO VQQ

Constants that do not require type checking are declared with the keyword O/27BM2

36-&

36-&±$PO TUBOUT UIBU EP OPU SFR O/27BQWS2F Q SP DF TT JOH EJSFDUJWF

36-&

36-&±$PO TUBOUT SFRVJSJOH FYQM UIF LF+ZQXMPlSiE

36-&

36-&±$PO TUBOU WBMVFT NVTU CF UIF UZQF

To avoid misinterpretation, constant values must use a suffix based on their type:

- the suffix l must be used for all unsigned integer type constants;
- to indicate a HQ Kor HQ M; HoQ M99) type constant, the suffix G(or G Ge-spectively) must be used instead of H(or H Hespectively) in order to avoid any ambiguity with the number R;
- floating values are by default considered as /Q m #use2the suffix 7for the float type (or /for the double type respectively).

8BSOJOH

By default, integer values are considered B Mtype and floating values are considered /Q m #type2.

36-& ± 5IF TJ[F PG UIF UZQF BTTPDJBU CF TVG¾DJFOU UP DPOUBJO JU

It must be ensured that the constant values or expressions used do not exceed the type associated with them.

To avoid any confusion, octal constants are to be prohibited. Some cases can be tolerated such as UNIX file modes, but they will be systematically identified and commented on.

3&$0..&/%"5*0/ ± 1SPIJCJU PDUBM DP

Do not use octal constants or escape sequences.

#BE FYBNQMF

The following example does not centralise the definition of the constants. Certain constants have not been declared. There is also an absence of specific naming for the constants.

```
O/27BMQ2+iHn+QMbify QO8+iH# b2MmK2`B+QHMbi MM/M K2BMHQr2+ b2f

+QMbBiMie9m#i4 yHc f  H M/MQiG- M/+QMbi MiKBMT;`Q#H2fK
mBMi3#im77(2yt3kc f  +QMbi MiQi/2+H `2/M/ivT2+?2+M22/27Q`i?Bb
    +QMbi Mfi
BMiRebc

7QUB 4 ycBI yt3kc BYYV &  ? `/@+Q/2p Hm2f
  XXX
'

T`BMU7J2bb $2yR]kVcf  Q+iH# b22b+ T22[m2M$+y2Rk4 $M f
```

(PPE FYBNQMF

The following code applies the different rules and recommendations for the declaration of constants.

```
+QMbmiBMijkAiLAhno Q4lytRkN0lf /2+H `2+QMbi MM/rBi?i?2M2+2bb `v
    ivT2+?2+Ff

O/27BM"1661_naAw1 yft3k12+H `2+QMbi MBi?ivT2+?2+MQiM2+2bb `fv

+QMbBiMie9m#i4 yGc f  +Q``2+iBQ7M?2bm77BM/bT2+B7MB+KBMQ;7i?2
    +QMbi Mfi

mBMi3#im77(2'1661_naAw1
mBMiRebc

7QUB 4 ycBI "1661_naAwBY1YV &
  XXX
'
```

3FGFSFODFT

[Misra2012] Rule 11.8 A cast shall not remove a const or volatile qualification from the type pointed to by a pointer.

[Misra2012] Rule 7.1. Octal constants shall not be used.
[Misra2012] Rule 7.2. A mor l suffix shall be applied to all integer constants that are represented in an unsigned type.
[Misra2012] Rule 7.3. The lowercase character Hshall not be used as a literal suffix.
[Misra2012] Rule 7.4. A string literal shall not be assigned to an object unless the object's type is "pointer to a const-qualifed char".
[Misra2012] Rule 12.4 Evaluation of constant expressions should not lead to unsigned integer wrap-around.
[Misra2012] Dir. 4.5 Identifiers in the same name space with overlapping visibility should be typographically unambiguous.
[Cert] Rec. DCL16-C Use Gnot H,to indicate a long value.
[Cert] Rec. DECL00-C Const-qualify immutable objects.
[Cert] Rec. STR05-C Use pointers to const when referring to string literals.
[Cert] Rec. DECL18-C Do not begin integer constants with 0 when specifying a decimal value.
[Cert] Rule EXP40-C Do not modify constants objects.
[Cert] Rule STR30-C Do not attempt to modify string literals.
[Cert] Rec. EXP05-C Do not cast away a const qualification.
[Cert] Rec. DCL02-C Use visually distinct identifiers.
[Cert] Rec. DCL06-C Use meaningful symbolic constants to represent literal values.
[Cwe] CWE-547 Use of Hard-coded, security relevant constants.
[Cwe] CWE-704 Incorrect type conversion or cast.
[IsoSecu] Modifying string literals [strmod].

- J N J U F E V T F P G H M P C E

When global variables are used, it is difficult to identify every function that modifies these variables. Furthermore, if a global variable is not named according to clear naming conventions, reading the code of a function using this variable does not immediately identify the side effect of the function on this global variable. This specific naming scheme must be clear and remains the choice of the developer or of the developing team (use of upper case, prefix ; ŋ ‰ ƫ. ‡ À

In addition, the use of global variables can quickly lead to problems of concurrency in the case of a multi-tasking application. For each of the global variables, the developer must study the possibility of limiting the scope of the variable systematically.

3 6 - &

3 6 - & ± - J N J U H M P C B M W B S J B C M F T U

Limit the use of global variables and give preference to function parameters in order to propagate a data structure through an application.

B E F Y B N Q M F

The following code uses a global variable. However, its use could easily be avoided.

```
bi iB +m B Mijk ŋ mbi i2

pQB 7 QUp Q B/ &
   XXX
```

```
  ;nbi i24 Rc
'

pQB# `UpQBV// &
  XXX
  ;nbi i24 kc
'

BMiK BUBMi`;+- +? ` `;p()V &
  7QUQVc
  # `UVc
  XXX
'
```

(PPE FYBNQMF

The following example does not use a global variable. The variable bi i2is propagated from function to function by passing it as a parameter:

```
pQB7QUQmBMijknbi i2V &
  XXX
  Ubi i2V 4 Rc
'

pQB# `UmBMijknbi i2V &
  XXX
  Ubi i2V 4 kc
'
BMiK BUBMi`;+- +? ` `;p()V &
  mBMijknbi i24 yc
  7QUQbi i2Vc
  # `Ubi i2Vc
  XXX
'
```

3FGFSFODFT

[Cert] Rec. DCL02-C Use visually distinct identifiers.
[Cert] Rule DCL30-C Declare objects with appropriate storage durations.
[Cert] Rec. DCL19-C Minimize the scope of variables and functions.
[Misra2012] Dir. 4.5 Identifiers in the same name space with overlapping visibility should be typographically unambiguous.
[Misra2012] Rule 8.9 An object should be defined at block scope if its identifier only appears in a single function.

6TFbPGBULIFFZXPSE

When a function is declared, defined and used only within a single source file, the bi iBstorage-class specifier is often forgotten. Conflicts may then arise at link-time. In addition, the absence of the bi iBspecifier makes code review more difficult because it does not allow to quickly notice that a function is “private/local”. The bi iBkeyword tells the compiler that the variable/function is indeed a global variable/function but that its visibility must be limited to the source file in which it is declared.

The same applies for variables global to a file that are not used outside that file. Global variables of this type should be systematically declared as b i i B. This limits the scope of these variables only to the other functions defined in the same file, and therefore limits the exposure of said variables. These global functions and variables should not be declared in a header file.

3 6 - &

3 6 - & ± 4 Z T U F N b B U i B E F Q M D J Z ¾ V F T S F G U P S E F

The b i i B storage-class specifier must be used for all global functions and variables that are not used outside the source file in which they are defined.

3 F G F S F O D F T

[Cert] Rec. DCL15-C Declare file-scope objects or functions that do not need external linkage as static.
[Cert] Rule MSC40-C Do not violate constraints.
[Misra2012] Rule 8.7 Functions and objects should not be defined with external linkage if they are referenced in only one translation unit.
[Misra2012] Rule 8.8 The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage.

6 T F p Q G U i B E F 2 X P S E

The p Q H i B keyword must be used to qualify either a variable corresponding to a hardware area that represents an input/output port in memory, or a variable read or written to by an asynchronous interrupt function. Accesses to such variables should indeed be protected from compiler optimizations.

3 6 - &

3 6 - & ± 0 O M Z W B S J B C M F T U I B U D B O C T I P V M E Q p Q E F i D B M 2 S F E

Only variables associated with input/output ports or asynchronous interrupt functions should be declared as p Q H i B to prevent optimisation or reorganisation on compilation.

Moreover, to avoid undefined behaviour, only a pointer that is itself qualified as p Q H i B can access a p Q H i B variable.

3 6 - &

3 6 - & ± p Q H M Z B R V 2 B M J ¾ F E Q p Q U i B S T 2 W B S J B C M F T

3 F G F S F O D F T

[Cert] Rec. DCL17-C Beware of miscompiled volatile-qualified variables.

[Cert] Rec. DCL22-C Use volatile for data that cannot be cached.
[Cert] Rule EXP32-C Do not access a volatile object through a nonvolatile reference.
[Misra2012] Rule 2.2 There shall be no dead code.
[Misra2012] Rule 11.8 A cast shall not remove any const or volatile qualification from the type pointed to by a pointer.
[Cwe] CWE-704 Incorrect type conversion or cast.
[Cwe] CWE-561 Dead code.

*NQMJDJU UZQF EFDMB

C90 allows the implicit declaration of variables in terms of omission of the type in certain circumstances, such as for the parameters of a function, elements of a structure or the declaration of a i v T 2 /.2 7

*OGPSNBUJPO

In practice, compilers issue a warning (@ q B K T H B) but compile by assume that the type is B M i

36-&

36-& ±/P UZQF PNJTTJPO JT BDDFQUF

All variables used must have been explicitly declared before use.

Furthermore, the [8] function declaration syntax, such as for example:

```
B M i7 QU-TV
   B M i c
   + ?  `  Tc
&  X X X
'
```

is also prohibited. Firstly, this type of declaration is obsolete and, secondly, it reduces the readability of the code and therefore, potentially, the checks made at compiler level.

#BE FYBNQMF

The following code (C90) contains several implicit type declarations.

```
X X X
+ Q M b i* P L a 4  9 k d    T ` Q ? B # B i 2 v T 2 Q 7 + Q M b i  M Q i 2 t T H B + B i 2 v T 2 Q 7 B M 2 U /
     B K T H B + B M V  f
m M b B ; M 2 c /f   T ` Q ? B # B i 2 v T 2 Q 7 2  M Q i 2 t T H B + B i 2 v T 2 Q 7 B M 2 U B K T H B + B M V b B ; M 2 M V
     f
b B ; M 2 7 c  f   T ` Q ? B # B i 2 v T 2 Q 7 + Q M b i  M Q i 2 t T H B + B i 2 v T 2 Q 7 B M 2 /
U B K T H B + B i ; M 2 B M V  f
X X X
B M i7 QU+ ?  `  -  + Q M b # V  f   T ` Q ? B # B i 2 v T 2 Q 7 #  M Q i 2 t T H B + B i 2 v T 2 Q 7 B M 2 U B K T H B + B i
     B M V  f
&
X X X
'
```

8. Kernighan & Ritchie's C syntax before ANSI standards

```
# `U+ ? `+- + Q M bBi M i/ V f T ` Q ? B # B, i 2v T 2Q 7 ` 2 i m ` MQ 7i ? 2 7 m M + i BMQQMi
2 t T H B + B i/2 7 B M 2UB K T H B +BBMV f
&
X X
'
```

(PPE FYBNQMF

All types are now explicit.

```
X X X
+ Q M bBi M i * P L a 4 9 k c
m M b B ; MB2Mi2c
b B ; M 2B Mi7c
X X X
B M i7 QUQm M b B ; M+2?/ ` - + Q M bBi M i#V
&
X X X
'
B M i# `Um M b B ; M+2?/ `+- + Q M bBi M i/ V
&
X X X
'
```

3FGFSFODFT

[Cert] Rule DCL31-C Declare identifier before using them.
[Misra2012] Rule 8.1 Types shall be explicitly specified.

$PNQPVOE MJUFSBMT

“ ‘ ” “ ™ ’ ^ • • ˜ %were introduced by C99 and allow the creation of unnamed objects from a list of initialisation values. This construction is often used with particular structures that have been passed as function parameters. The lifetime of a ‡ “ ‘ ” “ ™ ’ ^ • • ˜ %is either static or automatic depending on whether it is declared at file or block statement level.

Attempting to access the associated object outside of its scope will result in undefined behaviour. It is therefore essential to fully understand the scope associated with this type of construction.

3&$0..&/%"5*0/ ±-JNJU UIF VTF PG D

Due to the risk of mishandling ‡ “ ‘ ” “ ™ ’ ^ • • ˜ %their use must be limited, documented and special attention must be paid to their scope.

#BE FYBNQMF

```
OB M + H m i b2 / B Q X ? =
OB M + H m i b2 / B M i X ? =
O/ 2 7 B MJ2 s R y

b i ` m + T Q B M &i
m B M i 3 n-ivc
' c
```

```
B M iK  BUp Q BV//
&
  m B M i 3 Bd
  b i ` m + TQ B M i  #( J  s) c
  7 Q `UB 4  y cB l  J  sc  BY Y V &
    i  #( B)  4  Ub i ` m + TQ B Wl &- k  B' c
  '
  7 Q `UB 4  y cB l  J  sc  BY Y V &
    T ` B MUJ7W $M - i  #( B) @t=V c f  m M / 2 7 B M 2? p B Q#m2`+  m bi 2 2 + Q K T Q mHB/ 2 `  H
      / 2 7 B M 2B/Mi ? 2 T ` 2 p B Q Hm Q Q/TQ 2 bM Q i2 t B bi M v K Q ` f2
  '
  X X X
'
```

(PPE FYBNQMF

```
OB M + H mi b2 / B Q X ? =
OB M + H mi b2 / B M i X ? =

b i ` m + TQ B M &i
  m B M i 3 n- iv c
' c

O/ 2 7 B MJ2 s  R y
B M iK  BUp Q BV//
&
  m B M i 3 Bd
  b i ` m + TQ B M i #( J  s) c
  7 Q `UB 4  y cB l  J  sc  BY Y V &
    i  #( B)  4  Ub i ` m + TQ B Wl &- k  B' c
  '
  7 Q `UB 4  y cB l  J  sc  BY Y V &
    T ` B MUJ7W $M - i  #( B) Xt V c
  '
  X X X
'
```

3FGFSFODFT

[Cert] Rec. DCL21-C Understand the storage of compound literals.
[Cert] Rule DCL30-C Declare objects with appropriate storage durations.
[IsoSecu] Escaping of the address of an automatic object [addrescape].

&OVNFSBUJPOT

The non-explicit value of a constant in an enumeration is 1 higher than the value of the previous constant. If the first constant value is not explicit then it is 0. If all the values in the enumeration are implicit, no problem arises, but if the developer makes certain values explicit, an error is possible. It is therefore better to avoid mixing constants with explicit and implicit values. If constants of the same enumeration have the same value, this leads to undefined behaviour. If values are made explicit then all values of the enumeration constants must be made explicit to ensure that none of the given values are repeated.

36-&±%POPUNJYFYQMJDJUBOEJN

Either all the constants in an enumeration must be made explicit with a single value, or none at all.

The constants of an enumeration are also subject to the rules of section 6.3, such as the use of upper case for the declaration of constants.

A use sometimes observed around enumerations is the declaration of anonymous enumerations for the declaration of constants. For example:

```
2 M m&K
  w 1 _-P
  P L 1
' c
```

is used instead of:

```
+ Q M bBi M iw 1 _4Py c
+ Q M bBi M iP L 4 R c
```

Enumerations are not made for this purpose; it is a misuse that can make the code harder to understand.

36-&±%POPUVTFBOPOZNPVTFOVN

#BEFYBNQMF

```
2 M mK M 2 n 2 M&m K
  2 M m4KRR-
  2 M m-K k
  2 M m-K j
  2 M m4Kj9f   2 M m K29 2 M m KQj M iH  K ü K 2  H 2 m f
' c
```

(PPEFYBNQMF

All constants have a unique value and are in upper case.

```
2 M mK M 2 n 2 M&m K
  1 L l J4Ry -
  1 L l J4kR -
  1 L l J4 k -
  1 L l J49j
' c
```

3FGFSFODFT

[Misra2012] Rule 8.12 Within an enumerator list, the value of an implicitly-specified enumeration shall be unique.
[Cert] Rec. INT09-C Ensure enumeration constants maps to unique values.

*OJUJBMJTJOH WBSJB

If variables are not initialised on declaration, there is a risk of using the variable when it has not been initialised. The behaviour is then undefined.

*OGPSNBUJPO

Global and static variables are automatically initialised when they are defined, but with a default value specified by the standard. Due to the possible lack of knowledge of these default values, it is recommended to explicitly initialise all variables.

One easy way to ensure this is to do it systematically when declaring a variable if it is declared alone, or immediately after declaration for multiple declarations.

3&$0..&/%"5*0/ ± 7BSJBCMFT TIPVME BGUFS EFDMBSBUJPO

All variables should be systematically initialised when they are declared, or immediately afterwards in the case of multiple declarations.

*OGPSNBUJPO

The compiler can detect certain missing initialisations. GCC for example provides the @ q m M B M B i ptiHnB Sub section 5.3.3 gives more details and also highlights the limits of such options. In particular, the absence of warning raised by this option is not sufficient to guarantee that all variables are properly initialized.

#BE FYBNQMF

In the following example, variables are not initialized when they are used.

```
f   /2+H  `  iBQBMMi?2#Q/vQ7   7mM+iBQM
mBMijkni c
mBMijkn#c
mBMijkn+c

  4#Y+cf   p  `B  #H2b2 mb2#mirBi?QmBiMBiB  HBb  iBQM
```

(PPE FYBNQMF

In the following code, the variables are correctly initialised before being used (as soon as they are declared here).

```
f   /2+H  `  iBQBMMi?2#Q/vQ7i?27mM+iBQM
mBMijkni 4  yc
mBMijkn# 4  yc
mBMijkn+ 4  yc

  4#Y+c
```

3FGFSFODFT

[Misra2012] Rule 9.1: The value of an object with automatic storage duration shall not be read before it has been set.
[Cert] Rule Exp33-C Do not read uninitialized memory.
[Cwe] CWE-457 Use of uninitialized variable.
[Cwe] CWE-758 Reliance on undefined, unspecified, or Implementation-defined behavior.
[Cwe] CWE-908 Use of uninitialized Resource.
[IsoSecu] Referencing uninitialized memory [uninitref].

*OJUJBMJTBUJPO PG T

The C language offers multiple possibilities for initialising arrays, structures and other structured variables. As there are many such possibilities, they can be confusing and can also be misinterpreted.

36-&

36-& ± 6TF POMZ POF JOJUJBMJTBUJ

For the initialisation of a structured variable, only one initialisation syntax must be chosen and used.

#BE FYBNQMF

```
BMii #(Ry) 4 & y- (9) 4 j- 8- e- (R) 4 R- k 'c
bi`m+ivT2nQ 4 & X4 Ry- y]-#Q]#'c
```

(PPE FYBNQMF

```
BMii #(Ry) 4 & y- R- k- j- 8- e- y- y- y- y 'c
bi`m+ivT2nQ 4 & Ry- y]#Q]#'c
bi`m+ivT2niT 4 & X4 Ry-#X4 y- X 4]#Q]#'c
```

An initialisation of structured variables often used and accepted is:

```
BMii #(L) 4 &y'c
mM2nbi`m+bm`42&y'c
```

This initialisation ensures that …elements/fields of the structured variable are initialised to zero.

8BSOJOH

However, the semantics of this notation should not be misunderstood:

```
BMii #(L) 4 &R'c /Q2bMQiK2 M? i HHi?22H2K2Mi2R-#mii? i HH`2 i
     x2`QM/QMHiv?27B`bi2H2K2Mi2R f
```

This is because, in case of an incomplete initialisation of a structured variable (• Àif‰ À not all fields/elements are explicitly initialised), then the unlisted fields/elements are initialised at 0 by default. Please note that this initialisation does not extend to the padding space.

In addition, C99 introduced the possibility of initialising (a) given element(s) of an array, which adds yet another possible source of error and confusion or even multiple initialisations of the same elements with potentially different values.

36-&

36-&±4USVDUVSFE WBSJBCMFT NVTU JOJUJBMJTBUJPO WBMVF BOE FBDI ¾ CF JOJUJBMJTFE

Non-scalar variables must be initialised explicitly: each element must be initialised with a clear identification, without superfluous initialisation values. Alternatively, the & yinitializer can be used in the declaration. Finally, arrays must have their size explicitly set when initializing them.

#BE FYBNQMF

The initialisations are not precise in the following example: there are non-explicit initialisations of the elements of the structured variables and superfluous initialisation values.

```
BMijknv(8) 4 &R- k- jfci?2BMBiB HBb BBbQMbH2 /B?M2;2@BM`2 HBivi?2H biirQ2H2K2Mib2BMBiB HBbi2x2`Qf

BMijknx(k) 4 &R- k- jfc b #Qp2@BM`2 HBiv?2p Hmj2BbB;MQ`2f

BMiRepnp(8) 4 & (y) 4 @k- (R) 4 @N- (j) 4 @3- (fk)b4QRk3+Q272``Q-`i?2
    BM/2t2kb M/j `2MQiBMBM+`2 bBQM/2`M/9Bb7Q`;Qii2fM

bi`m+iT2`bQ&M
  mMbB;M+2/`M K(2ky)c
  mBMiRe`QiHcH
  7HQ K `Fdb
  BMi;` /2k(Ry)c
'c

bi`m+iT2`bQTMR4 & ]- y'cf Q#b+m`f2
bi`m+iT2`bQTMk4 &iQiQ-ed-d3Xj-&y'-Rfk'2p2`vi?BMb+Q``2+iBvMBiB HBb2/
    BM+Hm/BM;
 HHi?22H2K2MQ7bi?2 `` v;` /2br?B+?`2b2iiQy-#miRkBbB;MQ`2f
```

(PPE FYBNQMF

Initialisations are now explicit and encompass all elements of the structured variables without superfluous initialisation values.

```
BMijknv(8) 4 & R- k- j- 9- 8f 'c7mHBMBiB HBb iBQM

BMijknx(k) 4 & R- k fc MQbmT2`7Hm2QHm2k2Mfb

BMijknr(j) 4 & y 'cf ++2Ti2MQi iBQQMBMBiB HBHbb2H2K2MiBbi?i?2
    pHmy2 f

BMiRepnp(8) 4 & (y) 4 @k- (R) 4 @N- (k) 4 R3- (j) 4 @3f (Q)F4 jj 'c

bi`m+iT2`bQ&M
  mMbB;M+2/`M K(2ky)c
  mBMiRe`QiHcH
  7HQ K `Fdb
  BMi;` /2k(Ry)c
'c

bi`m+iT2`bQTMR4 & M K24 jiBiB- X`QH4 Rk-KX`Fb4 RyX-y XMQi24 &y''c
f HH2H2K2Mib2 2tTHB+BiBHMBiB HBbf2/
```

bi`m+T2`bQTMk 4 & M K4 iQiQ- X`QH4Hed-KX`F4d3Xj-;X/2k4&y''cf MQi?2`
2t KTHQ27`2+Q;MBBb2MBiB HBb-iB?QBMiBK2rBi?QmbimT2`7Hm2QHm2k2Mib
f

3FGFSFODFT

[Misra2012] Rule 9.2 The initializer for an aggregate or union shall be enclosed in braces.
[Misra2012] Rule 9.3 Arrays shall not be partially initialized.
[Misra2012] Rule 9.4 An element of an object shall not be initialized more than once.
[Misra2012] Rule 9.5 Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly.
[Cert] Rec. ARR02-C Explicitly specify array bounds, even if implicitly defined by an array initializer.
[Cwe] CWE-665 Incorrect or incomplete initialization.

.BOEBUPSZ VTF PG EF

When identifiers are declared but are not used afterwards, it may mean that the developer made a mistake when writing the code and that one element was used instead of another or that its use was removed from the program.

*OGPSNBUJPO

GCC and CLANG compilation options such as @qmMmb2/@ and B #H2 @qmMmb2/@ T`kt2i2`ible to detect this kind of patterns.

3&$0..&/%"5*0/±&WFSZ EFDMBSBU

All declared identifiers must be used, whether they are variables, functions, labels, function parameters or anything.

8BSOJOH

When developing a library, not all declared identifiers are necessarily used : functions and variables exported by the library may obviously not be used by the library itself.

#BE FYBNQMF

In the following code, variables declared but not used must be deleted.

```
mBMijkBiMBinHUBHBibini TTnHBVbi&
    HBbiniTnHBb4iLlGcG
    HBbin2H2K2MTmi2H2K24MLilGcG
    mBMijkmiBjknHBbim4Hy2M

    B7ULlGc44TTnHBVbi&
        `2im`Mc
    '

    UTTnHBVbi4 HBbinVK HHQbBx2QH7BbinVic
```

```
B7ULIGG44 UTTnHBVbVi &
    `2im`Mc
'

UTTnHBVb@T=n?2 4 LIGcG
UTTnHBVb@T=ni B4 LIGcG

`2im`Rc
'
```

(PPE FYBNQMF

In the following example, all declared variables are used.

```
mBMijkBiMBinHUBHBibini TTnHBVbi&
    B7ULIGG44 TTnHBVbi&
        `2im`Mc
    '

    UTTnHBVbi4 HBbinVK HHUQb+Bx2UQH7BbinVic

    B7ULIGG44 UTTnHBVbVi &
        `2im`Mc
    '

    UTTnHBVb@T=n?2 4 LIGcG
    UTTnHBVb@T=ni B4 LIGcG

    `2im`Rc
'
```

3FGFSFODFT

[Misra2012] Rule 2.2 There shall be no dead code.
[Misra2012] Rule 2.3 A project should not contain unused type declarations.
[Misra2012] Rule 2.4 A project should not contain unused tag declarations.
[Misra2012] Rule 2.5 A project should not contain unused macro declarations.
[Misra2012] Rule 2.6 A project should not contain unused label declarations.
[Misra2012] Rule 2.7 There should be no unused parameters in functions.
[Cert] Rec. MSC07-C Detect and remove dead code.
[Cert] Rec. MSC13-C Detect and remove unused values.
[Cert] Rec. MSC12-C Detect and remove code that has no effect or is never executed.

/BNJOH PG WBSJBCMFT

It is imperative to use separate variables used to store sensitive and non-sensitive data. In the absence of a well-defined naming convention, the developer risks using variables to successively store sensitive and non-sensitive data.

36-&

36-& ± 6TF TFQBSBUF WBSJBCMFT GP

Separate variables should also be used for unencrypted sensitive data and sensitive data protected in confidentiality and/or integrity.

36-& ± 6TF EJGGFSFOU WBSJBCMFT DPO¾EFOUJBMJUZ BOE PSJOUFHSJ EBUB

These rules are more of a principle of secure coding to avoid handling non-sensitive, encrypted sensitive and unencrypted sensitive data in the same variable.

It goes without saying that hard-coding any sensitive information of any kind (password, login, encryption key, ‰) is À forbidden.

36-& ± /FWFS IBSE DPEF TFOTJUJWF

#BE FYBNQMF

The code below does not use a naming convention.

```
O/27BME21unaAjwk1
O/27BM"2661_naA8wR1k

bBx2ni F2vnH24Myc
bBx2ni +H2 `n/i RnH42yMc
bBx2ni2M+`vTi2/n/i krH4Hy2cM
mBMi3nF2(vE1unaAcw1
mBMi3n/ii R("l661_naAcw1
mBMi3n/ii k("l661_naAcw1
mBMijk2i`Q`n+Q4/2ycM
2``Q`n+Q4/2-BT?2`n/U-iH2 `n/-i+H2 `n/ inH2-2M-v F2vnH-2M
2M+`vTi2/n/ 2M+`vTi2/n/ iv H2M
```

(PPE FYBNQMF

In the following example, a naming convention is used so that the same variables are not used for encrypted or unencrypted sensitive data.

```
O/27BME21unaAjwk1
O/27BM"2661_naA8wR1k

f +QMp2MiB,QMb
bm77Bbt7Q`b2MbBiB/pi2 p `B #Hi2b
+H2 T`27Bt7Q`mM2M+`vTi2/ii
2M+`vTi2/27B7Q`2M+`vTi2/i f

bBx2ni2M+`vTi2/nF2vnH24MybM nb
bBx2ni+H2 `n/inH24MybM nb
bBx2ni2M+`vTi2/n/inH42MybM nb
mBMi3n2M+`vTi2/nF(2Ev1unaAcw1
mBMi3n+H2 `n/i(n"b661_naAcw1
mBMi3n2M+`vTi2/n/ ("hb661_naAcw1
mBMijk2iM+`vTi2/n2``Q`n+4Qy2nb

2M+`vTi2/2``Q`4+Q/2T?2`n/U-iH2 `n/i-nb+H2 `n/ inH2MnbM
2M+`vTi2/nF-22MbM+`vTi2/nF2vnH-22MM+nbvTi2/n/-i 2bM+`vTi2/n/ inH2MnbM
```

3FGFSFODFT

[Misra2012] Dir. 4.5 Identifiers in the same name space with overlapping visibility should be typographically unambiguous.
[Cert] Rec. DCL02-C Use visually distinct identifiers.
[Cert] Rule MSC41-C Never hard code sensitive information.
[Cwe] CWE-259 Use of Hard-Coded Password.
[Cwe] CWE-798 Use of Hard-Coded Credentials.

7
Types and type conversions

&YQMJDJUTJ[FGPSJO

8BSOJOH

The C standard does not define an explicit size for each type of integer. In particular, for the B Mtype, depending on the architecture, it can be on 16, 32 or 64 bits.

Therefore, use of the B Mtype is risky since it is necessary to be sure of the associated size and possible values to avoid any overflow or unexpected behaviour such as a value wrap (for unsigned integers).

It is therefore best to avoid using this type unless the developer is certain that the associated value range is contained within its range (e.g. in loop counters).

The type name must include its size on the target machine explicitly, like those defined in the header file b i / B M j available in C99. Its use is to be preferred to the generic B Mtype. In C90, equivalent types must be defined and used. The redefinition of integer types is possible but this redefinition must be explicit on both the associated size and sign.

3&$0../%"5*0/±0OMZJOUFHFSUZTIPVMECFVTFE

Furthermore, the plain + ? type should not be used for numeric values as its sign is not specified by the C standard and is implementation-defined. This type must be restricted to character handling.

36-&±b0BO;M2/+B?0nEMbB;M2/U+Z?QFTNVTVTFEUPIBOEMFOVNFSJDWBMVFT

#BEFYBNQMF

```
O/27BMJ2slALhRe e88j8l
BMip Hmc2
+? `+ 4 j8cf bB;MBbMQibT2+B7B2/
```

```
B 7 Up  H m=24 J  s l A L h VRe
&
  f   /2 T 2 M / BQMMi ? 2  `+ ? B i 2 + i m i 2
'
```

(P P E F Y B N Q M F

```
OB M + H m i b 2 / B M i X ? = B 7 * N N f
O/ 2 7 B MJ 2 s l A L h Re  e 8 8 j 8 l
m M b B ; M+ 2 2 / `+ 4  j 8 c f   b B ; M B b 2 t T H B + B 7 i Q `M m K 2 `p B + H m i 2  M B T m H  i B Q M
X X X
m B M i j k n p  H m c 2 f   B 7 * N N f

i v T 2 / 2 7 m M b B ; M+ 2 2 / `m B M i 3 n f  i v T 2 / 2 7 B M B i B Q M M N y f
B 7 Up  H m=24 J  s l A L h VRe
&
  f   X X X f
'
```

3 F G F S F O D F T

[Misra2012] Rule 10.1 Operands shall not be of an inappropriate essential type.
[Misra2012] Rule 10.3 The value of an expression should not be assigned to an object with a narrower essential type or of a different essential type category.
[Misra2012] Rule 10.4 Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.
[Misra2012] Rule 8.1 Types shall be explicitly specified.
[Misra2012] Directive 4.6 i v T 2 / 2 7 at indicate size and signedness should be used in place of the basic numerical types.
[Cert] Rec. INT00-C Understand the data model used by your implementation(s).
[Cert] Rec. INT07-C Use only explicitly signed or unsigned char type for numeric values.
[Cert] Rule INT35-C Use correct integer precisions.
[Cert] Rec STR00-C Represent characters using an appropriate type.
[Cwe] CWE-682 Incorrect calculation.

5 Z Q F B M J B T

The i v T 2 / 2 7 operator allows the redefinition of a type that has itself been redefined using i v T 2 / . 2 7
It is then difficult to follow the actual type checking of a variable. There is a significant risk of confusion and duplication of type definitions. The types defined by the language or by external libraries should therefore not be redefined more than once. If this is done intentionally by the developer for — ˜ – “ ’ ‹ ˜ • ” ‰ ± ¡ should be commented on and explained, but also controlled to avoid any type confusion.

3 & $ 0 . . & / % " 5 * 0 / ± % P O P U S F E F ¾ O F U Z

#BE FYBNQMF

The following example (C90) shows multiple aliases of the same type.

```
ivT2/27mMbB;M2?/Q`mBMiRemi/27BMBiBQQ7MvT2mBMiRemi
ivT2/27mMbB;M2?/Q`mBMiRencvfT2ivT2mBMiReniBvfT2M HB bQ7ivT2
    mBMiRemi
ivT2/27mBMiRemmMbB;M2/nbQ7QivT2mMbB;M2/nbB?bQ`i2/27BMBiBQQ7MvT2
    mBMiRemi
```

(PPE FYBNQMF

In the following example (in C90, • À‰ before the introduction of bi/BM), only one type is correctly defined from the definition of a standard type of the language.

```
ivT2/27mMbB;M2?/Q`mBMiRemi/27BMBiBQQ7MvT2mBMiRemi
```

3FGFSFODFT

[Cert] Rec. PRE03-C Prefer typedefs to defines for encoding non-pointer types.

5ZQF DPOWFSTJPOT

C compilers perform implicit conversions from one type to another. However, these type promotions and implicit conversions can lead to errors (loss of information, miscalculations). In addition, the absence of explicit conversions does not facilitate proofreading of the code. It is therefore necessary to add the type conversion operator systematically and not to mix signed and unsigned types in the same arithmetic operation (operators: +, -, *, /, %, ~, >, <, >=, <=, <<, >>, ‰)‡À

Multiple implicit conversions are made, whether in C90 or C99.

On the one hand, •'˜‰‹‰–"– is performed on integer values whose type is smaller than the BMtype and when these integer values are subjected to an operation (binary operators, unary operators, shifts, ‰)‡À These integer values are then automatically and systematically converted to BMor mMbB;M2/BMi

On the other hand, ˜•"‰†…• corresponds to the usual conversion to a common type when operands are of different types. Finally, the last implicit conversion corresponds to the assignment of a value in a different type.

*OGPSNBUJPO

Details of integer promotion can be found in sections 6.2.1.1. and 6.3.1.1. of standards [AnsiC90] and [AnsiC99] respectively. For type balancing, the relevant sections are 6.2.1.5 and 6.3.1.8 of standards [AnsiC90] and [AnsiC99] respectively.

36-& – %FUBJMFE BOE QSFDJTF VOEF

The developer needs to know and understand all the implicit conversion rules for integer types.

The developer must make explicit the conversions implicit in the code to avoid any errors. The classic case that is often a source of errors is an implicit conversion between signed and unsigned types.

36-& – &YQMJDJU DPOWFSTJPOT CFU

Prohibit implicit type conversions. Use explicit conversions, particularly between signed and unsigned types.

#BE FYBNQMF

```
bB;M2B/MipR4 @Rc
mMbB;MB2/Mipk4 Rc
B7UpRI pkV
&
  f  pR+QMp2`ii2Q/mMbB;MB2/Mi M/p Hm@R#2+QK2AbLhnJ-si?2`27Qi22B7
     +QM/BiBBQbMHr vb7 Hb2f
'
```

(PPE FYBNQMF

```
bB;M2B/MipR4 @Rc
mMbB;MB2/Mipk4 Rc
B7UpRI UbB;M2B/MV/pkV
&
  f  pkBb2tTHB+Bi+HQvMp2`ii2Q/ bB;M2B/Mi2;2@i?2+QM/BiBBQbM`m2f
'
```

Again for the same reasons, no implicit conversion should be made between an integer type and a floating type or from an integer type to a smaller integer type.

#BE FYBNQMF

In the following lines, conversions are implicit.

```
mBMijkmijkc
BMijknbjkc
mBMiRemmRce
/Qm#H#2Hc
mBMi3B/tc

bjk4 9kc
mjk4 bjkc f  BKTHB++BQiMp2`bBfQM
mRe4 mjkY k  bjkc f  BKTHB++BQiMp2`bBQQMbK HH2vT2f
/#H4 mjkf mRcef i?2`2bmHBiby UBMi2;2/`BpBbBVQfM
bjk4 /#Hc f  BKTHB++BQiMp2`bBkQQM i@=BMi2;2`f

f  i?27QHHQrBkQ/QBTbBM7BM,BB2t#2BMm/MbB;MB2/t=4y Bb Hr vbi`m2
   bBM+2MmMbB;Mp2/Hm+2 MMQ#2M2; iBp2f
7QUB/t4 kdcB/t=4 ycB/t@ @V &
  XXX
'
```

5PMFSBUFE FYBNQMF

The following example shows a code in which the type conversions are explicit. This is called a tolerated example because other cleaner solutions such as incrementing the loop index exist.

```
m B M i j k mijk
B M i j k nbjk
m B M i R emRce
/ Q m # H 2Hd
m B M i 3 B/ tc

b j k 4 9 k c
m j k 4 Um B M i j k Vbjkc
m R e4 Um B M i R V e bUBM i j k Vmjk Y k bjkV c
/ # H 4 U/ Q m # V H m 2R e U/ Q m # V H m 2j k

f i ? 2 b B ; M 2B/ M i 2 ; 2+` b i B b m b 2 /Q p Q B / M B M 7 B M BHiQ2Q T
7 QUB / t 4 k d c BUM i 3 VB / t = 4 y cB / t@ @ V &
X X X
'
```

(PPE FYBNQMF

```
B M i j k nbjkc
m B M i R emRce
/ Q m # H 2Hd
m B M i 3 B/ tc

b j k 4 9 k c
m j k 4 Um B M i j k Vbjkc
m R e4 Um B M i R V e bUBM i j k Vmjk Y k bjkV c
/ # H 4 U/ Q m # V H m 2R e U/ Q m # V H m 2j k

f + ? M ; Q 7 i ? 2 H Q Q T
B / t4 k d c
r ? B H Q B / t= y V
&
X X X
'
```

*OGPSNBUJPO

@ q + Q M p 2and B@QqM B ; M @ + Q Mwarnings provided by GCC and CLANG can help to detect such implicit conversions.

3FGFSFODFT

[Misra2012] Rule 10.1 Operands shall not be of an inappropriate essential type.
[Misra2012] Rule 10.3 The value of an expression should not be assigned to an object with a narrower essential type or of a different essential type category.
[Misra2012] Rule 10.4 Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.
[Misra2012] Rule 10.5 The value of an expression should not be cast to an inappropriate essential type.
[Misra2012] Rule 10.6 The value of a composite expression shall not be assigned to an object with wider essential type.

[Misra2012] Rule 10.7 If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type.
[Misra2012] Rule 10.8 The value of a composite expression shall not be cast to a different essential type category or a wider essential type.
[Cert] Rec. INT02-C Understand integer conversion rules.
[Cert] Rule INT30-C Ensure that unsigned integer operation do not wrap.
[Cert] Rule INT31-C Ensure that integer conversions do not result in lost or misinterpreted data.
[Cert] Rule INT32-C Ensure that operations on signed integers do not result in overflow.
[Cert] Rec. INT18-C Evaluate integer expressions in a larger size before comparing or assigning to that size.
[Cert] Rec. EXP14-C Beware of integer promotion when performing bitwise operations on integer types smaller than int.
[Cwe] CWE-190 Integer overflow or wraparound.
[Cwe] CWE-192 Integer coercion error.
[Cwe] CWE-197 Numeric Truncation Error.
[Cwe] CWE-681 Incorrect conversion between numerical types.
[Cwe] CWE-704 Incorrect Type Conversion or Cast.
[IsoSecu] Conversion of signed characters to wider integer types before a check for EOF [signconv].
[IsoSecu] Overflowing signed integers [intoflow].

5ZQF DPOWFSTJPO PG Q EJGGFSFOU UZQFT

Type conversion from or to structured variables via pointers can result in overflows when the target type is larger than the memory area pointed to. Indeed, a type conversion from one structure to a larger structure, for example, gives undesirable access to areas of memory outside the initial structure.

In addition, type conversion from/to structured variables makes proofreading the code more complex.

3&$0..&/%"5*0/±%P OPU VTF QPJOUF UVSFE EJGGFSFOUMZ

#BE FYBNQMF

In the following code, a structure type conversion will result in an overflow.

```
O/27BMh2"naAw1 Rel

ivT2/2Ti`m+&
  BMijknK ;Bc+
' bnc

ivT2/2Ti`m+&
```

```
  BMijknK ;Bc+
  BMiRebdc
  mBMi3n(ih "naA)vc1
' bn#

pQB7QUQbn  bi`m+V &
  bn# T 4 Ubn#Vbi`m+c f  ivT2+QMp2`bBQQ#M22t+Hm/2f/

  T@K= ;B+4 yt"  .* 6dc
  T@b= 4 yt.1 .c f  Qp2`7HQmibBQ27i?2bi`m+imb2n f
  T@t=(y) 4 9d   M/`BbFQ7Qp2`r`Bii2M U#m772Qp2`7HVQf
'
```

(PPE FYBNQMF

In the following code, the structure type conversion is no longer performed.

```
O/27BMh2 "naAw1 Rel

ivT2/27bi`m+&
  BMijknK ;Bc+
' bnc

ivT2/27bi`m+&
  bn ?c
  BMiRebdc
  mBMi3n(ih "naA)vc1
' bn#

pQB7QUQbn# bi`m+V# &
  bi`m+@#?XK ;B+4 yt* 61" "c1
  bi`m+@#b= 4 yt"116
  bi`m+@#t=(y) 4 9c
'
```

3FGFSFODFT

[Misra2012] Rule 11.2 Conversions shall not be performed between a pointer to an incomplete type of any other type.
[Misra2012] Rule 11.3 A cast shall not be performed between a pointer to object type and a pointer to a different object type.
[Misra2012] Rule 11.8 A cast shall not remove a const or volatile qualification from the type pointed to by a pointer.
[Cert] Rule EXP36-C Do not cast pointer into more strictly aligned pointer types.
[Cwe] CWE-704 Incorrect type conversion or cast.
[IsoSecu] Converting pointer values to more strictly aligned pointer types [alignconv].

8
Pointers and arrays

In this section we are referring only to one-dimensional arrays, but as any multi-dimensional array can also be represented via a one-dimensional array, all the rules and recommendations therefore also apply to multi-dimensional arrays.

4UBOEBSEJTFE BDDFTT

Confusion between arrays and pointers is common, and it is often assumed that an array behaves as a constant pointer to its first element. This statement is a shortcut that proves to be false in general.

Therefore, for the following code:

```
BMijknii #R(e)c
BMijkni i #k 4 K HHUQe+ bBx2UQB7Mijkn iVc
BMijkni i #j 4 i #Rc
BMijkni i #9 4 i #kc

T`BMUJ7i #R, WT- WT- WT$M]- i #R  i #R(y)- i #RVc
T`BMUJ7i #k, WT- WT- WT$M]- i #k-  i #k(y)- i #kVc
T`BMUJ7i #j, WT- WT- WT$M]- i #j-  i #j(y)- i #jVc
T`BMUJ7i #9, WT- WT- WT$M]- i #9-  i #9(y)- i #9Vc
```

the result obtained is as follows:

```
i #R, R88Nk93Nk3- R88Nk93Nk3- fR88#NRk93#NRky3)4i #R HH`2T`2b2M?i2 //`2bbQ7
                                                i?27B`bi2H2K2BMi?2 `` v f
i #k, NRRkN8ydk- NRRkN8ydk- R88Nik9 3Nky9?2 //`2bbQ7i?2TQBMi?22im`M2#v
                                                K HH-Q+TQBMiBiQj?2 `` v- M/i #kUQ` i #k(y)V
                                                Bbi?2 //`2bbQ7i?27B`bi2H2K2BMi?2 `` v
                                                i #k f
i #j, R88Nk93Nk3- R88Nk93Nk3- fR88BNkB9 3+NR2iQ i #k f
i #9, NRRkN8ydk- NRRkN8ydk- R88N89K3BNk+yb2iQ i #k f
```

The nuances between arrays and pointers are numerous and we can only draw the reader's attention to this point and urge them to tread carefully.

Another example of a confusing code is:

```
BMip `(L)c
BMiUp `kV(L)c
```

The first line involves declaring L BM type pointers in memory, • À‰ array of L BM type pointers. The second line declares a pointer to an array of L BM type elements in memory.

The standard clarifies this point by explaining that any array type ‰ œ" – % is converted to a pointer type expression pointing to the first element of the array and is not an • š … except when

the initial expression is used as an operand of the operators b B x 2 Q 7H B ; MrQ 7r if the expression is a literal string used to initialise an array.

&YQSFTTJPO

An expression is not an object in memory but a piece of source code such as Y #r , for example.

-WBMVF

An • š …(location value) is an expression with a type, even incomplete, but different from void which is associated with an address in memory.

An array is not a modifiable • š …, which means that it cannot be assigned, incremented or modified in general.

```
B M ii  #( L) c
i  # 4  y cff  2 ` ` Q `
i  #@ @fd  2 ` ` Q `
```

When the array expression is converted to a pointer type expression, this expression then produces a simple value and is no longer an • š …• ™ ‰

8BSOJOH

For an array i #, the i #and i #(y) notations represent the address of the first element of the array created in memory. The i #notation, on the other hand, will vary. When an array is declared statically, the array address cannot change and there is no pointer creation as such on the array: the i #notation is similar to a label managed by the compiler containing the address of the array.

- Therefore, if the array is declared statically (case of i #in the previous example), i #always represents the address of the first element of the array, • À‰ the address of the array.
- On the other hand, if the array is declared dynamically, the i #notation represents the pointer containing the address of the array created in memory and therefore, i #represents the address of the pointer to the array (case of i #in the example).

The C standard allows access to the iþelement of a i #array to be represented in a variety of ways, which can be a source of errors or confusion.

8BSOJOH

For a tab array, access to the iþelement can be written:

```
  Ui  #YBVcf  mbm MHQi iBQRMf
i  #( B) c f  mbm MHQi iBQRMf
  UBYi  #Vcf  BMi2 ` + ?  M ; 2  #`H2  M / BM / 25  f
B( i  #) c f  BMi2 ` + ?  M ; 2  #`H2  M / BM / 25  f
```

These notations are all recognised by the standard and are therefore correct, but this can quickly reduce understanding of the code. Note that, even with demanding compilation options, neither GCC nor CLANG will issue alerts on these types of notations,

which can be a source of errors.

In order to avoid any ambiguity and misunderstanding of the code and therefore potential errors, notations tolerated by the standard and intended to invert the index and the name of an array will not be used.

36-&±"DDFTTUPUIFFMFNFOUPGBBTUIF¾STUBUUSJCVUFUIFBSSBZBFMFNFOUDPODFSOFE

Access to the iþelement of an array will always be written with the name of the array first followed by the index of the element to be reached.

Furthermore, again for reasons of transparency, the typical notation of the arrays using square brackets ( )will be preferred.

3&$0..&/%"5*0/±"DDFTTUPFMFNFOUTRVBSFCSBDLFUT

In the case of an array type variable, the dedicated notation (via square brackets) must be used to avoid any ambiguity.

#BEFYBNQMF

```
7Q`UB 4 ycBI bBx2nic#BYYV &
   UBYi #V 4Bc f i?2b[m `2#` +F2ib`2MQimb2/M/i?2BM/2BbBMi?27B`bi
      TQbBiBQM
  XXX
'
```

(PPEFYBNQMF

```
7Q`UB 4 ycBI bBx2nic#BYYV &
  i #(B) 4 Bc
  XXX
'
```

3FGFSFODFT

[Cert] ARR00-C Understand how arrays work.

/PO VTF PG 7-"T

The VLAs[9] introduced with C99 correspond to arrays whose size is not associated with a constant integer expression at compilation but with an integer variable. This therefore corresponds to implementing an object of variable size on the stack. If the size of the array is not strictly positive,

9. Variable-Length Array

3FGFSFODFT

[Misra2012] Rule 8.11 When an array with external linkage is declared, its size should be explicitly specified.
[Misra2012] Rule 9.5 Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly.
[Cert] Rule ARR30-C Do not form or use out-of-bounds pointers or array subscripts.
[Cert] Rule ARR02-C Explicitly specify array bounds, even if implicitly defined by an initializer.
[Cwe] CWE-655 Incorrect or incomplete initialization.

4ZTUFNBUJD DIFDL GPS

Accessing an array element outside the allocated size is a classic coding error. For each access to the element of an array, it must be checked whether the index used is strictly positive or null and strictly less than the number of elements allocated for that array.

36-&

36-& ± 6TF VOTJHOFE JOUFHFST GPS

36-&

36-& ± %P OPU BDDFTT BO BSSBZ FMF UIF VTFE JOEFY

The validity of the used array index must be checked systematically: an array index is valid if it is greater than or equal to zero and strictly less than the declared size of the array. In the case of a character array, the end of string character '\0' must be taken into account.

#BE FYBNQMF

```
BY Y c
i  #( B)  4  Bc  f    M QQ p 2 ` 7 H-Q? 2 + Ff
```

(PPE FYBNQMF

```
7 Q `UB 4  y cB I  b B x 2 n ic #BY Y V &  b B x 2 n i B#bi ? 2 M m K # Q 7 2 H 2 K 2 MBiMi ? 2  ``  v
      f
  i  #( B)  4  Bc
  X X X
'
```

3FGFSFODFT

[Cert] Rec. ARR02-C Explicitly specify array bounds, even if implicitly defined by an initializer.
[Cert] Rule ARR30-C Do not form or use out-of-bounds pointers or array subscripts.

[Cert] Rule STR31-C Guarantee that storage for strings has sufficient space for character data and the null terminator.
[IsoSecu] Tainted, potentially mutilated, or out-of-domain integer values are used in a restricted sink [taintsink].
[IsoSecu] Forming or using out-of-bound pointers or array subscripts [invptr].
[IsoSecu] Using a tainted value to write to an object using a formatted input or output function [taintformatio].
[IsoSecu] Tainted strings are passed to a string copying function [taintstrcpy].
[Cwe] CWE-119 Improper Restriction of Operations within the bounds of a Memory buffer.
[Cwe] CWE-120 Buffer Copy without Checking Sizeof Input (Classic Buffer Overflow).
[Cwe] CWE-123 Write-what-where Condition.
[Cwe] CWE-125 Out-of-bounds read.
[Cwe] CWE-129 Improper Validation of Array Index.
[Cwe] CWE-170 Improper Null termination.

% P O P U E F S F G F S F O D F

Dereferencing a pointer leads to u i

3FGFSFODFT

[Cert] Rule EXP34-C Do not dereference null pointers.
[IsoSecu] Dereferencing an out-of-domain pointer [nullref].
[Cwe] CWE-476 NULL Pointer Dereference.
[AnsiC99] Section 6.5.3.2.
[AnsiC90] Section 6.3.3.3.

"TTJHONFOU UP /6-- PG

Following the deallocation of the memory pointed to by a pointer, the pointer variable still stores its address. This is known as a ˆ … ' ‹ • • ' ‹ " " • ' ˜ ‰ –

%BOHMJOH QPJOUFS

A ˆ … ' ‹ • • ' ‹ " " is a ‰ pointer that contains the memory address of an element that has been freed.

In case of bugs and incorrect use of the deallocated pointer, the memory may be corrupted. Once freed, the memory may (or may not) be reused by the system. The result of the use of the memory area (via the pointer) is then undefined and not necessarily visible, and may cause security problems (™ — ‰ Â .). By assigning the pointer to L I G after deallocation, you can specify that the pointer no longer points to a valid memory area. And in case of an accidental use of the pointer, no memory area will be corrupted since the pointer no longer points to any valid memory area.

36-&

36-& ± " QPJOUFS SLNVBGUCFSBEFFBMMPDE

A pointer must be systematically assigned to L I G following the deallocation of the memory it points to.

#BE FYBNQMF

In the code below, the pointer is not set to NULL following its deallocation.

```
H B b i n i T n H B b 4 i L I G C G
T n H B b 4 i + ` 2  i 2 n H B b V i c
X X X
B 7 U T n H B b 5 i 4 L I G V G  &
   7 ` 2 2 n H B b T b n i H B b V b c i
'
f   b 2 i i B M Q 7 T n H B b i Q L I G G B b K B b b B M ;
```

(PPE FYBNQMF

In the following example, the pointer is correctly set to L I G following the deallocation of the area being pointed to.

```
H B b i n i T n H B b 4 i L I G C G
T n H B b 4 i + ` 2  i 2 n H B b V i c
X X X
B 7 U T n H B b 5 i 4 L I G V G  &
```

7 ` 2 2 n H B T b i H B V d
T n H B b i L I G G

3 F G F S F O D F T

[Cert] Rule MEM30-C Do not access freed memory.
[Cert] Rec MEM01-C Store a new value in pointers immediately after free().
[Misra2012] Rule 18.6. The address of an object with automatic storage shall not be copied to another object that persists after the first object has ceased to exist.
[Cwe] CWE-415. Double free.
[Cwe] CWE-416. Use after free.
[Cwe] CWE-672 Operation on a resource after expiration or release.
[IsoSecu] Accessing freed memory [accfree].
[IsoSecu] Freeing memory multiple times [dbfree].

6 T F 2 P G i U B H F i Z Q F R V B M J ¾

The ` 2 b i ` B qualifier, introduced in C99, is a means of indicating to the compiler that the area being pointed to cannot be accessed without passing via the pointer marked ` 2 b i ` B + i ` 2 b i ` B + i qualified pointer therefore implies that the object it points to is reached directly or indirectly only via this pointer. This requires that there are no other aliases on the object you are pointing to.

" M J B T

Two aliases are two variables or access paths to the same memory area.

8 B S O J O H

The ` 2 b i ` B qualifier is a declaration of the developer's intention to associate a single pointer with a memory area, not an actual fact. In practice, nothing prevents the code from reaching the same area via a different pointer.

The behaviour becomes undefined if objects pointed to by ` 2 b i ` B pointers have common memory addresses. In addition, it is necessary to check for the absence of common memory addresses on each function call with ` 2 b i ` B type parameters, but also during the execution of said functions.

8 B S O J O H

Several functions in the standard library have restrict type parameters since C99 (K 2 K ; T i ` + , i b i ` + , T ‰). ‡ À

It is very easy to introduce undefined behaviour through the use of ` 2 b i ` B, as it must be ensured that none of the pointers concerned share a memory area, while taking into account function calls

from the standard library which also have `2bi`B+type parameters since C99. The use of the `2bi`B+qualifier directly by the user is therefore to be prohibited.

36-&

36-&±%PÒ2bUiÙVBTQiPUJOTUFSRVBMJ¾FS

The `2bi`B+qualifier must not be used directly by the developer. Only indirect use,
•À%aÀthe standard library function call, is tolerated, but the developer must ensure that no undefined behaviour will result from the use of such functions.

*OGPSNBUJPO

The @q`2bi`B+ option makes it possible to alert to the misuse of `2bi`B+i pointers.

#BEFYBNQMF

In the following example, `2bi`B+type pointers share memory areas and therefore cause undefined behaviour.

```
mBMiReniì2bi`B+Ti/2#
mBMiReniì2bi`B+Ti7BcM
mBMiReinì(Rk)c
mMbB;M+2?/`  TiRc
mMbB;M+2?/`  Tikc
mMbB;M+2?/`+nbi() 4]#H  #Hc
XXX
Ti/2#4  i #(y)c
Ti7BM4  i #(RR)c
Ti/2#4 Ti7BcMf  mM/27BM42?  pBQm`
XXX
TiR4 TikY kc
K2K+UTvik TiR jVcf  mM/27BM42?  pBQ@m``2bi`B+iivT2K2K+TTv`  K2i2`bi
```

(PPEFYBNQMF

In the following example, the `2bi`B+qualifiers have been deleted and there is no longer any undefined behaviour.

```
mBMiReniTi/2# f  /2H2iBQM7`2bi`B+[im  HB7B2`
mBMiReniTi7BcMf  /2H2iBQM7`2bi`B+[im  HB7B2`
mBMiReinì(Rk)c
mMbB;M+2?/`  TiRc
mMbB;M+2?/`  Tikc
mMbB;M+2?/`+nbi() 4]#H  #Hc
XXX
Ti/2#4  i #(y)c
Ti7BM4  i #(RR)c
Ti/2#4 Ti7BcMf  QF f
XXX
TiR4 TikY kc
K2KKQTpi2k TiR jVcf  +?  M;Q77mM+iBQM
```

3FGFSFODFT

[Misra2012] Rule 8.14 The restrict type shall not be used.
[IsoSecu] Passing pointers into the same objects as arguments to different restrict-qualified parameters [restrict].

[Cert] Rule EXP43-C Avoid undefined behavior when using restrict-qualified pointers.

-JNJU PO UIF OVNCFS P

When a pointer has more than two levels of indirection (for example: pointer of pointer of pointer B M i j k n i T T T), it becomes difficult to understand the developer's intentions and the behaviour of the code.

3&$0..&/%"5*0/±5IF OVNCFS PG MF CF MJNJUFE UP UXP

The number of levels of indirection for a pointer should not exceed two.

#BE FYBNQMF

The following code shows excessive levels of indirection.

```
pQB7mM+iBBQMi3ni  ``nTVf  j H2p2Hb
&
  BMi3ni  Tic
XXX
'
```

(PPE FYBNQMF

In the following example, temporary pointers are introduced to facilitate access to the data and limit the number of nesting levels.

```
ivT2/2BMi3niBMi3Ti`cni
pQB7mM+iBBQMi3Ti`ni ``nTVf  `2/m+iBQQMirQH2p2Hb
&
  BMi3niTini2kTf  i2KTQ`  TvQBMi2f`
  BMi3Ti`niTic
XXX
'
```

3FGFSFODFT

[Misra2012] Rule 18.5 Declarations should contain no more than two levels of pointer nesting.

(JWF QSFGFSFODF UP

Two writing methods are possible in the C language to reach a structure field via a pointer: the indirection operator T i ` @ = 7 a2Hd and dereferencing U T i ` V X 7. However, the second method is often a source of errors and comprehension problems. It is therefore best to avoid using dereferencing U T i ` V X 7 to reach a field of a structure via a pointer.

3&$0

3&$0..&/%"5*0/±(JWF QSFGFSFODF BU@PS

The indirection operator @should be used to reach the fields of a structure via a pointer.

#BE FYBNQMF

In the following example, the access should be rewritten with the indirection operator.

U HBbXTn?2V/XTL2ti4 LIGcG

(PPE FYBNQMF

In the following code, the indirection operator is correctly used.

HBbXTn?2@TL2ti4 LIGcG

1PJOUFS BSJUINFUJD

The C language allows direct access to the memory using pointers. Arithmetic operations can be applied to the value of a pointer either to increment or decrement it.

1PJOUFS BSJUINFUJD

Pointer arithmetic is the use of pointer values as integer values in an elementary arithmetic operation (subtraction and addition).

Pointer arithmetic is very often used in the case of a pointer to an array element to navigate between the different elements of the array. Apart from this case, arithmetic on memory addresses is very risky.

36-&

36-&±0OMZ JODSFNFOUJOH PS EFDS

Incrementing or decrementing pointers should only be used on pointers representing an array or an element of an array.

Arithmetic on pQBtype pointers is therefore prohibited. No memory size is in fact associated with the pQBtype, which causes undefined behaviour, in addition to the violation of the previous rule.

36-&

36-&±/P BSpJQUBNQFUJDDUFST JT BVUIPSJ

The use of any arithmetic on pQBtype pointers must be prohibited.

Even in the case of pointer arithmetic on elements in an array, a special care must be taken to ensure that the arithmetic will not cause dereferencing outside of the array.

3&$0

3&$0..&/%"5*0/±$POUSPMMFE QPJ

Arithmetic on pointers representing an array or an element of an array must be carried out ensuring that the resulting pointer will still point to an element of the same array.

As a result, subtractions or comparisons between pointers will only be relevant for pointers on the same array.

36-&

36-&±4VCUSBDUJPO BOE DPNQBSJ POMZ

Only subtractions and comparisons of pointers on the same array are authorised.

Finally, the assignment of a fixed address to a pointer is strongly discouraged.

3&$0

3&$0..&/%"5*0/±"¾YFE BEESFTT TI BQPJOUFS

#BE FYBNQMF

```
OBM+Hmlb2//27X?=
OBM+Hmlb2/BMiX?=
pQB7mM+iBBQMMi3niTi`nT `VK
&
  BMi3nii#R(Ry)c
  BMi3nii#k(Ryy)c

  BMi3niTiR4i#R(y)c
  BMi3niTik4i#k(y)c

  Ti`nT ` kXYcf  BiBbMQiFMQrm?2i?2Ti`nT ` kQBMiQ M `` vXXXf
  TiRYYcf  TiRTQBMiQi?2M2ti2H2K2Q7i#Rf
  Ti`nT ` k TiRYTikcf  BHH2;K2KQ`v+2bbf

  B7UTik=4 R8fV MQi`2H2p Mfi
  &
    f XXXf
  '

  mBMi3Mi#n2H4RTik@TiRcf i?2irQTQBMi2`b2MQiQMi?2b K2`` v M/
      i?2ivT2BbMQi/Ti2/f
 XXX
'
```

(PPE FYBNQMF

```
OBM+Hmlb2//27X?=
OBM+Hmlb2/BMiX?=
pQB7mM+iBBQMMi3niTi`nT `VK
&
  BMi3nii#R(Ry)c
  BMi3nii#k(Ryy)c

  BMi3niTiR4 i#R(y)c
  BMi3niTik4 i#k(y)c
```

```
TiRYYcf  TiRTQBMiQi?2M2ti2H2K2Q7i #R f
Tik4 TikY jcf  TikTQBMiQi #kj) f
TiR4 TiRY 3cf  TiRTQBMiQi?2H bi2H2K2MQ7i #R f

B7UTiR=4i #RVf  b K2`` vQF f
&
  f  XXXf
'

Ti`/B77mM#n2H4RTik@i #kcf  #Qi?TQBMi2``b2 QMi?2b K2`` v M/
    /2/B+ i2i/vT2mb2Ui F2M7`QkBi//2X?V f
XXX
'
```

3FGFSFODFT

[Misra2012] Rule 18.1 A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand.
[Misra2012] Rule 18.2 Substraction between pointer shall only be applied to pointers that address elements of the same array.
[Misra2012] Rule 18.3 The relational operators shall not be applied to objects of pointer type exception where they point into a same object.
[Misra2012] Rule 18.4 The +, -, += and -= operators shall not be applied to an expression of pointer type.
[Cert] Rule ARR36-C Do not substract or compare two pointers that do not refer to the same array.
[Cert] Rule ARR37-C Do not add or subtract an integer to a pointer to a non-array object.
[Cert] Rule ARR39-C Do not add or substract a scaled integer to a pointer.
[Cert] Rec. EXP08-C Ensure pointer arithmetic is used correctly.
[IsoSecu] Subtracting or comparing two pointers that do not refer to the same array [ptrobj].
[IsoSecu] Forming or using out-of-bounds pointers or array subscripts [invptr].
[Cwe] CWE-469 Use of pointer substraction to determine size.
[Cwe] CWE-468 Incorrect pointer scaling.
[Cwe] CWE-466 Return of pointer value outside of expected range.
[Cwe] CWE-587 Assignment of a fixed address to a pointer.
[AnsiC99] Sections 6.2.5, 6.3.2.3, 6.5.2.1, 6.5.6.
[AnsiC90] Sections 6.1.2.5, 6.2.2.3, 6.3.6, 6.3.8.

9
Structures and unions

%FDMBSBUJPO PG TUS

In order to model an entity within a program, it is often necessary to want to associate several scalar data items (integers, characters, ‰). The definition of independent variables to represent this entity makes it difficult to understand the code and tedious to pass parameters to a function. A structure must be used to group together data representing the same entity. And as many structures must be defined as there are entities to model. You should not use a single structure and group data relating to different entities in it.

36-&

36-& ± " TUSVDUVSF NVTU CF VTFE U FOUJUZ

Linked data must be grouped within a structure.

*OGPSNBUJPO

This rule is not linked to an immediate security risk, but is a common-sense rule to be applied to all developments.

#BE FYBNQMF

In the following example, the lack of structure results in function prototypes that are difficult to understand.

```
pQB7/2+i M;HUZ HQ ty- 7HQ vy- 7HQ tR- 7HQ vR 7HQ tk-
7HQ vk- 7HQ tj- 7HQ vjVc
pQB7Tv` KBU72HQ i+QQ`Vbc f  +QQ`/Bb M `` v f
```

(PPE FYBNQMF

The following example correctly uses independent structures to represent different geometric shapes.

```
ivT2/2Tbi`m+TQBMi&b
    7HQ tc
    7HQ vc
' SQBMicni

ivT2/2Tbi`m+i2+i M;H2&ab
    TQBMitvjc
    TQBMitvRc
    TQBMitvkc
    TQBMitvjc
' `2+i M;H2ni
```

```
i v T 2 / 2 7 B i ` m + T v ` K B / n &b
    ` 2 + i M ; H 2 # n i b 2
    T Q B M i n i i Q T c
' T v ` K B / c n i

p Q B / 2 + i M ; H U 2 2 + i M ; H 2 n i 2 + V c

p Q B 7 T v ` K B T 7 T v ` K B / n T v ` V c
```

4 J [F P G B T U S V D U V S F

The size of a structure should not be assumed to be equal to the sum of the size of its elements. This is due to structure padding. It corresponds to a rearrangement of the fields in memory in order to properly align the structure (it is referred to as padding fields). For this reason, the size of a structure should not be calculated by adding up the size of its fields, as this does not take into account the size of the padding fields.

36-&

3 6 - & ± % P Q P U D B M D V M B U F U I F T J [F P J U T ¾ F M E T

Because of the padding, the size of a structure should not be assumed to be the sum of the size of its fields.

B E F Y B N Q M F

```
O / 2 7 B M a 2 A w 1 n h " G R y y
X X X
i v T 2 / 2 7 B i ` m + & i
  B M i i # H ( a A w 1 n h ) c " G
  b B x 2 n b i B x c 2
' K v n b i ` m c + i
X X X
b B x 2 n b i B x 2 b i ` m 4 + b i B x 2 Q K 7 i v n b i ` m X i + i # H V Y b B x 2 Q K 7 i v n b i ` m X b B x V 2 c
f b b m K 2 7 b i i ? 2 b B x 2 Q 7 i ? 2 b i ` m + i m B 2 b i ? 2 b m K Q 7 i ? 2 b B x 2 Q 7 i ? 2 2 H 2 K 2 M f b
X X X
```

(P P E F Y B N Q M F

```
O / 2 7 B M a 2 A w 1 n h " G R y y
X X X
i v T 2 / 2 7 B i ` m + &
  B M i i # H ( a A w 1 n h ) c " G
  b B x 2 n b i B x c 2
' K v n b i ` m c + i
X X X
b B x 2 n b i B x 2 b i ` m 4 + b i B x 2 Q K 7 i v n b i ` m V + c f ; Q Q / b B x 2 f
X X X
```

8 B S O J O H

The use of non-standard attributes such as T + F i 2 / not considered in this guide.

3FGFSFODFT

[Cert] EXP42-C Do not compare padding data.
[Cert] EXP03-C Do not assume the size of a structure is the sum of the sizes of its members.
[Cert] Rule DCL39-C Avoid information leakage when passing a structure across a trust boundary.

CJU ¾FME

In C, you can specify the size (in bits) of elements of a structure or union, in particular to use memory more efficiently. Precautions must be taken when using † • ˜ Â ± % on the one hand, an B Mtype bit-field will not necessarily be signed. In fact, an B Mvariable is indeed signed by default except in the case of bit-field, where the sign becomes dependent on the compiler implementation.

36-&

36-& ± "MM CJU ¾FMET NVTU CF FYQ

Furthermore, the internal representation of structures with bit-fields is also implementation-dependent, so that no assumption should ever be made about this representation.

36-&

36-& ± %P OPU NBLF BTTVNQUJPOT TUSVDUVSFT XJUI CJU ¾FMET

#BE FYBNQMF

```
ivT2/27i`m+bi`m+im&2
  BMiQF Rcf  #B@7B2HQ7bBx2R f
  BMip Hm2dcf  #B@7B2H7Q`r?B+??2bB;M2T2M/QMi?2+QKTBHm2b`2/f
' bi`m+in#Bi7B2H/
XX
bi`m+in#Bi7B2H/
BMiTinb
Tin4UBMiV bc
bXQ4Rc
XX
B7bXQ44RV B7+QKTBH2B/?;++7Q`2t KTH2#v/27 mHi?2#B@7B2H/b`2
bB;M2/i?2`27Q#22BMQ7bBx2R-bXQF2[m Hyb Q`@R5 f
&
  TinbYcf \ f
   Tin4RyybC \ f
'
```

(PPE FYBNQMF

```
ivT2/27i`m+bi`m+im&2
  mMbB;MB2MiQF Rcf  mMbB;M#B/@7B2H/f
  mMbB;MB2Mip Hm2dcf  mMbB;M#B/@7B2H/f
' bi`m+in#Bi7B2H/
XX
```

```
b i ` m + i n # B i 7 B b 2 H /
bXQ 4 R c
X X
B T bXQ 4 4 R V   M Q H Q M ; 2 ` Q K T B H 2 T 2 M / 2 M i
&
   bXp  H m 4 2 R y y c
'
```

3FGFSFODFT

[Cert] Rule EXP11-C Do not make assumptions regarding the layout of structures with bit-fields.
[Cert] Rec. EXP12-C Do not make assumptions about the type of a plain int bit-field when used in an expression.
[Misra2012] Dir. 1.1. Any implementation-defined behaviour on which the output of the program depends shall be documented and understood.

6TF PG '".T

FAMs [10] were introduced with C99. This corresponds to declaring as the last member of a structure an array without dimensions and therefore of flexible size by nature. If the associated structure is not allocated (or copied) dynamically but on the stack, no space is allocated for this array and accessing it causes undefined behaviour.

Furthermore, this means accepting arrays of undefined size, which contradicts the rule in section 8.3. FAMs are therefore prohibited.

36-&

36-& ± %P OPU VTF '".T

3FGFSFODFT

[Misra2012] Rule 18.7 Flexible array member shall not be used.
[Cert] MEM33-C Allocate and copy structures containing a flexible array member dynamically.
[Cert] Rule DCL38-C Use the correct syntax when declaring a flexible array member.

%P OPU VTF VOJPOT

The C language, via the union mechanism, allows the same memory space to be used to store different data types. However, there is a risk of misinterpretation and misuse of the data. The use of the same space for several data types should therefore be avoided as far as possible.

10. Flexible Array Member

3 & $ 0

3 & $ 0 . . & / % " 5 * 0 / ± % P O P U V T F V O J P O

The use of the same memory space for different data types is not authorised.

The use of unions must be strictly limited to cases where the type is verified by other means and only if it is necessary (for network frame parsing for example), and this must be justified with a comment in the code.

3 F G F S F O D F T

[Misra2012] Rule 19.2 The union keyword should not be used.

10
Expressions

*OUFHFS FYQSFTTJPO

Several basic precautions should be taken whenever expressions handle integers.

For signed integer operations, it must be ensured that there is no overflow of the size of the associated type and for unsigned integer operations, that there will be no value wrap.

36-& ± 3FNPWF BMM QPTTJCMF WBMV

3&$0..&/%"5*0/ ± %FUFDU BMM QPTTJ HFST

#BE FYBNQMF

In the following function, no overflow is checked.

```
OB M + H m1 b2 / B M i X ? =
p Q B 7 Um B M i 3 B-i B M i 3 nD V
&
  m B M i 3 B i # B 4 BY  k c
  B M i 3 nD n # B 4 D Y j c
 f   X X X f
'
```

(PPE FYBNQMF

In the following function, overflows are checked.

```
OB M + H m1 b2 / B M i X ? =

p Q B 7 Um B M i 3 B-i B M i 3 nD V
&
  m B M i 3 B i # Bcb
  B M i 3 nD n # 6 b
  B 7 UB = UA L h 3 n J @s  k V V
  &
    f   2 ` ` Q ` f
  '
  2 H b 2
  &
    B # B4BY k c
```

```
'
B 7 UD = UA L h 3 n J @s jV V
&
  f   2 ` ` Q ` f
'
2 H b 2
&
  D # B4DY j c
'
f   X X X f
'
```

Similarly, all potential errors due to division by zero must be avoided.

3 6 - & ± % F U F D U B O E S F N P W F B O Z Q P U

This check must be systematic for any division or computation of remainder of a division.

B E F Y B N Q M F

In the following function, no check on a possible division by zero is performed.

```
OB M + H m1 b2 / B M i X ? =
p Q B 7 m MUB- M i 3 nB-  B M i 3 nDV
&
  B M i 3 n`i2 b m tHi
  ` 2 b m H4i B f  Dc
  X X X
'
```

(P P E F Y B N Q M F

In the following function, there is a check on a possible division by zero.

```
OB M + H m1 b2 / B M i X ? =
p Q B 7 m MUB- M i 3 nB-  B M i 3 nDV
&
  B M i 3 n`i2 b m tHi
  B 7 U y  4 4DV
  &
    f   2 ` ` Q ` f
  '
  2 H b 2
  &
  ` 2 b m H4i B f  Dc
  '
  X X X
'
```

3 F G F S F O D F T

[Cert] Rule INT30-C Ensure that unsigned integer operation do not wrap.
[Cert] Rule INT31-C Ensure that integer conversions do not result in lost or misinterpreted data.
[Cert] Rule INT32-C Ensure that operations on signed integers do not result in overflow.
[Cert] Rule INT33-C Ensure that division and remainder operations do not result in divide-by-zero errors.
[Cert] Rec. INT08-C Verify that all integer values are in range.

[Cert] Rec. INT10-C Do not assume a positive remainder when using %operator.
[Cert] Rec. INT18-C Evaluate integer expressions in a larger size before comparing or assigning to that size.
[Cert] Rec. INT16-C Do not make assumptions about representation of signed integers.
[Cwe] CWE-190 Integer overflow or wraparound.
[Cwe] CWE-682 Incorrect calculation.
[Cwe] CWE-369 Divide by Zero.
[IsoSecu] Integer division errors [diverr].

3FBEBCJMJUZ PG BSJU

Understanding an arithmetic calculation can be complex if no effort has been made to make it legible. Moreover, depending on the writing method chosen for the calculation, it may prove ambiguous.

A complex expression will need to be simplified to aid understanding. If the complexity is relevant (optimisation, ‰), a comment should explain and accompany the expression. A fairly common example is to use a n-bit left shift for a multiplication by 2^n (or a right shift for a division). Thus, the following expression:

```
II #c
```

can be used to perform the operation a 2^b. Such expressions do not help with understanding the code. In addition, these shifts must comply with precise rules taking into account the number of bit shifts requested and the size of the type concerned (Section 10.7). It is recommended to use bit shifts only when the purpose is to handle the bits of a register, for example.

3&$0

3&$0..&/%"5*0/"SJUINFUJD PQFSB UIBU BTTJTUT XJUI SFBEBCJMJUZ

Arithmetic operations that are as explicit (natural) as possible and follow the logic of the program must be used.

#BE FYBNQMF

In the following example, the arithmetic operations are not readable. Understanding of the operations is not immediate.

```
f  AMi?27QHHQrBMH;+mH i-BrQMi MiiQ + H+mH i-2Y 9 + Y #- f
mBMie9`2ibc
mBMijkni #- +c
`2b4      Y UU  +V II kV Y   #c f   M2tTH M iBrQMmH#2r2H+QKf2
f   II# Bb2[m HBp 2QMi  k # #mi?2`2i?2#Bib?B7BBbmb27Q`
    KmHiBTHB+ ifBQM
XXX
f  K bF+QKTmi iBfQM
mBMijk#iBi7B24i/yt* 61" "c1
mBMijkMi #BiK bcF
M4 9c
#BiK b4F Rc
7QUM4 Mc M= ycM@@V &
  #BiK b4F k  #BiK bcF
' f  QMi?2+QMi` `?v2`2i?2#Bib?B7iQmH?/ p2#22MKQ`2HQ;B+ fH
```

```
#Bi7B24l/#Bi7B2H/U#BiK bF
```

(PPE FYBNQMF

The following code performs calculations using simple arithmetic operations.

```
f  +QKTmi iBQ7M Y9+Y#- f
mBMie9`r2ibc
mBMijknd
mBMijk#nd
mBMijk#nd
`2b4 U  V Y U9  U +VV Y#U #Vcf  THm+bH B`f
XXX
f  K bF+QKTmi iBfQM
mBMijk#iBi7B24l/yt* 61" "c1
mBMijkMi4 9c
mBMijk#iBiK bF
#BiK b4F k llMc f  #Bib? M/HBM;
#Bi7B24l/#Bi7B2H/U#BiK bF
```

3FGFSFODFT

[Cert] Rec. INT14-C Avoid performing bitwise and arithmetic operations on the same data.

6TF PG QBSFOUIFTFT UPQFSBUPST

The C language has many operators, with different levels of priority in terms of their associativity. However, the absence of parentheses in an expression makes it difficult to understand and proofread.

The systematic use of parentheses in the calculations makes it possible to clearly show and choose the priority of the operations and the order in which the calculation is performed.

*OGPSNBUJPO

The C language operators and their priorities are presented in appendix D.

36-&

36-& ± &YQMBOBUJPO PG UIF PSEFS VTF PG QBSFOUIFTFT

To avoid any ambiguity in an expression, its subexpressions must be surrounded by parentheses to make the order of evaluation of a calculation more explicit.

3FGFSFODFT

[Cert] EXP10-C Rec. Do not depend on the order of evaluation of subexpressions or the order in which side effects take place.
[Cert] Rule EXP30-C Do not depend on the order of evaluation for side effects.
[Misra2012] Adv. 12.1 The precedence of operators within expressions should be made explicit.
[Misra2012] Rule 13.2 The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders.
[Misra2012] Rule 13.5 The right hand operand of a logical or % %operator shall not contain persistant side effects.
[Cwe] CWE-783 Operator Precedence Logic Error.

/P NVMUJQMF DPNQBS. QBSFOUIFTFT

It is common to want to check the value of a variable against a lower and an upper limit, and the shortcut of doing this in a single statement without parentheses is an error. Let us take the following expression as an example: U y I 4 t I 4 MV The left side, • À %oI4, is evaluated first. The result of this evaluation (0 or 1) is then compared with the value to the right, which will always be checked for any value of n greater than or equal to 1. The statement U y I 4 t I 4 isMV therefore semantically equivalent to U U y I 4 t V and not to U U y I 4 t V U t. I 4 M V V

Another classic error is the combined B 7 U 4 4 # equality test whose objective is, a priori, to verify that the three variables are equal. In practice, as in the previous case, this test does not behave as the developer expects. Indeed, this conditional will only be true if the three variables are 1 or if + is 0 and and # are different.

#PPMFBO FYQSFTTJPO

The C language does not have a true boolean type in C90. The boolean type was introduced with C99. It has an associated library (b i / # Q Q. However, we will use the term boolean expression for expressions in the C language, even before C99, where the result of the evaluation corresponds to a truth value, as is typically the case for comparison expressions. A boolean expression corresponds to the false truth value for an evaluation returning the value 0. Any other value returned by a boolean expression (whether it is 1 or a negative, positive, integer or non-integer value) corresponds to the true truth value.

Boolean expressions containing at least two relational operators are prohibited without parentheses and must be broken down either into nested conditionals or into several relational expressions.

3 & $ 0

3 & $ 0 . . & / % " 5 * 0 / ± " W P J E F Y Q S F T T J P O J U Z

36-&

36-&±"MXBZT VTF QBSFOUIFTFT JO FRVBMJUZ

Boolean expressions of comparison or equality containing at least two relational operators are prohibited without parentheses.

#BE FYBNQMF

```
O/27BML2 Ryy
XXX
B7Uy I4t I4 LV &
  f  bi i2K2MRi Gq ua2t2+mi2f
' 2Hb2&
  f  bi i2K2MkiL1o12t2+mi2f
'
XXX
B7Ujy It I 9yV &
  T`BMUJ7T#jVcf  Gq ua2t2+mi2f
'
XXX
```

(PPE FYBNQMF

```
O/27BML2 Ryy
XXX
B7UUy It4V  tU I4 LVV &  + b2R, #`2 F/QrBMMiQ`2H iBQM2tHT`2bbBQfMb
  f  bi i2K2MRi f
' 2Hb2&
  f  bi i2K2Mki f
'
 XXX
B7Ujy ItV &f  + b2k, #`2 F/QrBMMiQ M2bi2+QM/BiBQbM iH2K2Mib
  B7UtI9yV &
    T`BMUJ7T#jVc
 XXX
''
```

3FGFSFODFT

[Misra2012] Rule 10.1 Req. Operands shall not be of an inappropriate essential type.
[Misra2012] Rule 12.1 The precedence of operators within expressions should be made explicit.
[Cert] EXP00-C Rec. Use parentheses for precedence of operation.
[Cert] EXP13-C Rec. Treat relational and equality operators as if they were nonassociative.
[Cwe] CWE-783 Operator Precedence Logic Error.

1BSFOUIFTFT BSPVOE

When writing boolean expressions, the absence of parentheses and the exclusive use of associativity makes it difficult to understand the code. The systematic use of parentheses avoids programming errors by making the order of evaluation of operations explicit.

36-&

36-& ± 1BSFOUIFTFT BSPVOE UIF FMF

The different elements of a boolean expression must always be placed in parentheses, so that there is no ambiguity in the order of evaluation.

#BE FYBNQMF

In the following example, it is necessary to know the associativity between the operators and the priority between them in order to understand the order of evaluation.

```
B 7Ut = y    v   x = H 2 M ; V? &
    X X X
'
m 4 x = R y y     R y y t 4 % %  Y v I xc
```

(PPE FYBNQMF

In the following code, the use of parentheses makes it possible to explicitly know the order of evaluation.

```
B 7 U tU = yV     UU xV =H 2 M ; V?V &
  X X X
'
m 4 Ut = R y y V     U U R ytV 4 % tUUvV I xV V c
```

3FGFSFODFT

[Cert] EXP00-C Rec. Use parentheses for precedence of operation.
[Misra2012] Rule 12.1 The precedence of operators within expressions should be made explicit.
[Cwe] CWE-783 Operator Precedence Logic Error.

*NQMJDJU DPNQBSJTPO

In C90, the “true” value corresponds to any value other than 0 (whether that value is negative, positive, integer or non-integer) and the “false” value corresponds to the value 0. As a result, it is possible to write boolean expressions where a comparison with 0 is implicitly made. Implicit comparisons make understanding and maintenance of the code difficult. Boolean expressions must use an explicit comparison operator:

==, !=, <, >, <=, >=

36-&

36-& ± *NQMJDJU DPNQBSJTPO XJUI

All boolean expressions must use comparison operators. No implicit test with a value equal to 0 or different from 0 must be performed.

3&$0../%"5*0/±6TJOHUIFCPPMUZ

In C99, the #QQoHn"QQype must be used for variables with boolean values.

In C99, the use of the #Q-Qped variables directly in a conditional is accepted.

#BEFYBNQMF

The comparisons implicit in the following code should be removed in favour of explicit comparisons.

```
O/27BMJ2sRy
mBMix3c
XX
r?BH2tV &
  XXX
'
B7Ut IvV &
  XXX
' 2Hb2&
  XXX
'
B7U5xV & f  BKTHB+BQiKT `BbQiMy  M/xb?QmH2Q7i?2#QQiHvT2f
  XXX
'
B7UTi`V &
  XXX
'
7Q`Ut 4 Jsctct@@V &
  XXX
'
```

(PPEFYBNQMF

In the following example, no implicit comparison is performed and the dedicated header file is used.

```
OBM+Hmb2/#QQHX?=

#QQxtcf  mb2Q7i?2#QQiHvT2f
r?BH2t = yV &
  XXX
'
B7Ut IvV &
  XXX
' 2Hb2&
  XXX
'
B7U6 Ga44xV &f  +QMbi MQMi?2H27iM/2tTHB+BQKT `BbQQ7Mc x #2BMQ7
    i?2#QQiHvT2 B7U5xV + M#2mb2/f
  XXX
'
B7ULIGG54Ti`V &
  XXX
'
7Q`Ut 4 Jsct = yct@@V &
  XXX
'
```

3FGFSFODFT

[Cert] Rec. EXP20-C Perform explicit tests to determine success, true and false, and equality.

#JUXJTF PQFSBUPST

Generally speaking, bitwise operators should only be used with expressions of unsigned types. This way, many undefined or implementation-defined behaviours are avoided.

3&$0..&/%"5*0/ ± #JUXJTF PQFSBUPS PQFSBOET PO MZ

Furthermore, some bitwise operators such as , %or can easily be mistakenly used — especially the first two — instead of logical operators such as , % 8nd 5 To avoid this confusion, it is important to check that the bitwise operators used in boolean expressions are actually the desired operators. Bitwise operators have no reason to be applied to any operand of type boolean or similar.

36-& ± /P CJUXJTF PQFSBUPS PO BO P

#BE FYBNQMF

```
B7UlU `=4 yV pU`I RkyVVf&QT2` iQ+`QM7mbBQM
 XXX
B7UlU H 6G V 54 yV QT2` iQ+`QM7mbBQM
 XXX
'
```

(PPE FYBNQMF

```
B7UlU `=4 yV pU`I RkyVVf&+Q``2+iBQM
 XXX
B7UlU H 6G V 54 yV +Q``2+mb2?2`2Q7i?2#BirBbQT2` iQ`
                    BM #QQH22MT`2bbBQM
 XXX
'
```

3FGFSFODFT

[Cert] Rec. INT13-C Use Bitwise operators only on unsigned operands.
[Cert] Rec. EXP14-C Beware of integer promotion when performing bitwise operations on integer types smaller than int.
[Cert] Rule INT34-C Do not shift an expression by a negative number of bits or by greater or equal the number of bits that exist in the operand.

[Cwe] CWE-682 Incorrect calculation.
[Cert] EXP46-C Do not use a bitwise operator with a Boolean-like operand.
[Cwe] CWE-480 Use of incorrect operator.
[Misra2012] Rule 10.1 Operands shall not be of an inappropriate essential type.

#PPMFBO BTTJHONFO

The C language assignment operator returns a value. It is therefore possible to use this value. However, this is often an unintentional assignment of the developer resulting from confusion between the assignment operator 4and the equality operator 4 4.

(00% 13"$5

(00% 13"$5*$& ± %P OPU VTF UIF WBM

*OGPSNBUJPO

During the compilation phase with the right options (@ q ,HHJ), a warning will be emitted suggesting in particular to place the assignment in parentheses in the boolean expression (@ q T `2 M i 2 2 2).

36-&

36-& ± "TTJHONFOU QSPIJCJUFE JO

An assignment must not be made in any boolean expression. An assignment must be made in an independent statement.

In order to limit the risks of writing an assignment with the 4operator instead of a comparison with the 4 4operator, when the comparison is made between a variable and a constant operand, the constant operand should be written as the left operand of the 4 4operator and the variable as the right operand. The compiler raises a warning when trying to assign a value to a constant operand.

(00% 13"$5

(00% 13"$5*$& ± $PNQBSJTPO XJUI D

When a comparison involves a constant operand, this should preferably be set as the left operand to avoid an unintended assignment.

*OGPSNBUJPO

This good practice is debatable, and is therefore not enforced but merely advised. Compiling this type of code with strict options (in particular with @ q), as required by section 5.2, is actually sufficient to detect the use of the assignment operator instead of the comparison operator in a boolean expression.

#BE FYBNQMF

The following code contains assignments in Boolean expressions. One of the assignments in a conditional expression is a programming error.

```
B7UtU 4 v Y xV = yV f&  bbB;MK2BMi "QQH22MT`2bbBQMmb2Q7i?2p Hm2
    `2im`M2#vi?2 bbB;MK2Mi+QMbi QMi?2`B;?if
    XXX
'
B7Ux 4 o GlV &f +QMbi QMi?2`B;?iM/4BMbi2Q744 f
    XXX
'
```

(PPE FYBNQMF

In the following example, all assignments are made in independent statements and the various problems are corrected.

```
t 4 v Y xc
B7Uy ItV &
  XXX
'
B7Uo Gl144xV &
  XXX
'
```

3FGFSFODFT

[Misra2012] Rule 13.4 The result of an assignment operator should not be used.
[Cert] Rule EXP45-C Do not perform assignments in selection statements.
[IsoSecu] No assignment in conditional expressions [boolasign].
[Cwe] CWE-480 Use of incorrect operator.
[Cwe] CWE-481 Assigning instead of comparing.
[Cwe] CWE-482 Comparing instead of assigning.

.VMUJQMF BTTJHONFO

The C language allows the same value to be assigned to several variables with a single statement. This multiple assignment is often used for variable initialisations.

However, code containing multiple assignments is difficult to read and also difficult to maintain. Break the multiple variable assignment statement down into as many assignment statements as there are variables.

36-& ± .VMUJQMF BTTJHONFOU PG WB

Multiple assignment of variables is not authorised.

#BE FYBNQMF

The following example shows a multiple assignment.

```
XXX
  4 # 4 + 4 / 4 Rcf  KmHiBTHb2B;MK2Mi
```

(PPE FYBNQMF

The following code contains one assignment per variable.

```
XXX
  4 Rc
# 4 Rc
+ 4 Rc
/ 4 Rc
```

0OMZ POF TUBUFNFO

The end of a statement in the C language is marked by the semicolon. The C language does not require the developer to write only one statement per line of code.

However, when several statements are present on the same line, the code is less readable. Debugging is also more difficult, since it is not possible to check the execution of code one statement at a time.

The presence of multiple statements per line of code also distorts the metric of the number of lines of code.

36-& ± 0OMZ POF TUBUFNFOU QFS MJ

#BE FYBNQMF

In the following example, the code is difficult to understand.

```
BMijknic BMie9#c
  4 9c# 4  f ecT`BMUJ7 4 W- # 4 WH HSM- - #Vc
```

(PPE FYBNQMF

```
BMijknic
BMie9#c
  4 9c
# 4  f ec
T`BMUJ7 4 W- # 4 WH HSM- - #Vc
```

6TF PG ¿PBUJOH QPJ

*OGPSNBUJPO

By floating-point numbers, we are referring to numbers using a floating-point representation.

The representation of floating-point numbers in a machine is a complex notion which is often not well known or badly understood and moreover, it is dependent on the precision associated with the type. These floating-point numbers are often a source of errors.

Not all actual values can be represented as floating-point numbers, and other “unnatural” phenomena such as … † — “ –[11] ãnd ’‡ … ’ ‡ ‰ •[12] can occur with the use of floating-point numbers, although these points will not be detailed in this guide. Further details are given in the standard IEEE754 [float], which ensures reproducible inter-compiler and inter-architecture behaviour in the presence of floating-point numbers.

In addition, the error associated with the use of these floating-point numbers can become greater than the result of the calculation using them.

Finally, certain elementary properties of real arithmetic are no longer true when using floats: commutativity, associativity, ‰ ˜ ‡ À

For all of these reasons, the use of floating-point numbers is strongly discouraged. Should the use of floating-point numbers prove necessary for numerical processing for example, the developer will have to ensure that the constant float values are representative and that they are correctly used in accordance with the associated precision.

(00% 13"$5*$& ±"WPJE ¿PBUJOH DPO

Do not use floating numerical constants in order to avoid loss of precision and other phenomena related to floating-point numbers. If this cannot be avoided, the representativeness of the float value in question must be checked.

3&$0..&/%"5*0/ ±-JNJU UIF VTF PG ¿P TUSJDUMZ OFDFTTBSZ

The use of floating-point numbers should be limited.

Float type loop counters are sources of error due to the limited representativeness of this type and the associated complexity.

11. Phenomenon related to the precision of floats, such as LargeFloatValue + FloatingEpsilon = LargeFloatValue, • ÀaLarge float value will “absorb” a small float value

12. Another phenomenon always related to the precision and representation of floating-points numbers, such that for two close float values, FloatValue1 FloatValue2 = 0, whereas formally, FloatValue1 6 FloatValue2

3 6 - &

3 6 - & ± / P ¿ P B U U Z Q F M P P Q D P V O U F S

Loop counters must be of the integer type only, with type overflow verification of the counter values.

The handling of float values in Boolean expressions is also always very risky in connection with problems of representation and precision of these values. The use of the logic operators 5 4or 4 4on floats is incorrect in most cases. The results may depend on the level of optimisation, the compiler itself and the platform used.

3 6 - &

3 6 - & ± % P O P U V T F ¿ P B U J O H Q P J O U J O F R V B M J U Z

Qi

B E F Y B N Q M F

```
7 H Q  v  4  y X Rfc  M Q i ` 2 T ` 2 b 2 M i  #BHM2 B K T H T 2 2 + B b B Q M
 X X X
B 7 Uv  4 4  y X Rf V  + Q K T  ` B bQQ7 MH Q  jp  H m 2L . y X R/ Q m # Hp 2 H mb 2 Q T ` Q K Q i BQQ7M
      f
   T ` B M U J 7 2 [ m  $HM] V c
2 H b 2
   B 7 Uv  4 4  y X7R f  y X R 7 H Q  jp  H m @ + Q M / B i B+Q?M + F 2?/2 ` 2 f
      T ` B M U J 7 2 [ m  H$M] V c
   2 H b 2  T ` B M U J 7 M Q i 2 [ m  $HM] V c
X X X
7 Q ` U 7 H Q  ti  4  y X7R t  I 4  R X7y t  Y 4  y X7R  &f  i ? 2 H Q Q rTB H H 2 +  ` ` B 2 /
```

[Cwe] CWE-682 Incorrect calculation.

$PNQMFY OVNCFST

Since C99, the C language supports the calculation of complex numbers with three new built-in types, /Qm#H2n*QKTH2t/Qm#H2 and 7HQi+QKTH2t. With the associated header file +QKTH2t, these types are also accessible via /Qm#H2+QKTH2t/Qm#H2 and 7HQi+QKTH2t. In addition, the three associated imaginary types are also supported: /Qm#H2nAK,;BMHQM; /Qm#H2nAK and 7BHQivAK ;BMHQM`;v /Qm#H2 BK,;BMHQM`;v /Qm#H2 BK ;BMHQM`; and 7HQi BK ;BMHQM`;.

As these complex numbers are based on a floating representation, their use is strongly discouraged.

3&$0

3&$0../%"5*0/±/PVTFPGDPNQMFY

Complex numbers introduced since C99 should not be used.

11
Conditional and iterative structures

6 T F P G C S B D F T G P S D P

(PPE FYBNQMF

The following example delimits all the conditionals and loops with braces.

```
B 7 U y  4 4 V  &
  T ` B M U ] s  4  y $ M ] V c
'  2 H b 2 &
  B 7 U t  I  y V  &
    X X X
    r ? B H 2 t  I  y V  &
      t Y Y c
      X X X
    '
  '  2 H b 2 &
    r ? B H 2 t  =  y V  &
      t @ @ c
      X X X
    '
  '
'
f  2 t  K T H Q 2 7 ; Q i Q 7  B H @  T T H 2 f
B 7 U 2 ` `  4  a a G >  b ? a X m T R / i U 2 ?  b ? * i t  b B ; M 2 / S  ` V k b 5 4  y V  &
  ; Q i Q 7  B t H
'
f  Q i ? 2 ` + ? 2 + F b
7  B , H
  f  + H 2  M B M ; / ` 2 H 2  b B Q 7 # m 7 7 2 ` b
  ` 2 i m ` 2 ` ` c
```

3FGFSFODFT

[Misra2012] Rule 15.6 The body of an iteration-statement or a selection statement shall be a compound-statement.
[Cert] Rec. EXP19-C Use braces for the body of an if, for, or while statement.

$PSSFDUDPOTUSVDU

The b r B i + ? 2 statement of the C language provides an elegant way of writing the handling of different cases based on the value of a variable or an expression. However, this statement is also a source of errors when omitting the break statement: unwanted code may be executed. In fact, since the successive conditions of the b r B i + ? 2 statement are not exclusive, several cases can be activated. It must also be ensured that processing is performed if the value of the expression does not correspond to any of the cases of the b r B i + ? 2 (default case).

36-&

36-&±4ZTUFNBUJDEF5¾BOJ-J7JPOPGB

A b r B i + ? @ must always contain a default case placed last.

3&$0..&/%"5*0/±4&5FFBDb DBBiTFPUPBGU F NFOUT

3&$0..&/%"5*0/±4&5FFBDb DBBiTFPUPBGU FNFOUT

By default, a b r B i + ? @ must always contain a break for each case. The absence of a break to avoid duplicating code is tolerated but must be made explicit in a comment.

*OGPSNBUJPO

The @qBKTHB+Bi@7 option is used to check the correct application of this recommendation.

The code in each +b2 should be simple and contain few statements. If complex processing is to be carried out in a +b2, then a function must be defined for this processing and this function must be called from the +b2.

3&$0

3&$0..&/%"5*0/ ± /P OFTUJOH PG brBi+?@+b2

Even though C allows it, the nesting of control structures inside a brBi+? should be avoided.

Finally, it is forbidden to declare, initialise variables or introduce code statements within a brBi+? statement before the first label associated with the first case of the brBi+?@+b2.

36-&

36-& ± %P OPU JOTFSU TUBUFNFOT brBi+?@+b2

#BE FYBNQMF

In the following example, the final /27 must be present.

```
brBi+Up `V &
  BMiB4 ycf  MQimi?Q`Bx2/
  +b2o G,R
    XXX
    #`2 eF
  +b2o G,k
    XXX
  +b2o G,j
    XXX
    #`2 eF
  f  #b2M+Q27/27 mHiM/#`2 FM/MQ+QKK2iMQi2tTH BiM2+b2QMi?2p Hm2
     o Gkf
'
```

(PPE FYBNQMF

In the following example, a break statement is present for each +b2. There is also a final default statement to ensure that processing is carried out if the value did not match any of the cases.

```
BMiB4 ycf /BbTH +22/+H `iBQMM/BMBiB HBb iBQM
brBi+Up `V &
  +b2o G,R
     XXX
     #`2 eF
  +b2o G,k
     XXX
     f #`2 FpQHmMi `BBHvbBMf;
  +b2o G,j
     XXX
     #`2 eF
  /27 mHi
```

3FGFSFODFT

[Misra2012] Rule 16.1 All switch statements shall be well-formed.
[Misra2012] Rule 16.2 A switch label shall only be used when the most closely-enclosing compound statement is the body of the switch statement.
[Misra2012] Rule 16.3 An unconditional break statement shall terminate every switch-clause.
[Misra2012] Rule 16.4 Every switch statement shall have a default label.
[Misra2012] Rule 16.6 Every switch statement shall have at least 2 switch-clauses.
[Misra2012] Rule 16.7 A switch expression shall not have a essentially boolean type.
[Cert] Rule DCL41-C Do not declare variables inside a switch statement before the first case label.
[Cert] Rec. MSC01-C Strive for logical completeness.
[Cert] Rec. MSC17-C Finish every set of statements associated with a case label with a break statement.
[Cwe] CWE-484: Omitted Break Statement in Switch.
[IsoSecu] Use of an implied default in a b r B i statement [swtchdflt].

$PSSFD7UQDMFPOPTQUTSVDUJPOPG

The C language allows several expressions to be added to the first and last element of a 7 Qstatement, separated by a comma. This allows for example the initialisation of several variables in the first element of the 7 Q or the incrementation of several variables in the third element of the 7 Q However, the presence of comma-separated expressions in the 7 Qstatement makes it difficult to understand the code and is a source of errors. In addition, the sequencing of statements is already prohibited in section 14.1. If other variables are to be initialised at the start of the loop, they must be initialised just before the 7 Qstatement. If several variables are to be incremented or decremented, they must be modified at the end of the loop.

Furthermore, the C language does not require each element (initialisation, stop condition and increment/decrement) of the 7 Qstatement to be completed. It is possible to leave the initialisation empty, for example, or even to leave each element empty, resulting in an infinite loop. In the case of an infinite loop, the form r ? B H 2 U R V & is to be preferred to the form 7 Q ` U c c V & .XXX '

36-&

36-&±$PSSFD7UQDMFPOPTQUTSVDUJPOPG

Each element of a 7 Qloop must be completed and contain exactly one statement. Thus a 7 Qloop must contain an initialisation of its counter, a stop condition on its counter, and a loop counter increment or decrement.

#BEFYBNQMF

In the following example, the comma is used to separate several initialisations and modifications of variables in the first and third element of the 7 Q In addition, 7 Q ` loops should be replaced by r ? B H 2 U V & or X X Q & XXX ' r ? B H 2 U V

```
O/ 2 7 B MJ2 s n G P P S      R y l
7 Q Ù c c V  &
  /  i   4  ` 2   /U V c
  B 7 U y   4 4  i  V  &
    # ` 2  e
  '
  X X X
'
7 Q `UB  4   y c c V  f&   K B b b B MH 2 K 2 M f b
  K   t 4  b Q K 2 n 7 m MU BVBcQ M
  BY Y c
  B 7 UB  = 4 K   tV  &
    # ` 2  e
  '
'
X X X
7 Q `UB  4   y -   4   y c B  l   J   s n G P cPBBY Y -   Y 4   k V  &
  X X X
  b Q K 2 n 7 m MU  VBcQ M
  X X X
'
```

(P P E F Y B N Q M F

The following code only initialises the loop counter in the first 7 Qelement, and only increments the counter in the third element. The for loops contain all its elements (initialisation, stop condition, counter increment).

```
O/ 2 7 B MJ2 s n G P P S      R y l
7 Q `UB  4   y c B  l    ``   v a B x 2BY Y V  &
  X X X
  /  i   ``   v( B)   4  b Q K 2 n 7 m MU BVBcQ M
  X X X
'
r ? B H Ø R V  &
  /  i   4  ` 2   /U V c
  B 7 U y   4 4  i  V  &
    # ` 2  e
  '
  X X X
'
B  4   y c
/ Q &
  K   t 4  b Q K 2 n 7 m MU BVBcQ M
  BY Y c
'
r ? B H ØB  l   K   tV c
X X X
  4   y c
7 Q `UB  4   y c B  l   J   s n G P cPBBY Y V  &
  X X X
  b Q K 2 n 7 m MU  VBcQ M
  X X X
    Y 4   k c
'
```

3 F G F S F O D F T

[Misra2012] Rule 14.2 A for loop shall be well-formed.
[Misra2012] Rule 15.6 The body of an iteration-statement or a selection statement shall be a compound-statement.
[Cert] Rec. EXP15-C Do not place a semicolon on the same line as an if, for, or while statement.
[Cert] Rec. EXP19-C Use braces for the body of an if, for, or while statement.

$IBOHJOH PG B GPS MP UIF MPPQ

Changing the counter of a 7 Qlòop within the loop makes it difficult to understand and maintain the code. Furthermore, this can lead to an infinite loop when the comparison operator in the loop condition is an equality or inequality. If the loop accesses an array, there is also a risk of overflow.

The loop counter should only be modified in the third part of the 7 Qlòop.

It is common to modify within the body of the loop a flag or other variable that intervenes at the conditional stop expression of the 7 Qlòop. This scenario must then be replaced by the use of a # ` 2 allowing the exit from the loop.

36-&

36-& ± $IBOHF UIF QPJOUFS PG B GPS MPPQ

The counter of a 7 Qlòop must not be changed inside the body of the 7 Qlòop.

#BE FYBNQMF

The example below shows a 7 Qlòop with a modification of its counter in its body. There is a risk of an infinite loop.

```
O/27BMJ2snGPPS Ryl

7Q`UB4 ycB54J snGPcPBSYYV &
  XXX
  B7UXXXV &
    f B7i?2+QM/BiBBQbK2irBi?B44N-r22M/mTBM MBM7BMBiQ2QTf
    BYYc
  '
  XXX
'
```

(PPE FYBNQMF

In the following code, no changes are made to the counter in the body of the 7 Q ` loop.

```
O/27BMJ2snGPPS Ryl

7Q`UB4 ycBI J snGPcPBSYYV &
  XXX
  B7UbQK2n7mMU+VBVQ&M
    #`2 Ff HQQbTiQTT2f
  '
  XXX
'
```

3FGFSFODFT

[Misra2012] Rule 14.2

[Cert] Rule ARR30-C Do not form or use out-of-bounds pointers or array subscripts.

12
Jumps in the code

% P O P U ; Q Y i T Q F C B D L X B S E

The use of a backward ; Q i Q akes proofreading and maintenance of the code very difficult and is a source of errors such as unwanted infinite loops. If this need arises, it is because the algorithm to be implemented actually comprises a loop. Then use the control structures proposed by the language for the loops, in other words 7 Q ` r ? B H r 2 / Q @ r ? B H 2

3 6 - &

3 6 - & ± / P V T F ; P Q G Q C B D L X B S E

Prohibit within a function the use of ; Q i Q atements referring to a label that is placed before this ; Q i Q atement.

B E F Y B N Q M F

The following example contains a backward ; Q i. Q This choice of implementation corresponds to an assembler approach, and does not take advantage of the higher level possibilities offered by the C language.

```
O 2 7 B M " 1 6 6 1 _ n a A w 1  R y y l
p Q B 7 Q U Q V  &
  m B M i 3 h ( i " l 6 6 1 _ n a A c w 1
  m B M i 3 h i 4  y c
  K v n H Q Q T
    b ( t )  4 t c
  t Y Y c
  B 7 U t  l  " l 6 6 1 _ n a A w 1
    &
      ; Q i Q K v n H Q Q T     f  #  + F r  ` / Q i Q T ` Q ? B # B i 2 /
    '
'
```

(P P E F Y B N Q M F

The following example uses a loop type control structure. There is no need for a backward ; Q i. Q

```
O 2 7 B M " 1 6 6 1 _ n a A w 1  R y y l
p Q B 7 Q U Q V  &
  m B M i 3 h ( i " l 6 6 1 _ n a A c w 1
  m B M i 3 h i 4  y c
  7 Q U t  4  y c t  l  " l 6 6 1 _ n a A w 1 Y V  &
    b ( t )  4 t c
  '
'
```

3 F G F S F O D F T

[Misra2012] Rule 15.1 The goto statement should not be used.

- J N J U F E V T F P G G P S X

The Š " – › ..; Q î Q can simplify error management and reduce the number of exit points from a function. However, with a Š " – › ..; Q î Q outside of a conditional statement, code may be executed with variables that have not been initialised. Another possible serious consequence is, for example, forgetting to free memory or resources among other things. The code must therefore be modified in order to use control structures that avoid the use of ; Q i. Q

The Š " – › ..; Q î Q should only be used for error management, and the number of labels should be kept to a minimum.

3 & $ 0

3 & $ 0 . . & / % " 5 * 0 / ± - J N ; J Q U G E V T F P G G P

The use of a forward ; Q i Q is tolerated only in cases where it allows:

- the number of exit points from the function to be significantly limited;
- the code to be made much more readable.

The label(s) referenced by the ; Q i Q statements must all be located at the end of the function.

(P P E F Y B N Q M F

The following code does not use a forward ; Q i, Q but uses the control structures proposed by the C language.

```
O 2 7 B M " 1 6 6 1 _ n G 1 L     U R k 3 l V
B M i j k n K v n 7 m M + U B N Q i Y k n i V  &
  6 A G 1 7  4  L l G G
  m B M i 3 n # m 7 7 2 4 ` L l G G
  B M i j k n ì 2 b m H 4 i 1 _ _ n l L . 1 6 A L 1 .
  7  4  7 Q T 2 J M K v T  i ? -  ] ` ] V c
  B 7 U L l G G 4 4 7 V  &
    ` 2 b m H 4 i 1 _ _ n 6 P G S 1 L
  ' 2 H b 2 &
    # m 7 7 2 4 ` U n B M i 3 n V  V  K  H H U Q H 6 6 1 _ n G 1 b B x 2 Q m 7 B M i 3 V n V c
    B 7 U L l G G 4 4 # m 7 7 V 2 ` &
      ` 2 b m H 4 i 1 _ _ n J  G G P *
    '
    2 H b 2
    &
      X X X
      7 ` 2 Q # m 7 7 V 2 c
      # m 7 7 2 4 ` L l G G
      ` 2 b m H 4 i 1 _ _ n L P 1 c _ P _
    '

    7 + H Q U 7 V 2 c
    7  4  L l G G
  '
  ` 2 i m ` M 2 b m H i
'
```

5PMFSBUFE FYBNQMF

The following example uses the forward ;Qi for error management. This scenario is tolerated.

```
BMijknK vn7mM+UBMQiMknV &
  6AG17 4 LlGcG
  mBMi3n#m7724`LlGcG
  BMijkn`i2bmH4i1__nlL.16AcL1.
  7 4 7QT2JMKv T i?- ]`]Vc
  B7ULlGG44 7V &
    `2bmH4i1__n6PcS1L
      ;QiQ+H2 McmT
  '
  #m7742UmBMi3nVK HHUQ+661_nG1bBx2Qm7BMi3VmVc
  B7ULlGG44#m77V2`&
    `2bmH4i1__nJ GcG P*
    ;QiQ+H2 McmT
  '
  XXX
  `2bmH4i1__nLP1c_P_
+H2 M,mT
    B7ULlGG547V &
        7+HQU7V2c
        7 4 LlGcG
      '
  B7ULlGG54#m77V2`&
    7`2D#m77V2c
    #m7724`LlGcG
  '
  `2im`M2bmeHi
'
```

3FGFSFODFT

[Misra2012] Rule 15.1 15.5.

13
Functions

$PSSFDU BOE DPOTJT

%FDMBSBUJPO 1SPUPUZQF PG GVO

The declaration of a function or its ”–“˜“˜is a statement which defines three elements: the return type of the function, its name and the list of its arguments, followed by a semicolon.

%F¾OJUJPO PG GVODUJPO

The definition of a function is the body of the function, • À the set of statements it executes. A function definition also contains a prototype of the function.

C90 allows for the implicit declaration of functions, whether it be the absence of a return type or the absence of a function declaration. C99 is stricter and imposes at least one type specifier.

The C language, in its successive versions, proposes different forms for the declaration of functions. Combining these different forms of function declaration is not recommended since this risks resulting in a much less precise analysis of the code, and leading to problems when editing links.

*OGPSNBUJPO

Compilers raise a warning (@ q B K T H B + B i @ 7 m M + i B Q r M @ @ q / B 2 K + T H H ` B i + B Q Q M B M or @ q ` 2 i m ` M , @ u t a T 2 me an implicit extern int type, • À by default, a function not associated with a return type has an integer type.

36-& ±"OZ OPO TUBUJD GVODUJPO QSPUPUZQF

36-& ±5IF QSPUPUZQF EFDMBSBUJPO TUT EF¾OJUJPO

The types of parameters used to define and declare a function must be the same.

36-& ± &WFSZ GVODUJPO NVTU IBWF BO FYQMJDJU BTTPDJBUFE XJUI JU

Each function is explicitly defined with a return type. Functions without a return value must be declared with a pQB type parameter. In the same way, a function without a parameter must be defined and declared with pQB as argument.

Activating the compiler warnings is used to find out which functions are not correctly declared (missing return type, inconsistency of types between declaration and definition).

#BE FYBNQMF

In the example below, the return type is missing for a declaration, a function without parameters is not correctly declared and there is an inconsistency between the declaration and the definition.

```
 f   ?2  /2X?  f
7QUQmBMi3nVc f  i?2`2im`MvT2BbKBbbBM;
mBMijk#i`UmBMiRe#Vic
pQB+ `UVc   f  pQBBbKBbbBMMi?2T ` K2i2i`vT2iQBM/B+ i2 i?2
    7mM+iB/QM2bMQii F2T ` K2i2`bf
f  7BHX2  f
7QUQmBMi3nV &
    XXX
'
mBMijk#i`UBMijkn#V &f   7i2`i?2/2+H ` iB-Q#MKmb#2  mBMiRefi
    XXX
'
pQB+ `UV &
    XXX
'
```

(PPE FYBNQMF

The following example makes a correct declaration and definition of functions.

```
f  ?2  /2X?  f
pQB7QUQmBMi3nVc
mBMijk#i`UmBMiRe#Vic
pQB+ `UpQBV/c
f  7BHX2  f
pQB7QUQmBMi3nV &
    XXX
'
mBMijk#i`UmBMiRe#Vi &
    XXX
'
pQB+ `UpQBV/ &
    XXX
'
```

3FGFSFODFT

[Misra2012] Rule 8.1 Types shall be explicitly specified
[Misra2012] Rule 8.2 Function types shall be in prototype form with named parameters
[Misra2012] Rule 8.3 All declarations of an object or function shall use the same names and type qualifiers
[Misra2012] Rule 17.3 A function shall not be declared implicitly

[Misra2012] Rule 17.5 The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements
[Cert] Rec. DCL07-C Include the appropriate type information in function declarators
[Cert] Rec. DCL20-C Explicitly specify void when a function accepts no arguments
[Cert] Rec. DCL31-C Declare identifier before using them
[Cwe] CWE-628 Function call with incorrectly specified arguments
[IsoSecu] Using a tainted value as an argument to an unprototyped function pointer [taintnoproto].
[IsoSecu] Calling functions with incorrect arguments [argcomp].

%PDVNFOUBUJPO PG

Incomplete documentation of a function can lead to programming errors. This includes the functionality of the function, a precise description of the parameters justifying the passages by value or reference, but also the conditions to be checked for the correct use of the function.

1BTTJOH PG QBSBNFUFS CZ WBMVF

When passing a parameter by value (or by copying), the value of the actual argument on the function call is sent (copied) to the respective formal argument of the called function. The direct consequence is that any change made to a formal argument is not propagated to the actual argument.

1BTTJOH PG QBSBNFUFST CZ SFGFS

When passing a parameter by reference, the address of the value of the actual argument on the function call is sent to the respective formal argument of the called function. The direct consequence is that any change made to a formal argument will be propagated to the actual argument.

3&$0

3&$0..&/%"5*0/ ± %PDVNFOUBUJPO

All functions must be documented. This includes:

- a description of the function and the processing carried out;
- the documentation of each parameter, the direction of the parameter (input, output, input and output) and any condition existing on it;
- the possible return values must be described.

This also includes the conditions for proper use of the function to be specified in the prototype, especially in the case of portable code (Linux, Windows).

3&$0

3&$0..&/%"5*0/ ± 4QFDJGZ DBMM DP

7BMJEBUJPO PG JOQVU

Programming errors can lead to invalid values being passed as function parameters. If there is no parameter validity check, the behaviour of the function is undefined.

It is therefore necessary to check:

- address consistency (non-null, alignment, ‰ Ï;‡ À
- parameter values (value ranges, ‰ Ï.‡ À

Some generic measures must be applied, such as the return of an error code for example.

36-&

36-&±5IF WBMJEJUZ PG BMM UIF QBSBNFUFS RVFTUJPOFE

This includes:

- the validity of the addresses for pointer-type parameters must be checked (pointers must be non-null, properly aligned, ‰ Ï;‡ À
- the parameters must be checked to ensure that they belong to their domain.

This applies to the functions defined by the developer (‡ Section 13.2) as well as to the functions of the standard library.

#BE FYBNQMF

In the following example, the validity of the parameters is not checked.

```
/Qm#H2pBbBUQMijknM+ BMijkn/iV &
  f  / 54y MQip2`B7B2f/
  `2im`MUUQm#VMZ f UUQm#VMZc
'
```

(PPE FYBNQMF

Example 1:
The following code shows an example where the validity of the parameters is checked:

```
/Qm#H2pBbBUQMijknM+ BMijkn/iV &
  /Qm#H2b4 yXyc
  B7Uy 44V &
    f  2``Q`? M/HBMf;
  '
  2Hb2&
    `2b4 UUQm#VMZ f UUQm#VMZc
  '
  `2im`M2bc
'
```

Example 2:
The following code shows a second example where the validity of the parameters is checked:

```
mBMi32M+`vUmiBMi3nQmiT+miBMijkniQmiTmin-H2M
                +QMbmiBMi3nBMT+mi+QMbBiMijknBMTmin-H2M
                2M+`vTi2/n++iitV &
    mBMi32i`4 yc
```

```
B7UUlGG44QmiTVmi%%LlUGG44QmiTminH2%%LlUGG442M+`vTi2/nV+Vit&
    2``YYcf  2``Q`?  M/HBMf;
'
B7UUlGG44BMTVmi%%BMTminH2MyV %%BMTminH2MsnALSlhnVG1&
    2``YYcf  2``Q`?  M/HBMf;
'
B7Uy 442`V &
    f   SA+Q/2f
'
`2im`M``c
'
```

3FGFSFODFT

[Misra2012] The validity of values passed to library shall be checked.
[Misra2012] Dir 4.1 Run-time failures shall be minimized
[Cert] API00-C Functions should validate their parameters
[Cert] Rule ARR38-C Guarantee that library functions do not form invalid pointers.
[Cert] Rec. MEM10-C Define and use a pointer validation function.
[Cwe] CWE-20 Insufficient input validation
[Cwe] CWE-628 Function call with incorrectly specified arguments.
[Cwe] CWE-686 Function call with incorrect argument type.
[Cwe] CWE-687 Function call with incorrectly specified argument value.
[IsoSecu] Calling functions with incorrect arguments [argcomp].

6TFPG+IQIMFGRPVSBQMPJJ³/OFUSF QBSBNFUFST

When reading a function prototype with pointer-type parameters, the absence of the +Q Mqualifier may suggest that a modification will be made to the memory area pointed to. The absence of this qualifier when it should be used makes the definition of interfaces unclear and complicates the proofreading of code. When declaring a function with parameter pointers, the developer should immediately consider how the pointers will be used and use +Q Mby default unless the memory area pointed to is changed when the function is executed.

36-&

36-&±1PJOUFS UZQF GVODUJPO QE OPU UP CF DIBOHFQ MiTU CF EFDMB

Mark as +Q Mall pointer-type parameters of a function that point to a memory area that is read-only in the body of the function. The +Q Mqualifier must be applied to the pointed object.

#BE FYBNQMF

The following example should use +Q Mfor its parameter.

```
mBMijk7iQmBMijknpiHV &
  f   p HHm2f
  mBMijkn2i4 yc
```

```
B7Uh1ahno G=l1p HV &
    `2i4 Up HV   kc
' 2Hb2&
    `2i4 Up HVc
'
`2im`M2ic
'
```

(PPE FYBNQMF

In the following example, + Q Mis incorrectly used for the pointer parameter. The memory area pointed to is not in fact modified in the body of the function.

```
mBMijkn7iQU+QMbmiBMijknpi HV &
  mBMijkn2i4 yc
  B7ULlGG54p HV&
    B7Uh1ahno G=l1p HV &
      `2i4 Up HV   kc
    ' 2Hb2&
      `2i4 Up HVc
    '
  '
  `2im`M2ic
'
```

3FGFSFODFT

[Cert] Rec. DECL00-C Const-qualify immutable objects.
[Cert] Rec DECL13-C Declare function parameters that are pointers to values not changed by the function as const.
[Cert] Rule EXP40-C Do not modify constants objects.
[Misra2012] Rule 8.13 A pointer should point to a const-qualified type whenever possible.
[Cwe] CWE-20 Improper Input Validation.
[Cwe] CWE-369 Divide by Zero.

6BTMOHBGW2O DUJPOT

C99 introduced the new BMHBGW2 keyword as a function specifier. An BMHBGW2 function declared with external linkage but not defined in the same file leads to undefined behaviour. The declaration and definition of an BMHBGW2 function must therefore be in the same compilation unit.

*OGPSNBUJPO

An BMHBGW2 function can be accessed by multiple files by being declared in a header file.

36-&

36-&BMHBGW2O DUJPOT NbV TiBU+CF EFDMB

To avoid undefined behaviour, an BMHBGW2 function is systematically bi iB+

3FGFSFODFT

[Misra2012] Rule 8.10 An inline function shall be declared with the static storage class.
[Cert] Rec. DCL15-C Declare file-scope objects or functions that do not need external linkage as static.
[Cert] MSC40-C Do not violate constraints.

3FEF¾OJOH GVODUJP

A function name can be declared by the programmer even if it is a name already defined in the standard library or in another library. This declaration may lead to confusion. Every function must have its own name.

36-&

36-& ± %PPUSFEF¾OF GVODUJPOT BOPUIFS MJCSBSZ

Identifiers, macros, or function names that are part of the standard library or another library used must not be redefined.

#BE FYBNQMF

In the following example, confusion will occur due to the use of a function name that already exists in the standard library.

```
f   /QMQi`2mb2?2M  K2Q7i?2bi  M/  `/HB#`  `vf
pQB/K  HHQUbBx2nii BHHV2c
```

(PPE FYBNQMF

The following example defines a name that does not clash with the name of a function in the standard library.

```
f   `2M  KBQ7i?27mM+iBQM
pQB/KvK  HHQbBx2nii BHHV2c
```

3FGFSFODFT

[Misra2012] Rule 5.8 Identifiers that define objects or functions with external linkage shall be unique.
[Misra2012] Rule 5.9 Identifiers that define objects or functions with internal linkage shall be unique.

.BOEBUPSZ VTF PG UI

A function, whose return type is not pQ, returns a value indicating the success or failure of the processing or the computation performed by this function. This function return is a very important

source of information and allows unexpected behaviour or even errors to be identified as soon as possible. These function returns must therefore always be read and managed.

The calling function must test the value returned by the function to ensure its validity against the interface documentation (returned value within the value range or returned value corresponding to a success or error code).

36-&

36-& ± 5IF SFUVSO WBMVF PG B GVODU

When a function returns a value, the returned value must systematically be tested.

#BE FYBNQMF

In the following code, the value returned by the function is not tested and no processing is performed if an error has occurred.

```
bi`m+bi i Qnbi in#m7c72`
bi iUjbQK27Bi2-  Qnbi in#m7c72`
f  bm++2Q7i?2bi i 7mM+iBMQQi2bi2/ f
XXX
```

(PPE FYBNQMF

In the following code, the value returned by the function is correctly tested.

```
bi`m+bi i Qnbi in#m7c72`
mBMi3ni      Bn`2bm4Hiyc
Bn`2bm4Hbi iUjbQK27Bi2-  Qnbi in#m7c72`
B7Uy 54Bn`2bmVH&
    f  2``2m`f
    `2im`Mc
'
XXX
```

3FGFSFODFT

[Cert] Rec. EXP12-C Do not ignore values returned by functions.
[Misra2012] Dir. 4.7: If a function returns error information, then that error information shall be tested.
[Misra2012] Rule 17.7 The value returned by a function having non-void return type shall be used.
[Cwe] CWE-252 Unchecked Return Value.
[Cwe] CWE-253 Incorrect Check of Function Return Value.
[Cwe] CWE-754 Improper check for unusual or exceptional conditions.

*NQMJDJU SpFQUBSVSOD DQUSJ

In the absence of an explicit return value for all paths of a function returning a value (non- p Q B / function), some compilers do not always generate an error. The behaviour of the code is then undefined. Some compilers return an arbitrary value.

36-& ± *NQMJDJU SpFQUBWSZQ QSVDUJDF

All paths of a non- p Q Bunction must return a value explicitly.

#BE FYBNQMF

In the following example, there are paths that do not explicitly return a value.

```
mBMijk2iM+`n/U+QMbmiBMi3nTn/i- mBMijkmiBjkn/inH2M
                mBMi3niTTn2M+`vTi2/n/miBMijknmBjkn2M+`vTi2/n/VinH2M
                    &
  mBMi3nTn2M+`vTi2/n4 LlGcG
  B7ULlGG54Tn/i
       LlGG54TTn2M+`vTi2/n/i
       LlGG54mBjkn2M+`vTi2/n/V &H2M
     B7UmBjkn/inH2MV &
       Tn2M+`vTi2/n4 iUmBMi3nV+ HHQm+Bjkn/inH2MBx2Qm7BMi3VnVc
       XXX
       `2im`Rc
     '
   '
   f  BKTHB+2im`Mf
 '
```

(PPE FYBNQMF

In the following code, the function code always explicitly returns a value.

```
mBMijk2iM+`n/U+QMbmiBMi3nTn/i- mBMijkmiBjkn/inH2M
                mBMi3niTTn2M+`vTi2/n/miBMijknmBjkn2M+`vTi2/n/VinH2M
                    &
  mBMijkmiBjkn`2bmHim4+Qc2
  mBMi3nTn2M+`vTi2/n4 LlGcG
  B7ULlGG44Tn/i
  %%LlGG44TTn2M+`vTi2/n/i
  %%LlGG44mBjkn2M+`vTi2/n/V &H2M
    mBjkn`2bmHim4+Qc2
    ;QiQ1Mc
  '
  B7Uy 44mBjkn/inV2&M
    mBjkn`2bmHim4+Qc2
    ;QiQ1Mc
  '
  Tn2M+`vTi2/n4 iUmBMi3nV+ HHQm+Bjkn/inH2MBx2Qm7BMi3VnVc
  XXX
  UTTn2M+`vTi2/V/4iTn2M+`vTi2/nc/i
  mBjkn`2bmHim4+Qc2
1M,/
  `2im`MmBjkn`2bmHicn+Q/2
'
```

3FGFSFODFT

[Misra2012] Rule 17.4 All exit paths from a function with non-void return type shall have an explicit return statement with an expression.
[Cert] Rule MSC37-C Ensure that control never reaches the end of a non-void function.

/P QBTTJOH CZ WBMVF QBSBNFUFS

It is possible with the C language to pass structure as parameters of a function. These are then copied to the stack. However, this is detrimental to performance and increases the risk of stack overflow or even leakage of sensitive data.

The parameter corresponding to a structure must be passed in the form of a +Q M-qualified pointer. Only the address of the structure is then copied to the stack. Furthermore, the +Q M bi modifier prevents changes to the pointed object (which is desirable when passing the structure by value).

36-&

36-& ± 4USVDUVSFT NVTU CF QBTTFE

Do not pass structure type parameters by copying when calling a function.

#BE FYBNQMF

In the following example, the parameter is passed by value and not by address.

```
O/27BMa2h_naAw1 kyl
ivT2/2Ti`m+i
&
  mMbB;M+2/`bm`M(ka2h_naAw1
  mMbB;M+2/`7B`biM(ka2h_naAw1
' T2`bQMi
mBMijkn//nT2`bQTM2`bQMTn2`bQYM&
  bBx2nibxnbm`M K24HyRcM
  bBx2nibxn7B`biM K24HyRcM
  bxnbm`M K24HyRcM2`MH2UM2`bQXbm`M VKc2
  bxn7B`biM K24HyRcM2`MH2UM2`bQX7B`biM VKc2
  B7Uy 54bxnbm`M K2nHy2M54bxn7B`biM K2Vn&l2M
    XXX
    mBjkn`2bm4HRc
  ' 2Hb2&
    mBjkn`2bm4Hyic
  '
  `2im`MmBjkn`2bomHi
'
XXX
pQBbQK2n7mMU+VB&QM
  T2`bQMTn2`bQQM
  XXX
   //nT2`bQTM2`bQYbM
  XXX
'
```

(PPE FYBNQMF

The following code correctly passes a structure type parameter using a pointer.

```
O/27BMa2h_naAw1 kyl
ivT2/2Ti`m+i
&
  mMbB;M+2/`bm`M(ka2h_naAw1
  mMbB;M+2/`7B`biM(ka2h_naAw1
' T2`bQMi
```

```
m B M i j k n / / n T 2 ` b Q M b T i 2 ` b Q M n T i 2 ` b Q M &
  m B M i j k n i B j k n ` 2 b m 4 H y i c
  b B x 2 n i b x n b m ` M K 2 4 n H y 2 c M
  b B x 2 n i b x n 7 B ` b i M K 2 4 n H y 2 c M
  B 7 U L I G 5 4 T 2 ` b Q M &
    b x n b m ` M K 2 4 n H y 2 M 4 T 2 ` b Q M 4 m ` M V K c 2
    b x n 7 B ` b i M K 2 4 n H y 2 M 4 T 2 ` b Q M 4 B ` b i M V K c 2

    B 7 U y 5 4 b x n b m ` M K 2 n H y 2 M 5 4 b x n 7 B ` b i M K 2 n & 2 M
      X X X
      m B j k n ` 2 b m 4 H R i c
    ' 2 H b 2 &
      m B j k n ` 2 b m 4 H y i c
    '
  '
  2 H b 2 &
      m B j k n ` 2 b m 4 H y i c
  '
  ` 2 i m ` M m B j k n ` 2 b c m H i
'

p Q B b Q K 2 n 7 m M U + V B & Q M
  T 2 ` b Q M T n 2 ` b Q M
  X X X
   / / n T 2 ` b Q M 2 ` b Q M
  X X X
'
```

1BTTJOH BO BSSBZ BT

When a function takes a pointer as a parameter, it is not possible to determine whether the pointer is the address of the first element of an array or whether it points to a single element.

To suppress this ambiguity, it is preferable to use the form with ( )for an array type parameter as indicated in sub-section 8.1.

3&$0..&/%"5*0/ ± 1BTTJOH PG BO BS

There are several ways to pass an array as a parameter for a function. When passing by pointer, it must be specified in the function documentation that the parameter corresponds to an array and also use the dedicated array notation.

8BSOJOH

For a multi-dimensional array, only the first dimension of the array can remain undefined when passing as a parameter, which therefore means defining the following dimensions. For example, for a two-dimensional array, using i # ( ) (a)s a parameter is a mistake, as a minimum the second dimension must be specified.

#BE FYBNQMF

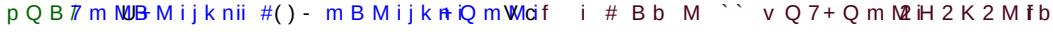
The following example shows a prototype function with a pointer-type parameter. It is an array passed as a parameter but this cannot be guessed from the function signature. In this example, tab can:

- either be an integer passed by address

- or be an array of integers.

EXAMPLE

In this second example, the notation and comments ensure that it can be immediately determined that the parameter is indeed an array.



UNUSED PARAMETERS

Failure to use a parameter in the implementation of a function is usually a developer error. It also consumes unnecessary space on the stack.

The prototype of the function must be modified if the parameter is not useful.

However, in some cases, it may be justified not to use one (or more) parameters of a function:

- the function corresponds to a callback function whose prototype is imposed;
- for reasons of compatibility with existing code when upgrading a library. A previously used parameter is no longer used;
- in the case of a future development in which the parameter will be used.

In all these cases, a comment must then explicitly state why the parameter is ignored.

RULE

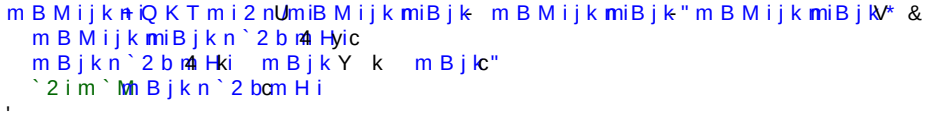
All the parameters present in the prototype of the function must be used in its implementation.

INFORMATION

The option is used to provide alerts in this scenario.

EXAMPLE

In the following example, a parameter of the function is not used and should therefore be deleted.



(PPE FYBNQMF

In the following code, there are no unused parameters in the implementation of the function.

```
mBMijkniQKTmi2nUmiBMijkmiBjk- mBMijkmiBjkV" &
  mBMijkmiBjkn`2bm4Hyic
  mBjkn`2bm4Hki  mBjkY k  mBjkc"
  `2im`MmBjkn`2bcmHi
'
```

3FGFSFODFT

[Misra2012] Rule 2.7 There should be no unused parameters in functions.
[Cert] Rule EXP37-C Call functions with the correct number and type of arguments.

7BSJBEJD GVODUJPO

Variadic functions (• Àt‰oÀe with a variable number of arguments or with varying types) can pose several problems. It is not advisable to define variadic functions, but the standard library itself contains several such definitions that are often used. The type of arguments of a variadic function is not checked by the compiler, by default, which can, if these functions are used incorrectly, lead to some surprises such as abnormal termination or unexpected behaviour.

*OGPSNBUJPO

The @ q 7 Q ` Kopti4k, required by subsection 5.3.1, allows the compiler to extend its checks to arguments of variadic functions.

When L I GisGpassed to a classic function, L I GisGconverted to the correct type. This type conversion does not work with the variadic functions since the “right type” is not known. In particular, the standard allows L I GtoGbe an integer constant or a pointer constant so on platforms where L I G G is also an integer constant, passing L I GfGr variadic functions can lead to undefined behaviour.

36-& ± %PPOUDBMMIWBTSBEBJSDHVWOF

#BE FYBNQMF

```
XXX
mMbB;M+2/`bi`BM4 LIGcG
T`BMU7Wb W$Mj- bi`BM;RVd  mM/27BM2?  pBQn`
XXX
```

(PPE FYBNQMF

XXX
mMbB;M+2/` bi`BM4 LIGcG
T`BMU7Wb W$M]- Ubi`BM\ bi`BM;]MmHV+ RVfc MQiT bbBMLJGGf
XXX

3FGFSFODFT

[Misra2012] The features of Ibi/ `; X 9hall not be used.
[Cert] Rec. DCL10-C Maintain the contract between the writer and the caller of variadic functions.
[Cert] Rec. DCL11-C Understand the type issues associated with variadic functions.
[Cert] Rule EXP47-C Do not call va_arg with an argument of the incorrect type.
[Cert] Rule MSC39-C Do not call va_arg on a va_list that has an indeterminate value.
[Cwe] CWE-628 Function call with incorrectly specified arguments.
[IsoSecu] Calling functions with incorrect arguments [argcomp].

14
Sensitive operators

6TF PG UIF DPNNB QSP

The comma should be used as a separator for the parameters of a function or as a separator for initialising fields in a structure or array. Use of the comma is also tolerated when making a declaration, subject to compliance with the other rules on multiple declarations. However, the use of commas to string together statements in the C language makes the code difficult to read, and may lead to unexpected results.

36-&

36-& ± 6TF PG UIF DPNNB QSPIJCJUF

The comma is not authorised when sequencing code statements.

The comma must be replaced by a semicolon for statement sequences. This means that:

- braces become necessary;
- the parameters of 7 Qlòops must be reorganised.

#BE FYBNQMF

The following example makes use of the comma in expressions. It is not possible to know the result of these statements when reading the code.

```
BMijknB4 UD4 k- RVc
v4t\ UYY- Y 9V ,+c
x4j  # Y k- d + Y 9kc
  4 U# 4 k-+4 j-/ 4 9Vc
7QÙB4 y-D4 awnJcsBI awnJcsBYY-D@ @V &
  XXX
'
```

(PPE FYBNQMF

In the following code, the comma is only used for the declaration of variables.

```
BMijknB- Dc
B4Rc
D4kc
B7Uy 54V &
   YYc
  v4  Y 9c
' 2Hb2&
  v4 +c
'
x4j  # Y kc
D4awnJcs
```

```
7 QÙB 4  ycB I  a w n JcsBY Y V  &
  X X X
  D@ @ c
'
```

3 F G F S F O D F T

[Misra2012] Rule 12.3: The comma operator shall not be used.

6 T J O H Y Q B S Q E Q F T U B 4 P S T
D P N Q P V O E B T T J H O N F O U

When the pre/postfix Y Y and @ @ operators are used within a calculation, it is very difficult to establish the result of the calculation when analysing the code. Moreover, it is also a source of confusion and even errors for the developer. These operators must therefore be used alone in a statement. As a result, since pre/postfix operators are semantically equivalent when used in isolation in a statement (• À % À Y c Y, Y postfix operators are authorised in order to avoid any ambiguity [13]. Prefix operators will not be used.

3 & $ 0

3 & $ 0 . . & / % " 5 * 0 / ± 5 Y Q E P Y V P M Q E F O S B U

Pre-increment and pre-decrement operators will not be used.

Lastly, complex statements will be broken down into simple elements.

3 & $ 0

3 & $ 0 . . & / % " 5 * 0 / ± / P D P N C J O F E V T F P P Q F S B U P S T

Post-increment and post-decrement operators should not be mixed with other operators.

Lastly, and again for readability reasons, it is recommended not to use combined assignment operators (= =, 4 4, 4 ‰). ‡ À

3 & $ 0

3 & $ 0 . . & / % " 5 * 0 / ± " W P J E U I F V T F P G D

B E F Y B N Q M F

The code below uses postfix operators mixed with other operators. The behaviour of this code is not specified. It depends on the compiler used.

13. The choice of postfix operators can be discussed since the two operators used in isolation are equivalent.

```
O 2 7 B Mh2 " n a A w 1  k 8 l

X X X

m B M i j k n d
m B M i 3 n # ( h  " n a A ) w 4  &  y  ' c
m B M i R e B n 4  y c
t  4  7 Q U Q B Y Y - B V c  f   M Q i b T 2 + B 7 , B 7 2 / Q # H 2 B K i ? i ? 2 Q ` / 2 ` Q 7 2 p  H m  i B Q Q 7 M ? 2
    T  `  K 2 i 2 ` b f

X X X

m B M i j k n 7 i Q U Q m B M i R e n i m B M i R e # N i &
  ` 2 i m ` M   # c
'
```

(PPE FYBNQMF

In the following example, postfix operators are used in isolated statements.

```
O 2 7 B Mh2 " n a A w 1  k 8 l

X X X

m B M i j k n d
m B M i 3 n # ( h  " n a A ) w 4  &  y  ' c
m B M i R e B n 4  y c
B Y Y c f   L X " X ` 2 T H  + B i M ? 2 B b b i  i 2 K 2 M i B i ? Y Y B c  r Q m H M Q i + ?  M ; 2 ? 2 # 2 ?  p B Q Q m 7 `
    i ? 2 T ` Q ; `  K B M  M v r  v  f
t  4  7 Q U Q B -  B V c

X X X

m B M i j k n 7 i Q U Q m B M i R e n i m B M i R e # N i &
  ` 2 i m ` M   # c
'
```

3FGFSFODFT

[Misra2012] Rule 13.3 A full expression containing an increment or decrement operator should have no other potential side effects other than that caused by the increment or decrement operator.
[Cert] Rule EXP30-C Do not depend on the order of evaluation for side effects.

/POFTUFE VTF PG UIF

The ternary operator `\?`, can be used to concisely write an assignment of a variable based on a condition.

However, when the ternary operator is used with a complex conditional expression, or if several ternary operators are nested, understanding the code and its maintenance becomes difficult.

In the case of complex expressions, an `if` conditional must be used.

36-&

36-& ± /POFTUFE VTF PG UIF UFSOBS

The nesting of ternary operators `\?`, is prohibited.

Moreover, if the types of expressions are different in the two “branches”, this implies an implicit cast according to the value of the condition of the ternary operator.

36-& ± $PSSFDU DPOTUSVDUJPO PG U UP,S

The expressions resulting from the ternary operator \ , must be exactly of the same type to avoid any type conversion.

#BE FYBNQMF

In the following example, the nested ternary expression makes it difficult to understand the code, and the expressions of the two branches are not of the same type.

```
v 4 UI 9kV \ Ry9ki = UyV \ @Ry9k , yXyc
f  BMi2;2`M/7HQ-ii?2`27QB2KTHB+BbiI f
```

(PPE FYBNQMF

The following example uses several B 7 @ 2dmd2 ionals in order to handle the assignment of a value to the variable y which depends on several conditions.

```
BTUI 9kV &
  v 4 Ry9kc
' 2Hb2&
  BTUI = yV &
    v 4 @Ry9kc
  ' 2Hb2&
      v 4 yc
  '
'
```

3FGFSFODFT

15
Memory management

%ZOBNJDNFNPSZBMM

For all objects dynamically allocated by the developer, different rules must be respected. First of all, the developer must ensure that they have allocated enough memory for the object in question. A common error is to apply the b B x 2 Q7 operator on a pointer of the object to be allocated instead of the object to be allocated directly, or not to apply this operator to the correct type.

36-&

36-&±%ZOBNJDBMMZBMMPDBUFTVPCKFDU

For a T i ` pointer, it is preferable to use T i ` 4 K H H Q + U b B x 2 Q7 whenever possible.

In addition, any dynamically allocated memory should be freed as soon as possible.

36-&

36-&±'SFFEZOBNJDBMMZBMMPDB

Any dynamically-allocated memory space must be freed up when it is no longer needed.

This rule echoes that of section 8.6.

For objects storing sensitive data, the memory areas must be reset before being freed.

36-&

36-&±4FOTJUJWFNFNPSZBSFBTNV

8BSOJOH

It is crucial to ensure that this memory reset code is not optimised and is retained on compilation. Most compilers consider this reset as dead code since the associated variables are not used afterwards. Generally speaking, the levels of optimisation should not be pushed at compilation but sometimes, even at a low level of optimisation, one must unfortunately recode one's own K 2 K to avoid this kind of inconvenience.

It is also important to note that memory release is only authorised for dynamically-allocated objects.

36-&±%POPUGSFFNFNPSZOPUBMM

Finally, `2 HHQust not be used to modify memory space allocated dynamically. This function can in fact change the memory space allocated to an object by increasing or decreasing its size, but can also free the memory of the object passed as parameter. Because of the risks associated with memory handling or the potential double memory release corresponding to undefined behaviour, use of this function should be avoided.

36-&±%POPUDIBOHF`2IHHQ⊕BNJDB

8BSOJOH

If this fails, the `2 HHfQnction returns LIG, but the initial memory location remains intact and is therefore still accessible.

#BE FYBNQMF

```
OB M + H m1 b2 / H B # X ? =
p Q B 7 Q MU+ B x 2 nHi 2 WM&
  H Q M ;Tc
  T 4 UH Q M ;V K  H H QH+2 M b B x 2 QB7MV V cf  B M + Q ` ` 2 i ⊻ T 2 f
  X X X
  T 4 UH Q M ;V ` 2  H H QT+y V cf  ` 2 H 2 bQ2 7T pB ` 2  H H Q f+
  X X X
  7 ` 2 QTV c f  / Q m # H 2 H 2 bQ2 7T  f
'
```

(PPE FYBNQMF

```
OB M + H m1 b2 / H B # X ? =
p Q B 7 Q MU+ B x 2 nHi 2 WM&
  H Q M ;Tc
  T 4 UH Q M ;V K  H H QH+2 M b B x 2 QH7Q M ;V V cf  + Q ` ` 2 + i 2 ⊻ T 2 f
  X X X
  7 ` 2 QTV c f  ` 2  H H Q 2 H 2 i 2 /M / ` 2 T H  + 2#v 7 ` 2 2 f
'
```

3FGFSFODFT

[Cert] Rec. MEM00-C Allocate and free memory in the same module, at the same level of abstraction.
[Cert] Rule MEM31-C Free dynamically allocated memory when no longer needed.
[Cert] Rule MEM34-C Only free memory allocated dynamically.
[Cert] Rule MEM35-C Allocate sufficient memory for an object.
[Cert] Rule MEM36-C Do not modify the alignment of objects by calling realloc.
[Cert] Rec. MEM03-C Clear sensitive information stored in reusable resources.

[Cert] Rec. MEM04-C Beware of zero-length allocations.
[Misra2012] Rule 22.1 All resources obtained dynamically by means of Standard Library functions shall be explicitly released.
[Misra2012] Rule 22.2 A block of memory shall only be freed if it was allocated by means of a Standard Library function.
[Cwe] CWE-226 Sensitive information uncleared before release.
[Cwe] CWE-244 Failure to clear heap memory before release (”heap inspection”).
[Cwe] CWE-590 Free of memory not on the heap.
[Cwe] CWE-672 Operation on a resource after expiration or release.
[Cwe] CWE-131 Incorrect Calculation of Buffer Size.
[Cwe] CWE-680 Integer Overflow to Buffer Overflow.
[Cwe] CWE-789 Uncontrolled Memory Allocation.
[IsoSecu] Accessing freed memory [accfree].
[IsoSecu] Freeing memory multiple times [dblfree].
[IsoSecu] Reallocating or freeing memory that was not dynamically allocated [xfree].
[IsoSecu] Taking the size of a pointer to determine the size of the pointed-to type [sizeofptr].
[IsoSecu] Allocating insufficient memory [insufmem].

6 T H E B Y G 2 0 0 F S B U P S

The b B x 2 0 7 rator is essential in C in order to know the size of an object in memory. However, careless use of this operator can lead to unexpected behaviour and result in an incorrect memory size or lead to an unevaluated expression.

It is preferable to use the object type and not the identifier of a variable as a parameter of the b B x 2 0 7 rator. Using the identifier has the advantage of being “resistant” to the associated type change, but it is then necessary to ensure that the use of the b B x 2 0 7 rator is correct.

In order to avoid problems related to the alignment of the members of a structure, it is common to use:

- either the pre-compilation directive " … ;‡ •
- or an explicit padding field.

8 B S O J O H

The " … directive is not standard.

If an alignment of the members of a structure is required, the pre-compilation " … directive or padding fields can be used.

Moreover, the use of the idiomatic expression b B x 2 Q 7 U ` ` v V f b B x 2 Q 7 U determin( ) V the number of elements in an array is quite traditional, but great care must be taken in its use. This expression is only correct if the b B x 2 0 7 rator is applied to the array in the block in which the array is declared. The result of this expression will be quite different if the b B x 2 0 7 rator

is applied to an array passed as parameter because this array will then be a pointer type and no longer an array.

3 6 - &

3 6 - & ± $ P S S Bb Bx 2nQ7t UPFS

An expression contained in a b B x 2nQ7t not:

- contain the operator “=”, since the expression will not be evaluated;
- contain pointer dereferencing;
- be applied to a pointer representing an array.

8 B S O J O H

The b B x 2oQe7rator does not return the size of the object, but the size used in memory.

B E F Y B N Q M F

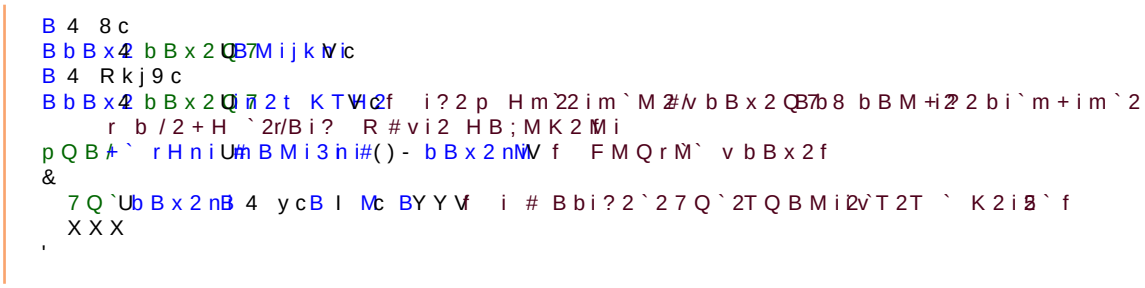
The following example shows incorrect use of the b B x 2oQe7rator.

```
mBMi3ni#(G1)Lc
ivT2/27i`m+bn2t KTH2
  mBMijkmiBn7B2cH/R
  mBMi3miBn7B2cH/k
' in2t KTcH2

BMijknB- BbBxc2
in2t2KTH2bic
in2t2KTH2i`c

B 4 8c
Ti` 4 LlGcG
BbBx4bBx2QB74 Rkj9Vfc i?22tT`2bbBBQMRkj9rBHHMQi#2T`Q+2bb2/
f B? bi?2p Hm2- M/MQiRkj9XBbBx2[m H9b f
BbBx4bBx2Q7i2t KTVHc2f i?2p Hm`22im`M2#vbBx2QB7b37Q`jk@Bi
    HB;MK2Mi
BbBx4bBx2Q7i`Vcf i?22tT`2bbBbQBMx2Q7i`V`2im`Mib?2bBx2Q7i?2
    bi`m+imi22t KTH2
pQB+` rHniUmBMi3ni#()V
&
  7Q`UbBx2nB 4 ycBl bBx2Q7#V fbBx2Q7#(y)VcBYYVf i # Bbi?2`27Q`2
    TQBMi2v`T2T ` K2i2`f
  XXX
'
```

5 P M F S B U F E F Y B N Q M F

The following example makes good use of the alignment directive and does not use an expression as a parameter of the call to the b B x 2oQe7rator.

```
OT` ;KT +FUTmb?f RRV#vi2 HB;MK2@MLiPhah L. _@ f
mBMi3ni#(G1)Lc
ivT2/27i`m+bn2t KTH2
&
  mBMijknmBn7B2cH/R
  mBMi3nimBn7B2cH/k
' in2t2KTcH2
OT` ;KT +FUTQ7TV2im`M27 mHiHB;MK2Mi

BMijknBc
bBx2nBbBxc2
```



3FGFSFODFT

[Cert] Rec. EXP09-C Use sizeof to determine the size of a type of a variable.
[Cert] Rule EXP44-C Do not rely on side effects in operand to sizeof, _Alignof, or _Generic.
[Cert] Rec. ARR01-C Do not apply the sizeof operator to a pointer when taking the size of an array.
[Misra2012] Rule 13.6 The operand of the sizeof operator shall not contain any expression which has potential side effects.
[IsoSecu] Taking the size of a pointer to determine the size of the pointed-to type [sizeofptr].
[Cwe] CWE-131 Incorrect Calculation of Buffer Size.
[Cwe] CWE-467 Use of b B x 2 Qo7 nu a pointer type.
[Cwe] CWE-805 Buffer access with incorrect length value.

.BOEBUPSZ WFSJ¾DB BMMPDBUJPO

When a memory allocation is made, it may fail if there is no more free memory in the system. Failure to test the pointer returned by the allocation function will cause the program to crash the first time the pointer is used.

Usually, if allocation fails, the memory allocation function returns a L l G pointer. It is therefore necessary to check that the pointer returned by the allocation function is not L l G G

If the allocation function behaves differently on an allocation error, refer to the documentation of the function for how to handle the error appropriately.

36-& ± .BOEBUPSZ WFSJ¾DBUJPO PG

36-&

The success of a memory allocation must always be checked.

This rule is a special case of section 13.7 but with special attention paid to the handling of memory allocation errors.

#BE FYBNQMF

In the following example, the memory allocation success check is missing.

```
TQBMinTnTQBcMi
TnTQB4MiUTQBMinVK HHUb+Bx2QT7QBMVmMc
f MQ+?2+e M7mM+iB`Q2Mm`Mf
TnTQB@M=i4 yX7yc
TnTQB@M=i4 yX7yc
```

(PPE FYBNQMF

In the following code, the success of the memory allocation is checked before the pointer is used.

```
TQBMiniTnTQB4MLLIGcG

TnTQB4MiUTQBMinVK HHUb+Bx2QT7QBMVmMc

B7ULIGG54TnTQBVM&
    TnTQB@M=i4 yX7yc
    TnTQB@M=i4 yX7yc
' 2Hb2&
    f 2``Q`? M/HBMf;
'
```

*TPMBUJPO PG TFOTJU

When sensitive data is loaded into the memory (e.g. encryption keys), it remains in memory after the program has finished accessing it. Another program can access our program's memory via … ™ œ • • • … –•.‡ Œ … ’ ’ ‰ • —[14]

It is therefore necessary to associate memory areas with their use: data representing different values is stored in separate memory spaces. If a shared memory area is recycled, make sure that it is erased before being reused.

All memory areas that contain sensitive data must be explicitly deleted once the program no longer needs to access this data.

8BSOJOH

Clearing buffers so that data does not remain on the stack via a K2Kbf2i example, may be considered unnecessary by the compiler and the associated calls can therefore be deleted in order to optimise the code. The developer should be aware of this risk and consult the documentation of the compiler used in order to ensure that the calls in question are properly stored.

36-&

36-& ± 4FOTJUJWF EBUB NVTU CF JTPI

Check the correct use of a memory area storing sensitive data, • À‰ÀmiÀmise memory exposure, minimise copying and delete the area(s) that contained the sensitive data as soon as possible.

14. There are many illustrations of this: Meltdown, Spectre, ZombieLoad, ‰ ˜ ‡ À

#BE FYBNQMF

In the example below, the same buffer is used to store the key and then the initialisation vector.

```
O/27BMcq2P_EnaAw1 jkl

OBM+Hmlb2/HB#X?=

pQB7T`Q+2Ub+bQMbmiBMi3nF2v mBMiReFm2vnbBx+2QMbmiBMi3nBMBi
            mBMiReBmMBinbVBx&2
  mBMi3#im77(2q`P_Ena)Acw1

  B7UUlGG44F2V %%LlGG44BMBViV &
    f 2``Q`? M/HBMf;
    XXX
  '

  K2K+U#vm77-2F2v KBUUF2vnbBxq2P_EnaVAVwc1
  T`BMinU#2m77V2c

  f #m772+`QMi BMHvi2bQ7i?2BMBiB HBb pE2QMQ`M/i?2H biRk#vi2bQ7
     i?2F2v f
  K2K+U#vm77-2`BMBiKBUUBMBinb+Bxq2P_EnaVAVwc1
  T`BMiU#om77V2c
  XXX
  f MQb2+m`/22H2iBQfM
'
```

(PPE FYBNQMF

The following example shows a partition between the key and the initialisation vector.

```
O/27BME21unaAw1 Rel
O/27BMA2onaAw1 9l

OBM+Hmlb2/HB#X?=

pQB7T`Q+2Ub+bQMbmiBMi3nF2v mBMiReFm2vnbBx+2QMbmiBMi3nBMBi
            mBMiReBmMBinbVBx&2
  mBMi3KivnF(2E1unaA)ov1
  mBMi3Bi(AonaA)wv1

  B7UUlGG44F2V %%LlGG44BMBViV &
    f 2``Q`? M/HBMf;
    XXX
  '

  K2K+UKvvnF2vF2v KBUUF2vnbBxE21unaAVwV1c
  T`BMinUK2vvnFV2vc

  K2K+UBvp BMBiKBUUBMBinb+BxA2onaAVwV1c
  T`BMiUBqpVc
  XXX
  f /2H2iBQQMi?2#m772-`bQi? ii?2/ i /Q2bMQi`2K BQQMi?2bi +F
   hh1LhA,P122+QKTBH+2MQTiBKBbM2//2H2i2??2b2+ HH+br?B+1K v#2
      +QMbB/2`m2MM2+2bXb `v
  AiBbi?2`27QM22+2bbi`Qv+QMbmiP12+QKTBH/Q`+mK2Mi iBMQQM/2`7Q`i?2
```

[Cert] Rec. MSC18-C Be careful while handling sensitive data, such as passwords, in program code.

16
Error management

$ P S 2 F `D M U Q V T F P G

The 2 ` ` M Q variable, activated by the header file, I 2 ` ` M Q, is of type B M and different functions in the standard library change its value with a positive value in case of error. It is therefore important to initialise 2 ` ` M Q before any function call from the standard library that changes its value, and it is therefore also necessary to consult its value at the end of the execution of such functions.

36-&

3 6 - & ± * O J U J B M J T 2 F ` B M Q Q E Q V P J S F X B U O F E V B E
D V U J P O P G B T U B O E B S E M J C S B S Z G

B E F Y B N Q M F

```
OB M + H mi b2i / H B # X ? =
pQ B i/ ` v R U+ Q M bmi M b B ; M+2?/ `  H 2Wi
&
  m M b B ; M+H2Q/ M ` ;2 bc
  ` 2 b 4 bi ` iQ mUHH2+ML lG-G8 V cf  + Q M p 2 ` b+B?Q M  + i 2 bi ` B Mi;Q m M b B ; M+H2Q/ M ;f
  f  i ? 2 7 m M + i BbQ M iQ m+H B i 2 bB M2 ` ` M Qf
  B 7 U ` 2 b 4 4 lG P L : n J V s
  &
    f  T ` Q # H 2XK M  ; 2 K 2 Mi
  '
X X X
'
```

(P P E F Y B N Q M F

```
OB M + H mi b2i / H B # X ? =
OB M + H mi 22 ` M Q X ? =
pQ B i/ ` v R U+ Q M bmi M b B ; M+2?/ `  H 2Wi
&
  m M b B ; M+H2Q/ M ` ;2 bc
  2 ` ` M Q4 y cf  B M B 2 ` ` M Qf
  ` 2 b 4 bi ` iQ mUHH2+ML lG-G8 V cf  + Q M p 2 ` b+B?Q M  + i 2 bi ` B Mi;Q m M b B ; M+H2Q/ M ;f
  f  bi ` iQ m+H B i i 2 MB M2 ` ` M Qf
  B 7 U ` 2 b 4 4 lG P L : n J s 2 ` ` M Q4 y Vf  2 ` ` M Q2 / B M ;f
  &
    f  T ` Q # H 2XK M  ; 2 K 2 Mi
  '
X X X
'
```

3FGFSFODFT

[Cert] Rule ERR30-C Set errno to zero before calling a library function known to set errno and check errno only after the function returns a value indicated failure.
[Cert] Rule ERR32-C Do not rely on indeterminate values of errno.
[IsoSecu] Incorrectly setting and using errno [inverrno].
[Cwe] CWE-456 Missing Initialisation of a variable.

4ZTUFNBUJD DPOTJEF
TUBOEBSE MJCSBSZ GV

Most of the standard library functions return values to indicate the correct operation of the function, but also an error when the function is executed. Failure to test the return value may lead to the use of erroneous data produced by the function.

36-&

36-& ± "MM FSSPST SFUVSOFE CZ TUB

Any function return must be read in order to set up the appropriate processing following the execution of the function.

This rule is a specific case of section 13.7 but with special attention paid to the error management of standard library functions.

#BE FYBNQMF

```
OB M + H m1 b2 / B Q X ? =
OB M + H m1 b2 / H B # X ? =
B M iK  BU0 Q B// 
&
  6 A G 17 T 4  7 Q T 2JM v 7 B X 12i ] -  ] r ] V c f   ` 2 i m ` M 2p/ H m 2 Q i 2  /  f
  7 T m Up ? 2 H 5MQ -  7 TV c
  X X X
'
```

(PPE FYBNQMF

```
OB M + H m1 b2 / B Q X ? =
OB M + H m1 b2 / H B # X ? =
B M iK  BU0 Q B// 
&
  6 A G 17 T 4  7 Q T 2JM v 7 B X 12i ] -  ] r ] V c
  B 7U7 T 5 4 L I GV& 
    7 T m Up ? 2 H 5MQ -  7 TV c
  X X X
  '
'
```

3FGFSFODFT

[Misra2012] Dir. 4.7: If a function returns error information, then that error information shall be tested.
[Cert] EXP12-C Do not ignore values returned by functions.
[Cert] ERR33-C Detect and handle standard library errors.
[Cert] FIO37-C Do not assume that fgets() or fgetsw() returns a nonempty string when successful.
[Cwe] CWE-241 Improper Handling of Unexpected Data Type.
[Cwe] CWE-252 Unchecked Return Value.
[Cwe] CWE-253 Incorrect Check of Function Return Value.
[Cwe] CWE-391 Unchecked Error Condition.
[IsoSecu] Failing to detect and handle library errors [liberr].
[IsoSecu] Forming invalid pointers by library function [libptr].

%PDVNFOUBUJPO BOE

Incomplete documentation of the prototype of a function can lead to programming errors, especially in error management, if all return codes are not indicated with their meaning.

A documentation template for error codes must be defined. This should contain, for each return code, the associated error and, in the event that several error codes may occur at the same time, the priority between these codes must be specified for error management.

36-&

36-& ± &SSPS DPEF EPDVNFOUBUJPO

All error codes returned by a function must be documented. If several error codes can be returned at the same time by the function, the documentation must define the priority for handling these codes.

Error codes must contain information. Without structuring, the information indicated by the return code is often insufficient. The structuring of return codes via masks is one possibility. The return codes must also be structured in such a way that it can be determined whether the value comes from a normal execution of the function, or on the contrary whether an external element has interfered (buffer overflow, ‰ ʖ.‡ À

3&$0

3&$0..&/%"5*0/ ± 4USVDUVSJOH PG

The return codes must be structured in such a way that information on the progress of the function called can be obtained easily:

- error;
- error type;
- alarm;
- alarm type;
- ok;

n ‰ ˜ ‡ À

3FGFSFODFT

[Cert] Rec. ERR00-C Adopt and implement a consistent and comprehensive error-handling policy.

3FUVSO DPEF PG B $QS FYFDVUFE TVDDFTTGVM

The management of the return code of a program is not identical from one operating system to another or from one shell to another. This can therefore cause problems of code portability. From one operating system to another, or from one shell to another, the authorised value range for the return code of a program varies:

- on Windows, the shell + K / X 2 authorises signed 32-bit integers (value accessible in the 1 _ _ P _ G 1 o 1 G variable);
- on Linux, the shell authorises a value between 0 and 255 (even if some codes are reserved for signals; the value is accessible via the variable 0).

The use of a return code between 0 and 127 protects against the risks of modification (by type conversion) or misinterpretation of the return code of a program:

- values between 0 and 127 can be coded over 7 bits;
- they have the same coding whether the integer type is signed or unsigned.

36-&

36-&±3FUVSO DPEF PG B $QSPHSBN B

The return code of a C program must have a meaning in order to indicate that the program has run correctly or that an error has occurred:

- the value of the return code must be between 0 and 127;
- the value 0 indicates that the program was executed without errors;
- the value 2 is generally used in Unix to indicate an error in the arguments passed as parameters for the program.

The meaning of the program's return codes must be indicated in its documentation.

#BE FYBNQMF

The following code presents a portability problem between Windows and Linux. The value -1 is in fact converted to 255 on Linux with the bash shell.

```
BMiK BUBMi`;+- +? ` `;p()V &
  B7U`;+54 kV &
    f BM+Q``2M+mK#Q7`;mK2Mifb
    `2im`M@Rc i?2`2im`MQ/2BHMQi#2BMi2`T`2i2Q``2+ikQvGBMmft
'
XXX
`2im`Mc
```

'

(PPE FYBNQMF

In the following example, the return codes do not pose a portability problem.

```
O27BM_21alGhnPE UylV
O27BM2_:n1__P_ UklV

BMiK BUBMi`;+- +?  `  `;p()V &
  B7U`;+54 kV &
    f BM+Q``2M+mK#Q7 `;mK2Mifb
    `2im`M_:n1__oP_
  '
  XXX
  `2im`M1alGhncPE
'
```

&OEJOH PG B $QSPHS

When multiple exit points are defined in a C program, this makes it difficult to set up tests for that program or the libraries used by that program. Error management must be carried out using error codes. If a critical error is encountered, the program should not be terminated by calling the # Q`ifunction or the n1tBifunction (C99) in the code where the error was detected. In fact, these two functions do not terminate the program ‡•‰,..•À they bypass the normal termination routines (closing files, deleting temporary files, writing data, ‰)..‡À The error must be traced by means of an error code to the main function K BM, which will then terminate the program.

3&$0..&/%"5*0/±(JWF QSFGFSFODF UIF NBJO GVODUJPO

A C program must have a minimum K BMfunction. Error returns are made by a dedicated (and therefore documented) code return of this function.

36-&±%PO#PQ`VPS1uBFiGW ODUJPOT

The 2tBifunction results in a normal termination of the program and is not dependent on the implementation. This exit from the program can be used, but excessively frequent use of this function in the program can make it difficult to understand.

3&$0..&/%"5*0/2t-BiNUU DBMMT UP

Calls to the 2tBifunction must be commented and not overused. The developer should replace them wherever possible with an error code return in the main function.

Finally, the b 2 i D K functions defined in the b 2 i D K library mainly used for exception handling in C can easily lead to undefined behaviour and should therefore not be used. In particular, their use creates problems with signal management.

36-&

36-&±%PObP2UDVKTBFUOBWEQFM;DmGKVTOUJDVUJPO

#BEFYBNQMF

```
OBM+Hmib2/HB#X?=
OBM+Hmib2/BQX?=
BMi`2 /n7BUpQBV//
&
  6AG17 4 7QT2JM,$$Kv7BXH2i]- ]r]Vc
  B7ULIGG44 7V
  &
    f  T`Q#H2K2MQT2MB7MB;H2f
    n1tBURkVfc MQimi?Q`Bx2i/
  '
  7T`BMU77- ]Wb]- ]#H #H #]Vc
  XXX
   #Q`UVf MQimi?Q`Bx2i/
  XXX
  `2im`Mc
'
BMiK BUpQBV//
&
  BMip H4 `2 /n7BUHV2c
  XXX
  `2im`Rc
'
```

(PPEFYBNQMF

```
OBM+Hmib2/HB#X?=
OBM+Hmib2/BQX?=
BMi`2 /n7BUpQBV//
&
  6AG17 4 7QT2JM,$$Kv7BXH2i]- ]r]Vc
  B7ULIGG44 7V
  &
    f  T`Q#H2K2MQT2MB7MB;H2f
    `2im`Rkcf  2``Q`+Q/2iQ+mK2M7iQ/i?BbT`Q#H2fK
  '
  7T`BMU77- ]Wb]- ]#H #H #]Vc
  XXX
  `2im`Rycf  Qi?2i/Q+mK2M2i2`/Q`+Q/2f
  XXX
  `2im`Mcf  MQT`Q#H2fK
'
BMiK BUpQBV//
&
  BMip H4 `2 /n7BUHV2c
  B7Up H44 yV
  &f  MQT`Q#H2BKMi?227mM+iBQM
    XXX
    `2im`Mc
  '
  2Hb2
  &f  2``Q`?  M/HBM+;+Q`/BMQi?222``Q`+Q/2f
    XXX
    `2im`Rc
```

3FGFSFODFT

[Cert] Rule SIG30-C Call only asynchronous functions with signal handlers.
[Cert] ERR00-C Adopt and implement a consistent and comprehensive error-handling policy.
[Cert] ERR04-C Choose an appropriate termination strategy.
[Cert] ERR06-C Understand termination behavior of assert() and abort().
[IsoSecu] Calling functions in the C Standard Library other than # Q , ` in 1 t B and b B ; M from within a signal handler [asyncsig].
[IsoSecu] Calling b B ; M from interruptible signal handlers [sigcall].
[Cwe] CWE-479 Signal Handler Use of a Non-reentrant Function.

17
Standard library

1SPIJCJUFE TUBOEBSE

Several header files in the standard library only introduce functions that contradict the rules or recommendations of this guide:

- b 2 i D K;T X ?
- b i / ` ; X ?

Therefore, these header files must not be used as they violate several of the above rules.

The library I b i / ` ; X,?For example, introduced in the C90 standard, declares a type and defines 3 macros: p n b i ,`p n ` ;p n 2.MA/new macro (p n + Q3 Tintroduced with C99. The purpose of this library is to allow the definition of functions with a variable number and type of arguments. In addition, the use of these features can, in many cases, lead to undefined behaviour.

Inconsistent typing in the call of a variadic function can lead to an unexpected termination of the function or even undefined behaviour.

36-&

36-& ± %P 0b2LDVKTBX0EF/F`;XT?UBOEBSE M

3FGFSFODFT

[Misra2012] Rule 17.1 The features of <stdarg.h> shall not be used.
[Cert] Rec. DCL10-C Maintain the contract between the writer and caller of variadic functions.
[Cert] Rec. DCL11-C Understand the type issues associated with variadic functions.
[Cert] Rule MSC39-C Do not call va_arg() on a va_list that has an indeterminate value.
[Misra2012] 21.4 The standard header file <setjmp.h> shall not be used.
[Cert] MSC22-C Use the setjmp(), longjmp() facility securely.
[Cert] ERR04-C Choose an appropriate termination strategy.
[Cert] ERR05-C Application independent code should provide error detection without dictating error handling.

/PUSFDPNNFOEFE TUI

Use of the following libraries should be limited and retained only if necessary: 7 H Q j X ? Q K T H, 2 t X ? 7 2 M pXd? K i ? X ?

3&$0..&/%"5*0/±-JNJUUIFVTFPGTU QPJOUOVNCFST

The standard libraries 7 H Q j X ? Q K T H, 27t2XM pXd? K i ? X ? should only be used if absolutely necessary as in the case of digital processing.

3FGFSFODFT

[Misra2012] 21.11 The standard header file <tgmath.h> shall not be used.
[Misra2012] 21.12 The standard header file <fenv.h> shall not be used.
[Cert] FLP32-C Prevent or detect domain and range errors in math functions.
[Cert] FLP03-C Detect and handle floating point errors.
[Cwe] CWE-682 Incorrect calculation.

1SPIJCJUFE TUBOEBS

Other libraries contain dangerous functions such as the i Q B,U VQ H,U VQ 7 and i Q H H-functions of b i / H B #wXi? h lead to undefined behaviour if the resulting value cannot be represented. The b i `i Q fUV functions are to be preferred as they have the same action, but without the risk of undefined behaviour.

36-&±%P OPU VTFTQFBUUF-GVQHBUVP OiTQ BOiEQ H HGUSVP N UbF/MB@XSB S Z

Equivalent functions b i `i Q bV e to be used.

The ` M / bUV nction of the standard library for pseudo-random number generation does not give any guarantee as to the quality of the generated randomness.

36-&±%P ÒPMU/VGTVFOUDIEJPO PG UIF TU

3FGFSFODFT

[Misra2012] The i Q,7 i Q,B i Q and i Q H-functions shall not be used.
[Cert] ERR07-C Prefer functions that support error checking over equivalent functions that don't.
[Cert] Rec MSC25-C Do not use insecure or weak cryptographic algorithms.

[Cert] Rule MSC30-C Do not use the ` M / UVnction for generating pseudorandom numbers.
[Cwe] CWE-327 Use of a Broken or Risky Cryptographic Algorithm.
[Cwe] CWE-338 Use of Cryptographically Weak Pseudo-random Number Generator (PRNG).
[Cwe] CWE-676 Use of potentially dangerous functions.

$IPJDF CFUXFFO EJGG GVODUJPOT

When a function from the standard library offers a “less dangerous” version — in the sense that it adds extra security —, this version should be favored.

8BSOJOH

We refer to “less dangerous” versions because such functions for instance add a limit to the size of an input parameter, but they can still result in undefined or unspecified behaviours.

*OGPSNBUJPO

In later versions of the C language (especially for C11), new and genuinely more secure versions are proposed such as bi`+TvnbUV

String handling functions of the type bi`ttwill be replaced by the equivalent functions bi`Mtt when it is possible to limit the number of characters to be handled.

36-&

36-& ± 6TF UIF ²NPSF TFDVSF³ WFSTJP

When different versions of functions from the standard library exist, the “more secure” version must be used.

Likewise, all obsolete or outdated functions must not be used. The best known example is that of the ;2ibUfunction, deprecated in the third technical patch of C99 [AnsiC99] and which was removed from subsequent standards.

36-&

36-& ± %PPUVTF PCTPMFUF MJCSBSZ TPMFUF JO TVCTFRVFOU TUBOEBSET

36-&

36-& ± %PPUVTF MJCSBSZ GVODUJPOT CVGGFS TJ[F BT BO BSHVNFOU

3FGFSFODFT

[Cert] Rec. PRE09-C Do not replace secure functions with deprecated or obsolescent functions.
[Cert] Rec. MSC24-C Do not use deprecated or obsolescent functions.
[Cwe] CWE-20 Insufficient input validation.
[Cwe] CWE-120 Buffer Copy without checking Size of Input('Classic Buffer Overflow').
[Cwe] CWE-676 Use of potentially dangerous function.
[Cwe] CWE-684 Failure to provide specified functionality.

18
Analysis, evaluation of the code

1SPPGSFBEJOH PG UIF

It is good for any developer, although not mandatory, to have their code read through at least once by a dedicated proofreader or another developer to check the maintainability and clarity of their code.

(00%
13"$5

(00% 13"$5*$&±"MM DPEF TIPVME CF

*OEFOUBUJPO PG MPO

For long expressions, in the absence of adequate indentation, it is very difficult to understand the code and the intention of the developer. The use of space characters for indentation of expressions and statements allows more flexibility for indentation than the use of the tab character.

3&$0

3&$0..&/%"5*0/±*OEFOUBUJPO PG M

When a statement or expression is spread over several lines, indentation is essential in order to facilitate understanding of the code. compréhension du code.

#BE FYBNQMF

The code in the following example should be re-indented in order to make it easier to understand.

```
B7U UP Shn4R4 Q TV
 %% tU U4 V     Uty/2 7#2247 4#V
    ULlGG44 TnTQBVMV
&
  f  T`Q+2bbBfM;
'
B7Uy 54n7mM+iBQMnM K2n`2 HHvniQn2ti2M/niQn#2n bnØtTHB+Bin bnTQbbB#
     n*PLah Lhn Gq uanqAh>n no1_un1sSGA*AhnL J1
     na1*PL.n*PLah Lhn Gq uanqAh>n nL 81m8hyPV1sh1L.
&
  f  T`Q+2bbBfM;
'
```

(PPE FYBNQMF

The following code shows a correct indentation of a conditional over several lines.

```
B 7U P Shn4R4 Q TV
  %%  U U4  V
     Uy/2  7#2247 4 #V
     UL I G G4 4 TnT Q BVMi
  V V
  &
    f   T`Q+2bbBfM;
  '
B 7Uy  54n7mM+iBQMnM  K2n`2  HHvniQn2ti2M/niQn#2n  bnUtTHB+Bin  bnTQbb
       n*PLah-Lm*PLah Lhn Gq uanqAh>n no1_un1sSGA*AhnL J1
       na1*PL.n*PLah Lhn Gq uanqAh>n nL J1nhPn1sh1L.
      8-
      8yV V
  &
    f   T`Q+2bbBfM;
  '
```

*EFOUJGZJOH BOE SFNPWJOH EFBE DPEF

The presence of dead code or unreachable code hinders the proofreading and understanding of the code.

6OSFBDIBCMF DPEF

Code is considered unreachable if there is no input that allows this point of the program to be reached (statements in an always false conditional, statements located after a `2im statement, ‰Ÿ.‡À

%FBE DPEF

“Dead code” is understood to mean code for which the execution has no effect (no modification of variables, no impact on the control flow, ‰Ÿ.‡À

Furthermore, from a security point of view, dead or unreachable code can be used during a bypass of the execution stream. This unreachable code may be debugging code, disabling security checks.

36-&

36-& ± *EFOUJGZ BOE SFNPWF BOZ E

36-&

36-& ± 5IF DPEF NVTU IBWF OP VOSF DPEF BOE JOUFSGBDF DPEF

There must never be any unreachable code, except for defensive code or interface code, and in both cases it must be specified as a comment.

3FGFSFODFT

[Misra2012] Rule 2.1 A project shall not contain ™ ’ – ‰ … ŧ ¢ Œ … † • ‰
[Misra2012] Rule 2.2 There shall be no dead code.
[Cwe

It is often observed that the greater the cyclomatic complexity, the more difficult the computer program is to test and maintain. High cyclomatic complexity indicates a high probability of introducing errors during development or maintenance of the program.

In the event of significant cyclomatic complexity, the code should be reorganised in order to simplify it. This can be done, for example, by writing additional functions.

3&$0

3&$0..&/%"5*0/±-JNJUBUJPOPGDZ

The cyclomatic complexity of a function should be limited as far as possible.

-JNJUJOHUIFMFOHUI

In line with the previous section, each function of a program must have a clear corresponding action. Too often, C functions actually perform several actions/processes at once, which complicates the reading of the code, its updating and maintenance. A function that is too long, in terms of the number of lines of code, is often a sign of a function that is too complex with multiple actions, and which could therefore be split into several sub-functions. In such cases, the function code should be reorganised in order to simplify it and reorganise it into different functions of smaller sizes, associated with precise processing.

3&$0

3&$0..&/%"5*0/±-JNJUBUJPOPGUI UJPO

A function should ideally be associated with a single process and should therefore correspond to a reasonable number of lines of code.

%PPUVTF$LFZXPS

C and C++ are two **different** programming languages, although they have many similarities, and C++ incorporates most of the features of the C language.

A developer may inadvertently use C++ keywords (for example: +H ,bM2,rT`Bp,i2m#,HB+/2H2,i2‰)±within a C code, whether to name a function, variable or something else. However, this hinders proofreading of code, and risks confusing the analysis tools. Moreover, this can hamper maintenance and can cause compilation problems if the compiler also includes C++. A search for these keywords in the sources of a C program can be easily automated. When one of the keywords is found, the name of the variable, type or function must be changed.

Appendix C provides the list of C++ keywords.

36-&

36-&±%POPUVTF$LFZXPSET

No C++ keywords must be used in the source code of a C program.

#BE FYBNQMF

In the code below, the names of the M2 and /2H2 functions should be changed to M2rnTQUBMM2H2i2nTfor example.

```
f  TQBXP i f

ivT2/2Zi`m+i
&
  7HQ ic
  7HQ vc
' TQBMicni

TQBMinM2uVc

pQB/2H2iUTQBMinTVc
```

(PPE FYBNQMF

In the following example, no C++ keywords are used.

```
f  TQBXP i f
ivT2/2Zi`m+i
&
  7HQ ic
  7HQ vc
' TQBMicni
TQBMinM2rnTQUBVc
pQB/2H2i2nTQUBQMBMinTVc
```

19
Miscellaneous

$PNNFOU GPSNBU

Comments accepted according to C90 can only take the form:

```
f  +QKK2M?bi+ M#2Qp2b2p2` HBM2b
```

In C99, the notation of comments on a line is extended with the following format:

```
ff +QKK2MQiM bBM;H2BM2
```

The character sequences f and ff are prohibited in all comments, and the line splicing character $ is prohibited in a comment introduced by ff because it leads to undefined behaviour.

36-&

36-&±1SPIJCJUFE DIBSBDUFS TFRV

The f and ff sequences are prohibited in all comments. And a comment on a line introduced by ff must not contain a line splicing character $

3FGFSFODFT

[Misra2012] Rule 3.1 The characters sequences /* and // shall not be used within a comment.
[Misra2012] Rule 3.2 Line-splicing shall not be used in // comments.

*NQMFNFOUBUJPO PG

Already introduced in subsection 5.3.5, “canaries” provide a protection against some programming errors that could for instance enable control flow hijacking by overwriting a function return address saved on the stack.

If the toolchain does not support automatic insertion of canaries, such a mechanism must be implemented by the developer himself. This can be achieved by passing an additional argument to each critical function and verifying its value at the beginning and at the end of this function, as illustrated in the code sample hereinbelow.

36-&±.BOVBMMZJNQMFNFOUB²DBOQPSUFECZUIFUPPMDIBJO

This mechanism must at least be applied to critical program functions.

When this is not feasible, it is still possible to undertake a thorough analysis of the source code in order to, for example, guarantee that no local arrays are used, to avoid any control flow hijacking due to a stack buffer overflow.

8BSOJOH

Preference should be given to the “automatic” use of canaries by means of the toolchain. Indeed, developing a canary mechanism remains a complicated task, often prone to programming errors or even vulnerabilities.

(PPEFYBNQMF

The keyword pQH iBshd2 to prevent possible optimisations of the compiler for access to the values of the + M àrd + M `v variables. In fact, it is necessary to systematically go and read the + M àrd + M `v values in memory.

```
ivT2/27pQH iBhm2BMijkm7B/ni

OB7/27*hAo h1n* L _A1a
bi iB+BMHBpMQ2Bp2`B7+ MU7`B/ni+ M `-B7B/ni+ M `B_V27&
  mBMi3`2b4 55+U M `B54+ M `B_V2c7

  B7Uy 542bV
  &
    f  +QMi2dbiT2+B7B`+Q+2bbBfM;
  '
'

O2Hb2 f  B7/27*hAo h1n* L _A1a
bi iB+BMHBpMQ2Bp2`B7+ MU7`B/ni+ M `-B7B/ni+ M `B_V27& '
O2M/B7f  B7/27*hAo h1n* L _A1a
pQB7QU7B/ni+ M `B &
  f  +?2+FBQM7i?2+ M `T ` K2i2`i i?22#2;BMMBQM7i;?27mM+iBQM
  p2`B7+ MU+`BM `-B6A.n6Vc
  f  #Q/vQ7i?27mM+iBXQXf
  f  +?2+FBQM7i?2+ M `T ` K2i2`i i?22M/Q7i?27mM+iBQM
  p2`B7+ MU+`BM `-B6A.n6Vc
'
```

"TTFSUJPOTPGEFWFM

Two types of assertions can be distinguished in software:

- assertions for the purpose of development. These are intended to be removed from the software once the qualification phase is over (for example, to check that a pointer parameter is not null);
- assertions designed to check the integrity of the software during execution: these are intended to ensure that the software runs normally and to detect a hardware failure or an attempt to modify it externally (e.g. a fault attack).

A software integrity assertion should not be written using the macro bb2`i. Indeed, this macro is deleted from the code generated on compilation in release mode. Furthermore, these assertions

should only be used for debugging purposes and are in particular not recommended for verification purposes, especially due to initialisations activated in debug mode which will no longer be present outside of debug mode.

It may be that a code checking the integrity of a software program is detected as code unreachable by the compiler or a static analysis tool (in fact, the code can check a set of conditions that cannot occur during normal program execution). The purpose of this code must be clearly documented, and it must be ensured that compiler optimisations do not result in the deletion of this code in the generated binary.

3 6 - &

3 6 - & ± / P E F W F M P Q N F O U B T T F S U J P C

Development assertions must not be present in production.

3 & $ 0

3 & $ 0 . . & / % " 5 * 0 / ± . B O B H F N F O U P G J C F N F S H F O D Z E B U B E F M F U J P O

Integrity assertions should appear in production. If an integrity assertion is triggered, the processing code should result in the emergency deletion of sensitive data.

- B T U M J O F P G B O P O F

The absence of a line break at the end of a non-empty file leads to undefined behaviour according to the C90 and C99 standards.

8 B S O J O H

The vast majority of publishers, particularly in a Linux environment, automatically and invisibly add this line break when closing files.

In addition, all preprocessor directives and comments must be closed.

3 6 - &

3 6 - & ± " M M O P O F N Q U Z ¾ M F T N V T U F C E J S F D U J W F T B O E D P N N F O U T N V T U C

A non-empty file must not end in the middle of a comment or preprocessor directive.

Appendix A
Acronyms

ANSI American National Standards Institute

API Application Programming Interface

ASLR Address space layout randomization

FAM Flexible Array Member

IDE Integrated Development Environment

ISO International Standards Organization

MISRA Motor Industry Software Reliability Association

VLA Variable Length Array

Appendix B
Further information on gcc and Clang options

Information given in this appendix originates from [GccRef] and [ClangRef], for GCC 11 and CLANG 13 respectively.

%F¾OJUJPOPGUIF$M

Option @ b i allows for specifying the version of the C standard — or of the corresponding GNU dialect — used by the compiler. Without this option, GNU dialect of ISO C17 is selected by default.

*OGPSNBUJPO

Option @ M is equivalent to option @ b i / 4 , which is itself equivalent to @ b i / 4 + 3 N and @ b i / 4 B b Q N 3.N N , R N N y

*OGPSNBUJPO

Option @ b i / 4 B b Q N 3 N corresponds to ISO C90 as modified in amendment 1 in 1995.

*OGPSNBUJPO

Option @ b i / 4 B b Q N 3 is equivalent to @ b i / 4 + N N

"EEJUJPOBMXBSOJOH

The following options are neither included in @ q hoH @ q 2 t inor @ q T 2 / Mand Bwere not mentioned in chapter 5, but may nonetheless prove useful [15]:

- @ q # / @ 7 m M + i B Q M @ + b i
- @ q + b i @ H B ; M
- @ q + b i @ (warns whenever a pointer is cast so as to remove — or introduce in an unsafe way — a type qualifier like + Q M b i

15. Only options whose names are deemed not meaningful enough are explained. In any case, the reader is encouraged to refer to GCC and CLANG compilers manuals for more detailed explanations.

- @ q + Q M p 2 ` b r B i Q M (warns for implicit conversions that may alter a value, including conversions between signed and unsigned)
- @ q 7 H Q i @ 2 [m H
- @ q M m H H @ / 2 ` 2 7 2 ` 2 M + 2
- @ q b ? (warns whenever a local variable or type declaration reuses an identifier that is already bound to another variable, parameter or type)
- @ q b i + F @ T ` Q i 2 + i Q ` (warns about functions that are not instrumented with a stack canary)
- @ q b i ` B + i @ T ` Q i Q i v T 2 b
- @ q b r B i + ? @ 2 M m K (warns whenever the controlling expression of a b r B i + statement is of an enumerated type but lacks a + b for one or more of the named constants defined with this type)
- @ q K B b b B M ; @ T ` Q i Q i v T 2 b
- @ q m M / 2 7
- @ q p H

The following options are specific to GCC:

- @ q / m T H B + i 2 / @ # ` M + ? 2 b
- @ q / m T H B + i 2 / @ + Q M /
- @ q 7 Q ` K i @ b B ; M 2 / M 2 b b
- @ q D m K T @ K B b b 2 b @ B M B i
- @ q H Q ; B + (warns about suspicious uses of logical operators)
- @ q M 2 b i 2 / @ 2 (warns about declarations that use the 2 t i 2 ` storage class specifier within a function)
- @ q M Q ` K (warns about any identifier that is not in normalized form)
- @ q Q H / @ b i v H 2 @ / 2 7 B M B i B Q M
- @ q b ? B 7 i @ M 2 ; i B p 2 @ p H m 2
- @ q b ? B 7 i @ Q p 2 ` 7 H Q r 4 k
- @ q b i ` B + i @ Q p 2 (warns about a number of cases where the compiler optimizes based on the assumption that signed overflow does not occur)
- @ q b m ; ; 2 b i @ i i ` B # m (warns for cases where adding a 7 Q ` K attribute may be beneficial)
- @ q b m ; ; 2 b i @ i i ` B # m (warns for cases where adding a K H H Q + attribute may be beneficial)
- @ q b r B i + ? @ / 2 7 (warns whenever a b r B i + statement does not have a default case)
- @ q i ` / B i B Q M H @ + (warns whenever a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype)

- @ q i ` K T Q H B M 2 f (warns if trampolines, mentioned in a footnote in subsection 5.3.5, are generated)
- @ q r ` B i 2 @ b (a d d s t h e + Q M b i qualifier to constant strings so that copying the address of one into a pointer to a non- + Q M b i -qualified type produces a warning; this helps the developer to find at compile-time code that tries to write into a string constant — provided that the + Q M b i keyword has indeed been used in declarations and prototypes, otherwise this warning becomes unhelpfully very noisy)

The following options are specific to CLANG:

- @ q ` ` v @ # Q m M / b @ T Q B M i 2 ` @ ` B i ? K 2 i B +
- @ q b b B ; M (w a r n s w h e n e v e r an integer constant assigned to a variable of an enumerated type does not belong to the range of values defined for this type)
- @ q + b i @ 7 m M + i B Q M @ i v T 2
- @ q + Q K K
- @ q + Q p 2 ` 2 / @ b r B i + ? @ / 2 7 m H i
- @ q / m T H B + i 2 @ 2 M m K
- @ q B / B Q K i B + @ T (w a r n s w h e n e v e r an assignment expression is used as a condition without being enclosed in parentheses)
- @ q H Q Q T @ M H v b B b
- @ q 7 Q ` K i @ M Q M @ B b Q
- @ q 7 Q ` K i @ T 2 / M i B +
- @ q 7 Q ` K i @ i v T 2 @ + Q M 7 m b B Q M
- @ q 7 Q m ` @ + ? ` @ + Q M b i M i b
- @ q B K T H B + B i @ 7 H H i ? ` Q m ; ?
- @ q T Q B M i 2 ` @ ` B i ?
- @ q T ` ; K b
- @ q ` 2 b 2 ` p 2 / @ B / 2 M i B 7 B 2 `
- @ q b ? B 7 i @ b B ; M @ Q p 2 ` 7 H Q r
- @ q b B ; M 2 / @ 2 M m K @ # B i 7 B 2 H /
- @ q b i i B + @ B M @ B M H B M 2
- @ q i m i Q H Q ; B + H @ + Q M b i M i @ B M @ ` M ; 2 @ + Q K T ` 2
- @ q i ? ` 2 / @ b 7 2 i v
- @ q m M ` 2 + ? # H 2 @ + Q / 2
- @ q m M ` 2 + ? # H 2 @ + Q / 2 @ ; ; ` 2 b b B p 2

- @ q m M m b 2 / @ K + ` Q b
- @ q m b 2 / @ # m i @ K ` F 2 / @ m M m b 2 /
- @ q p ` B / B + @ K + ` Q b
- @ q x 2 ` Q @ b @ M m H H @ T Q B M i 2 ` @ + Q M b i M i

*OGPSNBUJPO

With CLANG, option @ q a[illegible] automatically enables option @ q K,Q b i which itself enables many additional warnings. Therefore, the options that correspond to the latter are not listed above.

$ M B @ [illegible]

CLANG features a @ q 2 p 2 ` v i o p B M [16] which enables … of the warnings supported by CLANG without exception.

Using @ q 2 p 2 ` v i may be interesting to discover new warnings supported by the compiler or in case of highest level of requirement on a given code base. It should not be used systematically though, since it might for instance cause trouble for project builds following a tool update.

16. Not to be confused with @ q [illegible] and @ q K.Q b i

Appendix C
C++ reserved words

The following list contains reserved words from C++ that do not belong to the C language. Words suffixed with an asterisk are reserved words added in C++11. Additional semantics have been added to the reserved word delete in C++11 when declaring a class.

alignas *	const_cast	not_eq	this
alignof *	decltype *	nullptr *	throw
and	delete *	operator	
and_eq	dynamic_cast	or	try
asm	explicit	or_eq	typeid
thread_local *	export	override *	typename
bitand	final*	private	using
bitor	friend	protected	virtual
char16_t *	mutable	public	xor
char32_t *	namespace	reinterpret_cast	xor_eq
catch	new	static_assert *	
class	noexcept *	static_cast	
compl		template	
constexpr *			

Appendix D
Operator priority

The order adopted is in descending order of priority. Operators present in the same cell of the table have the same priority level, even if they are located on a different row of the cell.

L. to R. stands for “left-to-right associativity”, and R. to L. stands for “right-to-left associativity”.

Category	Operator	Name	Associativity
Reference	() [] -> .	Function call Access to an element in an array Access to a field of a given address in a structure Access to a field of a structure	L. to R.
Unary	+ - ++ – ! ~ & (cast) sizeof	Identity Opposite Increment Decrement Logic negation Inversion of all bits Pointer referencing Pointer dereferencing Type conversion Size of an object	R. to L.
Arithmetic	* / % + -	Product Division Modulo Sum of two numbers or a pointer and a number Substraction of two numbers or two pointers	L. to R.
Shift	<< >>	Binary shift to the left Binary shift to the right	L. to R.
Comparison	< <= > >= == !=	Strictly less than, less than or equal to Strictly greater than, greater than or equal to Equal à Different from	L. to R.
Bit processing	& ^ |	Bitwise And Bitwise exclusive Or Bitwise Or	L. to R.
Logic	&& ||	Logic And Logic Or	L. to R.

Conditional	?:	Ternary conditional operator	R. to L.
Assignment	= += -= *= /= %= &= ^= |= <<= >>=	Assignment Increment, decrement, product, division then assignment Modulo, logic operation, shift then assignment	R. to L.
Sequence	,	Argument or expression separator	L. to R.

Appendix E
Example of development conventions

At the beginning of an IT project, the development team should always agree on the coding conventions to be applied. The aim is to produce a coherent source code. Furthermore, the right choice of conventions helps to reduce programming errors.

*OGPSNBUJPO

The following points are an **example** of coding conventions. Some choices are arbitrary and open for discussion. This example of conventions can be used or taken as a foundation, if no development convention has been defined for the project to be produced. Different tools or advanced editors are able to automatically implement some of these coding conventions.

Where agreements have been defined in the context of the implementation of a project, the document clearly specifying these conventions must accompany the project in question.

& 'JMFT FODPEJOH

The source files are encoded in UTF8 format.

The line feed character is \$ M("line feed" in Unix format).

& $PEF MBZPVU BOE JOE

& .BYJNVN MFOHUIT

A line of code or comment should not exceed 100 characters.

A line of documentation should not exceed 100 characters.

A file should not exceed 4000 lines (including documentation and comments).

A function should not exceed 500 lines.

& $PEF JOEFOUBUJPO

The code is indented with spaces: one level of indentation corresponds to 4 space characters. The use of the tab character as an indentation character is prohibited.

The declaration of variables and their initialisation must be aligned using indentations.

A space character is systematically left between a keyword and the opening parenthesis that follows it.

The opening brace of a block is placed on a new line. The block closing brace is also placed on a new line.

A space character is left before and after each operator.

A space character is left after a comma.

The semicolon indicating the end of a statement is stuck to the last operand of the statement.

For a function call with many parameters, if it is necessary to place the parameters on several lines, these parameters are indented to be positioned at the opening parenthesis of the function call.

```
(PPE FYBNQMF

XXX
mBMijkniTi`Q+2bbBM;n7nUMB-MBFQ2MnHBbTimT ` KRmBMijkmiBjknT `-Kk
                              +QMbmiMbB;M+2/` bnT ` KVj
&
  mBMijkmiBjkn`2bm4Hiyc
  2H2K2MiniTiTnQminT `4 K9GcG

  B7UUlGG44TnT ` KRR%%LlUGG44bnT ` KVjV
  &
    mBjkn`2bm4Hiyc
    ;QiQ1Mc/
  '

  mBjkn`2bm4H7imM+iBQMnrBi?nK NUTmT ` KRmBjknT `-KbknT ` Kj
      TTnQminT VcK9

  B7UR 44mBjkn`2bVmHi
  &
    XXX
  '

  1M,/
  `2im`mBjkn`2bcmHi
'
```

& 4UBOEBSE UZQFT

If the bi/BMi header is present, it must be included in order to benefit from the integer types that it defines. In its absence, it is necessary to define the integer types as presented in section 7.

If the bi/#QQH header is present, it must be included in order to benefit from the boolean type it defines. In its absence, the #QQType must be defined as presented on the following code (header file from GCC version 4.8.2). The n"QQType is defined for the compilers compatible with standard C99 and subsequent standards.

```
f  *QTv`B;2*iV RNN3@k6y`R2j2aQ7ir `2QmM/ i-BQMX+ f
```

```
f
   A a P* a i  M /  `,/  d X R e" Q Q H 2 i WT 2  M / p  H m 2 b b i / # Q Q ?H
  f

OB 7 M / 2n7a h . " P P G n >
O/ 2 7 B Mn2a h . " P P G n >

OB 7 M / 2n7n + T H m b T H m b

O/ 2 7 B M#2Q Q H      n " Q Q H
O/ 2 7 B Mi 2 m 2      R
O/ 2 7 B M72 H b 2     y

O2 H b 2f  n n + T H m b T H m b

f   a m T T Q ` i B b M i / # Q Q ?H  B M* Y Y B b   :** 2 t i 2 M b B K Q M
O/ 2 7 B Mn2" Q Q H    # Q Q H
O/ 2 7 B M#2Q Q H      # Q Q H
O/ 2 7 B M72 H b 2     7  H b 2
O/ 2 7 B Mi 2 m 2      i ` m 2

O2 M / B f7f  n n + T H m b T H m b

f   a B ; M  i H P  i   H Hi ? 2 / 2 7 B M B i B Q ` M b T ` 2 b 2 X M i f
O/ 2 7 B Mn2n # Q Q H n i ` m 2 n 7  H b 2 n  ` 2 n / 2 7 B M 2 /    R

O2 M / B 7f f  b i / # Q Q ?H
```

& /BNJOH

& -BOHVBHF GPS JNQMFNFOUBUJP

The language used for naming libraries, header files, source files, macros, types, variables and functions must be English. This use of English avoids mixing words in French with the keywords of the C language which are in English within the code. The entire source code produced is thus more coherent.

The language used for documentation and comments should be English from the beginning of development and for all documentation and comments.

& /BNJOH PG TPVSDF ¾MF EJSFDU

The source files must be organised in libraries. In the case of a large library, it is recommended to create a tree structure to organise the source files. The top-level directory must be named with the name of the library. Sub-directories must be named in such a way that they reflect the criteria for grouping source files.

The following example shows the organisation of directories for a library containing utility functions:

Tree structure	Comment
utils	Basic directory of the library
utils/includes	Directory containing all the header files of the library (API)
utils/collection	Directory containing the implementation of all collection type data structures (lists, stack, array, hash table, ‰)˜ ‡ À
utils/concurrency	Directory containing the implementation of mutex, semaphores, conditional variables
utils/threads	Directory containing the implementation of threads
…	…

& /BNJOH PG IFBEFS ¾MFT BOE JN

Header files and source files must be prefixed with the name of the library to which they belong. If the library name is long, it is advisable to use an abbreviation as a prefix. This abbreviation must be chosen in such a way that it does not conflict with an already existing library (standard libraries, third party libraries, ‰).‡ À

The following list gives examples of header file and source file names: m i B H b n H B M F 2, / n H B b i X
m i B H b n H B M F 2, /m H B H b i X K m i m 2 i t B X H ? b n K m i m 2 i t B X H b n i ? ` , 2 m / i X B ? H b n i ? ` 2 . / X +

& /BNJOH PG NBDSPT

Preprocessor macros must have upper case names. The words making up a macro name must be separated by the underscore character. The name of a macro should not match an already existing name of another macro: for example a macro belonging to a header file of a standard library. The parameters of a macro must respect the variable naming convention.

(PPE FYBNQMF

O / 2 7 B M G 2 P : n . 1 " l : U b J 2 b b ; 2 V r ` B i 2 n H Q ; n K 2 b b ; 2 U b J 2 b b ; 2 V

The name of a macro, defined to avoid the multiple inclusion of a header file, uses the name of the header file in upper case. The full stop character is replaced by an underscore character.

(PPE FYBNQMF

O / 2 7 B M l 2 n A G a n G A L E 1 . n G A a h n >

& /BNJOH PG UZQFT

The name of a type defined using the keyword i v T 2 / 2 must be written in lower case, with the suffix n i . The words making up the type name must be separated by the underscore character.

When defining a type for an enumeration or structure, the name following the keyword 2 M m r K b i ` m must have the suffix n i ; The name of the type after the closing brace defining the type must be the same name, with n i replaced by n i .

```
ivT2/22MmbKi imbni &
  XXX
' bi imbnci
ivT2/2BB;M2HQMbBMijkcni
ivT2/2Bi`m+HBMF2/nHBbini ;
&
  XXX
' HBMF2/nHBbini
```

& /BNJOH PG GVODUJPOT

The name of a function must be prefixed by the name (or abbreviation of the name) of the library to which it belongs. The words making up the name of the function must be separated by the underscore character. The name of a function must be written in lower case.

```
bi imbnmiBHbn+`2 i2nHBMHF2BMHF2BinHBbiTmTinHBVbci
bi imbnmiBHbn/2H2i2nHBMUHF2BMHF2BinHBbiTmTinHBVbci
```

& /BNJOH PG WBSJBCMFT

Variable identifiers will consist of words separated by the underscore character, without spaces or upper case letters. Each element of the identifier is used to specify the associated variable (type, sign, size, role, ‰ Ĵ.‡ À

The following table shows the prefixes for the variable names according to type, as well as an example for each type of variable:

Prefix	Variable type	Example
i8	Signed 8-bit integer	int8_t i8_byte = 0;
ui8	Unsigned 8-bit integer	uint8_t ui8_byte = 0U;
i16	Signed 16-bit integer	int16_t i16_option = 0;
ui16	Unsigned 16-bit integer	uint16_t ui16_port = 0U;
i32	Signed 32-bit integer	int32_t i32_value = 0L;
ui32	Unsigned 32-bit integer	uint32_t ui32_counter = 0UL;
i64	Signed 64-bit integer	int64_t i64_big_value = 0LL;
ui64	Unsigned 64-bit integer	uint64_t ui64_big_counter = 0ULL;
b	Boolean	bool b_is_set = false;
c	Character	char c_letter = '\0';
f	Float	float f_value = 0.0f;
d	Double	double d_precised_result = 0.0d;
sz	Type size_t	size_t sz_string_length = 0U;
e	Enumerated type variable	status_t e_status_code = STATUS_ERR;
st	Structure type variable	linked_list_t st_list;
a	Array	uint32_t a_values[10];
p	Pointer type variable	linked_list_t* p_list = NULL;

Prefix	Variable type	Example
pp	Pointer of pointer type variable	linked_list_t** pp_list = NULL;
s	String type variable	char* s_message = NULL;
ws	String type variable in unicode	wchar_t* ws_message = NULL;

& %PDVNFOUBUJPO

& 'PSNBU PG UBHT GPS EPDVNFOU

The source code documentation must be produced using the “œ•‹‰ol’s tag system. “œ•‹‰’ tags must all begin with the ! character. The “œ•‹‰ol also allows the backslash character. However, in order to have uniformity for the documentation of the source code, the at-sign prefix for “œ•‹‰ommands is imposed.

A documentation comment begins with f 5and ends with f.

The following points outline the minimum documentation that must be present in a header file.

& 'JMF IFBEFS UJUMF CMPDL

All header files and all source files must begin with a header title block used to identify:

- the software and / or the library to which the header/source file belongs;
- the company (and if necessary the author) and copyright associated with the file;
- the “œ•‹‰7 B H2. The ! 7 B H2 can optionally be followed by the file name. In the absence of the file name, the file name is automatically deduced from the file in which the ! 7 B H2 is located.

  It is essential to use the ! 7 B H2 **in the header files and the source files.** In its absence the documentation on functions, global variables, type definitions and enumerations present in the file is not included in the “œ•‹‰ f ronoduced t heelo opp_es he ame f he l e sed n ll les tonmend ust e ofiibrary he f b outionall b osed to iame the al F uhe docu

& %PDVNFOUBUJPO PG BO FOVNFS

The definition of an enumeration must be documented with a comment preceding its definition. This comment must indicate in which framework the enumeration is to be used. Each value in the enumeration must be documented.

& %PDVNFOUBUJPO PG B HMPCBM

A global variable must be documented with a comment preceding its definition. This comment must indicate the role of the variable, its initialisation value, and any invariants that must be respected.

& %PDVNFOUBUJPO PG B GVODUJP

The documentation of a function must precede the definition of the function prototype in the header file. The documentation of a function consists of:

- a brief comment;
- a detailed comment explaining the feature offered by the function;
- the presentation of each parameter, specifying whether it is an input, output or both input and output parameter;
- the value returned by the function. In the case of an error code, the success case(s) must be indicated, along with the different error codes that can be returned and their priority;
- a pre-condition, if any, when the function is called;
- a post-condition, if any, after calling the function;
- any additional remarks or warnings.

(PPE FYBNQMF

The following lines show the minimum documentation for a header file.

```
OB 7 M / 2 l 7 nA G an G A L E 1 . n G A a h n >
O/ 2 7 B M l 2 nA G an G A L E 1 . n G A a h n >

f 5
  ! 7 B H 2 l B M F 2 / n X ? B b i
  ! mi ? Q . ` 1 o R

  ! # ` B 2 7 G B M F G 2 / B b i

  6 m M + i B / Q 2 M H ` iB Q 7 M Q b i ? 2 K M B T m H iQ B 7 Q H B M F 2 l / B b X i

  ! //iQ ; ` Q mm Ti B H G B # ` `l v i B H b
  !&
 f

  f 5
    ! # ` B 2 7 1 M m K 2 ` i B Q 7 b M i mb + Q / 2 b

    ai mb + Q / 2 b Q B M / B + i 2 2 b m + + 2 Q b i ? 2 7 B H m Q 2 7 7 m M + i B Q M b
   f
  iv T 2 / 2 7 2 M m b K i mb n i &
    ah hlanal** 14 a x -    ff 5 l b m + + 2 b b
```

```
    ah hlan:1L1_A*n-1_ffF5l ;2M2`B2+``Q`
    ah hlanJ1JP_un-1__fF5l K2KQ`vHHQ+ iB2QMQ`
    ah hlanALo GA.nS ff5J lBMp HB/` K2i2`
' bi imbnni

f 5
  !#`B27lH2K2QM7i?2HBMF2l/Bbi
 f
ivT2/2Ti`m+HBMF2/nHBbin2H2K2Mini ;
&
  bi`m+HBMF2/nHBbin2H2K2TMLi2tni ;   ff5l M2ti2H2K2Mi
  bi`m+HBMF2/nHBbin2H2K2TMSh2p;BQ fff5l T`2pBQ2mHb2K2Mi
  pQB/                          T. ic    ff5l/ i  Q7i?22H2K2Mi
' HBMF2/nHBbin2Hc2K2Mini

f 5
  !#`B27Qm#HH2BMF2l/Bbi

  ai`m+imiQ/27BM2/Qm#HH2BMF2l/BbXih?2ivT2/ i  Q7i?2HBbBbpQBX/
 f
ivT2/2Ti`m+HBMF2/nHBbini ;
&
  HBMF2/nHBbin2H2TK22Mi/nff5l 7B`bi2H2K2Mi
  HBMF2/nHBbin2H2TKl2MBdHnff5l H bi2H2K2Mi
' HBMF2/nHBBbini

f 5
  !#`B27l2rHBMF2l/Bbi

  *`2 iBQQM7  M2rHBMF2l/Bbi#v HHQ+ iBiM2K2KQ`7vQ`i?2bi`m+im`Ml/#v
     BMBiB HBxiB2MH;BbXi
  h?2M2rHBbBb2KTiXv

  !T ` KQm)iTTGBBbibi?2M2rHBbi
  !`2im`MQah hlanal**1a2a+`2 iBQMl/i?2BMBiB HBx iB2Q/MQM2Bi?
    bm++2bb
  !`2im`MQah hlanALo GA.nBS7_TTJGBBbibLlGGQ`
                          B7UTTGBVb54LlGG
  !`2im`MQah hlanJ1JP_un7l_B_HQ_7i?2K2KQ`vHHQ+ iBQM
  !T`2TTGBB5i4LlGGM/UTTGBVb44LlGG
  !MQi2??2+`2 i2/HBb?? biQ#2/2H2i2/
       #v+ HHBKQm;iBHbn/2H2i2nHBMF2/nHBbi
 f
bi imbnniBHbn+`2 i2nHBMUH2BMHF2b/nHBbiTnTiGBVbd

f 5
  !#`B272H2iBQQM7i?2HBbi

   HHi?22H2K2MQl7i?2HBbi`2/2H2i2/M/i?2mb2K2KQ`Bvb7`22X
  !r `MBMh;?2K2KQ`nvb2#vi?2/ i  BMi?2HBbBbMQi7`22XX

  !T ` KBMQm)iTTGBbi?2HBbiQ/2H2iX2
  !`2im`MQah hlanal**1Ba7ai?2/2H2iBQQM7i?2HBbBb  bm++2bb
  !`2im`MQah hlanALo GA.nBS7_TTJGBBbibLlGG
                          Q`B7UTTGBVbBbLlGG
  !T`2TTGBB5i4LlGGM/UTTGBVb54LlGG
  !TQbUTTGBVb44LlGG
 f
bi imbnniBHbn/2H2i2nHBMUH2B/MHF2b/nHBbiTnTiGBVbd

XXX

f 5 !'  f

O2M/Bf7f lhAGanGALE1.nGAahn>
```

Index

List of rules, recommendations and good practices

qyons

Bibliography

[float] ˜ … ’ ˆ … – ˆ Š “ – • “ … ˜ • ’ ‹ Â .“ • ’ ˜ – • ˜ Œ ‘ ‰ ˜ • ‡
Standard, IEEE.

[AnsiC90] Ú ¿ ½ – “ ‹ – … ‘ ‘ • ’ ‹ .¨. ’ ‹ ™ … ‹ ‰ — Â
Standard, International Organization for standardization.

[AnsiC99] Ú ¿ ½ – “ ‹ – … ‘ ‘ • ’ ‹ .¨. ’ ‹ ™ … ‹ ‰ — Â
Standard, International Organization for standardization.

[Cert] “ ˆ • ’ ‹ ˜ … ’ ˆ … – ˆ
Standard, Carnegie Mellon University.

[ClangRef] Ë “ ‡ ™ ‘ ‰ ’ ˜.… ˜ • “ ’
Public documentation, https://clang.llvm.org/docs/.

[Cwe] “ ‘ ‘ “ ’ ‰ … • ’ ‰ — — ’ ™ ‘ ‰ – … ˜ • “ ’
Technical report, MITRE.

[ETALAB] • ‡ ‰ ’ ‡ ‰ “ ™ š ‰ – ˜ ‰ Ú .” ‰ ’ • ‡ ‰ ’ ‡ ‰ š À
Page web, Mission Etalab, avril 2017.
?iiTb,ffrrrX2i H #X;QmpX7`fHB+2M+2@Q.mp2`i2@QT2M@HB+2M+

[GccRef] ¿ ‰ Š ‰ – ‰ ’ ‡ ‰ “ ‡ ™ ‘ ‰ ’ ˜ … ˜ • “ ’
Public documentation, http://www.gnu.org/software/gcc/onlinedocs.

[IsoSecu] Ú ’ Š “ – ‘ … ˜ • “ ’ ‰ ‡ Œ ’ “ • “ ‹ • Â – “ ‹ – … ‘ ‘ • ’ ‹ • … ’ ‹ ™ … ‹ ‰ — ½
… ’ ˆ — • — ˜ ‰ ‘ — “ ¹ › … – ‰ • ’ ˜ ‰ – Š … ‡ ‰ — Â ‰ ‡ ™ – ‰ “ ˆ • ’ ‹ ™ • ‰ —
Technical report, Switzerland, Genève.

[Misra2012] Â ¿ ™ • ˆ ‰ • • ’ ‰ — Š “ – ˜ Œ ‰ ™ — ‰ “ Š ˜ Œ ‰ • … ’ ‹ ™ … ‹ ‰ • ’ ‡ –
Guidelines, https://www.misra.org.uk/MISRAHome/MISRAC2012.

7FSTJPO "/44* 1"

-JDFODF PVWFSUF 0QFO -JDFODF cUBMBC W

AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION

"/44* CPVMFWBSE EF -B 5PVS .BVCPVSH 1"3*4

XXX TTJ HPVW GS DPOTFJM UFDIOJWU HPVW GS

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

Premier ministre

SGDSN