IPA Better Life with IT

IPA/JEITA共催セミナー

ESCR C言語Ver3.0 概要紹介

～ セキュアコーディングへの対応 ～

2018年９月28日

独立行政法人 情報処理推進機構（IPA）
社会基盤センター 調査役
久野　倫義

© Information-technology Promotion Agency, Japan All rights reserved 2018 IPA IT Knowledge center

お話すること

1. ESCRについて
2. ESCRの改訂
   ～セキュアコーディングに向けて～
3. ESCR[C言語版]Ver.3.0の発行
4. ESCRの利用方法
5. 活動状況と今後について

参考:プログラマの研修のため学習教材
～問題～

© Information-technology Promotion Agency, Japan All rights reserved 2018

1. ESCRについて

© Information-technology Promotion Agency, Japan All rights reserved 2018

IPA

ESCRとは

組込みシステムの信頼性向上をミッションとして、2004年に開始されたIPA活動の成果物として作成

目的	：	実用的なコーディング規約の策定・運用の促進
対象言語	：	C言語、C++言語
対象読者	：	規約の作成者、プログラマー、レビューアー

ESCRの特徴

- 品質特性により、ルールをトップダウンに体系化
  - 信頼性、保守性、移植性、効率性 といった品質特性で整理し、理解促進
- すぐに使えるルールのリファレンス
  - 作法に対応し、MISRAや、GNU、参加企業のコーディングルールなどを提示
- ルールの特性を提示
  - 対応する品質特性を保つために重要なルール、など
- 初級者にもわかり易い表現

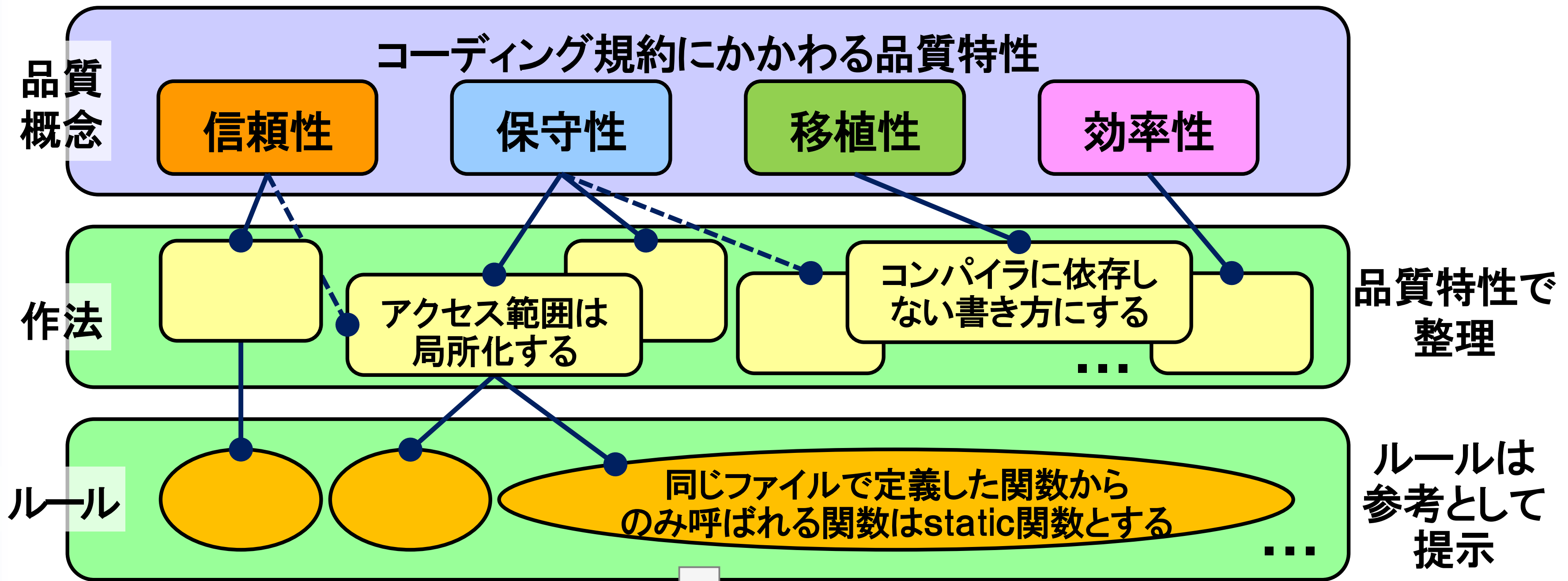
© Information-technology Promotion Agency, Japan All rights reserved 2018

コーディング規約を分かりやすくするために

IPA

■ ソフトウェアの品質特性をもとに、作法とルールを体系化する

作　法　：　品質向上のために守るべき具体的な実装の考え方
ルール　：　言語依存性を考慮した具体的なコーディングの決め事



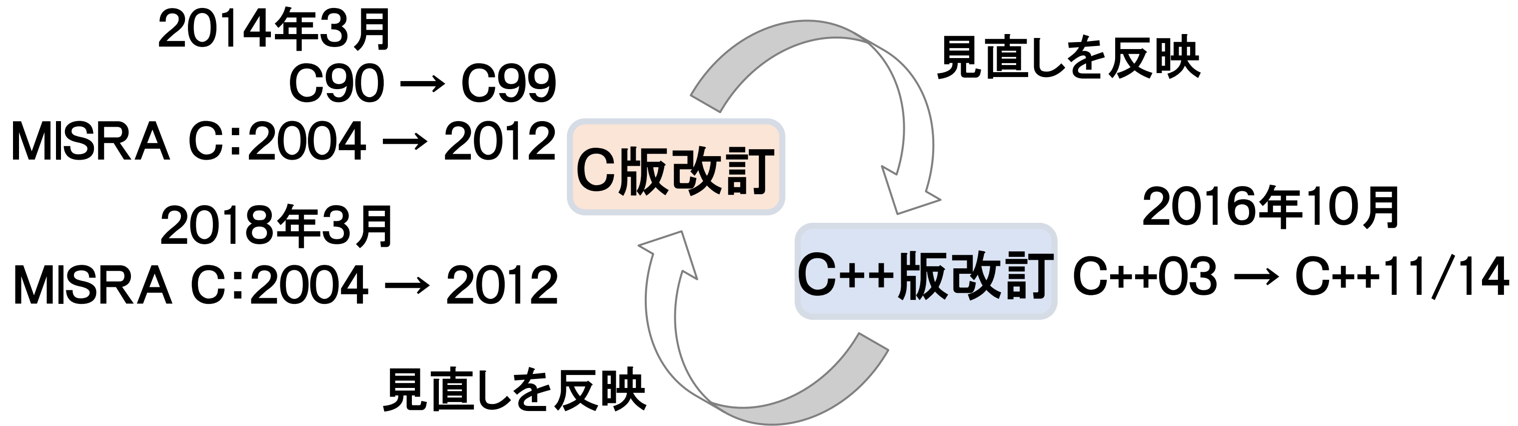
© Information-technology Promotion Agency, Japan All rights reserved 2018

2. ESCRの改訂

© Information-technology Promotion Agency, Japan All rights reserved 2018

ESCRの改訂方針

■ **準拠する言語規格などについての改訂**



■ **セキュリティへの対応（品質規格改訂への対応）**

- JIS X 0129の後継である 25010 でセキュリティが特性に格上げされた
  → セキュリティは基本的には設計段階の特性と考え、
  セキュリティ特性は採用せず、バッファオーバーフローなど セキュリティに影響するコーディングの説明を追加した。
- セキュリティに配慮したコーディングの重要性の高まり
  → 脆弱性との対応付け、CERT Cコーディングスタンダードとの対応付けを明確化する

© Information-technology Promotion Agency, Japan All rights reserved 2018

ESCRでのセキュリティ対応の強化

IPA

- 現状のESCRのルールにはセキュリティの観点から重要なものが含まれる
  - それらとCERT C コーディングスタンダードのルールとの対応付けを行い、
    コーディング規約の作成段階からセキュリティを念頭に置くことの重要性を示す

CERT C ルール

EXP34-C	nullポインタを参照しない
INT33-C	除算および剰余演算がゼロ除算エラーを引き起こさないことを保証する

ESCR ルール

R3. 2. 2
ポインタは、ナルポインタでないことを確認してからポインタの指す先を参照する

R3. 2. 1
除算や剰余算の右辺式は、0 でないことを確認してから演算を行う

✓ **攻撃可能な脆弱性を作り込まない**

✓ **コードの保守性、移植性の向上**

※ CERT C コーディングスタンダード
脆弱性に繋がる恐れのあるコーディング作法や未定義の動作を極力減らすことを目的にまとめられたコーディング規約。C言語を使ってセキュアコーディングを行うためのルールとレコメンデーションを定めたもの　https://www.jpcert.or.jp/sc-rules/

© Information-technology Promotion Agency, Japan All rights reserved 2018

IPA

3．ESCR［C言語版］Ver.3.0の発行

© Information-technology Promotion Agency, Japan All rights reserved 2018

ESCR［C言語版］の改訂 ～ Ver.3.0へ

IPA

■ 今回の改訂のポイント
CERT Cルールの本編への取り込みの検討、
IPAセキュリティセンター提案ルールの取込み、の検討
→ 54のルールや作法をリストアップして検討

- セキュリティの考慮が不足したコーディングに関する注意点
  - バッファオーバーフローによる脆弱性
  - 整数オーバーフローによる脆弱性
  - ディレクトリトラバーサルによる脆弱性
  - パスワード処理などによる脆弱性
  - 同期処理での脆弱性

  ・・・

© Information-technology Promotion Agency, Japan All rights reserved 2018

IPA

■ CERT Cルール(48項目)の本編への取り込みの検討、IPAセキュリティセンター提案ルール(6項目)の取込み、の検討
→ 54項目のルールや作法をリストアップして検討

- 対象としたCERT Cルールの例 (48項目)

ルール番号	内　容
PRE11-C	単一文からなるマクロ定義をセミコロンで終端しない
DCL05-C	typedef を使いコードの可読性を改善する
EXP09-C	型や変数のサイズは sizeof を使って求める
EXP13-C	関係演算子および等価演算子は、結合則が成り立たないものとして扱う
INT04-C	信頼できない入力源から取得した整数値は制限する
ARR01-C	配列のサイズを求めるときに sizeof 演算子をポインタに適用しない
STR08-C	新たに開発する文字列処理するコードには managed string を使用する
STR31-C	文字データと null 終端文字を格納するために十分な領域を確保する
MEM01-C	free() した直後のポインタには新しい値を代入する
・・・	

© Information-technology Promotion Agency, Japan All rights reserved 2018

IPA

● IPAセキュリティセンター提案ルール（6項目）

項番	内　容
1	ポインタ変数に対してsizeof 関数を使用しない
2	ジャンクションを考慮し、無限ループにならないような処理にする
3	【作法詳細】入力値にプログラム上の特殊な意味を持つ文字が含まれる可能性を考慮する 【ルール】　入力値が特殊な意味として解釈されない関数を利用する
4	【作法詳細】ファイルやディレクトリに関する操作ではアクセス範囲を意識した書き方にする。 【ルール】　ディレクトリの指定は、それ以外のディレクトリにアクセスされないような記述をする
5	【作法詳細】メモリ管理は適切に行う 【ルール】　確保したメモリは確実に解放する
6	【作法詳細】一連の処理を複数のプロセスで実施しない 【ルール】　共有リソースに対して処理を行う際には排他制御を行う

© Information-technology Promotion Agency, Japan All rights reserved 2018

新ルールの例(1)

IPA

■ 【作法】共有リソースに対して処理を行う際には排他制御を行う。

R3.11.1	並行処理ではvolatileを同期プリミティブとして使用しない。

並行処理や非同期的なシグナル処理では、データの更新結果を他のスレッドに適切に反映する必要がある。 他のスレッドによる更新の不可分性を保証するための目的でvolatileが利用されていることがあるがこれは誤りである。

・・・

なお、同期用プリミティブの取得と解放は、同じ翻訳単位内の同一抽象レベルで行うことが望ましい。

例

```
int v = 0;
mtx_t flag; // 排他制御用ミューテックス
 …
mtx_lock(&flag);
v++; // クリティカルセクション内。不可分に処理される
mtx_unlock(&flag);
 …
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

修正の例(1)・・・ 解説の追加

R3.2.2	ポインタは、ナルポインタでないことを確認してからポインタの指すメモリを参照する。

ナルポインタや適正でないメモリを指すポインタを介してメモリにアクセスするとハードウェアトラップやメモリ破壊が発生するため、対策が必要である。

対策の例：

(1)使用済みのポインタにナルポインタを代入する。 ポインタの指す先を参照する前に検査することをルール化することで、メモリの解放後使用や二重解放が回避できる。

(2)近年の組込みOSにはポインタの値の適正を検査するシステムサービスを提供するものがある。 これらのOSを使用する場合は、このシステムサービスを必ず使用する。 ポインタによるメモリアクセスの前にポインタの値が適正であることを確認することで、不当なメモリへのアクセスを回避できる。

© Information-technology Promotion Agency, Japan All rights reserved 2018

修正の例(2)･･･ 解説の追加

IPA

M4.4.1	《ヘッダファイルに記述する内容(宣言、定義など)とその記述順序を規定する。》

･･･ ヘッダファイルには、複数のソースファイルで共通に使用するマクロの定義、構造体 ･･･ を記述する。

例えば、以下の順序で記述する。

(1) ファイルヘッダコメント

･･･

なお、typedef やマクロを利用することで、プログラムを一目でわかりやすくしたり、変更箇所を局所化したりできるが、規律なく利用すると同じ内容の別のマクロが定義されるなど、逆効果にもなる。

プロジェクトで利用するマクロや typedef は、プロジェクト開始時に規定し、一箇所で定義して統一的に利用するとよい。

© Information-technology Promotion Agency, Japan All rights reserved 2018

コラムによる解説

- ESCRのルールにはしないが重要な項目であるため、「コラム」を設けて解説する
  - 設計に近いと考えられる項目、ライブラリに関連する項目 など
    - 文字列ライブラリの使用、
    - 機密情報の取り扱い、
    - 信頼できない可能性のあるデータの取り扱い
      - 整数値
      - 書式文字列
      - ファイル名やパス名
      - ファイルの特定

    に分けて概説する
  - 対応するCERT Cルールを提示する

© Information-technology Promotion Agency, Japan All rights reserved 2018

コラム：セキュリティに配慮したコーディングでの注意事項の概要（その1）

IPA

■ **文字列ライブラリの使用**

ライブラリの多くの文字列操作関数には長さを検査する機能が組み込まれておらず、その使用がバッファオーバーフローの原因となり得る

- 古い規格の文字列処理関数は使用を避けるようにするとともに、文字列操作には境界チェックインターフェースを使用する
- C11では標準文字列処理関数の代替関数を規定
  CERT Cでは領域あふれが生じにくい文字列操作ライブラリの使用を推奨

■ **機密情報の取り扱い**

© Information-technology Promotion Agency, Japan All rights reserved 2018

コラム：セキュリティに配慮したコーディングでの注意事項の概要（その2）

■ 信頼できない可能性のあるデータの取り扱い

- **整数値**
  その上限と下限を特定できるかどうかを確認するようにする（INT04-C）
- **書式文字列**
  printf 関数などの引数として外部から制御可能な書式文字列が与えられると、バッファオーバーフローを招く可能性がある。「%n」は・・・
  - 入力をそのまま書式文字列として処理することのないよう注意する（FIO30-C）
  - 全ての書式文字列関数が、ユーザが制御できない静的な文字列であること、さらに、その関数に適切な数の引数が渡されていることを確認する（FIO47-C）
  - 可能であれば、書式文字列において「%n」をサポートしない関数を使用する（C11では「%n」書式が廃止）
- **ファイル名やパス名**
- **ファイルの特定**

© Information-technology Promotion Agency, Japan All rights reserved 2018

4. ESCRの利用方法

4.1　新規コーディング規約の作成／既存コーディング規約の充実

4.2　プログラマの独習、研修のための学習教材

© Information-technology Promotion Agency, Japan All rights reserved 2018

4.1 新規コーディング規約の作成／既存コーディング規約の充実

IPA

1)新規コーディング規約の作成

Step-1 コーディング規約の作成方針を決定
Step-2 決定した作成方針に沿ってルールを選択
Step-3 ルールのプロジェクト依存部分を定義
Step-4 ルール適用除外の手順を決定

2)既存コーディング規約の充実

・抜けモレの防止
Step-1 既存コーディング規約のカテゴリ分け(信頼性、保守性等)
Step-2 既存コーディング規約への観点の追加(信頼性3.1等)

・ルールの必要性の明確化
ESCRの作法とルールの適合例を参照し、規則の必要性を認識

© Information-technology Promotion Agency, Japan All rights reserved 2018

4.2(1) プログラマの独習のための学習教材

IPA

ESCRを読み、
・信頼性を高くするコーディング方法
・バグを作り込まないようにするコーディング方法
・デバッグ・テストがしやすいようにするコーディング方法
・他人が見て、見やすいようにコーディング方法とその必要性

などを学習できます。

© Information-technology Promotion Agency, Japan All rights reserved 2018

4.2(2) プログラマの研修のため学習教材～問題～

ESCRを使って、テストを作る！
テストを通じて、コーディング作法を周知徹底しよう！

さあ、どこに問題があるでしょうか？

不適合例

```
void func1(char *cp) {
  size_t x;
  x = sizeof(cp);


void func2(int arg[MAX],size_t n) {
   size_t argsize;
   argsize = sizeof(arg);
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

4.2(2) プログラマの研修のため学習教材 ～問題(上級者／レビューアー向け)～

問1：さあ、どこに問題があるでしょうか？
問2：該当する品質概念は何ですか？

不適合例

```
void func1(char *cp) {
  size_t x;
  x = sizeof(cp);



void func2(int arg[MAX],size_t n) {
   size_t argsize;
   argsize = sizeof(arg);
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

4.2(2) プログラマの研修のため学習教材 ～問題(トレーナー向け)～

IPA

問1：さあ、どこに問題があるでしょうか？
問2：該当する品質概念は何ですか？
問3：品質特性、品質副特性(JIS X 25010)と「コード品質」の関係を説明してください。

不適合例

```
void func1(char *cp) {
  size_t x;
  x = sizeof(cp);


void func2(int arg[MAX],size_t n) {
   size_t argsize;
   argsize = sizeof(arg);
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

4.2(2) プログラマの研修のため学習教材～回答(問1)～

不適合例

```
void func1(char *cp) {
  size_t x;
  x = sizeof(cp);    // NG:cpはポインタ型の変数
                     // (引数)


void func2(int arg[MAX],size_t n) {
   size_t argsize;
   argsize = sizeof(arg);
                          // NG:argは配列型の引数
```

MAX*sizeof(int) と等しくならない。
sizeof 演算子は、配列型または関数型として宣言された引数に適用されると、たとえ引数宣言で長さが指定されていても、型調整された(ポインタ)型のサイズを求めるからである。

© Information-technology Promotion Agency, Japan All rights reserved 2018

4.2(2) プログラマの研修のため学習教材～回答(問1)～

適合例

```
void func1(char *cp) {
  size_t x;
  x = sizeof(*cp);   // OK：*cpはポインタ型の変数
                     // でない
  size_t y;
  y = sizeof(int *); // OK：int*はポインタ型の変
                     // 数でない

void func2(int arg[MAX], size_t n) {
   size_t argsize;
argsize = sizeof(arg[0]) * n;
                // OK：arg[0]は配列型の引数でない
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

IPA

- **信頼性3.1に関する品質概念：領域の大きさを意識した書き方にする。**
- **sizeof 演算子**

R3.1.5	(1) sizeof演算子はポインタ型の変数に用いてはならない。 (2) sizeof演算子は配列型の引数に用いてはならない。

sizeof(配列型の変数)は配列のサイズとなることから、sizeof(ポインタ型の変数)もポインタが指す領域のサイズになると勘違いすることがある。実際はポインタのサイズとなる。

・・・

【より詳しく知りたい人への参考文献】
CERT ARR01-C CWE-467 MISRA C:2012 Amendment 12.5

参考となる文献を複数掲載

例

```
void func1(char *cp) {
  size_t x;
  x = sizeof(cp);  // NG cpはポインタ型の変数(引数)
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

4.2(2) プログラマの研修のため学習教材～回答(問3)～

IPA

表１　ソフトウェアの品質特性とコードの品質

品質特性 (JIS X 25010)		品質副特性 (JIS X 25010)		コードの品質
信頼性	明示された時間帯で、明示された条件下に、システム、製品又は構成要素が明示された機能を実行する度合い。	成熟性	通常の運用操作の下で、システム、製品又は構成要素が信頼性に対するニーズに合致している度合い。	使い込んだときのバグの少なさ。
		可用性	使用することを要求されたとき、システム、製品又は構成要素が運用操作可能及びアクセス可能な度合い。	
		障害許容性（耐故障性）	ハードウェア又はソフトウェア障害にもかかわらず、システム、製品又は構成要素が意図したように運用操作できる度合い。	バグやインターフェース違反などに対する許容性。
		回復性	中断時又は故障時に、製品又はシステムが直接的に影響を受けたデータを回復し、システムを希望する状態に復元することができる度合い。	

© Information-technology Promotion Agency, Japan All rights reserved 2018

5.　活動状況と今後について

© Information-technology Promotion Agency, Japan All rights reserved 2018

ESCR策定の沿革

’04 ’05 ’06 ’07 ’08 ’09 ’10 ’11 ’12 ’13 ’14 ’15 ’16 ’17 ’18

▲ コーディング作法ガイド WG活動開始

ESCR C 作成開始

ESCR C Ver.1.0

ESCR C Ver.1.1

ESCR C++ 対応開始

ESCR C++ Ver.1.0

ESCRのJIS化 (JIS X 0180)

ESCR C 改訂開始

ESCR C Ver.2.0

◆ 英語版 pdf

ESCR C Ver.3.0

◆ 英語版 pdf

ESCR C++ 改訂開始

ESCR C++ Ver.2.0

◆ 英語版 pdf

組込みソフトウェア開発向け コーディング作法ガイド [C言語版]

組込みソフトウェア開発向け コーディング作法ガイド [C++言語版]

- 品質と生産性の向上に有用との評価
- 書籍および pdfのダウンロード 3万部以上

© Information-technology Promotion Agency, Japan All rights reserved 2018

ESCRに関する活動状況

IPA

- 日本語版pdf公開　2018年1月
- 英語版pdf公開　2018年3月
- CERT C++ と ESCR C++言語版のルール対応表公開
  2018年3月
- ESCR C言語版 日本語版書籍発行
  2018年6月29日
- 今後について
  MISRA C++ 改版への対応
  - MISRAの改版後に実施予定

  ESCR関連セミナー：18年11月頃予定

© Information-technology Promotion Agency, Japan All rights reserved 2018

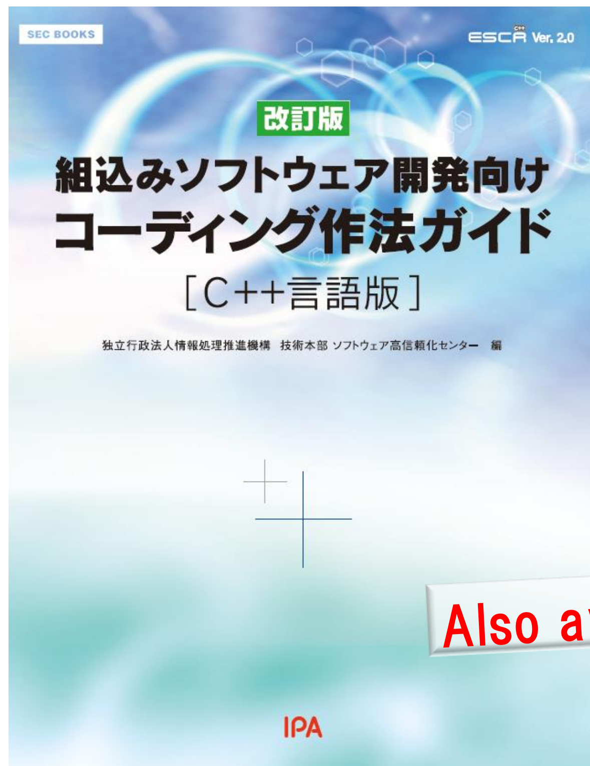
CERTとESCRのルール対応表

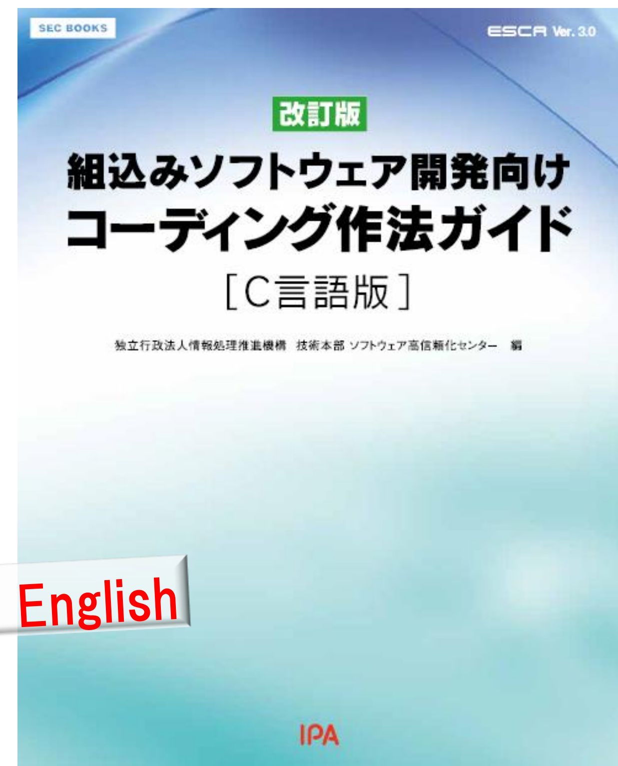
作法詳細		ルール		MISRAルールとの関係			CERT C	CERT C++	CWE
				C:2004	C:2012	C++:2008			
[信頼性 1] R1 領域は初期化し、大きさに気を付けて使用する。									
R1.1	領域は、初期化してから使用する。	R1.1.1	自動変数は宣言時に初期化する。または値を使用する直前に初期値を代入する。	9.1	R9.1	8-5-1	EXP33-C	EXP53-CPP	CWE-119 CWE-456 CWE665
		R1.1.2	const 型変数は、宣言時に初期化する。						CWE-456
R1.2	初期化は過不足無いことが分かるように記述する。	R1.2.1	要素数を指定した配列の初期化では、初期値の数は、指定した要素数と一致させる。				ARR02-C STR11-C STR31-C		CWE-119 CWE-120 CWE-193
		R1.2.2	列挙型(enum型)のメンバの初期化は、定数を全く指定しない、すべて指定する、または最初のメンバだけを指定する、のいずれかとする。	9.3	R8.12	8-5-3	INT09-C		CWE-665
R1.3	ポインタの指す範囲に気を付ける。	R1.3.1	(1)ポインタへの整数の加減算(++、-- も含む)は使用せず、確保した領域への参照・代入には [ ] を用いる配列形式で行う。 (2)ポインタへの整数の加減算(++、-- も含む)は、ポ インタが配列を指 している場合だけとし、結果は配列の範囲内を指すようにする。	17.1 17.4	R18.1 R18.4	5-0-15 5-0-16	ARR30-C ARR37-C ARR39-C	ARR30-C ARR37-C ARR39-C	CWE-119 CWE-122 CWE-129 CWE-468 CWE469 CWE-788 CWE-823
		R1.3.2	ポインタ同士の減算は、同じ配列の要素を指すポインタにだけ使用する。	17.2	R18.2	5-0-17	ARR36-C	ARR36-C	CWE-469
		R1.3.3	ポインタ同士の比較は、同じ配列の要素、または同じ構造体のメンバを指すポインタにだけ使用する。	17.3	R18.3	5-0-18	ARR36-C	ARR36-C	CWE-188 CWE-469
		R1.3.4	restrict 型修飾子は使用しない。		R8.14		EXP43-C		
R1.4	オブジェクトは完全に構築してから使用する。	R1.4.1	コンストラクタでは、すべてのデータメンバを初期化する。初期化の方法は次の通りとする。 (1)初期化は、コンストラクタ初期化子で行う。ただし、クラス型以外の複数のメンバを同じ値で初期化する場合はこの限りではない。 (2)コンストラクタ初期化子では、基底クラス、データメンバをその宣言順に記述する。 (3)コンストラクタ初期化子では、初期化のために他のデータメンバを使用しない。または、他のデータメンバを使用する場合は、そのデータメンバより前に宣言されたデータメンバだけとする。			8-5-1		EXP53-CPP OOP53-CPP	
		R1.4.2	コピーコンストラクタとコピー代入演算子では、非静的データメンバすべてを複写する。						
		R1.4.3	データメンバを読取り参照するメンバ関数の呼出しは、コンストラクタではオブジェクトが完全に初期化されてからとし、デストラクタではオブジェクトの解体が始まる前とする。						
		R1.4.4	コンストラクタとデストラクタでは、仮想関数を呼び出さない。			12-1-1		OOP50-CPP	
		R1.4.5	コンストラクタとコピー代入演算子は、オブジェクト構築の失敗に対応する。			15-1-1 15-3-1		MEM51-CPP MEM52-CPP	CWE-590 CWE-415 CWE-404 CWE-762
		R1.4.6	コンストラクタとデストラクタに記述した catch ハンドラでは、データメンバを参照しない。			15-3-3		ERR53-CPP	
R1.5	オブジェクトの生成・破棄に気を付ける。	R1.5.1	対応する new と delete は同じ形式([ ]付きかどうか)を使用する。					MEM51-CPP	CWE-590 CWE-415 CWE-404 CWE-762

© Information-technology Promotion Agency, Japan All rights reserved 2018

ご清聴ありがとうございました

今後とも ESCR C/C++ のご活用をよろしくお願いします





Also available in English

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考：プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか？（問1）

不適合例

R1.1.1

```
void func() {
   int var1;
   var1++;
    ...
}
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考:プログラマの研修のため学習教材～問題～

さあ、どこに問題があるでしょうか？（問2）

不適合例

R1.1.2

```
const int N;
```

IPA

さあ、どこに問題があるでしょうか？（問3）

不適合例

R1.2.1

```
char var[3] = "abc";
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考：プログラマの研修のため学習教材～問題～

さあ、どこに問題があるでしょうか？（問4）

不適合例

```
enum etag { E1, E2=10, E3, E4=11 };
enum etag  var1;
var1 = E3;

if (var1 == E4)
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考:プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか？（問5）

不適合例

R1.3.1

```
define N 10
char buf[N];
char *p = buf;

...
*(p + 1) = 'a'; // NG
p += 2; // NG

...
*(p + 20) = 'z'; // NG
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

IPA

さあ、どこに問題があるでしょうか？（問6）

不適合例

R1.3.2

```
#define N 10
ptrdiff_t off;

int var1[N];
int var2[N];
int *p1 = &var1[0];
int *p2 = &var2[N-1];



...
off = p2 - p1;
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考:プログラマの研修のため学習教材～問題～

さあ、どこに問題があるでしょうか？（問7）

R1.3.3

不適合例

```
#define N 10
char var1[N];
char var2[N];
char* p = var1;

   ...

if (p < var2+N) { ... }  // NG
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考:プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか？（問8）

不適合例

R1.3.4

```
void f(int n, int * restrict p,
int * restrict q) {
  while (n-- > 0) {
    *p++ = *q++;
  }
}

void g(void) {
  extern int d[100];

  f(50, d+1, d);
}
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考：プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか？（問9）

R2.1.1

不適合例

```
void func(double d1, double d2) {
  if (d1 == d2) {
    ...
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

IPA

さあ、どこに問題があるでしょうか？（問10）

不適合例

R2.1.2

```
void func() {
  double d;
  for (d = 0.0; d < 1.0; d += 0.1) {
     ...
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考:プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか?(問11)

不適合例 R2.1.3

```
struct TAG {
  char c;
  long w;
};
struct TAG var1, var2;
void func() {
  if (memcmp(&var1, &var2, sizeof(var1)) == 0)
{
    ...
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考:プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか？（問12）

不適合例

R2.2.1

```
#define TRUE 1

void func2() {
  if  (func1() == TRUE) {
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

IPA

さあ、どこに問題があるでしょうか？（問13）

不適合例

R2.3.1

```
#define MAX 0xffffU
unsigned int i = MAX;
if (i < MAX + 1)
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考：プログラマの研修のため学習教材～問題～

さあ、どこに問題があるでしょうか？（問14）

不適合例

```
void func(int i1, int i2, long w1) {
  i1 = (i1 > 10) ? i2 : w1;
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考：プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか？（問15）

不適合例

R2.3.3

```
void func(int arg) {
    unsigned char i;
    for (i = 0; i < arg; i++) {
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考:プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか？（問16）

不適合例

R2.4.1

```
int i1, i2;
long w;
double d;
void func() {
   d = i1 / i2;
   w = i1 << i2;
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考:プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか？（問17）

不適合例

R2.4.2

```
int i;
unsigned int ui;
void func( ) {
    i = i / ui;
    if (i < ui) {
      ...
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考：プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか？（問18）

不適合例

R2.5.1

```
short s;
long w;
void func() {
   s = w;
   s = s + 1;
}

unsigned int var1, var2;

var1 = 0x8000;
var2 = 0x8000;
if (var1 + var2 > 0xffff) {
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

IPA

さあ、どこに問題があるでしょうか？（問19）

不適合例

R2.5.2

```
unsigned int ui;
void func() {
   ui = -ui;
```

© Information-technology Promotion Agency, Japan All rights reserved 2018

参考：プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか？（問20）

不適合例

R2.5.3

```
uc = 0x0f;
if((~uc) >= 0x0f)
```

参考：プログラマの研修のため学習教材～問題～

IPA

さあ、どこに問題があるでしょうか？（問21）

不適合例

R3.1.4

```
void func(int  n) {
  int  a[n];
```

© Information-technology Promotion Agency, Japan All rights reserved 2018