SWEST16

第16回
組込みシステム技術に関するサマーワークショップ
2014年8月29 10:30～12:00 下呂温泉 水明館

～自動車業界外の開発者のための～
MISRA-Cガイドライン入門

矢崎総業株式会社　技術研究所　脇田貴文

全体の流れ

- イントロダクション
- 安全のためのガイドライン
- MISRA-Cガイドライン
- サブセット言語という概念
- MISRA-C:2012
  - 日本特有の問題
  - 何がうれしいの？
- 最後に

イントロダクション

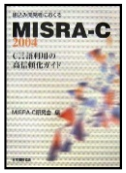
～自己紹介＆主旨説明～

自己紹介

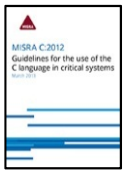
矢崎技術研究所に所属。
主に、制御系ソフト系のＲ＆Ｄを担当。

MISRA-C研究会に所属（2002年～）
下記、日本語解説書や欧州原案のレビューに従事。

組込み開発者におくる MISRA-C 組込みプログラミングの高信頼化ガイド
MISRA-C研究会



組込み開発者におくる MISRA-C 2004 C言語利用の高信頼化ガイド
MISRA-C研究会



MISRA C:2012 Guidelines for the use of the C language in critical systems
欧州MISRA

日本の組込みコミュニティ（皆さんのこと）に育ててもらった。
今日は、出来る範囲で恩返し。

本セミナーの主旨

安全のためのガイドラインについて
多くの開発者に興味をもってもらいたい。

MISRA-Cは、自動車業界を中心に運用され、10年以上にわたり保守されてきた安全なソフトウェアを開発するためのガイドライン。チェックツールも安くなってきたので自動車以外の開発にも利用できるはず。

入門レベルの内容とする。

前提知識なしで聞ける入門レベルの内容とする。
１００以上あるルールを個々に説明するのは困難。
まずは興味を持ってもらいたい。
あとは勉強に必要な書籍、問い合わせ方法を紹介。

最初に、いくつか質問

・企業所属の人 or 大学所属の人？
・技術者、研究者 or 管理者？
・自動車業界？
・すでにMISRA-Cを知っているか？

質問について

・簡単な質問については、その場で割り込みを。

スライドに出てくる用語がわからない。スライドの意味がわからないなど。

・個々のルールについての質問はメールで。

プログラミングの話は、口頭で簡単に説明できないことが多い。

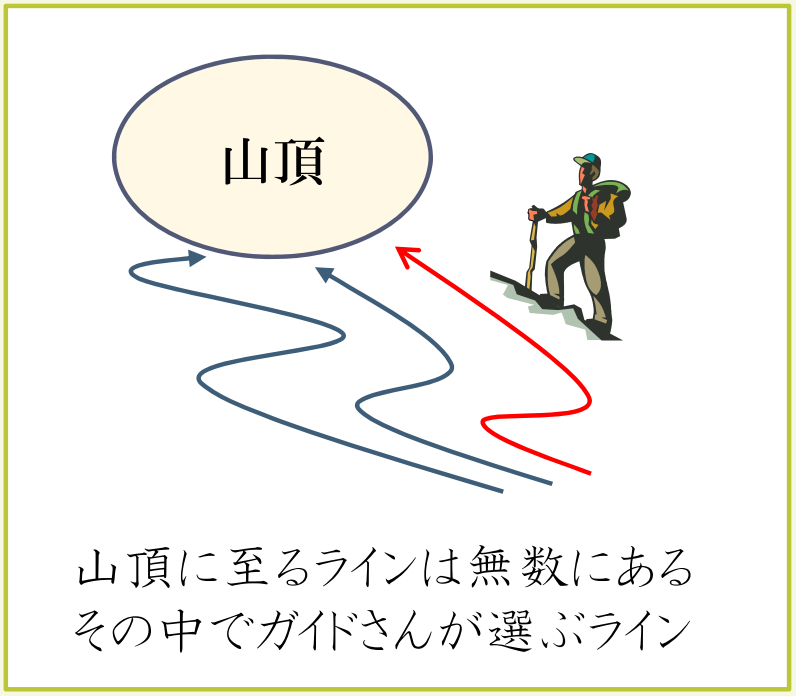
安全のためのガイドラインであるから、安易な回答は避けたい。
内容によっては、MISRA-C研究会のメンバに回覧した上で回答。

安全のためのガイドライン

そもそも・・・ガイドラインとは？

日本語で「行動規範？」

頭のなかに描いてほしいイメージ



山頂に至るラインは無数にある
その中でガイドさんが選ぶライン

C言語は記述の自由度が高い。
いろいろな書き方ができる。

無条件に、盲目に守るものではない
組込みソフトウェア開発は多種多様
であり逸脱が必要となることもある。

正しい理由、正しい手順であれば
逸脱してもかまわない。

欧州発の国際規格・ガイドラインは多い

地図データ ©2014 Basarsoft, Google, ORION-ME

欧州

電気、ガス、水道、鉄道、道路
などのインフラ基盤を隣国とシェア。

国際規格がないと生活ができない。

日本

隣国とインフラ基盤をシェアする必要がない。

国際規格の必要性を感じない。

地図データ ©2014 AutoNavi, Google, Kingway, SK planet, ZENRIN

安全は非競争分野であるべき

たとえば、全ての車に標準装備されている安全技術、２点。

３点式シートベルト

1959年 Volvo社が開発。
特許を公開。

Licensing: GNU FDL 1.2

SRSエアバッグシステム

1980年 Mercedes-Benz社が開発。
特許を公開。

Licensing: GNU FDL 1.2

いずれも欧州の自動車メーカーが開発し公開した技術。

人の命を救うための基本技術は非競争分野であってほしい。

MISRA-Cガイドライン

C言語は
Freedom!

**ルールがない
のは危険です**

source:
http://uguu.org/src_misaka_c.html

解読困難なプログラム

1987年 David Korn 作 IOCCC Best One Liner受賞プログラム

```
main() { printf(&unix["¥021%six¥012¥0"],(unix)["have"]+"fun"-0x60);}
```

ふつうにコンパイルでき、実行すると「unix」と表示される。
でも、これじゃダメ。

じゃあ、なんでダメなの？
それらを明文化したものが
「C言語のコーディングガイドライン」。

※ IOCCC (The International Obfuscated C Code Contest)

MISRA-Cとは

MISRA-Cを言葉で説明すると

安全なC言語のプログラムを書くために開発された
欧州発のガイドライン。始まりは自動車向け。

非競争分野

欧州発であるが、欧州だけで決めているわけではない。

・エディタ　10人
・レビューア　52人（日本から8人）

自動車ソフトの不具合は人の命に係わる。
社会全体の問題でもある。企業が単独で解決できないことは多い。

MISRA-Cのバージョン

三つの版がある。初版は自動車向け。

MISRA-C(1998)	Guidelines For The Use Of The C Language In Vehicle Based Software 自動車用ソフトウェアにおけるC言語利用のガイドライン
MISRA-C:2004	Guidelines for the use of the C language in critical systems クリティカルシステムにおけるC言語利用のガイドライン
MISRA-C:2012	Guidelines for the use of the C language in critical systems クリティカルシステムにおけるC言語利用のガイドライン

クリティカルシステムとは、たとえば、航空、宇宙、船舶、鉄道、プラント、医療機器、金融などの分野

多くのルールが静的解析ツールで適合・非適合の判定ができる。
静的解析は検証コストが低いので、他の分野でも利用できるはず。

MISRA-Cは価値のある技術文書

価値のある技術文書である。

・コーディング規約のコンセプトそのものは、新しいものではない
・1998年の初版から継続的に保守がなされ、現在は第三版
・レビュープロセスを経て発行される
その結果
・きわめて正確な技術文書である
・多くの専門家の知恵が詰まった文書である

比較的オープンなプロセスにて開発。

・世界中から約50人のレビューアが参加して開発された技術文書である。
・ネット上の掲示板でQ&Aができる。
The MISRA Bulletin Board (www.misra.org.uk)

MISRA-Cの勉強方法

日本語解説書を熟読する。わからない用語はＪＩＳ規格で調べる。
基礎なので近道はない。

必要な書籍

	タイトル	発行元
日本語 解説書	組込み開発者におくる MISRA-C:2004 Ｃ言語利用の高信頼化ガイド	日本規格協会
ＪＩＳ 規格	JIS X 3010 - 1993 プログラム言語Ｃ （※開発組織の中に一冊でＯＫ）	日本規格協会

日本語解説書はJIS規格なしでも読めるよう配慮しているが、あるほうが良い

自動車業界は未だにISO-Cの90年度版をベースに品質保証をしている。
190年度版ISO-Cの日本語版は、JIS X 3010:1993。

※古い規格なので入手方法が特殊。
日本規格協会　営業サービスチームへ
コピーサービスを依頼する（2万7千円）

サブセット言語という概念

・簡単に理解できそうなルールをいくつか紹介
・サブセット言語の概念を説明
ベテラン開発者にはあたりまえのことを説明します。

3文字表記

ドイツ語キーボード　(～1985年)

{　　}　　|　　がナイ!?

写真：Copy rights - Hubert Berberich 2012 / Creative Commons Attribution-Share Alike 2.5 Generic license.

そもそも、対応する文字コードがない

16進	米	独	日
40	@	§	@
5B	[	Ä	[
5C	＼	Ö	¥
5D	]	Ü	]
7B	{	ä	{
7C	|	ö	|
7D	}	ü	}
7E	~	ß	‾

ウムラウトを使用するため、文字コードがたりない。

３文字表記とは

Ｃ言語の機能の１つ

コンパイラは、表に示される３文字の並びがソースファイル中に現れると、対応する１文字に置き換える。

もう、3文字表記は必要ないがC言語の機能として残っている。

3文字表記	置き換えられる文字
??=	#
??(	[
??/	¥
??)	]
??'	^
??<	{
??!	|
??>	}
??-	~

3文字表記に関するルール

MISRA-C：2004 Rule 4.2

3文字表記は、用いてはならない

偶発的に右表の3文字の並びが出現してしまうと危険である。

使わない言語機能だからこそ明確なルールで禁止する。

3文字表記	置き換えられる文字
??=	#
??(	[
??/	¥
??)	]
??'	^
??<	{
??!	|
??>	}
??-	~

8進数

偶発的な混乱を引き起こすケース

下記プログラムの危険性を説明できる人、挙手ねがいます。

```
switch (input) {
    case 001:
        mode = "Mode01" ; break;
    case 002:
        mode = "Mode02" ; break;
    case 010:
        mode = "Debug"  ; break;
    case 999:
        mode = "Error"  ; break;
    default:
        mode = "Exception";
        break;
}
printf("mode=%s¥n",mode);
```

もしかしたら、新人がこんなコードを書いてしまうかもしれない。

1960年代のコンピュータ

1960年代のDEC社PDPシリーズは、
18bit、12bitなど、３bitの倍数を採用していた。

18 bit

世界初のUnixマシン

DEC PDP-1(1960)

DEC PDP-7(1965)

12 bit

DEC PDP-8(1965)

3bit、つまり、８進数。

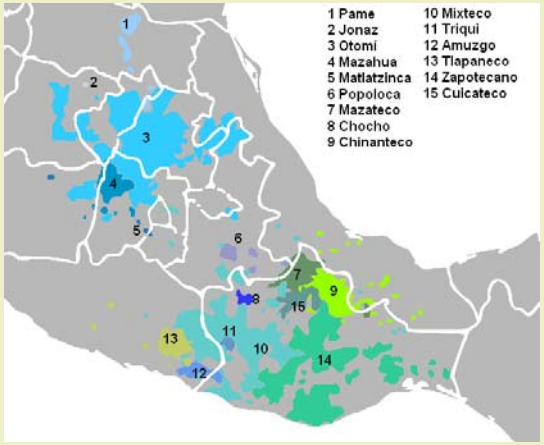
Source:
http://commons.wikimedia.org/wiki/File:PDP-1.jpg
http://commons.wikimedia.org/wiki/File:Pdp7-oslo-2005.jpeg
http://commons.wikimedia.org/wiki/File:PDP_8_e_Trondheim.jpg

じゃあ、今、8進数は必要？

たぶん・・・必要ない。強いて言えば・・・

パメ族対応

例えば、メキシコのパメ族
普段の生活が８進数とのこと



パメ族用の腕時計開発とか・・

ファイル属性

Unixのファイル属性を変更

```
> chmod 600 ~/.rhosts
```

DECの名残り。いまだに８進数

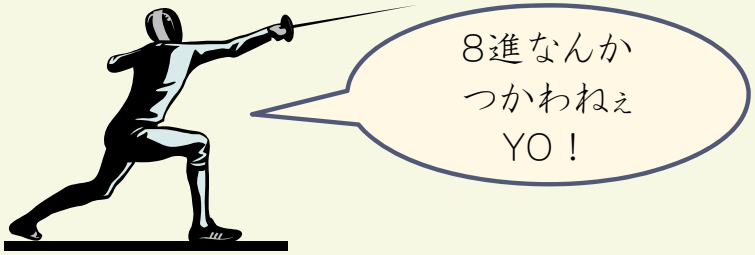
他に、８進数が必要となるケースが思い浮かばない。

【出典】The typology of Pame number system and the limits of Mesoamerica as a linguistic area / HERIBERTO AVELINO

8進数に関するルール

MISRA-C:2004 Rule 7.1（必要）

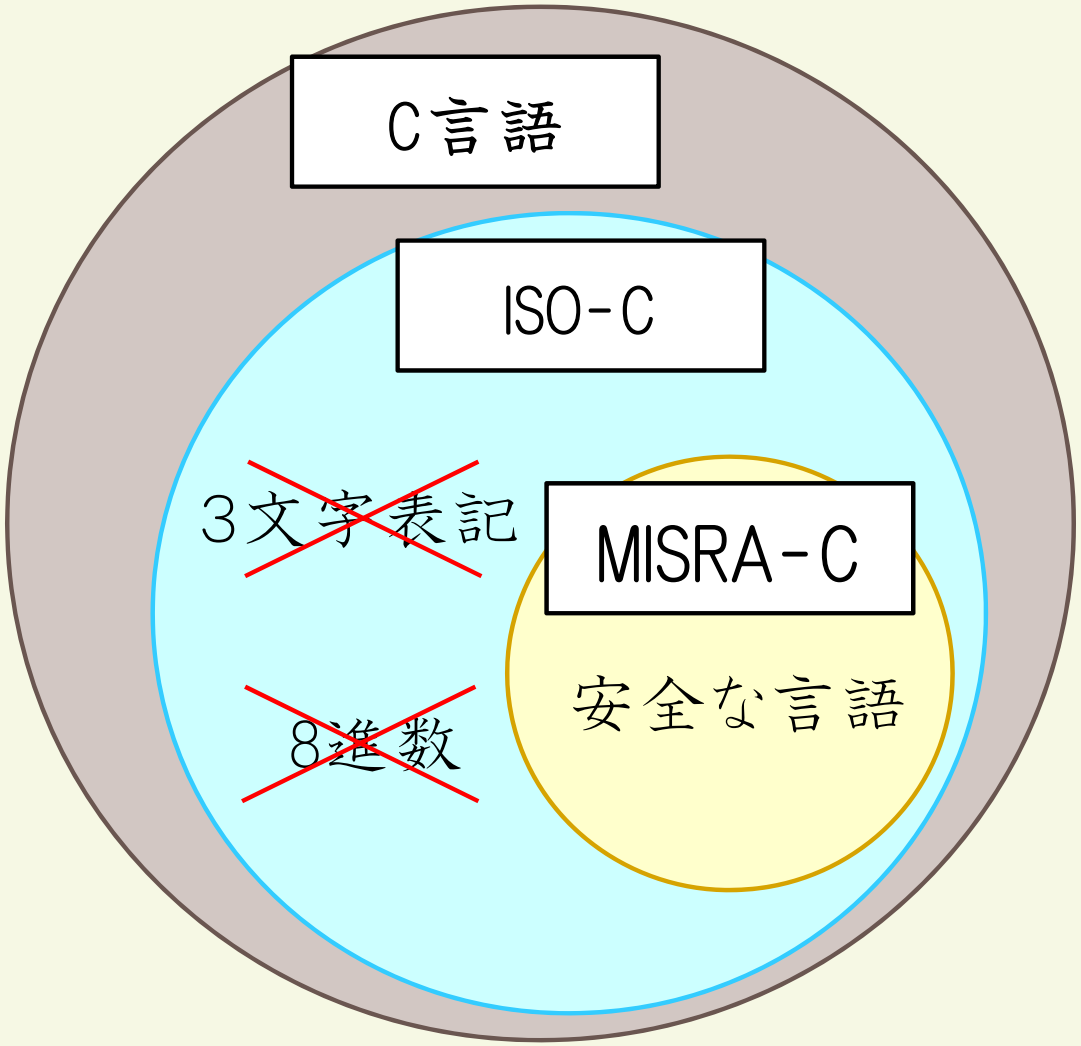
（0以外の）８進定数および８進拡張表記は、
用いてはならない



使わないから、こんなルールいらない・・・・・ではない！

ベテランにとっては、あたりまえでバカバカしいルールかもしれない。

サブセット言語



使わない言語機能は
・偶発的な混乱をもたらす

必要最小限の言語機能にする

それがサブセット言語の考え方。

MISRA-Cの特徴の一つ。

未規定、未定義、処理系定義の回避

C言語規格で明確に定義されていないところは、同じCプログラムであっても、利用するコンパイラによって動作が異なるため、回避するべき。

C90 未定義41

自動記憶域期間をもち、初期化されていないオブジェクトの値を、代入する前に使用している場合

回避策

MISRA-C：2004 Rule 9.1（必要）

すべての自動変数は、用いる前に値を代入しなければならない

強い型付け

C言語は、暗黙の型変換が発生する。知らないと危険。

MISRA-C:2004では潜在型、MISRA-C:2012は本質型という概念を導入。

ISO-C規格	汎整数拡張 通常の算術型変換
MISRA-C:2004	潜在型 (Underlying Type)
MISRA-C:2012	本質型 (Essential Type)

MISRA-Cは、処理系の整数型に依存しないルールを目指している。

ルールからの逸脱

ルールから逸脱してもかまわない。ただし・・・
「正当な理由」と「正当な逸脱プロセス」が必要。

正当な理由の例

【参考】JASO TP 14004 より
R1: システムの性能
R2: サードパーティライブラリとの整合性
R3: 開発環境の制約
R4: 既存コードとの整合性
R5: バリエーション開発
R6: ハードウェアアクセス
R7: 防衛的プログラミング
R8: 適切な言語機能の使用

正当な逸脱プロセス

・文書化
・レビュー
・承認

【引用元】JASOテクニカルペーパー　TP 14004 自動車用C言語ガイドライン(TP01002:2006)の運用レポート

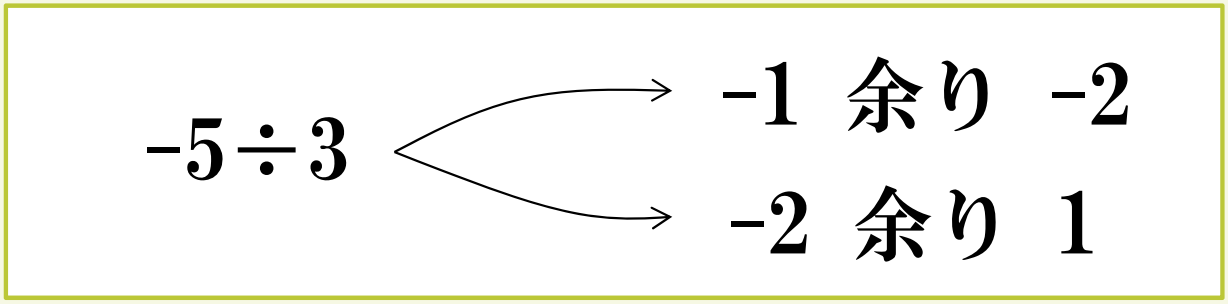
「必要ルール」と「推奨ルール」

推奨ルールは、「正式な逸脱の手続き」が必要ない。
適合or逸脱の判断基準があいまいなルールであることが多い。

MISRA-C：2004 Rule 3.3 （推奨ルール）

選定したコンパイラの整数除算の実装について確認し、
文書化し、十分に配慮すべきである

例えば



整数除算は
処理系依存

【参考】 組込み開発者におくるMISRA-C2004 C言語利用の高信頼化ガイド

入門編はここまで

今日は、本当に「入り口」の部分だけ
・安全のための技術は非競争分野であってほしい
・ガイドラインの必要性
・サブセット言語のほうが安全
ということを理解し、興味をもってもらえれば十分。

個々のルールについての勉強は、MISRA-C研究会の日本語解説書で。

座学で個々のルールを説明すると、みんな寝てしまう。

コンピュータ言語は、サンプルをコンパイルしながら勉強したほうが良い。

いったん休憩？

MISRA-C:2012

MISRA-C:2012は良くできている・・・が！

C99に対応

MISRA-C:2012：一番大きな変更点はC99に対応。

C90というのは、1990年版のISO-C規格のこと

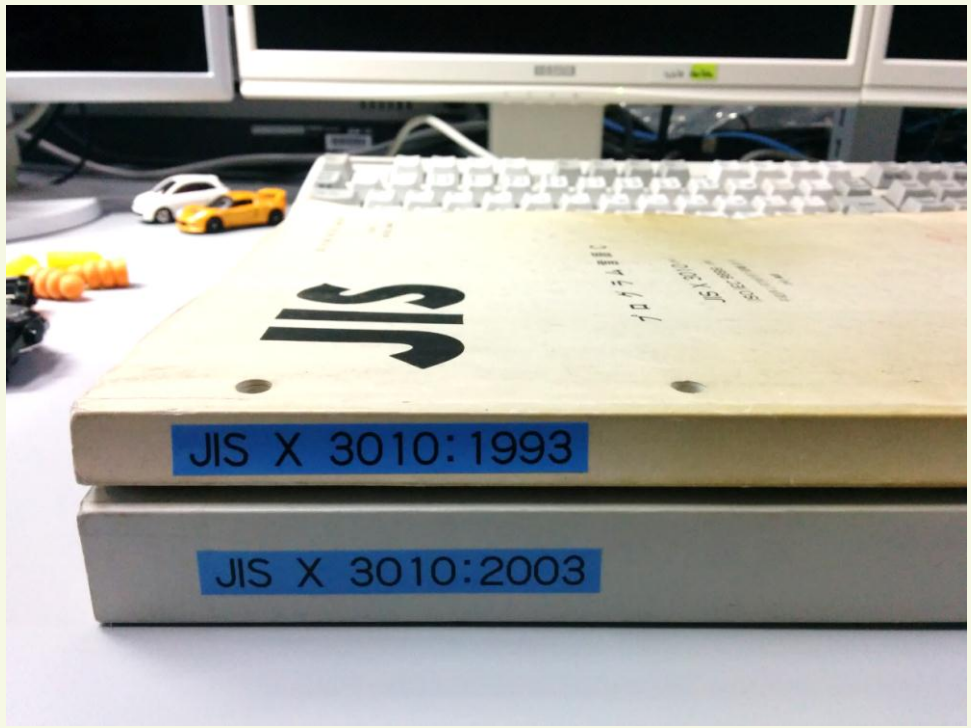
【英語】ISO/IEC 9899:1990
【日本語】JIS X 3010-1993

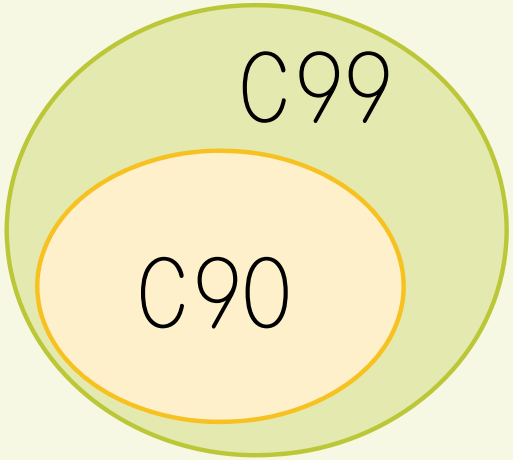
C99というのは、1999年版のISO-C規格のこと

【英語】ISO/IEC 9899:1999
【日本語】JIS X 3010-2003

MISRA-C:2012のジレンマ

C99はC90に対して後方互換性がある。C90は、ほぼC99のサブセット

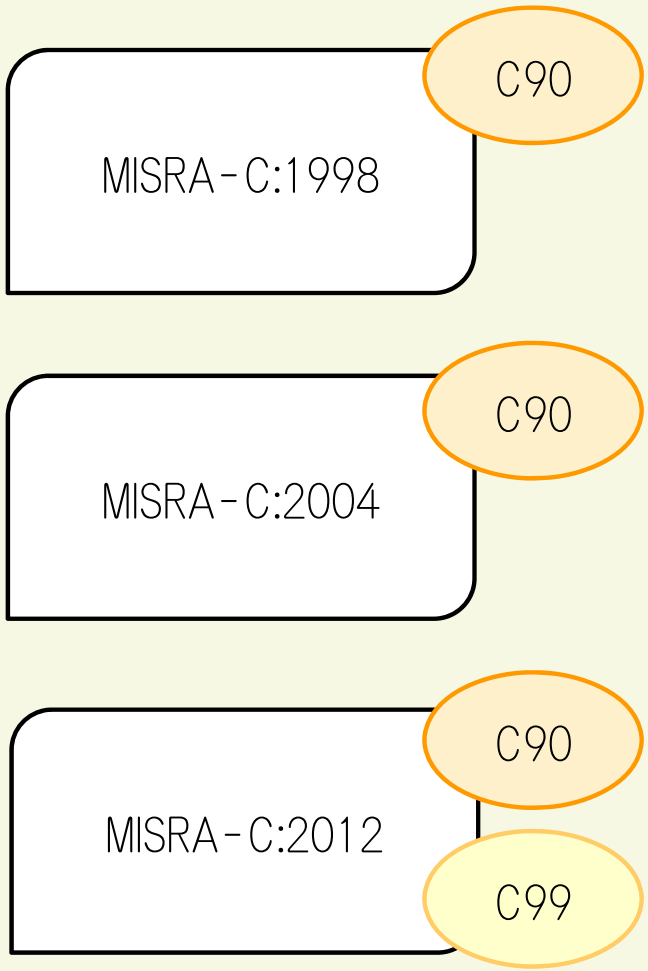




・C90のほうが言語機能が少ない。だから安全？
・C99は言語機能が多い。だからこそガイドラインは重要

悩ましいところ

MISRA-C:2004か2012か



悩ましい判断

保守的な選択

MISRA-C:2004とC90の組み合わせは、シンプルかつ完成度が高い。厳しすぎる、ヒステリックに感じるルールもある。
→　自動車など

合理的な選択

MISRA-C:2012は2004でヒステリックに感じた部分が緩和された。品質を保ちつつ実用的。
C90とC99の場合分けが必要。少しややこしい。
→　自動車業界以外は2012で良いかも

MISRA-C:2012
日本特有の問題

みなさんと、議論したいこと

C99の導入で起こり得る問題。

int あ;

が書けてしまうコンパイラが認められている。

欧州は、当然、この問題を議論していない。

ここは日本人同士でディスカッションして決めるべきこと。

翻訳限界

MISRA-C:2004 ルール 5.1

> （内部及び外部） 識別子は、31文字を超える特徴に依存してはならない。

ルールに逸脱する例

```
/*                 12345678901234567890123456789012345678901    */
extern void communication_data_buffering_process_send(void)
extern void communication_data_buffering_process_recv(void)
```

先頭31文字

識別子があまりに長いとコンパイラの翻訳限界を超える。

では　int あ; は、内部的に何文字として処理されるのでしょうか？

出典：MISRA-C2004 C言語利用の高信頼化ガイド/ MISRA-C研究会

結局、何文字になるのか分からない

必要な場合分け1	各文字が1つの文字コードに対応するとは限らない

が ＝ か ＋ ゛

1文字として扱うか
2文字として扱うか

必要な場合分け2	リンカが拡張文字を受け付けるかどうか

X 3010 : 2003 (ISO/IEC 9899 : 1999)

リンカが拡張文字を受け付けることができないシステムでは、有効な外部識別子を形成するために、国際文字名の表現形式を使用してもよい。

国際文字名の例） **¥U01234567**（10文字分）

独自ルールとして追加すべき?

社内ルール、国内ルールとして、下記のルールを追加するべきでは？

> 識別子に拡張文字および国際文字名を使用してはならない

つまり、識別子に使用してよい文字を、下記の63文字のみに限定する。

```
_ a b c d e f g h i j k l m n o p q r s t u v w x y z
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  0 1 2 3 4 5 6 7 8 9
```

実質、文字種を限定しないとMISRA-Cのルールを守れない。

コメントの中、文字列リテラルの中に拡張文字が出現するのは問題ない。

ここは、もうすこし議論が必要。

MISRA-C:2012
何がうれしいの？

MISRA-C:2012のうれしさ

一番うれしいのは、技術文書としての価値が高くなったこと

ルールの数は2004とあまり変わらないのに厚さが2倍。
ルール解説が充実し、サンプルプログラムも増えた。

英語圏の開発者はMISRA-C:2012のほうが便利のはず。

2004(第二版) → 2012(第三版) の変更点について

細かなルール変更はあるが正常進化と言える範囲。

MISRA-C:2004と比較して、ルールが緩くなった部分が多い。
品質を保ちつつ、以前よりもハードルが下がっている。

日本語の文献が揃うのに、もう少し時間が必要。

正常進化の例①

2004ではgoto文の使用が全面禁止であった。
2012では同じブロック内での後方ジャンプが許されるようになった。

MISRA-C:2004

Rule 14.4（必要）	goto文を用いてはならない

MISRA-C:2012

Rule 15.1（推奨）	goto文は用いるべきではない。（仮訳）
Rule 15.2（必要）	goto文は、同一関数内の後方に宣言されるラベルにジャンプしなければならない。（仮訳）

※関連ルール Rule 15.3, Rule 15.4

正常進化の例②

代入されているのに使用されない変数について、明示的に禁止された。
※代入には副作用が発生する

MISRA-C:2004

Rule 14.2（必要）
空文でない場合には、次のいずれかの条件を満たさなければならない。
a) 実行方法にかかわらず、1つ以上の副作用があること。
b) 制御フローを変更すること。

MISRA-C:2012

Rule 2.2（必要）
デッドコードは、存在してはならない。

【デッドコード】
実行されているが除去されてもプログラムの動作に影響を与えない演算。

正常進化の例③

one entry one exitの原則に関するルールの変遷

MISRA-C (1998) Rule 82 (推奨)

関数は1つの出口しか持ってはならない。

MISRA-C:2004 Rule 14.7 (必要)

関数では、関数の最後に唯一の出口がなくてはならない。

MISRA-C:2012 Rule 15.5 (推奨)

関数では、関数の最後に唯一の出口があるべきである。

組込みシステムは異常系の例外処理が多くなる傾向がある。
引数チェックの結果がエラーとなるような場合は、即returnできるようになった。

MISRA-C研究会と日本語解説書

MISRA-C研究会は、主に自動車部品、開発環境ベンダの集まり
・MISRA-Cの解釈について議論
・日本語版(翻訳レビュー)
・日本語解説書(執筆)
・セミナーなど

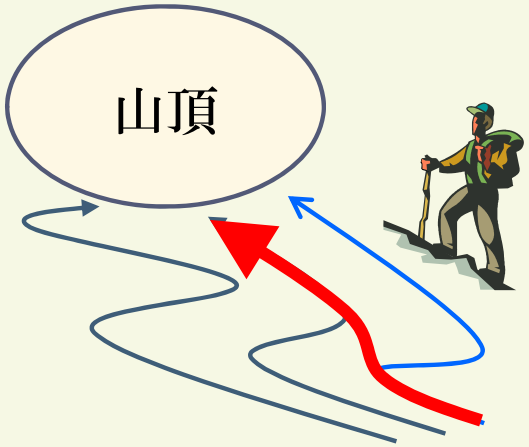
現在
・MISRA-C:2012の翻訳版についてのレビュー作業中
・MISRA-C:2012の日本語解説書について作業中
いつ、どのような形で文書を公開できるかは口頭で説明

今のタイミング(2012の作業中)ではメンバの募集をしてない。
MISRA-C次回改訂の際は、改めてメンバを募集。
世代交代しながら継続させていく必要がある。

最後に

「業界のベストプラクティス」はあなたをトップにするのではなく、単に平凡にするだけだ。

【引用】 ポールグラハム著 「ハッカーと画像」 www.paulgraham.com/icad.html



逸脱することが
最善かもしれない

MISRA-Cガイドラインは
・背景を理解せずに盲目的に守るだけじゃだめ
・逸脱をしても安全と言い切れる理解度がほしい

特に自動車以外の開発者は、全てのルールを盲目的に守る必要などない。

よく勉強し納得した上で、必要なルールを自社の標準に組み込めるとベスト

本セミナーについての問い合わせ

今日は、
・Ｃ言語ガイドラインの必要性
・サブセット言語という概念
が分かってもらえれば十分

個々のルールの理解を座学で勉強するのは難しい。
それはMISRA-C研究会の日本語解説書でじっくりと。

【問い合わせ】
本日のセミナーの内容のほかに
・MISRA-C研究会の日本語解説書
の内容についての質問も可。

【拡散希望】
MISRA-Cは安全のためのガイドライン。幅広い開発で利用してほしい。