SWEST18

【演習で学ぶ！】
C 言語に潜むリスクと、MISRA Cの有効性

MISRA-C研究会

株式会社サニー技研

尾仲　洋和

Sunny Giken Inc.

このコードのリスクは何でしょうか？

```
a = b = 0;
```

解説は、~~番組~~講義の最後に！

イントロダクション（自己紹介）

【自己紹介】

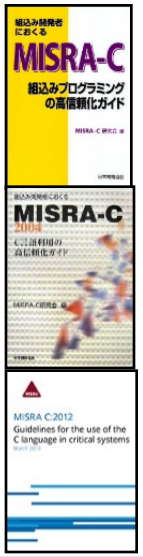
株式会社　サニー技研に所属
半導体工場向け装置の開発などを経て、現在は、車載向けソフトウェア開発を担当。MISRA-C研究会には、第3期から参加。

【MISRA-C研究会とは】

「MISRA C」を正しく理解して、実用的な形でまとめて日本の組み込み業界に紹介するグループ
参加メンバーは、規格の専門家、コンパイラ開発者(言語規格の専門家)、静的開発ツール開発者、組み込みソフト開発者　など

【歴史】

① 第1期（2002-2004）　対象：MISRA-C:1998
　取り纏め：東陽テクニカ、参加23社/28名
② 第2期（2004-2006）　対象：MISRA-C:2004
　取り纏め：東陽テクニカ、参加31社/35名
③ 第3期（2013-2016）　対象：MISRA C:2012
　取り纏め：名古屋市工業研究所、参加16社/18名



推敲中

Sunny Giken Inc.

イントロダクション（目的）

【目的】

本講義では、非常にシンプルなコードを参考にして、
Ｃ言語（プログラミング言語）の持つ**奥の深さ、リスク、おもしろさ**を感じ取って頂き、

どうすれば、**楽しくかつ安全なコーディング**ができるかなどを皆さんが考えるきっかけになればと願っています。

そして、それらの**C言語のリスクをMISRA C**でどのように**防いでいく**かを、実感いただきたいと考えております。

【依頼事項】

問題を出しますので、**正解と思う時に手を挙げて下さい**！

コード事例１　（問題）

下記の範囲は？

A）char

①-128~127　②0~255　③その他

B）short

①-32768~32767　②0~65535　③その他

C）unsigned int

①0~65535　②0~4294967295　③その他

コード事例１　（回答）

A）char　　0 ～ 255 or -128 ～ 127

charは、処理系により符号ありなしが異なる

B）short　　-32768 ～ 32767

C）unsigned int　　0 ～ 65535 or
0 ～ 4294967295

int, unsigned intは、処理系によりサイズが異なる

処理系により範囲が異なる！

コード事例１　（解説）

< MISRA C:2012で検出 >

Directive 4.6(推奨)
基本的な数値型の代わりに、サイズと符合属性を示すtypedefを用いるべきである。

処理系により符号やサイズが異なるchar、intなどを直接使わず、int16型やuint16型といった型定義を用いる事でソースコードの移植性を高める

例）

```
typedef signed char      int8_t;
typedef unsigned char    uint8_t;
typedef signed short     int16_t;
typedef unsigned short   uint16_t;
```

ISO/IEC 9899:1999(以降C99)では、サイズが保障される左記の型が用意されている

コード事例１　（補足）

<第3版(MISRA C:2012)の変更点>

Directive 4.6(推奨)
基本的な数値型の代わりに、サイズと符合属性を示すtypedefを用いるべきである。

MISRA-C:2004まではRuleとして括られていたものが、MISRA C:2012では適合/非適合がソースコードのみで判断できないものをDirective に、判断できるものを Rule に分類するようになりました。

コード事例２（問題）

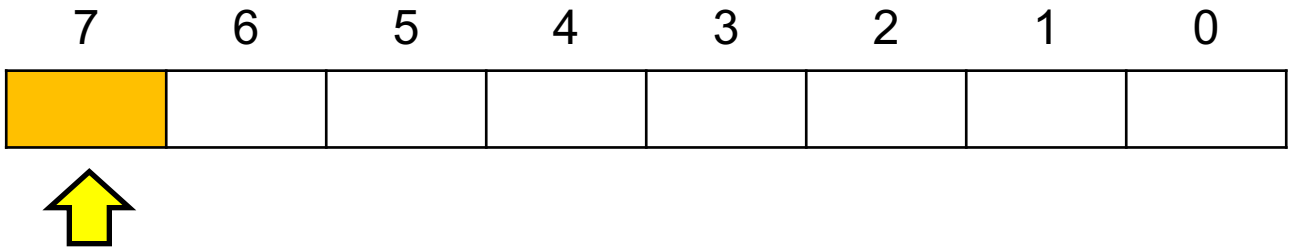
aの値を2進数で表すと？

```
int8_t      a;
a = −18;
```

①10010010　②11101101　③11101110

コード事例２（回答）

int8_t(符号付き1バイト)の場合



左端が、符号ビット(0:正の数、1:負の数)

18から-18を作る手順

①18を2進数に直す	00010010
②ビットを反転	11101101
③1を加える	11101110

2の補数と呼びます

コード事例２（補足）

aの値を2進数で表すと？

```
int8_t      a;
a = −18;
```

①10010010　②11101101　③11101110

絶対値法
符号＋仮数部

1の補数

2の補数

C言語の負数の内部 表現は処理系定義であるため、必ずしも２の補数ではない。

コード事例３（問題）

dの値は？

```
a = 0x55;
b = 0x0f;
c = 1;
d = a & b + c;
```

①0x06、②0x10、③その他

コード事例３（回答）

```
a = 0x55;
b = 0x0f;
c = 1;
d = a & b + c;
```

aとb+cの評価の値を&演算し、0x10となる

&演算子と+演算子では、+演算子の方が優先順位が高いためb+cが先に処理され、評価の値は、0x10

よって、dは0x10

コード事例３（解説）

< MISRA C:2012で検出 >

Rule １２.１
式中における演算子の優先順位は、明示的にすべきである。

演算子の優先順位を明確にすることにより，設計者の意図が誤解されるリスクを軽減します。

例）

```
d = a & (b + c);
```

コード事例３（補足）

説明	演算子あるいはオペランド	優先順位
プライマリ	識別子，定数 文字列リテラル，(式)	16（高）
後置	[]，() (関数呼出し)，.，->，++ (後置インクリメント)，-- (後置デクリメント)，() {}　（C99：複合リテラル）	15
単項	++ (前置インクリメント)，-- (前置デクリメント)，&，*，+，-，~，!，sizeof，defined (前処理系)	14
キャスト	()	13
乗除	*，/，%	12
加減	+，-	11
ビット単位のシフト	<<，>>	10
関係	<，>，<=，>=	9
等価	==，!=	8
ビット単位のAND	&	7
ビット単位の排他OR	^	6
ビット単位のOR	|	5
論理AND	&&	4
論理OR	||	3
条件	?:	2
代入	=，*=，/=，%=，+=，-=，<<=，>>=，&=，^=，|=	1
カンマ	,	0（低）

コード事例４（問題）

a、b、cのそれぞれの値は？

```
a = 0;
b = 0;
c = a +++ b;
```

①aは0、bは1、cは1
②aは1、bは0、cは0
③aは1、bは0、cは1
④その他

コード事例４（回答）

```
a = 0;
b = 0;
c = a +++ b;
```

演算子の優先順位より、
(a++) + bとなる

a++は、後置インクリメントのため、c = a + bの計算には、インクリメント前のaの値が使われる

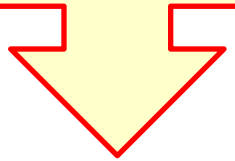
よって、aは1、bは0、 cは0

コード事例４（解説）

< MISRA C:2012で検出 >

Rule 13.3
インクリメント(++)またはデクリメント(--)演算子を含む完全式には、インクリメント演算子またはデクリメント演算子に起因する副作用以外の潜在的副作用があってはならない。

インクリメントまたはデクリメント演算子を含む式は、オブジェクトの変更という副作用を生じる。そのため、他の副作用と混在させると式が分かりにくくなるため、インクリメントまたはデクリメント演算子を他の演算子と切り離して使用すると、式が分かりやすくなります。



例）

```
c = a + b;
a++;
```

完全式

他の式の一部でもなく宣言子の一部でもない式

副作用

実行環境に変化を生じる次の内容を副作用という。
- ◆ volatile オブジェクトへのアクセス
- ◆ オブジェクトの変更
- ◆ ファイルの変更
- ◆ これらいずれかの操作を行う関数の呼出し

コード事例５（問題）

yの値は？

```
x = 0;
y = x/++x;
```

①0　②1　③その他

コード事例５（回答）

```
x = 0;
y = x/++x;
```

分子を先に評価すれば、
０/１で「yは０」となる

分母を先に評価すれば、
１/１で「yは１」となる

yの値は、不定

２つのオペランドの評価順序は
C言語規格で 未規定

コード事例５（解説）

< MISRA C:2012で検出 >

Rule 13.2
式の値と持続的な副作用は、すべての許可された評価順序の元で同じでなければならない。

Rule 13.3
インクリメント(++)またはデクリメント(--)演算子を含む完全式には、インクリメント演算子またはデクリメント演算子に起因する副作用以外の潜在的副作用があってはならない。

コード事例６（問題）

xの値は？

```
x = 1;
if(x = 0)
{
        x = 2;
}
```

①0　②1　③2　④その他

コード事例６（回答）

```
x = 1;
if(x = 0)
{
        x = 2;
}
```

“==”と
”=“の
誤りかも?

よって、x は0

コード事例６（解説）

< MISRA C:2012で検出 >

Rule 13.4
代入演算子の結果は、用いるべきではない。

Rule 13.4が、非適合となるその他の例

例）

```
x = 0;
a[ x ] = a[ x = 1 ];
```

左辺のa[x]のxの値が、０、1のどちらになるかは、未規定のため、「Rule 13.2 式の値と持続的な副作用は、すべての許可された評価順序の元で同じでなければならない。」も非適合となる。

コード事例７（問題）

cの値は？

```
uint16_t a, b;      /* 2byte */
uint32_t c;         /* 4byte */
```

```
a = 0xffff;
b = 0x0002;
c = a + b;
```

①0x00000001　②0x00010001　③その他

コード事例７（回答）

```
uint16_t    a, b;       /* 2byte */
uint32_t    c;          /* 4byte */
```

a = 0xffff;

b = 0x0002;

intｻｲｽﾞが16bitなら”Yes”
intｻｲｽﾞが32bitなら”No”

uint32_tで代入

値を入れ

0x00000001

uint16_tで加算

0xffff + 0x0002 → 0x0001

Cは、0x00000001 というのは誤り

コード事例７（回答）

intサイズが、32bit(32bitマイコン)の場合

```
c = a + b;
```

どういう型（サイズ）で演算されるか？

左辺がuint32_tなので
uint32_tで代入

両方ともにuint16_tであるがint32_tで加算
（汎整数拡張によりintｻｲｽﾞになる）

値を入れてみると

0x00010001

0x0000ffff + 0x00000002
→ 0x00010001

intサイズが16bit時は、0x00000001
intサイズが32bit時は、0x00010001

コード事例７（解説）

< MISRA C:2012で検出 >

Rule 10.6
より大きな本質型を持つオブジェクトに複合式の値を代入してはならない。

リスクを回避するなら

例）ラップアラウンドさせたい場合

```
c = (uint16_t)(a + b);
```

例）ラップアラウンドさせたくない場合

```
c = (uint32_t)a + b;
```

コード事例８（問題）

バイトオーダーがリトルエンディアンの場合、どのアドレスにどの値が書き込まれるか？

```
uint8_t *p;
p = (uint8_t *)0x401;
*(uint16_t *)p = 0x1234;
```

Address	Value
400h	?
401h	?
402h	?
403h	?

①401h番地が0x12、402h番地が0x34
②401h番地が0x34、402h番地が0x12
③その他

コード事例８（回答）

```
uint8_t *p;
p = (uint8_t *)0x401;
*(uint16_t *)p = 0x1234;
```

Address	Value
400h	?
401h	0x34
402h	0x12
403h	?

リトルエンディアンの場合
401h番地に0x34、402h番地に0x12となる

エンディアン

エンディアンとは、多バイトデータをメモリ上に配置する方式のことであり、C標準規格では処理系定義となります。
エンディアンには主にリトルエンディアンとビッグエンディアンの2種類があり、それぞれデータをメモリに配置する方向に違いがあります。

```
*(uint16_t *)p = 0x1234;
```

リトルエンディアンの場合

Address	Value
400h	?
401h	0x34
402h	0x12
403h	?

ビッグエンディアンの場合

Address	Value
400h	?
401h	0x12
402h	0x34
403h	?

コード事例８（回答）

```
uint8_t *p;
p = (uint8_t *)0x401;
*(uint16_t *)p = 0x1234;
```

Address	Value
400h	0x34
401h	0x12
402h	?
403h	?

マイコンによっては、奇数アドレスに配置された1バイト境界線のポインタを、2バイト境界線のポインタにキャストした場合、意図通りにアクセスできない可能性がある

よって、マイコンによっては、意図した場所に配置されない場合がある

アライメント

アライメントとは、データ境界とも呼ばれるメモリへのデータアクセスの単位であり、C標準規格では処理系定義となる。一般的には２ⁿバイトの汎用レジスタやデータバスのサイズと同じサイズとなります。またこれをアクセスするサイズに合わせ、「バイト境界」、「ワード（２バイト）境界」などと呼びます。

①2バイトに合わせたアドレスに配置された変数へのアクセス

Address	Value
400h	0x34
401h	0x12
402h	0xF0
403h	0xDE

アクセスは1度

0x1234

②2バイトにあっていないアドレスに配置された変数へのアクセス

Address	Value
400h	0x56
401h	0x34
402h	0x12
403h	0xF0

アクセスは1度
各シフト後に論理和

0x3456 → 右シフト → 0x0034

+

0xF012 ← 左シフト ← 0x1200

論理和 ↓

0x1234

コード事例８（解説）

< MISRA C:2012で検出 >

Rule 11.3
オブジェクト型へのポインタと異なるオブジェクト型へのポインタとの間でキャストを行ってはならない。

リスクを回避するなら

例）

```
*p = 0x34;
*(p+1) = 0x12;
```

コード事例９（問題）

このコードのリスクは何でしょうか？

```
a = b = 0;
```

コード事例９（回答）

```
uint8_t a, b;
a = b = 0;
```

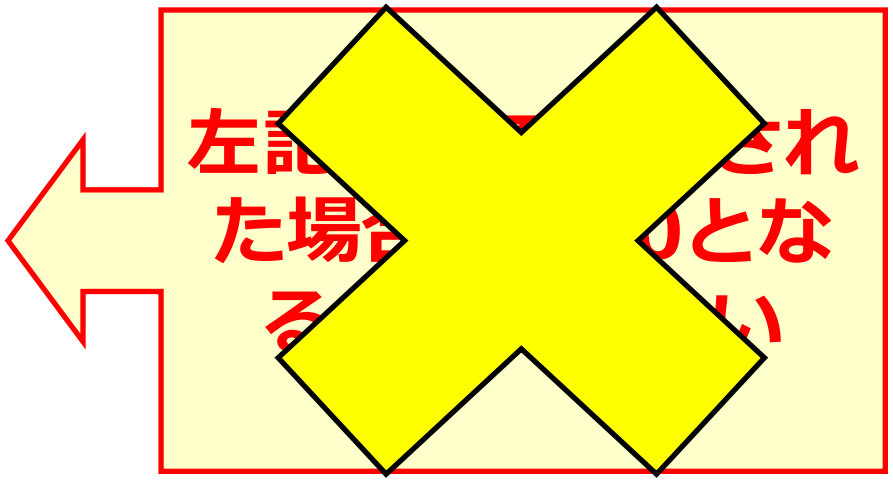
```
a = 0;
b = 0;
```

a、ｂそれぞれに0を代入する処理となり、リスクなし

コード事例９（回答）

```
volatile uint8_t a, b;
a = b = 0;
```

```
b = 0;
a = b;
```



コード事例９（解説）

JIS(C90) 6.3.16 より

代入演算子は、左オペランドで指し示されるオブジェクトに値を格納する。代入式は、代入後の左オペランドの値をもつが、左辺値ではない。代入式の型は、左オペランドの型とする。ただし、左オペランドの型が修飾型である場合を除く。この場合、**その型は、左オペランドの型の非修飾版とする。**左オペランドに格納されている値を更新する副作
用は、前の副作用完了点から次の副作用完了点までの間に起こらなければならない。

赤字の箇所の規定で連続代入中はvolatileではなくなります。

コード事例９（解説）

```
volatile uint8_t a, b;
a = b = 0;
```

代入処理の順序は？

```
b = 0;
a = 0;
```

or

```
a = 0;
b = 0;
```

リスクは？

- Volatileの場合、代入の順序が影響を与える可能性がある。
- 実装者は、b = 0、a = bを期待していた可能性がある。

コード事例９（解説）

< MISRA C:2012で検出 >

Rule 13.4
代入演算子の結果は、用いるべきではない。

リスクを回避するなら

例）

```
b = 0;
a = 0;
```

まとめ

C言語は、多くのプロセッサで利用可能、効率的な機械コードにコンパイルできる、シンプル等、多くの利点がある一方、これまで出題してきた問題にもあった通りC言語には、いくつかの弱点があります。

- 言語の未定義、未規定の領域がある
- 他の言語より多くの演算子があり、それらの優先順位や型ルールが分かりづらい
- 良くありがちな問題(演算例外、オーバーフロー等)の実行時チェックをコンパイラが提供していない。

MISRA Cは、C言語のサブセットであり、上記のようなC言語の弱点よる問題を除去または削減するためのガイドラインです。

MISRA Cは、ルールを守るだけのものではなく、MISRA Cを理解することにより、C言語を理解するための教材としても有効です。

最後に

組込み開発者におくる
MISRA C:2012
C言語利用の高信頼化ガイド

2016年10月電子書籍(kindle)にてリリース予定

Coming soon!