お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

User's Manual

MISRA C ルールチェッカ SQMlint V.1.03

ユーザーズマニュアル

ルネサスコンパイラ用拡張機能

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2006.08

安全設計に関するお願い

- 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

- 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは責任を負いません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、予告なしに、本資料に記載した製品又は仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前に株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ（http://www.renesas.com）などを通じて公開される情報に常にご注意ください。
- 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズはその責任を負いません。
- 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、適用可否に対する責任は負いません。
- 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へご照会ください。
- 本資料の転載、複製については、文書による株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズの事前の承諾が必要です。
- 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたら株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店までご照会ください。

製品の内容及び本書についてのお問い合わせ先

インストーラが生成する以下のテキストファイルに必要事項を記入の上、ツール技術サポート窓口support_tool@renesas.comまで送信ください。

¥SUPPORT¥製品名¥SUPPORT.TXT

株式会社ルネサス ソリューションズ

ツール技術サポート窓口	csc@renesas.com
ユーザ登録窓口	regist_tool@renesas.com
ホームページ	http://japan.renesas.com/tools

1. はじめに

MISRA C ルールチェッカ(以降 SQMlint と記します)は、MISRA C[1]:1998 で規定されている各ルールに C ソースコードが適合しているかを静的に検査するチェッカです。

MISRA C のルールはいずれもユーザガイドラインであり、プログラムを作成するユーザがレビューなどで検査する項目ですが、ルールの中には検査ツールで検出できるものが多数含まれています。SQMlint では、C ソースコードに対して MISRA C の各ルールを検査し、違反項目があったものに対してレポートすることで、ユーザのソースコードレビューを支援します。

このマニュアルに記述している MISRA C ルールの表題、 および MISRA C ルールチェッカの MISRA C ルール検査項目は、自動車技術会が翻訳した自動車用 C 言語利用のガイドライン JASO TP-01002 に基づいています。

[1] MISRA C とは、英国の MISRA(Motor Industry Software Reliability Association)が発行している自動車用ソフトウェア開発での C 言語利用ガイドラインのことです。

2. 概要

SQMlint は、ユーザが作成した C ソースコードが MISRA C のルールに違反していないかを検査します。

SQMlint は、C ソースコードに MISRA C のルールに違反する箇所があった場合、レポートメッセージ(4 章「レポートの仕様」を参照)を出力します。

例）

```
typedef unsigned short UINT16;
extern volatile UINT16 port1 = 0;
extern volatile UINT16 port2 = 0;

void func(void);
void func(void)
{
   while(port1 != 0) {
      if (port2 == 0) {
         break;
      }
   }
}
```

上記のプログラムを SQMlint で検査すると、[MISRA(58) Complaining : test.c, 10] 'break' statement shall not be used (except in a 'switch')というレポートメッセージを出力します。これは、10 行目の break 文の使用がルール 58 に違反していることを示しています。

注意

SQMlint は MISRA C で定義されているルールの一部を検査することができません。検査できるルールの一覧に関しては、後述の 6 章「MISRA C ルール検査項目一覧」を参照ください。

2.1. 位置付け

SQMlint は、コンパイルの一環としてコンパイラから起動します。従って、MISRA C のルールチェック後にコンパイルも実行されます。

補足 MISRA C のルールチェックを行っても、コンパイラが生成するコードに影響ありません。

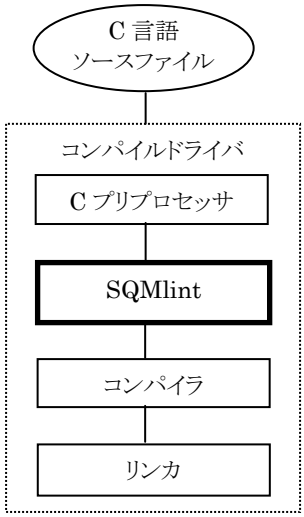

図 2.1 SQMlint の位置付け

2.2. 入出力ファイル

SQMlint は、C 言語ソースファイルを MISRA C のルールに対して検査し、検査結果をレポートファイル、あるいは標準エラー出力にレポートメッセージとして出力します(図 2.2 参照)。レポートファイル、レポートメッセージの仕様詳細については、後述の 4 章「レポートの仕様」を参照ください。

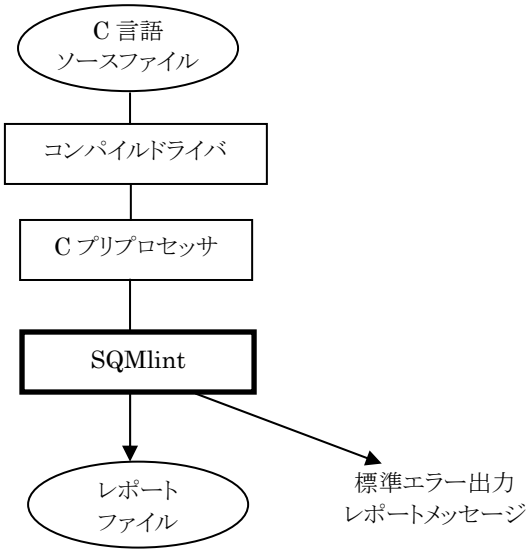

図 2.2 SQMlint の入出力ファイル

3. SQMlint の使い方

3.1. M16C ファミリ用 C コンパイラパッケージをお使いの場合

MISRA C ルールの検査をするには、M16C ファミリ用 C コンパイラ実行時に、表 3.1.1 のオプションを指定してください。

起動例） nc30 test.c -c -misra_all -misra_report report.csv

nc308 test.c -c -misra_all -misra_report report.csv

nc100 test.c -c -misra_all -misra_report report.csv

サポートしている MISRA C のルールをすべて検査し、結果を report.csv に出力します。

test.c のコンパイルも実行します。

表 3.1.1 コンパイラに追加される MISRA C ルール検査用オプション

オプション	機能
-misra_all	サポートしている MISRA C のルールをすべて検査します。 このとき、ルール番号は指定できません。
-misra_apply ルール番号[,ルール番号,...]	指定した MISRA C のルール番号の項目のみを検査対象とします。 ● ルール番号は、必ず 10 進数値で 1 つ以上指定してください。 ● 指定したルール番号が、0 以下か 128 以上、あるいは検査対象外ルール番号の場合、オプションエラーとなります。 ● ルール番号と','の間、あるいは','とルール番号の間にスペースは入れないでください。
-misra_ignore ルール番号[,ルール番号,...]	指定した MISRA C のルール番号の項目を、検査対象から除外します。 ● ルール番号は、必ず 10 進数値で 1 つ以上指定してください。ここで指定したルール番号以外のルールは、全て検査対象となります。 ● 指定したルール番号が、0 以下か 128 以上の場合、オプションエラーとなります。 ● ルール番号と','の間、あるいは','とルール番号の間にスペースは入れないでください。
-misra_required	サポートしている MISRA C のルールのうち、ルールの分類が「required」となっているものをすべて検査します。 このとき、ルール番号は指定できません。
-misra_required_add ルール番号[,ルール番号,...]	サポートしている MISRA C のルールのうち、ルールの分類が「required」となっているのもすべてと、ルール番号で指定した項目を検査対象とします。 ● ルール番号は、必ず 10 進数値で 1 つ以上指定してください。 ● 指定したルール番号が、0 以下か 128 以上の場合、オプションエラーとなります。 ● ルール番号と','の間、あるいは','とルール番号の間にスペースは入れないでください。
-misra_required_remove ルール番号[,ルー	サポートしている MISRA C のルールのうち、ルールの分類

ル番号,...]	が「required」となっているものの中から、ルール番号で指定されたルールを除外したすべてのルールを検査対象とします。 ● ルール番号は、必ず 10 進数値で 1 つ以上指定してください。 ● ルール番号と','の間、あるいは','とルール番号の間にスペースは入れないでください。
-misra_report 出力レポートファイル名	MISRA C のルール検査結果を保存する出力レポートファイル名を指定します。このオプションを指定した場合、出力レポートファイル名は省略できません。 拡張子には、".csv"を指定してください。 このオプションが省略された場合、検査結果を標準エラー出力に出力します。
-ignore_files_misra <ファイル名>[,<ファイル名>...]	指定したファイルを MISRA C の検査対象外します。 ● ディレクトリ付き，無し共に指定可能です。(file, dir/file, drive:/dir/file) ● ディレクトリ区切り文字は、'/'および'¥'が使用可能です。
-check_language_extension	拡張キーワード(__abs8 等)や拡張仕様(1 バイト Enum)をルール 1 違反としてレポートします。 デフォルトでは、拡張キーワードおよび拡張機能はルール 1 違反にしません。
-check_no_prototype_extension	__entry, __interrupt 指定した関数のプロトタイプ宣言がない場合、ルール 71 違反としてレポートします。 デフォルトでは、これらの関数はルール 71 違反にしません。

注意

-misra_all, -misra_apply, -misra_ignore, -misra_required, -misra_required_add,

-misra_required_remove は同時に指定できません。

3.2. M32R ファミリ用 C/C++コンパイラパッケージをお使いの場合

MISRA C ルールの検査をするには、M32R ファミリ用 C/C++コンパイラ実行時に、表 3.2.1 のオプションを指定してください。

起動例) cc32r test.c -c -misra_all -misra_report report.csv

サポートしている MISRA C のルールをすべて検査し、結果を report.csv に出力します。

test.c のコンパイルも実行します。

表 3.2.1 コンパイラに追加される MISRA C ルール検査用オプション

オプション	機能
-misra_all	サポートしている MISRA C のルールをすべて検査します。 このとき、ルール番号は指定できません。
-misra_apply ルール番号[,ルール番号,...]	指定した MISRA C のルール番号の項目のみを検査対象とします。 ● ルール番号は、必ず 10 進数値で 1 つ以上指定してください。 ● 指定したルール番号が、0 以下か 128 以上、あるいは

	検査対象外ルール番号の場合、オプションエラーとなります。 ● ルール番号と','の間、あるいは','とルール番号の間にスペースは入れないでください。
-misra_ignore ルール番号[,ルール番号,...]	指定した MISRA C のルール番号の項目を検査対象から除外します。ルール番号は、必ず 10 進数値で 1 つ以上指定してください。ここで指定したルール番号以外のルールは、全て検査対象となります。 ● 指定したルール番号が、0 以下か 128 以上の場合、オプションエラーとなります。 ● ルール番号と','の間、あるいは','とルール番号の間にスペースは入れないでください。
-misra_required	サポートしている MISRA C のルールのうち、ルールの分類が「required」となっているものをすべて検査します。 このとき、ルール番号は指定できません。
-misra_required_add ルール番号[,ルール番号,...]	サポートしている MISRA C のルールのうち、ルールの分類が「required」となっているのもすべてと、ルール番号で指定した項目を検査対象とします。 ● ルール番号は、必ず 10 進数値で 1 つ以上指定してください。 ● 指定したルール番号が、0 以下か 128 以上の場合、オプションエラーとなります。 ● ルール番号と','の間、あるいは','とルール番号の間にスペースは入れないでください。
-misra_required_remove ルール番号[,ルール番号,...]	サポートしている MISRA C のルールのうち、ルールの分類が「required」となっているものの中から、ルール番号で指定されたルールを除外したすべてのルールを検査対象とします。 ● ルール番号は、必ず 10 進数値で 1 つ以上指定してください。 ● ルール番号と','の間、あるいは','とルール番号の間にスペースは入れないでください。
-misra_report 出力レポートファイル名	MISRA C のルール検査結果を保存する出力レポートファイル名を指定します。このオプションを指定した場合、出力レポートファイル名は省略できません。 拡張子には、".csv"を指定してください。 このオプションが省略された場合、検査結果を標準エラー出力に出力します。

注意

-misra_all, -misra_apply, -misra_ignore, -misra_required, -misra_required_add,

-misra_required_remove は同時に指定できません。

3.3. SuperH ファミリ用 C/C++コンパイラパッケージをお使いの場合

3.3.1. オプション

MISRA C ルールの検査をするには、SuperH ファミリ用 C/C++コンパイラ実行時に表 3.3.1 のオプションを指定してください。オプションの大文字は、短縮形指定時の文字を意味します。

起動例） shc test.c -misra=all -report_misra=report.csv

サポートしている MISRA C のルールをすべて検査し、結果を report.csv に出力します。

test.c のコンパイルも実行します。

表 3.3.1 コンパイラに追加される MISRA C ルール検査用オプション

オプション	機能

MIsra={
ALl |
APply=<番号>[,<番号>,...] |
IGnore=<番号>[,<番号>,...] |
REQUIRED |
REQUIRED_Add=<番号>[,<番号>,...] |
REQUIRED_Remove=<番号>[,<番号>,...]
}
	(1) **misra=all を指定した場合**
	サポートしている MISRA C のルールをすべて検査します。
	このとき、ルール番号は指定できません。
	(2) **misra=apply ルール番号[,ルール番号,...] を指定した場合**
	指定した MISRA C のルール番号の項目のみを検査対象とします。
	ルール番号は、必ず 10 進数値で 1 つ以上指定してください。
	指定したルール番号が、0 以下か 128 以上、あるいは検査対象外ルール番号の場合、オプションエラーとなります。
	(3) **misra=ignore ルール番号[,ルール番号,...] を指定した場合**
	指定した MISRA C のルール番号の項目を検査対象から除外します。ルール番号は、必ず 10 進数値で 1 つ以上指定してください。ここで指定したルール番号以外のルールは、全て検査対象となります。
	指定したルール番号が、0 以下か 128 以上の場合、オプションエラーとなります。
	(4) **misra=required を指定した場合**
	サポートしている MISRA C のルールのうち、ルールの分類が「required」となっているものをすべて検査します。
	このとき、ルール番号は指定できません。
	(5) **misra=required_add を指定した場合**
	サポートしている MISRA C のルールのうち、ルールの分類が「required」となっているのもすべてと、ルール番号で指定した項目を検査対象とします。
	ルール番号は、必ず 10 進数値で 1 つ以上指定してください。
	指定したルール番号が、0 以下か 128 以上の場合、オプションエラーとなります。
	(6) **misra=required_remove を指定した場合**
	サポートしている MISRA C のルールのうち、ルールの分類が「required」となっているものの中から、ルール番号で指定されたルールを除外したすべてのルールを検査対象とします。
	ルール番号は、必ず 10 進数値で 1 つ以上指定してください。
	
	ルール番号と','の間、あるいは','とルール番号の間にスペースは入れないでください。
	このオプションが省略された場合、MISRA C の検査を行いません。

	注意）all, apply, ignore, required, required_add, required_remove を同時に指定できません。
REPort_misra [= <ファイル名>]	MISRA Cのルール検査結果を保存する出力レポートファイル名を指定します。このオプションを指定して、かつ"=レポートファイル名"部分を記述しない場合は、"コンパイル対象ファイル名.csv"となります。 このオプションが省略された場合、検査結果を標準エラー出力に出力します。

注意

-misra=all, -misra=apply, -misra=ignore, -misra=required, -misra=required_add,

-misra=required_remove は同時に指定できません。

3.3.2. エラーメッセージ

SQMlint をインストールすると、SH コンパイラに以下のエラーメッセージが追加されます。

エラーレベルは、エラーの重要度に従い、5 種類に分類されます。

エラーレベル		動作
(I)	インフォメーション	処理を継続します。
(W)	ウォーニング	処理を継続します。
(E)	エラー	オプション解析処理を継続し、処理を中断します。
(F)	フェータル	処理を中断します。
(-)	インターナル	処理を中断します。

表 3.3.2　SH コンパイラの MISRA C 検査に関するエラーメッセージ一覧

出力内容	意味
C3330 (F) MISRA Error	コマンドラインの記述が間違っています
C3331 (F) MISRA Internal Error	SQMlint の内部処理で何らかの障害が生じました。お求めになった営業所あるいは代理店にエラーの発生状況をご連絡ください。
C1340 (W) 'MISRA' option ignored	同時指定不可能なオプション-prep と-misra を指定されました。
C1340 (W) 'MISRA' option ignored	同時指定不可能なオプション-lang=C++と-misra を指定されました。
C3327 (F) Cannot find "sqmlint.exe"	SQMlint の実行プログラム（sqmlint.exe）が見つかりません。

3.4. H8SX,H8S,H8 ファミリ用 C/C++コンパイラパッケージをお使いの場合

3.4.1. オプション

MISRA C ルールの検査をするには、H8SX,H8S,H8 ファミリ用 C/C++コンパイラ実行時に表 3.4.1 のオプションを指定してください。オプションの大文字は、短縮形指定時の文字を意味します。

起動例） ch38　test.c　-misra=all　-report_misra=report.csv

サポートしている MISRA C のルールをすべて検査し、結果を report.csv に出力します。

test.c のコンパイルも実行します。

表 3.4.1 コンパイラに追加される MISRA C ルール検査用オプション

オプション	機能

MIsra={
ALl |
APply=<番号>[,<番号>,...] |
IGnore=<番号>[,<番号>,...] |
REQUIRED |
REQUIRED_Add=<番号>[,<番号>,...] |
REQUIRED_Remove=<番号>[,<番号>,...]
}
	(1) **misra=all を指定した場合**
	サポートしている MISRA C のルールをすべて検査します。
	このとき、ルール番号は指定できません。
	(2) **misra=apply ルール番号[,ルール番号,...]を指定した場合**
	指定した MISRA C のルール番号の項目のみを検査対象とします。
	ルール番号は、必ず 10 進数値で 1 つ以上指定してください。
	指定したルール番号が、0 以下か 128 以上、あるいは検査対象外ルール番号の場合、オプションエラーとなります。
	(3) **misra=ignore ルール番号[,ルール番号,...]を指定した場合**
	指定した MISRA C のルール番号の項目を検査対象から除外します。ルール番号は、必ず 10 進数値で 1 つ以上指定してください。ここで指定したルール番号以外のルールは、全て検査対象となります。
	指定したルール番号が、0 以下か 128 以上の場合、オプションエラーとなります。
	(4) **misra=required を指定した場合**
	サポートしている MISRA C のルールのうち、ルールの分類が「required」となっているものをすべて検査します。
	このとき、ルール番号は指定できません。
	(5) **misra=required_add を指定した場合**
	サポートしている MISRA C のルールのうち、ルールの分類が「required」となっているのもすべてと、ルール番号で指定した項目を検査対象とします。
	ルール番号は、必ず 10 進数値で 1 つ以上指定してください。
	指定したルール番号が、0 以下か 128 以上の場合、オプションエラーとなります。
	(6) **misra=required_remove を指定した場合**
	サポートしている MISRA C のルールのうち、ルールの分類が「required」となっているものの中から、ルール番号で指定されたルールを除外したすべてのルールを検査対象とします。
	ルール番号は、必ず 10 進数値で 1 つ以上指定してください。
	
	ルール番号と','の間、あるいは','とルール番号の間にスペースは入れないでください。
	このオプションが省略された場合、MISRA C の検査を行いません。
	注意）all, apply, ignore, required, required_add, required_remove を同時に指定できません。

REPort_misra [= <ファイル名>]	MISRA C のルール検査結果を保存する出力レ

	ポートファイル名を指定します。このオプションを指定して、かつ"=レポートファイル名"部分を記述しない場合は、"コンパイル対象ファイル名.csv"となります。 このオプションが省略された場合、検査結果を標準エラー出力に出力します。
IGnore_files_misra=<ファイル名>[,<ファイル名>...]	指定したファイルをMISRA Cの検査対象外します。 ● ディレクトリ付き，無し共に指定可能です。(file, dir/file, drive:/dir/file) ● ディレクトリ区切り文字は、'/'および'¥'が使用可能です。
CHECK_Language_extension	拡張キーワード(__abs8 等)や拡張仕様(1 バイトEnum)をルール 1 違反としてレポートします。 デフォルトでは、拡張キーワードおよび拡張機能はルール 1 違反にしません。
CHECK_No_prototype_extension	__entry,__interrupt 指定した関数のプロトタイプ宣言がない場合、ルール 71 違反としてレポートします。 デフォルトでは、これらの関数はルール 71 違反にしません。

注意

-misra=all, -misra=apply, -misra=ignore, -misra=required, -misra=required_add,

-misra=required_remove は同時に指定できません。

3.4.2. エラーメッセージ

SQMlint をインストールすると、H8 コンパイラに以下のエラーメッセージが追加されます。

エラーレベルは、エラーの重要度に従い、5 種類に分類されます。

エラーレベル		動作
(I)	インフォメーション	処理を継続します。
(W)	ウォーニング	処理を継続します。
(E)	エラー	オプション解析処理を継続し、処理を中断します。
(F)	フェータル	処理を中断します。
(-)	インターナル	処理を中断します。

表 3.4.2　H8 コンパイラの MISRA C 検査に関するエラーメッセージ一覧

出力内容	意味
C3330 (F) MISRA Error	コマンドラインの記述が間違っています
C3331 (F) MISRA Internal Error	SQMlint の内部処理で何らかの障害が生じました。お求めになった営業所あるいは代理店にエラーの発生状況をご連絡ください。
C1340 (W) 'MISRA' option ignored	同時指定不可能なオプション-prep と-misra を指定されました。
C1340 (W) 'MISRA' option ignored	同時指定不可能なオプション-lang=C++と-misraを指定されました。
C3327 (F) Cannot find "sqmlint.exe"	SQMlint の実行プログラム(sqmlint.exe)が見つかりません。

4. レポートの仕様

MISRA C のルールに対する検査結果には、以下の 2 つのレベルがあります。

1. 違反レベル

   MISRA C のルールに違反している場合

2. 警告レベル

   MISRA C のルールに違反している可能性がある場合

4.1. レポートメッセージ

MISRA C ルールの検査結果を、標準エラー出力に出力する場合のメッセージをレポートメッセージと呼びます。MISRA C のルールに違反、あるいは警告があった場合以下の形式でメッセージを出力します。

1. CC32R をお使いの場合

[MISRA(ルール番号) Complaining： ファイル名, line 行番号] メッセージ ⇨ 違反している場合
[MISRA(ルール番号) Warning： ファイル名, line 行番号] メッセージ ⇨ 警告がある場合

2. SH コンパイラ、H8 コンパイラ、NC30、NC308 をお使いの場合

ファイル名(行番号)： メッセージ番号 (I) [MISRA(ルール番号) Complaining] メッセージ ⇨ 違反している場合
ファイル名(行番号)： メッセージ番号 (I) [MISRA(ルール番号) Warning] メッセージ ⇨ 警告がある場合

4.2. レポートファイル

MISRA C ルールの検査結果をレポートファイルに出力する場合、多くの表計算ソフトで読み込むことのできる CSV(Comma Separated Value)形式のファイルとして出力します。

ソースコードを検査した結果、MISRA C ルールに違反あるいは警告があった場合、レポートファイルに、MISRA C ルール番号、違反/警告の種別、ファイル名、行番号、レポートメッセージの順にカンマ(,)で区切って出力します。レポートメッセージの後には改行コードを出力します。

レポートファイル出力例)

```
Rule,Level,File,Line,Message<改行>                    ⇨ ヘッダ
57,Complaining,"test.c",6 ,"the 'continue' ..."<改行> ⇨ 違反があった場合（test.c の 6 行目）
58,Warning,"test.c",10 ,"the 'break' ..."<改行>       ⇨ 警告があった場合（ test.c の 10 行目）
```

1 行目には、列に対するヘッダが出力されます。

2 行目以降に、検査結果が出力されます。

4.3. コンパイルエラーに関して

SQMlint が検出したコンパイルエラーは、MISRA C のルール1に対する違反としてレポートします。

コンパイルエラーが 500 以上出力された場合、検査を中断し、SQMlint の実行を終了します。

コンパイルエラーでない MISRA C のルールに対するレポートメッセージの出力数には制限がありません。

注意

SQMlint で MISRA C ルールを検査した後に実行されるコンパイラで検出されたコンパイルエラーは、レポートメッセージとして出力されません（通常のコンパイルエラーとして標準[エラー]出力に出力されます）。レポートファイルにも出力されません。

5. 検査結果の確認方法

MISRA C ルールの検査結果を確認する方法には、以下の 3 通りの方法があります。検査結果を検証する目的に応じて確認方法を選択してください。

5.1. レポートファイルで確認する

検査結果をソースコードレビューに活用する場合は、レポートファイル(CSV 形式)に結果を保存して確認すると便利です。

[レポートファイルの活用方法]

1. レポートファイルを表計算ソフトで読み込んで、ルールごとに検査結果を並べ替えたりすることができます。
2. 付属のSQMmergerを使用して混合表示ファイル(Cソースファイルの該当箇所に検査結果を挿入したテキストファイル)を作成することができます。SQMmerger の活用方法に関しては、後述の 5.3 項をご参照ください。

5.2. レポートメッセージで確認する

検査結果をコンパイルしながら一時的に確認する場合の方法です。通常のコンパイルエラーを確認するような検証方法としてご使用ください。

5.3. SQMmerger を使用して確認する

SQMmergerを使用すると、検査結果を保存したレポートファイルとCソースファイルから、Cソース行と対応するレポートメッセージの混合表示ファイルを作成できます。混合表示ファイルは、ソースコードレビューなどにご活用ください。

SQMmerger の使用方法に関しては、第8章「マージユーティリティ SQMmerger」をご参照ください。

混合表示ファイル出力例)

```
1 : void  func(void);
2 : void  func(void)
3 : {
4 : LABEL:
[MISRA(55) Complaining] label ('LABEL') should not be used

5 :
6 :        goto LABEL;
[MISRA(56) Complaining] the 'goto' statement shall not be used

7 : }
```

6. MISRA C ルール検査項目一覧

各 MISRA C ルールに対する検査の対応可否を表 6.1 に示します。

表 6.1 MISRA C ルール検査項目一覧表（○: 検査可、○*: 制限付きで検査可、×: 検査対象外）

ルール番号の左がグレーの場合は「必須(required)ルール」、白い場合は「推奨(advisory)ルール」であることを示しています。

ルール	可否
1	○
2	×
3	×
4	×
5	○
6	×
7	×
8	○
9	×
10	×
11	×
12	○
13	○
14	○
15	×
16	×
17	○*
18	○
19	○
20	○
21	○*
22	○*
23	×
24	○
25	×
26	×
27	×
28	○
29	○
30	×
31	○
32	○

ルール	可否
33	○
34	○
35	○
36	○
37	○
38	○
39	○
40	○
41	×
42	○
43	○
44	○
45	○
46	○*
47	×
48	○
49	○
50	○
51	○*
52	×
53	○
54	○*
55	○
56	○
57	○
58	○
59	○
60	○
61	○
62	○
63	○
64	○

ルール	可否
65	○
66	×
67	×
68	○
69	○
70	○*
71	○
72	○*
73	○
74	○
75	○
76	○
77	○
78	○
79	○
80	○
81	×
82	○
83	○
84	○
85	○
86	×
87	×
88	×
89	×
90	×
91	×
92	×
93	×
94	×
95	×
96	×

ルール	可否
97	×
98	×
99	○
100	×
101	○
102	○
103	○
104	○
105	○
106	○*
107	×
108	○
109	×
110	○
111	○
112	○
113	○
114	×
115	○
116	×
117	×
118	○
119	○
120	×
121	○
122	○
123	○
124	○
125	○*
126	○
127	○

7. MISRA C の各ルールの取り扱い

この章では、MISRA C の各ルールの取り扱いを、ルール順に示します。各ルール説明の読み方は以下のとおりです。

節 **MISRA C ルール番号**

(ルールの分類): MISRA C ルールの表題

- **解釈**

  MISRA C ルールに対する弊社の解釈

- **機能仕様**

  検査内容の詳細

- **注意事項**
- **制限事項**

※ 「MISRA C ルールの表題」は、自動車技術会が翻訳した自動車用 C 言語利用のガイドライン JASO TP-01002 に基づいています。

- 各ルールの説明で使用する語句の意味について以下に説明します。

語句	語句の説明
識別子	ラベル名、タグ名、typedef 名、変数名、関数名、メンバ名、列挙子名のことです。
翻訳単位	1つのソースファイルに記述されているインクルードファイルおよびソースファイルのことです。
オブジェクト	指定した型に応じた大きさと値をもつデータ記憶域のことです。 例えば、変数はオブジェクトです。

- 各ルールの説明で使用している typedef 名について

  各ルールの説明の中で変数などの型を表すのに以下の typedef 名を使用しています。

typedef 名	typedef 名の意味
SCHAR	1 バイトの符号付きの文字型 (signed char 型)
UCHAR	1 バイトの符号なしの文字型 (unsigned char 型)
SINT16	2 バイトの符号付き整数型 (signed short 型)
UINT16	2 バイトの符号なし整数型 (unsigned short 型)
SINT32	4 バイトの符号付き整数型 (signed int 型)
UINT32	4 バイトの符号なし整数型 (unsigned int 型)
SLONG	4 バイトの符号付き整数型 (signed long 型)
ULONG	4 バイトの符号なし整数型 (unsigned long 型)
FLOAT	4バイトの浮動小数点型 (float 型)
DOUBLE	8 バイトの浮動小数点型 (double 型)

7.1. ルール 1

(必要): 全てのコードは ISO9899 を満たしていなければならず、拡張機能は許されない。

- **解釈**

ISO9899:1990 で定義されていない、コンパイルメーカー独自の言語拡張機能は使用できません。
すべてのコードは ISO9899:1990 を満たしていなければなりません。

- **機能仕様**

ISO9899:1990 を満たしていない場合(コンパイルエラー)、「違反」レベルでレポートします。ルネサス製コンパイラ独自のキーワード、および言語拡張機能は、" -CHECK_Language_extension"オプションを指定した場合のみ「違反」レベルでレポートします。

- **注意事項**

コンパイルエラーの出力は、オプション(-misra ignore 1 など)で抑止できません。

7.2. ルール 2 (検査対象外)

(推奨): C言語以外の言語で書かれたコードは、C言語とその言語のコンパイラやアセンブラがどちらにも準拠したオブジェクトコードに対するインターフェース規格が定義されている場合にのみ使用可能である。

- **解釈**

C 言語で書いたソースコードから生成したオブジェクトコードと、C 以外の言語(例えば C++など)で書いたソースコードから生成したオブジェクトコードをリンクする場合、互いにアクセス可能なインタフェース規格が定義されていなければなりません。

- **機能仕様**

このルールは検査しません。弊社Cコンパイラは弊社アセンブラとのインターフェース規格を規定しています。規定されているインターフェースの詳細については、C コンパイラのマニュアルを参照してください。

7.3. ルール 3 (検査対象外)

(必要): C 言語から呼び出されるアセンブリ言語の関数は、インラインアセンブリ言語のみが含まれている C 言語の関数として表現されなければならず、インラインアセンブリ言語は一般の C コード内に組み込まれてはならない。

- **解釈**

C で記述したソースコードと、アセンブリで書いたルーチンのインタフェース統一を確実にするため、たとえアセンブリだけで構成されるルーチンであっても、単独のアセンブリソースファイルとして作成するのではなく、インライ

ンアセンブリを含んだ C の関数として作成するべきです。

ただしCは言語規格としてインラインアセンブリ機能を持たないため、この記述方法はルール１に違反します。

- **機能仕様**

このルールは検査しません。

7.4. ルール 4 （検査対象外）

(推奨)： 適切な実行時チェックのための規定を設けるべきである。

- **解釈**

表題通りです。

- **機能仕様**

このルールは検査しません。

7.5. ルール 5

(必要)： ISO9899 で定義されている文字や拡張表記のみ使用可能である。

- **解釈**

ISO9899:1990 で定義されている文字とエスケープシーケンスのみが使用可能です。

- **機能仕様**

文字定数、文字列リテラル中に漢字コード、カタカナ、規定されていないエスケープシーケンスが使用された場合、「違反」レベルでレポートします。

7.6. ルール 6 （検査対象外）

(必要)： 文字型の値は、ISO 10646-1 で、定義・文書化されたサブセットに制限される。

- **解釈**

表題通りです。

- **機能仕様**

このルールは検査しません。

7.7. ルール 7 （検査対象外）

(必要)： 3 文字表記は使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.8. ルール 8

(必要)： 多バイト文字や拡張文字列リテラルは使用してはならない。

- **解釈**

  多バイト文字、拡張文字リテラル、拡張文字定数は、使用できません。

- **機能仕様**

  多バイト文字とは、'漢'のような記述です。

  拡張文字定数とは、L'AB'のような記述です。

  拡張文字列リテラルとは、L"ABCD"や L"漢字"のような記述です。

  これらの文字、文字列を検出した場合、「違反」レベルでレポートします。

7.9. ルール 9 （検査対象外）

(必要)： コメントは入れ子であってはならない。

- **解釈**

  コメントの入れ子を許可するコンパイラもありますが、C 言語の規格には違反しています。コメントを入れ子にすると、移植性がなくなります。

- **機能仕様**

  このルールは検査しません。

7.10. ルール 10 （検査対象外）

(推奨)： コード部は’コメントアウト’してはならない。

- **解釈**

C のソースコードだけを含むコメントは、「必要に応じて有効にする機能を抑止することが目的」なのか「覚え書きとしての単なるコメント」なのか、保守担当者が変われば理解できなくなります。ソースコードとして意味ある箇所を無効化するためには、コメントでなく#if を使うことを推奨しています。

- **機能仕様**

このルールは検査しません。

7.11. ルール 11 （検査対象外）

(必要): (内部および外部)識別子は 31 文字を超える特徴に依存してはならない。さらに、コンパイラやリンカが外部識別子に対して 31 文字特徴と大文字小文字の区別をサポートしていることを保証していることの検査が必要である。

- **解釈**

全ての識別子の名前が 31 文字まで同じ名前であってはいけません。

- **機能仕様**

このルールは検査しません。

弊社 C コンパイラ、リンカは、31 文字以上の識別子をサポートしています。また、大文字小文字の区別もしています。

7.12. ルール 12

(推奨): あるネームスペースの識別子は他のネームスペースの識別子と同一スペルになってはならない。

- **解釈**

ネームスペース（名前空間）はラベル、タグ、メンバ、およびその他の識別子（関数名、メンバ以外の変数名、列挙子名、typedef 名）の 4 種類に分類できます。このルールではネームスペースの種類が異なっても、同一スペル（同一名）を使用してはならないことを意味します。

同じ種類のネームスペースで同一スペルが使われることについては、このルールの対象外です。例えば、異なる構造体に同一スペルのメンバがある場合などは対象外です。なおこのルールでは、スコープ（有効範囲）が異なれば同じ識別子を使用してもよいと解釈します。

- **機能仕様**

同一スコープにおいて、ラベル、タグ、メンバ、その他識別子が異なるネームスペースの識別子の名前と同一スペルの場合、「違反」レベルでレポートします。

例）

```
void    func(void)
```

```
{
    struct  tag  { SINT32 m; }  s;
    SINT32  tag;  /* ISO9899:1990 の仕様では、タグ名と変数名は重複しても正しい記述ですが、
                     MISRA C ではルール 12 違反となります。 */
}
```

同一スコープにおいて、同じ名前のメンバが異なる構造体の中で使用されていても警告しません。

例)

```
struct  tag1 { SINT32 m; }  s1;
struct  tag2 { SINT32 m; }  s2;  /* tag2 構造体のメンバ変数 m と、tag1 構造体の
                                    メンバ変数 m は同じ名前であってもよい。 */
```

7.13. ルール 13

(推奨): char, int, long, float および double という基本型は使用しないこと。かわりに、個々のコンパイラに対して特定長の等価物を typedef した上でコードではこれらが使用されるべきである。

- **解釈**

コンパイラに依存しない C ソースコードを記述するために、データのサイズを明確にしてプログラミングすることを意図しています。したがって表題に挙げられている基本型だけではなく、signed int など全ての基本型が typedef されているかどうかを検査します。

- **機能仕様**

宣言およびキャスト式で、typedef ではなく直接に基本型を使用した場合に「違反」レベルでレポートします。レポートする型名の詳細を以下に記します。

(1) char, int, short, long, float, double, long double, _Bool(NC のみ), long long(NC、SHC のみ)

例)

```
int  i;
int* p;
int  ary[5];
```

(2) (1)の signed 型、および unsigned 型

例)

```
signed int  si;
```

(3) 暗黙で型が int に決定される記述

例)

```
const   i;
signed  func(void);
```

7.14. ルール 14

(必要): char 型は常に unsigned char か signed char で定義されなければならない。

- **解釈**

表題通りです。

- **機能仕様**

unsigned および signed が無い char を用いた宣言、およびキャストに対して、「違反」レベルでレポートします。ポインタの指す先に unsigned および signed が無い char を用いた場合も同様に、「違反」レベルでレポートします。

- **注意事項**

下記記述のように char 型を typedef して使用した場合は、typedef 宣言時のみレポートします。typedef 型名を使用する際にはレポートしません。

例)

```
typedef   char   CHAR;      /*  違反メッセージを出力  */
CHAR   c;                   /*  ここでは違反メッセージを出力しません  */
char*   p;                  /*  違反メッセージを出力  */
```

7.15. ルール 15 （検査対象外）

(推奨): 浮動小数点の実装は定義された浮動小数点規格に従う。

- **解釈**

C コンパイラが実装している浮動小数点規格を明確にすることを要求しています。弊社Cコンパイラは、IEEE754 にしたがっています。

- **機能仕様**

このルールは検査しません。

7.16. ルール 16 （検査対象外）

(必要): 隠蔽されている浮動小数点数のビット表現は、プログラマはいかなる方法でも使用してはならない。

- **解釈**

表題通りです。

- **機能仕様**

  このルールは検査しません。

7.17. ルール 17

(必要): typedef 名は再使用してはならない。

- **解釈**

  typedef 名は、あらゆるコード、ファイルの中で再使用してはいけません。

- **機能仕様**

  typedef 名が、翻訳単位にある全スコープ中の識別子名と1つでも同じ場合、「違反」レベルでレポートします。

- **制限事項**

  翻訳単位をまたがっての検査はしません。

7.18. ルール 18

(推奨): 適切な接尾語が利用できるならば、定数は型を示す接尾語をつけなければならない。

- **解釈**

  定数が式で使用されるとき、暗黙に型変換される場合があります。表題の適切な接尾語(u, l, ul, f など)とは、その変換後の型を表すことができる接尾語のことです。

- **機能仕様**

  演算に使用される定数が次の条件を同時に満たす場合に「違反」レベルでレポートします。

  - その定数が暗黙に変換される場合。
  - その変換後の型が、接尾語で指定できる型にもかかわらず接尾語が付いていないか、もしくは、異なる型を表す接尾語が付く場合。
  - 単項+演算子のオペランドの定数、単項-演算子の定数、( )括弧で囲まれた定数、およびそれらの組合せについても検査対象とします。

  例 1)

  ```
  unsigned int ui;
  ui = ui + 3;   /* 3(int 型)が unsigned int 型に変換されるので違反  */
  ```

  これは、

  ```
  ui = ui + 3u;
  ```

  とします。

例 2)

```
long l;
l = -3;      /* -3(int 型)が long 型に変換されるので違反。int が 32 ビットの場合であっても、int と long
                は異なる型なので違反である。 */
```

これは、

```
l = -3L;
```

とします。

- **注意事項**

例 1)

```
FLOAT  f ;
f = f + 1.0;
```

1.0 は double 型なので、変数の f が double に拡張されます。定数である 1.0 は変換されずに double のまま演算されるので、このルールに違反したことにはなりません。

例 2)

```
UINT32  l ;
l = l + 'a';      /* 'a'  は数を表す定数ではないので検査の対象外です。 */
```

7.19. ルール 19

(必要): (0 以外の)8進数定数は使用してはならない。

- **解釈**

8進数定数は、10進数定数表記と紛らわしいため、定数を記述するときに間違いを引き起こします。そのため、0 以外の 8 進数定数は使用してはいけません。

- **機能仕様**

0 以外の 8 進数定数が使用された場合、「違反」レベルでレポートします。

7.20. ルール 20

(必要): すべてのオブジェクトと関数識別子は使用する前に宣言しなければならない。

- **解釈**

このルールは、次のことを禁止しています。

- 関数宣言がない、もしくは関数宣言よりも前にその関数を呼び出すこと
- 宣言されていない変数を使用すること

（宣言されていない変数を使用すると、ISO9899:1990 に準拠した C コンパイラではコンパイルエラーになります。）

- **機能仕様**

未宣言の変数および関数を使用した場合、「違反」レベルでレポートします。

7.21. ルール 21

(必要)： 識別子を隠してしまうことになるため、内部有効範囲の識別子は、外部有効範囲の識別子と同じ名前で使用してはならない。

- **解釈**

ある有効範囲（スコープ）の中に別の有効範囲がある場合の、内側の有効範囲を内部有効範囲、外側を外部有効範囲と解釈します。このルールは、例えばファイルスコープで宣言した変数と同じ名前の変数を、関数内で宣言することを禁止しています。

ただし、識別子とメンバ名は同じ名前でも構いません。

- **機能仕様**

関数内識別子が、外側のスコープの識別子と同じ名前の時、「違反」レベルでレポートします。

例）

```
SINT32 val;
void func(void)
{
    SINT32 val;   /* この変数 val とグローバル変数 val の名前が同じなので違反 */
}
```

- **注意事項**

次の例のような場合は、内側の変数名を隠さないため検査対象外です。

例）

```
void func(void)
{
    SINT32 val;
}
SINT32 val;   /* グローバル変数 val は上のローカル変数 val を隠さない */
```

- **制限事項**

翻訳単位をまたがっての検査はしません。

7.22. ルール 22

(推奨): オブジェクト宣言の有効範囲はより広い範囲が必要でない限り関数範囲とすべきである。

- **解釈**

ファイルスコープで定義された「extern でない変数」が、ただ 1 つの関数内だけで使用されるのであれば、その変数は関数内で定義するべきです。

- **機能仕様**

ファイルスコープに定義された変数(extern 宣言された変数を除く)が、2つ以上の関数で使用されていない場合、「警告」レベルでレポートします。

- **制限事項**

翻訳単位をまたがっての検査はしません。

7.23. ルール 23 （検査対象外）

(推奨): ファイルスコープの全ての宣言は可能な限り静的(static)にしなければならない。

- **解釈**

表題通りです。

- **機能仕様**

このルールは検査しません。

7.24. ルール 24

(必要): 識別子は同一翻訳単位において内部および外部結合を同時に持ってはならない。

- **解釈**

同じ名前をもった識別子は、スコープに関係なく異なるリンケージ(extern、static など)を持ってはいけません。

- **機能仕様**

同じ名前の識別子が、異なるリンケージを持つ場合、「違反」レベルでレポートします。

例)

```
static SINT32 a;
SINT32 a;    /* static 宣言された後に、変数 a を仮定義しているので違反 */

static SINT32 b;
extern SINT32 b; /* static 宣言された後に、extern 宣言しているので違反 */

static SINT32 s1;
static SINT32 s2;
void func(void)
{
    extern SINT32 s1;   /* static 宣言された後に、extern 宣言しているので違反 */
    SINT32 s2;          /* このルールには違反していませんが、ルール 21 に違反 */
```

7.25. ルール 25 (検査対象外)

(必要): 外部結合の識別子は唯一の外部定義を持たなければならない。

- **解釈**

表題通りです。

- **機能仕様**

このルールは検査しません。

このルールに違反した場合、リンカでエラーとなります。

7.26. ルール 26 （検査対象外）

(必要)： オブジェクトや関数が一度以上宣言された場合、互換性のある宣言を持たなければならない。

- **解釈**

  2つの異なるファイル間において、一方のファイルでは"extern int a;"、もう一方のファイルでは"extern float a;" というような宣言をしてはいけません。

- **機能仕様**

  異なるファイル間で変数が宣言された場合、このルールは検査しません。

同一ファイルで宣言された変数の型が異なる場合、ルール1のコンパイルエラーとしてレポートします。

7.27. ルール 27 （検査対象外）

(推奨)： 外部オブジェクトは 1 つのファイル以上で宣言すべきではない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.28. ルール 28

(推奨)： register 記憶クラス指定子は使用すべきではない。

- **解釈**

  表題通りです。

- **機能仕様**

  register 記憶クラス指定子を記述した場合に「違反」レベルでレポートします。

  例）

```
void   func( register   SINT32   arg)      /* 違反 */
{
      register   SINT32   i;                /* 違反 */
}
```

7.29. ルール 29

(必要): タグの使用は宣言と一致しなければならない。

- 解釈

(1) タグの与えられた構造体や共用体を初期化するとき、そのメンバを初期化する式が代入できない場合、このルールに違反していると考えます。これは、異なるファイルで同じタグ名の、異なる構造体や共用体が定義されている場合に、意図していない方が使用されるのを防ぐためです。

なお、ISO9899:1990 に準拠した C コンパイラではコンパイルエラーになります。

例)

```
struct S1 { SINT32 i; } s1 = { "abc" };   /* 違反かつコンパイルエラー */
struct { SINT32; } s2 = { "abc" };        /* タグが与えられていないのでコンパイルエラーのみ */
struct S2 { SLONG l; } s3 = { 0ul };      /* 検査対象外 */
struct S3 { ULONG ul; } s4 = { (ULONG)"abc" }; /* 検査対象外*/
struct T t;
void func(void)
{
    struct S1 s = t;  /* 違反かつコンパイルエラー */
}
```

(2) 列挙型の変数を列挙子でない定数や式で初期化するのはこのルールに違反していると考えます。

例)

```
enum NUM { ONE = 1, TWO = 2 };
enum NUM n = 1;       /* 違反 */
```

(3) 定数や整数型の変数を列挙型でキャストするのもこのルールに違反していると考えます。

例)

```
enum NUM { ONE = 1, TWO = 2 };
SINT32 n = 2;
(enum NUM)1;      /* 違反 */
(enum NUM)n;      /* 違反 */
```

- **機能仕様**

列挙型の変数を列挙子でない定数式で初期化している場合に「違反」レベルでレポートします。

定数や整数型の変数を列挙型でキャストしている場合に「違反」レベルでレポートします。

タグの与えられた構造体や共用体を初期化するとき、そのメンバを初期化する式が代入できない場合に「違反」レベルでレポートします。

構造体や共用体を、異なる構造体や共用体で初期化する場合に「違反」レベルでレポートします。

7.30. ルール 30 （検査対象外）

(必要): 全ての自動変数は使用する前に値を代入していなければならない。

- **解釈**

表題通りです。

- **機能仕様**

このルールは検査しません。

7.31. ルール 31

(必要): 配列や構造体の非ゼロの初期化においては、構造を示し、それに合わせるために中かっこ{ }が使用されなければならない。

- **解釈**

配列や構造体を初期化子リスト(カンマで区切られた初期値を表す式の並び)で初期化するとき、その配列や構造体を表す括弧を記述しなければなりません。

例）

```
struct S {
    struct T {  SINT32  i;  SINT32  j;  }  t;
    SINT32  x;
} s = {  { 1, 2 },  3  };      /* { 1, 2 }の括弧が必要 */
```

- **機能仕様**

初期化子リスト中に、構造体を初期化する初期化子リスト部分、および、配列を初期化する初期化子リスト部分を囲む中かっこ{}が省略されている場合に「違反」レベルでレポートします。

ただし、共用体を初期化する初期化子(初期値を表す式)についてはこのルールに該当しません。また、初期化子が文字列リテラル(“”で囲まれた文字列)の場合もこのルールに該当しません。

例）

```
SINT32 arr[2][3] = { 1,2,3, 4,5,6+1 };   /* 違反 */
```

これは、

```
SINT32 arr[2][3] = { {1,2,3}, {4,5,6+1} };
```

とします。

7.32. ルール 32

(必要): 列挙子リストにおいて全ての項目が明白に初期化されない限り、第 1 項以外のメンバを'='を使用して初期化してはならない。

- **解釈**

表題通りです。

- **機能仕様**

列挙子に'='を指定できるのは以下のどちらかの場合のみとします。以下のどちらにも当てはまらない列挙宣言は、「違反」レベルでレポートします。

- 第 1 項のみが'='で初期化されている。
- 全ての項が'='で初期化されている。

例)

```
/* 違反にならないパターン */
enum E1 { a1, b1, c1 };
enum E2 { a2 = 1, b2, c2 };
enum E3 { a3 = 1, b3 = 3, c3 = 5 };

/* 違反になるパターン */
enum E4 { a4, b4 = 3, c4 };
enum E5 { a5 = 1, b5, c5 = 5 };
```

7.33. ルール 33

(必要): &&や||演算子の右辺は副作用を含んではならない。

- **解釈**

&&および||演算子は、左辺式の評価結果によって、右辺式を評価するかしないかが変化します。右辺式が評価されるときに副作用を生じるような式を記述すると、左辺式の評価結果によって、グローバル変数が書き換えられるなどの副作用が生じたり生じなかったりします。そのような不定の振る舞いを防ぐことがこのルールの目的です。

以下の操作を副作用と呼びます。

- オブジェクトの変更(変数やポインタが指す先を変更するような場合)
- volatile オブジェクトの参照や変更
- ファイルの変更
- 上記の操作のいずれかを行う関数の呼び出し

- **機能仕様**

&&および||演算子の右辺に副作用が生じる式がある場合にレポートします。このとき、右辺式の副作用の原因が関数呼び出しのみの場合は「警告」レベルでレポートし、それ以外の場合は「違反」レベルでレポートします。

関数呼び出しに対して「警告」とする理由は、その関数呼び出しに副作用があるかどうかは、呼び出される側の関数定義の記述に依存するためです。呼び出される側の関数定義の中に副作用を生じる式があれば、その関数呼び出しは副作用を生じます。しかし呼び出される側の関数定義の中に副作用を生じる式が無ければ、その関数呼び出しは副作用を発生しません。

例)

```
volatile UINT16 us;
SINT16 n;
if ( (n == 0) || is_empty() ) {      /* 警告 */
if ( (n == 0) && (us == 0) ) {       /* 違反 */
```

7.34. ルール 34

(必要): 論理結合演算子(&&や||)の被演算項は一次式でなくてはならない。

- **解釈**

ここで一次式とは( )で囲まれた式、変数、関数、列挙子、および、定数を指します。&&と||の左右を、一次式だけに限定することによって、意図する論理式であることを保証し、読みやすくします。

- **機能仕様**

&&と||の左右のいずれか一方でも一次式でなければ、「違反」レベルでレポートします。

例 1)

```
if ( a==0 && b==0)  {              /*  違反  */
```

これは、

```
if ((a==0) && (b==0)) {
```

とします。

例 2)

```
if (is_empty() || is_zero) {          /*  関数も変数も一次式なので違反ではない  */
```

例 3)

```
if ((a==0) && (b==0) || (c==0)) {   /* ||の左辺式が一次式ではないので違反  */
```

これは、

```
if (((a==0) && (b==0)) || (c==0)) {
```

とします。

7.35. ルール 35

(必要): ブーリアン値を返す式の中で代入演算子が使用されてはならない。

- **解釈**

以下の式を「ブーリアン値を返す式」と考えます。

- 条件式
  - if、else if の()の中
  - while の()の中
  - for の括弧の中の第二式
  - ?: の第一項目の式
- 論理結合演算子(&&と||)の左右の式
- !演算子の右辺式

- **機能仕様**

条件式、論理結合演算子(&&と||)の左右の式、および、論理否定演算子(!)の右辺式に代入演算子が含まれているとき「違反」レベルでレポートします。

例 1)

```
if ( a = b)  {              /*  違反  */
```

これは、

```
a = b;
if (a != 0 ) {
```

とします。

7.36. ルール 36

(推奨): 論理演算子はビット演算子と混同すべきではない。

- **解釈**

例えば、if (a & b)という記述は、if ((a & b) != 0)を省略したものですが、if(a && b)を意図して&演算子と&&演算子を誤用しているか、もしくは記述間違いの可能性もあります。このルールでは、次の式中で、このような誤用の可能性がある記述を検出します。

- 条件式
  - if、else if の()の中
  - while の()の中
  - for の括弧の中の第二式
  - ? : の第一項目の式
- 論理結合演算子(&&と||)の左右の式
- !演算子の右辺式

- **機能仕様**

条件式中の式、論理結合演算子(&&と||)の左右の式、および、論理否定演算子(!)の右辺式で、最後に評価される演算子がビット演算子の&、|あるいは~である場合、「違反」レベルでレポートします。

7.37. ルール 37

(必要): ビット演算子を符号付き整数型で実行してはならない。

- **解釈**

表題の通りです。

符号付き整数型に^演算を実行するのも禁止されていると考えます。

- **機能仕様**

以下の式の型が符号付きの整数型の場合、「違反」レベルでレポートします。

- ビット演算子(~)の右辺式
- ビット演算子(& 、^および|)の左右の式
- ビットシフト演算子(<<、および>>)の左辺式
- ビット演算子の複合代入演算子(&=、^=、および|=)の左右の式
- ビットシフト演算子の複合代入演算子(<<=、および >>=)の左辺式

例)

```
SINT32 si;
si = si >> 1u;   /* signed int 型に対してシフト演算をするのは違反  */
```

- **注意事項**

検査の対象となる式が符号付きの定数値となる場合も「違反」レベルでレポートされます。

例)

```
0xFFFF << 4u;
```

このとき、0xFFFF は int 型であり、符号付きの定数値となるので、「違反」レベルでレポートされます。これを回避するには次のように修正します。

```
0xFFFFU << 4u;
```

7.38. ルール 38

(必要): シフト演算子の右辺の項はゼロ以上、左辺の項(をすべて含む)のビット幅未満でなければならない。

- **解釈**

表題通りです。

シフトされる値が、その型のビット幅以上でシフトする場合(例えば unsigned short 型の変数を 17 ビットシフトする)、およびシフト回数が負の値の場合、シフト結果の値はコンパイラによって異なる可能性があります。あるいは同一のコンパイラであっても、コンパイルオプションを変えたり、コンパイラのバージョンを変えるとシフト結果の値が異なる可能性があります。

- **機能仕様**

シフト演算子(<<、>>、<<=、および、>>=)の右辺(シフト数)が定数式の場合、その値が左辺式の型のビット幅以上であれば「違反」レベルでレポートします。

シフト演算子の右辺(シフト数)の項が定数式でない場合、未定義の動作を引き起こす可能性があるので、「違反」レベルでレポートします。

7.39. ルール 39

(必要): 単項演算子 '-' は符号なしの式に適用してはならない。

- **解釈**

表題通りです。

例えば unsigned int 型の変数に、単項'-'演算子を作用させた結果の型は unsigned int のままです。このように、型によっては注意して演算する必要があります。また符号無しである式に、符号を反転させるための単項'-'演算子を作用させることは、ソースコードの意味を読み取りにくくする可能性があります。

- **機能仕様**

単項演算子 '-'の右辺式が、符号なしの型を持つ式のとき、「違反」レベルでレポートします。

例)

```
UINT16 us;
-us;                /* 違反 */
```

7.40. ルール 40

(推奨): sizeof 演算子は副作用を含む式において使用すべきではない。

- **解釈**

以下の操作を副作用と呼びます。

- オブジェクトの変更(変数やポインタが指す先を変更するような場合)
- volatile オブジェクトの参照や変更
- ファイルの変更
- 上記の操作のいずれかを行う関数の呼び出し

これらの式を sizeof 演算子のオペランドに書いても、プログラムの実行時に副作用は発生しません。そのように、設計者以外の第 3 者(保守担当者など)が読んで誤解を生じるような記述を避けることを要求しています。

- **機能仕様**

sizeof 演算子のオペランド(sizeof(この式))に副作用を生じる式が含まれている場合にレポートします。該当の式に関数呼び出しが含まれていない場合は、「違反」レベルでレポートします。関数呼び出しが含まれている場合は、「警告」レベルでレポートします。

例)

```
#include <stddef.h>
size_t   size;
int   func(int);
size = sizeof( func(5) );          /* 違反 */
```

7.41. ルール 41 （検査対象外）

(推奨): 選定したコンパイラの整数除算の実装は確認し、文書化し、考慮するべきである。

- **解釈**

表題通りです。

整数の除算結果（例えば負の値の除算結果、-1 / 3 の結果はゼロか、あるいはマイナス 1 か）は処理系によって異なる可能性があるため、マニュアルなどを確認して文書化する必要があります。

- **機能仕様**

このルールは検査しません。

7.42. ルール 42

(必要): for ループの制御式を除いて、コンマ演算子は使用してはならない。

- **解釈**

表題通りです。

- **機能仕様**

for 文の( )中の式以外で、カンマ演算子が使用された場合に「違反」レベルでレポートします。

7.43. ルール 43

(必要): 情報の損失を引き起こす暗黙的な変換は使用してはならない。

- **解釈**

情報の損失を引き起こす変換とは、変換後の型では変換前の値を表現できない場合を指します。ある値がそれよりも精度の低い型やサイズの小さい型へ代入、引数渡し、戻り値渡し、および、初期化される場合に、その型への変換で情報の損失が生じることがあります。

以下の場合の変換が情報の損失を引き起こします。

- 式の結果が、より小さなサイズの型に変換される場合
- 符号付きが符号なしの型に変換される場合
- 符号なしが符号付きの型に変換される場合
- 浮動小数点型が、整数型に変換される場合
- 定数式が、その値を保持できない型に変換される場合

- **機能仕様**

以下の文脈で情報の損失を引き起こす変換があった場合に「違反」レベルでレポートします。

- 代入のとき(複合代入を含む)
- 関数呼び出しの引数として渡されるとき
- 関数定義中で戻り値として返されるとき(return 文)
- 初期化のとき

また定数式を、その値を保持できないサイズのビットフィールドに代入する、もしくは初期化する場合も情報の損失が生じるので、「違反」レベルでレポートします。このルールは、整数型、浮動小数点型、列挙型、ポインタに対して検査します。

- **注意事項**

intよりも小さいサイズの型の値で演算する場合、Cの規約では被演算項をint型に拡張してから演算します。この拡張を汎整数拡張(integral promotion)と呼びます。

次の例は一見したところ情報の損失が起きているようには見えませんが、汎整数拡張が発生するため、C の構文規則だけから判断すると情報損失が起きていることと同義になります。このため次の行に対しても、ルール 43 違反のレポートが出力されます。

例)

```
/* uint8_var2 と 10U は加算前に int に拡張される。int 型で加算した後、uint8_var1 に代入される。
 * このため、加算結果として保持している uint8 を超えたオーバーフロー分のビット情報が、代入で
 * 欠落する可能性がある。 */
uint8_var1 = uint8_var2 + 10U;

/* uint8_var2 と 0xFU は&演算の前に int に拡張される。int 型で&演算した後、uint8_var1 に代入
 * される。このため論理的には、uint8 を超えた領域にはゼロのビットパターンが埋まっていることになる。
 * 更に代入により、その上位のビットパターンが欠落することになる。 */
uint8_var1 = uint8_var2 & 0xFU;
```

7.44. ルール 44

(推奨): 冗長に明示的なキャストを使用すべきではない。

- **解釈**

冗長に明示的なキャストは以下の場合と考えます。

- 同じ型にキャストしている場合

- **機能仕様**

同じ型にキャストしている場合には、直接キャストしている場合と、複数のキャストを経て同じ型にキャストしている場合の二種類があります。前者を「違反」レベルでレポートし、後者を「警告」レベルでレポートします。

例)

```
SCHAR c;
SINT16 s;
SINT32 n;
SLONG l;
ULONG ul;
DOUBLE d;
(SINT32)n;              /* 違反 */
(SLONG)(SLONG)n;   /* 違反 */
(SINT16)(SLONG)s;   /* 警告 */
(SINT16)(SCHAR)s;   /* 警告 */
```

7.45. ルール 45

(必要): 全ての型からポインタへの、またはポインタからの全ての型へのキャスティングは使用してはならない。

- **解釈**

表題通りです。

- **機能仕様**

ポインタ型でキャストしている場合、ポインタ型の変数に対してキャストしている場合、およびポインタ型を返す関数呼び出しに対してキャストしている場合、「違反」レベルでレポートします。

7.46. ルール 46

(必要): 式の値は、規格が認めるどのような順序で評価されようとも同じでなければならない。

- **解釈**

副作用完了点(関数呼び出しの( )、 && 、 || 、 ?: 、 ; 、カンマ演算子)から次の副作用完了点までの間の式については、その評価順序が規定されていません。その間に同じ副作用を生じる式が含まれていた場合、コンパイラによってその評価順が異なるために、意図した結果を得られない可能性があります。このルールは、そのような記述を禁止しています。

以下の操作を副作用と呼びます。

- オブジェクトの変更(変数やポインタが指す先を変更するような場合)
- volatile オブジェクトの参照や変更
- ファイルの変更
- 上記の操作のいずれかを行う関数の呼び出し

- **機能仕様**

副作用完了点から次の副作用完了点までに以下の式が含まれている場合にレポートします。

(1) 同一の変数を変更する式(代入演算、++、および--演算)

(2) 同一の volatile 変数を含む式

(3) 二つ以上の関数呼び出し

ただし、次の場合は「警告」レベルでレポートし、それ以外は「違反」レベルでレポートします。

- (1)および(2)の場合で、そのオブジェクトが構造体および共用体のメンバ変数の場合
- (1)および(2)の場合で、そのオブジェクトが配列の要素の場合
- (3)の場合

例)

```
ULONG   ul = 10UL;
volatile   ULONG   v;
ul = (ul++) + ul;                    /*  違反  */
ul = v + v;                          /*  違反  */
ul = func (pop(), push());           /*  警告  */
```

- **注意事項**

実引数の評価は、関数呼び出しよりも必ず先に行われます。このため、以下の例は違反とみなしません。

例)

```
x = func( y = z / 3 );
```

- **制限事項**

ポインタが指す同じオブジェクトに副作用がある場合については、検査しません。

例)

```
*p = *p++ + *p++;
```

など。

7.47. ルール 47 （検査対象外）

(推奨): 式中にはC言語の演算子の優先順位規則に関するどんな依存もあるべきではない。

- **解釈**

表題通りです。

- **機能仕様**

このルールは検査しません。

7.48. ルール 48

(推奨): 予期する結果を生成するには混合算術精度は明白なキャストを使用すべきである。

- **解釈**

このルールは、暗黙の変換によって意図しない低い精度で演算が行われ、情報が欠落するのを防ぐことを目的としています。ある演算の結果が、次にそれよりも高い精度で演算される場合、その結果は期待する精度で演算されたものではない可能性があると考えられるためです。

- **機能仕様**

算術四則演算(+、-、*、/)の結果を、次にそれよりも高い精度で以下の演算に使用する場合、「違反」レベルでレポートします。

- 算術演算(+、-、*、/、および、%)
- 代入演算(=、+=、-=、*=、/=、および、%=)
- キャスト

また、算術四則演算の結果を、それよりも高い精度の型を持つ変数の初期化に使用する場合も「違反」レベルでレポートします。

例)

```
UINT16 x = 65535u;
UINT16 y = 10u;
UINT32 ul1 = (UINT32) (x + y);   /*  違反(x+y は 9 になる)*/
```

意図する正しい記述にするには、

```
UINT32 ul2 = (UINT32)x + (UINT32)y;   /* x+y は 65545 になる  */
```

とします。

7.49. ルール 49

(推奨): 非演算子が実質的に bool 型でない限り、ゼロとの比較テストは明示的にすべきである。

- **解釈**

実質的に bool 型である式とは、最後に評価される演算子が、'&&'、'||'、'!'、'<'、'>'、'<='、'>='、'=='および'!='である演算の式のことであるとします。

以下の式が、実質的に bool 型でない式のとき、ゼロとの比較が必要と考えます。

- 条件式

  if、else if の()の中

  while の()の中

  for の括弧の中の第二式

  ? : の第一項目の式

- 論理結合演算子(&&と||)の左右の式
- !演算子の右辺式

- **機能仕様**

条件式、論理結合演算子(&&および||)の左右の式、および、!演算子の右辺式の最後に評価される演算が、'&&'、'||'、'!'、'<'、'>'、'<='、'>='、'=='および'!='演算子のいずれでもない場合、「違反」レベルでレポートします。

例)

```
ULONG *pl;
if (pl) {        /*  違反  */
```

これは、

```
if (pl != 0) {
```

とします。

7.50. ルール 50

(必要): 浮動小数点変数は厳密な等式、非等式のテストをしてはならない。

- **解釈**

表題通りです。

浮動小数点型は、常にいくらかの誤差を含んでいることを考慮した上で使用する必要があります。'=='または'!='による厳密な等価比較をする場合、この誤差が原因で期待する結果が得られない可能性があります。

また期待する結果が得られた場合でも、コンパイラのオプションを変えたり、コンパイラのバージョンが変わるだけで結果が変わる可能性があります。

- **機能仕様**

'=='および'!='演算子の左右の式が、いずれか一方でも浮動小数点型のとき、「違反」レベルでレポートします。

例)

```
if (d == 1.0) {     /*  違反  */
```

7.51. ルール 51

(推奨): 符号なし整数定数式の評価は結果の型にはまるべきである。

- **解釈**

符号なし整数定数式を評価した結果、その値が結果の型をオーバーフローもしくはアンダーフローする場合に結果の型にはまっていないとみなします。

- **機能仕様**

結果の型が符号なしの整数定数になる定数式を評価した結果、オーバーフローする場合と、アンダーフローする場合に「違反」レベルでレポートします。

例)

```
ULONG ul = 3UL – 8UL;                    /*  違反(アンダーフロー) */
```

- **制限事項**

マクロ中の定数式については検査しません。

例)

```
#if (1u - 2u)
```

など。

7.52. ルール 52 （検査対象外）

(必要): 到達しないコードがあってはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.53. ルール 53

(必要): 空でない文は全て副作用がなければならない。

- **解釈**

  表題通りです。

  以下の操作を副作用と呼びます。

  - オブジェクトの変更（変数やポインタが指す先を変更するような場合）
  - volatile オブジェクトの参照や変更
  - ファイルの変更
  - 上記の操作のいずれかを行う関数の呼び出し

- **機能仕様**

  関数呼び出しでない式で、副作用の生じない式があれば、「違反」レベルでレポートします。ただし、副作用を生じない関数呼び出しがあっても、レポートしません。

7.54. ルール 54

(必要): 空の文はそれ自身を一行だけにおかなければならず、同じ行にはその他どんなテキストもおいてはならない。

- **解釈**

空文(;)の前後にはその行に、コメントも式も記述してはいけません。

例)

```
a = 1; ;      /* 違反 */
; a = 1;      /* 違反 */
if (1);       /* 違反 */
;;            /* 違反 */
```

- **機能仕様**

空文の前後に、式が記述されたとき、「違反」レベルでレポートします。

- **制限事項**

空文の前後に、コメント、マクロ記述がある場合の検査はできません。

例)

```
#define TEXT
void func(void)
{
    TEXT;   /* この場合の検査ができません。 */
}
```

7.55. ルール 55

(推奨): switch 文以外では、ラベルを使用するべきでない。

- **解釈**

表題通りです。

- **機能仕様**

ラベルが定義された場合、「違反」レベルでレポートします。

7.56. ルール 56

(必要): goto 文を使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  goto 文が使用された場合、「違反」レベルでレポートします。

7.57. ルール 57

(必要): continue 文を使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  continue 文が使用された場合、「違反」レベルでレポートします。

7.58. ルール 58

(必要): (switch 文の条件を終了する以外は)break 文は使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  switch 文の条件を終了する以外で、break 文が使用された場合、「違反」レベルでレポートします。case 節のローカルブロックで break 文を使用した場合、「違反」レベルでレポートします。

  例)

```
case 1:
{
    a = 0;
    break;  /* 違反 */
}
```

7.59. ルール 59

(必要): if, else if, else, while, do...while および for 文の本文を構成する文には必ず中カッコ{ }で括られていなければならない。

- **解釈**

  表題通りです。

  また、switch 文についても同様のルールが適用されます。

- **機能仕様**

  { }がなかった場合、「違反」レベルでレポートします。

7.60. ルール 60

(推奨): 全ての if, else if 構成は必ず最後に else 節を置くべきである。

- **解釈**

  以下の例のように解釈します。

  例)

```
if (a == 1) {
} else if (a < 1) {
}   /* ここに必ず else 文が必要です。 */

if (x < 0) {
}   /* ここには、else 文がなくてもよい。 */
```

- **機能仕様**

  else if に対応する else がない場合、「違反」レベルでレポートします。

7.61. ルール 61

(必要): switch 文の中の空でない case 節は全て break 文で終了しなければならない。

- **解釈**

  表題通りです。

- **機能仕様**

  空でない case 節の最後に break 文がない場合、「違反」レベルでレポートします。

7.62. ルール 62

(必要): switch 文は全て最後に default 節をおかなければならない。

- **解釈**

表題通りです。

- **機能仕様**

default 節が switch 文の最後にない場合、「違反」レベルでレポートします。

7.63. ルール 63

(推奨): switch 式はブール値を表現するべきではない。

- **解釈**

ブール値を表現する式とは、最後に評価される演算子が、'&&'、'||'、'!'、'<'、'>'、'<='、'>='、'=='もしくは'!='演算子の式とみなします。

以下の switch 文は、変数 a が 1 かそれ以外の値かを判別する処理になっているので、この記述も違反であると考えます。

例)

```
switch (a) {
case 1:
    break;
default:
    break;
}
```

- **機能仕様**

switch 式の最後に評価される演算子が、'&&'、'||'、'!'、'<'、'>'、'<='、'>='、'=='および'!='演算子のいずれかの場合、「違反」レベルでレポートします。switch 文に case 節1つと default 節だけがある場合、「違反」レベルでレポートします。

7.64. ルール 64

(必要): 全ての switch 文には少なくとも1つの case がなければならない。

- **解釈**

  表題通りです。

- **機能仕様**

  switch 文に case がない場合、「違反」レベルでレポートします。

7.65. ルール 65

(必要): 浮動小数点変数はループカウンタとして使用してはならない。

- **解釈**

  ループカウンタとは、該当ループを終了する条件式に使用されている変数と考えます。

- **機能仕様**

  while 文の条件式、および、for 文の第二式に浮動小数点型を持つ変数および浮動小数点型を持つメンバ変数が使用されている場合は全て、「警告」レベルでレポートします。

7.66. ルール 66 （検査対象外）

(推奨): ループ制御に関わる式のみが for 文の内部には記述されるべきである。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.67. ルール 67 （検査対象外）

(推奨): forループの中でforループをカウントする変数はループの本文中で変更するべきではない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.68. ルール 68

(必要): 関数はいつもファイルスコープで宣言しなければならない。

- **解釈**

  表題通りです。

- **機能仕様**

  ファイルスコープ以外で関数が宣言されている場合、「違反」レベルでレポートします。関数プロトタイプ宣言をせずに関数呼び出しをした場合も「違反」レベルでレポートします。

  例)

```
void main(void)
{
    func(); /*  違反  */
}
```

7.69. ルール 69

(必要): 可変個引数関数は使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  関数宣言あるいは関数定義に"..."が含まれている場合に「違反」レベルでレポートします。

  例)

```
void    func1( const SCHAR* fmt, ... );      /*  違反  */
void    func2( SINT32 i, ... )                /*  違反  */
{
}
```

7.70. ルール 70

(必要): 関数は直接的に、または間接的に再帰呼出ししてはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  直接的に再帰呼び出ししている場合のみ、「違反」レベルでレポートします。

- **制限事項**

  間接的に再帰呼び出ししている場合は検査しません。

7.71. ルール 71

(必要): 関数はいつもプロトタイプ宣言をし、プロトタイプは関数定義および関数呼出しの両方から見えなければならない。

- **解釈**

  関数が定義される前、および関数を呼び出す前にプロトタイプ宣言が必要です。これは、ルール20 にも違反します。

- **機能仕様**

  関数が定義される前、および関数を呼び出す前に、その関数のプロトタイプ宣言がない場合、「違反」レベルでレポートします。

  例)

  ```
  void func(void) {}      /* 違反 */
  ```

  これは、

  ```
  void func(void);
  void func(void) {}
  ```

  とします。

  関数を呼び出す前にプロトタイプ宣言がない場合、ルール20 とあわせてレポートします。

  __entry および__interrupt 指定された関数は、" -CHECK_No_prototype_extension"オプションを指定した場合のみルール違反としてレポートします。

7.72. ルール 72

(必要): 宣言や定義で与えられる関数の引数の型は等しくなければならず、戻り値の型も等しくなければならない。

- **解釈**

「プロトタイプ宣言に書いた引数の型および戻り型」と「関数定義の引数の型および戻り型」は完全に一致していないといけません。

- **機能仕様**

プロトタイプ宣言と関数定義の引数の型、戻り型が完全に一致していない場合、「違反」レベルでレポートします。

- **制限事項**

翻訳単位をまたがっての検査はしません。

7.73. ルール 73

(必要): 関数のプロトタイプ宣言では全ての引数となる識別子が与えられるか、または全く与えられないかのどちらかでなければならない。

- **解釈**

ここで識別子名とは、関数の引数名を意味します。このルールでは、1 つのプロトタイプ宣言中に複数の引数がある場合、「型名だけを記述した引数(例:int)」と「型名と引数名の両方を記述した引数(例:int　param)」を混在させることを禁止します。

- **機能仕様**

プロトタイプ宣言で引数名を記述した引数と、型名だけの引数が混在する場合、「違反」レベルでレポートします。

例)

```
/* 違反としてレポートする例 */
void   func1( UINT32 n, UINT32 );      /* 第一引数だけ引数名を記述している */
void   func2( UINT32, UINT32 i);       /* 第二引数だけ引数名を記述している */

/* 違反ではない例 */
void   func3( UINT32, UINT32 );        /* 第一、第二引数ともに引数名を記述していない */
void   func4( UINT32 n, UINT32 i);     /* 第一、第二引数ともに引数名を記述している */
void   func5( const UINT32* fmt, ... ); /* ...には引数名が無いので対象外 */
```

7.74. ルール 74

(必要): 引数の一部でも識別子が与えられた場合、宣言や定義で使用された識別子は等しくなければならない。

- **解釈**

プロトタイプ宣言の引数に識別子(引数名)を与えた場合、関数定義の引数の識別子と同じ名前でなくてはいけません。

- **機能仕様**

プロトタイプ宣言の識別子と関数定義の識別子の名前が異なる場合、「違反」レベルでレポートします。

7.75. ルール 75

(必要): 関数は全て明白な戻り値の型を持たなければならない。

- **解釈**

C 言語の規格では関数宣言時に戻り型として char, int, short などの型名を省略した場合、暗黙に int 型を宣言したとみなします。本ルールは、このような暗黙の戻り型の宣言を違反とみなします。

- **機能仕様**

関数宣言や関数定義、および関数型キャストに戻り型が無い場合は「違反」レベルでレポートします。

以下のように型名は記述していないが, 型修飾子(const や volatile など)や記憶クラス(static や extern)のみを記述している形式は, 明白に型を指定していないとみなして「違反」レベルでレポートします。

例)

```
/* 違反になる例 */
func1() {  }                    /* 戻り型は暗黙で int になります */
const   func2() {  }            /* 戻り型は暗黙で const int になります */
const   func3();                /* 戻り型は暗黙で const int になります */
static   func4();               /* 戻り型は暗黙で int になります */
```

7.76. ルール 76

(必要): 引数を持たない関数は、引数型を void としなければならない。

- **解釈**

引数を持たない関数とは、"void func();" のように引数リストが空の関数型を意味します。コンパイラはこのような関数の引数の型を検査しないので、このルールでは引数に void を記述しなくてはいけないとしています。

- **機能仕様**

関数宣言、および関数型を含むキャストに引数リストが無い場合は「違反」レベルでレポートします。

7.77. ルール 77

(必要): 関数に引き渡される引数の非修飾型は関数プロトタイプで定義された期待する非修飾型と互換性がなければならない。

- **解釈**

このルールは、関数の呼び出し時に指定された引数の型と、プロトタイプ宣言で定義された引数の型の非修飾型(const,および volatile を取り除いたもの)とが一致することを要求しているとみなします。つまり、暗黙の型変換が発生する引数は違反とみなします。

- **機能仕様**

関数の呼び出し時に指定された引数の型と、プロトタイプ宣言で定義された引数の型を比較して、修飾子(const および volatail)を除いた型が一致しない場合、「違反」レベルでレポートします。

例)

```
void    func(const SLONG);
void    xxx(void)
{
    SINT32    i;
    SLONG     l;
    func( l );          /*  違反ではない  */
    func( i );          /*  違反(int と signed long は一致しない) */
}
```

7.78. ルール 78

(必要): 関数に引き渡される引数の数は関数プロトタイプと一致していなければならない。

- **解釈**

  表題通りです。

- **機能仕様**

  関数呼び出しに記述された引数の数と、プロトタイプ宣言の引数の数が一致していなければ、「違反」レベルでレポートします。ただし、これは ISO9899:1990 に準拠した C コンパイラではコンパイルエラーになります。

- **注意事項**

  拡張機能であるデフォルト引数を利用した関数呼び出しの場合、このルールに違反しているとみなします。

7.79. ルール 79

(必要): void 関数が返した値は使用してはならない。

- **解釈**

  表題通りです。

  関数の戻り型が void であるにも係わらず、その関数の戻り値を使用するような式を書いてはいけません。同一の翻訳単位で、その関数呼び出しの前にプロトタイプ宣言を書いていればコンパイルエラーになります。しかしプロトタイプ宣言が無い場合は、その関数の戻り型が void 型であることがコンパイラには分からないため、戻り型が int 型の関数であるかのような実行コードを生成する可能性があります。

- **機能仕様**

  戻り値の型が void 型の関数の呼び出しを、部分式に使用した場合に、「違反」レベルでレポートします。
  ただし、これは ISO9899:1990 に準拠した C コンパイラではコンパイルエラーになります。

7.80. ルール 80

(必要): void 式は関数の引数として引き渡してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  戻り値の型が void 型の関数の呼び出しを別の関数の引数として使用した場合や、void* p;と宣言した変数を func(*p)として引数に渡した場合に、「違反」レベルでレポートします。ただし、これは ISO9899:1990 に準拠した

C コンパイラではコンパイルエラーになります。また、ルール 79 にも違反しています。

7.81. ルール 81 （検査対象外）

(推奨): その関数が決して変数を書き換えないことを意図する場合は、参照渡しの関数の引数に const 修飾子を使用すべきである。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.82. ルール 82

(推奨): 関数は1つの出口しか持ってはならない。

- **解釈**

  関数に戻り値がある場合、return 文は関数の最後に1つだけ持つことができます。関数の戻り型が void の場合、関数内に return 文を1つも持たないか、あるいは関数の最後に1つだけ持つことができます。

- **機能仕様**

  (1) 関数に戻り値がある場合

  return 文が複数ある場合、あるいは return 文が関数の最後にない場合、「違反」レベルでレポートします。

  (2) 関数の戻り型が void の場合

  return 文が複数ある場合、あるいは関数の最後以外に return 文があった場合、「違反」レベルでレポートします。

7.83. ルール 83

(必要): 非 void の戻り値を持つ関数は、ⅰ)(プログラムの最後を含む)全ての出口について1つの return 文を持たなければならない、ⅱ)各 return は式を持たなければならない、ⅲ)式は戻り型に適合しなければならない。

- **解釈**

関数に戻り型がある場合、return 文は式を持ち、その型が戻り型と一致していなくてはいけません。また、関数の最後の return 文の後に文があってはいけません。

- **機能仕様**

return 文が関数の最後にない場合、「違反」レベルでレポートします。

return 文に戻り値がない場合、「違反」レベルでレポートします。

return 文の戻り値が戻り型と一致いない場合、「違反」レベルでレポートします。

7.84. ルール 84

(必要): void の戻り値を持つ関数では、return 文が式をもってはならない。

- **解釈**

表題通りです。

- **機能仕様**

return 文に戻り値がある場合、「違反」レベルでレポートします。

7.85. ルール 85

(推奨): 引数なしで呼び出された関数は空のカッコを持つべきである。

- **解釈**

関数名を( )なしで記述すると、関数のアドレスを表します。このような表記は、関数呼び出しを意図して記述を間違えたのか、アドレスの参照を意図したものかの区別ができません。この問題を防ぐために関数のアドレスを参照する場合は明示的に単項演算子&を使用することを推奨しているとみなします。

また、関数名に間接演算子(*)を作用させた式(*func)も、(func のみと)同様に違反とみなします。ただし、このような式が関数呼び出しに使用される場合(つまり、(*func)())は、違反しているとはみなしません。

- **機能仕様**

式中に関数名が含まれているとき、単項演算子&も、呼び出しのための( )演算子も無い場合に「違反」レベルでレポートします。

ただし、式が関数型へのポインタ変数の初期化子(初期化の初期値)として使用されている場合には、これを呼び出しとして使用することは ISO9899:1990 の仕様上できないことになっていますので、違反とはみなしません。

7.86. ルール 86 （検査対象外）

(推奨): 関数がエラー情報を戻り値として戻す場合、エラー情報はテストされるべきである。

- **解釈**

表題通りです。

- **機能仕様**

このルールは検査しません。

7.87. ルール 87 （検査対象外）

(必要): 前処理命令とコメントだけがファイル中の#include 文より前置されていることが許される。

- **解釈**

表題通りです。

- **機能仕様**

このルールは検査しません。

7.88. ルール 88 （検査対象外）

(必要)： 非標準文字は#include 指令のヘッダファイル名に現れてはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.89. ルール 89 （検査対象外）

(必要)： #include 指令は、<ファイル名>か”ファイル名”が続かなければならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.90. ルール 90 （検査対象外）

(必要)： C言語のマクロはシンボリック定数、関数マクロ、型修飾子、あるいは記憶クラス指定子としてのみ使用されなければならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.91. ルール 91 （検査対象外）

(必要)： マクロはブロック内で、定義される(#define)と同時に定義を解除(#undef)されてはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.92. ルール 92 （検査対象外）

(推奨)： #undef は使用すべきではない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.93. ルール 93 （検査対象外）

(推奨)： 関数形式マクロよりは、関数を使用すべきである。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.94. ルール 94 （検査対象外）

(必要)： 引数の全てを指定せずに関数マクロを’呼び出し’してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.95. ルール 95 （検査対象外）

(必要)： 関数マクロの引数は前処理指令と似ているトークン(字句)を含んではならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.96. ルール 96 （検査対象外）

(必要): 関数マクロの定義において、定義全体とパラメータのそれぞれのインスタンスにはカッコをつけなければならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.97. ルール 97 （検査対象外）

(推奨): 前処理指令内の識別子は使用する前に定義するべきである。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.98. ルール 98 （検査対象外）

(必要): 単一マクロ定義の中には前処理演算子#と##の出現はどちらかが高々1回でなければならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.99. ルール 99

(必要): #pragma 指令の使用は全て文書にし、説明しなければならない。

- **解釈**

  表題の通りです。

- **機能仕様**

  #pragma が使用された場合、「警告」レベルでレポートします。

7.100. ルール 100 （検査対象外）

(必要): 前処理演算子 defined は 2 つの標準形式のうちどちらかのみを使用しなければならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.101. ルール 101

(推奨): ポインタ算術は使用するべきではない。

- **解釈**

  ポインタを++演算もしくは、--演算した結果のアドレスにアクセスしたり、[ ]を使ってポインタを配列としてアクセスしたりすることには問題がないとみなします。しかし、それ以外のポインタ計算によって得られたアドレスへのアクセスは、不正アクセスになる可能性がより高いと考えられます。このルールではこのようなポインタ演算を禁止しています。

- **機能仕様**

  ++および--以外のポインタ演算があれば「違反」レベルでレポートします。ただし、!演算子の右辺式やポインタに[ ]演算子を使用する場合は、検査の対象ではありません。

  例)

```
SINT32 a[10];
SINT32 *p = &(a[0]);
p++;                    /*  違反ではない  */
p[4] = 1;               /*  違反ではない  */
*(p + 5) = 0;           /*  違反  */
```

7.102. ルール 102

(推奨): 2 段階を超えるポインタ指定は使用するべきではない。

- **解釈**

  表題通りです。

- **機能仕様**

  - 3 段以上のポインタ変数を宣言した場合に「違反」レベルでレポートします。
  - 3 段以上のポインタ型でキャストした場合に「違反」レベルでレポートします。
  - 同時に 3 回以上間接演算子(*)を使用した場合(***p など)に「違反」レベルでレポートします。

7.103. ルール 103

(必要): 2 つの被演算子が同じ型で、同じ配列や構造体、共用体を指し示す場合を除いては、関係演算子はポインタには適用してはならない。

- **解釈**

  ポインタ同士の大小比較をするとき、比較するポインタが指している先はどちらも同じ配列の中、同じ構造体オブジェクトの中、同じ共用体の中のいずれでもない場合の比較結果は未定義とされています。このため、「同じ配列や構造体、共用体」というのは、同じ配列オブジェクト、同じ構造体オブジェクト、あるいは、同じ共用体オブジェクトのことを指しているとします。

  関係演算子とは、<、>、<=、>=のことであり、==と!=は含みません。

- **機能仕様**

  ポインタ同士を比較する関係演算子が記述された場合は、常に「警告」レベルでレポートします。レポートされた記述が常にルール 103 を違反しているとは限りません。同じ配列オブジェクト、同じ構造体オブジェクト、あるいは、同じ共用体オブジェクトを指している場合も、レポートされます。

  例)

```
SINT32* pi1, pi2;
pi1 > pi2;      /*  異なるオブジェクトに対するポインタを比較してはいけません  */
```

7.104. ルール 104

(必要): 関数への非定数のポインタは使用してはならない。

- **解釈**

このルールは、アドレスがコンパイル時に決まらない関数へのポインタを、呼び出し以外の式の一部として使用すること、および関数へのポインタでキャストしたりすることを禁止しているとみなします。

- **機能仕様**

関数へのポインタ変数の宣言が、定数式あるいは関数名で初期化されていなければ「違反」レベルでレポートします。また、関数へのポインタによるキャスト、および、関数へのポインタへのキャストがある場合にも「違反」レベルでレポートします。

例)

```
void func(void)
{
    /*  関数へのポインタ fp1 を、アドレスがコンパイル時に決まる関数へのポインタで
        初期化しているので違反ではありません  */
    void (*fp1)(void) = &func;

    /*  アドレスがコンパイル時に決まらない関数へのポインタで初期化すると違反です  */
    void (*fp2)(void) = fp1;
}
```

7.105. ルール 105

(必要): 単一のポインタが指している全ての関数は全て引数の数、および型、戻り値の型が同じでなければならない。

- **解釈**

以下の場合にそれぞれが関数へのポインタのとき、その二つのポインタが指す関数型は一致しているべきとみなします。ここで関数型の一致とは、引数の数、対応する引数の型、および、戻り値の型が一致していることです。

- 代入元のポインタと代入先のポインタ
- プロトタイプ宣言の仮引数と呼び出し時の実引数
- 関数定義の戻り型と return 文の戻り値の型
- 関数へのポインタ宣言とその初期化式

- **機能仕様**

上記の解釈において関数型が一致していない場合、「違反」レベルでレポートします。ただし、ISO9899:1990 に準拠した C コンパイラではコンパイルエラーになります。

例)

```
SINT32   func1(SINT32);
SINT32   func2();
SINT32   (fp*)(SINT32);

fp = &func1;            /*  違反ではない  */
fp = &func2;            /*  違反  */
```

7.106. ルール 106

(必要): 自動領域にあるオブジェクトのアドレスをオブジェクトの存在が終了した後にまで持続期間をもつオブジェクトに代入してはならない。

- **解釈**

ローカルスコープの変数のアドレス値が、そのスコープの外側のスコープにあるポインタ変数に代入されたり、戻り値としてスコープ外で使用されるのを防ぐのが目的と考えます。

- **機能仕様**

自動変数のアドレス値を、そのスコープより外側のスコープにあるポインタ変数に代入する場合、「違反」レベルでレポートします。また、自動変数のアドレスを戻り値とする return 文があれば、「違反」レベルでレポートします。

- **制限事項**

自動変数のアドレスを別の自動変数に代入した後、外側のスコープにあるポインタに代入する場合や、それを戻り値とする return 文がある場合のように、間接的に代入する場合については検査しません。

また、ポインタが指す先に自動変数のアドレスを代入する場合も、このルールを検査しません。

例)

```
SINT32** pp;
SINT32* p;
void func(void)
{
    SINT32    a;
    SINT32*   pa = &a;

    p = &a;                    /* 違反 */
    *pp = &pa;                 /* 検査しません */
}
```

7.107. ルール 107 （検査対象外）

(必要): ヌルポインタが指す先を参照してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.108. ルール 108

(必要): 構造体や共用体の記述は、構造体や共用体のメンバを全て指定しなければならない。

- **解釈**

  ここで「メンバを全て指定する」とは、構造体、共用体の定義時に、そのメンバに名前（変数名）を付けることを意味します。

  ビットフィールドの宣言では、「メンバに名前が無い、パディングだけからなる構造体」を宣言することができます。しかしその構造体変数を参照や変更した場合の動作は未定義です。本ルールはこの未定義動作を防ぐことにあると解釈します。

  例）

```
struct  S {
        unsigned  int  :1;
        unsigned  int  :1;
        unsigned  int  :1;
        unsigned  int  :1;
} s;                    /* 以降、この変数 s に対する参照や変更の結果は未定義 */
```

- **機能仕様**

  構造体および共用体中に名前を持つメンバが 1 つも無い場合に「違反」レベルでレポートします。

7.109. ルール 109 （検査対象外）

(必要): 変数領域の重複は使用してはならない。

- **解釈**

表題通りです。

- **機能仕様**

このルールは検査しません。

7.110. ルール 110

(必要): より大きなデータ型の一部分にアクセスするために共用体を使用してはならない。

- **解釈**

以下の条件を同時に満たす場合に、「より大きなデータ型の一部分にアクセスするため」に共用体を使用するとみなします。

- 共用体に含まれる全メンバが同一サイズである。
- その共用体定義中に、1 個以上の構造体または配列を含む。

ただし共用体中の一部分に同一サイズのメンバが含まれている場合でも、全メンバが同一サイズでない場合は「より大きなデータ型の一部分にアクセスするため」に共用体を使用するとみなしません。

- **機能仕様**

上記の解釈において「より大きなデータ型の一部分にアクセスするため」に共用体を使用するとみなす場合、「違反」レベルでレポートします。

例1）

```
/* arr と ul が同じサイズの領域を使うので違反 */
union U1 {
    unsigned  char  arr[4];
    unsigned  long  ul;
};
```

例2）

```
/* bits と all,other が同じサイズの領域を使うので違反 */
union U2 {
    struct {
        unsigned  char  b0:1;
        unsigned  char  b1:1;
```

```
        unsigned  char  b2:1;
        unsigned  char  b3:1;
        unsigned  char  b4:1;
        unsigned  char  b5:1;
        unsigned  char  b6:1;
        unsigned  char  b7:1;
    }  bits;
    unsigned  char  all;
    struct {
        unsigned  char  c1:4;
        unsigned  char  c2:4;
    } other;
};
```

例3）

```
/* arr と ul のサイズが異なるので違反ではない  */
union U3 {
    unsigned  char  arr[2];
    unsigned  long  ul;
};
```

7.111. ルール 111

(必要)： ビットフィールドは、unsigned int 型か signed int 型に対してのみ定義しなければならない。

- **解釈**

表題通りです。

- **機能仕様**

unsigned int型、signed int型のどちらでもないビットフィールドを記述した場合に「違反」レベルでレポートします。

例）

```
struct S {
    unsigned char    b0 : 1;      /*  違反  */
    int              b1 : 1;      /*  違反  */
    signed  int      b2: 2;       /*  違反ではない  */
};
```

7.112. ルール 112

(必要): signed int 型のビットフィールドの幅は 2 ビット以上でなければならない。

- **解釈**

符号付きの型のビットフィールドの幅は 2 ビット以上でなければならないと解釈します。

ルール 111 では、ビットフィールド記述できる型として unsigned int 型、signed int 型に限定していますが、処理系依存で unsigned char 型や signed char 型を使用できるコンパイラもあります。安全性を高めるため、符号ビットのみのビットフィールドは許可しないと考えるならば、符号付きの型すべてに対して検査するほうが好ましいため、符号付きの型すべてに対してレポートします。

- **機能仕様**

1 ビット幅のビットフィールドを、符号付きの型で記述した場合に「違反」レベルでレポートします。

7.113. ルール 113

(必要): 構造体(または共用体)のメンバには全て名前をつけ、その名前を経由してのみアクセスしなければならない。

- **解釈**

構造体(または共用体)に名前が無いメンバを記述した場合に、このルールに違反するとみなします。ただし ISO9899:1990 に従った C 言語では、パディングとしてビットフィールドを記述するときのみ名前が無いメンバを使用できます。このため、ビットフィールドのパディングに名前が無い場合のみこのルールに違反するとみなします。

構造体(または共用体)のメンバのアドレスを取ると、名前を経由しないでアクセスできる可能性があるため、このルールに違反するとみなします。

- **機能仕様**

構造体および共用体の定義時に、名前が無いメンバを記述した場合に「違反」レベルでレポートします。ただしビット幅がゼロのビットフィールドは、ISO9899:1990 で「名前を持ってはいけない」と規定されているのでレポートしません。

また、構造体および共用体のメンバのアドレスをとった場合に「違反」レベルでレポートします。

7.114. ルール 114 （検査対象外）

(必要): 予約語や標準ライブラリ関数名は再定義してはならないし、定義を解消してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.115. ルール 115

(必要): 標準ライブラリ関数名は再使用してはならない。

- **解釈**

  標準ライブラリ関数名と同じ関数名の関数を定義してはいけません。

- **機能仕様**

  標準ライブラリ関数名と同じ関数名の関数定義があった場合、「違反」レベルでレポートします。

7.116. ルール 116 （検査対象外）

(必要): 生成コードで使用するライブラリは全て、この文書の示す規定に従って書かれていなければならず、適当な評価を受けていなければならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.117. ルール 117 （検査対象外）

(必要): ライブラリ関数に引き渡される値の正当性はチェックされなければならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.118. ルール 118

(必要): 動的なヒープ領域割り当ては使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  calloc()、malloc()、realloc()、および、free()関数の呼び出しがあれば、「違反」レベルでレポートします。

7.119. ルール 119

(必要): エラー指示子 errno を使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  errno が使用されていれば、「違反」レベルでレポートします。

7.120. ルール 120 （検査対象外）

(必要): ライブラリ<stddef.h>で定義されているマクロ offsetof は使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  このルールは検査しません。

7.121. ルール 121

(必要): <locale.h>と setlocale 関数は使用してはならない。

- **解釈**

  表題通りです。

  また、localeconv 関数も使用してはいけません。

- **機能仕様**

  setlocale()、localeconv()関数の呼び出しがあれば、「違反」レベルでレポートします。

7.122. ルール 122

(必要): setjmp マクロおよび longjmp 関数は使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  setjmp()、および、longjmp()関数の呼び出しがあれば、「違反」レベルでレポートします。

7.123. ルール 123

(必要): <signal.h>にあるシグナル操作は使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  signal()、および、raise()関数の呼び出しがあれば、「違反」レベルでレポートします。

7.124. ルール 124

(必要): 入出力ライブラリ<stdio.h>は製品用コードでは使用してはならない。

- **解釈**

  stdio.h をインクルードしてはいけません。

- **機能仕様**

  stdio.h がインクルードされた場合、「違反」レベルでレポートします。ただし、stdio.h をインクルードせずに、stdio.h の関数を使用した場合は、ルール71でレポートします。

7.125. ルール 125

(必要): ライブラリ<stdlib.h>のライブラリ関数 atof()、atoi()、および atol()は使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  atof()、atoi()および、atol()関数の呼び出しがあれば、「違反」レベルでレポートします。

- **制限事項**

  atof()、atoi()および、atol()関数が stdlib.h にマクロで定義されている場合は検査できません。

7.126. ルール 126

(必要): ライブラリ<stdlib.h>のライブラリ関数 abort()、exit()、getenv()および system()は使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  abort()、exit()、getenv()および、system()関数の呼び出しがあれば、「違反」レベルでレポートします。

7.127. ルール 127

(必要): ライブラリ<time.h>の時間操作関数は使用してはならない。

- **解釈**

  表題通りです。

- **機能仕様**

  clock()、difftime()、mktime()、time()、asctime()、ctime()、gmtime()、localtime()および、strftime()関数の呼び出しがあれば、「違反」レベルでレポートします。

8. 付録　マージユーティリティ SQMmerger

SQMmergerは、SQMlintで生成したレポートファイル(CSV形式)とCソースファイルから、Cソース行と対応するレポートメッセージの混合表示ファイルを作成するツールです。

8.1. SQMmergerの処理概要

8.1.1. SQMmergerの概要

SQMmerger は、C 言語ソースファイルと SQMlint で作成したレポートファイルから、C ソース行と対応するレポートメッセージの混合表示ファイルを作成します。SQMmerger の入出力ファイルの関係を図 8.1.1.1に示します。

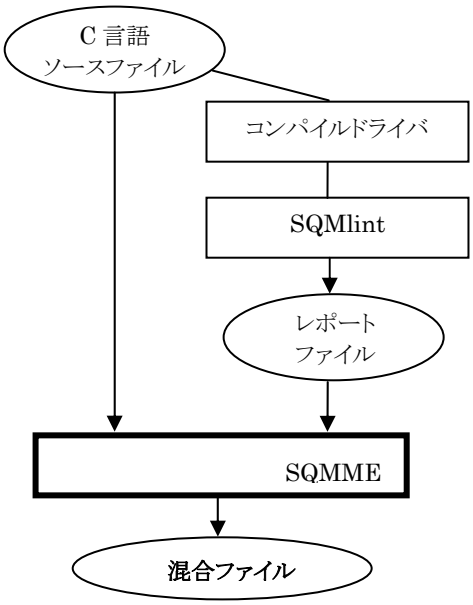

図 8.1.1.1 SQMmerger の入出力

混合表示ファイル出力例）

```
1 : void  func(void);
2 : void  func(void)
3 : {
4 : LABEL:
[MISRA(55) Complaining] label (LABEL) should not be used

5 :
6 :        goto LABEL;
[MISRA(56) Complaining] the 'goto' statement shall not be used

7 : }
```

8.2. SQMmerger の使い方

8.2.1. SQMmerger コマンドの入力書式

SQMmerger 起動コマンドのコマンド行の入力書式および規則を以下に示します。起動オプションの詳細は、後述の 8.2.2 節「SQMmerger の起動オプション」を参照してください。

```
sqmmerger   -src C ソースファイル名  -r  レポートファイル名  -o  混合表示出力ファイル名
                [-v]   [-V]  [-h]

※[ ]で囲まれていない項目：必ず入力しなければならない項目
※[ ]で囲まれた項目：必要に応じて入力する項目
```

起動例) SQMmerger　-src test.c -r test.csv -o test.vi

8.2.2. SQMmerger の起動オプション

SQMmerger の起動オプションと機能について示します。

表 8.2.2.1　SQMmerger の起動オプション

オプション	機能
-src C ソースファイル名	SQMlint で検査対象にした C ソースファイル名を指定します。
-r レポートファイル名	SQMlint で生成したレポートファイル名を指定します。
-o 混合表示出力ファイル名	SQMmerger で生成する混合表示出力ファイル名を指定します。
-v	実行中のコマンドプログラム名及びコマンドラインを表示します。
-V	起動時メッセージを表示し、SQMmerger の実行を終了します。
-h	SQMmerger のコマンドラインオプションを表示します。

8.3. SQMmerger で出力される混合表示ファイルの仕様

8.3.1. C ソース行の出力形式

C ソース行の先頭に行番号と':'を付加して出力します。

例)

```
1 : typedef signed int INT;
2 : INT  glb;
```

8.3.2. MISRA C の検査結果の出力形式

検査結果の出力形式には、2 通りあります。

(1)　検査結果のレポートメッセージが、-src オプションで指定したソースファイル名に対するものであった場合 MISRA C のルール番号とレポートメッセージ内容が出力されます。

例)

```
4 : LABEL:
[MISRA(55) Complaining] label (LABEL) should not be used
```

(2) 検査結果のレポートメッセージが、-src オプションで指定したソースファイル名と異なるソースファイル名に対するものであった場合。(これは、ソースファイル中でインクルードされたヘッダファイル等のことです。)
MISRA C のルール番号、エラーの発生したソースファイル名と行番号、レポートメッセージ内容が出力されます。

例)

```
[MISRA(124) Complaining:stdio.h,line 10] the 'stdio.h' library shall not
                be used in production code
```

検査結果のレポートメッセージ(エラー)が、-src オプションで指定したソースファイルと異なるソースファイルに対するものであった場合、それらのレポートメッセージは、混合表示ファイルの先頭に出力されます。

例)

```
[MISRA(8) Complaining:locale.h,line 15] multibyte characters shall not be
used

1 : #include <locale.h>
2 :
3 : void comp1(void);
4 : void comp1(void)
5 : {
6 :       setlocale(0,0);
[MISRA(68) Complaining] function (setlocale) shall always be declared at file
scope
[MISRA(20) Complaining] (setlocale) has not been declared yet
[MISRA(71) Complaining] (setlocale) has not been declared as prototype
[MISRA(121) Complaining] a (setlocale) function has been used

7 : }
```

インクルードされたファイルに対するレポートメッセージはファイルの先頭に出力されます。

9. 付録　フォーマット変換ユーティリティ SQMform

SQMform は、SQMlint で生成した複数のレポートファイル(CSV 形式)を、1 つの CSV ファイルにまとめる、およびテキストエディタのタグジャンプフォーマットに変換するためのツールです。

9.1. 概要

SQMform は SQMlint が生成した複数の CSV ファイルを読み、連結して出力します。このとき「完全に重複するメッセージ」は 1 つにまとめるため、同一ヘッダファイルを複数の C ソースでインクルードするときに出力される、複数の同一メッセージを集約して表示することができます。また出力ファイルのフォーマットを指定することで、テキストエディタ用のタグジャンプ形式に変換することもできます。

入出力ファイルの関係を図 9.1.1 に示します。

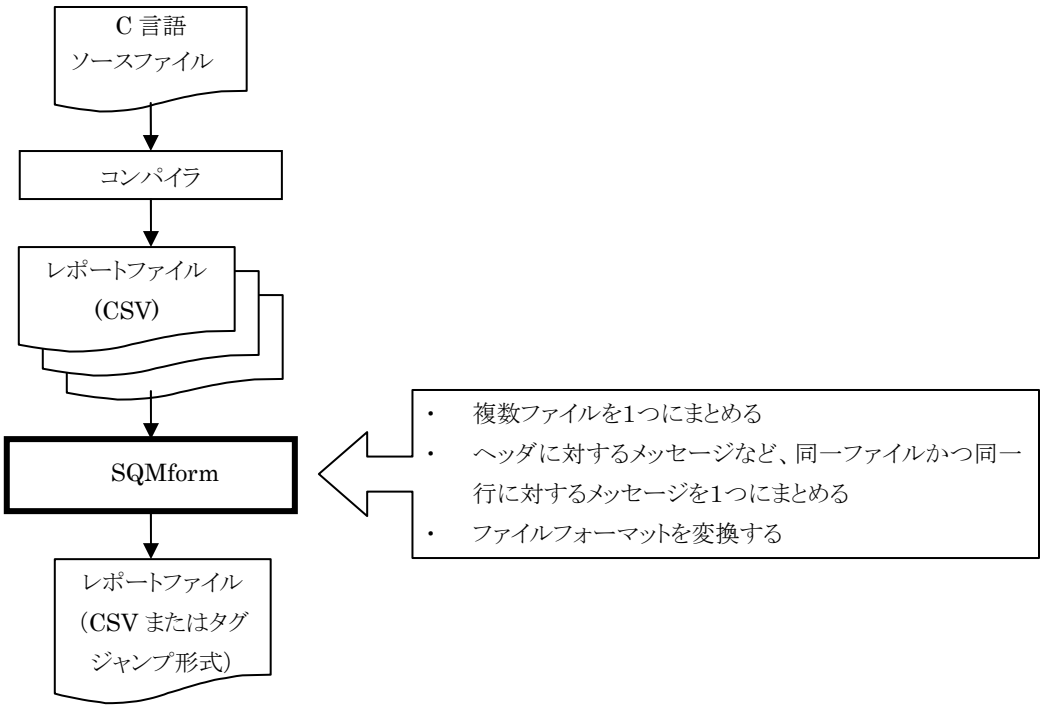

図 9.1.1 SQMform の入出力ファイル

9.2. 使い方

9.2.1. コマンドの入力書式

コマンド行の入力書式および規則を以下に示します。起動オプションの詳細は、後述の 9.2.2 節「起動オプション」を参照してください。

```
sqmform△[オプション]△入力ファイル[△入力ファイル...]

※△                          :空白文字またはタブ文字
※[ ]で囲まれていない項目:必ず入力しなければならない項目
※[ ]で囲まれた項目          :必要に応じて入力する項目
```

起動例) SQMform - verbose -o summary.csv test1.csv test2.csv test3.csv

9.2.2. 起動オプション

起動オプションと機能について示します。

表 9.2.2.1 起動オプションの一覧

オプション	機能
-o△出力ファイル名	出力結果として生成するファイルを指定します。 ● このオプションを指定しない場合は、最初の入力ファイル名の拡張子を".msg"に置換した名前を出力ファイル名とします。 ● 出力ファイル名と同一のファイルが既に存在する場合は、既存のファイルを上書きします。
-form△{tag1|tag2|tag3|tag4|csv}	変換するフォーマットを指定します。 (1) tag1 形式:ファイル名(行番号)△内容 (2) tag2 形式:ファイル名△行番号:内容 (3) tag3 形式:ファイル名△行番号△内容 (4) tag4 形式:ファイル名:行番号:内容 (5) CSV 形式:SQMlint の出力形式と同一です。 ● このオプションを指定しない場合は、"-form tag1"を指定した時と同じ動作をします。 ● tag1、tag2、tag3、tag4 はテキストエディタ用のタグジャンプ形式です。使用されるテキストエディタに合った形式を選択してください。CSV 形式は、SQMlint の出力形式と同一です。SQMlint の出力ファイルをまとめるときに使用してください。 ● tag1、tag2、tag3、tag4 を指定すると、入力ファイルに含まれているパス区切り文字が"/"の場合は"¥"に置換して出力します。これはタグジャンプ用のフォーマットとして、パス区切り文字が"/"ではタグジャンプできないことがあるため、これに対応するための変換です。
- verbose	フォーマット変換と同時に、標準出力に次の情報を表示します。 ● 開始時間 ● 終了時間
-summary	フォーマット変換と同時に、標準出力に次の情報を表示します。 ● 出力ファイルに書き込んだMISRA Cルール番号と、その番号でメッセージを出した回数。ただしメッセージ内容、発生ファイル名および発生行番号がすべて同じ場合、複数メッセージをまとめて 1 回と数えます。
- full_path	入力ファイル中のパス文字列が相対パスの場合、出力ファイル中のパス文字列には**カレントディレクトリ**を追加して絶対パスに変更します。 **使用するテキストエディタのタグジャンプ機能が、絶対パスを必要とする場合はこのオプションを使用してください。**
-V	バージョン番号を表示する。出力ファイルは生成しない。

ルネサスコンパイラ用拡張機能
ユーザーズマニュアル
MISRA C ルールチェッカ SQMlint

発行年月日　2006年8月1日　　Rev.1.00

発行　　株式会社 ルネサス テクノロジ　営業企画統括部
　　〒100-0004　東京都千代田区大手町2-6-2

編集　　株式会社 ルネサス ソリューションズ　ツール開発部

MISRA C ルールチェッカ　SQMlint V.1.03
ユーザーズマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ10J1600-0100