FUJITSU Software
Agile^{+} Relief C/C++ V1.1.1

Manual for IPA/SEC-C/C++ Check

B1WD-3608-01ENZ0(00)
January 2023

Preface

IPA/SEC-C/C++ check of Agile+ Relief C/C++ (hereinafter called Agile+ Relief) is an application program for IPA/SEC-C/C++ coding verification by analyzing the source program and include file in C/C++ language.

This document specifies the IPA/SEC-C/C++ coding violation issued in Agile+ Relief.

About the message output for program fault in Agile+ Relief, please refer to "Message Indications Manual".

About the error message output in Agile+ Relief, please refer to "Command Manual".

References

[Embedded System development Coding Reference guide [C Language Edition]] - Software Engineering Center, Technology Headquarters, Information-technology Promotion Agency, Japan.

[Embedded System development Coding Reference guide [C++ Language Edition]] - Software Engineering Center, Technology Headquarters, Information-technology Promotion Agency, Japan.

Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States and other countries.

MISRA and its logo are registered trademarks of MIRA Ltd, held on behalf of the MISRA Consortium.

CERT is registered trademarks of Carnegie Mellon University in the United States.

The names of other products and services referred to in this document are trademarks of their respective developers and manufacturers.

particular technologies disclosed in this document may be subject to the Foreign Exchange and Foreign Trade Control Law. Export of this document or any part thereof, or supply of this document or any part thereof to a foreign resident, must be undertaken only in compliance with the provisions of the law.

FUJITSU LIMITED

Note

- Transmission or copying of this document in whole or in part is prohibited.
- The content of this document is subject to change without prior notice.

Contents

1. GUI for IPA/SEC-C/C++ Coding Check

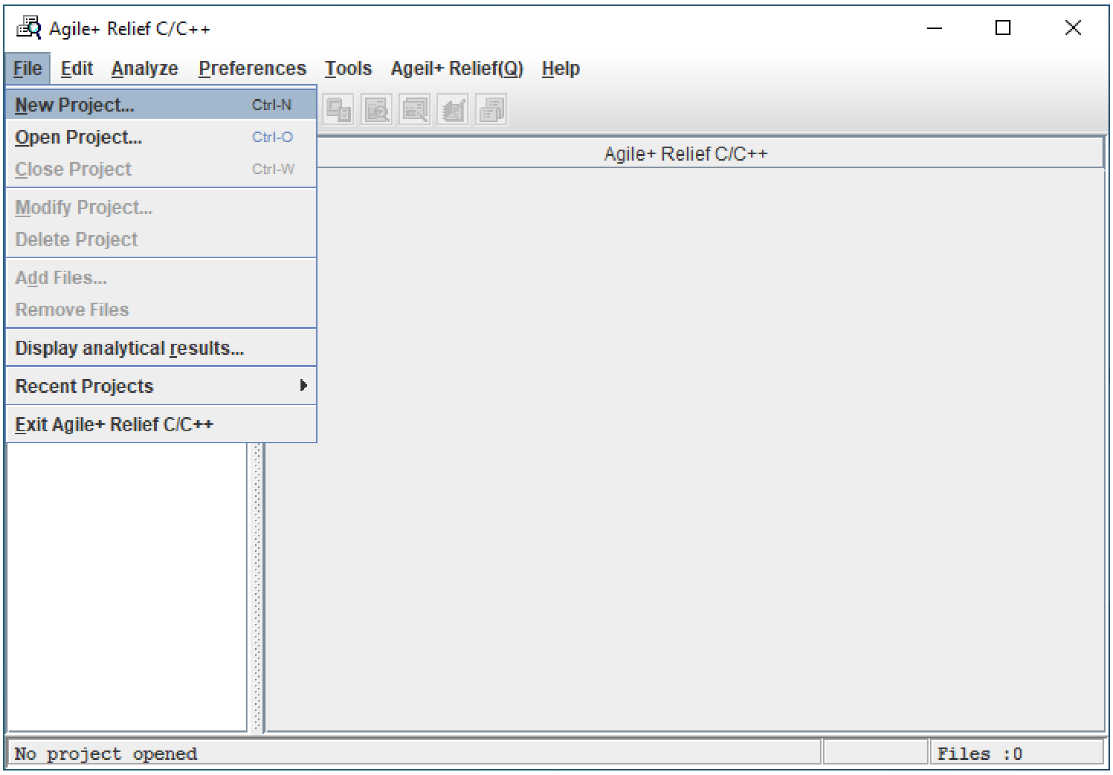
This section specifies how to check IPA/SEC-C/C++ coding through GUI.

1.1 Create Project

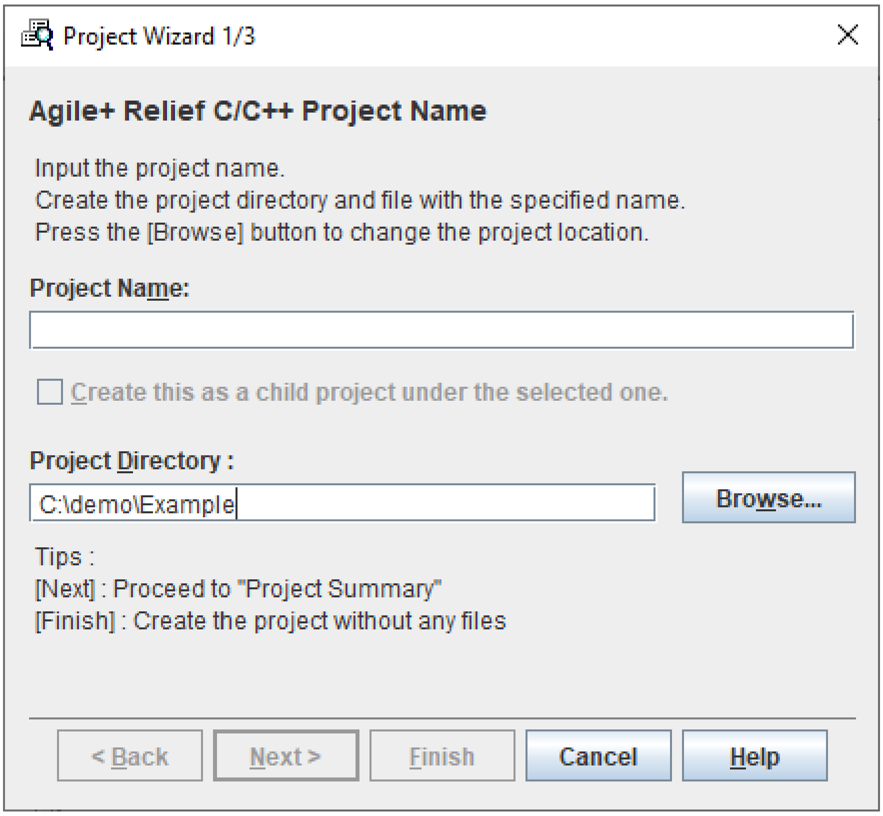
First, create the desired project in Agile$^+$ Relief for use.

In Agile$^+$ Relief, the management of source file in the unit of project is required for the analysis of source file (source program) in C/C++.

Select [New Project…] from [File] menu in [Main Window].



[Project Wizard 1/3] dialog box is displayed.



Project Wizard 1/3

Agile+ Relief C/C++ Project Name

Input the project name.
Create the project directory and file with the specified name.
Press the [Browse] button to change the project location.

Project Name:

□ Create this as a child project under the selected one.

Project Directory :

C:\demo\Example

Tips :
[Next] : Proceed to "Project Summary"
[Finish] : Create the project without any files

Please enter [sample] for project name.

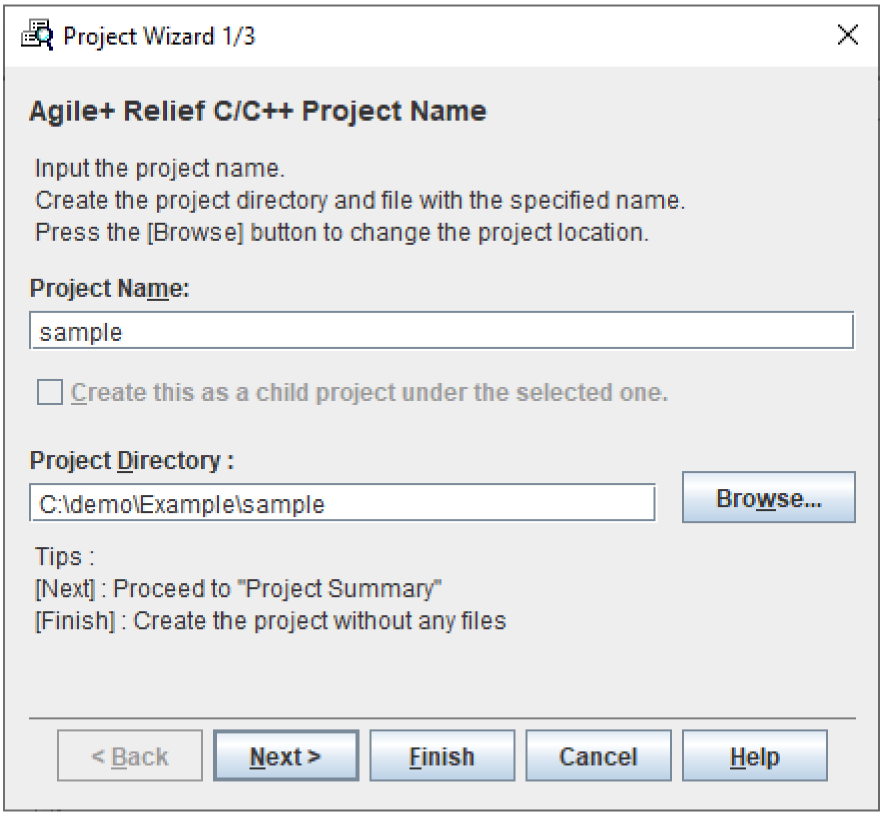
Please enter the project name with half-width numbers and English letters only. In addition, do not spell the project name in Japanese.

The option of [Create this as a child project under the selected one] is not available here.

Entry of the desired directory is required for project saving. In this Example, please enter [C:\demo\Example].

When a project name is entered with the directory name already existing, it will be automatically added to the end of directory.

After the above settings are done, the dialog box is displayed as follows:



Project Wizard 1/3

Agile+ Relief C/C++ Project Name

Input the project name.
Create the project directory and file with the specified name.
Press the [Browse] button to change the project location.

Project Name: sample

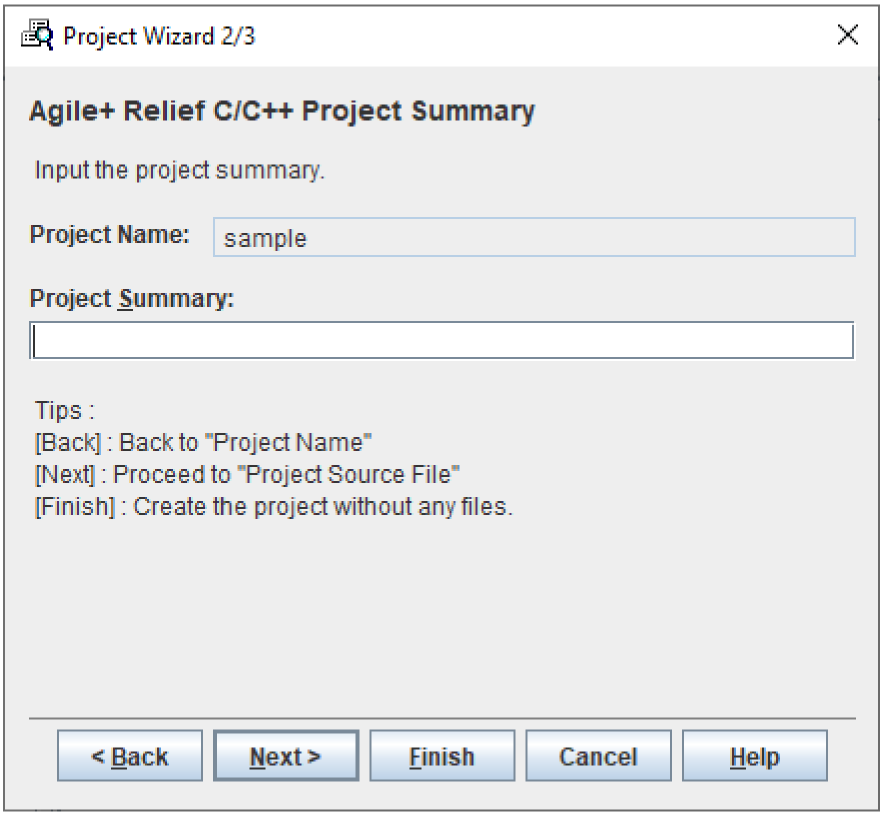
□ Create this as a child project under the selected one.

Project Directory : C:\demo\Example\sample

Tips :
[Next] : Proceed to "Project Summary"
[Finish] : Create the project without any files

Choose [Next].

[Project Wizard 2/3] dialog box is displayed.



Project Wizard 2/3

Agile+ Relief C/C++ Project Summary

Input the project summary.

Project Name: sample

Project Summary:

Tips :
[Back] : Back to "Project Name"
[Next] : Proceed to "Project Source File"
[Finish] : Create the project without any files.

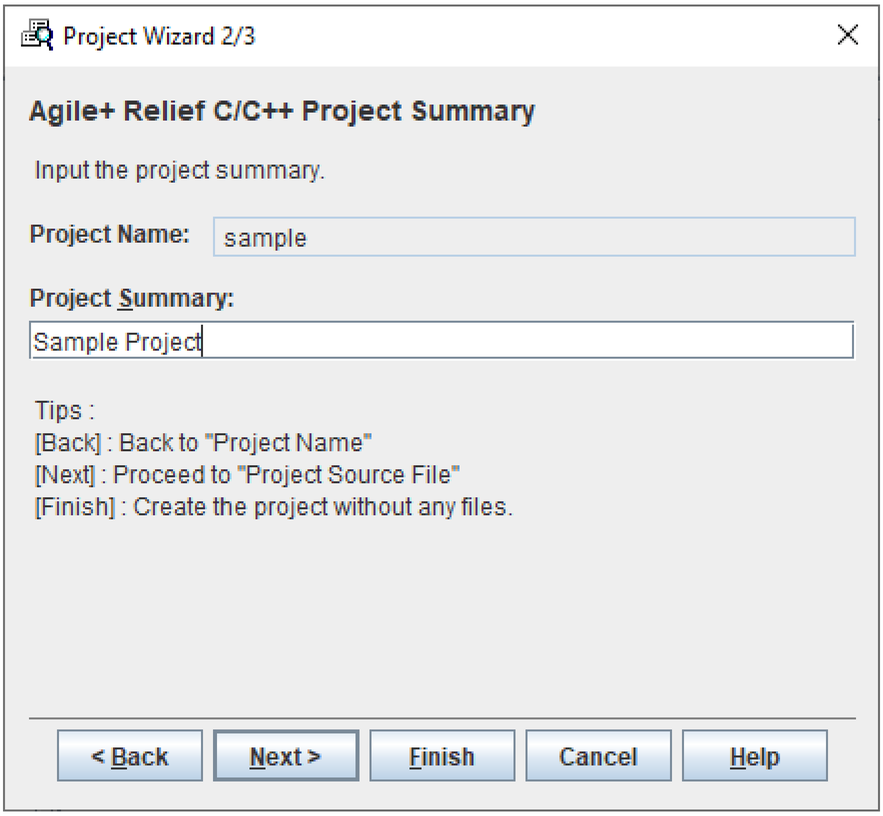
Perform project summary settings in this dialog box.

For the contents set here will be displayed in main window, an easy to understand description will facilitate the convenience of later project reference.

> The punctuations, including Japanese (except half-width kana) and space are allowed in project summary. (Also can be omitted).

[Sample Project] is set for this Example.

After the above settings are done,the dialog box is displayed as follows:



Project Wizard 2/3

Agile+ Relief C/C++ Project Summary

Input the project summary.

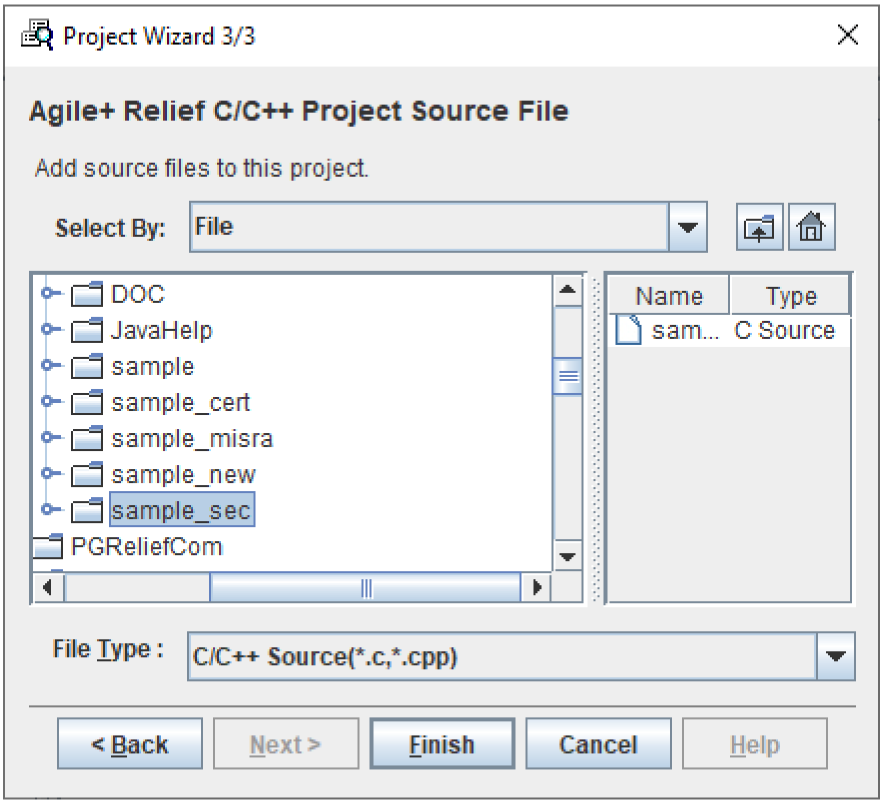
Project Name: sample

Project Summary: Sample Project

Tips :
[Back] : Back to "Project Name"
[Next] : Proceed to "Project Source File"
[Finish] : Create the project without any files.

Choose [Next].

[Project Wizard 3/3] dialog box is displayed.



Project Wizard 3/3

Agile+ Relief C/C++ Project Source File

Add source files to this project.

Select By: File

File Type : C/C++ Source(*.c,*.cpp)

In this dialog box, specify the source file to be registered in project.

[File] is predetermined for file selection.

[C/C++ Source] is predetermined for file type.

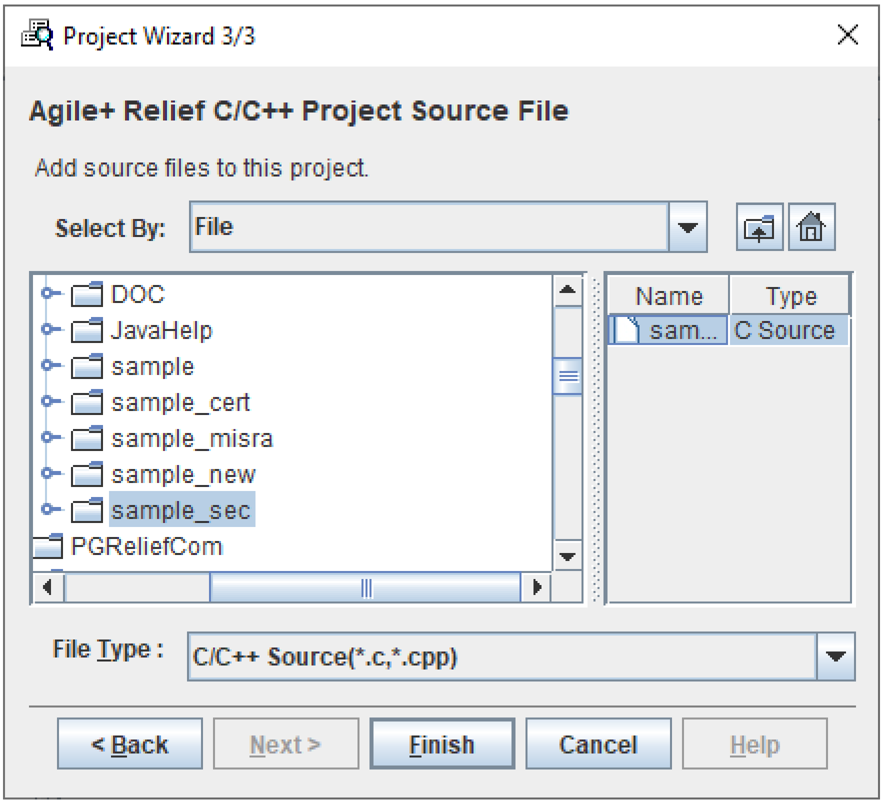
The directory will be displayed on the left to the center of the dialog box.

Select [sample_sec] from the installation directory for Agile$^+$ Relief C/C++.

For usual Agile$^+$ Relief C/C++ installation, the directories of [sample], [sample_new], [sample_misra], [sample_sec] and [sample_cert], in which Example is installed, will be created in the sub directory under the Agile$^+$ Relief C/C++ installation directory.

The source files to be used in Example are displayed in the right view to the center of the dialog box.

Drag the source files displayed in the view, and let the Example file in the state of select all.



Project Wizard 3/3

Agile+ Relief C/C++ Project Source File

Add source files to this project.

Select By: File

Name	Type
sam...	C Source

File Type : C/C++ Source(*.c,*.cpp)

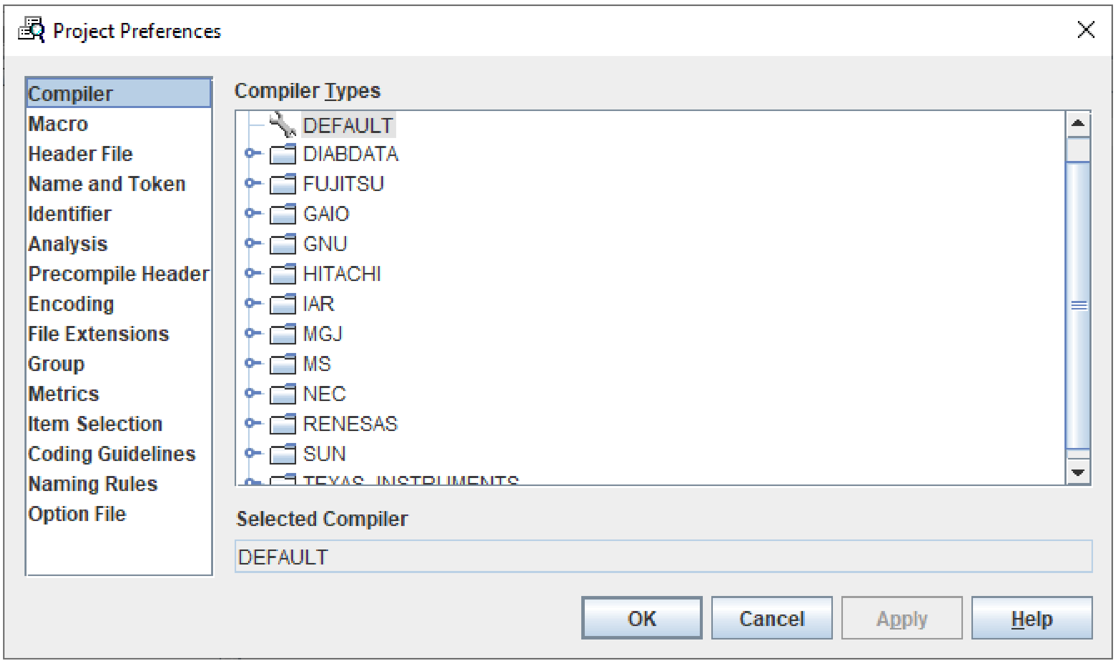
[Project Wizard 3/3] settings are described above.

Choose [Finish].

The project creation has been finished.

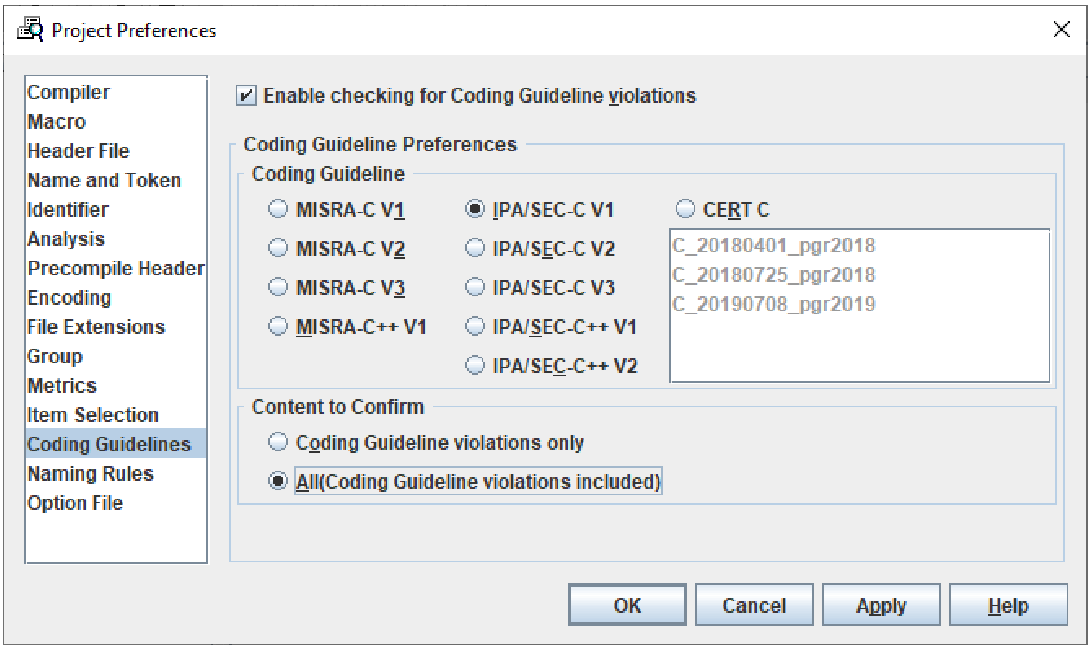
The settings (option settings) required for analysis are as follows:

[Project Preferences] dialog box is displayed.



Selected Compiler: DEFAULT

Select [Coding Guidelines] option, and do check of [All (Coding Guideline violations included)].



☑ Enable checking for Coding Guideline violations

○ MISRA-C V1 ● IPA/SEC-C V1 ○ CERT C

○ MISRA-C V2 ○ IPA/SEC-C V2

○ MISRA-C V3 ○ IPA/SEC-C V3

○ MISRA-C++ V1 ○ IPA/SEC-C++ V1

○ IPA/SEC-C++ V2

○ Coding Guideline violations only

● All(Coding Guideline violations included)

Select the rules for check in [Coding Guidelines].

[MISRA-C V1] to check according to MISRA-C:1998

[MISRA-C V2] to check according to MISRA-C:2004

[MISRA-C V3] to check according to MISRA-C:2012 and MISRA-C:2012 Amendment 1.

[MISRA-C++ V1] to check according to MISRA-C++:2008

[IPA/SEC-C V1] to check according to IPA/SEC-C V1

[IPA/SEC-C V2] to check according to IPA/SEC-C V2

[IPA/SEC-C V3] to check according to IPA/SEC-C V3

[IPA/SEC-C++ V1] to check according to IPA/SEC-C++ V1

[IPA/SEC-C++ V2] to check according to IPA/SEC-C++ V2

[CERT C] to check according to CERT C.

- [MISRA-C V1], [MISRA-C V2] and [MISRA-C++ V1] cannot be selected when no license for MISRA option reserved.
- [MISRA-C V3] checks Coding Guideline violations by the rule added by MISRA-C:2012 Amendment 1 in addition to MISRA-C:2012. Please refer to "3.3 MISRA-C V3" in "MISRA Option Manual", when you want to check Coding Guideline violations by the rule of MISRA-C:2012.
- If you don't have CERT C option license, you can't select [CERT C].

Select [IPA/SEC-C V1] here.

Select the contents for rule violation check in [Content to Confirm].

[Coding Guideline violations only] means to check the specified rule violations only.

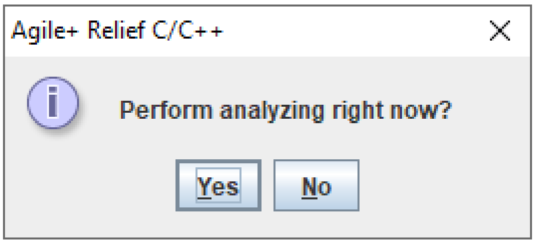
[All (Coding Guideline violations included)] means to check not only the specified rule violations, but also the problems according to Premier’s view.

In order to improve the program quality, it is recommended to check both coding manner and the problems according to Premier’s view.

Select [All (Coding Guideline violations included)] here.

Choose [OK].

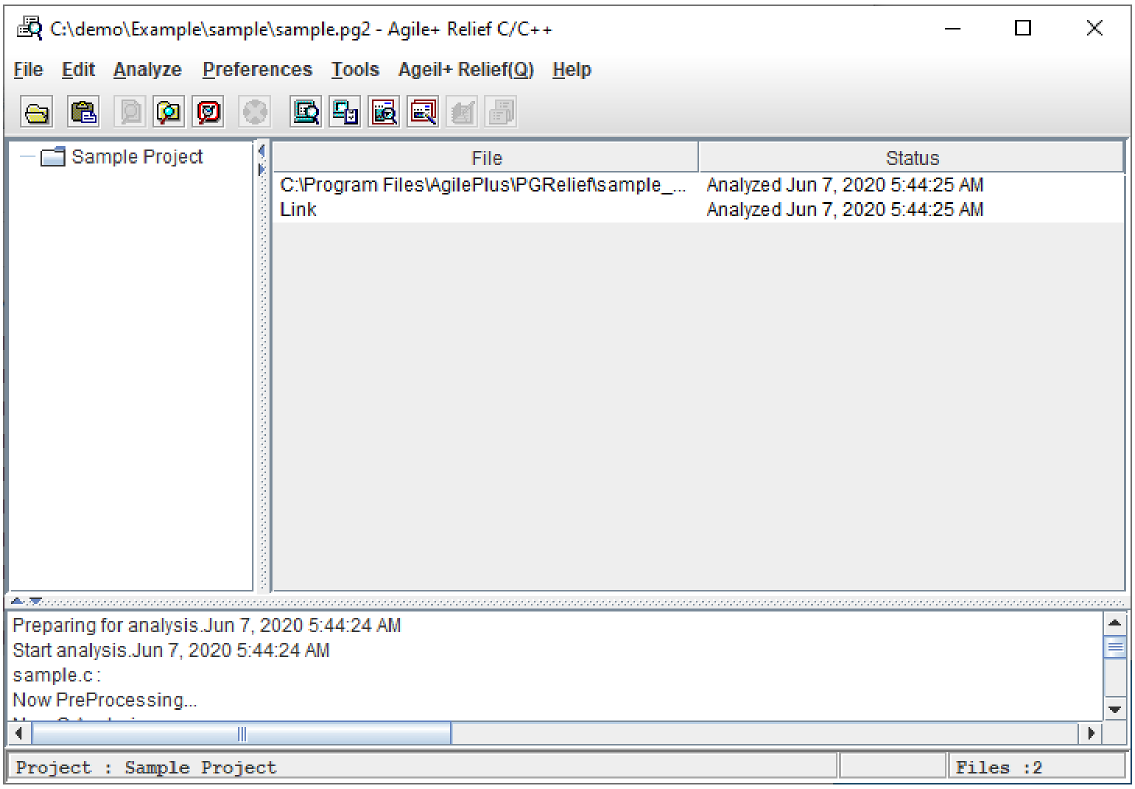
When the [Perform analyzing right now?] message is displayed, choose [Yes].



Perform analyzing right now?

If [No] selected, please select [Analyze All] from [Analyze] menu.

During analysis, the result and state of analysis are displayed in log display window.



C:\demo\Example\sample\sample.pg2 - Agile+ Relief C/C++
File Edit Analyze Preferences Tools Ageil+ Relief(Q) Help
Sample Project

File	Status
C:\Program Files\AgilePlus\PGRelief\sample_...	Analyzed Jun 7, 2020 5:44:25 AM
Link	Analyzed Jun 7, 2020 5:44:25 AM

```
Preparing for analysis.Jun 7, 2020 5:44:24 AM
Start analysis.Jun 7, 2020 5:44:24 AM
sample.c :
Now PreProcessing...
```

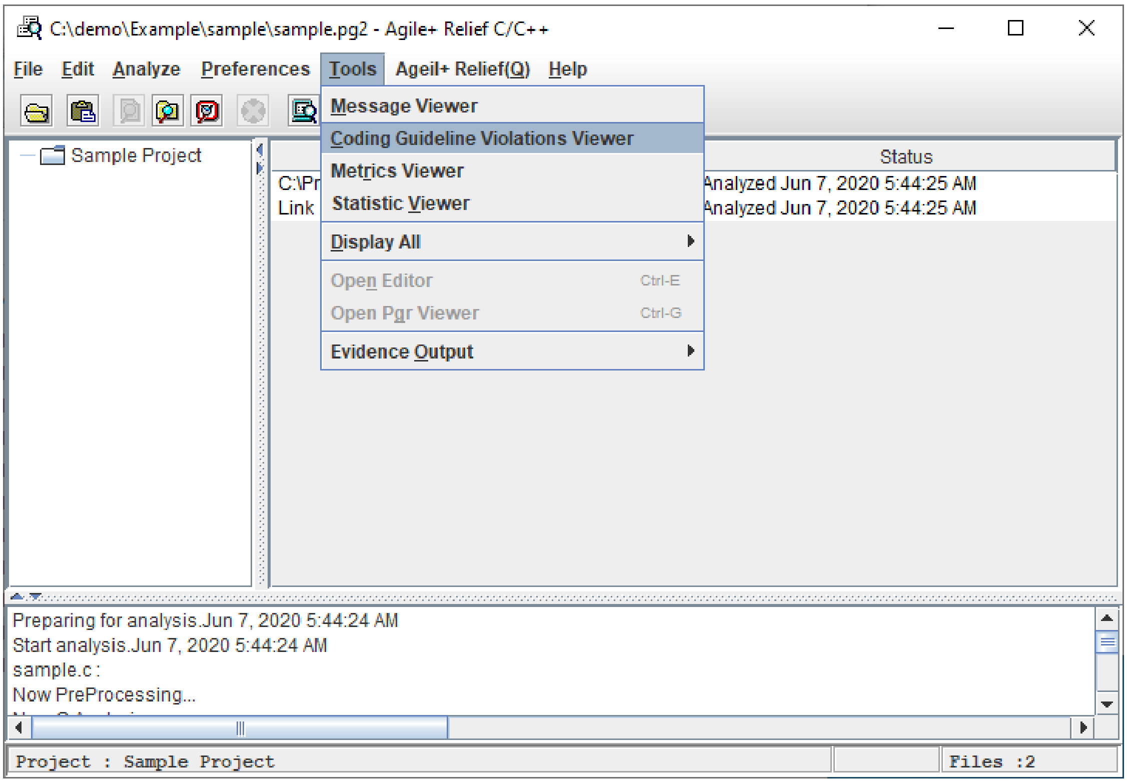
Project : Sample Project
Files :2

There are no errors in the Example analysis. When the analysis is not performable because of error in actual operation, please recheck the settings for compiler type, include and macro, etc.

1.2 Check for IPA/SEC-C/C++ Coding Manner Violation

Be sure check for IPA/SEC-C/C++ coding manner violation is selected.

Select Example.c in [Main Window], and select [Coding Guideline Violations Viewer] from [Tools] menu.

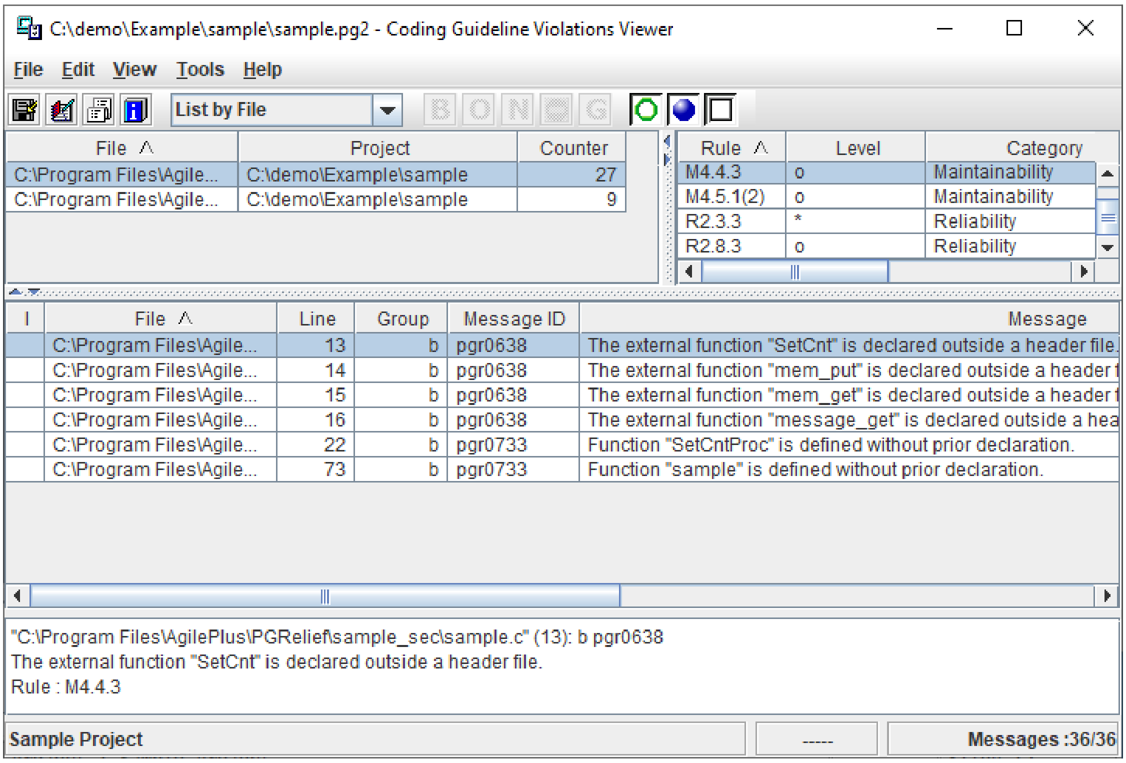


```
Analyzed Jun 7, 2020 5:44:25 AM
Analyzed Jun 7, 2020 5:44:25 AM
```

```
Preparing for analysis.Jun 7, 2020 5:44:24 AM
Start analysis.Jun 7, 2020 5:44:24 AM
sample.c :
Now PreProcessing...
```

```
Project : Sample Project                Files :2
```

[Coding Guideline Violations Viewer] window is displayed.


C:\demo\Example\sample\sample.pg2 - Coding Guideline Violations Viewer
File Edit View Tools Help
List by File
B O N G

File ^	Project	Counter
C:\Program Files\Agile...	C:\demo\Example\sample	27
C:\Program Files\Agile...	C:\demo\Example\sample	9

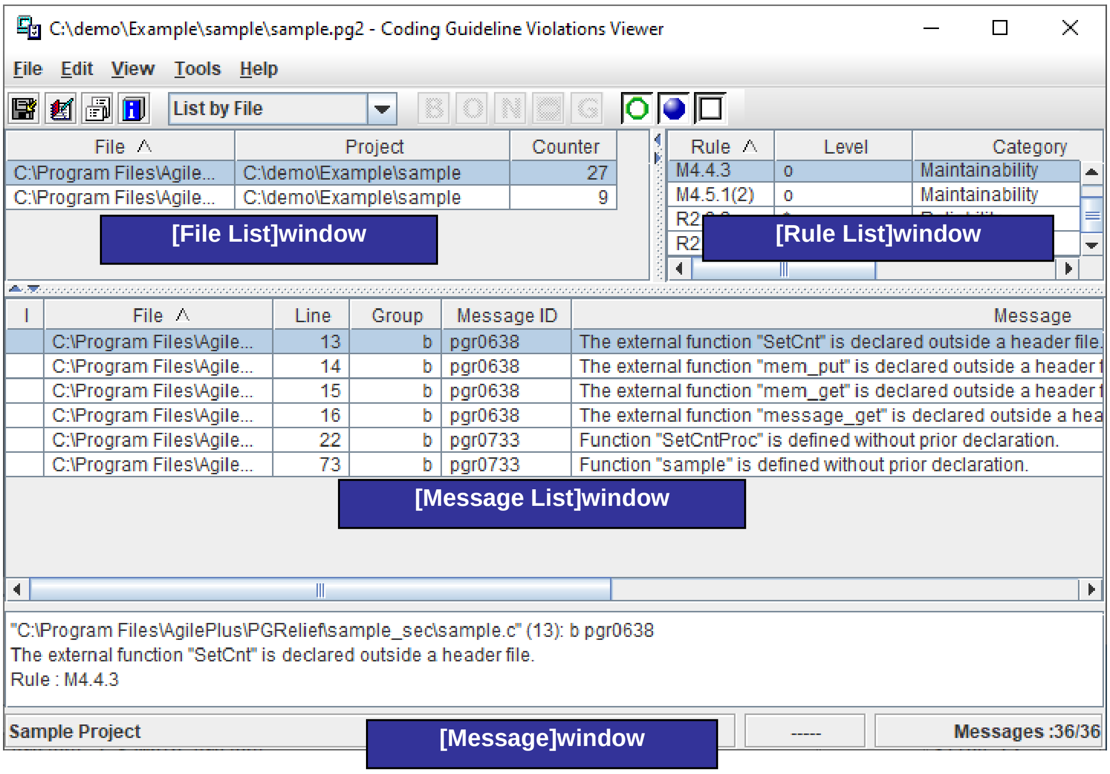
Rule ^	Level	Category
M4.4.3	o	Maintainability
M4.5.1(2)	o	Maintainability
R2.3.3	*	Reliability
R2.8.3	o	Reliability

I	File ^	Line	Group	Message ID	Message
	C:\Program Files\Agile...	13	b	pgr0638	The external function "SetCnt" is declared outside a header file.
	C:\Program Files\Agile...	14	b	pgr0638	The external function "mem_put" is declared outside a header f
	C:\Program Files\Agile...	15	b	pgr0638	The external function "mem_get" is declared outside a header f
	C:\Program Files\Agile...	16	b	pgr0638	The external function "message_get" is declared outside a hea
	C:\Program Files\Agile...	22	b	pgr0733	Function "SetCntProc" is defined without prior declaration.
	C:\Program Files\Agile...	73	b	pgr0733	Function "sample" is defined without prior declaration.

```
"C:\Program Files\AgilePlus\PGRelief\sample_sec\sample.c" (13): b pgr0638
The external function "SetCnt" is declared outside a header file.
Rule : M4.4.3
```

Sample Project ----- Messages :36/36

Briefly introduce how to use [Coding Guideline Violations Viewer] window.


C:\demo\Example\sample\sample.pg2 - Coding Guideline Violations Viewer
File Edit View Tools Help
List by File
B O N G
[File List]window
[Rule List]window
[Message List]window
[Message]window

File ^	Project	Counter
C:\Program Files\Agile...	C:\demo\Example\sample	27
C:\Program Files\Agile...	C:\demo\Example\sample	9

Rule ^	Level	Category
M4.4.3	o	Maintainability
M4.5.1(2)	o	Maintainability

I	File ^	Line	Group	Message ID	Message
	C:\Program Files\Agile...	13	b	pgr0638	The external function "SetCnt" is declared outside a header file.
	C:\Program Files\Agile...	14	b	pgr0638	The external function "mem_put" is declared outside a header f
	C:\Program Files\Agile...	15	b	pgr0638	The external function "mem_get" is declared outside a header f
	C:\Program Files\Agile...	16	b	pgr0638	The external function "message_get" is declared outside a hea
	C:\Program Files\Agile...	22	b	pgr0733	Function "SetCntProc" is defined without prior declaration.
	C:\Program Files\Agile...	73	b	pgr0733	Function "sample" is defined without prior declaration.

```
"C:\Program Files\AgilePlus\PGRelief\sample_sec\sample.c" (13): b pgr0638
The external function "SetCnt" is declared outside a header file.
Rule : M4.4.3
```

Sample Project ----- Messages :36/36

The files violating IPA/SEC-C/C++ coding manner are displayed in [File List] window.

The number of rules violating IPA/SEC-C/C++ coding manner are displayed in [Rule List].

The contents of message indication and the locations of indicated source program are displayed in [Message List].

If [File List] on toolbar is set to be [Rule List], the [File List] window and [Rule List] window will be substituted with each other.

When rule ID is selected in [Rule List] window, the corresponding message indication will be displayed in [Message List] window.

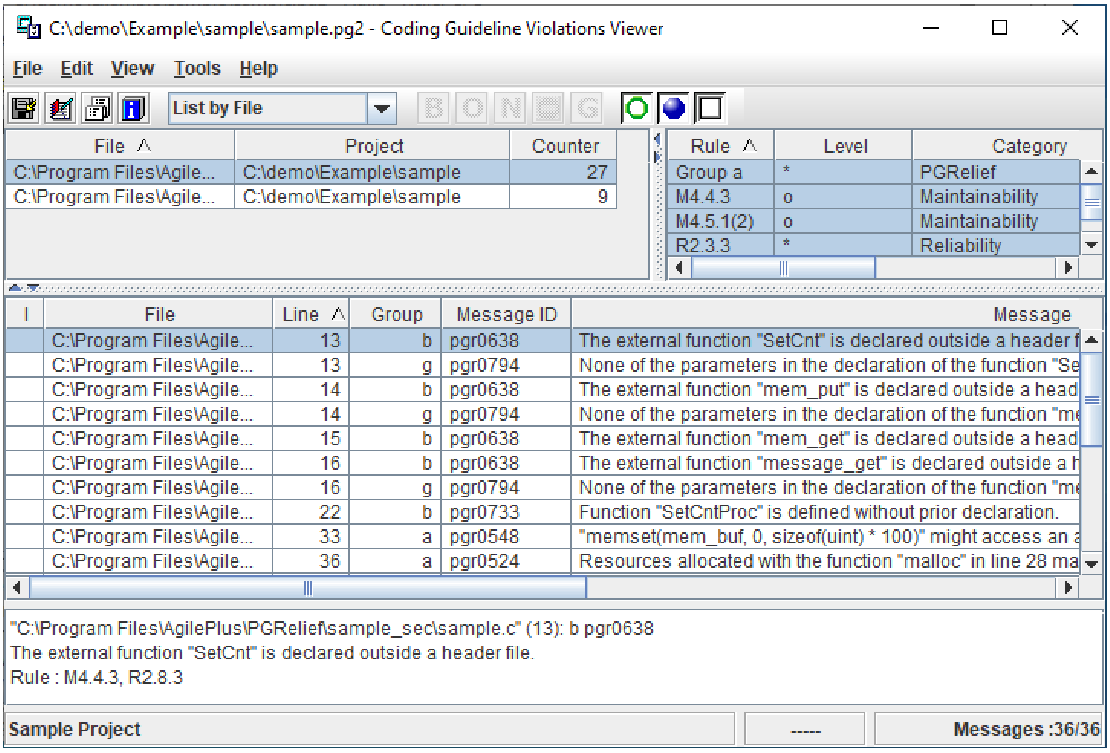
Confirm the indicated locations for violation in turn and then correct source file.

Some of Rule ID for IPA/SEC-C/C++ coding manner may point to the same message indication.

For example: The message ID indicated in line 22 of Example: The problem indicated in pgr0733 is the same with that of R2.8.3 and M4.4.3.

It is recommended to select all rules in [Rule List] window, and perform corrections according to the line number of source, which is of less repeatability and higher efficiency than to select rule one by one in [Rule List] window and correct them in turn.

In Example source, please click [Line] title for sequencing incrementally and then correct them in turn.


C:\demo\Example\sample\sample.pg2 - Coding Guideline Violations Viewer
File
Edit
View
Tools
Help
List by File
B
O
N
G

File ^	Project	Counter
C:\Program Files\Agile...	C:\demo\Example\sample	27
C:\Program Files\Agile...	C:\demo\Example\sample	9

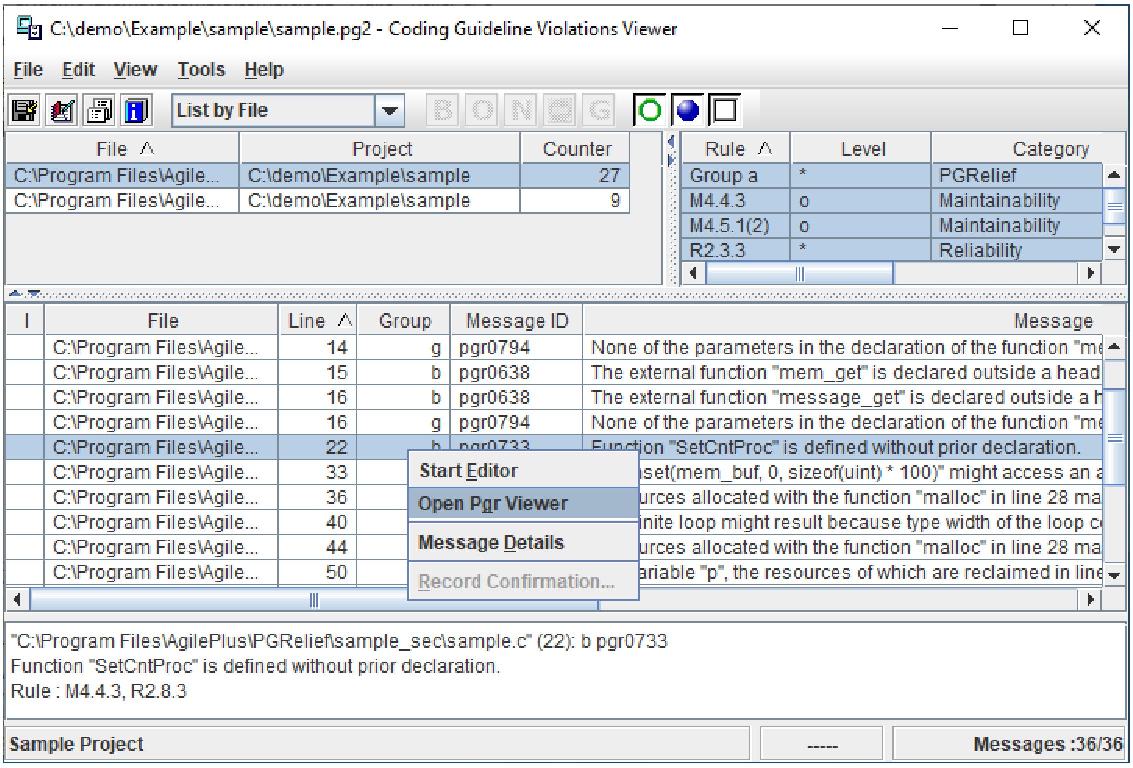
Rule ^	Level	Category
Group a	*	PGRelief
M4.4.3	o	Maintainability
M4.5.1(2)	o	Maintainability
R2.3.3	*	Reliability

I	File	Line ^	Group	Message ID	Message
	C:\Program Files\Agile...	13	b	pgr0638	The external function "SetCnt" is declared outside a header f
	C:\Program Files\Agile...	13	g	pgr0794	None of the parameters in the declaration of the function "Se
	C:\Program Files\Agile...	14	b	pgr0638	The external function "mem_put" is declared outside a head
	C:\Program Files\Agile...	14	g	pgr0794	None of the parameters in the declaration of the function "me
	C:\Program Files\Agile...	15	b	pgr0638	The external function "mem_get" is declared outside a head
	C:\Program Files\Agile...	16	b	pgr0638	The external function "message_get" is declared outside a h
	C:\Program Files\Agile...	16	g	pgr0794	None of the parameters in the declaration of the function "me
	C:\Program Files\Agile...	22	b	pgr0733	Function "SetCntProc" is defined without prior declaration.
	C:\Program Files\Agile...	33	a	pgr0548	"memset(mem_buf, 0, sizeof(uint) * 100)" might access an a
	C:\Program Files\Agile...	36	a	pgr0524	Resources allocated with the function "malloc" in line 28 ma

"C:\Program Files\AgilePlus\PGRelief\sample_sec\sample.c" (13): b pgr0638
The external function "SetCnt" is declared outside a header file.
Rule : M4.4.3, R2.8.3

Sample Project | ----- | Messages :36/36

Select pgr0733 in line 22, and right click to select [Open Pgr Viewer].


C:\demo\Example\sample\sample.pg2 - Coding Guideline Violations Viewer
File Edit View Tools Help
List by File
B O N G
Start Editor
Open Pgr Viewer
Message Details
Record Confirmation...

File ^	Project	Counter
C:\Program Files\Agile...	C:\demo\Example\sample	27
C:\Program Files\Agile...	C:\demo\Example\sample	9

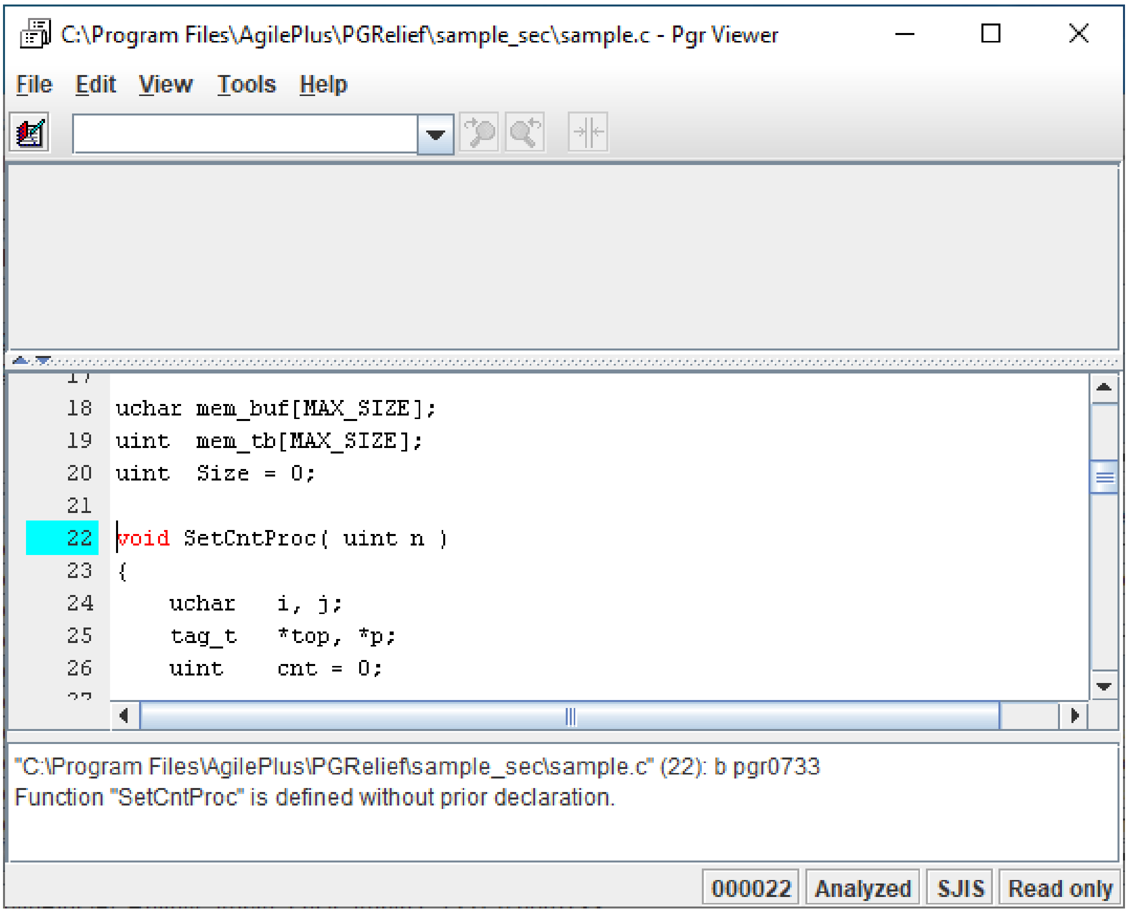
Rule ^	Level	Category
Group a	*	PGRelief
M4.4.3	o	Maintainability
M4.5.1(2)	o	Maintainability
R2.3.3	*	Reliability

I	File	Line ^	Group	Message ID	Message
	C:\Program Files\Agile...	14	g	pgr0794	None of the parameters in the declaration of the function "me
	C:\Program Files\Agile...	15	b	pgr0638	The external function "mem_get" is declared outside a head
	C:\Program Files\Agile...	16	b	pgr0638	The external function "message_get" is declared outside a h
	C:\Program Files\Agile...	16	g	pgr0794	None of the parameters in the declaration of the function "me
	C:\Program Files\Agile...	22	b	pgr0733	Function "SetCntProc" is defined without prior declaration.
	C:\Program Files\Agile...	33			set(mem_buf, 0, sizeof(uint) * 100)" might access an a
	C:\Program Files\Agile...	36			urces allocated with the function "malloc" in line 28 ma
	C:\Program Files\Agile...	40			inite loop might result because type width of the loop c
	C:\Program Files\Agile...	44			urces allocated with the function "malloc" in line 28 ma
	C:\Program Files\Agile...	50			ariable "p", the resources of which are reclaimed in line

"C:\Program Files\AgilePlus\PGRelief\sample_sec\sample.c" (22): b pgr0733
Function "SetCntProc" is defined without prior declaration.
Rule : M4.4.3, R2.8.3

Sample Project ----- Messages :36/36

Viewer will be started.


C:\Program Files\AgilePlus\PGRelief\sample_sec\sample.c - Pgr Viewer
File Edit View Tools Help

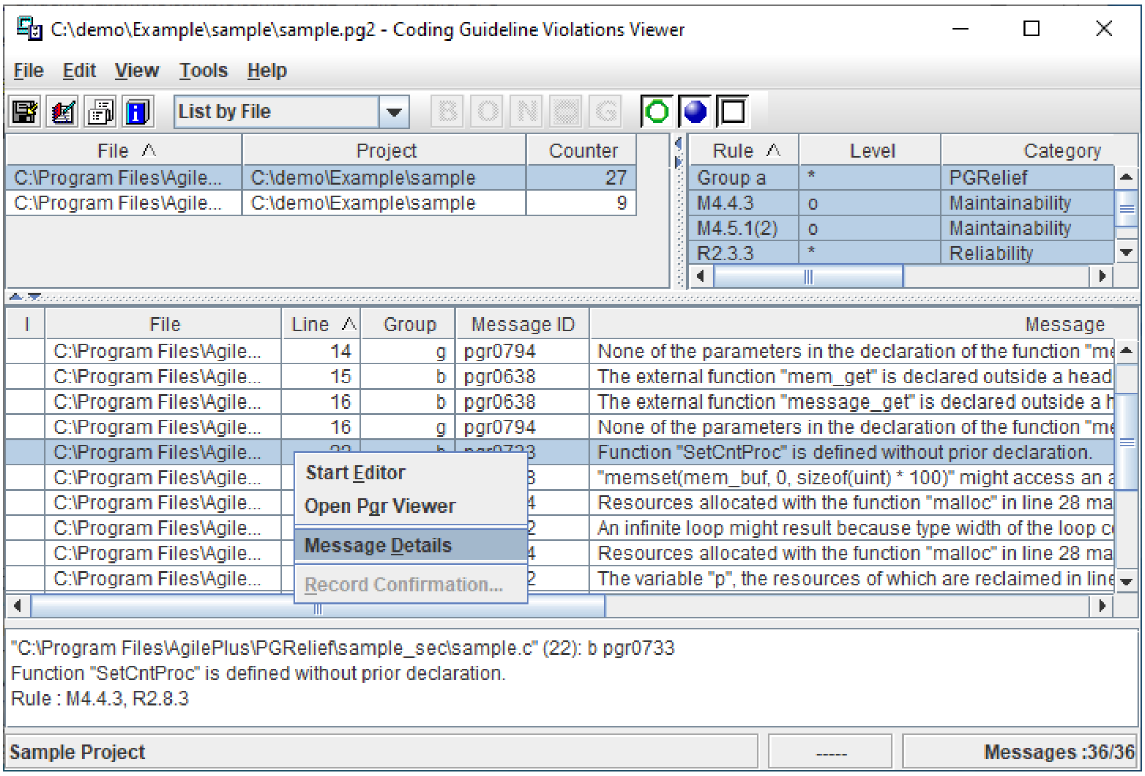
```
uchar mem_buf[MAX_SIZE];
uint  mem_tb[MAX_SIZE];
uint  Size = 0;

void SetCntProc( uint n )
{
    uchar   i, j;
    tag_t   *top, *p;
    uint    cnt = 0;
```

"C:\Program Files\AgilePlus\PGRelief\sample_sec\sample.c" (22): b pgr0733
Function "SetCntProc" is defined without prior declaration.

000022 Analyzed SJIS Read only

Confirm the revision point, and do certain modification with the existing editor.

If the meaning of the message indication is hard to understand, please select the message and right click to select [Message Details].



File	Project	Counter
C:\Program Files\Agile...	C:\demo\Example\sample	27
C:\Program Files\Agile...	C:\demo\Example\sample	9

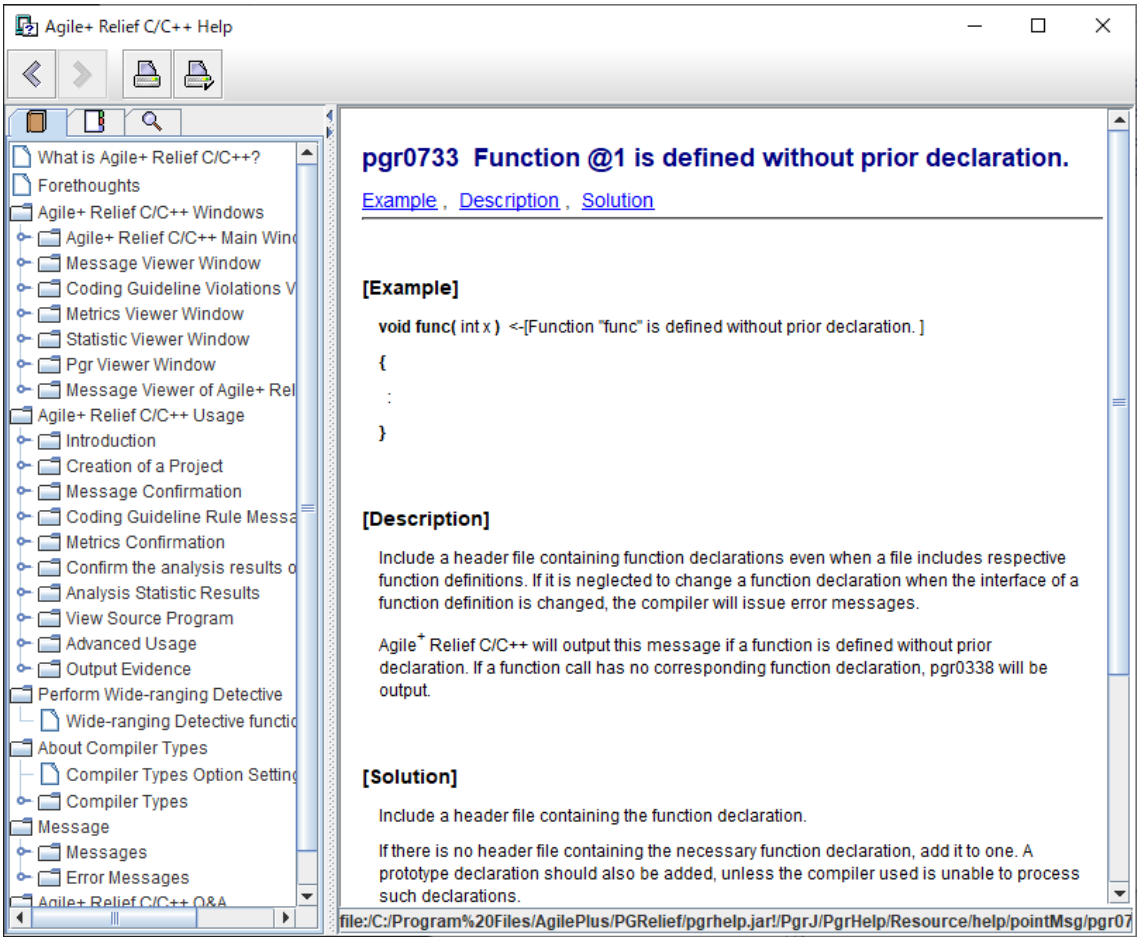
Rule	Level	Category
Group a	*	PGRelief
M4.4.3	o	Maintainability
M4.5.1(2)	o	Maintainability
R2.3.3	*	Reliability

I	File	Line	Group	Message ID	Message
	C:\Program Files\Agile...	14	g	pgr0794	None of the parameters in the declaration of the function "me
	C:\Program Files\Agile...	15	b	pgr0638	The external function "mem_get" is declared outside a head
	C:\Program Files\Agile...	16	b	pgr0638	The external function "message_get" is declared outside a h
	C:\Program Files\Agile...	16	g	pgr0794	None of the parameters in the declaration of the function "me
	C:\Program Files\Agile...	22	b	pgr0733	Function "SetCntProc" is defined without prior declaration.
	C:\Program Files\Agile...			3	"memset(mem_buf, 0, sizeof(uint) * 100)" might access an a
	C:\Program Files\Agile...			4	Resources allocated with the function "malloc" in line 28 ma
	C:\Program Files\Agile...			2	An infinite loop might result because type width of the loop c
	C:\Program Files\Agile...			4	Resources allocated with the function "malloc" in line 28 ma
	C:\Program Files\Agile...			2	The variable "p", the resources of which are reclaimed in line

"C:\Program Files\AgilePlus\PGRelief\sample_sec\sample.c" (22): b pgr0733
Function "SetCntProc" is defined without prior declaration.
Rule : M4.4.3, R2.8.3

Sample Project | ----- | Messages :36/36

Help window will be started. Please do correction with reference to help.


Agile+ Relief C/C++ Help
What is Agile+ Relief C/C++?
Forethoughts
Agile+ Relief C/C++ Windows
Agile+ Relief C/C++ Main Winc
Message Viewer Window
Coding Guideline Violations V
Metrics Viewer Window
Statistic Viewer Window
Pgr Viewer Window
Message Viewer of Agile+ Rel
Agile+ Relief C/C++ Usage
Introduction
Creation of a Project
Message Confirmation
Coding Guideline Rule Messa
Metrics Confirmation
Confirm the analysis results o
Analysis Statistic Results
View Source Program
Advanced Usage
Output Evidence
Perform Wide-ranging Detective
Wide-ranging Detective functic
About Compiler Types
Compiler Types Option Setting
Compiler Types
Message
Messages
Error Messages
Agile+ Relief C/C++ Q&A

pgr0733 Function @1 is defined without prior declaration.

Example , Description , Solution

[Example]

```
void func( int x )  <-[Function "func" is defined without prior declaration. ]
{
  :
}
```

[Description]

Include a header file containing function declarations even when a file includes respective function definitions. If it is neglected to change a function declaration when the interface of a function definition is changed, the compiler will issue error messages.

Agile$^+$ Relief C/C++ will output this message if a function is defined without prior declaration. If a function call has no corresponding function declaration, pgr0338 will be output.

[Solution]

Include a header file containing the function declaration.

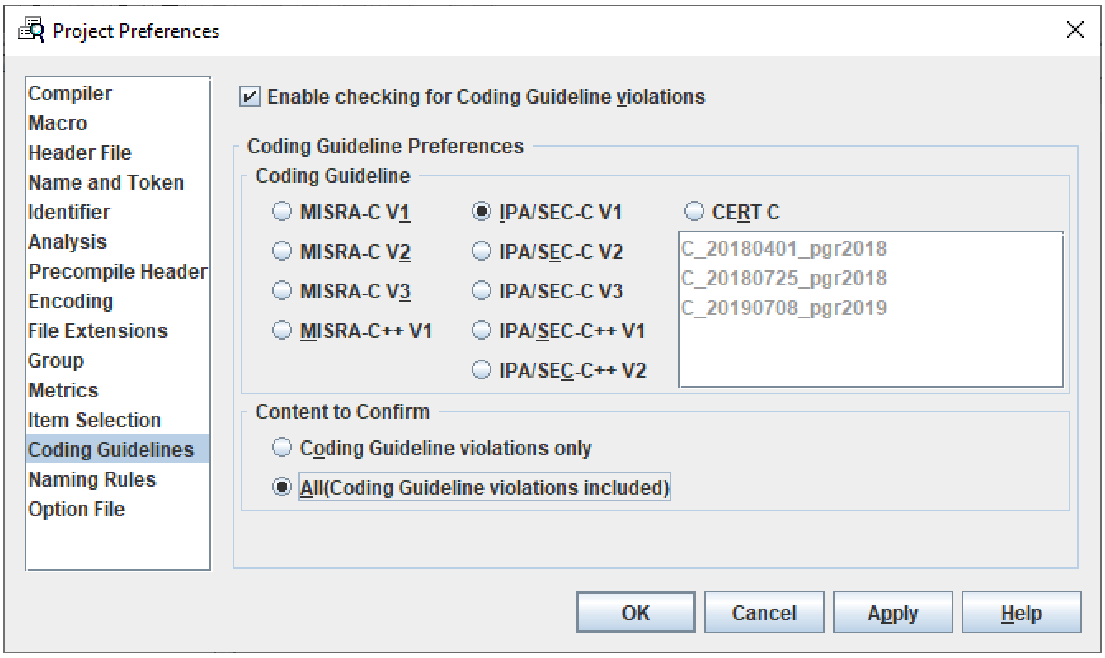
If there is no header file containing the necessary function declaration, add it to one. A prototype declaration should also be added, unless the compiler used is unable to process such declarations.

file:/C:/Program%20Files/AgilePlus/PGRelief/pgrhelp.jar!/PgrJ/PgrHelp/Resource/help/pointMsg/pgr07

1.3 Experienced with Agile^{+} Relief C/C++

The procedures for a new project of IPA/SEC-C/C++ coding manner check are the same with the normal project .

The only difference is the addition of procedures for [Coding Guidelines] of [Project Preferences].



☑ Enable checking for Coding Guideline violations

Coding Guideline Preferences

Coding Guideline

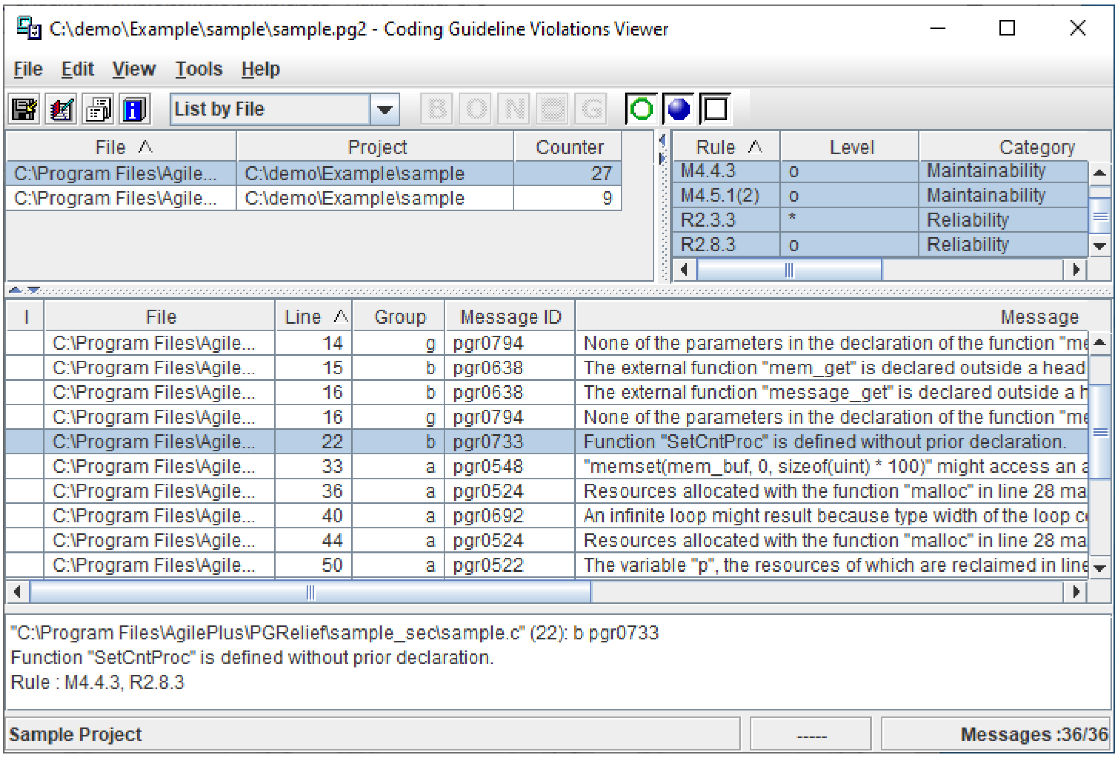
○ MISRA-C V1 ● IPA/SEC-C V1 ○ CERT C
○ MISRA-C V2 ○ IPA/SEC-C V2
○ MISRA-C V3 ○ IPA/SEC-C V3
○ MISRA-C++ V1 ○ IPA/SEC-C++ V1
○ IPA/SEC-C++ V2

C_20180401_pgr2018
C_20180725_pgr2018
C_20190708_pgr2019

Content to Confirm

○ Coding Guideline violations only
● All(Coding Guideline violations included)

In addition, the display of [Message] in [Message List] of [Coding Guideline Violations Viewer] window is a little different from that of [Rule List]. The rest, also the procedures are similar to each other.



C:\demo\Example\sample\sample.pg2 - Coding Guideline Violations Viewer

File	Project	Counter
C:\Program Files\Agile...	C:\demo\Example\sample	27
C:\Program Files\Agile...	C:\demo\Example\sample	9

Rule	Level	Category
M4.4.3	o	Maintainability
M4.5.1(2)	o	Maintainability
R2.3.3	*	Reliability
R2.8.3	o	Reliability

I	File	Line	Group	Message ID	Message
	C:\Program Files\Agile...	14	g	pgr0794	None of the parameters in the declaration of the function "me
	C:\Program Files\Agile...	15	b	pgr0638	The external function "mem_get" is declared outside a head
	C:\Program Files\Agile...	16	b	pgr0638	The external function "message_get" is declared outside a h
	C:\Program Files\Agile...	16	g	pgr0794	None of the parameters in the declaration of the function "me
	C:\Program Files\Agile...	22	b	pgr0733	Function "SetCntProc" is defined without prior declaration.
	C:\Program Files\Agile...	33	a	pgr0548	"memset(mem_buf, 0, sizeof(uint) * 100)" might access an a
	C:\Program Files\Agile...	36	a	pgr0524	Resources allocated with the function "malloc" in line 28 ma
	C:\Program Files\Agile...	40	a	pgr0692	An infinite loop might result because type width of the loop c
	C:\Program Files\Agile...	44	a	pgr0524	Resources allocated with the function "malloc" in line 28 ma
	C:\Program Files\Agile...	50	a	pgr0522	The variable "p", the resources of which are reclaimed in line

"C:\Program Files\AgilePlus\PGRelief\sample_sec\sample.c" (22): b pgr0733
Function "SetCntProc" is defined without prior declaration.
Rule : M4.4.3, R2.8.3

Sample Project ----- **Messages :36/36**

1.4 Setting up Rule Checking

When only the necessary rules other than all of them are required for checking, you need to create a definition file for rule checking.

Please create the select message file in the form as follows:

Please write comma (;) at the line beginning of comment

Please write a rule ID or group name in one line.
The barced number can be recorded in certain condition.

Please use half-width English lowercase character for group name.

For IPA/SEC-C V1, write explicitly at the beginning of the file.
;Rule=SEC-C V1

For IPA/SEC-C V2, write explicitly at the beginning of the file.
;Rule=SEC-C V2

For IPA/SEC-C V3, write explicitly at the beginning of the file.
;Rule=SEC-C V3

For IPA/SEC-C++ V1, write explicitly at the beginning of the file.
;Rule=SEC-C++ V1

For IPA/SEC-C++ V2, write explicitly at the beginning of the file.
;Rule=SEC-C++ V2

For example:) Take R1.1.1, M1.2.1(1), M1.7.1 and Agile$^+$ Relief 's a group as the objects to check.

```
;Rule=SEC-C V1
R1.1.1
M1.2.1(1)
M1.7.1
group-a
```

The definition file for rule checking uses the [Item Selection] option of the[Project Preferences]dialogue box.

If you need to check the necessary rules only, select [Customizing point], and add the check indication definition file that describes the check rules.

If you need to check all rules, select [All points].

If [Default value] is selected, the following check indication definition file will be applied.

[Directory]

For Windows:

"(Agile$^+$ Relief C/C++ Setup Directory)\Analyze\EPOM\MessageInfo"

For Linux:

"(Agile$^+$ Relief C/C++ Setup Directory)/Analyze/EPOM/MessageInfo"

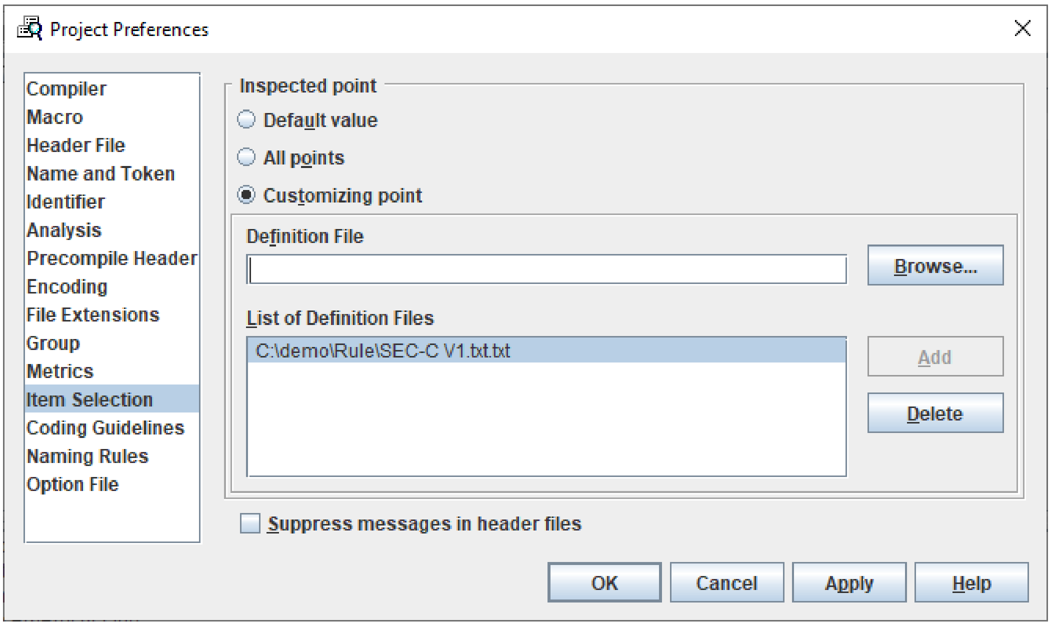
[File name]

IPA/SEC-C V1	: default-secv1.rul
IPA/SEC-C V2	: default-secv2.rul

IPA/SEC-C V3 : default-secv3.rul

IPA/SEC-C++ V1 : default-secpv1.rul

IPA/SEC-C++ V2 : default-secpv2.rul



Inspected point

○ Default value

○ All points

● Customizing point

Definition File

List of Definition Files

C:\demo\Rule\SEC-C V1.txt.txt

□ Suppress messages in header files

2. IPA/SEC-C/C++ Coding Manner Check with Command Line

This section specifies how to use pgrsec command in command line.

2.1 Functions of Command

The pgrsec command is used for the check of message indications output by pgr5 command, and output message for IPA/SEC-C/C++ coding manner violation. In addition, it is necessary that the message information output by pgr5 command should be output as files in csv format.

2.2 Format of Command

```
% pgrsec   [-V]   [--pgr]   -T IPA/SEC-C/C++ version     [-Z definition file for rule
checking]
               [--qm]   input file
```

Note: [] means omittable.

2.3 Command Options

option	Description
-V	Display the version, level and release of pgrsec command. When the option is specified, others will be ignored.
--pgr	The option shall be specified when a concurrent output of pgr5 command contents is required.
-T IPA/SEC-C/C++ version	Specify the version of IPA/SEC-C/C++ to ouput.Please make sure it is specified. Please don't insert space between -T option and IPA/SEC-C/C++ version. 1: Check according to IPA/SEC-C V1 rules. 2: Check according to IPA/SEC-C V2 rules. 3: Check according to IPA/SEC-C V3 rules. P1: Check according to IPA/SEC-C++ V1 rules. P2: Check according to IPA/SEC-C++ V2 rules.
-Z definition file for rule checking	Specify the select message file when only the output of desired rules is required. About the methods for the description of

	select message file, please refer to [1.4 Setting up Rule Checking]. All rules will be output when this option is not specified.
--qm	This is the output format for the analysis result consolidation mode of the Agile+ Relief(*). Specify this option to use the analysis result consolidation mode of the Agile+ Relief. For the analysis result consolidation mode of the Agile+ Relief, please refer to "2.3.2 Procedure of using under analysis result consolidation mode" in "Agile+ Relief Manual". (*)Agile+ Relief is a function that enables the visualization of problems related to quality by checking quality data that analyzed the source programs on a daily basis. Please note that using this Agile+ Relief requires a license.
Input File	CSV format is specified to save the filename of contents indicated by pgr5 command. Files will be saved in the following codes: Chinese OS GB2312 code Other Language OS Windows : SJIS code Red Hat Enterprise Linux 5 or 6 : UTF-8 code Others : EUC code

2.4 Return Value

0 will be returned when pgrsec command normally ended, a value other than 0 will be returned for error occurence.

2.5 Output Files

File will be output in CSV.

Example of File Output

“C:\Example.c”,22,”R2.8.3”,”○”, “Reliability”, “b”, “pgr0773”, “No function declaration found before the function definition ‘SetCntProc’”.

Format Description

source file name	“C:\Example.c”
line number	22
rule ID	“R2.8.3”
rule classification	“○”
rule group	“Reliability”
Message group	“b”
Message ID	“pgr0773”
Message Indication	“No function declaration found before the function definition ‘SetCntProc’”.

2.6 Error Messages

PGRSEC_0001 Input file was not specified.

Input file is not specified. Please check the command format, and then reexecute.

PGRSEC_0002 Option cannot be specified after input file designation.

Command format is incorrect. Please check the command format and then reexecute.

PGRSEC_0003 Multiple input files cannot be specified.

Multiple input files are specified. Please check the command format and then reexecute.

PGRSEC_0004 Unknown option was specified.

Option Error. Please be sure the option is right.

PGRSEC_0005 -T option was not specified.

-T1 option, -T2 option, -T3 option, -TP1 option, or -TP2 option are not specified. Please be sure to specify one of options.

PGRSEC_0006 -T option cannot be specified at the same time.

-T1 option, -T2 option, -T3 option, -TP1 option, and -TP2 option cannot be specified at the same time. Please specify them one-at-a-time.

PGRSEC_0040 Parameter for –Z option was not specified.

No select message file was specified. Please check the command formation and then reexecute.

2.7 Example for Use

2.7.1 Analyze with pgr5 command

Source file is analyzed with pgr5 command at first to obtain the message indication to be checked in pgrsec command.

<Example>

```
pgr5 -F SEC_C_V1.idt --csv Example.c > message.csv
     _______________ _____ _________ _____________

           (1)        (2)    (3)         (4)
```

(1): Specify the identifier file for IPA/SEC-C/C++ analysis.

The identifier files for IPA/SEC-C/C++ analysis are saved in the following directories.

[Identifier File Saving Directory]

In Windows:

"(Agile$^+$ Relief C/C++ Installation Directory)\Analyze\EPOM\SECInfo"

In Linux:

"(Agile$^+$ Relief C/C++ Installation Directory)/Analyze/EPOM/SECInfo"

[Type of identifier files]

IPA/SEC-C V1	: SEC_C_V1.idt
IPA/SEC-C V2	: SEC_C_V2.idt
IPA/SEC-C V3	: SEC_C_V3.idt
IPA/SEC-C++ V1	: SEC_C++_V1.idt
IPA/SEC-C++ V2	: SEC_C++_V2.idt

(2):Output the contents of message indications in CSV format.

(3):Analyze objective source file.

(4):Target file for saving the message output in CSV format.

For more details of pgr5 command, please refer to "Command Manual".

2.7.2 Message Checking in the pgrsec Command

IPA/SEC-C/C++ rule violations will be obtained after the check of message indication contents obtained through pgr5 command.

<Example>

```
pgrsec -T1 -Z"c:\demo\Rule\SEC-C V1.txt" message.csv
       ___ ____________________________ ___________
       (1)            (2)                   (3)
```

(1): Specify the version of IPA/SEC-C/C++ rule.
Please specify the following options based on IPA/SEC-C/C++ Rules Version determined by -F option in the pgr5 command.

IPA/SEC-C V1	: -T1
IPA/SEC-C V2	: -T2
IPA/SEC-C V3	: -T3
IPA/SEC-C++ V1	: -TP1
IPA/SEC-C++ V2	: -TP2

(2): The check indication definition file to be checked is recorded.

(3): The file of message indication contents output by pgr5 command in CSV format is saved.

[Notes]

Please note the codes of input files for pgrsec((3) in the above sample).

In Chinese OS, the input file for the pgrsec command for both Windows and Linux needs to be output in GB2312 code.

In other language OS, the input file for the pgrsec command needs to be output in SJIS code for Windows and UTF-8 code for Red Hat Enterprise Linux 5 or 6, and the rest in EUC code.

When the "--output_code" option is used with the pgr5 command, specify the option to enable output with the above code. For further details regarding the pgr5 command, please refer to the "Command Manual".

3. Points of Message Indication

This section specifies the examples for the points of message indications to the rules of four qualities respectively in Agile$^+$ Relief, such as [Reliability], [Maintainability], [Portability] and [Efficiency].

The example for the points of message indication to [Frequent Coding Miss] is also provided in Agile$^+$ Relief. However, no definition has been made for the rule ID of [Frequent Coding Miss], a prefix of [Miss] will shall be added in Agile$^+$ Relief for unique.

The following explains the correspondence of IPA/SEC-C V1/V2/V3 rule. Please contact us about the correspondence of IPA/SEC-C++ V1/V2 rule.

3.1 Reliability

R1.1.1

[Points of Message Indication in Agile$^+$ Relief]

The automatic variable is referenced without having been set a value.

[Example 1]

```
int   i;
if ( data == 0 ) {
      i = 1;
}
data = i;                              <-[Variable "i" may be referenced before it has been set with a
                                           value.]
```

R1.1.2

[Points of Message Indication in Agile^{+} Relief]

- The declaration of const variable not yet initialized.

[Example 1]

```
const   int   x;        <-[The declaration “x” of const variable was not initialized.]
```

R1.2.1

[Points of Message Indication in Agile+ Relief]

- Attempt to use the string exceeding the length of array to perform the initialization.

[Example 1]

```
char   str[3] = "abc";        <-[Attempt to use the string “abc” exceeding the size of array “str” to
                                perform the initialization.]
```

R1.2.2

[Points of Message Indication in Agile^{+} Relief]

Some enumeration constants are assigned values while others are not.

[Example 1]

enum E1 { E11, **E12 = 3**, E13 } ; <-[In the declaration of the enumeration type "E1", assigned members and unassigned members coexist.]

enum E2 { **E21=1**, E22, **E23 = 5** } ; <-[In the declaration of the enumeration type "E2", assigned members and unassigned members coexist.]

R1.3.1(1)

[Points of Message Indication in Agile$^+$ Relief]

- The array factor is not referenced in the format of array.

[Example 1]

```
int    data[] = { 1,2,3,4,5 };
int    *p = data;
return *(p+1);                          <-[The mathematic operation of “p+1” is not compatible with
                                        the format of array.]
```

R1.3.1.(2)

[Points of Message Indication in Agile$^+$ Relief]

An arithmetic operation performed on a pointer of non-array type.

Use of a pointer of non-array type as an array.

[Example 1]

```
void   func ( int   *array )
{
   int   x;
   int   *p = &x;
   *(array + 1) = 0;          <-["array" in the pointer arithmetic operation "array + 1" is not a pointer
                                           pointing to an array.]
   p++;                                 <-["p" in the pointer arithmetic operation "p++" is not a pointer
                                           pointing to an array.]
   *p = 1;
```

[Example 2]

```
void   func ( int   *array )
{
   int   x;
   int   *p = &x;
   array[1] = 0;   <-["array" in ["array[1]" is not a pointer pointing to an array.]
   p[1] = 1;       <-["p" in "p[1]" is not a pointer pointing to an array.]
:
}
```

R1.3.2

[Points of Message Indication in Agile^{+} Relief]

Subtraction performed on the pointers that point to different arrays.

[Example 1]

```
int    array1[10], array2[10];
int    *p1, *p2;
int    n;
p1 = array1;
p2 = array2;
n = p2 - p1;                    <-["p2" and "p1" of the pointer subtraction "p2 - p1" do not point to
                                      the same array.]
```

R1.3.3

[Points of Message Indication in Agile$^+$ Relief]:

Comparison is made between the addresses of objects of different types.

[Example1]

```
char   *p;
long   *q;
if( p < q ) {                    <-[The expression "p < q" compares two object addresses of different
                                           types (char *, long int   *).]
```

R1.3.4　(IPA/SEC-C V2, V3 only)

[Points of Message Indication in Agile^{+} Relief]:

There is a parameter of pointer type or array type to which restrict is modified.

[Example1]

```
void func( int* restrict p ) {    }          <-[There is a pointer that modified by restrict to the parameter
                                             of the function "func".]
```

R2.1.1

[Points of Message Indication in Agile^{+} Relief]

One of the expressions on the left or right side of the operators ==, !=, <=, >=, is of floating type.

[Example 1]

```
double   d1, d2;
   :
if( d1 == d2 )                    <-[The result of the comparison "d1== d2" between floating types
                                          might not be that expected.]
```

R2.1.2

[Points of Message Indication in Agile$^+$ Relief]

The loop counter of the for-statement is of floating-point type.

[Example 1]

```
float   f;
for( f = 0.0 ; f < 1 ; f += 0.1) { ~ }            <-[Using the floating type variable "f" as the loop counter
                                                              might result in unintended iterations.]
```

R2.1.3

[Points of Message Indication in Agile$^+$ Relief]

- Pointer type to structure or union is recorded in 1st or 2nd parameter of memcmp function.

[Example 1]

```
struct   STRUCT {
    unsigned   char   a;
    unsigned   char   b;
    unsigned   char   c;
}
int   func( struct STRUCT* s1, struct STRUCT* s2)
{
    if( memcmp( (char*)s1, (char*)s2,sizeof(struct STRUCT) )==0 ) <-[It is not ideal to perform the
                                                comparison of structure or union with function "memcmp".]
```

R2.2.1

[Points of Message Indication in Agile$^+$ Relief]

- The conditional judgment other than the followings is performed to the function call or variable recorded as if (boolean) and !if (boolean).

```
If(boolean)
if(!boolean)
if(boolean==0)
if(boolean!=0)
```

[Example 1]

```
file name:file.c
11: if( foo(10) ) {
 12:        ~

  if( foo(20) == 1 ) {
```

<-[For “foo(10)”(line 11 of “file.c”) used to judge the condition to be true or false, error might occur like in “foo(20) == 1” when !=0 and ==0 is not applied.]

R2.3.1

[Points of Message Indication in Agile$^+$ Relief]:

The operational result can not be represented in the unsigned type.

[Example1]

```
#define   ABC   1U
unsigned   int   x;
x  =  ABC  -  2;                <-[The value of the unsigned expression "1U - 2" cannot be
                                    represented by an unsigned type.]
if(x < 10)
```

[Example2]

```
#if  (1u  -  2u)  >  128        <-["1u-2u", which is performed with unsigned types, results in a
                                    value that cannot be represented by an unsigned type.]
```

[Example3]

```
unsigned int ui = 0xffffffff + 2 ;   <- [The result of constant expression “0xffffffff + 2” containing
                                     unsigned constant expression “0xffffffff” exceeds the bit width
                                     of unsigned int type.]
```

R2.3.2

[Points of Message Indication in Agile$^+$ Relief]

- Binary operation shall be braced when it is taken as an operand of ternary operation.
- According to the different return value from ternary operation, the conversion from underlying type to other different types might occur.

[Example 1]

x = **a > 0** ? **b + c** : 0;	<-[Operand"a > 0"of ternary operation "x = a > 0 ? b + c : 0"is not braced with (). For a clear priority, please embrace it.] <-[Operand"b+c"of ternary operation "x=a>0?b+c:0"is not braced with (). For a clear priority, please embrace it.]

[Example 2]

unsigned char uc1,uc2;	
unsigned short us;	
uc1 = (uc2==0) ? uc2 : us;	<-"(uc2 == 0) ? uc2 : us" of "uc1=(uc2 == 0) ? uc2 : us" will be converted to different types after operation. (underlying type before conversion : unsigned short int , and underlying type after conversion : unsigned char)

R2.3.3

[Points of Message Indication in Agile^{+} Relief]

- The type width of loop counter in for statement is less than that of comparison expression.

[Example 1]

```
unsigned int      n;
undigned char    counter;
            :
for( counter = 0 ;
        counter < n ;
       counter++ )                          <-[The type width of loop counter "counter" (unsigned char
                                            type) is less than that of comparison "n" (int type), it may lead
                                            to an infinite loop.]
{
```

R2.4.1

[Points of Message Indication in Agile$^+$ Relief]:

Expected precision cannot be obtained with the expression.

[Example1]

```
double   f;
int      x, y;
         :
f = x * y;                        <-[The expression "f = x * y" might not obtain the correct value. Use "f
                                         = (double)x * y" instead.]
if ( f < x * y )                <-[The expression "f < x * y" might not obtain the correct value. Use "f <
                                         (double)x * y" instead.]
```

[Example2]

```
long   long   x1, x2, x3 ;
int   i1, i2 ;
x1 = i1 * i2 ;                   <-[The correct value of the * operation in "x1 = i1 * i2" might not be able
                                         to be obtained.]
x2 = i1 << 2 ;                   <-[The correct value of the << operation in "x2 = i1 << 2" might not be
                                         able to be obtained.]
if ( x3 < ( i1 << 2 ) )      <-[The correct value of the << operation in "x3 < (i1 << 2)" might not be able
                                         to be obtained.]
```

R2.4.2

[Points of Message Indication in Agile^{+} Relief]

- The signed and unsigned descriptions coexist in comparison expression.
- Minus operation is performed to the signed and unsigned types.

[Example 1]

```
signed char      sc;
unsigned char    uc;
         :
if( sc == uc )                  <-[Unsigned "uc" and signed "sc" coexist in comparison
                                expression.]
```

[Example 2]

```
unsigned short   x = 1 ;
if( ( x – 2 ) > 10 )            <-[Minus operation "x-2"is done to the signed and unsigned
                                types.]
```

R2.5.1

[Points of Message Indication in Agile^{+} Relief]:

Data may be lost due to type conversion.

[Example1]

```
long   long   x1, x2, x3 ;
int   i1, i2 ;
x1 = i1 + i2 ;          <-[The correct value of the + operation in "x1 = i1 + i2" might not be able to be obtained.]
x2 = i1 - i2 ;          <-[The correct value of the - operation in "x2 = i1 - i2" might not be able to be obtained.]
if ( x3 < i1 + i2 )     <-[The correct value of the + operation in "x3 < i1 + i2" might not be able to be obtained.]
```

[Example2]

```
unsigned   char   func( ) {
                  :
return   256;           <-[Return value "256" exceeds the range of the return type of the function "func"  (return value: int, return type : unsigned char).]
```

[Example3]

```
Filename: file1.c
50: void   func( double x, int y ) { ~ }

Filename: file2.c
int   x, y;
:
func( x, y );           <-[The correct value cannot be passed because the 1st argument "x" of the function "func" and the corresponding parameter "x" in the function definition in line 50 of "file1.c" are of different types: integer and floating(argument : int , parameter : double ).]
```

R2.5.2

[Points of Message Indication in Agile$^+$ Relief]

A unary operator (the minus sign) has been attached to an unsigned variable.

[Example 1]

```
unsigned   long   ul = 0x80000001;
long   long   x;
x  =  -ul  ;                              <-[Appending a "minus" to the unsigned variable "ul" will not
                                               necessarily make it negative.]
```

R2.5.3

[Points of Message Indication in Agile$^+$ Relief]

The operational result may exceed the bit-width of the underlying type.

[Example 1]

```
unsigned   char   a = 0x1U ;
unsigned   char   b = 0xA0U ;
unsigned   int   x;
unsigned   int   y;
x = ~a ;                          <-["~a" might exceed the bit width of the underlying type unsigned
                                        char of "a".]
y = ( a + b ) << 2 ;           <-["(a + b) << 2" might exceed the bit width of the underlying type
                                        unsigned char of "(a + b)".]
```

R2.5.4

[Points of Message Indication in Agile^{+} Relief]

The number of shifts of the bitwise shift operator is a negative constant, or has exceeded the type size of the variable on the left.

[Example 1]

```
unsigned   int     x, y ;
x = y << -2 ;                <-[The result of the shift operation "y << -2" may vary depending on
                                    the compilers.]
x <<= -2 ;                   <-[The result of the shift operation "x <<= -2" may vary depending
                                    on the compilers.]
```

[Example 2]

```
unsigned   int     u1, u2 ;
u1 = u2 << 32 ;              <-[The number of bits specified for the shift operation "u2 << 32"
                                      exceeds the type width of the result.]
if ( u1 == ( u2 >> 32 ) )    <-[The number of bits specified for the shift operation "u2 >> 32"
                                      exceeds the type width of the result.]
u1 <<= 32 ;                  <-[The number of bits specified for the shift operation "u1 <<=
                                      32" exceeds the type width of the result.]
```

R2.6.1 (IPA/SEC-C V1 only)

[Points of Message Indication in Agile$^+$ Relief]:

A bit field of 1 bit and signed type is declared.

[Example1]

```
struct   STAG {
   signed      int      m1:1 ;       <-[The bit width of signed bit field "m1" is only 1 bit.]
   unsigned   int      m2: 1 ;
   signed      int      m3: 2 ;
};
```

R2.6.1(1)　(IPA/SEC-C V2, V3 only)

[Points of Message Indication in Agile$^+$ Relief]:

A bit field of 1 bit and types other than unsigned type is declared.

[Example1]

```
struct   STAG {
   signed      int      m1:1 ;        <-[The bit width of signed bit field "m1" is only 1 bit.]
   unsigned   int      m2: 1 ;
   signed      int      m3: 2 ;
};
```

R2.6.1(2)　(IPA/SEC-C V2, V3 only)

[Points of Message Indication in Agile$^+$ Relief]:

A bit field of 1 bit and types other than unsigned type or _ Bool type is declared.

[Example1]

```
struct   STAG {
   signed      int     m1:1 ;       <-[The bit width of signed bit field "m1" is only 1 bit.]
   unsigned   int     m2: 1 ;
   _Bool               m3: 2 ;
};
```

R2.6.1(3)　(IPA/SEC-C V2, V3 only)

[Points of Message Indication in Agile$^+$ Relief]:

The same with the message indication of R2.6.1(2). Please refer to R2.6.1(2).

R2.6.2

[Points of Message Indication in Agile^{+} Relief]

A bit-filed operation has been applied to the signed integer, type char, or a negative constant.

[Example 1]

```
int     x, y ;
x = y >> 2 ;                 <-[When the signed type variable "y" is assigned with a negative
                                    value, the result of a right shift may vary depending on the
                                    compiler.]
```

[Example 2]

```
int     i1, i2 ;
i1 = i2 & 0x30 ;    <-[The operand "i2" of the bit operation "i2 & 0x30" is signed.]
i1 <<= 3 ;           <-[The operand "i1" of the bit operation "i1 <<= 3" is signed.]
if ( ^i1 == 0xfe ) <-[The operand "i1" of the bit operation "^i1" is signed.]
```

R2.7.1(1)

[Points of Message Indication in Agile$^+$ Relief]

One side of the cast operator is of type pointer to an object, the other side is not integer type/void*/a pointer to an object.

[Example 1]

```
int    *ip;
double    d;
:
d  =  (  double  )ip;                     <-[The cast expression "(double)ip" is a conversion involving an
                                                 object pointer type and a type which is not an integer type,
                                                 object pointer type or void* type.]
```

R2.7.1(2)

[Points of Message Indication in Agile^{+} Relief]

- Cast is performed between the pointer to object type and that not addressing to object type, the type of large width, and void * type.

[Example 1]

```
int    *pv;
pv = (int *)( ( unsigned long ) pv | 1);       <-[ The pointer type in cast expression “(unsigned
                                                long)pv”is cast to a smaller type. (Cast Target Type: unsigned
                                                long int, Expression to be Cast: int *)]
```

R2.7.1(3)

[Points of Message Indication in Agile+ Relief]

- Cast is performed between the pointer to object and that to the type of less type width.
- Cast is performed between the pointer to function and the pointer to other functions, or that not of int type.

[Example 1]

```
int   *pv;
pv = (int *)( ( unsigned long ) pv | 1);      <-[The pointer type in cast expression “(unsigned
                                              long)pv”is cast to smaller type.   (Cast Target Type: unsigned
                                              long int, Expression to be Cast: int *)]
```

[Example 2]

```
int      (*fp)(void);
int      *ip;
ip   =   (int *)fp;                           <-[ Cast expression “(int)fp” is the conversion between the
                                              pointer type to function and the type other than int.   (Cast
                                              Target Type: unsigned long int, Expression to be Cast: int *)]
```

R2.7.2

[Points of Message Indication in Agile$^+$ Relief]

The cast has deleted the const or volatile qualifying the space that is pointed by a pointer.

[Example 1]

```
const   int   x = 5;
int   *p = (int*)&x;               <-["(int *)&x" negates the const/volatile qualifying the space pointed to by
                                          the pointer "&x" of type "const int*".]
```

R2.7.3 (IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile$^+$ Relief]

- The assignment expression and the comparison expression for negative values are recorded in the variable of pointer type.

[Example 1]

```
char *p;
   :
p = -1;                         <-[Whether pointer type “p” can be used as negative is
                                subjected to the compiler type.]
```

R2.7.3 (IPA/SEC-C V3 only)

[Points of Message Indication in Agile^{+} Relief]

- The assignment expression and the comparison expression for negative values are recorded in the variable of pointer type.
- The left and right types of the conditional expression are different for the pointer type and the non-pointer type.

[Example 1]

```
char *p;
   :
p = -1;                                  <-[Whether pointer type “p” can be used as negative is
                                         subjected to the compiler type.]
```

[Example 2]

```
int data, *dp;
       :
if ( data == dp )                        <-[The left operand and right operand of the conditional
                                         expression "data == dp" are of different types : pointer and
                                         non-pointer(left : int, right : int *).]
```

R2.8.1

[Points of Message Indication in Agile^{+} Relief]

The function is defined in K&R format or does not specify parameters.
No parameter is found in the function decelerator.

[Example 1]

```
int   func1( x )            <-[Function "func1" is defined in K&R style or non-parameter form.]
int    x;
{
   :
}
void   func2( )             <-[Function "func2" is defined in K&R style or non-parameter form.]
{
   :
}
```

[Example 2]

```
int   func( ) ;             <-[Parameters are not specified in the function decelerator.]
void   (*fp)( );            <-[Parameters are not specified in the function decelerator.]
```

R2.8.2(1)

[Points of Message Indication in Agile$^+$ Relief]:

There are variable parameters in the function declaration or definition.

No parameter has been specified in the function decelerator.

[Example1]

```
int   func( int   n, ... ) ;                  <-[The parameter declaration of the function "func( int n, ...)"
                                                          contains "...".]
int   func( int   n, ... ) {                  <-[The parameter declaration of the function "func( int n, ...)"
                                                          contains "...".]
```

[Example2]

```
int   func( ) ;          <-[Parameters are not specified in the function decelerator.]
void (*fp)( );          <-[Parameters are not specified in the function decelerator.]
```

R2.8.2(2)

[Points of Message Indication in Agile$^+$ Relief]

- Uncertain number of parameters is used in function declaration and function definition.

[Example 1]

```
int   func( int   n, ... ) ;                 <-[… is used in the parameter declaration of function "func" (int
                                             n, …).]
int   func( int   n, ... ) {                 <-[… is used in the parameter declaration of function "func" (int
                                             n, …).]
}
```

R2.8.3

[Points of Message Indication in Agile^{+} Relief]:

The function is called without the corresponding function declaration.

A function is defined without the corresponding function declaration.

The function decelerator has not been designated with a parameter.

[Example1]

```
x = func( 10 );                <-[There is no function declaration corresponding to the function call
                                     "func".]
```

[Example2]

```
void   func( int x )   <-[Function "func" is defined without prior declaration.]
{
   :
}
```

[Example3]

```
int   func( ) ;          <-[Parameters are not specified in the function decelerator.]
void (*fp)( );           <-[Parameters are not specified in the function decelerator.]
```

R3.1.1(1)

[Points of Message Indication in Agile^{+} Relief]

The array is declared without specifying its length.

[Example 1]

```
Filename: file.c
   #include   "head.h"
   int   data[ 256 ] ;

Filename: head.h
   extern   int   data[ ] ;        <-[The length of an array has been omitted.]
```

R3.1.1(2)

[Points of Message Indication in Agile^{+} Relief]

The same with the message indication of P3.1.1(1). Please refer to P3.1.1(1).

R3.1.2

[Points of Message Indication in Agile^{+} Relief]

- Loop counter is not judged in the conditional expression of for statement.

[Example 1]

```
int   GetId(void);
int   id;
int   array[10];
int   i;
for( i = 0 ; ( id = GetId() ) != 0 ; i++ ) {   <-[ The check of loop counter scope is not contained in loop
                                                condition.]
    if( i >= 10 ) { /*Check for Array Scope*/
        break;
    }
    array[i] = id;
}
```

R3.1.3　(IPA/SEC-C V2 only)

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

R3.1.3 (IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

- The array initialization indicator is out of range.
- The number of elements in the array using the initialization indicator is omitted.
- The end of an array using an initialization indicator has not been initialized.

[Example 1]

```
char str[5] = {'a','b','c','d', [5] = '\0'};     <-[The initialization indicator "5" for array "str" is  greater than
                                                 the array size.]
```

[Example 2]

```
char str[] = {'a','b','c','d','\0',[2] = ','};     <-[The length of the array using the initializer is omitted.]
    :
char buf[3];
    :
strcpy(buf, str);
```

[Example 3]

```
#define END -1
#define BUFSIZE 10
int array1[5] = {1,2,3,[2] = 4,END};     <-[The end of the array using the initialization indicator has not
                                         been initialized.]
int array2[BUFSIZE] = {1,2,3,4,[5] = END};     <-[The end of the array using the initialization indicator
                                               has not been initialized.]
```

R3.1.4　(IPA/SEC-C V2 only)

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

R3.1.4　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

- A variable length array is declared.

[Example 1]

```
void　sub( int n ) {
      char　buff [ n ] ;          <-[Variable-length array "buff" is declared.]
              :
}
void　func( void ) {
      sub( 0 ) ;
      sub( 4096 ) ;
      sub( -1 ) ;
            :
```

R3.1.5(1)　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile^{+} Relief]

- Use the sizeof operator to determine the size of a pointer.

[Example 1]

```
char   strdata[8];
char   *p   =   strdata;
char   check[8];
if(memcmp(p, check, sizeof(p)) == 0 )       <-[An error may exist in "sizeof(p)" which trying to get the
                                            size of a pointer type.]
```

R3.1.5(2)   (IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

- Use the sizeof operator to determine the size of the parameter of the array type.

[Example 1]

```
#define ARRAYSIZE 10
void sub( char c[ARRAYSIZE] ) {
  printf( "%d\n", sizeof(c) ); /* Display 4   */  <-[ The sizeof() operation that is operated for argument
                                                   "c" of array type may be unreasonable.]
}
void main( ) {
  char c[ARRAYSIZE];
  printf( "%d\n", sizeof(c) ); /* Display 10   */
  sub( c );
}
```

R3.2.1

[Points of Message Indication in Agile^{+} Relief]

- The contents of division by zero or possible are found.

[Example 1]

```
x = y / 0 ;                             <-[“y / 0”is the division by zero.]
```

[Example 2]

```
int   x = 0;
if( data != 1 ) {
    x = 2;
}
return   data/x ;                       <-[A variable "x" possibly being zero is used in division.]
```

R3.2.2

[Points of Message Indication in Agile^{+} Relief]:

The return value of functions malloc or fopen is referenced without checking if it is NULL.

The parameter of pointer type is referenced to without checking if it is NULL.

An automatic variable that may be set to address 0 is referenced.

Checking NULL after the pointer operation.

[Example1]

```
char   c, a[10], x ;
11: char   *p = NULL ;
12: char   *q = NULL ;
:
if ( mode == 0) {
p = &x ;
q = a ;
}
c = *p ;                          <-[ The variable "p", assigned with the 0 address in line 11, might
                                       access the 0 address.]
strcpy ( q, "abc" ) ;          <-[ The variable "q", assigned with the 0 address in line 12, might access
                                       the 0 address.]
```

[Example2]

```
char   *p ;
10: p = malloc( sizeof(char) * 100 ) ;
*p = ' \0'   ;                <-[The 0 address might be referenced because the variable "p" might
                                     have been assigned with the 0 address in line 10.]
:
20: fp = fopen( filename, "r" ) ;
size = fread( buf, sizeof(buf), 1, fp ) ;    <-[The 0 address might be referenced because the variable
                                                  "fp" might have been assigned with the 0
                                                  address in line 20.]
```

[Example3]

```
10: int    func( int    *p )
{
        int     x;
        x = *p;                       <-[The 0 address might be referenced because the variable "p", as
                                              a parameter, might have been assigned with the 0 address in
                                              line 10.]
```

[Example4]

```
p = G_xp ;
*p = 10 ;                             <-[The variable "p" compared with the 0 address in line 12 might
                                              access the 0 address.]
12: if ( p == NULL ) {
:
}
```

R3.3.1

[Points of Message Indication in Agile^{+} Relief]:

The return value of the non-void function is ignored.

The return value of the function returning an abnormal value has not been checked.

The function using the same name as that of the identifier registered under the labels [NULL_RETURN_FUNCTION], [RETURN_CHECK_FUNCTION], and [SET_VARIABLE_FUNCTION] of the identifier file is checked as an object. For more details regarding the registration to an identifier file, see the -f option in "Command Manual".

[Example1]

```
int     func( );
   :
func( 10 ) ;                        <-[The return value of the function call "func" is not used.]
```

[Example2]

```
10: c = fgetc( in_fp ) ;
fputc( c, out_fp ) ;             <-[The variable "c" is assigned the return value of the function "func" in
                                       line 10 and then referenced without being judged.]
```

R3.3.2 (IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile^{+} Relief]:

The return value of the functions malloc or fopen is referred to without checking if it is NULL.

An automatic variable, which may be set to address 0, has been referenced.

[Example1]

```
char *p ;
10: p = malloc( sizeof(char) * 100 ) ;
*p = ' \0'   ;                  <-[The 0 address might be referenced because the variable "p" might
                                        have been assigned with the 0 address in line 10.]
:
20: fp = fopen( filename, "r" ) ;
size = fread( buf, sizeof(buf), 1, fp ) ;   <-[The 0 address might be referenced because the variable
                                                     "fp" might have been assigned with the 0
                                                     address in line 20 .]
```

[Example2]

```
 char   c, a[10], x ;
11: char   *p = NULL ;
12: char   *q = NULL ;
:
if ( mode == 0 ) {
p = &x ;
q = a ;
}
c = *p ;                           <-[The variable "p", assigned with the 0 address in line 11, might
                                        access the 0 address.]
strcpy ( q, "abc" ) ;         <-[The variable "q", assigned with the 0 address in line 12, might access
                                        the 0 address.]
```

R3.3.2 (IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]:

The return value of the functions malloc or fopen is referred to without checking if it is NULL.

An automatic variable, which may be set to address 0, has been referenced.

The parameter of pointer type is referenced to without checking if it is NULL.

[Example1]

```
char *p ;
10: p = malloc( sizeof(char) * 100 ) ;
*p = ' \0'   ;                <-[The 0 address might be referenced because the variable "p" might
                                   have been assigned with the 0 address in line 10.]
:
20: fp = fopen( filename, "r" ) ;
size = fread( buf, sizeof(buf), 1, fp ) ;   <-[The 0 address might be referenced because the variable
                                                 "fp" might have been assigned with the 0
                                                 address in line 20 .]
```

[Example2]

```
  char   c, a[10], x ;
11: char   *p = NULL ;
12: char   *q = NULL ;
:
if ( mode == 0 ) {
p = &x ;
q = a ;
}
c = *p ;                      <-[The variable "p", assigned with the 0 address in line 11, might
                                   access the 0 address.]
strcpy ( q, "abc" ) ;       <-[The variable "q", assigned with the 0 address in line 12, might access
                                   the 0 address.]
```

[Example3]

```
10: int    func( int    *p )
{
        int     x;
        x = *p;                         <-[The 0 address might be referenced because the variable "p", as
                                                a parameter, might have been assigned with the 0 address in
                                                line 10.]
```

R3.4.1

[Points of Message Indication in Agile$^+$ Relief]

There is a recursive function.

[Example 1]

```
int    func( int    x )      <-[The function "func" is a recursive function.]
{
    :
y = func( z );
    :
}
```

R3.5.1

[Points of Message Indication in Agile$^+$ Relief]:

The statements if and else if are not followed by an else statement.

[Example1]

```
if ( x > 0 ) {                <-[There is no else statement at the end of an if else if statement. Even
                                     when all cases have been accounted for, it is best to include
                                     an empty else statement.]
        :
} else if ( x < 0 ) {
        :
}
```

R3.5.2

[Points of Message Indication in Agile^{+} Relief]:

There is no default label in the switch statement.

The default label is placed before the case label.

[Example1]

```
switch ( x ) {                  <-[This switch statement contains no default label.]
case 1:
    x = a + b;
    break;
case 2:
    x = c + d;
    break;
}
```

[Example2]

```
switch ( x ) {
default:                         <-[The label default has been included before the label case in a
                                       switch statement.]
    no++
    break;
     :
case 2 :
    no += x;
}
```

R3.5.3

[Points of Message Indication in Agile^{+} Relief]

- “==“ or “!=“ is used in the conditions of for statement.

[Sample1]

```
for( i = 0 ; i == n ; i++ )     <-[ In loop processing of “for (i = 0 ; I == n ; i++)”, the loop condition “I ==
                                n” may be incorrect.]
    case 1:
          x = a + b;
          break;
    case 2:
          x = c + d;
          break;
}
```

R3.6.1

[Points of Message Indication in Agile^{+} Relief]

- Increment / decrement and assignment with update sequence undefined are found in ANSI.

[Example 1]

```
i = j + j++;                  <-[The update timing of “j++” is not defined in ANSI, thus the value a of”j”
                              cannot be guaranteed.”j++”]
```

R3.6.2

[Points of Message Indication in Agile^{+} Relief]

- The function call with calling sequence undefined is found in ANSI.

[Example 1]

```
word_data = get_byte1( ) << 8 | get_byte2( ) ;
                                    <-[The calling sequence of function call "get_byte1"and
                                    "get_byte2" are not defined in ANSI, it is better to separate
                                    into two lines.]
```

[Example 2]

```
 file name:file.c
      x = Gx + func( );            <-[The call timing of function "func" for update variable "Gx" in
                                   line 100 of "file.c" is not defined, thus the value of "Gx" cannot
                                   be guaranteed.]
          :
      int   func( void )
      {
          :
100:      Gx += 2;
          :
      }
```

R3.6.3　(IPA/SEC-C V2, V3 only)

[Points of Message Indication in Agile$^+$ Relief]

- Within the sizeof operator, there is an update expression, a function call.

[Example 1]

```
x = sizeof ( y++ ) ;                    <-["y++" is in sizeof, so updating is not performed.]
```

R3.11.1 (IPA/SEC-C V3 only)

[Points of Message Indication in Agile^{+} Relief]

- The variable modified with volatile is not under exclusive control in multithread function.

[Example 1]

```
#define ON 1

#define OFF 0

pthread_mutex_t lock;

volatile int comm_io;

void thread_1() {        /* thread_1 is multithread function */

  comm_io = ON;       <-[The variable "comm_io" qualified by volatile which used in multithread
                          function "thread_1" is not in exclusive control.]

}

void thread_2() {        /* thread_2 is multithread function */

  comm_io = OFF;      <-[The variable "comm_io" qualified by volatile which used in multithread
                          function "thread_2" is not in exclusive control.]

}

void thread_3() {        /* thread_3 is multithread function */

  while(comm_io == ON) {    <-[The variable "comm_io" qualified by volatile which used in
                               multithread function "thread_3" is not in exclusive control.]

        :

  }

}
```

R3.11.2 (IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

- The bit field is not under exclusive control in multi-threaded functions.

[Example 1]

```
#define ON 1
pthread_mutex_t lock;
struct FLAG {
  unsigned int flag1 : 1;
  unsigned int flag2 : 1;
} f;
int thread_1(void) {   /* thread_1 is multithread function. */
  f.flag1 = 1;     <-[The bit field "f.flag1" which used in multithread function "thread_1" is not in
                     exclusive control.]
         :
  return data;
}
int thread_2(void) {   /* thread_2 is multithread function. */
  f.flag2 = 1;     <-[The bit field "f.flag2" which used in multithread function "thread_2" is not in
exclusive control.]
         :
  return data;
}
```

3.2 Maintainability

M1.1.1 (IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile$^+$ Relief]

- The unused functions, internal variables, external variables, parameters and labels are found.

[Example 1]

```
labelA;                    <-[The label of “labelA” is not used.]
  :
```

LabelA is not referenced by goto statement.

M1.1.1　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

- The unused types, tags, macros, functions, internal variables, external variables, parameters and labels are found.

[Example 1]

```
labelA;                    <-[The label of “labelA” is not used.]
   :
  LabelA is not referenced by goto statement.
```

M1.1.2　(IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

M1.1.2(1)   (IPA/SEC-C V3 only)

[Points of Message Indication in Agile^{+} Relief]

- Line-splicing is used in "// " comments.
- Comment out the source code.

[Example 1]

```
extern bool_t b ;
void f ( void ) {
    int n = 0 ;    // Note \     <-[Line-splicing is used in "// " comments.]
    if ( b )          The judgment sentence is not evaluated because of the comment continuation line.
    {
         ++n ;   /* It always executes it. */
    }
```

[Example 2]

```
/* if ( x >= 16 ) { */      <- [The source code may be commented out.]
if ( x > 16 ) {
      :
```

M1.1.2(2)　(IPA/SEC-C V3 only)

The same with the message indication of M1.1.2(1), please refer to M1.1.2(1).

M1.2.1(1)

[Points of Message Indication in Agile$^+$ Relief]

- Commas are found in the declaration statement.
- Uses a comma outside an initialization expression or update expression in a for statement.

[Example 1]

```
int   *a,b ;                                        <-[ Comma is found in the declaration statement.]
```

[Example 2]

```
x=1 ,y=2;                          <-[" x=1,y=2"   uses a comma outside an initialization expression or update
                                      expression in a for statement.]
```

M1.2.1(2)

[Points of Message Indication in Agile$^+$ Relief]

- Uses a comma outside an initialization expression or update expression in a for statement.

[Example 1]

```
x=1 ,y=2;                    <-[“ x=1,y=2”   uses a comma outside an initialization expression or update
                                expression in a for statement.]
```

M1.2.2

[Points of Message Indication in Agile$^+$ Relief]

- Suffix of l is added (lowercase English character l).
- The type of integer constant is unsigned, no suffixes of U and u are added.

[Example 1]

```
long   x = 32768l ;                    <-[The easily confused suffix of “l” is used in “327681”.]
```

[Example 2]

```
unsigned   int   x = 0xA123 ;          <-[Suffix of “U” is not used in “0xA123”.]
```

M1.2.3

[Points of Message Indication in Agile$^+$ Relief]

- A line feed is found in character constant (‘‘).

[Example 1]

```
‘¥0
12’ ;                                   <-[According ANSI, no line feed code is allowed in character
                                                  constant.]
```

M1.3.1

[Points of Message Indication in Agile$^+$ Relief]:

There are a comparing operator, a logic operator and a constant in the conditional expression of switch statement.

There is only one case label in the switch statement.

[Example1]

```
switch ( x > 0 )                <-["x > 0" is used as the condition of a switch statement.]
{ ~ }
switch ( 1 )                    <-["1" is used as the condition of a switch statement.]
{ ~ }
```

[Example2]

```
switch ( x ) {                  <-[The switch statement contains only one case label.]
case ONE :
        :
        break;
default:
        break;
}
```

M1.3.2

[Points of Message Indication in Agile$^+$ Relief]

The case label or default label is not contained in the { } after the switch statement.

[Example 1]

```
switch( x ){
case   0 :
   {
      y = 10;
case   1:                                  <-[The label case is not within the { } immediately following the
                                                 switch.]
      z = 100;
      break;
   }
```

M1.3.3

[Points of Message Indication in Agile^{+} Relief]

A variable is declared without specifying the specifier.

A function without return value should be defined as void.

[Example 1]

```
const   x = 1, y = 2;        <-[A declaration is made with no explicit type specifier.]
z;                            <-[A declaration is made with no explicit type specifier.]
```

[Example 2]

```
func( int   x )                <-[The function "func" has no return value and should be declared as
                                      void.]
{
   :
   return;
}
```

M1.4.1

[Points of Message Indication in Agile$^+$ Relief]

The expression on both sides of the operators && or ||, is not a primary expression.

[Example 1]

```
if( p != 0 && *p == 1 )            <-["p != 0" in the && expression is not a primary expression.]
                                         <-["*p == 1" in the && expression is not a primary
                                            expression.]
if ( x == 1 || y > 5 && z != 0 )   <-["x == 1" in the || expression is not a primary expression.]
                                         <-["y > 5" in the && expression is not a primary
                                            expression.]
                                         <-["z != 0" in the && expression is not a primary
                                            expression.]
                                         <-["y > 5 && z != 0" in the || expression is not a
                                            primary expression.]
```

M1.4.2

[Points of Message Indication in Agile$^+$ Relief]

It is better to add a pair of parentheses.

[Example 1]

```
if ( c = input( ) != 0 )             <-[In the expression "c = input() != 0", an assignment operation coexists
                                                   with a comparison operation. Parentheses ( ) may
                                                   have been accidentally omitted."]
```

[Example 2]

```
if ( 0 < x < 10 )                    <-[This compares the result of "0 < x" with "10", which may not be the
                                              intention.]
```

[Example 3]

```
if( x == 0 && y == 0 || z == 0)     <-["x == 0 && y == 0 || z == 0" contains both a && operation and a ||
                                                   operation. Parentheses ( ) may have been omitted
                                                   accidentally.]
```

[Example 4]

```
x = ( a > 0 ) ? 0 : ( b > 0 ) ? 1 : 2 ;     <-[In the expression "(a>0)?1:(b>0)?1:2", multiple ternary
                                                       operations are used together. Use parentheses ( )
                                                       to show association explicitly.]
```

[Example 5]

```
x = a >> b & c ;                     <-[ "a >> b & c" mixes bit and shift operations with no parentheses ( ).
                                              Use parentheses ( ) to   show precedence explicitly.]
```

M1.5.1

[Points of Message Indication in Agile^{+} Relief]

Cast a function name to a non-function pointer.

[Example 1]

```
int   x , func( void ) ;
x = func ;                    <-[The expression "x = func" referencing the function name "func"
                                        might contain a mistake.]
if ( func != 0 )              <-[The expression "func != 0" referencing the function name "func"
                                        might contain a mistake.]
```

M1.5.2

[Points of Message Indication in Agile^{+} Relief]:

A non-boolean value is used in the conditional expression.
Comparison with 0 in the conditional expression.

[Example1]

```
if ( x = y )                <-[The assignment expression "x = y" is used as a condition.]
```

[Example2]

```
Filename: file.c
  int    x, y ;
2: x = 100;
3: y = 100;
  if ( x )                  <-["x" is not a bool value (a non-bool value is assigned in line 2 of
                                 "file.c").]
         :
  if ( !y )                 <-["y" is not a bool value (a non-bool value is assigned in line 3 of
                                 "file.c").]
```

[Example3]

```
Filename: file.c
11: if ( foo(10) ) {
:
if ( foo(20) == 1 ) {       <-[The comparison operation "foo(20) == 1", which is not of the form !=0
                               or ==0 and is performed with "foo(10)" (in line 11 of "file.c")
                               which is used as a boolean value might contain a mistake.]
```

M1.6.1

[Points of Message Indication in Agile^{+} Relief]

- The same loop counter is used both outside and innerside of the for statement.

[Example 1]

```
20: for( i = 0 ; i < 5 ; i++ ) {          <-[The same loop counter "i" is used in the for statements on
                                            this line and line 30.   ]
        {
            :
30:     for( i = 10 ; i < 10 ; i++ )
```

M1.6.2(1)

[Points of Message Indication in Agile$^+$ Relief]

A union is used.

[Example 1]

```
union   UN         <-[A union is used.]
{
   long     m;
   short   n[2];
};
```

M1.6.2(2)

[Points of Message Indication in Agile^{+} Relief]

- The different member name is referenced for the same union variable of the same process route.

[Example 1]

```
union {
    int      m1;
    char   m2;
} x;
10: x.m1 = 1;
c = x.m2;                          <-[The different member “x.m2”is referenced for the union
                                   member “x.m1” set in line 10.]
```

M1.7.1

[Points of Message Indication in Agile^{+} Relief]

- The identifier of the same name valid in outside is declared innerside.
- Typedef name is reused.
- Tag name is reused.
- The name of variable and function containing union is reused.
- The same identifier is reused within the same block and for different name space.

[Example 1]

```
file name :file.c
10: unsigned   char   *cp;
    void   func( )
    {
        unsigned   char   *cp;          <-[The variable "cp" declared is of the same name with the
                                          variable "cp"outside the function (Line 10 of "file.c").]
 }
```

[Example 2]

```
file name:file.c
   void   func1(void)
   {
3:      typedef   char   BYTE;
      :
    }
    void   func2(void)
    {
        typedef   char   BYTE;          <-[Typedef name "BYTE" is the same with that in line 3 of
                                          "file,c".]
      :
    }
```

[Example 3]

```
file name:file1.c
  10: struct   tag{ int   a; long   b; };
file name:file2.c
        struct   tag{ int   x; long   y; };
                                         <-[Tag declaration "tag" is different from that in line 10 of
                                         "file1.c".]
```

[Example 4]

```
   file name:file.c
  1: static   int   name;
      void   func (void)
      {
         enum {
             name ,                     <-["name" is the same with that of the static variable in line 1 of
                                        "file.c".]
      }
```

[Example 5]

```
   file name:file.c
  10: struct   name {
           int   name;                  <-[Member name is the same with that of the tag in line 10 of
                                        "file.c".]
         :
       };
       char   *name;                    <-[Variable name is the same with that of the tag in line 10 of
                                        "file.c".]
```

M1.7.2

[Points of Message Indication in Agile^{+} Relief]

The following macros are redefined or invalidated:
__LINE__
__FILE__
__DATE__
__TIME__

The ANSI reserved identifiers have been defined or invalidated as macro names.

[Example 1]

```
#define   __FILE__   abc        <-[The predefined macro "__FILE__" is defined again with #define.]
```

[Example 2]

```
#define   errno   -1                              <-[In the #define line, "errno" is used as macro.]
#define   malloc( a )   mymalloc( a )     <-[In the #define line, "malloc" is used as macro.]
```

M1.7.3　(IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

M1.7.3 (IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

- It uses identifiers that begin with an underscore.

[Example 1]

```
#define __MACRO_A    0        <-[The identifier "__MACRO_A" that begin with an underscore is
                              reserved for ANSI, it is possible to lead to undefined behavior. ]
int _valA;                    <-[The identifier "_valA" that begin with an underscore is reserved
                              for ANSI, it is possible
to lead to undefined behavior. ]
void _funcA();                <-[The identifier "_funcA" that begin with an underscore is reserved
                              for ANSI, it is possible to lead to undefined behavior. ]
```

M1.8.1

[Points of Message Indication in Agile$^+$ Relief]:

There is a side-effect expression in the right part of the && operator, || operator or the second and third expressions of the ternary operator.

[Example1]

```
int    i, x[100];
volatile    int    z;
if ( i == 0 && x[ i++ ] == 0 )                  <-[Update expression "i++" is not executed in all cases.]
if ( x[ i ] == 0 || func( i ) == 0 )        <-[Function "func" is not executed in all cases.]
x[ i ] = ( x[ i ] == 0 )? i : z ;                <-[volatile variable "z" is not executed in all cases.]
```

M1.8.2　(IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile$^+$ Relief]:

Mismatch of the parentheses, (and), in the replacement string of the #define statement.

Mismatch of the { and }　within the replacement string of the #define statement.

The replacement string of the #define statement consists only of the following types: char, short, int, long, float, double, signed, unsigned and void.

[Example1]

```
#define　AddTen(x)　 (x + 10　　　<-[ The replacement string of the macro function "AddTen" is not
                                  enclosed as a whole with parentheses ( ) or curly
                                  brackets { }.]
```

[Example2]

```
#define　PSI32　int*　　　<-[ typedef can be used instead of macro "PSI32".]
PSI32　a, b;
```

M1.8.2 (IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]:

Mismatch of the parentheses, (and), in the replacement string of the #define statement.

Mismatch of the { and } within the replacement string of the #define statement.

The replacement string of the #define statement consists only of the following types: char, short, int, long, float, double, signed, unsigned and void.

A formal argument specification in the macro definition is not complete.

The closing parenthesis of the macro call is missing.

Arguments in the macro substitution string are not enclosed in ().

The macro function substitution string is not entirely enclosed in () or {}.

Reserved words are used in macro names.

[Example 1]

```
#define   AddTen(x)    (x + 10
```

<-[The replacement string of the macro function "AddTen" is not enclosed as a whole with parentheses ( ) or curly brackets { }.]

[Example 2]

```
#define   PSI32   int*
PSI32   a, b;
```

<-[typedef can be used instead of macro "PSI32".]

[Example 3]

```
#define   F(a, b
```

<-[Specification of parameters in a macro definition is incomplete.]

[Example 4]

```
#define   F(a,b)   ((a) + (b))
          F(1, 2)

End of file
```

<-[The macro call "F" is made with no ending parentheses).]

[Example 5]

```
#define   F(a, b)     (a * b)

x = F( 1 + 5, 10);
```

<-[The parameters "a","b" in the replacement string of the macro "F" are not enclosed with parentheses ( ).]

[Example 6]

```
#define   AddTen(a)     (a) + 10

x = AddTen( 20 ) * 2 ;
```

<-[The replacement string of the macro function "AddTen" is not enclosed as a whole with parentheses ( ) or curly brackets { }.]

[Example 7]

```
#define while ( EXP )   for( ; ( EXP ) ; );
#define unless( EXP )   if ( ! ( EXP ) )
```

<-[The macro name "while" is a keyword.]

/* The reserved word "If" in the substitution character string is not pointed out. */

M1.8.3

[Points of Message Indication in Agile$^+$ Relief]

- #line is used.

[Example 1]

```
#line   100   “abc”                    <-[#line is not recommended.   shall be avoided.]
```

M1.8.4

[Points of Message Indication in Agile$^+$ Relief]

A trigraph sequence is used.

[Example 1]

```
??=include    "head.h"    <-[The trigraph sequence "??=" is used.]
```

M1.8.5

[Points of Message Indication in Agile^{+} Relief]

An octal constant other than 0 or an octal escape sequence is used.

[Example 1]

```
int    x = 010 ;                        <-[The octal constant "010" is used.]
```

[Example 2]

```
c = '\12' ;                             <-[The octal escape sequence "\12" is used.]
strcpy ( s, "abc\14" ) ;       <-[The octal escape sequence "\14" is used.]
```

M1.9.1

[Points of Message Indication in Agile$^+$ Relief]

A semicolon is immediately followed by the ')' of the statements if, for and while.

There is a null statement such as :; or ;;.

[Example 1]

```
if ( x == 0 ) ;                    <-[A semicolon immediately following an if statement, for statement
                                          or while statement may cause an error.]
        x = y;
```

[Example 2]

```
int    x ;    ;            <-[An unnecessary null statement may exist.]
switch( y ) {
case 1: ;                  <-[An unnecessary null statement may exist.]
```

M1.9.2

[Points of Message Indication in Agile$^+$ Relief]

- The variables used in the conditions of for, while, do-while statements are not updated.

[Example 1]

```
while ( x * 10 – y > 0 ) {        <- [“x * 10 – y > 0” is the unchanged loop condition.]
        :
}
```

x and y are not updated in while statement.

M1.10.1 (IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile^{+} Relief]

The violation against this rule is not checked in Agile^{+} Relief.

M1.10.1　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

- Using a literal.

[Example 1]

```
#define　SIZE　256　　/* It is not indicated in the macro definition */
int　N[256] ;　　　　　<-[The literal 256 is used. (literal type INTEGER)]
```

M1.11.1

[Points of Message Indication in Agile^{+} Relief]

- Some parameters of pointer type and array type are not updated.

[Example 1]

```
void   func( int   p[ ], unsigned   int   n )
                                          <-[Because the space pointed by the parameter of"int [ ]" is not
                                          updated, the const qualification is not performable.]
{
    int   i;
    int   num = 0;
    for( i = 0; i < n; i++ ) {
        num += p[ i ];
    }
}
```

M1.11.2

[Points of Message Indication in Agile$^+$ Relief]

- Among the function groups of the same name with the identifiers registered under the label of [PARALLEL_FUNCTION], the common external variable and static variable are not qualified by volatile.

  For the contents registered in the identifier file, please refer to –F option of "Command Manual".

[Example 1]

```
     /* This message will be output when intr01, intr02 and intr03 are registered under the
                                              [PARALLEL_FUNCTION] label of the identifier file. */
int   common ;                    <-[The external variable"common"used in multiple functions is
                                  not qualified by volatile.]
void intr01()
{
    x1 = common ;
    x2 = common ;
    if( x1 != x2 )
          :
void intr02()
{
    common = 2;
          :
void intr03()
{
    common = 3;
```

M1.11.3

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

M1.12.1

[Points of Message Indication in Agile^{+} Relief]

An undefined redundant sign exists in the preprocessing directives.

An undefined preprocessing directive exists.

[Example 1]

```
#ifdef   DEG   1                  <-[In ANSI, the extra symbol "1" is not defined and cannot be
                                     used.]
```

[Example 2]

```
#asm                              <-[In ANSI, the preprocessing directive "#asm" is not defined and
                                     cannot be used.]
```

M2.1.1

[Points of Message Indication in Agile$^+$ Relief]:

Mismatch between the initializer list and the construction of the initialized array/struct/union.

[Example1]

```
int   data[2][3]  =                        <-[Initialization of the array/structure/union "data" is
                                                              inconsistent with its makeup.]
      1, 2, 3, 4, 5, 6
} ;
struct   A {
      int   a1;
      int   a2;
};
struct   A   adata[2] = { 1, 2, 3, 4 } ;   <-[Initialization of the array/structure/union "adata" is
                                                              inconsistent with its makeup.]
struct B {
      int           b1;
      struct   A   b2;
} ;
struct   B   bdata = { 1, 2, 3 } ;        <-[Initialization of the array/structure/union "bdata" is inconsistent
                                                             with its makeup.]
int   cdata[7] = { 1, 2, 3, 4, 5 } ;      <-[Initialization of the array/structure/union "cdata" is inconsistent
                                                              with its makeup.]
```

M2.1.2

[Points of Message Indication in Agile^{+} Relief]

- The statement to be controlled in if, else, for, while, do-while and switch statement is not braced with { }.

[Example 1]

```
if ( x == 0 )              <-[It is recommended to brace the if statement with { }.]
    x = 10;
```

M2.2.1

[Points of Message Indication in Agile$^+$ Relief]

There is an external variable or a global static variable used only in one function.

[Example 1]

Filename: head.h

```
  extern   int    xx ;          <-[The external variable "xx" that can be used in multiple functions
                                       was referenced only in the function "func" in line 20 of file.c.
                                       An internal static variable could be used instead.]
```

Filename: file.c

```
  #include   "head.h"
            :
20: void   func( int   in )
{
  xx = 0;
/*The external variable xx is used only in this function.*/
}
```

M2.2.2

[Points of Message Indication in Agile^{+} Relief]

- The external variable only used in one file is found.

[Example 1]

```
file name: file.c
int     x = 0 ;          <-[The external variable “x” is only used in file “file.c”.]
void   x_add( void )   { x++; }
int   x_ref( void )   { return x; }
  The variable x is only used in file.c.
```

M2.2.3

[Points of Message Indication in Agile$^+$ Relief]

- The function only used in one file is found.

[Example 1]

```
file name: file.c
void   func1(void ) ;
void   func2(void);
void   func1(void)                          <-[The function “func2” is only used in file “file.c”.]
{
        func2();
             :
}
void   func2(void)
{
       :
}
```

Function func2 is only used in file.c.

M2.2.4

[Points of Message Indication in Agile$^+$ Relief]

- Integer constant expression is used as the expression of the case label in this switch statement.

[Example1]

```
int n;
switch( n )                          <-[Integer constant expression is used as the expression of the
                                     case label in this switch statement.]
{
   case -1 :   ~   break;
   case   0 :   ~   break;
   case   1 :   ~   break;
   case   2 :   ~   break;
   default   :
}
```

M3.1.1

[Points of Message Indication in Agile$^+$ Relief]

Multiple break statements that jump out of a loop exist.

[Example 1]

```
while( 1 ) {              <-[Two or more break statements appear in a loop.]
    if( data == 1 ) {
        :
        break;
    }
    else if( data == 2 ) {
      :
    }
    else {
        break;
    }
}
```

M3.1.2(1)

[Points of Message Indication in Agile$^+$ Relief]:

A goto statement is used.

[Example1]

```
LOOP:
func(&data);
if( data == 0 ) {
        goto    LOOP;                  <-[A goto statement is used.]
}
```

M3.1.2(2)

The same with the message indication of M3.1.2(1), please refer to M3.1.2(1).

M3.1.3　(IPA/SEC-C V1 only)

[Points of Message Indication in Agile^{+} Relief]:

A continue statement is used.

[Example1]

```
Filename: file.c
      int   func( char   buf[ ],   unsigned   int   n )
      {
         int   i ;
         int   not_space = 0;
 8:     for( i = 0; i < n && buf[i] != ' \0' ; i++ ) {
             if( isspace( buf[i] ) ) {
                continue;                <-[The keyword "continue" is used (location of relevant loop
                                                statement: line 8 of "file.c").]
         }
         not_space++;
      }
   return not_space;
 }
```

M3.1.4(1)

[Points of Message Indication in Agile$^+$ Relief]:

The case label has no corresponding break statement.

[Example1]

```
switch ( x ) {
10:     case 1:
11:             no++;                  <-[A break statement for "case 1:" in line 10 may have been
                                             accidentally omitted.]
12:     case 2:
13:             no++;                  <-[A break statement for "case 2:" in line 12 may have been
                                             accidentally omitted.]
default: no++;
}
```

M3.1.4(2)

The same with the message indication of M3.1.4(1), please refer to M3.1.4(1).

M3.1.5(1)

[Points of Message Indication in Agile^{+} Relief]:

More than one exits in the function.

[Example1]

```
void    func( void )      <-[The function "func" has 2 or more exits.]
{
         :
     return;
         :
     return;
}
```

M3.1.5(2)

The same with the message indication of M3.1.5(1), please refer to M3.1.5(1).

M3.2.1(1)

[Points of Message Indication in Agile$^+$ Relief]

A comma expression is used.

[Example 1]

```
x = 1, y = 2 ;                 <-["x=1,y=2" uses a comma outside an initialization expression or
                                   update expression in a for statement.]
```

[Example 2]

```
struct  tag { struct  tag  *next;  int  data; };
struct  tag  * top;
struct  tag  *p;
int  code;
  :
for( code = 1, p = top ;      <-["code = 1, p = top" uses a comma in the initialization expression or
                                   update expression of a for statement.]
    p != 0;
    p = p->next) {
```

M3.2.1(2)

[Points of Message Indication in Agile^{+} Relief]

- In for statement, comma expression is used outside initialization and update expression.

[Example 1]

```
x = 1, y = 2 ;                    <-[In for statement, comma expression is used in
                                  “x=1,y=2”outside initialization and update expression.]
```

M3.2.2

[Points of Message Indication in Agile$^+$ Relief]

- The increment (++) and decrement (--) expressions are used in operation.

[Example 1]

```
a = b++ ;                    <-[The result of “b++” is referenced in other operations.]
i = j + j++ ;                <-[The result of “j++” is referenced in other operations.]
x++, y = 1 ;                 <-[The result of “x++” is referenced in other operations.]
```

M3.3.1

[Points of Message Indication in Agile^{+} Relief]

There is an expression unrelated to the loop control in the for-statement.

[Example 1]

```
for( i = 0 , flag = 0 ; i < n; i++, counter++ ) {
```

<-[An expression "flag = 0" irrelevant to loop control exists in a for statement.]

<-[An expression "counter++" irrelevant to loop control exists in a for statement.]

M3.3.2

[Points of Message Indication in Agile$^+$ Relief]

Update a loop counter of the for-statement within the loop body.

[Example 1]

```
Filename: file.c
  5: for( i = 0; i < n; i++) {
         :
         i ++ ;                          <-[The loop counter "i" of a for statement is updated in the loop
                                                 body (for statement position: line 5 of "file.c" ).]
```

M3.3.3(1)

[Points of Message Indication in Agile$^+$ Relief]:

There is assignment expression in the conditional expression.
The assignment operation and comparison operation coexist.

[Example1]

```
if( x = func( ) )          <-[The assignment operator "=" is used in the conditional expression "if(x = func( ))".]
if( ( y = z ) != 0 )       <-[The assignment operator "=" is used in the conditional expression "( y = z ) != 0".]
```

M3.3.3(2)

The same with the message indication of M3.3.3(1), please refer to M3.3.3(1).

M3.4.1

[Points of Message Indication in Agile$^+$ Relief]

The declaration contains more than 2 levels of indirect pointers.

[Example 1]

```
int    ***x;     <-[The pointer "x" exceeds 2 levels.]
```

M4.1.1

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

M4.2.1

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

M4.3.1

[Points of Message Indication in Agile^{+} Relief]

- The name violating the naming rule (*) is found.

About the description and conditions for message indication of naming rule, please refer to "Check for Naming Rule" of "Command Manual".

[Example 1]

```
#define   _MACRO   1                <-[Macro “_MACRO” violates the naming rule
                                    (UNDERLINE=FALSE).]

bool         g_Flag ;               <-[Global variable “g_Flag3” violates the naming rule
                                    (bool=b).]
```

M4.3.2

[Points of Message Indication in Agile^{+} Relief]

The violation against this rule is not checked in Agile^{+} Relief.

M4.4.1

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

M4.4.2

[Points of Message Indication in Agile^{+} Relief]

The violation against this rule is not checked in Agile^{+} Relief.

M4.4.3

[Points of Message Indication in Agile^{+} Relief]

- External function and variable are declared outside header file.
- The function declaration for function call is not found.
- No function declaration is found before function definition.

[Example 1]

```
int func( int n );     <-[The external function “func” is declared outside header file.]
extern   int   x;     <-[The external variable “x” is declared outside header file.]
```

[Example 2]

```
x = func( 10 );      <-[The function declaration for function call “func” is not found.]
```

[Example 3]

```
void   func( int   x ) <-[No function declaration is found before function definition “func”.]
{
   :
}
```

M4.4.4

[Points of Message Indication in Agile^{+} Relief]

- There is the definition of the variables of the same name and with external union.

[Example 1]

```
file name:file1.c
10: short *data;

 file name:file2.c
    short data;                <-[The variable definitions of “data” and that in line 10 of
                               “file1.c” are of pointer type and non-pointer type respectively.
                               (short int, short int *)]
```

[Example 2]

```
file name:file.c
10: int   v = 1;
           :
    float   v = 0.0;        <-[The variable definition of “x” and that in line 10 of “file.c” are of the
                            different types.   (float, int)]
```

M4.4.5

[Points of Message Indication in Agile$^+$ Relief]

Variable and function definitions exist in the header file.

[Example 1]

Filename: head.h

```
int     x ;                   <-[Space is allocated in the header file (variable "x").]
int   func( void )         <-[Space is allocated in the header file (function "func"). ]
 {
   return x;
 }
```

Filename: file.c

```
#include "head.h"
```

M4.4.6

[Points of Message Indication in Agile^{+} Relief]

Multiple inclusions of the same file exist.

[Example 1]

```
Filename: file.c
  #include   "head.h"
12: #include   "head.h"            <-[The file "file.h" cannot be included redundantly (location of
                                         redundancy: line 11 in "file.c").]
Filename: head.h
     extern   int   X;
```

M4.5.1(1)

[Points of Message Indication in Agile$^+$ Relief]

- The parameter declaration with parameter name and not are found in function declaration.

[Example 1]

```
int   func( int   a, int );  <-[The parameter declaration with parameter name and not are found in
                                          function “func”.]
    :
int   func( int   a, int   b )
{
    :
}
```

M4.5.1(2)

[Points of Message Indication in Agile$^+$ Relief]

- There is no parameter name in the parameter declaration of function declaration.
- The parameter names are different in function declaration and function definition.

[Example 1]

```
int   func( int   a, int );  <-[The parameter declaration with parameter name or not are found in
                                                function "func".]
   :
int   func( int   a, int   b )
{
  :
}
```

[Example 2]

```
void   func( int, int );                    <-[All parameters in parameter declaration of function "func"
                                            are without name.]
```

[Example 3]

```
 file name:file.c
1: int func( int data,                        <-[Parameter name "data" is different from the function
                                               definition "func" in line 3 of "file.c".]
2:              int size );                   <-[Parameter name "size" is different from the function
                                              definition "func" in line 3 of "file.c".]
3: int func( int size, int data )
     {
```

M4.5.2

[Points of Message Indication in Agile$^+$ Relief]

- Tag of struct, union and enum type and variable are declared at the same time.

[Example 1]

```
struct   STR {                              <- [Variable “Str” and struct tag are declared at the same
                                            time.]
   int id ;
}   str ;
```

M4.5.3(1)

[Points of Message Indication in Agile^{+} Relief]

- Comma is found at the end of initialization expression list.
- Comma is found at the end of enum type member list.

[Example 1]

```
struct   tag   data = { 'a' , 'b', } ;         <-[Because comma is found at the end of initialization
                                               expression list, the input may be incomplete.]
```

[Example 2]

```
enum   ETAG { a=0 , b=0, } ;                 <-[Because comma is found at the end of enum type member
                                             list, the input may be incomplete.]
```

M4.5.3(2)

[Points of Message Indication in Agile^{+} Relief]

The same with the message indication of M4.5.3(1), please refer to M4.5.3(1).

M4.6.1(1)

[Points of Message Indication in Agile$^+$ Relief]

- Macro “NULL” is used.

  This message will be output when NULL is registered under the [CHECK_IDENTIFIER] label of the identifier file. For the registered in the identifier file, please refer to –F option of "Command Manual".

[Example 1]

```
     /* This message will be output when NULL is registered under the [CHECK_IDENTIFIER] label of
                                                                              the identifier file. */
  p = NULL ;                          <-[Macro “NULL” is used, please recheck.]
```

M4.6.1(2)　(IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

M4.6.1(2)　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile^{+} Relief]

- '¥0' is used for pointer type.
- (void*)0 is used for char type.
- 0 is used for char type.

[Example 1]

```
int func(char *str) {
  int count = 0;
  if (str == NULL) {
    return -1;
  }
  while (str != '\0') {    <-['\0' is used for pointer type.]
    str++;
    count++;
      :
  }
  return count;
}
```

[Example 2]

```
#define NULL ((void*)0)
int func(char *str) {
  int count = 0;
  if (str == NULL) {
    return -1;
  }
  while (*str != NULL) {    <-[(void*)0 is used for char type.]
    str++;
    count++;
      :
  }
  return count;
}
```

[Example 3]

```
int func(char *str) {
  int count = 0;
  if (str == NULL) {
    return -1;
  }
  while (*str != 0) {     <-[0 is used for char type.]
    str++;
    count++;
      :
  }
  return count;
}
```

M4.7.1

[Points of Message Indication in Agile^{+} Relief]

- All replacement strings in function are not braced.
- Parameters in the replacement string of macro are not braced.

[Example 1]

```
#define    AddTen(a)     (a) +10          <-[All replacement strings in macro function "AddTen" are not
                                            braced with ( ) or { }.]
x = AddTen( 20 ) * 2 ;
```

[Example 2]

```
#define    F(a, b)     (a * b) <-[Parameter "a" and "b" in the replacement string of macro "F" are not
                                            braced with ( ).]
x = F( 1 + 5, 10);
```

M4.7.2

[Points of Message Indication in Agile^{+} Relief]

A mismatch exists in the beginning and end of a conditional judgment.

[Example 1]

```
#endif            <-[There are no preprocess directives corresponding to "#endif".]
```

[Example 2]

```
#ifdef   ABC     <-[There is no #endif corresponding to #if, #ifdef or #ifndef.]
```

M4.7.3

[Points of Message Indication in Agile$^+$ Relief]

- Only macro name is specified to the right of #if or #elif.
- The identifier is found after the macro expansion to the right of #if or #elif.

[Example 1]

```
#if   ABC                              <-[Defined may be missed in "ABC".]
```

[Example 2]

```
#elif   DEBUG                          <-[Macro "DEBUG"used as bool is found in #if and #elif statement.]
```

M4.7.4　(IPA/SEC-C V1 only)

[Points of Message Indication in Agile^{+} Relief]

Grammar errors exist in the #if or #elif statement.

A "defined" is generated by macro expansion from the #if or #elif statements.

[Example 1]

```
#if 1 1                    <-[The end of the condition expression in an #if or #elif statement
                              contains a syntax error and cannot be executed.]
```

[Example 2]

```
#define DEF defined
#if DEF MAXNAM             <-[An #if or #elif statement generates the string "defined" in the
                              process of macro expansion.]
```

M4.7.5

[Points of Message Indication in Agile^{+} Relief]:

The block contains a #define or a #undef statement.

[Example1]

```
void   func( int   n )
{
   #define   MAX   10             <-[Macro "MAX" is operated by #defined in a block.]
   int   i;
   for( i = 0; i < MAX;   i++)
:
}
```

M4.7.6

[Points of Message Indication in Agile^{+} Relief]:

#undef statement is used.

[Example1]

```
#undef   AAA                     <-[The macro "AAA" is undefined with #undef.]
```

M4.7.7　(IPA/SEC-C V2 only)

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

M4.7.7　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]:

An integer description in a # if or # elif statement is neither 0 nor 1.

[Example1]

```
#if   100       <-[The expression of #if or #elif is not evaluated to 0 or 1.]
   :
#endif
#if   VER < 100           /* Conditional expression is not pointed out. */
   :
#endif
#if   1                  /* 1 is not pointed out. */
   :
#endif
```

M5.1.1 (IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

M5.1.1　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]:

Comment out source code for debugging.

[Example 1]

```
/* if ( val > 0 ) { */       <- [The source code may be commented out.]
/*    fprintf(stderr, "val = %d\n", val); */
/* } */
if ( val > 16 ) {
        :
```

M5.1.2(1)

[Points of Message Indication in Agile^{+} Relief]

A # or ## operator is used in the macro.

[Example 1]

```
#define   AAA(x,y)   x###y      <-[Operator # or ## is used in a macro.]
#define   A(x)       #x          <-[Operator # or ## is used in a macro.]
#define   B(x, y)    x##y      <-[Operator # or ## is used in a macro.]
#define   C         x##y          <-[Operator # or ## is used in a macro.]
```

M5.1.2(2)

[Points of Message Indication in Agile$^+$ Relief]:

Both #operator and ##operator exist in the macro function.

[Example1]

```
#define   AAA(x, y)    x###y      <-[The operators # and ## are used together in the definition of the
                                         macro function "AAA".]
```

M5.1.3

[Points of Message Indication in Agile$^+$ Relief]

The same parameter has appeared more than once in the replacement string of the macro function.

[Example 1]

```
#define   ISNUM( a )   ( ('0' <= (a)) && ((a) <= '9') )    <-[In a #define statement, parameter "a" is
                                                                 used two or more times in the
                                                                 replacement string of the
                                                                 macro function "ISNUM".]
if( ISNUM(*p++) )
```

M5.2.1(1)

[Points of Message Indication in Agile$^+$ Relief]

The following dynamic heap allocation functions are used:

- malloc
- calloc
- realloc
- free

[Example 1]

```
int    *mp;
mp = (int *)malloc( sizeof(int) * 10 );       <-[function "malloc" is used.]
```

M5.2.1(2)　(IPA/SEC-C V1, V2 only)

The same with the message indication of M5.2.1(1), please refer to M5.2.1(1).

M5.2.1(2)　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile^{+} Relief]

The following dynamic heap allocation functions are used:

malloc

calloc

realloc

free

It refers to resources that have already been released.

It returns without releasing the resource acquired in the function.

The resource obtained in the function is used as the return value of the function.

The resource obtained in the function is assigned to an external variable and returned without being released.

Free the space reserved in the variable declaration using a function that frees the dynamic area.

Free a pointer to a static area using a function to free the dynamic area.

[Example 1]

```
int   *mp;
mp = (int *)malloc( sizeof(int) * 10 );       <-[function "malloc" is used.]
```

[Example 2]

```
Line
 11:   if ( mode == 0. ) {
 12:                   free( p );
 13:   }
 14:   *p = 0 ;                          <-[The variable "p" the resources of which are reclaimed in
                                          line 12, might be referenced.]
```

[Example 3]

```
Line
10:   char     *p ;
11:   if ( ( p = malloc( sizeof(char) * 100 ) ) == NULL ) {
12:      return 1 ;
13:   }
  :
21:   if ( err ) {
22:      free( p ) ;
23:      return 1;
24:   }
25:   return 0 ;                        <-[Resources allocated with the function "malloc" in line 11
                                        may not have been deallocated. ]
```

[Example 4]

```
Line
10:   void * func( int i , void* pBuff ) {
11:         void* p ;
12:         if (   i < 10 )
13:              p = malloc( i ) ;
14:         else
15:              p = pBuff ;
16:         return p ;                  <-[Resources point "p" allocated with the function "malloc" in
                                        line 13 of function "func" is returned.]
17:    }
```

[Example 5]

```
Line
1:    extern char *p ;
           :
10:    int func( ) {
11:       if ( ( p = malloc( sizeof(char) * 100 ) ) == NULL ) {
12:            return 1 ;
13:         }
                 :
21:       if ( err ) {
22:            free( p ) ;
23:            return 1;
24:       }
25:         return 0 ;             <-[Global point "p" to resources which was allocated with the function
                                   "malloc" in line 11 of function "func" may not have been deallocated.]
26:   }
```

[Example 6]

```
Line
10:   int     x, *p, data[10] ;
11:   p = data ;
12:   free ( &x ) ;                    <-[The function "free" may not be used to deallocate the static
                                       resource "&x".]
13:   free ( p ) ;                     <-[The function "free" may not be used to deallocate the static
                                       resource "data" assigned in line 11.]
```

[Example 7]

```
Line
10:   time_t   timer ;
11:   struct   tm   *localtimer ;
12:   time( &timer ) ;
13:   localtimer = localtime( & timer ) ;
14:   free( localtimer ) ;             <-[The return value "localtimer" of function "localtime" at line
                                       13 point to a static field, it is an error to free it by function
                                       "free".]
```

3.3 Portability

P1.1.1(1)

[Points of Message Indication in Agile$^+$ Relief]

There are descriptions not compliant with C90 (ISO/IEC 9899:1990).

[Example 1]

```
static   enum   A { /*declaration*/ };     <-[The storage class specifier "static" may not be used here.]
extern   struct   B { /*declaration*/ };   <-[The storage class specifier "extern" may not be used here.]
```

[Example 2]

```
int   a[-1];                          <-[The number of array elements "-1" is not greater than 0.]
 struct   Dynamic_Array {
   unsigned   int   size;
   int   data[0];                     <-[The number of array elements "0" is not greater than 0.]
};
```

P1.1.1(2)

[Points of Message Indication in Agile^{+} Relief]

The same with the message indication of P1.1.1(1), please refer to P1.1.1 (1).

P1.1.2

[Points of Message Indication in Agile$^+$ Relief]

- The operation is different according to the type of compiler.

[Example 1]

```
'ab';           <-[The value of char const 'ab' composed by multiple characters is different according
                                        to the type of compiler.]
```

[Example 2]

```
int     x,y;
x = y >> 2;     <-[Right shift is performed when a negative is found in signed "y",then the result is
                different to the type of compiler.]
```

P1.1.3

[Points of Message Indication in Agile^{+} Relief]

The violation against this rule is not checked in Agile^{+} Relief.

P1.2.1

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

P1.2.2

[Points of Message Indication in Agile^{+} Relief]

There exists an undefined escape sequence starting with \.

[Example 1]

```
c = '\8';                    <-[Escape sequence "'\8'" might be processed differently depending on the
                                   compiler.]
```

P1.3.1

[Points of Message Indication in Agile^{+} Relief]

The char type is not used for a character value.

Implicit conversion between the type char and signed char/unsigned char.

[Example 1]

```
char   x;
  :
x = 1;                              <-[The char "x" not specified as signed or unsigned is not treated
                                          as a character value.]
```

[Example 2]

```
signed   char   func( int   x ){
    char   c;
      :
    return   c;                     <-[If char is unsigned, the return value "c" and the function "func"
                                          are of different types: signed and unsigned (type of return
                                          value: char , function type: signed char).]

}
```

P1.3.2

[Points of Message Indication in Agile$^+$ Relief]

- Enum const is defined with a value exceeding the scope of int.

[Example 1]

```
enum TAG {   E1 = 0x123456789,        <-[Enum const “E1” exceeds the scope of int.]
               E2 = 1,
               E3 = 2 };
```

P1.3.3(1)

[Points of Message Indication in Agile$^+$ Relief]

- Struct and union with bit field member have been declared.

[Example 1]

```
struct TAG {              <-[The member of bit field in struct “TAG” has been defined. ]
   unsigned int m1:1 ;
   unsigned int m2:1 ;
};
```

P1.3.3(2)

The same message indication with that of P1.3.3(1), please refer to P1.3.3(1).

P1.3.3(3)

The same message indication with that of P1.3.3(1), please refer to P1.3.3(1).

P1.4.1

[Points of Message Indication in Agile$^+$ Relief]

The included file name in the #include statement has not been embraced by <>, or "".

[Example 1]

```
#include   head.h            <-[The name of the header file is not enclosed with angle brackets < >
                                       or quotation marks " ".]
```

P1.4.2

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

P1.4.3

[Points of Message Indication in Agile$^+$ Relief]:

There are ' (single quote), \(currency character), "(double-quote), /*(slash and asterisk), multiple-byte characters(Chinese characters, etc)in the #include-specified file name.

[Example1]

```
#include   "abc/*XY*/d.h"      <-[The file name "abc/*XY*/d.h" specified by #include contains one or
                                   more characters not defined in ANSI.]
```

P1.5.1

[Points of Message Indication in Agile$^+$ Relief]

- Absolute path is found in the file name specified with #include.

[Example 1]

```
#include   "/home/APL/inc/file.h"    <-[The file name "/home/APL/inc/file.h"specified with #include
                                       is an absolute path name.]
```

P1.5.2 (IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

- Set a constant as an argument to a function that specifies size.
- Setting variable values, such as variables, as arguments to functions that specify sizes..

[Example 1]

```
a. int *mp;
   mp = (int *)malloc(80);   /* 80 = intsize * 20 */    <-[sizeof/strlen could be used to obtain the value
                                                          for the 1st argument "80" of the function
                                                          "malloc".]
b. #define MAX 256
    char *op;
    char msg[MAX];
   memcpy(msg, op, MAX);                                <-[sizeof/strlen could be used to obtain the value
                                                          for the 3rd argument "256" of the function
                                                          "memcpy".]
c. char *cp;
   if(strncmp(cp, "OK", 2) == 0)                        <-[sizeof/strlen could be used to obtain the value
                                                          for the 3rd argument "256" of the function
                                                          "strncmp".]
```

[Example 2]

```
unsigned int Msize;
int *Mp;
void set(void)
{
    Msize = 4;   /* 4 is the size of int */
}
void alc(void)
{
    Mp = (int *)malloc(Msize);                         <-[Check whether sizeof/strlen is used to obtain
                                                         the value for the 1st argument "Msize" of the
                                                         function "malloc".]
}
```

P2.1.1

[Points of Message Indication in Agile$^+$ Relief]

Assembly language is used.

[Example 1]

```
asm(mov    r4,    r0);              <-[Assembly language "asm" is used.]
```

P2.1.2

[Points of Message Indication in Agile^{+} Relief]

The violation against this rule is not checked in Agile^{+} Relief.

P2.1.3(1)　(IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile^{+} Relief]

Basic types int/short/char/long/double/float are used directly.

[Example 1]

```
#include "file1.h"
void　func ( void )
{
int　short_length ;<-[In this file, a basic type (int/short/char/long/double/float) is used directly.]
  :
}
```

P2.1.3(1)　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

Basic types int/short/char/long/double/float are used directly.
The type name is defined by a macro.

[Example 1]

```
#include "file1.h"
void　func ( void )
{
int　short_length ;<-[In this file, a basic type (int/short/char/long/double/float) is used directly.]
  :
}
```

[Example 2]

```
#define　PSI32　int*　<-[typedef can be used instead of macro "PSI32".]
PSI32　a, b;
```

P2.1.3(2)

[Points of Message Indication in Agile$^+$ Relief]

The same with the message indication of P2.1.3(1), please refer to P2.1.3(1).

3.4 Efficiency

E1.1.1　(IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

E1.1.1　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile^{+} Relief]

- Macro functions are defined.

[Example 1]

```
#define max( a , b )   ( ( a > b ) ? a b )   <-[The macro function "max" can be replaced by the inline
                                               function or the template function.]
```

E1.1.2

[Points of Message Indication in Agile^{+} Relief]

- The value of the expression is unchanged in loop.

[Example 1]

```
10: for( i = 0 ; i < x * 10 – y ; i++ ) {      <-[“x * 10 - y” is applied in loop process of line 10 repeatedly. It
                                               may be placed outside loop.]
                    :
        }
x and y are not updated in for statement.
```

E1.1.3

[Points of Message Indication in Agile$^+$ Relief]

- Structure or union of non-pointer type is declared for the parameters of function definition.

[Example 1]

```
int func( struct tag x )          <-[The parameter “x” of pointer type or reference type can
                                  speed up the processing and bring reduction in code size.]
{
    /* function body */
}
```

E1.1.4

[Points of Message Indication in Agile$^+$ Relief]

- In an if statement with else if, the constant and == operator is applied by each conditional expression through the same variable.

[Example 1]

```
if( x == 1 )   {   ~   }                          <-[Not only confined to if statement, it is the same with swtich
                                                     statement.]
else if( x == 2 )   {   ~   }
else if( x == 3 )   {   ~   }
```

3.5 Coding Miss

MISS 1.1

[Points of Message Indication in Agile$^+$ Relief]

There is an unexecuted statement.

[Example 1]

```
if( x == 1)
{ process 1 }
else if( x != 1)
{ process 2 }
else                    <-[This else statement will never be executed.]
{ process 3 }

if( y == 1)
{ process 1 }
else if( y != 1)
{ process 2 }
else if(y < 0)        <-[This else statement will never be executed.]
{ process 3 }
```

[Example 2]

```
for ( ; ; ) {
  /* no break*/
}
x = y + z ;           <-[This will not be executed.]
```

MISS1.2

[Points of Message Indication in Agile$^+$ Relief]:

Within the sizeof, there is an update expression, a function call or a volatile variable.

[Example1]

```
x = sizeof( y++ ) ;                <-["y++" is in sizeof, so updating is not performed.]
```

MISS1.3

[Points of Message Indication in Agile$^+$ Relief]

- The internal variable and static external variable are not referenced after value settings.

[Example 1]

```
int   x ;                              <-[The value of variable “x” is not referenced for once.]
x = 1 ;
After then, x is not referenced.
```

MISS1.4　(Points of Message Indication in IPA/SEC-C V1)

[Points of Message Indication in Agile$^+$ Relief]

- For non volatile parameter and automatic variable, the post increment and decrement is found in return statement.

[Example 1]

```
int  func(void)
{
    int  x;
       :
    return  x++;                <-[The increment and decrement of return value “x++” is
                                meaningless.]
}
```

MISS1.4 (Points of Message Indication in IPA/SEC-C V2, V3)

[Points of Message Indication in Agile^{+} Relief]

- The value of parameter is not referenced for once.

[Example 1]

```
void   func( int arg )
{
    arg = 1 ;
    return;                                  <-[The value of parameter "arg" is not referenced.]
}
```

MISS1.5　(IPA/SEC-C V1 only)

[Points of Message Indication in Agile^{+} Relief]

- The value of parameter is not referenced for once.

[Example 1]

```
void　func( int arg )
{
    arg = 1 ;
    return;                                  <-[The value of parameter “arg” is not referenced.]
}
```

MISS2.1

[Points of Message Indication in Agile$^+$ Relief]

- The mathmethic description of the operator (< <= > >= == != ! && ||) and the operator (< <= > >= == !=) appears repeatedly.

[Example 1]

if(**0 < x < 10**)	**<-**[An error might occur when comparing the result of “0 < x” with “10”.]

MISS2.2

[Points of Message Indication in Agile$^+$ Relief]

- Some conditional expressions may be true or false according to being signed or not, or of different sizes.

[Example 1]

```
char   c;
   :
if ( c == -1 ) {                    <-[If char is unsigned, "-1" is not equal to "c".]
```

[Example 2]

```
short   s;
   :
if ( s == 65536 )                   <-[An error might occur when comparing "s" of short type with
                                    the constant "65536" exceeding the bitwidth of short int type.]
```

MISS2.3

[Points of Message Indication in Agile$^+$ Relief]

- A comparison with character const is made.

[Example 1]

```
char    *p;
    :
if ( p == “OK”    )    <-[In “p == “OK”“, an error might occur when comparing with string literal.]
```

MISS2.4

[Points of Message Indication in Agile^{+} Relief]

The function may have no return statement for the return value, or the return statement of the same type as the function does not exist.

[Example 1]

```
int   func( void )
{
}                    <-[The function "func" does not have a return statement.]
```

[Example 2]

```
int   func( int a )
{
   if ( a == 0)
   {   return 0; }
}                                  <-[In the function "func", there are routes that do not have return
                                       statements.]
```

[Example 3]

```
int   func( void )
{
        :
   return   0;
        :
   return;                         <-[The return value of a return statement might have been omitted
                                       accidentally.]
}
```

MISS2.5　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

The function may have no return statement for the return value, or the return statement of the same type as the function does not exist.

[Example 1]

```
struct block *bp, *wbp;
 int count;
 wbp = (char*)bp + (count * 256);          <-[The pointer addition/subtraction expression "bp +
                                           (count * 256)" might contain a mistake.]
```

MISS3.1

[Points of Message Indication in Agile^{+} Relief]

- Attempt to access outside the scope of array.

[Example 1]

```
   file name:file.c
1:    int   data[10] ;
            :
5:    data[10] =0 ;                  <-[The suffix ”10” of array “data” exceeds the socpe of array.
                                     (Array Declaration: Line 1 of “file.c”.]
```

[Example 2]

```
file name :file.c
1:    int   i , data[10] ;
            :
5:    for( i = 0 ; i <= 10 ; i++ ) {
6:         data[i] = i ;             <-[In for statement of line “5”, because “10” included in “i”
                                     exceeds the scope of “data[i]”. (Array Declaration: Line 1
                                     of”file.c”)]
```

MISS3.2

[Points of Message Indication in Agile^{+} Relief]

The address of an automatic variable has been set to a return value of the function.

The address of an automatic variable is assigned to a variable outside the effective scope.

[Example 1]

```
char   *func( void )
{
char   str[16];
       :
return   str;                <-[The address "str" is the address of an automatic variable and
                                   should not be used as the return value of a function.]
       :
}
```

[Example 2]

```
int   *p ;
{
      int    x;
      p = &x ;                <-[The address of the automatic variable x cannot be used outside
                                  the scope.]
}
*p = 0 ;
```

MISS3.3

[Points of Message Indication in Agile$^+$ Relief]

- The released resource is used by functions, such as free and fclose.

[Example 1]

```
11:   if( mode == 0 ) {;
12:         free(p) ;
13:   }
14:   *p = 0 ;                          <-[Variable “p” of released resource in line 12 maybe referenced.]
```

MISS3.4　(IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

MISS3.4　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

- A string literal is referenced without const qualification.

[Example 1]

```
void func1( char *s );
void func0( void ) {
      char *buf1 = "Agile+ Relief";    <-[Not specified by const but referred string literal.]
            :
      func1("Agile+ Relief");                    <-[Not specified by const but referred string literal.]
}
```

MISS3.5　(IPA/SEC-C V1, V2 only)

[Points of Message Indication in Agile^{+} Relief]

- For the memory copying functions, like strcpy, the target memory size is less than that of source memory.
- For the functions accessing the specified size, the memory of specified bytes cannot be guaranteed.

[Example 1]

```
char    a[10], b[20] ;
          :
strcpy( a , b ) ;                  <-[The memory space of “a” in “strcpy (a,b)” may be exceeded.
                                  (Target:10, Source:20)]
```

[Example 2]

```
int   x ;
struct { int m1; int m2 ; } sa ;
          :
memset( &x , 0 , sizeof(sa) ) ;  <-[“memset (&x, 0, sizeof(sa)” may access the contents outside the
                                  memory space of “x”. (Memory:4, size:8)]
```

MISS3.5　(IPA/SEC-C V3 only)

[Points of Message Indication in Agile$^+$ Relief]

- For the memory copying functions, like strcpy, the target memory size is less than that of source memory.
- For the functions accessing the specified size, the memory of specified bytes cannot be guaranteed.
- Copying information obtained using a function that retrieves an environment variable of an unpredictable length into an array with a function that copies the data.

[Example 1]

```
char     a[10], b[20] ;
          :
strcpy( a , b ) ;                <-[The memory space of “a” in “strcpy (a,b)” may be exceeded.
                                (Target:10, Source:20)]
```

[Example 2]

```
int   x ;
struct { int m1; int m2 ; } sa ;
          :
memset( &x , 0 , sizeof(sa) ) ;  <-[“memset (&x, 0, sizeof(sa)” may access the contents outside the
                                memory space of “x”. (Memory:4, size:8)]
```

[Example 3]

```
#define ENV_PATHMAX    128
void func( void ) {
      char path[ENV_PATHMAX];
strcpy( path, getenv("PATH") );        <-[Copying environment variable into "path" in may access
                                       addresses beyond array boundary.]
```

MISS4.1

[Points of Message Indication in Agile$^+$ Relief]

- Logic sum and logic product may be mistakenly used in some conditional expressions.

[Example 1]

```
if( x == 1   &&   x == 2 )          <-["x == 1 && x == 2" cannot be true.]
```

[Example 2]

```
if( x == 1   ||    x == 1 )         <-[Meaningless conditional expression is found in "x == 1 || x == 1".]
if( x < 10   &&   x < 20 )          <-[Meaningless conditional expression is found in "x < 10 && x < 20".]
```

[Example 3]

```
if( x >= 1   &&   x <= 1 )          <-[The conditional expressions that can be incorporated is found in "x >= 1 && x <= 1".]
if( x >= 2   ||   x == 2 )          <-[The conditional expressions that can be incorporated is found in "x >= 2 || x <= 2"."]
```

[Example 4]

```
if( x != 1   ||   x != 2 )          <-["x != 1 || x != 2" cannot be false.]
```

MISS4.2

[Points of Message Indication in Agile$^+$ Relief]

- Logical sum is used in the conditions of for statement, and the judgment of loop counter is contained in the conditional expression on one side.

[Example 1]

```
#define   MAX   10
int   h[MAX] ;
      :
for( i = 0 ; i < MAX || p != 0 ; i++ )        <-[ An error might occur in loop condition “I < MAX || p !=
                                              0”.]
{
      h[i] = 0 ;
       :
}
```

MISS4.3

[Points of Message Indication in Agile$^+$ Relief]

- Bit operation is found in the conditional expression.

[Example 1]

```
if ( a < 0 & b < 0 )                    <-[In “a < 0 & b <0”, & may be mistaken as &&, and | as ||.]
```

MISS5.1

[Points of Message Indication in Agile$^+$ Relief]

- Assignment is found in the conditional expressions.

[Example 1]

if(**x = y**) <-[Assignment “x = y” is used as condition.]

[Example 2]

if(**x = 0**) <-[Conditional expression “x = 0” is meaningless. = may be mistaken as ==.]

MISS6.1

[Points of Message Indication in Agile+ Relief]

- The macro with the same name is redefined with other values.

[Example 1]

```
  file name :file.c
10: #define   ABC   1
    #define   ABC   10            <-[Macro "ABC" is redefined with a different value in line 10 of
                                  "file.c"]
```

[Example 2]

```
#define   ABC   10                <-[Macro "ABC" is redefined with a value different from –D
                                  option]
```

As an option for Agile+ Relief analysis, specify –DABC=xxx (xxx is a value other than 10)

MISS6.2

[Points of Message Indication in Agile$^+$ Relief]

The violation against this rule is not checked in Agile$^+$ Relief.

Appendix A　Rule List

A.1 IPA/SEC-C V1 rule

Rule	Status
R1.1.1	○
R1.1.2	○
R1.2.1	○
R1.2.2	○
R1.3.1(1)	○
R1.3.1(2)	○
R1.3.2	○
R1.3.3	○
R2.1.1	○
R2.1.2	○
R2.1.3	○
R2.2.1	○
R2.3.1	○
R2.3.2	○
R2.3.3	○
R2.4.1	○
R2.4.2	○
R2.5.1	○
R2.5.2	○
R2.5.3	○
R2.5.4	○
R2.6.1	○
R2.6.2	○
R2.7.1(1)	○
R2.7.1(2)	○
R2.7.1(3)	○
R2.7.2	○
R2.7.3	○
R2.8.1	○
R2.8.2(1)	○
R2.8.2(2)	○(RuleA)
R2.8.3	○
R3.1.1(1)	○
R3.1.1(2)	○
R3.1.2	○
R3.2.1	○
R3.2.2	○
R3.3.1	○
R3.3.2	○
R3.4.1	○
R3.5.1	○(RuleB)
R3.5.2	○(RuleB)
R3.5.3	○
R3.6.1	○
R3.6.2	○
M1.1.1	○
M1.1.2	×
M1.2.1(1)	○
M1.2.1(2)	○
M1.2.2	○
M1.2.3	○
M1.3.1	○
M1.3.2	○
M1.3.3	○
M1.4.1	○
M1.4.2	○ (RuleB)
M1.5.1	○
M1.5.2	○
M1.6.1	○
M1.6.2(1)	○
M1.6.2(2)	○
M1.7.1	○
M1.7.2	○
M1.7.3	×
M1.8.1	○
M1.8.2	○
M1.8.3	○
M1.8.4	○
M1.8.5	○
M1.9.1	Δ (Note1)
M1.9.2	Δ (RuleB) (Note2)
M1.10.1	×
M1.11.1	○
M1.11.2	○
M1.11.3	× (RuleB)
M1.12.1	○
M2.1.1	○
M2.1.2	○
M2.2.1	○
M2.2.2	○
M2.2.3	○
M2.2.4	○
M3.1.1	○
M3.1.2(1)	○
M3.1.2(2)	○
M3.1.3	○
M3.1.4(1)	○
M3.1.4(2)	○(RuleB)
M3.1.5(1)	○
M3.1.5(2)	Δ (Note3)
M3.2.1(1)	○
M3.2.1(2)	○
M3.2.2	○
M3.3.1	○
M3.3.2	○
M3.3.3(1)	○
M3.3.3(2)	○
M3.4.1	○
M4.1.1	× (RuleB)
M4.2.1	× (RuleB)
M4.3.1	○ (RuleB)
M4.3.2	× (RuleB)
M4.4.1	× (RuleB)
M4.4.2	× (RuleB)
M4.4.3	○
M4.4.4	○
M4.4.5	○
M4.4.6	○ (RuleB)
M4.5.1(1)	○
M4.5.1(2)	○
M4.5.2	○
M4.5.3(1)	○
M4.5.3(2)	○
M4.6.1(1)	○
M4.6.1(2)	×
M4.7.1	○
M4.7.2	○ (RuleB)
M4.7.3	○
M4.7.4	○
M4.7.5	○
M4.7.6	○
M5.1.1	× (RuleB)
M5.1.2(1)	○
M5.1.2(2)	○
M5.1.3	Δ (Note4)
M5.2.1(1)	○
M5.2.1(2)	○ (RuleB)
P1.1.1(1)	○
P1.1.1(2)	○ (RuleA)
P1.1.2	○ (RuleA)
P1.1.3	×(RuleA&B)
P1.2.1	× (RuleB)
P1.2.2	○
P1.3.1	○
P1.3.2	○
P1.3.3(1)	○
P1.3.3(2)	○
P1.3.3(3)	○ (RuleA)
P1.4.1	○
P1.4.2	× (RuleB)
P1.4.3	○
P1.5.1	○
P2.1.1	○ (RuleB)
P2.1.2	× (RuleB)
P2.1.3(1)	○
P2.1.3(2)	○ (RuleB)
E1.1.1	×
E1.1.2	○
E1.1.3	○
E1.1.4	Δ (RuleB) (Note5)
MISS1.1	○
MISS1.2	○
MISS1.3	○
MISS1.4	○
MISS1.5	○
MISS2.1	○
MISS2.2	○
MISS2.3	○
MISS2.4	○
MISS3.1	○
MISS3.2	○
MISS3.3	○
MISS3.4	×
MISS3.5	○
MISS4.1	○
MISS4.2	○
MISS4.3	○
MISS5.1	○
MISS6.1	○
MISS6.2	×

(RuleA): It denotes the rules for file creation.
(RuleB): It denotes the rules formed for the need of each project.
○: Indicated ×: Not Indicated
Δ: Though it is hard to indicate the static analysis value error, the best effort will be made for possible indication.
Note 1: It will be indicated when ')' of if, for and while statement is followed with a colon.
Note 2: It will be indicated when the variable used in the condition of loop statement is not updated in loop.
Note 3: It will be indicated when more than 2 outlets found in function.
Note 4: It will be indicated for functions possibly leading trouble.
Note 5: It will be indicated for the if statement which seemly can be converted into a switch statement.

A.2 IPA/SEC-C V2 rule

Rule	Status
R1.1.1	○
R1.1.2	○
R1.2.1	○
R1.2.2	○
R1.3.1(1)	○
R1.3.1(2)	○
R1.3.2	○
R1.3.3	○
R1.3.4	○
R2.1.1	○
R2.1.2	○
R2.1.3	○
R2.2.1	○
R2.3.1	○
R2.3.2	○
R2.3.3	○
R2.4.1	○
R2.4.2	○
R2.5.1	○
R2.5.2	○
R2.5.3	○
R2.5.4	○
R2.6.1(1)	○
R2.6.1(2)	○
R2.6.1(3)	○
R2.6.2	○
R2.7.1(1)	○
R2.7.1(2)	○
R2.7.1(3)	○
R2.7.2	○
R2.7.3	○
R2.8.1	○
R2.8.2(1)	○
R2.8.2(2)	○(RuleA)
R2.8.3	○
R3.1.1(1)	○
R3.1.1(2)	○
R3.1.2	○
R3.1.3	×
R3.1.4	×
R3.2.1	○
R3.2.2	○
R3.3.1	○
R3.3.2	○

Rule	Status
R3.4.1	○
R3.5.1	○(RuleB)
R3.5.2	○(RuleB)
R3.5.3	○
R3.6.1	○
R3.6.2	○
R3.6.3	○
M1.1.1	○
M1.1.2	×
M1.2.1(1)	○
M1.2.1(2)	○
M1.2.2	○
M1.2.3	○
M1.3.1	○
M1.3.2	○
M1.3.3	○
M1.4.1	○
M1.4.2	○ (RuleB)
M1.5.1	○
M1.5.2	○
M1.6.1	○
M1.6.2(1)	○
M1.6.2(2)	○
M1.7.1	○
M1.7.2	○
M1.7.3	×
M1.8.1	○
M1.8.2	○
M1.8.3	○
M1.8.4	○
M1.8.5	○
M1.9.1	∆ (Note1)
M1.9.2	∆ (RuleB) (Note2)
M1.10.1	×
M1.11.1	○
M1.11.2	○
M1.11.3	× (RuleB)
M1.12.1	○
M2.1.1	○
M2.1.2	○
M2.2.1	○
M2.2.2	○
M2.2.3	○

Rule	Status
M2.2.4	○
M3.1.1	○
M3.1.2(1)	○
M3.1.2(2)	○
M3.1.4(1)	○
M3.1.4(2)	○(RuleB)
M3.1.5(1)	○
M3.1.5(2)	∆ (Note3)
M3.2.1(1)	○
M3.2.1(2)	○
M3.2.2	○
M3.3.1	○
M3.3.2	○
M3.3.3(1)	○
M3.3.3(2)	○
M3.4.1	○
M4.1.1	× (RuleB)
M4.2.1	× (RuleB)
M4.3.1	○ (RuleB)
M4.3.2	× (RuleB)
M4.4.1	× (RuleB)
M4.4.2	× (RuleB)
M4.4.3	○
M4.4.4	○
M4.4.5	○
M4.4.6	○ (RuleB)
M4.5.1(1)	○
M4.5.1(2)	○
M4.5.2	○
M4.5.3(1)	○
M4.5.3(2)	○
M4.6.1(1)	○
M4.6.1(2)	×
M4.7.1	○
M4.7.2	○ (RuleB)
M4.7.3	○
M4.7.5	○
M4.7.6	○
M4.7.7	×
M5.1.1	× (RuleB)
M5.1.2(1)	○
M5.1.2(2)	○
M5.1.3	∆ (Note4)
M5.2.1(1)	○

Rule	Status
M5.2.1(2)	○ (RuleB)
P1.1.1(1)	○
P1.1.1(2)	○ (RuleA)
P1.1.2	○ (RuleA)
P1.1.3	×(RuleA&B)
P1.2.1	× (RuleB)
P1.2.2	○
P1.3.1	○
P1.3.2	○
P1.3.3(1)	○
P1.3.3(2)	○
P1.3.3(3)	○ (RuleA)
P1.4.1	○
P1.4.2	× (RuleB)
P1.4.3	○
P1.5.1	○
P2.1.1	○ (RuleB)
P2.1.2	× (RuleB)
P2.1.3(1)	○
P2.1.3(2)	○ (RuleB)
E1.1.1	×
E1.1.2	○
E1.1.3	○
E1.1.4	∆ (RuleB) (Note5)
MISS1.1	○
MISS1.2	○
MISS1.3	○
MISS1.4	○
MISS2.1	○
MISS2.2	○
MISS2.3	○
MISS2.4	○
MISS3.1	○
MISS3.2	○
MISS3.3	○
MISS3.4	×
MISS3.5	○
MISS4.1	○
MISS4.2	○
MISS4.3	○
MISS5.1	○
MISS6.1	○
MISS6.2	×

(RuleA): It denotes the rules for file creation.
(RuleB): It denotes the rules formed for the need of each project.
○: Indicated ×: Not Indicated
∆: Though it is hard to indicate the static analysis value error, the best effort will be made for possible indication.
Note 1: It will be indicated when ')' of if, for and while statement is followed with a colon.
Note 2: It will be indicated when the variable used in the condition of loop statement is not updated in loop.
Note 3: It will be indicated when more than 2 outlets found in function.
Note 4: It will be indicated for functions possibly leading trouble.
Note 5: It will be indicated for the if statement which seemly can be converted into a switch statement.

A.3 IPA/SEC-C V3 rule

Rule	Status	Rule	Status	Rule	Status	Rule	Status
R1.1.1	○	R3.4.1	○	M2.2.3	○	P1.1.1(1)	○
R1.1.2	○	R3.5.1	○(RuleB)	M2.2.4	○	P1.1.1(2)	○ (RuleA)
R1.2.1	○	R3.5.2	○(RuleB)	M3.1.1	○	P1.1.2	○ (RuleA)
R1.2.2	○	R3.5.3	○	M3.1.2(1)	○	P1.1.3	×(RuleA&B)
R1.3.1(1)	○	R3.6.1	○	M3.1.2(2)	○	P1.2.1	× (RuleB)
R1.3.1(2)	○	R3.6.2	○	M3.1.4(1)	○	P1.2.2	○
R1.3.2	○	R3.6.3	○	M3.1.4(2)	○(RuleB)	P1.3.1	○
R1.3.3	○	R3.11.1	○	M3.1.5(1)	○	P1.3.2	○
R1.3.4	○	R3.11.2	○	M3.1.5(2)	Δ (Note3)	P1.3.3(1)	○
R2.1.1	○	M1.1.1	○	M3.2.1(1)	○	P1.3.3(2)	○
R2.1.2	○	M1.1.2(1)	○	M3.2.1(2)	○	P1.3.3(3)	○ (RuleA)
R2.1.3	○	M1.1.2(2)	○	M3.2.2	○	P1.4.1	○
R2.2.1	○	M1.2.1(1)	○	M3.3.1	○	P1.4.2	× (RuleB)
R2.3.1	○	M1.2.1(2)	○	M3.3.2	○	P1.4.3	○
R2.3.2	○	M1.2.2	○	M3.3.3(1)	○	P1.5.1	○
R2.3.3	○	M1.2.3	○	M3.3.3(2)	○	P1.5.2	○
R2.4.1	○	M1.3.1	○	M3.4.1	○	P2.1.1	○ (RuleB)
R2.4.2	○	M1.3.2	○	M4.1.1	× (RuleB)	P2.1.2	× (RuleB)
R2.5.1	○	M1.3.3	○	M4.2.1	× (RuleB)	P2.1.3(1)	○
R2.5.2	○	M1.4.1	○	M4.3.1	○ (RuleB)	P2.1.3(2)	○ (RuleB)
R2.5.3	○	M1.4.2	○ (RuleB)	M4.3.2	× (RuleB)	E1.1.1	○
R2.5.4	○	M1.5.1	○	M4.4.1	× (RuleB)	E1.1.2	○
R2.6.1(1)	○	M1.5.2	○	M4.4.2	× (RuleB)	E1.1.3	○
R2.6.1(2)	○	M1.6.1	○	M4.4.3	○	E1.1.4	Δ (RuleB) (Note5)
R2.6.1(3)	○	M1.6.2(1)	○	M4.4.4	○		
R2.6.2	○	M1.6.2(2)	○	M4.4.5	○	MISS1.1	○
R2.7.1(1)	○	M1.7.1	○	M4.4.6	○ (RuleB)	MISS1.2	○
R2.7.1(2)	○	M1.7.2	○	M4.5.1(1)	○	MISS1.3	○
R2.7.1(3)	○	M1.7.3	○	M4.5.1(2)	○	MISS1.4	○
R2.7.2	○	M1.8.1	○	M4.5.2	○	MISS2.1	○
R2.7.3	○	M1.8.2	○	M4.5.3(1)	○	MISS2.2	○
R2.8.1	○	M1.8.3	○	M4.5.3(2)	○	MISS2.3	○
R2.8.2(1)	○	M1.8.4	○	M4.6.1(1)	○	MISS2.4	○
R2.8.2(2)	○(RuleA)	M1.8.5	○	M4.6.1(2)	○	MISS2.5	○
R2.8.3	○	M1.9.1	Δ (Note1)	M4.7.1	○	MISS3.1	○
R3.1.1(1)	○	M1.9.2	Δ (RuleB) (Note2)	M4.7.2	○ (RuleB)	MISS3.2	○
R3.1.1(2)	○			M4.7.3	○	MISS3.3	○
R3.1.2	○	M1.10.1	×	M4.7.5	○	MISS3.4	○
R3.1.3	○	M1.11.1	○	M4.7.6	○	MISS3.5	○
R3.1.4	○	M1.11.2	○	M4.7.7	○	MISS4.1	○
R3.1.5(1)	○	M1.11.3	× (RuleB)	M5.1.1	○ (RuleB)	MISS4.2	○
R3.1.5(2)	○	M1.12.1	○	M5.1.2(1)	○	MISS4.3	○
R3.2.1	○	M2.1.1	○	M5.1.2(2)	○	MISS5.1	○
R3.2.2	○	M2.1.2	○	M5.1.3	Δ (Note4)	MISS6.1	○
R3.3.1	○	M2.2.1	○	M5.2.1(1)	○	MISS6.2	×
R3.3.2	○	M2.2.2	○	M5.2.1(2)	○ (RuleB)		

(RuleA): It denotes the rules for file creation.
(RuleB): It denotes the rules formed for the need of each project.
○: Indicated ×: Not Indicated
Δ: Though it is hard to indicate the static analysis value error, the best effort will be made for possible indication.
Note 1: It will be indicated when ')' of if, for and while statement is followed with a colon.
Note 2: It will be indicated when the variable used in the condition of loop statement is not updated in loop.
Note 3: It will be indicated when more than 2 outlets found in function.
Note 4: It will be indicated for functions possibly leading trouble.
Note 5: It will be indicated for the if statement which seemly can be converted into a switch statement.

A.4 IPA/SEC-C++ V1 rule

Rule	Status
R1.1.1	○
R1.2.2	○
R1.3.1(1)	○
R1.3.1(2)	○
R1.3.2	○
R1.3.3	○
R1.4.1	○
R1.4.2	○
R1.4.3	×
R1.4.4	○
R1.4.5	○
R1.4.6	×
R1.5.1	○
R2.1.1	○
R2.1.2	○
R2.1.3	○
R2.3.1	○
R2.3.2	○
R2.3.3	○
R2.4.1	○
R2.4.2	○
R2.5.1	○
R2.5.2	○
R2.5.3	○
R2.5.4	○
R2.6.1(1)	○
R2.6.1(2)	○
R2.6.1(3)	○
R2.6.2	○
R2.7.1(1)	○
R2.7.1(2)	○
R2.7.1(3)	○
R2.7.2	○
R2.7.3	○
R2.7.4(1)	○
R2.7.4(2)	○
R2.8.2(1)	○
R2.8.2(2)	○(RuleA)
R2.8.3	○
R3.1.1(1)	○
R3.1.1(2)	○
R3.1.2	○
R3.2.1	○
R3.2.2	○
R3.3.1	○
R3.3.2	○
R3.4.1	○
R3.5.1	○(RuleB)
R3.5.2	○(RuleB)

Rule	Status
R3.5.3	×
R3.6.1	○
R3.6.2	○
R3.7.1	○
R3.7.2	○
R3.7.3	○
R3.7.4	○
R3.7.5	○
R3.7.6	×
R3.8.1(1)	○
R3.8.1(2)	○(RuleB)
R3.8.2	○
R3.8.3	○
R3.8.4	○
R3.8.5	○
R3.8.6	○
R3.8.7	○
R3.8.8	○
R3.8.9	○
R3.9.1	×
M1.1.1	○
M1.1.2	×
M1.2.1(1)	○
M1.2.1(2)	○
M1.2.2	○
M1.2.3	○
M1.2.4	×(RuleB)
M1.2.5	○
M1.2.6	×
M1.2.7	×
M1.3.1	○
M1.3.2	○
M1.4.1	○
M1.4.2	○(RuleB)
M1.5.1	○
M1.5.2	○
M1.6.1	○
M1.6.2(1)	○
M1.6.2(2)	○
M1.7.1	○
M1.7.2	○
M1.7.3	×
M1.8.1	○
M1.8.2	○
M1.8.3	○
M1.8.4	○
M1.8.5	○
M1.8.6	○
M1.8.7	○

Rule	Status
M1.8.8	○
M1.9.1	∆ (Note1)
M1.9.2	∆ (RuleB) (Note2)
M1.10.1	×
M1.11.1	○
M1.11.2	×
M1.11.3	×(RuleB)
M1.12.1	○
M2.1.1	○
M2.1.2	○
M2.2.1	×
M2.2.2	×
M2.2.3(1)	×
M2.2.3(2)	×
M2.2.4	○
M2.2.5	○
M3.1.1	○
M3.1.2(1)	○
M3.1.2(2)	○
M3.1.3	○
M3.1.4(1)	○
M3.1.4(2)	○(RuleB)
M3.1.5(1)	○
M3.1.5(2)	∆(Note3)
M3.2.1(1)	○
M3.2.1(2)	○
M3.2.2	○
M3.3.1	○
M3.3.2	○
M3.3.3(1)	○
M3.3.3(2)	○
M3.4.1	○
M3.5.1	×
M4.1.1	×(RuleB)
M4.1.2	○
M4.2.1	×(RuleB)
M4.3.1	○(RuleB)
M4.3.2	×(RuleB)
M4.4.1	×(RuleB)
M4.4.2	×(RuleB)
M4.4.3	○
M4.4.6	○(RuleB)
M4.4.7	○
M4.4.8	×(RuleB)
M4.5.1(1)	○
M4.5.1(2)	○
M4.5.2	×
M4.5.3(1)	○

Rule	Status
M4.5.3(2)	○
M4.6.1(1)	○
M4.6.1(2)	×
M4.7.1	○
M4.7.2	○(RuleB)
M4.7.3	○
M4.7.4	○
M4.7.5	○
M4.7.6	○
M4.8.1	○
M4.8.2	○
M4.8.3	○
M5.1.1	×(RuleB)
M5.1.2(1)	○
M5.1.2(2)	○
M5.1.3	∆(Note4)
M5.2.1(1)	×
M5.2.1(2)	×(RuleB)
P1.1.1(1)	×
P1.1.1(2)	×(RuleA)
P1.1.2	○(RuleA)
P1.1.3	×(RuleA) (RuleB)
P1.2.1	×(RuleB)
P1.2.2	○
P1.3.1	×
P1.3.2	○
P1.3.3(1)	○
P1.3.3(2)	○
P1.3.3(3)	○(RuleA)
P1.4.1	○
P1.4.2	×(RuleB)
P1.4.3	○
P1.5.1	○
P2.1.1	○(RuleB)
P2.1.2	×(RuleB)
P2.1.3(1)	×
P2.1.3(2)	×(RuleB)
E1.1.1	×
E1.1.2	○
E1.1.3	○
E1.1.4	∆ (RuleB) (Note5)
E1.1.5	○
E1.1.6	○
E1.1.7	×
E1.1.8	○
E1.1.9	×
E1.1.10	×

(RuleA): It denotes the rules for file creation.
(RuleB): It denotes the rules formed for the need of each project.
○: Indicated ×: Not Indicated
∆: Though it is hard to indicate the static analysis value error, the best effort will be made for possible indication.
Note 1: It will be indicated when ')' of if, for and while statement is followed with a colon.
Note 2: It will be indicated when the variable used in the condition of loop statement is not updated in loop.
Note 3: It will be indicated when more than 2 outlets found in function.
Note 4: It will be indicated for functions possibly leading trouble.
Note 5: It will be indicated for the if statement which seemly can be converted into a switch statement.

A.5 IPA/SEC-C++ V2 rule

Rule	Status
R1.1.1	○
R1.2.2	○
R1.3.1(1)	○
R1.3.1(2)	○
R1.3.2	○
R1.3.3	○
R1.4.1	○
R1.4.2	○
R1.4.3	×
R1.4.4	○
R1.4.5	○
R1.4.6	×
R1.5.1	○
R2.1.1	○
R2.1.2	○
R2.1.3	○
R2.3.1	○
R2.3.2	○
R2.3.3	○
R2.4.1	○
R2.4.2	○
R2.5.1	○
R2.5.2	○
R2.5.3	○
R2.5.4	○
R2.6.1(1)	○
R2.6.1(2)	○
R2.6.1(3)	○
R2.6.2	○
R2.7.1(1)	○
R2.7.1(2)	○
R2.7.1(3)	○
R2.7.2	○
R2.7.3	○
R2.7.4(1)	○
R2.7.4(2)	○
R2.8.2(1)	○
R2.8.2(2)	○(RuleA)
R2.8.3	○
R3.1.1(1)	○
R3.1.1(2)	○
R3.1.2	○
R3.2.1	○
R3.2.2	○
R3.3.1	○
R3.3.2	○
R3.4.1	○
R3.5.1	○(RuleB)
R3.5.2	○(RuleB)
R3.5.3	×
R3.6.1	○
R3.6.2	○
R3.6.3	○
R3.7.1	○
R3.7.2	○
R3.7.3	○
R3.7.4	○
R3.7.5	○
R3.7.6	×
R3.7.7	×
R3.8.1(1)	○
R3.8.1(2)	○(RuleB)
R3.8.2	○
R3.8.3	○
R3.8.4	○
R3.8.5	○
R3.8.6	○
R3.8.7	○
R3.8.8	○
R3.8.9	○
R3.9.1	×
R3.10.1	×
R3.11.1	×
R3.11.2	○
M1.1.1	○
M1.1.2	×
M1.2.1(1)	○
M1.2.1(2)	○
M1.2.2	○
M1.2.3	○
M1.2.4	×(RuleB)
M1.2.5	○
M1.2.6	×
M1.2.7	×
M1.3.1	○
M1.3.2	○
M1.4.1	○
M1.4.2	○(RuleB)
M1.5.1	○
M1.5.2	○
M1.6.1	○
M1.6.2(1)	○
M1.6.2(2)	○
M1.7.1	○
M1.7.2	○
M1.7.3	×
M1.8.1	○
M1.8.2	○
M1.8.3	○
M1.8.4	○
M1.8.5	○
M1.8.6	○
M1.8.7	○
M1.8.8	○
M1.9.1	∆ (Note1)
M1.9.2	∆ (RuleB) (Note2)
M1.10.1	×
M1.11.1	○
M1.11.2	×
M1.11.3	×(RuleB)
M1.12.1	○
M2.1.1	○
M2.1.2	○
M2.2.1	×
M2.2.2	×
M2.2.3(1)	×
M2.2.3(2)	×
M2.2.4	○
M2.2.5	○
M2.3.1	○
M3.1.1	○
M3.1.2(1)	○
M3.1.2(2)	○
M3.1.4(1)	○
M3.1.4(2)	○(RuleB)
M3.1.5(1)	○
M3.1.5(2)	∆(Note3)
M3.2.1(1)	○
M3.2.1(2)	○
M3.2.2	○
M3.3.1	○
M3.3.2	○
M3.3.3(1)	○
M3.3.3(2)	○
M3.4.1	○
M3.5.1	×
M4.1.1	×(RuleB)
M4.1.2	○
M4.2.1	×(RuleB)
M4.3.1	○(RuleB)
M4.3.2	×(RuleB)
M4.4.1	×(RuleB)
M4.4.2	×(RuleB)
M4.4.3	○
M4.4.6	○(RuleB)
M4.4.7	○
M4.4.8	×(RuleB)
M4.5.1(1)	○
M4.5.1(2)	○
M4.5.2	×
M4.5.3(1)	○
M4.5.3(2)	○
M4.6.1(1)	×
M4.6.1(2)	○
M4.6.1(3)	×
M4.7.1	○
M4.7.2	○(RuleB)
M4.7.3	○
M4.7.5	○
M4.7.6	○
M4.7.7	×
M4.8.1	○
M4.8.2	○
M4.8.3	○
M5.1.1	×(RuleB)
M5.1.2(1)	○
M5.1.2(2)	○
M5.1.3	∆(Note4)
M5.2.1(1)	×
M5.2.1(2)	×(RuleB)
P1.1.1(1)	○
P1.1.1(2)	○(RuleA)
P1.1.2	○(RuleA)
P1.1.3	×(RuleA) (RuleB)
P1.2.1	×(RuleB)
P1.2.2	○
P1.3.1	×
P1.3.2	○
P1.3.3(1)	○
P1.3.3(2)	○
P1.3.3(3)	○(RuleA)
P1.4.1	○
P1.4.2	×(RuleB)
P1.4.3	○
P1.5.1	○
P1.6.1	○
P2.1.1	○(RuleB)
P2.1.2	×(RuleB)
P2.1.3(1)	○(RuleB)
P2.1.3(2)	○(RuleB)
E1.1.1	×
E1.1.2	○
E1.1.3	○
E1.1.4	∆ (RuleB) (Note5)
E1.1.5	○
E1.1.6	○
E1.1.7	×
E1.1.8	○
E1.1.9	×
E1.1.10	×

(RuleA): It denotes the rules for file creation.

(RuleB): It denotes the rules formed for the need of each project.

○: Indicated ×: Not Indicated

∆: Though it is hard to indicate the static analysis value error, the best effort will be made for possible indication.

Note 1: It will be indicated when ')' of if, for and while statement is followed with a colon.

Note 2: It will be indicated when the variable used in the condition of loop statement is not updated in loop.

Note 3: It will be indicated when more than 2 outlets found in function.

Note 4: It will be indicated for functions possibly leading trouble.

Note 5: It will be indicated for the if statement which seemly can be converted into a switch statement.