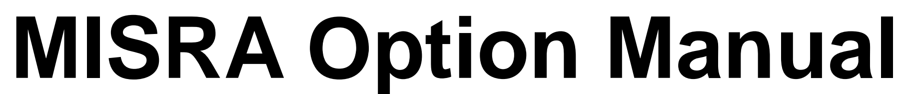
FUJITSU

FUJITSU Software
Agile+ Relief C/C++ V1.1.1



MISRA Option Manual

B1WD-3607-01ENZ0(00)
January 2023

Foreword

Agile+ Relief C/C++(hereafter referred to as Agile+ Relief) MISRA options are applied to analyze and output messages on source programs written in the C language/C++ language and header files for MISRA violations.

This manual illustrates MISRA violation conditions that apply to Agile+ Relief.

For more information regarding message information output for program defects in Agile+ Relief, please also refer to "Message Indications Manual".

For more information regarding error message output in Agile+ Relief, please refer to "Command Manual".

References:

[MISRA-C:2004 Guidelines for the use of the C language in critical systems]- The Motor Industry Software Reliability Association

[Use Guideline of C Language For Car (JASO/TP-01002)] - Corporations Motor Technology Institute.

[Guide of MISRA-C Embedded Program Design with High Reliability For Embedded Developer] written by - MISRA-C, Japan Standard Association.

[MISRA-C++:2008 Guidelines for the use of the C++ language in critical systems]- The Motor Industry Software Reliability Association.

[MISRA-C:2012 Guidelines for the use of the C language in critical systems]- The Motor Industry Software Reliability Association

[MISRA-C:2012 Amendment 1 Additional security guidelines for MISRA-C:2012]- The Motor Industry Software Reliability Association

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

All other product and service names mentioned in this document generally imply trademarks of their respective owners.

This written material contains specific technology that falls under the [Foreign Exchange and International Trade Law]. Therefore, the export of any part of this documentation without consent is illegal.

FUJITSU Limited

Note

- No part of this publication may be transcribed and printed without prior written permission.
- Information contained within this manual is subject to change without notice.

All rights reserved, Copyright(C) 2005-2023 FUJITSU LIMITED.

Contents

1. MISRA Options in GUI 1
1.1 Create Project 1
1.2 Check for MISRA Violation 10
1.3 Experienced with Agile+ Relief C/C++ 16
1.4 Setting up Rule Checking 18
2 MISRA Options in Command line 21
2.1 Functions of Command 21
2.2 Format of Command 21
2.3 Command Options 21
2.4 Return Value 22
2.5 Output Files 23
2.6 Error Messages 23
2.7 Example for Use 25
2.7.1 Analyze with pgr5 command 25
2.7.2 Message Checking in the pgrmisra Command 25
3 Corresponding Indication Messages for the Violation of MISRA Rules 27
3.1 MISRA-C V1 27
3.2 MISRA-C V2 157
3.3 MISRA-C V3 301
3.4 MISRA-C++ V1 464
Appendix A: A List of MISRA Rules 695
A.1 MISRA-C V1 Rule 695
A.2 MISRA-C V2 Rule 697
A.3 MISRA-C V3 Rule 699
A.4 MISRA-C++ V1 Rule 701

1. MISRA Options in GUI

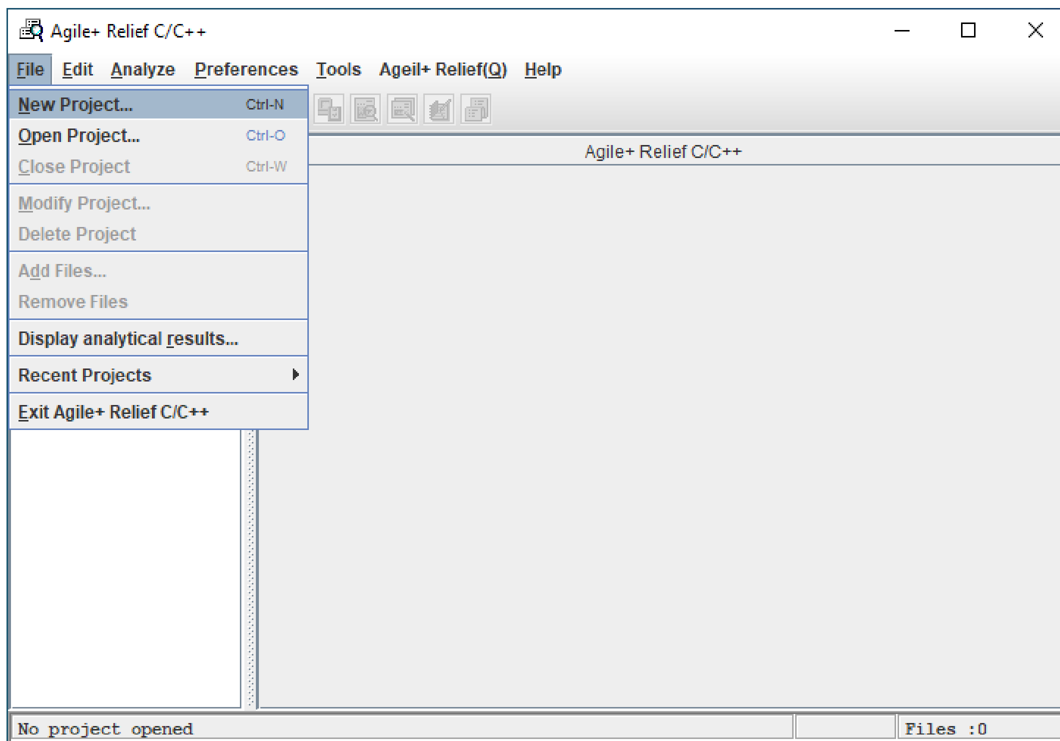
This chapter explains the settings of MISRA options in GUI.

1.1 Create Project

First, you are able to create projects for Agile$^+$ Relief.

When analyzing C/C++ source files in Agile$^+$ Relief, management must be performed on the source files in the unit of a project.

Select [New Project] from the [File] menu of the [Main Window].



The dialogue box "Project Wizard 1/3" will be 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:\

Browse...

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

< Back | Next > | Finish | Cancel | Help

Please enter "sample" in the Project Name.

Please enter single-byte alphanumeric characters. Multi byte characters in the Project Name is not allowed.

"Create this as a child project under the selected one" cannot be selected here.

Enter the directory to which you wish to save the projects in the Project Directory. Please enter "C:\demo" in the sample.

When a project name has been entered while a directory name is still required, the project name will automatically be appended to the end of the directory name.

When establishing the settings, the following dialogue box will be 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

Browse...

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

< Back | Next > | Finish | Cancel | Help

Select [Next].

"Project Wizard 2/3" dialogue box.

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.

< Back | Next > | Finish | Cancel | Help

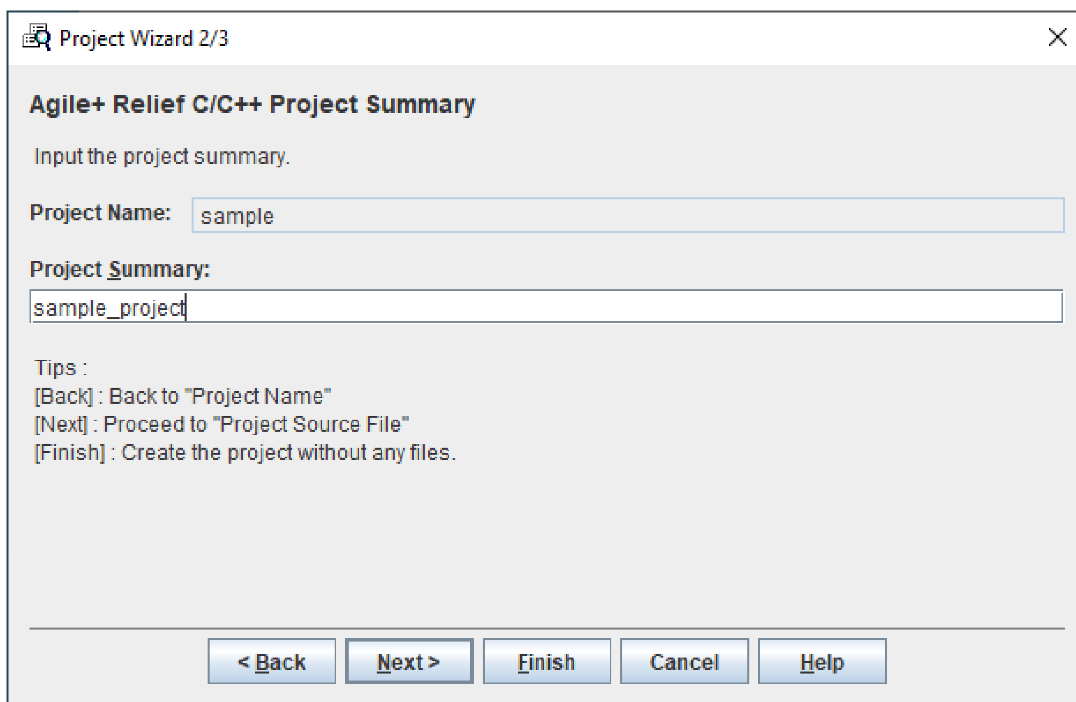
The dialogue box is for Project Summary settings.

In order to simplify illustration, the settings made here will be displayed in the Main Window to facilitate convenient reference for future projects.

Note: Symbols including multi byte characters or spaces, and other such symbols are allowed in the Project Summary. (Ellipses are available)

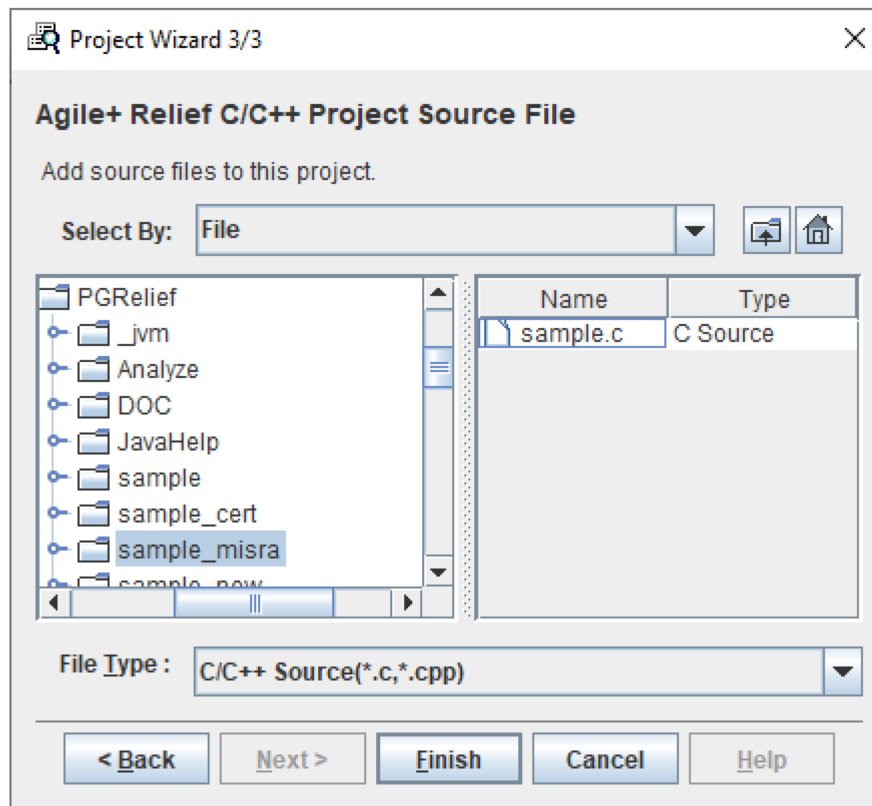
"Sample Project" is set for the illustration example.

When establishing the settings, the dialogue box will be displayed as follows:



Select [Next].

"Project Wizard 3/3" is displayed.



Add source files to this project in this dialogue box.

"File" is set in File Selection.

[C/C++ Source] is selected in File Category.

Index is displayed on the left side of the dialogue box.

Select "sample_misra" under the Agile⁺ Relief C/C++ installation directory .

◆Regarding classic installation, when Agile⁺ Relief C/C++ installation is complete, there will be installation samples in the sub-directory of "sample", "sample_new", "sample_misra" , "sample_sec" and “sample_cert" created under the Agile⁺ Relief C/C++ installation directory .

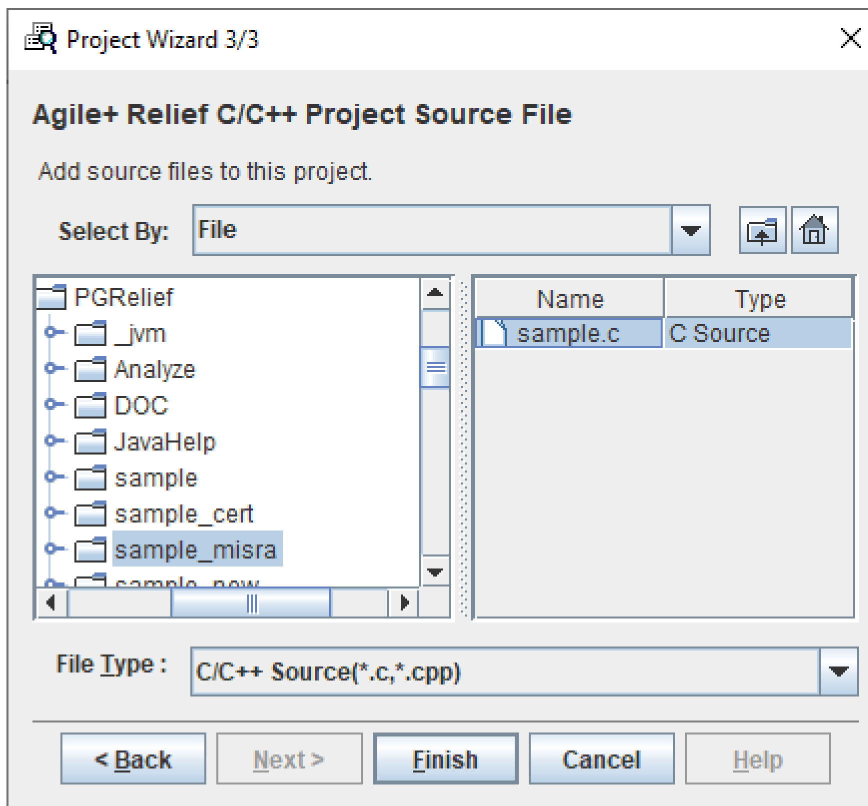
The source files used in the sample are displayed on the right side of the dialogue box.

Drag the source files displayed in the list, and select all sample files.

The settings of "Project Wizard 3/3" are as above.

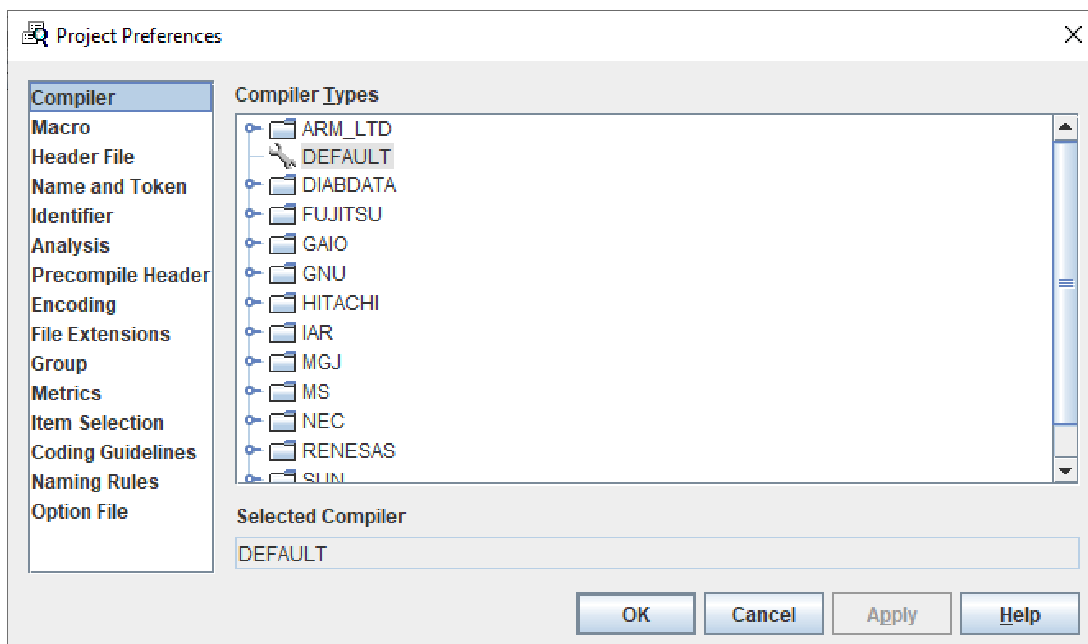
Select [Finish].

A Project has been successfully completed.



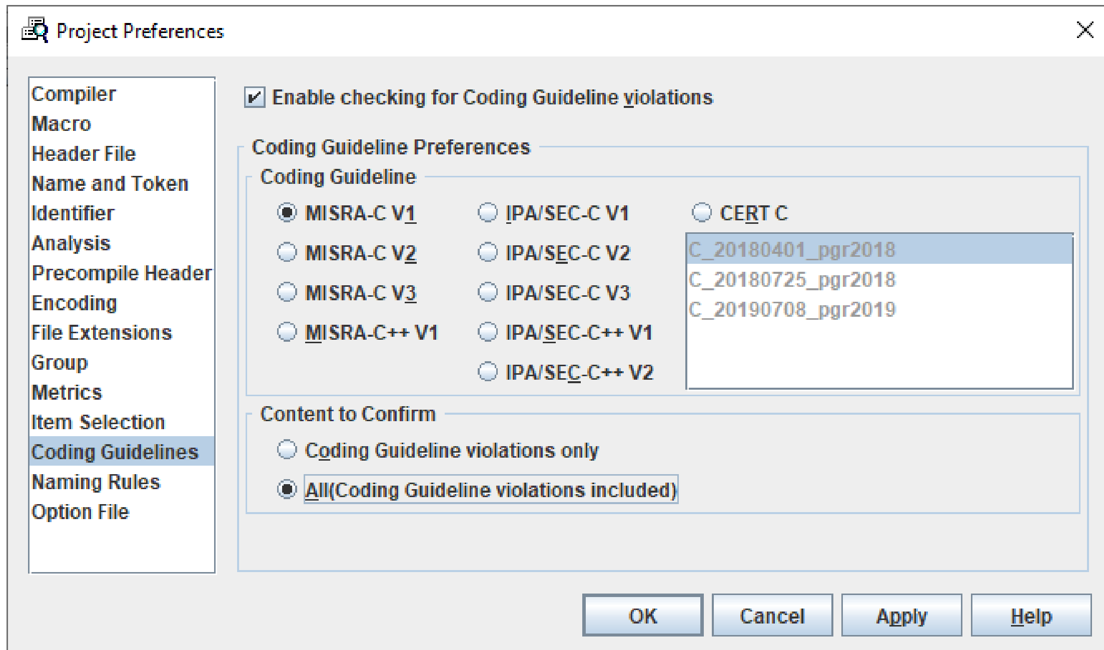
The settings for analysis are as follows (option settings):

The "Project Preference" dialogue box will be displayed:



Select the "Coding Guidelines" option and then do the check of [Enable checking for

Coding Guideline violations].



[Coding Guideline] For Rule Settings:

[MISRA-C V1] checks in accordance with MISRA-C:1998.

[MISRA-C V2] checks in accordance with MISRA-C:2004.

[MISRA-C V3] checks in accordance with MISRA-C:2012 and MISRA-C:2012 Amendment 1.

[MISRA-C++ V1] checks in accordance with MISRA-C++:2008.

[IPA/SEC-C V1] checks in accordance with IPA/SEC-C V1.

[IPA/SEC-C V2] checks in accordance with IPA/SEC-C V2.

[IPA/SEC-C V3] checks in accordance with IPA/SEC-C V3.

[IPA/SEC-C++ V1] checks in accordance with IPA/SEC-C++ V1.

[IPA/SEC-C++ V2] checks in accordance with IPA/SEC-C++ V2.

[CERT C] checks in accordance with 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 [MISRA-C V1] here.

[Content to Confirm] allows you to select and confirm the contents against Rules.

[Coding Guideline violations only] is applied to selected Rules Violation Checks only.

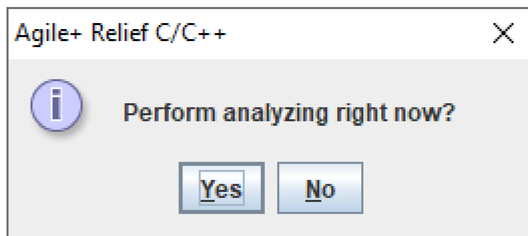
[ALL (Coding Guideline violations included)] is also applied to issues indicated in the viewpoints of Agile⁺ Relief besides selected Rules Violation Checks.

In order to increase program quality and precision, the confirmation of issues indicated both by Coding Guideline and the viewpoints of Agile⁺ Relief is recommended.

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

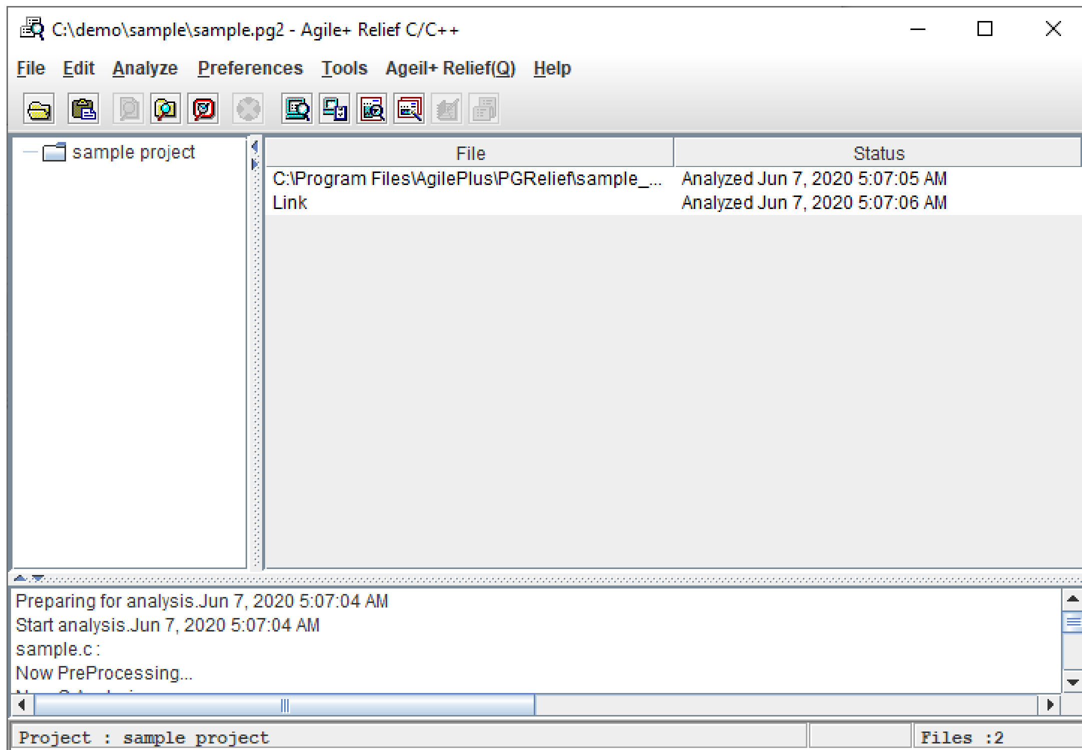
Select [OK].

When the "Perform analyzing right now" query is displayed, please select [Yes].



If you choose [No], please select [Analyze All] from the [Analyze] menu.

When the analysis is in progress, the current state of and results pertaining to the analysis will be displayed in the log viewer.

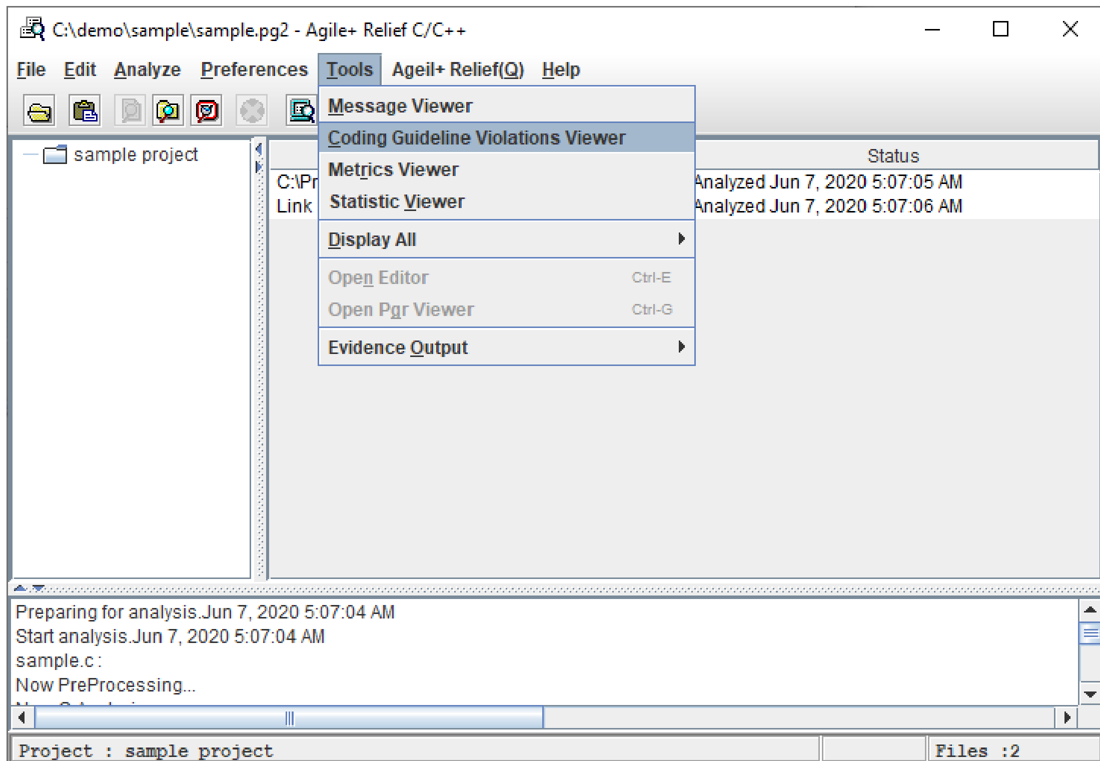


No error occurred in the sample analysis. If an error occurred in actual practice and the desired analysis cannot be performed, please review the Compiler Type, Include Directory or Macro, and other such settings.

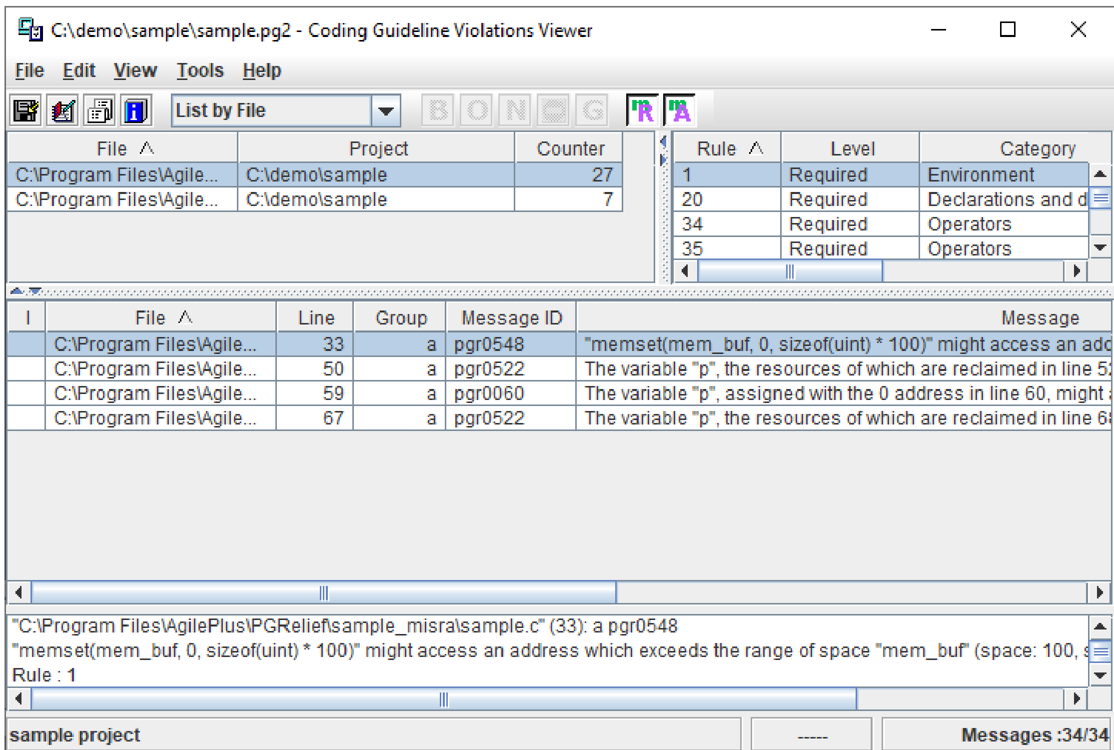
1.2 Check for MISRA Violation

Checking for MISRA Rule Violation.

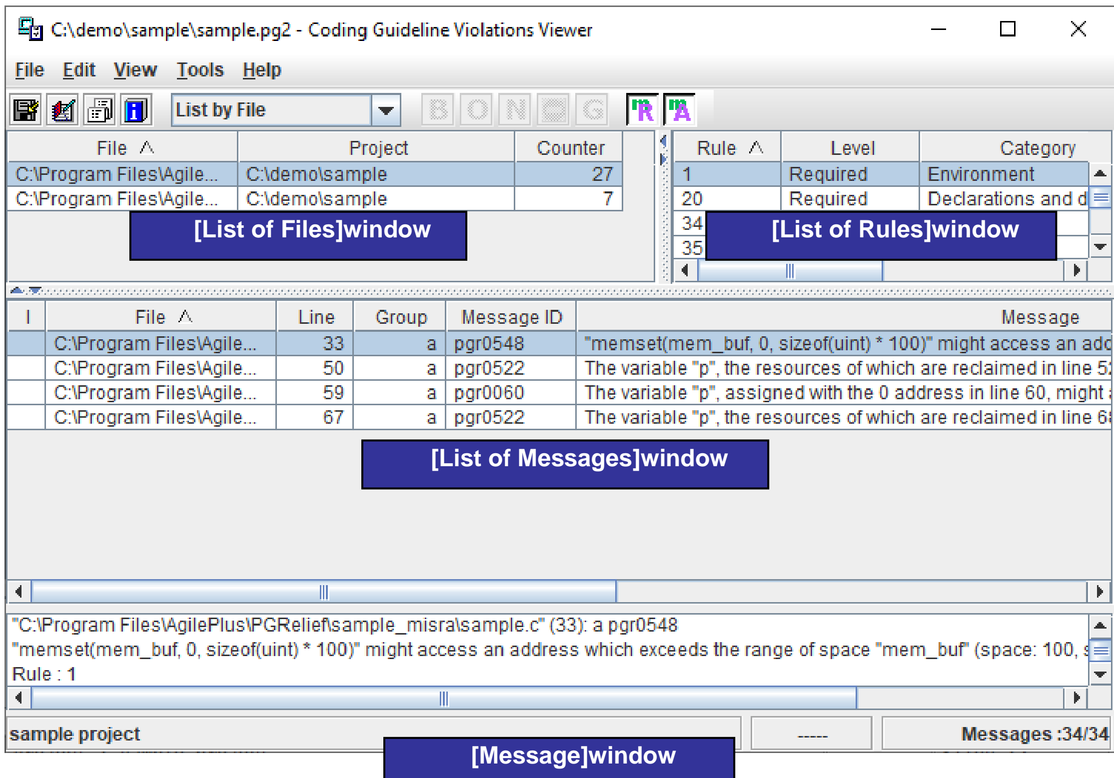
Select sample.c in the "Main Window", and then select [Coding Guideline Violations Viewer] from the [Tools] menu.



The "Coding Guideline Violations Viewer" window will be displayed:



Brief Illustration on how to use the "Coding Guideline Violations Viewer":



In the [List of Files] window, files that have violated MISRA are displayed.

In the [List of Rules] window, the formal number of the MISRA rule that has been violated is displayed.

In the [List of Messages] window, the contents and indicated row are displayed.

When "List by File" on the toolbar is changed to "List by Rule", the [List of Files] window will be switched to the [List of Rules] window.

When a rule number in [List of Rules] is selected, information corresponding to the message will be displayed in the [Message] window.

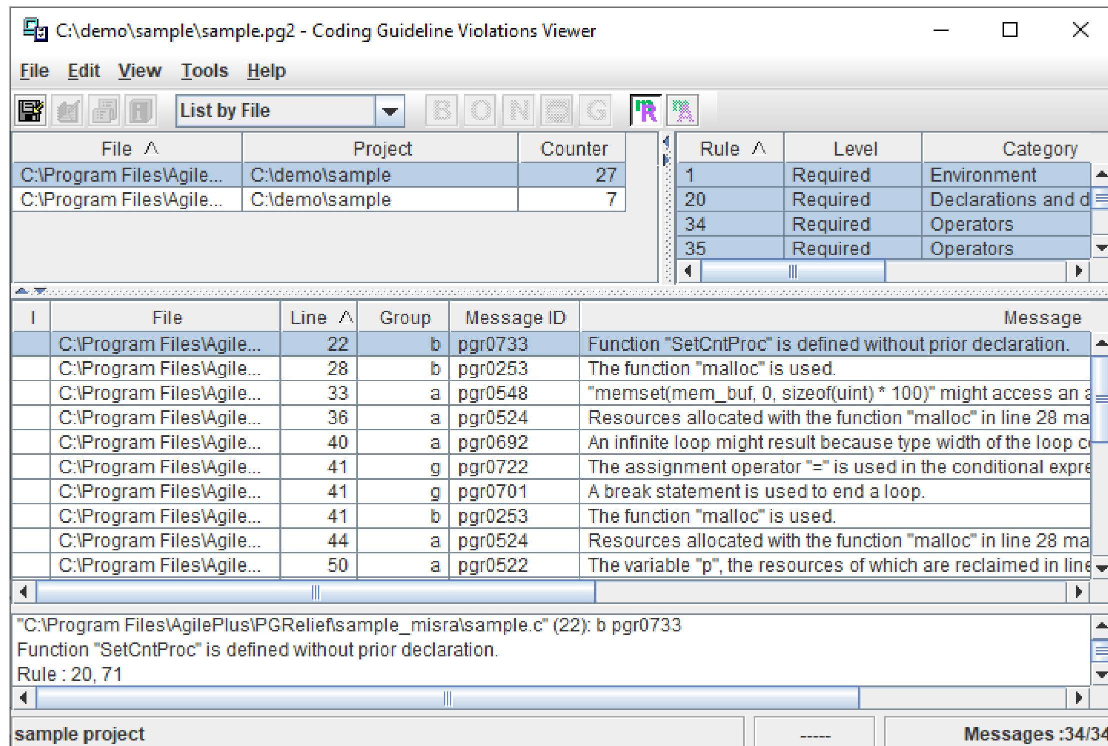
Please confirm the indicated locations in order and modify the source files.

Some MISRA rule numbers may sometimes refer to the same message.

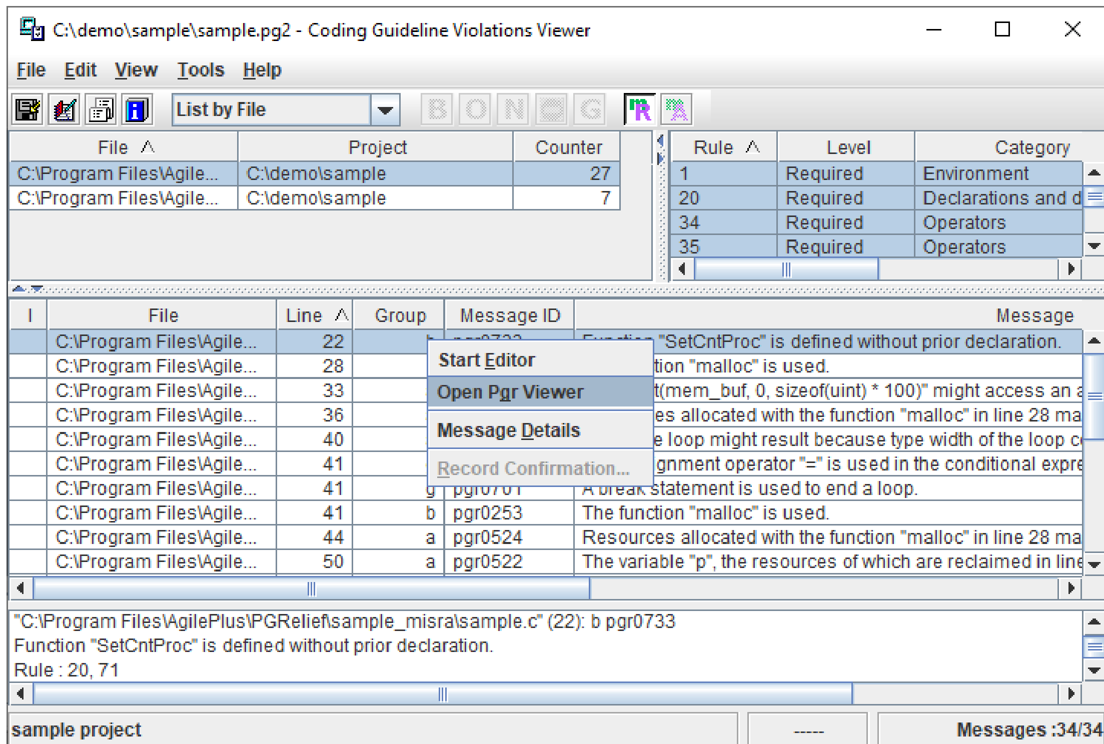
For example, the message ID: pgr0733 indicated in line 22 of the sample corresponds to MISRA-C V1 rules 20 and 71.

Please select the rules in the [List of Rules] window, and then make modifications accordingly; select all rules and make all modifications to each number of the source codes to ensure efficiency and avoid repetition.

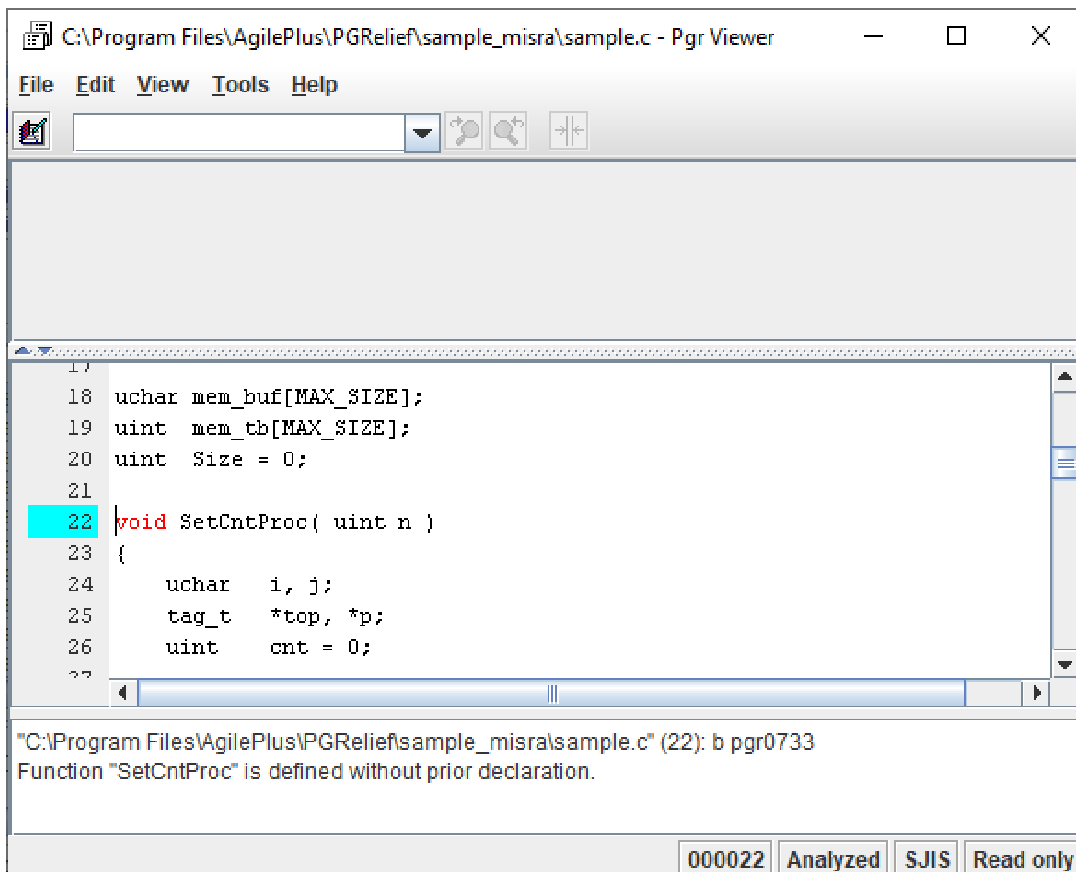
As the source codes of the sample, select the [Line] title to display line numbers in ascending order and modify them from top to bottom.



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

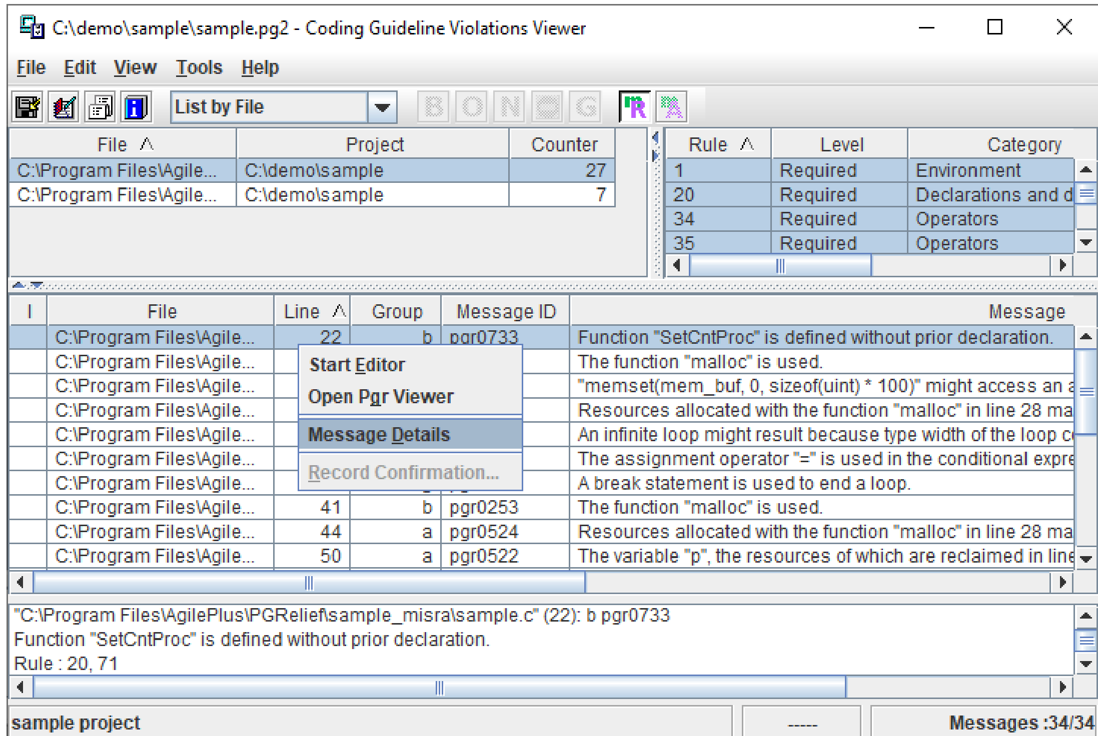


Pgr Viewer will startup.



Confirm the places where modifications are required, and make the according modifications by using the existing editor.

When message information is difficult to understand, please right click to select [Message Details].



The [Help] window will be opened. Please modify with regard to the relevant contents.

Agile+ Relief C/C++ Help

What is Agile+ Relief C/C++?
Forethoughts
Agile+ Relief C/C++ Windows
Agile+ Relief C/C++ Main Wind
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 functio
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.

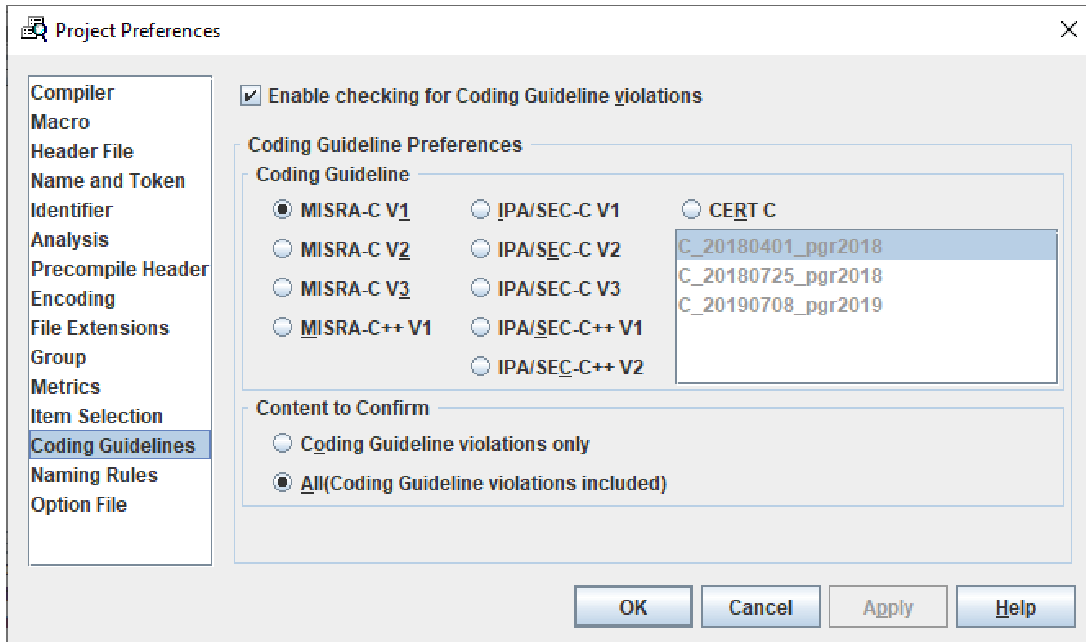
If there is no header file containing the necessary function declaration, add it to one. A prototype declaration should also be added, unless the compiler used is unable to process such declarations.

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

1.3 Experienced with Agile$^+$ Relief C/C++

Project creation using the MISRA option is similar to using normal project.

Indeed, the only point that requires explanation regards appending the [Coding Guidelines] settings in [Project Preference].



In addition, one other point concerning the [Coding Guideline Violations Viewer] window is different, that is "Message ID List" in "Message Information" window is displayed as "Rule Number List". The others are similar in operation.

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

File Edit View Tools Help

List by File

File ^	Project	Counter
C:\Program Files\Agile...	C:\demo\sample	27
C:\Program Files\Agile...	C:\demo\sample	7

Rule ^	Level	Category
1	Required	Environment
20	Required	Declarations and d
34	Required	Operators
35	Required	Operators

I	File	Line ^	Group	Message ID	Message
	C:\Program Files\Agile...	22	b	pgr0733	Function "SetCntProc" is defined without prior declaration.
	C:\Program Files\Agile...	28	b	pgr0253	The function "malloc" is used.
	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...	41	g	pgr0722	The assignment operator "=" is used in the conditional expre
	C:\Program Files\Agile...	41	g	pgr0701	A break statement is used to end a loop.
	C:\Program Files\Agile...	41	b	pgr0253	The function "malloc" is used.
	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_misra\sample.c" (22): b pgr0733
Function "SetCntProc" is defined without prior declaration.
Rule : 20, 71

sample project | ----- | Messages :34/34

1.4 Setting up Rule Checking

If only part of the rules is desired to be checked, create a definition file for rule checking.

Please follow the following format to create the definition file:
Put a semicolon at the beginning of the comment line.
Write only one rule number or group name in a line.
Use single-byte lower-case English letters for group name.
For MISRA-C V1, write explicitly at the beginning of the file

```
; Rule=MISRA-C V1
```

For MISRA-C V2, write explicitly at the beginning of the file

```
; Rule=MISRA-C V2
```

For MISRA-C V3, write explicitly at the beginning of the file

```
; Rule=MISRA-C V3
```

For MISRA-C++ V1, write explicitly at the beginning of the file

```
; Rule=MISRA-C++ V1
```

Example1: Checking MISRA-C V1 rule 10,20,30 and Agile+ Relief's a group

```
;Rule=MISRA-C V1
10
20
30
group-a
```

Example2: Checking MISRA-C V2 rule 1.1,1.2 and Agile+ Relief's a group

```
;Rule=MISRA-C V2
1.1
1.2
group-a
```

Example3: Checking MISRA-C V3 rule 1.3,6.1 and Agile⁺ Relief's a group

```
;Rule=MISRA-C V3
1.3
6.1
group-a
```

Example4: Checking MISRA-C++ V1 rule 0-1-1, 0-1-3 and Agile⁺ Relief's a group

```
;Rule=MISRA-C++ V1
0-1-1
0-1-3
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"

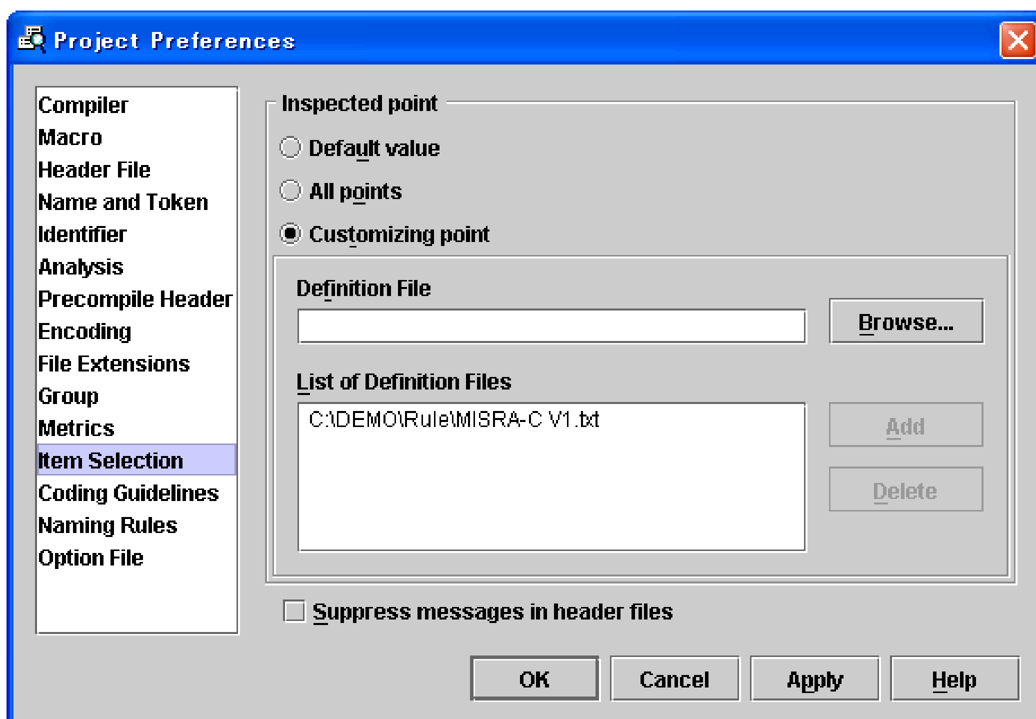
[File name]

MISRA-C V1 : default-misrav1.rul

MISRA-C V2 : default-misrav2.rul

MISRA-C V3 : default-misrav3.rul

MISRA-C++ V1 : default-misrapv1.rul



2 MISRA Options in Command line

This chapter specifies how to use the pgrmisra command in command line.

2.1 Functions of Command

The pgrmisra command is used to check the messages output in pgr5 command and output the messages for MISRA Violation. In addition, the message contents output in pgr5 must be output in files with the CSV format.

2.2 Format of Command

% pgrmisra [-V] [--pgr] -T*MISRA Version* [-Z *definition file for rule checking*] [--qm]
Input File

Note: [ ] means omittable.

2.3 Command Options

Options	Explanation
-V	Displays pgrmisra command version,level and release number. When this option is specified, other options are ignored.
--pgr	Outputs pgr5 command messages simultaneously.
-T*MISRA Version*	Specifies the output MISRA version. Please make sure the option is specified. Please don't insert space between -T option and MISRA version. 1:Checks in accordance with MISRA-C V1. 2:Checks in accordance with MISRA-C V2. 3:Checks in accordance with MISRA-C V3. P1:Checks in accordance with MISRA-C++ V1.
-Z *definition file for rule checking*	If you want to output only certain rules, specify them in the definition file. You may refer to section 1.4 [Setting up Rule Checking] for the syntax of the definition file. If the option on the left is not specified, all rules will be output.
--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 Files	Saves file names of the pgr5 command messages in the CSV format. File will be saved in the following codes. Chinese OS GB2312 code OS in other languages Windows : SJIS code Red Hat Enterprise Linux 5 or 6 : UTF-8 code Others : EUC code

2.4 Return Value

When the pgrmisra command ends normally, it will return to 0. If an error has occurred, it will be returned to a value other than 0.

2.5 Output Files

Output Files will be output in the csv format.

Output Result Samples:

C:\sample.c",36,"52","required","control flowchart","pgr0667","unexecuted."

Format Explanation:

source file name	"C:\sample.c"
line number	36
rule number	"52"
rule category	"required"
rule group	"control flowchart"
message group	"b"
message ID	"pgr0667"
message information	"unexecuted."

2.6 Error Messages

PGRMISRA_0001 Input file is not specified.

Input file is not specified. Please recheck the command format and then re-execute.

PGRMISRA_0002 The option cannot be specified when the input file is specified.

Command Format Error. Please confirm the command format and then re-execute.

PGRMISRA_0003 Multi input files cannot be specified.

Multi input files are specified. Please confirm the command format and then re-execute.

PGRMISRA_0004 Option error.

Option error. Please specify the correct option.

PGRMISRA_0005 -T option is not specified.

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

PGRMISRA_0006 -T option cannot be specified at the same time.

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

PGRMISRA_0040 Parameter of option -Z is not specified.

In option -Z, the definition file for rule checking is not specified. You may execute it again after confirming the command's format.

2.7 Example for Use

2.7.1 Analyze with pgr5 command

Use the pgr5 command to perform source file analysis for the purpose of obtaining a message to be checked by the pgrmisra command.

[Example]:

pgr5 <u>-F MISRA_C_V1.idt</u> <u>--csv</u> <u>sample.c</u> > <u>message.csv</u>

1 2 3 4

1: Specifies the identifier files for MISRA Analysis.

Please specify the following identifier files respectively with the

[Saving Directory for Identifier Files]

Windows:

"(Agile+ Relief C/C++ installation directory)¥Analyze¥EPOM¥MisraInfo"

Linux:

"(Agile+ Relief C/C++ installation directory)/Analyze/EPOM/MisraInfo"

[Type of identifier files]
abidance of MISRA-C Rule Versions:

MISRA-C V1 : MISRA_C_V1.idt

MISRA-C V2 : MISRA_C_V2.idt

MISRA-C V3 : MISRA_C_V3.idt

MISRA-C++ V1 : MISRA_C++_V1.idt

2: Outputs the message in CSV format.

3: Analysis the source files of the objects.

4: Saves the target file of message contents output in CSV format.

For further details regarding the pgr5 command, please refer to the "Command Manual".

2.7.2 Message Checking in the pgrmisra Command

Checks the messages output in the pgr5 command to obtain the messages against MISRA Rules.

[Example]

pgrmisra <u>-T1</u> <u>-Z"F:\Rule\MISRA-C V1.txt"</u> <u>message.csv</u>

1 2 3

1: Specifies the MISRA Rules Version.

Please specify the following options based on MISRA Rules Version determined by -F option in the pgr5 command.

MISRA-C V1 : -T1

MISRA-C V2 : -T2

MISRA-C V3 : -T3

MISRA-C++ V1 : -TP1

2: The check indication definition file specifies the rules you want to apply.

3: Saves the files of message contents of pgr5 command in CSV format.

[Notes]:

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

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

In other language OS, the input file for the pgrmisra 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 Corresponding Indication Messages for the Violation of MISRA Rules

This chapter deals with messages in Agile+ Relief that indicate a violation of MISRA rules has occurred.

3.1 MISRA-C V1

For the following rules, because the rules demand definite system handling behavior, Agile+ Relief cannot check the behaviors when parsing the source code:

Rule 2

Rule 6

Rule 15

Rule 41

Rule 99

Rule 116

Further, since the following rule concerns the coding style of the source code, Agile+ Relief cannot check it:

Rule 10

Rule 1

[Corresponding errors indicated by Agile+ Relief]:

- Unspecified/undefined behaviors, or unsupported descriptions by C90 (ISO/IEC 9899:1990).

[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.]

[Example2]

```
struct   tag {
   int   a : 10;
   int   b : 0;          <-[A bit field of size 0 must be anonymous.]
   int   c : 10;
};
```

Rule 2

Agile$^+$ Relief will not check if the rule has been violated.

Rule 3

[Corresponding errors indicated by Agile+ Relief]:

- Valid keyword asm of the assembly language is used in the C processing system.

[Example1]

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

Rule 4

[Corresponding errors indicated by Agile+ Relief]:

- Bitwise operation has exceeded the type size.
- Divided by zero.
- Out of the range of the array.
- Address "0" may be referenced.

[Example1]

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

[Example2]

```
int   data[10];
:
data[10]=0;             <-[The subscript "10" of the array "data" exceeds the range of
                          the array (array declaration: Line 1 of "file.c").]
```

Rule 5

[Corresponding errors indicated by Agile+ Relief]:

- There is an undefined escape sequence beginning with \.

[Example1]

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

Rule 6

Agile+ Relief will not check if the rule has been violated.

Rule 7

[Corresponding errors indicated by Agile+ Relief]:

- Trigraph sequence is used.

[Example1]

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

Rule 8

[Corresponding errors indicated by Agile+ Relief]:

- "L" denoting a wide character is used.
- A Multiple bytes character is used.

[Example1]

```
L'a'                    <-["L" is used, denoting a wide character.]
L"abc"                  <-["L" is used, denoting a wide character.]
```

[Example2]

```
char   *str = "a い u";          <-[""a い u"" contains a multibyte character.]
```

Rule 9

[Corresponding errors indicated by Agile^{+} Relief]:

- There is a nested comment.

[Example1]

```
/*   /* comment */                    <-[A nested comment is used.]
```

Rule 10

Agile$^+$ Relief will not check if the rule has been violated.

Rule 11

[Corresponding errors indicated by Agile+ Relief]:

- The identifier is more than 31 characters in length.
- The two identifiers differ only in the confusing characters.

[Example1]

```
int   int_length_hensuu_0123456789abcd;
```

<-[The length of the identifier "int_length_hensuu_0123456789abcd" exceeds 31 characters.]

[Example2]

Filename: file.c

```
1: int   xO;
  void   func( void )
  {
     int   x0;
  :
  }
```

<-[The identifier "x0" resembles "xO" in line 1 of "file.c", which may lead to confusion.]

Rule 12

[Corresponding errors indicated by Agile+ Relief]:

- Duplicate identifiers used in different namespaces within the same block.

[Example1]

Filename: file.c

```
10:  struct  name {
        int  name;          <-[The member "name" and the tag in line 10 of "file.c" share
                              the same name.]
       :
     };
     char   *name;          <-[The variable "name" and the tag in line 10 of "file.c" share
                              the same name.]
```

Rule 13

[Corresponding errors indicated by Agile+ Relief]:

- A basic type int/short/char/long/double/float is used.

[Example1]

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

Rule 14

[Corresponding errors indicated by Agile$^+$ Relief]:

- Only a *char* without signed/unsigned is used.

[Example1]

```
char    c ;                <-[A char is used in this file without specifying whether it is
                           signed/unsigned.]
```

Rule 15

Agile+ Relief will not check if the rule has been violated.

Rule 16

[Corresponding errors indicated by Agile+ Relief]:

- A member of floating-point type and members of other types exist in a union at the same time.
- Bitwise operation for floating-point or pointer type.
- A pointer type is cast to other pointer types.

[Example1]

```
union   tag {                  <-[There is a member in union "tag" whose type is different
                                  from that of float-type member "a".]
  double   a;
  unsigned   long   b[2];
};
```

[Example2]

```
double    d;
x = d << 1 ;                   <-[In ANSI, the operand "d" of the bit operation "d << 1" must
                                  be of an integer type.]
```

[Example3]

```
int      *adr1 ;
float     *adr2 ;
adr1 = (int*)adr2 ;            <-[The cast expression "(int*)adr2" casts the pointer type to a
                                  different pointer type. (cast type: float *, expression: int *)]
```

Rule 17

[Corresponding errors indicated by Agile+ Relief]:

- Duplicate typedef name.

[Example1]

```
Filename: file.c
10: typedef   char   *cp;
    void   func( )
    {
        char   *cp;                   <-[The declaration of the variable "cp" uses the same name
                                        as the typedef in line 10 of "file.c", outside the function.]
    }
```

[Example2]

```
Filename: file.c
     void   func1( void )
     {
 3:      typedef   char   BYTE;
         :
     }
     void   func2( void )
     {
         typedef   char   BYTE;      <-[The typedef name "BYTE" is the same as the typedef
                                        name defined in line 3 of "file.c".]
         :
     }
```

[Example3]

```
Filename: file1.h
  typedef   int    WORD ;            <-[The type "WORD" defined with type "int" is defined again
                                        with type "unsigned int" in line 20 of "file2.h".]
Filename: file2.h
20: typedef   unsigned   int    WORD ;
```

Rule 18

[Corresponding errors indicated by Agile^{+} Relief]:

- An integer constant of unsigned type not suffixed by U or u.
- In the subtraction, one operand is a signed integer constant greater than 0, the other is an unsigned expression.

[Example1]

unsigned int x = 0xA123 ;	<-[The unsigned value "0xA123" is used without the suffix "U".]

[Example2]

unsigned short x = 1;	
if((**x - 2**) > 10)	<-[The subtraction "x - 2" is performed with a signed and an unsigned operand.]

Rule 19

[Corresponding errors indicated by Agile+ Relief]:

- An octal constant is used.

[Example1]

```
int    x = 010 ;                    <-[The octal constant "010" is used.]
double    d = 012.3e-1;             <-[The octal constant "012.3e-1" is used.]
```

Rule 20

[Corresponding errors indicated by Agile^{+} Relief]:

- A function is called without the corresponding function declaration.
- A function is defined without the corresponding function declaration.

[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.]

Rule 21

[Corresponding errors indicated by Agile+ Relief]:

- Within the internal effective range, an identifier with the same name as the identifier effective within the external effective range is declared.

[Example1]

```
Filename: file.c
10: unsigned   char   *cp;
    void   func( )
    {
        unsigned   char   *cp;        <-[The declaration of the variable "cp" uses the same name
                                        as the typedef in line 10 of "file.c", outside the function.]
    }
```

[Example2]

```
Filename: file.c
10: struct   tag { short   a, b; }   x;
     void   func( )
     {
         struct   tag { int   a, b; }   y;    <-[The declaration of the tag "tag" uses the same name as
                                              "tag" in line 10 of "file.c", outside the function.]
     }
```

Rule 22

[Corresponding errors indicated by Agile+ Relief]:

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

[Example1]

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.*/
    }
```

Rule 23

[Corresponding errors indicated by Agile+ Relief]:

- The function or the external variable is used in only one file.

[Example1]

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

/*The variable x is used only in the file file.c.*/
```

Rule 24

[Corresponding errors indicated by Agile^{+} Relief]:

- The variables or functions have the same name but with different storage classes.

[Example1]

```
Filename: file.c
10: extern   int   flag;
        :
    static   int   flag;                <-[The variable "flag" has a different storage class in line 10 of
                                        "file.c".]
```

Rule 25

[Corresponding errors indicated by Agile$^+$ Relief]:

- The functions or variables with external connections are assigned identical names.

[Example1]

Filename: file1.c

```
10:   short   *data;
```

Filename: file2.c

```
      short   data;
```

<-[The variable definition "data" and the variable definition in line 10 of "file1.c" are of different types: pointer and non-pointer (short int , short int *).]

[Example2]

Filename: file.c

```
10: int   v = 1;
     :
     float   v = 0.0;
```

<-[The variable definition "v" and the variable definition in line 10 are of different types (float, int).]

Rule 26

[Corresponding errors indicated by Agile+ Relief]:

- The functions or variables with external connections are used with different types.

[Example1]

Filename: file1.c

```
10:   short   *data;
```

Filename: file2.c

```
      extern   short   data;
```

<-[The variable declaration "data" and the variable definition in line 10 of "file1.c" are of different types: pointer and non-pointer (variable declaration: short int, variable definition: short int *).]

[Example2]

Filename: file1.c

```
10:   int   func( char   *p ){ ~ }
```

Filename: file2.c

```
      int   func( long   *p);
```

<-[The 1st parameter "p" in the function declaration "func" and its corresponding parameter "p" in the function definition in line 10 of "file1.c" are of different types (long int *, char *).]

Rule 27

[Corresponding errors indicated by Agile+ Relief]:

- A variable with extern is declared outside the header file.

[Example1]

Filename: file.c

```
extern    int     x ;                              <-[The external variable "x" is declared outside a header file.]
```

Rule 28

[Corresponding errors indicated by Agile+ Relief]:

- The register specifier is used in the function.

[Example1]

```
void   func( void )                              <-[The specifier "register" is used in function "func".]
{
    register   int   i = 3;
    register   float   f = 3.0;
    static   double   D = 3.0;
    :
  return;
}
```

Rule 29

[Corresponding errors indicated by Agile+ Relief]:

- Type mismatch between the initial value and the declared data type.

[Example1]

```
struct   A {
      int   a1;
      int   a2;
};

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.]

[Example2]

```
unsigned   char   x = 0x8000;
```

<-[The initial value "0x80000" exceeds the bit width of "x" of type unsigned char.]

```
struct   T {
   unsigned   char   mem1;
   int   mem2;
};

struct   T   y = {
    65535,
```

<-[The initial value "65535" exceeds the bit width of "mem1" of type unsigned char.]

```
    1
};
```

Rule 30

[Corresponding errors indicated by Agile+ Relief]:

- The automatic variable is being referenced without having been assigned a value.

[Example1]

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

Rule 31

[Corresponding errors indicated by Agile+ Relief]:

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

[Example1]

Code	Message
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.]

Rule 32

[Corresponding errors indicated by Agile$^+$ Relief]:

- Some enumeration constants have been assigned a value while others have not.

[Example1]

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.]

Rule 33

[Corresponding errors indicated by 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.]
```

Rule 34

[Corresponding errors indicated by Agile+ Relief]:

- The expression on the left or the right side of && operator or || operator is not a primary expression.

[Example1]

```
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.]
```

Rule 35

[Corresponding errors indicated by 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".]

Rule 36

[Corresponding errors indicated by Agile^{+} Relief]:

- There is a bitwise operation in the conditional expression.
- A bitwise operation (&, |) is performed on the result of the conditional expression.

[Example1]

if (**!(a & mask)**)	<-[The logical operation "a & mask" uses the result of the operation "!(a & mask)" as a condition.]
if (**a & mask**)	<-[The logical operation "a & mask" uses the result of the operation "a & mask" as a condition.]

[Example2]

if (a < 0 & b < 0)	<-[In "a < 0 & b < 0", & or | may be mistakenly used to represent && or ||.]

Rule 37

[Corresponding errors indicated by Agile+ Relief]:

- Bitwise operation is performed on type char, a signed integer constant or a negative constant.

[Example1]

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

[Example2]

```
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.]
```

Rule 38

[Corresponding errors indicated by Agile^{+} Relief]:

- The number of times that the bitwise shift operator is applied to is a negative constant, or has exceeded the type size of the left variable.

[Example1]

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.]

[Example2]

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.]

Rule 39

[Corresponding errors indicated by Agile^{+} Relief]:

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

[Example1]

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

Rule 40

[Corresponding errors indicated by 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.]
```

Rule 41

Agile+ Relief will not check if the rule has been violated.

Rule 42

[Corresponding errors indicated by Agile+ Relief]:

- There is a comma expression outside the initialization expression and the update expression of the for-statement.

[Example1]

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

Rule 43

[Corresponding errors indicated by 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 ).]
```

[Example4]

```
unsigned int ui = 0xffffffff + 2 ;   <-[The result of constant expression “0xffffffff + 2” containing
```

unsigned constant expression “0xffffffff” is beyond the bit width of unsigned int type.]

Rule 44

[Corresponding errors indicated by Agile+ Relief]:

- The cast expression and the cast type are of different types.

[Example1]

```
unsigned   int    x, y;
   :
x = (unsigned int) y ;
```

<-[The cast operator in the cast expression "(unsigned int)y" is unnecessary (expression type: unsigned int).]

Rule 45

[Corresponding errors indicated by Agile$^+$ Relief]:

- Performing a type conversion for a pointer type.

[Example1]

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

[Example2]

```
Filename: file.c
10: void   func( char   *p ) { ~ }
     :
     int   i;
     func( i );             <-[The 1st argument "i" of the function "func" and the
                            corresponding parameter "p" in the function definition in line
                            10 of "file.c" are of different types: pointer and non-pointer
                            (argument: i, parameter: p).]
```

[Example3]

```
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.]
```

Rule 46

[Corresponding errors indicated by Agile+ Relief]:

- The ANSI operation sequence is indefinite in the expression.

[Example1]

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

[Example2]

```
word_data = get_byte1( ) << 8 | get_byte2( ) ;  <-[The execution sequence of the function calls
                                                "get_byte1" and "get_byte2" is not specified in ANSI.
                                                The function calls should be made in separate lines.]
```

[Example3]

```
Filename: file.c
        x = Gx + func( );          <-[The timing of the function call "func" in line 100 of
                                   "file.c" updating the variable "Gx" is undefined in ANSI,
                                   so the value of "Gx" in the expression cannot be
                                   guaranteed. ]
              :
        int   func( void )
        {
              :
100:       Gx += 2;
              :
        }
```

Rule 47

[Corresponding errors indicated by Agile+ Relief]:

- It's better to add a pair of parentheses.

[Example1]

```
if ( c = input( ) != 0 )
```

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

[Example2]

```
x = a > 0 ? b + c : 0 ;
```

<-[The operand "a > 0" of the ternary operation " a > 0 ? b + c : 0" is not enclosed in parentheses ( ). Use parentheses ( ) to show precedence explicitly.]

<-[The operand "b + c" of the ternary operation " a > 0 ? b + c : 0" is not enclosed in parentheses ( ). Use parentheses ( ) to show precedence explicitly.]

Rule 48

[Corresponding errors indicated by 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.]
```

Rule 49

[Corresponding errors indicated by 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.]
```

Rule 50

[Corresponding errors indicated by Agile+ Relief]:

- One of the expressions on both sides of the operators == or != is of floating-point type.

[Example1]

```
double    d1, d2;
   :
if( d1 == d2 )
```

<-[The result of the comparison "d1== d2" between floating types might not be that expected.]

Rule 51

[Corresponding errors indicated by 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.]

[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 short us = 0xffffffff + 2 ;
```

<- [The result of constant expression “0xffffffff + 2” containing unsigned constant expression “0xffffffff” is beyond the bit width of unsigned short int type.]

Rule 52

[Corresponding errors indicated by Agile+ Relief]:

- There is an unexecuted statement.
- There is an eternally true/false conditional expression.

[Example1]

Code	Comment
for(d = 0; d > 0; d++)	<-[The initialization "d = 0;" in a for statement ensures that the loop condition "d > 0;" is false.]

[Example2]

Code	Comment
if ((x & 0xf0) == 0x01)	<-[The bit pattern of "(x & 0xf0) "is not consistent with that of "0x01".]
if ((x | 0xf0) == 0x01)	<-[The bit pattern of "(x | 0xf0)" is not consistent with that of "0x01".]

[Example3]

Code	Comment
if (**x == 1 && x == 2**)	<-[The condition expression "x == 1 && x == 2" is never true.]
if (**x > 1 && x < 0**)	<-[The condition expression "x > 1 && x < 0" is never true.]

[Example4]

Filename: file.c

Code	Comment
1: if(**x == 1**)	
{ process }	
else if(x == 1)	<-[The condition "x==1" will never be evaluated as true because the same condition exists in line 1 of "file.c".]

Rule 53

[Corresponding errors indicated by Agile^{+} Relief]:

- There is an expression free of side effects.

[Example1]

```
x == 1;                       <-[The expression "x == 1" is meaningless.]
void   func(void);
:
func;                         <-[The expression "func" is meaningless.]
x, y = 1;                     <-[The expression "x" is meaningless.]
```

Rule 54

[Corresponding errors indicated by Agile+ Relief]:

- The parenthesis ')' of the if-statement, for-statement and while-statement is closely followed by a semicolon.
- There is a null statement such as :;or ;;.

[Example1]

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

        x = y;
```

[Example2]

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

Rule 55

[Corresponding errors indicated by Agile+ Relief]:

- A label is used.
- There is a label starting with ' d' , except "default" in the switch statement.

[Example1]

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

[Example2]

```
switch ( x ){
case 1:   ~
          break;
case 2:   ~
          break;
defaulte:                               <-[default may have been misspelled as "defaulte".]
          break;
}
```

Rule 56

[Corresponding errors indicated by Agile+ Relief]:

- A goto statement is used.

[Example1]

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

Rule 57

[Corresponding errors indicated by 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;
 }
```

Rule 58

[Corresponding errors indicated by Agile$^+$ Relief]:

- There is a break statement for jumping out of the loop.

[Example1]

```
for( i = 0; i < 7; i++ ) {
    if( data[ i ] == 0 ) {
        break;                          <-[A break statement is used to end a loop.]
    }
}
```

Rule 59

[Corresponding errors indicated by Agile+ Relief]:

- The if, else, for, while, do-while and switch statements are not embraced by curly brackets { }.

[Example1]

```
if ( x == 0 )
    x = 10;
```

<-[It might be better to add curly brackets { } to this if statement.]

Rule 60

[Corresponding errors indicated by Agile+ Relief]:

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

[Example1]

```
if ( x > 0 ) {
        :
} else 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.]

Rule 61

[Corresponding errors indicated by 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++;
    }
```

Rule 62

[Corresponding errors indicated by 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;
}
```

Rule 63

[Corresponding errors indicated by 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;
}
```

Rule 64

[Corresponding errors indicated by Agile+ Relief]:

- There is no case label in the switch statement.

[Example1]

```
switch ( x ) {                              <-[There is no case label in this switch body. ]
default :
    i += x;
    break ;
}
```

Rule 65

[Corresponding errors indicated by Agile$^+$ Relief]:

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

[Example1]

```
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.]

Rule 66

[Corresponding errors indicated by Agile^{+} Relief]:

- There is an expression irrespective of cycle control in the statement.

[Example1]

```
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.]
```

Rule 67

[Corresponding errors indicated by Agile^{+} Relief]:

- The loop counter in the for-statement has been updated in the loop body.

[Example1]

```
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" ).]
}
```

Rule 68

[Corresponding errors indicated by Agile+ Relief]:

- There is a function declaration within the function.

[Example1]

```
void   func( )
{
    int   func1( int ) ;                <-[A function declaration "func1" is made in a function.]
    int   i ;
```

Rule 69

[Corresponding errors indicated by 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.]

Rule 70

[Corresponding errors indicated by Agile+ Relief]:

- There is a recursive function.

[Example1]

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

Rule 71

[Corresponding errors indicated by 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.]
```

Rule 72

[Corresponding errors indicated by Agile+ Relief]:

- Type mismatch in functions or parameters in the function declaration and definition.

[Example1]

Filename: file1.c

```
10:    unsigned   int   func( char    *p ){ ~ }
```

Filename: file2.c

```
int   func( char *p );
```

<-[The function declaration "func" and the corresponding function definition in line 10 of "file1.c" are of different types (function declaration type: int, function definition type: unsigned int).]

[Example2]

Filename: file1.c

```
10: int   func( char    *p ){ ~ }
```

Filename: file2.c

```
int   func( long *p );
```

<-[The 1st parameter "p" in the function declaration "func" and its corresponding parameter "p" in the function definition in line 10 of "file1.c" are of different types (long int *, char *).]

Rule 73

[Corresponding errors indicated by Agile+ Relief]:

- In the parameter declaration, some parameters have been designated with parameter names, while others have not.

[Example1]

```
int  func( int  a,  int );        <-[In the parameter declaration of function "func", some
                                    parameters are declared with names while others are not.]
:
int  func( int  a,  int  b)
{

   :
}
```

Rule 74

[Corresponding errors indicated by Agile+ Relief]:

- Parameters in the function declaration and definition have been named differently.

[Example1]

Filename: file.c

```
    int   func( int data          <-[The parameter name "data" is inconsistent with the
                                    parameter name in the function definition "func" in line 3 of
                                    "file.c".]
          int size );             <-[The parameter name "size" is inconsistent with the
                                    parameter name in the function definition "func" in line 3 of
                                    "file.c".]
3:  int   func( int   size, int   data )
    {
```

Rule 75

[Corresponding errors indicated by Agile$^+$ Relief]:

- A function without a return value should be defined as void.
- A variable is declared without a type specifier.

[Example1]

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

[Example2]

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

Rule 76

[Corresponding errors indicated by Agile+ Relief]:

- The function is defined in K&R format or without specifying a parameter.
- No parameter has been specified in the function decelerator.

[Example1]

```
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.]

{
   :
}
```

[Example2]

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

Rule 77

[Corresponding errors indicated by Agile+ Relief]:

- Type mismatch between the function argument and the corresponding parameter.

[Example1]

```
Filename: file1.c
10:    void   fa( int *p ) { ~ }

Filename: file2.c
          char   *q;
          :
          fa( q );
```

<-[The 1st argument "q" of the function "fa" and the corresponding parameter "p" in the function definition in line 10 of "file1.c" are of different types (argument: char*, parameter: int *).]

[Example2]

```
Filename: file1.c
10:    void   fb( int   i ) { ~ }

Filename: file2.c
          enum   E { e1, e2};
          fb( e1 );
```

<-[The 1st argument "e1" of the function "fb" and the corresponding parameter "i" in the function definition in line 10 of "file1.c" are of different types (argument: enum E, parameter: int).]

[Example3]

```
Filename: file1.c
10:    void   fc( long   l ) { ~ }

Filename: file2.c
          double   d;
          :
          fc( d );
```

<-[The 1st argument "d" of the function "fc" and the corresponding parameter "l" in the function definition in line 10 of "file1.c" are of different types (argument: double, parameter: long).]

Rule 78

[Corresponding errors indicated by Agile^{+} Relief]:

- The function has different numbers of parameters and arguments

[Example1]

Filename: file1.c

```
10:    void   func( int   mode,   char   c ) { ~ }
```

Filename: file2.c

```
       func( 'a' );                    <-[The number of parameters in the function call "func" is
                                       different from the number of parameters in the corresponding
                                       function definition in line 10 of "file1.c".]
```

Rule 79

[Corresponding errors indicated by Agile+ Relief]:

- Attempt to refer to a return value that does not exist in a function.

[Example1]

Filename: file.c

```
    x = func( 0, 'a' ) ;        <-[An unpredictable value might be referenced because the
                                   function "func" called is defined in line 50 of "file.c" without a
                                   return value.]
50: void func( int mode, char c ) ;
```

Rule 80

[Corresponding errors indicated by Agile+ Relief]:

- The function argument is of a void type.

[Example1]

```
int    func1( int );
void   func2( void );

int    main()
{
    func1( func2( ) );          <-[The 1st argument "q" of the function "fa" and the corresponding parameter "p" in the function definition in line 10 of "file.c" are of different types (argument: char*, parameter: int*).]

    return   0;
}

13: void   func1( int   a )
{
:
   return;
}

void   func2( void )
{
    :
}
```

Rule 81

[Corresponding errors indicated by Agile+ Relief]:

- There is an un-updated parameter of point type or array type.

[Example1]

```
void   func( int   p[ ], unsigned   int   n ) {       <-[The space indicated by parameter "p" of type "int [ ]"
                                                       is not updated and can be qualified with const.]
    int   i;
    int   num = 0;
    for( i = 0; i < n; i++ ) {
        num += p[i];
    }
}
```

[Example2]

```
int   func( int *p ){                                  <-[The parameter "p" can use passing by value.]
                                                       /*   In the function, only referencing of the value of *p is
                                                       performed; *p is not updated and p is not referenced*/
}
```

Rule 82

[Corresponding errors indicated by Agile+ Relief]:

- More than one exits in the function.

[Example1]

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

Rule 83

[Corresponding errors indicated by Agile+ Relief]:

- The called function may have no return value, or types may be mismatched between the called function and the return value.

[Example1]

```
int   func( int a )
{
   if ( a == 0 )
   {   return 0; }
}
```

<-[In the function "func", there are routes that do not have return statements.]

[Example2]

```
int   func( void )
{
        :
   return   0;
        :
   return;
}
```

<-[The return value of a return statement might have been omitted accidentally.]

[Example3]

```
char   *func( int   x )
{
   int   i;
   :
   return   i;
}
```

<-[The return value "i" and the function "func" are of different types: pointer and non-pointer. (Return Value: int , Function: char*)]

Rule 84

[Corresponding errors indicated by Agile+ Relief]:

- There is an expression within the return statement of the void function.

[Example1]

```
void   func( void )
{
   :
   return   0 ;                    <-[The expression "0" appears in the return statement of the
                                     void type function "func".]
}
```

Rule 85

[Corresponding errors indicated by Agile^{+} Relief]:

- Attempt to convert a function name into a non-function pointer.

[Example1]

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.]

Rule 86

[Corresponding errors indicated by 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.]
```

Rule 87

[Corresponding errors indicated by Agile+ Relief]:

- The content before the #include line is neither a preprocessing directive, nor a comment.

[Example1]

Filename: file.c

```
1:  double  func( void )
    {  return  X; }
    #include  "head.h"
```

<-[Statements (in line 1 of "file.c") which are not preprocessing directives exist before this #include line.]

Filename: head.h

```
    extern  double  X;
```

Rule 88

[Corresponding errors indicated by 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.]

Rule 89

[Corresponding errors indicated by Agile^{+} Relief]:

- The include file of the #include statement has not been embraced by <>, or " ".

[Example1]

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

Rule 90

[Corresponding errors indicated by 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;
```

Rule 91

[Corresponding errors indicated by 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++)
:
}
```

Rule 92

[Corresponding errors indicated by Agile^{+} Relief]:

- #undef statement is used.

[Example1]

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

Rule 93

[Corresponding errors indicated by Agile+ Relief]:

- The same parameter appears more than once in the replacement string of the macro function.

[Example1]

```
#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++ ) )
```

Rule 94

[Corresponding errors indicated by Agile$^+$ Relief]:

- The macro function has different numbers of parameters in the definition and the function call.

[Example1]

Filename: file.c

```
10: #define   FUNC(x, y)      (((x)>(y))?(x):(y))
    a = FUNC(x);                          <-[The number of the arguments in a call to the macro
                                          "FUNC" is different from the  number of parameters in the
                                          corresponding macro definition in line 10 of "file.c", in violation
                                          of the ANSI standard.]
```

Rule 95

[Corresponding errors indicated by Agile+ Relief]:

- The argument of the macro function begins with #.

[Example1]

```
#define F(b)    b
F(#error    "memory error")
```

<-[A preprocessing directive is used in the argument "#error" of a macro function.]

Rule 96

[Corresponding errors indicated by Agile+ Relief]:

- The parameter in the macro has not been embraced by parentheses ().
- The replacement string of the macro has not been embraced by () or {}.

[Example1]

```
#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 ( ).]

[Example2]

```
#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 { }.]

Rule 97

[Corresponding errors indicated by Agile+ Relief]:

- The replacement string has not been specified in the macro of the #if or #elif statement.
- A statement, #undef, is used for an undefined macro.

[Example1]

/*The macro DEBUG is undefined, or no replacement string has been specified for it*/

#if DEBUG == 1	<-[The replacement string of the macro "DEBUG" in an #if or #elif statement is not specified.]
#elif DEBUG == 2	<-[The replacement string of the macro "DEBUG" in an #if or #elif statement is not specified.]

[Example2]

/* macro ABC,XYZ has not been defined */

#if ABC	<-["ABC" might have been intended to be appended with defined.]
#elif XYZ	<-["XYZ" might have been intended to be appended with defined.]

[Example3]

/* macro BBB has not been defined */

#undef BBB	<-[#undef is used on the undefined macro "BBB".]

Rule 98

[Corresponding errors indicated by 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".]

Rule 99

Agile[+] Relief will not check if the rule has been violated.

Rule 100

[Corresponding errors indicated by Agile+ Relief]:

- Symbol "defined" is generated from #if or #elif by macro expansion.

[Example1]

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

Rule 101

[Corresponding errors indicated by Agile+ Relief]:

- Arithmetic operation is performed on a pointer which is neither an array nor a pointer to an array.
- Subtraction performed on the pointers that point to different arrays.

[Example1]

```
int    data[ ]={ 1, 2, 3, 4, 5 };
int    *p = data;
return    *(p+1);                 <-[Pointer arithmetic operation "p+1" is not in the form of array
                                    indexing.]
```

[Example2]

```
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;
```

[Example3]

```
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.]
```

Rule 102

[Corresponding errors indicated by Agile+ Relief]:

- The declaration contains an indirect pointer of more than 2 levels.
- More than two levels of indirect pointer operation.

[Example1]

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

[Example2]

```
struct   S {
struct   S   *next;
int   x;
} *sp;
char     ***ppp;
  :
sp->next->next->next = adr;       <-[The expression "sp->next->next->next" has more than 2
                                     levels of pointers.]
***ppp = ' a' ;                   <-[The expression "***ppp" has more than 2 levels of
                                     pointers.]
```

Rule 103

[Corresponding errors indicated by 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    *).]
```

Rule 104

[Corresponding errors indicated by Agile+ Relief]:

- Converting a function name into a non-function pointer.

[Example1]

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.]

Rule 105

[Corresponding errors indicated by Agile+ Relief]:

- Converting a function name into a pointer to a different function.
- Converting a non-function pointer into a function pointer.

[Example1]

```
int   * f(void);
int   * (*fp)(void) = ( void*(*)(void) ) f ;
```

<-[The expression "(void*(*)(void))f" casts a function pointer of type int *(*)(void) to a function pointer of a different type.]

[Example2]

```
void   * vp;
( ( void (*)( int ) ) vp )( 0 ) ;
```

<-["(void(*)(int))vp" converts a pointer of type void * into a function pointer.]

Rule 106

[Corresponding errors indicated by Agile+ Relief]:

- The address of an automatic variable is set to the return value of the function
- The address of an automatic variable is assigned to a variable outside the effective range.

[Example1]

```
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.]
}
```

[Example2]

```
int   *p ;
{
      int     x;
      p = &x ;                   <-[The address of the automatic variable x cannot be used
                                  outside the scope.]
}
*p = 0 ;
```

Rule 107

[Corresponding errors indicated by 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 ;

12: if ( p == NULL ) {
    :
    }
```

<-[The variable "p" compared with the 0 address in line 12 might access the 0 address.]

Rule 108

[Corresponding errors indicated by Agile+ Relief]:

- The array length is omitted.

[Example1]

Filename: file.c

```
#include    "head.h"
int    data[ 256 ] ;
```

Filename: head.h

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

Rule 109

[Corresponding errors indicated by Agile+ Relief]:

- The same loop counter has been used in the for-statements both in the inner layer and the outer layer.
- For a union variable, a member of it has been assigned, but another is referenced.

[Example1]

```
    for( i = 0; i < 5; i++) {        <-[This for statement uses the same loop counter "i" as the for
                                       statement in line 30.]
            :
30:     for( i = 0; i < 10; i++) {
            :
        }
    }
```

[Example2]

```
    union {
        int     m1 ;
        char    m2 ;
    } x ;
10: x.m1 = 1 ;
    c = x.m2 ;                       <-[The union member "x.m1" whose value is assigned in line
                                       10 is referenced by another member "x.m2" of the union]
```

Rule 110

[Corresponding errors indicated by Agile+ Relief]:

- Union members of different sizes are declared.

[Example1]

```
union   u_tag1 {                         <-[Union "u_tag1" contains members of different sizes.]
    double   fdata;
    char     cdata[4];
};
union   u_tag2 {                         <-[Union "u_tag2" contains members of different sizes.]
    struct {
        short    high;
        short    low;
    } word1;
    int      dword;
};
union   u_tag3 {                         <-[Union "u_tag3" contains members of different sizes.]
    struct {
        unsigned   int   b1: 4;
        unsigned   int   b2: 4;
        unsigned   int   b3: 4;
        unsigned   int   b4: 4;
    } bit;
    unsigned int      word2;
};
```

Rule 111

[Corresponding errors indicated by Agile+ Relief]:

- A bit field of neither type signed int nor unsigned int is declared

[Example1]

```
struct   TAG {
    unsigned   short   m1:2 ;         <-[The bitfield "m1" is declared with type unsigned short. The
                                        declaration should be made with signed int or unsigned int.]
    unsigned   int     m2: 2 ;
    signed     int     m3: 2 ;
};
```

Rule 112

[Corresponding errors indicated by 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 ;
};
```

Rule 113

[Corresponding errors indicated by Agile+ Relief]:

- A nameless bit field of more than 1 bit has been declared.
- There is no member name in the declaration of a structure or a union.

[Example1]

```
union   uTAG {
   struct   sTAG {
      unsigned   int   s1: 10 ;
      unsigned   int   s2: 10 ;
      unsigned   int      : 12 ;          <-[No member name exists in the indicated bitfield declaration.]
   } bit;
   long   all;
};
```

Rule 114

[Corresponding errors indicated by Agile+ Relief]:

- The following macros have been redefined or undefined.
  __LINE__
  __FILE__
  __DATE__
  __TIME__

[Example1]

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

Rule 115

[Corresponding errors indicated by Agile+ Relief]:

- The identifier using the same name as one of the key words has been used in the definition and declaration.

The variable and function registered under the labels [RESERVED_IDENTIFIER] and [RESERVED_LIBRARY_IDENTIFIER] of the identifier file have been checked as objects. For more details regarding the registration to the identifier file, please refer to the option -F in "Command Manual".

[Example1]

```
int    errno ;                                    <-[The name "errno" is the same as an external name used in the system.]
void   *malloc( unsigned   int   size ) {         <-[The name "malloc" is the same as an external name used in the system.]
          :
}
```

Rule 116

Agile[+] Relief will not check if the rule has been violated.

Rule 117

[Corresponding errors indicated by 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.]
```

Rule 118

[Corresponding errors indicated by Agile+ Relief]:

- The following dynamic heap allocation functions are used:
  malloc
  calloc
  realloc
  free

[Example1]

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

Rule 119

[Corresponding errors indicated by Agile+ Relief]:

- Identifier errno is used.

[Example1]

```
#include    <errno.h>
:
if( errno == ENOENT )                     <-[The variable "errno" is used.]
```

Rule 120

[Corresponding errors indicated by Agile+ Relief]:

- Macro offsetof is used.

[Example1]

```
#include    <stddef.h>
struct    STAG {
        char    *sp;
        long    dt;
} *stagp;
long    *dtp = (long *)( (char*)stagp + offsetof( struct STAG, dt) );
                                                <-[The macro offsetof" is used.]
```

Rule 121

[Corresponding errors indicated by Agile+ Relief]:

- The file locale.h is included through the #include statement.
- Function setlocale is used.

[Example1]

```
#include    <locale.h>                  <-[The file "locale.h" is included and should be confirmed.]
       :
setlocale( LC_ALL, " " );               <-[The function "setlocale" is used.]
```

Rule 122

[Corresponding errors indicated by Agile+ Relief]:

- Function longjmp and macro setjmp are used.

[Example1]

```
#include    <setjmp.h>
static    jmp_buf    parse_error;
:
longjmp( parse_error, 1);                    <-[The function "longjmp" is used.]
```

Rule 123

[Corresponding errors indicated by Agile+ Relief]:

- File signal.h has been included through #include statement.

  The following functions are used:

  signal
  SIG_DFL
  SIG_IGN
  SIG_ERR
  SIGABRT
  SIGFPE
  SIGILL
  SIGINT
  SIGSEGV
  SIGTERM

[Example1]

```
#include    <signal.h>                  <-[The file "signal.h" is included and should be confirmed.]
extern    void    sigint_hnd( void );
:
signal( SIGINT, sigint_hnd );           <-[The function "signal" is used.]
                                        <-[The macro "SIGINT" is used.]
```

Rule 124

[Corresponding errors indicated by Agile+ Relief]:

- File stdio.h has been included by the #include statement.

  The following functions are used:

    fgetpos
    fopen
    ftell
    gets
    perror
    remove
    rename
    ungetc

[Example1]

```
#include   <stdio.h>                    <-[The file "stdio.h" is included and should be confirmed.]
FILE   *fp;
    :
fp = fopen( "abc.txt", "r");            <-[The function "fopen" is used.]
```

Rule 125

[Corresponding errors indicated by Agile+ Relief]:

- Functions atof, atoi and atoll are used.

[Example1]

```
#include   <stdlib.h>
int   i;
char   *str = "123";
i = atoi( str );                                   <-[The function "atoi" is used.]
```

Rule 126

[Corresponding errors indicated by Agile+ Relief]:

- Functions abort, exit, getenv and system are used.

[Example1]

```
#include    <stdlib.h>
:
if ( j < 0 ) {
abort( );                                   <-[The function "abort" is used.]
}
```

Rule 127

[Corresponding errors indicated by Agile+ Relief]:

- The following functions are used:
  clock
  difftime
  mktime
  time
  asctime
  ctime
  gmtime
  localtime
  strftime

[Example1]

```
#include    <time.h>
:
time_t tm:
time( &tm );                              <-[The function "time" is used.]
```

3.2 MISRA-C V2

For the following rules, because the rules demand definite system handling behavior, Agile+ Relief cannot check the behaviors when parsing the source code:

Rule 1.3
Rule 1.4
Rule 1.5
Rule 3.1
Rule 3.2
Rule 3.3
Rule 3.4
Rule 3.6

Further, since the following rules concerns the coding style of the source code, Agile+ Relief cannot check them:

Rule 2.4
Rule 18.3

Rule 1.1

[The corresponding error indicated by 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.]
} ;
```

Rule 1.2

[The corresponding error indicated by Agile+ Relief]

- Behaviors exist that are unspecified or undefined in C90(ISO/IEC 9899:1990):

[Example 1]

i = j + j++;	<-[The update timing of "j++" is not defined in ANSI, so the value of "j" cannot be guaranteed.]
i = ++i + j;	<-[The update timing of "++i" is not defined in ANSI, so the value of "i" cannot be guaranteed.]

[Example 2]

Filename: file1.c

```
10:    void   func( int   mode, char   c ) { ~ }
```

Filename: file2.c

func('a');	<-[The number of parameters in the function call "func" is different from the number of parameters in the corresponding function definition in line 10 of "file1.c".]

Rule 1.3

Agile+ Relief will not check if this rule has been violated.

Rule 1.4

Agile+ Relief will not check if this rule has been violated.

Rule 1.5

Agile+ Relief will not check if this rule has been violated.

Rule 2.1

[The corresponding error indicated by Agile+ Relief]

- Assembly language is used.

[Example 1]

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

Rule 2.2

[The corresponding error indicated by Agile+ Relief]

- // comment is used

[Example 1]

```
Filename: file.c
int    Data;    //input data                        <-[A "//" style comment is used in the file.]
```

Rule 2.3

[The corresponding error indicated by Agile$^+$ Relief]

- A nested comment exists.

[Example 1]

```
/*    /* comment */                    <-[A nested comment is used.]
```

Rule 2.4

Agile+ Relief will not check if this rule has been violated.

Rule 3.1

Agile+ Relief will not check if this rule has been violated.

Rule 3.2

Agile+ Relief will not check if this rule has been violated.

Rule 3.3

Agile+ Relief will not check if this rule has been violated.

Rule 3.4

Agile+ Relief will not check if this rule has been violated.

Rule 3.5

[The corresponding error indicated by 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 declared.]
    unsigned int m1:1 ;
    unsigned int m2:1 ;
};
```

Rule 3.6

Agile+ Relief will not check if this rule has been violated.

Rule 4.1

[The corresponding error indicated by 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.]

Rule 4.2

[The corresponding error indicated by Agile+ Relief]

- A trigraph sequence is used.

[Example 1]

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

Rule 5.1

[The corresponding errors indicated by Agile+ Relief]

- The length of the identifier exceeds 31.
- The two identifiers are different only with regard to the confusing characters.

[Example 1]

int int_length_hensuu_0123456789abcd;	<-[The length of the identifier "int_length_hensuu_0123456789abcd" exceeds 31 characters.]

[Example 2]

Filename: file.c	
1: int **xO**;	
void func(void)	
{	
int **x0**;	<-[The identifier "x0" resembles "xO" in line 1 of "file.c", which may lead to confusion.]

Rule 5.2

[The corresponding error indicated by Agile+ Relief]

- An identifier with a name identical to that of another identifier within the external scope has been declared within the internal effective scope.

[Example 1]

```
Filename: file.c
10: unsigned   char   *cp;
    void   func( )
    {
        unsigned   char   *cp;          <-[The declaration of the variable "cp" uses the same name
                                         as the typedef in line 10 of "file.c", outside the function.]
    }
```

[Example 2]

```
Filename: file.c
10: struct   tag { short   a, b; } x;
    void   func( )
    {
        struct   tag { int   a, b; } y;   <-[The declaration of the tag "tag" uses the same name as
                                           "tag" in line 10 of "file.c", outside the function.]
    }
```

Rule 5.3

[The corresponding error indicated by Agile+ Relief]

- Duplicate typedef name.

[Example 1]

Filename: file.c

```
10:  typedef  char  *cp;
     void  func( )
     {
          char  *cp;          <-[The declaration of the variable "cp" uses the same name
                                as the typedef in line 10 of "file.c", outside the function.]
     }
```

[Example 2]

Filename: file.c

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

[Example 3]

Filename: file1.h

```
      typedef  int   WORD ;     <-[The type "WORD" defined with type "int" is defined again
                                  with type "unsigned int" in line 20 of "file2.h".]
```

Filename: file2.h

```
20:  typedef  unsigned  int   WORD ;
```

Filename: file.c

```
      #include  "file1.h"
```

```
#include    "file2.h"
```

Rule 5.4

[The corresponding error indicated by Agile+ Relief]

- Duplicate tag name.

[Example 1]

```
Filename: file1.c
10:    struct   tag{ int   a; long   b; };
Filename: file2.c
       struct   tag{ int   x; long   y; };    <-[The tag declaration "tag" is inconsistent with that in line 10 of "file1.c".]
```

[Example 2]

```
Filename: file.c
10:   struct   tag{ int   a; long   b; } ;
        :
      struct   tag{ int   a; long   b; } ;    <-[The tag "tag" has another declaration in line 10 of "file.c". Delete one of the declarations.]
```

[Example 3]

```
Filename: file.c
10: struct   name {
       int   name;                 <-[The member "name" and the tag in line 10 of "file.c" share the same name.]
    :
    };
    char    *name;                 <-[The variable "name" and the tag in line 10 of "file.c" share the same name.]
```

Rule 5.5

[The corresponding error indicated by Agile+ Relief]

- Duplicate variable or function names having an external linkage.

[Example 1]

Filename: file.c

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

[Example 2]

Filename: file1.c

```
10:  short  *data;
```

Filename: file2.c

```
     extern  short  data;       <-[The variable declaration "data" and the variable definition in line 10 of "file1.c" are of different types: pointer and non-pointer (variable declaration: short int, variable definition: short int * ).]
```

[Example 3]

Filename: file1.c

```
10:  int  func( char  *p ){ ~ }
```

Filename: file2.c

```
      int  func( long  *p);     <-[The 1st parameter "p" in the function declaration "func" and its corresponding parameter "p" in the function definition in line 10 of "file1.c" are of different types (long int *, char *).]
```

Rule 5.6

[The corresponding error indicated by Agile+ Relief]

- Duplicate identifiers used within the same block of different namespaces.

[Example 1]

```
Filename: file.c
10: struct   name {
        int   name;              <-[The member "name" and the tag in line 10 of "file.c" share
                                   the same name.]
      :
   };
   char   *name;                 <-[The variable "name" and the tag in line 10 of "file.c" share
                                   the same name.]
```

Rule 5.7

[The corresponding errors indicated by Agile+ Relief]

- An identifier with a name identical to that of another identifier within the external scope has been declared within the internal effective scope.
- Duplicate typedef name.
- Duplicate tag name.
- Duplicate variable or function names having an external linkage.
- Duplicate identifiers used in the same block within different namespaces.

[Example 1]

```
Filename: file.c
10: extern   int   flag;
        :
    static   int   flag;          <-[The variable flag has a different storage class in line 10 of "file.c".]
```

[Example 2]

```
Filename: file.c
10: #define    ABC    1
    #define    ABC    10          <-[The macro "ABC" is redefined with a value different from that in line 10 of "file.c".]
```

[Example 3]

```
Filename: file.c
10: typedef   char   *cp;
    if( data == 0 ) {
          char   *cp;             <-[The declaration of the variable "cp" uses the same name as the typedef in line 10 of "file.c".]
    }
```

Rule 6.1

[The corresponding errors indicated by 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).]

Rule 6.2

[The corresponding error indicated by Agile+ Relief]

- The signed char type or unsigned char type is not used for numeric values.

[Example 1]

```
unsigned   char   x;
       :
x = 'a';                              <-[The signed/unsigned char "x" is not treated as a numeric value.]
```

Rule 6.3

[The corresponding error indicated by 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 arithmetic type (int/short/char/long/double/float) is used directly.]
  :
}
```

Rule 6.4

[The corresponding error indicated by Agile+ Relief]

- A bit field of type neither signed int nor unsigned int is declared.

[Example 1]

```
struct   TAG {
    unsigned   short   m1:2 ;          <-[The bitfield "m1" is declared with type unsigned short. The
                                         declaration should be made with signed int or unsigned int.]
    unsigned   int      m2: 2 ;
    signed     int      m3: 2 ;
};
```

Rule 6.5

[The corresponding error indicated by Agile+ Relief]

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

[Example 1]

```
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 ;
};
```

Rule 7.1

[The corresponding error indicated by 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.]
```

Rule 8.1

[The corresponding errors indicated by Agile+ Relief]

- No function is declared before the function call.
- No function declaration is found before the function definition.
- No specified parameter is found in the function decelerator.

[Example 1]

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

[Example 2]

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

[Example 3]

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

Rule 8.2

[The corresponding errors indicated by 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;
}
```

Rule 8.3

[The corresponding error indicated by Agile+ Relief]

- Type mismatch between function and parameter in the function declaration and definition.

[Example 1]

Filename: file1.c

```
10:    unsigned   func( char   *p ){ ~ }
```

Filename: file2.c

```
       int   func( char   *p );
```

<-[The function declaration "func" and the corresponding function definition in line 10 of "file1.c" are of different types (function declaration type: int, function definition type: unsigned int).]

[Example 2]

Filename: file1.c

```
10:    int   func( char   *p ){ ~ }
```

Filename: file2.c

```
       int   func( long   *p);
```

<-[The 1st parameter "p" in the function declaration "func" and its corresponding parameter "p" in the function definition in line 10 of "file1.c" are of different types (long int *, char *).]

Rule 8.4

[The corresponding error indicated by Agile+ Relief]

- The function, or a variable using the same name and having an external linkage is used.

[Example 1]

Filename: file1.c

```
10:   short   *data;
```

Filename: file2.c

```
      extern   short   data;
```

<-[The variable declaration "data" and the variable definition in line 10 of "file1.c" are of different types: pointer and non-pointer (variable declaration: short int, variable definition: short int *).]

[Example 2]

Filename: file1.c

```
10:   int   func( char   *p ){ ~ }
```

Filename: file2.c

```
      int   func( long   *p );
```

<-[The 1st parameter "p" in the function declaration "func" and its corresponding parameter "p" in the function definition in line 10 of "file1.c" are of different types (long int *, char *).]

Rule 8.5

[The corresponding error indicated by 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"
```

Rule 8.6

[The corresponding error indicated by Agile+ Relief]

- A function is declared in the function.

[Example 1]

```
void   func( )
{
    int   func1( int ) ;                        <-[A function declaration "func1" is made in a function.]
    int   i ;
```

Rule 8.7

[The corresponding error indicated by 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.*/
   }
```

Rule 8.8

[The corresponding error indicated by Agile+ Relief]

- There is an external variable declaration or external function declaration outside of the header file.

[Example 1]

Filename: file.c

```
extern   int    x ;
```

<-[The external variable "x" is declared outside a header file.]

[Example 2]

Filename: file.c

```
int   func( int   in ) ;
```

<-[The external function "func" is declared outside a header file.]

Rule 8.9

[The corresponding error indicated by Agile+ Relief]

- There are function and variable definitions that possess external linkages and are using identical names.

[Example 1]

Filename: file1.c

```
10:     short *data;
```

Filename: file2.c

short data; <-[The variable definition "data" and the variable definition in line 10 of "file1.c" are of different types: pointer and non-pointer (short int , short int *).]

[Example 2]

Filename: file.c

```
10: int    v = 1;
     :
```

float v = 0.0; <-[The variable definition "v" and the variable definition in line 10 are of different types (float, int).]

Rule 8.10

[The corresponding error indicated by Agile+ Relief]

- There exists a external function or an external variable used in only one file.

[Example 1]

```
Filename: file.c
int     x = 0 ;                              <-[The external variable "x" is used only in the file "file.c".]
void    x_add( void )    { x++; }
int     x_ref( void )    { return x; }
/*The variable "x" is only used in the file "file.c".*/
```

Rule 8.11

[The corresponding error indicated by Agile+ Relief]

- There is a variable or a function using identical names but having different storage classes.

[Example 1]

```
Filename: file.c
10: extern   int   flag;
     :
   static   int   flag;          <-[The variable flag has a different storage class in line 10 of "file.c".]
```

Rule 8.12

[The corresponding error indicated by 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.]
```

Rule 9.1

[The corresponding error indicated by 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.]
```

Rule 9.2

[The corresponding error indicated by Agile+ Relief]

- The initializer list and the initialized array/structure/union do not match in the construction.

[Example 1]

```
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.]
```

Rule 9.3

[The corresponding error indicated by 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.]

Rule 10.1

[The corresponding errors indicated by Agile+ Relief]

- There is an operation that performs underlying type conversion.
- There is an implicit conversion that may lead to data loss resulting from differences in signs, type size and types.

[Example 1]

```
void func( short s1, short s2)
{
  long L;
  signed char sc;
  L = s1 * s2;          <-["s1 * s2" in "L = s1 * s2" is converted to a different type by an operation (underlying type before conversion: signed short int; underlying type after conversion: signed long int).]
  sc = s1 + s2;         <-["s1 + s2" in "sc = s1 + s2" is converted to a different type by an operation (underlying type before conversion: signed short int; underlying type after conversion: signed char).]
```

[Example 2]

```
unsigned char x = 0x8000;     <-[The initial value "0x80000" exceeds the bit width of "x" of type unsigned char.]
struct T {
  unsigned char mem1;
  int mem2;
};
struct T y = { 65535,         <-[The initial value "65535" exceeds the bit width of "mem1" of type unsigned char.]
   1
};
```

[Example 3]

Filename: file1.c

```
50: void    func( char    c ) { ... }
```

Filename: file2.c

```
func( 256 );
```

<-[The bit width of the 1st argument "256" of the function "func" exceeds the bit width of the corresponding parameter "c" in the function definition in line 50 of "file1.c"(argument: int , parameter: char).]

Rule 10.2

[The corresponding error indicated by Agile+ Relief]

- There is an implicit conversion causing losses in floating-point data.

[Example 1]

```
double    data;
float    s;
s = data ;
```

<-[In the assignment expression "s = data", the size of the expression on the right is larger than that of the expression on the left, so the correct value might not be assigned (left : float , right : double).]

[Example 2]

```
float    func( void ){
    double    x;
    :
    return    x;
}
```

<-[The correct value might not be returned because the type size of the return value "x" is larger than that of the function "func" (return value: double , function type: float).]

Rule 10.3

[The corresponding errors indicated by Agile+ Relief]

- A complex expression of integer type is cast to a type of the same or larger size.
- A complex expression of integer type is cast to a type of different signs.
- A complex expression of integer type is cast to floating-point type.

[Example 1]

```
unsigned  short  a = 65432;
unsigned  short  b = 65432;
unsigned  long  x;
unsigned  short  y;
```

Code	Message
x = (unsigned long)(a+b);	<-[The cast expression "(unsigned long)(a + b)" casts an integer to a type that is not a smaller type of the same signed/unsigned status (cast type: unsigned long int, underlying type of the expression: unsigned short int).]
y = (unsigned short)(a+b);	<-[The cast expression "(unsigned short)(a + b)" casts an integer to a type that is not a smaller type of the same signed/unsigned status (cast type: unsigned short int, underlying type of the expression: unsigned short int).]

[Example 2]

```
unsigned  short  c = 32765:
```

Code	Message
if((unsigned)(c-2147483647) > 10)	<-[The cast expression "(unsigned)(c - 2147483647)" casts an integer to a type that is not a smaller type of the same signed/unsigned status (cast type: unsigned int, underlying type of the expression: signed int).]

[Example 3]

```
long  a = 3;
long  b = 2;
double  x;
```

Code	Message
x = (double)(a/b);	<-[The cast expression "(double)(a / b)" casts an integer to a floating point type (cast type: double, underlying type of the expression: signed long int).]

Rule 10.4

[The corresponding errors indicated by Agile+ Relief]

- A complex expression of floating-point type is cast to a type of the same or larger size.
- A complex expression of floating-point type is cast to integer type.

[Example 1]

float a = 1.0F/3;

float b = 1.0F/7;

long double x;

float y;

x = **(long double)(a+b);**	<-[The cast expression "(long double)(a + b)" casts a floating point type to a type that is not a smaller type (cast type: long double, expression: float).]
y = (float)(a+b);	<-[The cast expression "(float)(a + b)" casts a floating point type to a type that is not a smaller type (cast type: float, expression: float).]

[Example 2]

float a = 1.0F/3;

float b = 1.0F/7;

long x;

x = **(long)(a+b)**;	<-[The cast expression "(long)(a + b)" casts a floating point type to an integer type (cast type: long int, expression: float).]

Rule 10.5

[The corresponding errors indicated by 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)".]
```

Rule 10.6

[The Corresponding errors indicated by Agile^{+} Relief]

- The unsigned constant is not suffixed by "U".

[Example 1]

```
/* The size of int is 16 bits */
unsigned  int  x = 0xA123 ;        <-[The unsigned value "0xA123" is used without the suffix "U".]
```

Rule 11.1

[The Corresponding errors indicated by Agile+ Relief]

- One side of the cast expression is of type pointer to a function, the other side is of integer type.

[Example 1]

```
int   (*fp)(void);
int   *ip;
ip = (int *)fp;          <-[The cast expression "(int *)fp" is a conversion involving a
                           function pointer type and a non-integer type (cast type: int *,
                           expression type: int ( * ) ( void )).]
```

Rule 11.2

[The corresponding error indicated by 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.]

Rule 11.3

[The corresponding error indicated by Agile^{+} Relief]

- One side of the cast is of pointer type, the other side is of non-pointer type.

[Example 1]

```
int      adr , *p ;
p = (int *)adr ;                 <-[The cast expression "(int *)adr" casts a non-pointer type to
                                 a pointer type (cast type: int *, expression type: int).]
```

Rule 11.4

[The corresponding errors indicated by Agile+ Relief]

- Casting a pointer type to another pointer type.

[Example 1]

```
int      *adr1 ;
char     *adr2 ;
adr1 = (int*)adr2 ;
```

<-[The cast expression "(int*)adr2" casts a pointer type to a different pointer type (cast type: int *, expression type: char *).]

Rule 11.5

[The corresponding error indicated by 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*".]

Rule 12.1

[The Corresponding errors indicated by 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.]

Rule 12.2

[The corresponding error indicated by Agile+ Relief]

- The ANSI operation order of the expression is indefinite.

[Example 1]

i = j + j++ ;	<-[The update timing of "j++" is not defined in ANSI, so the value of "j" cannot be guaranteed.]
i = ++i + j ;	<-[The update timing of "++i" is not defined in ANSI, so the value of "i" cannot be guaranteed.]

[Example 2]

```
word_data = get_byte1( ) << 8 | get_byte2( ) ;
```

<-[The execution sequence of the function calls "get_byte" and "get_byte" is not specified in ANSI. The function calls should be made in separate lines.]

[Example 3]

Filename: file.c

```
        x = Gx + func( );
              :
        int    func( void )
        {
              :
100:       Gx += 2;
              :
        }
```

<-[The timing of the function call "func" in line 100 of "file.c" updating the variable "Gx" is undefined in ANSI, so the value of "Gx" in the expression cannot be guaranteed.]

Rule 12.3

[The corresponding error indicated by Agile+ Relief]

- There is an update expression, a function call or a volatile variable within the sizeof operator.

[Example 1]

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

Rule 12.4

[The corresponding error indicated by Agile+ Relief]

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

[Example 1]

```
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.]
```

Rule 12.5

[The corresponding error indicated by 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.]

Rule 12.6

[The corresponding errors indicated by Agile+ Relief]

- There is a bitwise operation in the conditional expression.
- A bitwise operation (&, |) has been performed on the result of the conditional expression.
- A non-boolean value is used as a conditional expression.

[Example 1]

```
if ( !( a & mask ) )        <-[The logical operation "a & mask" uses the result of the
                              operation "!(a & mask )" as a condition.]
if ( a & mask )             <-[The logical operation "a & mask" uses the result of the
                              operation "a & mask" as a condition.]
```

[Example 2]

```
if ( a < 0 & b < 0 )        <-[In "a < 0 & b < 0", & or | may be mistakenly used to
                              represent && or ||.]
```

[Example 3]

```
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").]
```

Rule 12.7

[The corresponding errors indicated by 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.]
```

Rule 12.8

[The corresponding errors indicated by 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]

Code	Message
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]

Code	Message
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.]

Rule 12.9

[The corresponding error indicated by 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.]
```

Rule 12.10

[The corresponding error indicated by 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) {
```

Rule 12.11

[The corresponding error indicated by Agile+ Relief]

- The operational result cannot be represented by an unsigned type.

[Example 1]

```
#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 )
```

[Example 2]

```
#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" is beyond the bit
                                       width of unsigned int type.]
```

Rule 12.12

[The corresponding errors indicated by Agile+ Relief]

- A member of floating-point type and members of other types exist in a union at the same time.
- Bitwise operation for floating-point or pointer type.
- Cast a pointer type to pointer of other types.

[Example 1]

```
union   tag {          <-[There is a member in union "tag" whose type is different from that of float-type member "a".]
  double   a;
  unsigned   long   b[2];
};
```

[Example 2]

```
double    d;
x = d << 1 ;           <-[In ANSI, the operand "d" of the bit operation "d << 1" must be of an integer type.]
```

[Example 3]

```
int      *adr1 ;
float     *adr2 ;
adr1 = (int*)adr2 ;    <-[The cast expression "(int*)adr2" casts the pointer type to a different pointer type. (cast type: char *, expression: int *)]
```

Rule 12.13

[The corresponding error indicated by Agile^{+} Relief]

- Mixed use of increment/decrement operators and other expressions.

[Example 1]

```
a = b++;              <-[The result of "b++" is used in another operation.]
i = j + j++;          <-[The result of "j++" is used in another operation.]
x++, y = 1;           <-[The result of "x++" is used in another operation.]
```

Rule 13.1

[The corresponding errors indicated by Agile^{+} Relief]

- There is an assignment expression in the conditional expression.
- The assignment operation and comparing operation coexist.

[Example 1]

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".]

Rule 13.2

[The corresponding errors indicated by Agile^{+} Relief]

- Use of a non-boolean value as a conditional expression.
- A conditional expression is being compared with 0.

[Example 1]

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

[Example 2]

```
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").]
```

[Example 3]

```
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.]
```

Rule 13.3

[The corresponding error indicated by 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.]

Rule 13.4

[The corresponding error indicated by 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.]
```

Rule 13.5

[The corresponding error indicated by 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.]

Rule 13.6

[The corresponding error indicated by 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" ).]
```

Rule 13.7

[The corresponding errors indicated by Agile+ Relief]

- There is an eternally true/false conditional expression.
- The conditional expression may be eternally true/false, depending on the signs and the type size.

[Example 1]

```
if ( x == 1 && x == 2 )        <-[The condition expression "x == 1 && x == 2" is never true.]
if ( x > 1 && x < 0 )          <-[The condition expression "x > 1 && x < 0" is never true.]
```

[Example 2]

```
while( x * 10 - y > 0 ){        <-[The loop condition "x * 10 - y > 0" does not change.]
   :
}
/* The variables x and y are not updated in the while statement.*/
```

[Example 3]

```
char   c;
   :
if ( c == -1 ) {               <-[If char is unsigned, "-1" cannot be equal to "c".]
```

[Example 4]

```
short   s;
:
if ( s == 65536 )              <-[Comparison of "s" of type short int with the constant
                                 "65536" that exceeds the bit width of type short int is a
                                 mistake.]
```

Rule 14.1

[The corresponding error indicated by 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.]
```

Rule 14.2

[The corresponding error indicated by Agile+ Relief]

- There is an expression free of side effect.

[Example 1]

```
x == 1;                     <-[The expression "x == 1" is meaningless.]
void   func( void );
:
func;                       <-[The expression "func" is meaningless.]
x, y = 1;                   <-[The expression "x" is meaningless.]
```

Rule 14.3

[The corresponding errors indicated by 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.]
```

Rule 14.4

[The corresponding error indicated by Agile+ Relief]

- A goto statement is used.

[Example 1]

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

Rule 14.5

[The corresponding errors indicated by Agile+ Relief]

- A continue statement is used.

[Example 1]

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;
  }
```

Rule 14.6

[The corresponding error indicated by 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;
    }
}
```

Rule 14.7

[The corresponding errors indicated by Agile+ Relief]

- More than one exit in the function.

[Example 1]

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

Rule 14.8

[The corresponding errors indicated by Agile+ Relief]

- Statements controlled by for, while, do-while and switch are not embraced by { } .

[Example 1]

```
while ( x > 0 )                  <-[It might be better to add curly brackets { } to this while statement.]

        x--;
```

Rule 14.9

[The corresponding errors indicated by Agile+ Relief]

- Statements controlled by if, else statement have not been embraced by { }.

[Example 1]

```
if ( x == 0 )                    <-[It might be better to add curly brackets { } to this if statement.]
        x = 10;
```

Rule 14.10

[The corresponding errors indicated by Agile+ Relief]

- The if and else if statements do not end with an else statement.

[Example 1]

```
if ( x > 0 ) {
    :
} else 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.]

Rule 15.1

[The corresponding errors indicated by 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;
  }
```

Rule 15.2

[The corresponding error indicated by Agile+ Relief]

- The case label has no corresponding break statement.

[Example 1]

```
    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++;
    }
```

Rule 15.3

[The corresponding errors indicated by Agile^{+} Relief]

- No default label has been found in the switch statement.
- A default label is placed before the case label.

[Example 1]

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

[Example 2]

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

Rule 15.4

[The corresponding errors indicated by Agile+ Relief]

- There is a comparing operator, logical operator and a constant in the conditional expression of switch statement.
- There is only one case label in the switch statement.

[Example 1]

```
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.]
{ … }
```

[Example 2]

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

Rule 15.5

[The corresponding errors indicated by Agile+ Relief]

- No case label is found in the switch statement.

[Example 1]

```
switch ( x ) {                          <-[There is no case label in this switch body.]
  default :
    i += x;
    break ;
}
```

Rule 16.1

[The corresponding errors indicated by Agile+ Relief]

- There is a variable length parameter in the function declaration and definition.
- No parameter is found in the function decelerator.

[Example 1]

Code	Message
`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 "...".]
`}`	

[Example 2]

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

Rule 16.2

[The corresponding error indicated by Agile+ Relief]

- There is a recursive function.

[Example 1]

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

Rule 16.3

[The corresponding errors indicated by Agile^{+} Relief]

- There is a nameless parameter in the parameter declaration of the function.

[Example 1]

```
int   func( int   a, int );          <-[In the parameter declaration of function "func", some
                                       parameters are declared with names while others are not.]
:
int   func( int   a, int   b )
{
   :
}
```

[Example 2]

```
void   func( int, int );             <-[None of the parameters in the declaration of the function
                                       "func" have names.]
```

Rule 16.4

[The corresponding error indicated by Agile+ Relief]

- The parameters of a function declaration and definition have different names.

[Example 1]

Filename: file.c

int func(int data,	<-[The parameter name "data" is inconsistent with the parameter name in the function definition "func" in line 3 of "file.c".]
int size);	<-[The parameter name "size" is inconsistent with the parameter name in the function definition "func" in line 3 of "file.c".]

```
3: int func( int size, int data )
   {
   :
   }
```

Rule 16.5

[The corresponding errors indicated by 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.]
```

Rule 16.6

[The corresponding error indicated by Agile+ Relief]

- The function has different numbers of arguments and parameters.

[Example 1]

Filename: file1.c

```
10:     void    func( int    mode, char c ) { ~ }
```

Filename: file2.c

```
        func( 'a' );
```

<-[The number of parameters in the function call "func" is different from the number of parameters in the corresponding function definition in line 10 of "file1.c".]

Rule 16.7

[The corresponding error indicated by Agile+ Relief]

- Un-updated parameters of pointer type or array type.

[Example 1]

```
void   func( int   p[ ], unsigned   int   n )<-[The space indicated by parameter "p" of type "int [ ]" is not
                                              updated and can be qualified with const. ]
{
    int   i;
    int   num = 0;
    for( i = 0; i < n; i++ ) {
        num += p[ i ];
    }
}
```

[Example 2]

```
int   func( int   *p )                        <-[The parameter "p" can use passing by value.]
{
/*   *p is referenced in the function, but *p has not been undated or p has been referenced before
                                              this.*/
}
```

Rule 16.8

[The corresponding error indicated by 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.]
}
```

Rule 16.9

[The corresponding error indicated by 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.]
```

Rule 16.10

[The corresponding errors indicated by Agile+ Relief]

- The return value of a non-void function is ignored.
- An abnormal value is returned by the function without having been checked.

  The function using a name identical to that of the identifier registered under the label [NULL_RETURN_FUNCTION] and [SET_VARIABLE_FUNCTION] of the identifier file is checked as an object. For details regarding registration to an identifier file, see the - F option in the "Command Manual".

[Example 1]

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

[Example 2]

```
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.]
```

Rule 17.1

[The corresponding errors indicated by Agile+ Relief]

- An arithmetic operation performed on a pointer of non-array type.
- Use of a non-zero array pointer 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.]
:
}
```

Rule 17.2

[The corresponding error indicated by 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.]
```

Rule 17.3

[The corresponding error indicated by Agile+ Relief]

- Make magnitude comparisons between the addresses of two different objects.

[Example 1]

```
char    *p;
long    *q;
if( p < q ) {
```

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

Rule 17.4

[The corresponding error indicated by Agile+ Relief]

- A pointer arithmetic operation is not in the array indexing form .

[Example 1]

```
int    data[ ]={ 1, 2, 3, 4, 5 };
int    *p = data;
return    *(p+1);                  <-[Pointer arithmetic operation "p+1" is not in the form of array
                                     indexing.]
```

Rule 17.5

[The corresponding error indicated by Agile+ Relief]

- The declaration contains more than 2 levels of indirect pointers.

[Example 1]

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

Rule 17.6

[The corresponding errors indicated by 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 ;
```

Rule 18.1

[The corresponding error indicated by Agile+ Relief]

- The array length has been omitted.

[Example 1]

```
struct   TAG {
   int   a;
   char   b[ ];                              <-[The length of an array has been omitted.]
};
```

[Example 2]

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

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

Rule 18.2

[The corresponding errors indicated by Agile+ Relief]

- For a union variable, a member of it has been assigned, but another is referenced.

[Example 1]

```
    union {
      int     m1 ;
      char   m2 ;
    } x ;
10: x.m1 = 1 ;
    c = x.m2 ;                <-[The union member "x.m1" whose value is assigned in line
                              10 is referenced by another member "x.m2" of the union]
```

Rule 18.3

Agile+ Relief will not check if this rule has been violated.

Rule 18.4

[The corresponding error indicated by Agile+ Relief]

- A union is used.

[Example 1]

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

Rule 19.1

[The corresponding error indicated by Agile+ Relief]

- Before the line #include, contents exist that are neither preprocessing directives nor comments.

[Example 1]

```
Filename: file.c
1: double   func( void )
   {   return   X; }
   #include "head.h"          <-[Statements ( in line 1 of "file.c" ) which are not
                               preprocessing directives exist before this #include line.]

Filename: head.h
  extern double X;
```

Rule 19.2

[The corresponding error indicated by Agile+ Relief]

- In the #include-specified file name, there are '(single quotation), \(currency character), "(double quote), /*(slash and asterisk) and multi-bytes characters(Chinese characters, etc).

[Example 1]

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

Rule 19.3

[The corresponding errors indicated by 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 " ".]

Rule 19.4

[The corresponding errors indicated by Agile^{+} Relief]

- The parenthesis, "(" and ")", in the replacement string of the #define statement do not match.
- The curly brackets "{" and "}" in the replacement string of the #define statement do not match.
- The replacement string of the #define statement consists only of the following types: char, short, int, long, float, double, signed, unsigned and void.

[Example 1]

```
#define   AddTen(x)    (x + 10        <-[The replacement string of the macro function "AddTen"
                                        defined with #define contains one or more unmatched
                                        parentheses.]
```

[Example 2]

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

Rule 19.5

[The corresponding error indicated by Agile+ Relief]

- #define or #undef statements exist in the block.

[Example 1]

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

Rule 19.6

[The corresponding error indicated by Agile+ Relief]

- A #undef is used.

[Example 1]

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

Rule 19.7

[The corresponding error indicated by 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++) )
```

Rule 19.8

[The corresponding error indicated by Agile+ Relief]

- Different numbers of parameters exist in the definition and call of a macro function.

[Example 1]

Filename: file.c

```
10: #define    FUNC(x,y)       (((x)>(y))?(x):(y))
    a = FUNC(x);                        <-[The number of the arguments in a call to the macro
                                        "FUNC" is different from the  number of parameters in the
                                        corresponding macro definition in line 10 of "file.c", in violation
                                        of the ANSI standard.]
```

Rule 19.9

[The corresponding error indicated by Agile+ Relief]

- The argument of the macro function begins with #.

[Example 1]

```
#define    F(b)    b
F(#error "memory error")
```

<-[A preprocessing directive is used in the argument "#error" of a macro function.]

Rule 19.10

[The corresponding error indicated by Agile^{+} Relief]

- The parameter in the replacement string of the macro is not embraced with parentheses ().

[Example 1]

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

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

Rule 19.11

[The corresponding errors indicated by Agile+ Relief]

- An unspecified replacement string exists in the #if or #elif statement.
- An #undef is used on the undefined macro.

[Example 1]

/*The macro DEBUG is undefined, or no replacement string has been specified for it*/

Code	Error
#if DEBUG == 1	<-[The replacement string of the macro "DEBUG" in an #if or #elif statement is not specified.]
#elif DEBUG == 2	<-[The replacement string of the macro "DEBUG" in an #if or #elif statement is not specified.]

[Example 2]

/* There are no definitions of macro ABC,XYZ*/

Code	Error
#if ABC	<-["ABC" might have been intended to be appended with defined.]
#elif XYZ	<-["XYZ" might have been intended to be appended with defined.]

[Example 3]

/* There are no definition of macro BBB*/

Code	Error
#undef BBB	<-[#undef is used on the undefined macro "BBB".]

Rule 19.12

[The corresponding error indicated by Agile+ Relief]

- The #operator and the ##operator coexist in the macro function.

[Example 1]

```
#define   AAA(x,y)   x###y
```

<-[The operators # and ## are used together in the definition of the macro function "AAA".]

Rule 19.13

[The corresponding error indicated by 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.]
```

Rule 19.14

[The corresponding errors indicated by 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.]

Rule 19.15

[The corresponding error indicated by 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;
```

Rule 19.16

[The corresponding errors indicated by 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.]

Rule 19.17

[The corresponding error indicated by 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.]

Rule 20.1

[The corresponding errors indicated by 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]

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

[Example 2]

Code	Comment
#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.]

Rule 20.2

[The corresponding error indicated by Agile+ Relief]

- An identifier using a name identical to that of the reserved identifier is used in the declaration and definition.

The variable and function using identical names registered under the label [RESERVED_IDENTIFIER] and [RESERVED_LIBRARY_IDENTIFIER] of the identifier file are checked as an object. For more information regarding the registration of an identifier file, see the -F option in the "Command Manual".

[Example 1]

```
int     errno ;                              <-[The name "errno" is the same as an external name used in
                                             the system.]
void   *malloc( unsigned   int   size ) {  <-[The name "malloc" is the same as an external name used
                                             in the system.]
            :
}
```

Rule 20.3

[The corresponding errors indicated by Agile+ Relief]

- The return value of the malloc or fopen functions has been referred to without checking whether it is NULL.
- The variable to whose value an address of "0" has been set is referenced.

[Example 1]

```
    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.]
```

[Example 2]

```
    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.]
```

Rule 20.4

[The corresponding error indicated by 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.]
```

Rule 20.5

[The corresponding error indicated by Agile+ Relief]

- An identifier errno is used.

[Example 1]

```
#include    <errno.h>
:
if( errno == ENOENT )                    <-[The variable "errno" is used.]
```

Rule 20.6

[The corresponding error indicated by Agile+ Relief]

- A macro offsetof is used.

[Example 1]

```
#include    <stddef.h>
struct    STAG {
        char    *sp;
        long    dt;
} *stagp;
long    *dtp = (long *)( (char*)stagp + offsetof( struct STAG, dt) );
                                                  <-[macro "offsetof" is used]
```

Rule 20.7

[The corresponding error indicated by Agile+ Relief]

- The longjmp and macro setjmp functions are used.

[Example 1]

```
#include    <setjmp.h>
static    jmp_buf    parse_error;
:
longjmp( parse_error, 1 );                    <-[The function "longjmp" is used.]
```

Rule 20.8

[The corresponding errors indicated by Agile+ Relief]

- File signal.h has been included by the #include statement.
- The following functions and macros are used:
  signal
  SIG_DFL
  SIG_IGN
  SIG_ERR
  SIGABRT
  SIGFPE
  SIGILL
  SIGINT
  SIGSEGV
  SIGTERM

[Example 1]

```
#include   <signal.h>                  <-[The file "locale.h" is included and should be confirmed.]
extern   void   sigint_hnd( void );
:
signal( SIGINT, sigint_hnd );          <-[The function "signal" is used.]
                                       <-[The macro "SIGINT" is used.]
```

Rule 20.9

[The corresponding errors indicated by Agile+ Relief]

- File stdio.h is included by the #include statement.
- The following functions are used:
  fgetpos
  fopen
  ftell
  gets
  perror
  remove
  rename
  ungetc

[Example 1]

```
#include   <stdio.h>                    <-[The file "locale.h" is included and should be confirmed.]
FILE   *fp;
     :
fp = fopen( "abc.txt", "r");            <-[The function "fopen" is used.]
```

Rule 20.10

[The corresponding error indicated by Agile+ Relief]

- The atof, atoi and atoll functions are used.

[Example 1]

```
#include    <stdlib.h>
int    i;
char    *str = "123";
i = atoi( str );                                    <-[The function "atoi" is used]
```

Rule 20.11

[The corresponding error indicated by Agile+ Relief]

- The abort, exit, getenv, and system functions are used.

[Example 1]

```
#include    <stdlib.h>
:
if ( j < 0 ) {
abort( );                                   <-[The function "abort" is used.]
}
```

Rule 20.12

[The corresponding error indicated by Agile+ Relief]

- The following functions are used:
  clock
  difftime
  mktime
  time
  asctime
  ctime
  gmtime
  localtime
  strftime

[Example 1]

```
#include    <time.h>
:
time_t    tm:
time( &tm );                              <-[The function "time" is used.]
```

Rule 21.1

[The corresponding errors indicated by Agile^{+} Relief]

- Bitwise shift operation out of the type size
- Divided by zero
- Out of the range of the array
- Address "0" may be referenced.

[Example 1]

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

[Example 2]

```
int   data[10];
:
data[10]=0;                    <-[The subscript "10" of the array "data" exceeds the range of
                               the array (array declaration: Line 1 of "file.c").]
```

3.3 MISRA-C V3

Agile+ Relief checks all rules provided for by MISRA-C:2012 and MISRA-C:2012 Amendment 1.

[MISRA-C V3] checks Coding Guideline violations by the rule added by MISRA-C:2012 Amendment 1 in addition to MISRA-C:2012. Please exclude the following rules from the inspection object by Check indication definition file when you want to check Coding Guideline violations by the rule of MISRA-C:2012.

Rule 12.5
Rule 21.16
Rule 21.17
Rule 21.18
Rule 21.19
Rule 21.20
Rule 22.7
Rule 22.8
Rule 22.9
Rule 22.10

* Please refer to "1.4 Setting up Rule Checking" to describe Check indication definition file.

Rule 1.1

[The corresponding error indicated by Agile+ Relief]

- There is a description that violates Standard C syntax and Restriction.

[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.]
 } ;
```

Rule 1.2

[The corresponding error indicated by Agile+ Relief]

- Assembly language is used.

[Example 1]

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

Rule 1.3

[The corresponding error indicated by Agile+ Relief]

- Behaviors exist that are unspecified or undefined:

[Example 1]

i = j + j++; <-[The update timing of "j++" is not defined in ANSI, so the value of "j" cannot be guaranteed.]

i = ++i + j; <-[The update timing of "++i" is not defined in ANSI, so the value of "i" cannot be guaranteed.]

[Example 2]

Filename: file1.c

```
10:    void    func( int    mode, char    c ) { ~ }
```

Filename: file2.c

func('a'); <-[The number of parameters in the function call "func" is different from the number of parameters in the corresponding function definition in line 10 of "file1.c".]

Rule 2.1

[The corresponding error indicated by 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.]
```

Rule 2.2

[The corresponding error indicated by Agile+ Relief]

- There is an expression free of side effect.

[Example 1]

```
x == 1;                                   <-[The expression "x == 1" is meaningless.]
void    func( void );
:
func;                                     <-[The expression "func" is meaningless.]
x, y = 1;    /* Comma expression */       <-[The expression "x" is meaningless.]
```

Rule 2.3

[The corresponding error indicated by Agile^{+} Relief]

- The type declared in the source file are not being used in the same source file.

[Example 1]

```
typedef struct {
   char Name[64] ;
   int    Age ;
} person_t ;       <-[The type "person_t" is not used.]
/* The type "person_t"is not used in the same source file declared the type "person_t". */
```

Rule 2.4

[The corresponding error indicated by Agile+ Relief]

- The tag declared in the source file are not being used in the same source file.

[Example 1]

```
struct person {      <-[The tag "person" is not used.]
   char Name[64] ;
   int   Age ;
} ;
/* The tag "person" is not used in the same source file declared the tag "person". */
```

Rule 2.5

[The corresponding error indicated by Agile+ Relief]

- The macro declared in the source file are not being used in the same source file.

[Example 1]

```
#define   TRUE       1      <-[The macro "TRUE" is not used.]
#define   FALSE      0
#define   MAXSIZE   8      <-[The macro "MAXSIZE" is not used.]
          :
int   f ( int n )   {
    int   data[ 8 ] ;          /* It intended to be "int data[ MAXSIZE ] ; ". */
    if   ( n < 8   )   {       /* It intended to be "if ( n < MAXSIZE ) {". */
          return data[ n ];
    }
    return   FALSE ;
}
/* The macro"TRUE" and the macro "MAXSIZE" are not used in the same source file declared the macro"TRUE" and the macro "MAXSIZE". */
```

Rule 2.6

[The corresponding error indicated by Agile+ Relief]

- There is a label not referred to by the goto statement.

[Example 1]

```
labelA:              <-[The label "labelA" is not used.]
     : [1]
```

[1] No goto statement references the label labelA.

Rule 2.7

[The corresponding error indicated by Agile+ Relief]

- An unused parameter exists.

[Example 1]

```
void func(  int  x , int  y  ) {          <-[The parameter "x" is not used.]
    cls( y );
    return;
}
```

Rule 3.1

[The corresponding error indicated by Agile+ Relief]

- A nested comment exists.

[Example 1]

```
/*    /* comment */                              <-[A nested comment is used.]
```

Rule 3.2

[The corresponding error indicated by Agile+ Relief]

- Line-splicing is described at the end of line in "//" comments.

[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. */
    }
```

Rule 4.1

[The corresponding error indicated by Agile+ Relief]

- The escape sequence of the octal number and the hexadecimal has mixed with a universal character in the string literal.

[Example 1]

char *s2 = **"a\x20z "** ; <-[The octal and hexadecimal escape sequences are mixed with universal characters.]

Rule 4.2

[The corresponding error indicated by Agile^{+} Relief]

- A trigraph sequence is used.

[Example 1]

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

Rule 5.1

[The corresponding error indicated by Agile+ Relief]

- The length of the identifier with the external linkage exceeds the length provided for by the ANSI standard. (In C90, the identifier is specified by 6 characters or less. In C99, the identifier is specified by 31 characters or less.)
- Two or more identifiers of the same name with only the distinction of uppercase and lowercase with the external linkage are defined.

[Example 1]

```
/* Pattern of C90 */
int     A23456G;   <-[The length of the external identifier "A23456G" exceeds 6 characters.]
/* Pattern of C99 (if having specified options corresponding to C99 syntax.) */
int     A234567890123456789012345678901G;   <-[The length of the external identifier
                                               "A234567890123456789012345678901G"
                                               exceeds 31 characters.]
```

[Example 2]

```
File: a.c
Line  10:  void  sub( void ) {
File: b.c
Line  20:  void  SUB( void ) {  <-[The name of "SUB" and "sub" at the line 10 of file "a.c" are only
                                   different in case.]
```

Rule 5.2

[The corresponding error indicated by Agile+ Relief]

- The length of the identifier without the external linkage exceeds the length provided for by the ANSI standard. (In C90, the identifier is specified by 31 characters or less. In C99, the identifier is specified by 63 characters or less.)

[Example 1]

```
/* Pattern of C90 */
struct   STR {
int   m2345678901234567890123456789012;
    <-[The length of the identifier "m2345678901234567890123456789012" exceeds 31 characters.]
        :
void func(void) {
int   n2345678901234567890123456789012;
    <-[The length of the identifier "n2345678901234567890123456789012" exceeds 31 characters.]
        :
/* Pattern of C99 (if having specified options corresponding to C99 syntax.) */
struct   STR {
int   m234567890123456789012345678901234567890123456789012345678901234;
    <-[The length of the identifier
    "m234567890123456789012345678901234567890123456789012345678901234" exceeds 63
    characters.]
        :
void func(void) {
int   n234567890123456789012345678901234567890123456789012345678901234;
    <-[The length of the identifier
    "n234567890123456789012345678901234567890123456789012345678901234" exceeds 63
    characters.]
        :
```

Rule 5.3

[The corresponding error indicated by Agile+ Relief]

- An identifier with a name identical to that of another identifier within the external scope has been declared within the internal effective scope.

[Example 1]

```
Filename: file.c
10: unsigned   char   *cp;
    void   func( )
    {
        unsigned   char   *cp;          <-[The declaration of the variable "cp" uses the same name
                                          as the typedef in line 10 of "file.c", outside the function.]
    }
```

[Example 2]

```
Filename: file.c
10: struct   tag { short   a, b; } x;
    void   func( )
    {
        struct   tag { int   a, b; } y;     <-[The declaration of the tag "tag" uses the same name as
                                             "tag" in line 10 of "file.c", outside the function.]
    }
```

Rule 5.4

[The corresponding error indicated by Agile+ Relief]

- There is a macro name that exceeds the length provided for by ANSI standard. (In C90, the identifier is specified by 31 characters or less. In C99, the identifier is specified by 63 characters or less.)

[Example 1]

/* Pattern of C90 */

#define **M2345678901234567890123456789012**;

<-[The length of the macro name "M2345678901234567890123456789012" exceeds 31 characters.]

:

/* Pattern of C99 (if having specified options corresponding to C99 syntax.) */

#define **M234567890123456789012345678901234567890123456789012345678901234**;

<-[The length of the macro name "M234567890123456789012345678901234567890123456789012345678901234" exceeds 63 characters.]

Rule 5.5

[The corresponding error indicated by Agile+ Relief]

- The identifier of the same name as the function-like macro is used.

[Example 1]

```
File: a.h
Line    1: #define MAX( x , y )     (((x)>(y))?(x):(y))
File: a.c
Line    1: #include    "a.h"
2:   int    MAX ;        <-["MAX" is the macro function which defined in the line 1 of "a.h" was not
                          replaced.]
3:
4:   void func( int x , int y ) {
5:        int    r ;
6:        r    =    MAX( x , y ) ; /* It is substituted for the macro function "MAX" */
```

Rule 5.6

[The corresponding error indicated by Agile+ Relief]

- Duplicate typedef name.

[Example 1]

Filename: file.c

```
10:   typedef   char   *cp;
      void   func( )
      {
           char   *cp;               <-[The declaration of the variable "cp" uses the same name
                                       as the typedef in line 10 of "file.c", outside the function.]
      }
```

[Example 2]

Filename: file.c

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

[Example 3]

Filename: file1.h

```
       typedef   int    WORD ;       <-[The type "WORD" defined with type "int" is defined again
                                       with type "unsigned int" in line 20 of "file2.h".]
```

Filename: file2.h

```
20:   typedef   unsigned   int    WORD ;
```

Filename: file.c

```
       #include   "file1.h"
```

```
#include    "file2.h"
```

Rule 5.7

[The corresponding error indicated by Agile+ Relief]

- Duplicate tag name.

[Example 1]

Filename: file1.c

10: struct tag{ int a; long b; };

Filename: file2.c

struct tag{ int x; long y; }; <-[The tag declaration "tag" is inconsistent with that in line 10 of "file1.c".]

[Example 2]

Filename: file.c

10: struct tag{ int a; long b; } ;

:

struct tag{ int a; long b; } ; <-[The tag "tag" has another declaration in line 10 of "file.c". Delete one of the declarations.]

[Example 3]

Filename: file.c

10: struct **name** {

int **name**; <-[The member "name" and the tag in line 10 of "file.c" share the same name.]

:

};

char ***name**; <-[The variable "name" and the tag in line 10 of "file.c" share the same name.]

Rule 5.8

[The corresponding error indicated by Agile+ Relief]

- The variable and the function of the same name are declared in a different type and the storage class or are defined.

[Example 1]

Filename: file1.c

```
10:    short    *data;
```

Filename: file2.c

```
        extern    short    data;
```

<-[The variable declaration "data" and the variable definition in line 10 of "file1.c" are of different types: pointer and non-pointer (variable declaration: short int, variable definition: short int *).]

[Example 2]

Filename: file1.c

```
10:    int    func( char    *p ){ ~ }
```

Filename: file2.c

```
        int    func( long    *p );
```

<-[The 1st parameter "p" in the function declaration "func" and its corresponding parameter "p" in the function definition in line 10 of "file1.c" are of different types (long int *, char *).]

Rule 5.9

[The corresponding error indicated by Agile+ Relief]

- The same name as a static variable and the function is used.

[Example 1]

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

Rule 6.1

[The corresponding error indicated by Agile^{+} Relief]

- A bit field of type neither signed int nor unsigned int is declared.

[Example 1]

```
struct   TAG {
   unsigned   short   m1:2 ;          <-[The bitfield "m1" is declared with type unsigned short. The
                                         declaration should be made with signed int or unsigned int.]
   unsigned   int     m2: 2 ;
   signed     int     m3: 2 ;
};
```

Rule 6.2

[The corresponding error indicated by Agile+ Relief]

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

[Example 1]

```
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 ;
};
```

Rule 7.1

[The corresponding error indicated by Agile+ Relief]

- An octal constant other than 0 is used.

[Example 1]

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

Rule 7.2

[The Corresponding errors indicated by Agile^{+} Relief]

- The unsigned constant is not suffixed by "U".

[Example 1]

```
/* The size of int is 16 bits */
unsigned   int   x = 0xA123 ;          <-[The unsigned value "0xA123" is used without the suffix "U".]
```

Rule 7.3

[The corresponding error indicated by Agile+ Relief]

- "l" has been used by integer suffix or floating suffix.

[Example 1]

```
long  x = 32768l ;                <-["l" that may cause error easily has been used by "32768l".]
```

Rule 7.4

[The corresponding error indicated by Agile+ Relief]

- Not specified by const but referred string literal.

[Example 1]

```
void   func1( char *s );
void   func0( void ) {
       char *buf = "abc";                    <-[Not specified by const but referred string literal.]
                 :
       func1( "abc" );                       <-[Not specified by const but referred string literal.]

}
```

Rule 8.1

[The corresponding errors indicated by 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;
}
```

Rule 8.2

[The corresponding errors indicated by Agile+ Relief]

- There is a nameless parameter in the parameter declaration of the function.
- The function is defined in K&R format or does not specify parameters.
- No parameter is found in the function decelerator.
- A variable is declared without specifying the specifier.

[Example 1]

Code	Comment
int func(**int a, int**);	<-[In the parameter declaration of function "func", some parameters are declared with names while others are not.]

[Example 2]

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

[Example 3]

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

[Example 4]

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

Rule 8.3

[The corresponding errors indicated by Agile^{+} Relief]

- Type mismatch between function and parameter in the function declaration and definition.
- Name mismatch between parameter in the function declaration and definition.
- Type mismatch between variable in the variable declaration and definition.

[Example 1]

Filename: file1.c

```
10:    unsigned   func( char   *p ){ ~ }
```

Filename: file2.c

```
int   func( char   *p );
```

<-[The function declaration "func" and the corresponding function definition in line 10 of "file1.c" are of different types (function declaration type: int, function definition type: unsigned int).]

[Example 2]

Filename: file1.c

```
10:    int   func( char   *p ){ ~ }
```

Filename: file2.c

```
int   func( long   *p);
```

<-[The 1st parameter "p" in the function declaration "func" and its corresponding parameter "p" in the function definition in line 10 of "file1.c" are of different types (long int *, char *).]

[Example 3]

Filename: file1.c

```
10:    short   *data;
```

Filename: file2.c

```
extern   short   data;
```

<-[The variable declaration "data" and the variable definition in line 10 of "file1.c" are of different types: pointer and non-pointer (variable declaration: short int, variable definition: short int *).]

[Example 4]

Filename: file.c

```
int  func( int  data,        <-[The parameter name "data" is inconsistent with the
                                parameter name in the function definition "func" in line 3 of
                                "file.c". ]
           int  size );      <-[The parameter name "size" is inconsistent with the
                                parameter name in the function definition "func" in line 3 of
                                "file.c". ]
3: int  func( int  size, int  data )
{
:
}
```

Rule 8.4

[The corresponding error indicated by Agile+ Relief]

- Type mismatch between function and parameter in the function declaration and definition.
- Type mismatch between variable in the variable declaration and definition.

[Example 1]

Filename: file1.c

```
10:    unsigned   func( char   *p ){ ~ }
```

Filename: file2.c

```
       int   func( char   *p );
```

<-[The function declaration "func" and the corresponding function definition in line 10 of "file1.c" are of different types (function declaration type: int, function definition type: unsigned int).]

[Example 2]

Filename: file1.c

```
10:    short   *data;
```

Filename: file2.c

```
       extern   short   data;
```

<-[The variable declaration "data" and the variable definition in line 10 of "file1.c" are of different types: pointer and non-pointer (variable declaration: short int, variable definition: short int *).]

Rule 8.5

[The corresponding error indicated by Agile+ Relief]

- There is an external variable declaration or external function declaration outside of the header file.

[Example 1]

Filename: file.c

extern int x ;	<-[The external variable "x" is declared outside a header file.]

[Example 2]

Filename: file.c

int func(int in) ;	<-[The external function "func" is declared outside a header file.]

Rule 8.6

[The corresponding error indicated by Agile+ Relief]

- The function and the variable of the same name are defined.

[Example 1]

Filename: file1.c

```
10:   int   x;
```

Filename: file2.c

```
      int   x;          <-[The variable "x" has another definition in line 10 of "file1.c". One of them
                          should be deleted.]
```

[Example 2]

Filename: file1.c

```
10:   int   func();
```

Filename: file2.c

```
      int   func();     <-[The function "func" has another definition in line 10 of "file1.c". One of them
                          should be deleted.]
```

Rule 8.7

[The corresponding error indicated by Agile+ Relief]

- There exists a external function or an external variable used in only one file.

[Example 1]

```
Filename: file.c
int     x = 0 ;                              <-[The external variable "x" is used only in the file "file.c".]
void    x_add( void )   { x++; }
int     x_ref( void )   { return x; }
/* The variable "x" is only used in the file "file.c". */
```

Rule 8.8

[The corresponding error indicated by Agile+ Relief]

- There is a variable or a function using identical names but having different storage classes.

[Example 1]

Filename: file.c

```
10: extern   int   flag;
       :
    static   int   flag;                <-[The variable flag has a different storage class in line 10 of "file.c".]
```

Rule 8.9

[The corresponding error indicated by 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.*/
   }
```

Rule 8.10

[The corresponding error indicated by Agile+ Relief]

- There is inline function that does not specify the static storage class.

[Example 1]

```
inline    void func( ) { }                    <-[There is no static specification in the inline function "func".]
```

Rule 8.11

[The corresponding error indicated by 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.]
```

Rule 8.12

[The corresponding error indicated by Agile+ Relief]

- Some enumeration constants are assigned values while others are not.

[Example 1]

Code	Comment
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.]

Rule 8.13

[The corresponding error indicated by Agile+ Relief]

- In the parameter of pointer type or array type not updated, there is no const.

[Example 1]

```
void   func( int   p[ ], unsigned   int   n )      <-[The space indicated by parameter "p" of type "int [ ]" is
                                                    not updated and can be qualified with const. ]
{
    int   i;
    int   num = 0;
    for( i = 0; i < n; i++ ) {
        num += p[ i ];
    }
}
```

[Example 2]

```
int   func( int   *p )                  <-[The parameter "p" can use passing by value.]
{
   /*   *p is referenced in the function, but *p has not been undated or p has been referenced before
                                        this.*/
}
```

Rule 8.14

[The corresponding error indicated by Agile+ Relief]

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

[Example 1]

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

Rule 9.1

[The corresponding error indicated by 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.]
```

Rule 9.2

[The corresponding error indicated by Agile^{+} Relief]

- It is not enclosed with parentheses by initializing the structure, the union, and the array according to the structure.

[Example 1]

```
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.]
```

Rule 9.3

[The corresponding errors indicated by Agile+ Relief]

- Initialization is not done by initializing the structure, the union, and the array as for a number of elements.

[Example 1]

```
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.]
```

Rule 9.4

[The corresponding errors indicated by Agile+ Relief]

- The same element has been initialized more than onceby the array declaration, the structure declaration, and the union declaration.

[Example 1]

```
int ary1[ 4 ] = { [0] = 4 , [1] = 3 , [2] = 2 , [2] = 1 };        <-[An element of object "ary1" is initialized more than once.]

struct STR {
      int m1;
      int m2;
      int m3;
      int m4;
}
struct STR    str1 = {
.m1 = 1 , .m2 = 2 , .m3 = 3 , .m3 = 4 };        <-[An element of object "str1" is initialized more than once.]
```

Rule 9.5

[The corresponding errors indicated by Agile+ Relief]

- Tthe array size phrase in array declaration is omitted, and each Array elements has been initialized

[Example 1]

int ary1**[ ]** = { [0] = 3 , [1] = 2 , [2] = 1 }; <-[The size of the array "ary1" is not specified explicitly when designated initializers are used to initialize its elements.]

int ary2**[ ]** = { [0...15] = 1 }; <-[The size of the array "ary2" is not specified explicitly when designated initializers are used to initialize its elements.]

Rule 10.1

[The corresponding errors indicated by Agile^{+} Relief]

- A wrong type to the paragraph of the expression is used.

[Example 1]

```
int    i, data;
      :
i += (data < 1);                    <-[Operation of bool value has been used" i += ( data < 1) " .]
```

[Example 2]

```
unsigned   int     x;
double             d;
       :
x = d << 1 ;                        <-[In ANSI, the operator "d" of the bit operation "d << 1" must
                                    be integer type.]
```

Rule 10.2

[The corresponding errors indicated by Agile+ Relief]

- The char type is not used for a character value.
- There is an operation that performs underlying type conversion.

[Example 1]

```
char c;
    :
c = 1;
```

<-[The char "c" not specified as signed or unsigned is not treated as a character value.]

[Example 2]

```
void func( short s1, short s2)
{
  char sc;
  sc = s1 + s2;
```

<-["s1 + s2" in "sc = s1 + s2" is converted to a different type by an operation (underlying type before conversion: signed short int; underlying type after conversion: signed char).]

Rule 10.3

[The Corresponding errors indicated by Agile^{+} Relief]

- There is an implicit conversion that may lead to data loss resulting from differences in signs and types.

[Example 1]

```
unsigned char x = 0x8000;     <-[The initial value "0x80000" exceeds the bit width of "x" of
                                type unsigned char.]
struct T {
  unsigned char mem1;
  int mem2;
};
struct T y = { 65535,         <-[The initial value "65535" exceeds the bit width of "mem1" of
                                type unsigned char.]
   1
};
```

[Example 2]

```
Filename: file1.c
50: void func( char c ) { ... }

Filename: file2.c
   func( 256 );              <-[The bit width of the 1st argument "256" of the function
                               "func" exceeds the bit width of the corresponding parameter
                               "c" in the function definition in line 50 of "file1.c"(argument:
                               int , parameter: char ).]
```

[Example 3]

```
double data;
float s;
s = data ;                   <-[In the assignment expression "s = data", the size of the
                               expression on the right is larger than that of the expression on
                               the left, so the correct value might not be assigned (left : float ,
                               right : double).]
```

Rule 10.4

[The corresponding error indicated by Agile+ Relief]

- The type that is irrelevant to enumeration type was compared.

[Example 1]

enum COLOR{ RED, GREEN, BLUE } color ;	
enum ERROR_CODE{ SUCCESS, ERROR } code ;	
int x;	
if(**color == x**)	<-["color" is compared with irrelative type.(enum type : enum COLOR, compared type : int)]
if(**color != ERROR**)	<-["color" is compared with irrelative type.(enum type : enum COLOR, compared type : enum ERROR_CODE)]

Rule 10.5

[The corresponding error indicated by Agile+ Relief]

- It is a cast to improper essential types.

[Example 1]

```
typedef   _Bool   bool_t;
typedef   int     int32_t;
enum      enuma { ana };
enum      enumb { anb };
enum      enumb enb;
     :
( int32_t ) 3U ;          /* do not indicate */
( bool_t ) 3U ;           <-[The cast expression "( bool_t ) 3U" is casted between two inappropriate
                            essential types. (cast type: _bool, expression type: unsigned int)]
( char ) enb ;            /* do not indicate */
( enum enuma ) enb ;      <-[The cast expression "( enum enuma ) enb" is casted between two
                            inappropriate essential types. (cast type: enum enuma, expression type:
                            enum enumb)]
```

Rule 10.6

[The corresponding error indicated by Agile+ Relief]

- The result of the operation exceeds the type width of the expression.

[Example 1]

```
long long   x ;
int         i1 ;
      :
x1 = i1 + i2 ;
```

<-[The correct value of the + operation in "x1 = i1 + i2" might not be able to be obtained.]

Rule 10.7

[The corresponding error indicated by Agile+ Relief]

- When the expression of the integer type is operated, underlying type of the expression is changed.

[Example 1]

```
void func( short s1, short s2)
{
  long L;
  signed  char sc;
  L = s1 * s2;     <-["s1 * s2" in "L = s1 * s2" is converted to a different type by an operation (underlying type before conversion: signed short int, underlying type after conversion: signed char).]
  sc = s1 + s2;    <-["s1 + s2" in "sc = s1 + s2" is converted to a different type by an operation (underlying type before conversion: signed short int, underlying type after conversion: signed char).]
```

Rule 10.8

[The corresponding error indicated by Agile+ Relief]

- The integer type is not converted into same sign and a small type by using cast expression.
- The integer type is changed into the floating-point type by using cast expression.
- The floating-point type is not converted into a small type by using cast expression.
- The floating-point type is changed into the integer type by using cast expression.

[Example 1]

```
unsigned short a = 65432;
unsigned short b = 65432;
unsigned long x;
x = (unsigned long)(a+b);
```

<-[The cast expression "(unsigned long)(a + b)" casts an integer to a type that is not a smaller type of the same signed/unsigned status (cast type: unsigned long int, underlying type of the expression: unsigned short int).]

[Example 2]

```
long a = 3;
long b = 2;
double x;
x = (double)(a/b);
```

<-[The cast expression "(double)(a / b)" casts an integer to a floating point type (cast type: double, underlying type of the expression: signed long int).]

[Example 3]

```
float a = 1.0F/3;
float b = 1.0F/7;
long double x;
x = (long double)(a+b);
```

<-[The cast expression "(long double)(a + b)" casts a floating point type to a type that is not a smaller type (cast type: long double, expression: float).]

[Example 4]

float a = 1.0F/3;

float b = 1.0F/7;

long x;

x = **(long)(a+b)**; <-[The cast expression "(long)(a + b)" casts a floating point type to an integer type (cast type: long int, expression: float).]

Rule 11.1

[The Corresponding errors indicated by Agile+ Relief]

- One side of the cast expression is of type pointer to a function, the other side is of other types.

[Example 1]

```
int   (*fp)(void);
int   *ip;
ip = (int *)fp;
```

<-[The cast expression "(int *)fp" is a conversion involving a function pointer type and a non-integer type (cast type: int *, expression type: int (*) (void)).]

Rule 11.2

[The Corresponding errors indicated by Agile+ Relief]

- A non-pointer type is cast to a pointer type.
- A pointer type is cast to a non-pointer type.
- A non-void* pointer is cast to a non-void* pointer of a different type.

[Example 1]

```
int     adr , *p ;
p = (int *)adr ;
```
<-[The cast expression "(int *)adr" casts a non-pointer type to a pointer type (cast type: int *, expression type: int).]

[Example 2]

```
int     *p , adr ;
adr = (int)p ;
```
<-[The cast expression "(int)p" casts a pointer type to a non-pointer type (cast type: int, expression type: int *).]

[Example 3]

```
int     *adr1 ;
char     *adr2 ;
adr1 = (int*)adr2 ;
```
<-[The cast expression "(int*)adr2" casts the pointer type to a different pointer type (cast type: int *, expression type: char *).]

Rule 11.3

[The corresponding errors indicated by Agile+ Relief]

- Casting a pointer type to another pointer type.

[Example 1]

```
int      *adr1 ;
char     *adr2 ;
adr1 = (int*)adr2 ;
```

<-[The cast expression "(int*)adr2" casts a pointer type to a different pointer type (cast type: int *, expression type: char *).]

Rule 11.4

[The corresponding error indicated by Agile+ Relief]

- One side of the cast is of pointer type, the other side is of non-pointer type.
- Conditional expression that does comparison of pointer type and enumeration constant.

[Example 1]

```
int     adr , *p ;
p = (int *)adr ;                    <-[The cast expression "(int *)adr" casts a non-pointer type to
                                      a pointer type (cast type: int *, expression type: int).]
```

[Example 2]

```
enum ERROR_CODE {
    ERROR_NORMAL    = 0,
    ERROR_FILE      = 1,
    ERROR_DATA      = 2
};

void func( ) {
    char*   p;
        :
    if( p == ERROR_FILE ) {         <-[Compare pointer with an enumeration constant in the
                                      conditional expression "p == ERROR_FILE". (pointer type :
                                      char *)]
        :
    }
}
```

Rule 11.5

[The corresponding errors indicated by Agile+ Relief]

- The pointer of the void has been converted into other pointer types by the assignment expression.

[Example 1]

```
uint16_t * p16 ;
uint32_t * p32 ;
char_t * pstr ;
void *      p ;
        :
p    = p32 ;                /* This is not indicated */
p    = (void *) p16 ;       /* This is not indicated */
p32 = p ;                   <-[The assignment expression "p32 = p" assigns a void * pointer type to
                             "unsigned int *" type.]
p16 = ( uint16_t * ) p ;    <-[The assignment expression "p16 = ( uint16_t * ) p" assigns a void *
                             pointer type to "unsigned short int *" type.]
pstr = ( char_t * ) malloc(80);    /* This is not indicated */
```

Rule 11.6

[The corresponding errors indicated by Agile+ Relief]

- The cast expression of the integer type is used for the void pointer.

[Example 1]

```
int       i ;
void      *p ;
i = (int)p ;                    <-[The cast expression "(int)p " casts a pointer type to a
                                non-pointer type (cast type: int *, expression type: void *).]
```

Rule 11.7

[The corresponding errors indicated by Agile+ Relief]

- One side of the cast expression is of type pointer, the other side is of integer type.

[Example 1]

```
float    f ;
int      *p ;
p = (int *)f ;                <-[The cast expression "(int *)f" casts a non-pointer type to a
                              pointer type (cast type: int *, expression type: float).]
```

Rule 11.8

[The corresponding error indicated by 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*".]

Rule 11.9

[The corresponding error indicated by Agile+ Relief]

- Macro NULL is defined.

[Example 1]

```
#define   NULL   0                    <-[In the #define line, "NULL" is used as macro.]
```

Rule 12.1

[The Corresponding errors indicated by Agile+ Relief]

- It is better to add a pair of parentheses.
- The expression on both sides of the operators && or ||, is not a primary expression.

[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.]

[Example 6]

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.]

Rule 12.2

[The corresponding errors indicated by 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.]
```

Rule 12.3

[The corresponding error indicated by 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) {
```

Rule 12.4

[The corresponding error indicated by Agile+ Relief]

- The operational result cannot be represented by an unsigned type.

[Example 1]

```
#define   ABC   1U
unsigned   int   x;
x = ABC - 2;
if ( x < 10 )
```

<-[The value of the unsigned expression "1U - 2" cannot be represented by an unsigned type.]

[Example 2]

```
#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” is beyond the bit width of unsigned int type.]

Rule 12.5 (It added it by MISRA-C:2012 Amendment 1)

[The corresponding error indicated by Agile+ Relief]

- The size of the parameter of array type is obtained with sizeof.

[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 );
}
```

Rule 13.1

[The corresponding error indicated by Agile+ Relief]

- The expression that generates the side effect is in the initializer list.

[Example 1]

```
int     f1( void ) ;
void   f2( int x , int y ) {
    int a[ 2 ] = { x + y , x - y } ;   /* There is no side effect */
    int b[ 2 ] = { 0 , f1( ) };         <-[The initializer lists of "b" contains an expression "f1( )" which
                                           occurred side effects.]
```

Rule 13.2

[The corresponding error indicated by Agile+ Relief]

- The ANSI operation order of the expression is indefinite.
- The value of static variable and the external variable has been changed without doing exclusive control in the multi-thread function. (CreateThread and pthread_create function, etc.)

[Example 1]

i = j + j++ ; <-[The update timing of "j++" is not defined in ANSI, so the value of "j" cannot be guaranteed.]

i = ++i + j ; <-[The update timing of "++i" is not defined in ANSI, so the value of "i" cannot be guaranteed.]

[Example 2]

word_data = get_byte1( ) << 8 | get_byte2( ) ;

<-[The execution sequence of the function calls "get_byte" and "get_byte" is not specified in ANSI. The function calls should be made in separate lines.]

[Example 3]

Filename: file.c

x = Gx + func( ); <-[The timing of the function call "func" in line 100 of "file.c" updating the variable "Gx" is undefined in ANSI, so the value of "Gx" in the expression cannot be guaranteed.]

```
        :
int   func( void )
{
        :
100:    Gx += 2;
        :
}
```

[Example 4]

```
static int account_num;

void *thread_func( void *param ) {
        :
    account_num++;          <-[The variable "account_num" which used in multithread function is not
                            in exclusive control.]
        :
}

void func( void ) {
    int *thread;
        :
    pthread_create( &thread, NULL, (void *)thread_func, NULL );
        :
}
```

Rule 13.3

[The corresponding error indicated by Agile^{+} Relief]

- Mixed use of increment/decrement operators and other expressions.

[Example 1]

a = **b++**; <-[The result of "b++" is used in another operation.]
i = j + **j++**; <-[The result of "j++" is used in another operation.]
x++, y = 1; <-[The result of "x++" is used in another operation.]

Rule 13.4

[The corresponding errors indicated by Agile+ Relief]

- There is an assignment expression in the conditional expression.
- The assignment operation and comparing operation coexist.

[Example 1]

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".]

Rule 13.5

[The corresponding error indicated by Agile+ Relief]

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

[Example 1]

```
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.]
```

Rule 13.6

[The corresponding error indicated by Agile+ Relief]

- There is an update expression, a function call or a volatile variable within the sizeof operator.

[Example 1]

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

Rule 14.1

[The corresponding error indicated by 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.]
```

Rule 14.2

[The corresponding error indicated by Agile+ Relief]

- There is an expression unrelated to the loop control in the for-statement.
- Update a loop counter of the for-statement within the loop body.

[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.]
```

[Example 2]

```
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" ).]
```

Rule 14.3

[The corresponding errors indicated by Agile+ Relief]

- There is an eternally true/false conditional expression.
- The conditional expression may be eternally true/false, depending on the signs and the type size.

[Example 1]

```
if ( x == 1 && x == 2 )        <-[The condition expression "x == 1 && x == 2" is never true.]
if ( x > 1 && x < 0 )          <-[The condition expression "x > 1 && x < 0" is never true.]
```

[Example 2]

```
while( x * 10 - y > 0 ){       <-[The loop condition "x * 10 - y > 0" does not change.]
   :
}
/* The variables x and y are not updated in the while statement.*/
```

[Example 3]

```
char   c;
   :
if ( c == -1 ) {               <-[If char is unsigned, "-1" cannot be equal to "c".]
```

[Example 4]

```
short   s;
:
if ( s == 65536 )              <-[Comparison of "s" of type short int with the constant
                                 "65536" that exceeds the bit width of type short int is a
                                 mistake.]
```

Rule 14.4

[The corresponding errors indicated by Agile$^+$ Relief]

- Use of a non-_boolean value as a conditional expression.
- A conditional expression is being compared with 0.

[Example 1]

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

[Example 2]

```
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").]
```

[Example 3]

```
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.]
```

Rule 15.1

[The corresponding error indicated by Agile+ Relief]

- A goto statement is used.

[Example 1]

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

Rule 15.2

[The corresponding errors indicated by Agile+ Relief]

- The jumping destination is located before the goto statement.

[Example 1]

```
11: L1:;
        if ( a < 0 ) {
            goto L1;   <-[It jumps from the goto statement to the previous line.(jump position : line 11)]
        }
```

Rule 15.3

[The corresponding error indicated by Agile+ Relief]

- The jumping destination of goto statements do not contain goto statements.

[Example 1]

```
        if ( a < 0 ) {
            goto L2;<-[Jump out of the scope of goto statement.(jump position : line 22)]
        }
        else {
22: L2:;
        }
```

Rule 15.4

[The corresponding error indicated by 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;
    }
}
```

Rule 15.5

[The corresponding errors indicated by Agile+ Relief]

- More than one exit in the function.

[Example 1]

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

Rule 15.6

[The corresponding errors indicated by Agile^{+} Relief]

- Statements controlled by for, while, do-while and switch are not embraced by { } .

[Example 1]

```
while ( x > 0 )        <-[It might be better to add curly brackets { } to this while statement.]
    x--;
```

Rule 15.7

[The corresponding errors indicated by Agile+ Relief]

- The if and else if statements do not end with an else statement.

[Example 1]

```
if ( x > 0 ) {
    :
} else 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.]

Rule 16.1

[The corresponding errors indicated by Agile+ Relief]

- The case label or default label is not contained in the { } after the switch statement.
- The case label has no corresponding break statement.
- No default label has been found in the switch statement.
- A default label is placed before the case label.
- There is a comparing operator, logical operator and a constant in the conditional expression of switch statement.
- There is only one case label in 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;
  }
```

[Example 2]

```
    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++;
   }
```

[Example 3]

```
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.]
        :
}
```

Rule 16.2

[The corresponding errors indicated by 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;
  }
```

Rule 16.3

[The corresponding error indicated by Agile^{+} Relief]

- The case label has no corresponding break statement.

[Example 1]

```
    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++;
    }
```

Rule 16.4

[The corresponding errors indicated by Agile^{+} Relief]

- No default label has been found in the switch statement.

[Example 1]

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

Rule 16.5

[The corresponding errors indicated by Agile+ Relief]

- A default label is placed before the case label.

[Example 1]

```
switch ( x ) {
  case 1 :
        no++;
        break;
  case 2 :
  default :                    <-[The label default has been included before the label case
                               in a switch statement.]
        no += x;
        break;
  case 3 :
        no += x;
        no++;

}
```

Rule 16.6

[The corresponding errors indicated by Agile+ Relief]

- There are no two case labels or more in the switch statement.

[Example 1]

```
switch ( x ) {                    <-[There is no case label in this switch body.]
  default :
    i += x;
    break ;
}
```

[Example 2]

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

Rule 16.7

[The corresponding errors indicated by Agile+ Relief]

- There is a comparing operator, logical operator and a constant in the conditional expression of switch statement.
- There is only one case label in the switch statement.

[Example 1]

```
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.]
{ … }
```

[Example 2]

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

Rule 17.1

[The corresponding errors indicated by Agile+ Relief]

- The function, macro, tag, and type defined by <stdarg.h> are used.

[Example 1]

```
void func( va_list ap )            <-[The macro "va_list" is used.]
{
```

Rule 17.2

[The corresponding error indicated by Agile+ Relief]

- There is a recursive function.

[Example 1]

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

Rule 17.3

[The corresponding errors indicated by Agile+ Relief]

- There is no function declaration corresponding to the function call.

[Example 1]

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

/* Corresponding neither function declaration nor function definition are described before the above-mentioned description. */

Rule 17.4

[The corresponding error indicated by 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.]
}
```

Rule 17.5

[The corresponding error indicated by Agile+ Relief]

- It found argument of function call and parameter of declarations different array size.
- It found argument of function call for the same parameter type, but array type of parameter is unknown size.

[Example 1]

```
File: a.h
Line   1:   void   f2( int array[4] );
File: a.c
Line   1:   #include   "a.h"
       2:
       3:   void f1( ) {
       4:         int a[3] ;
       5:         f2 ( a ) ;     <-[The array size of the 1st argument "a" of the function "f2" is different from
                                    the corresponding parameter "array" in the function declaration in line 1
                                    of "a.h". (argument: int[3], parameter: int[4])
```

[Example 2]

```
File: a.h
Line   1:   void   f2( int   array[] );
       2:   void   f3( int   *ptr );

File: a.c
Line   1:   #include   "a.h"
       2:   void   f1( ) {
       3:         int   a[ 3 ] ;
       4:         f2 ( a ) ;   <-[The array size of the parameter "array" in the function declaration in line 1
                                  of "a.h" which corresponds to the 1st argument "a" of the function "f2" is not
                                  specified. (argument: int[3], parameter: int[])]
```

Rule 17.6

[The corresponding error indicated by Agile+ Relief]

- In the parameter of function definition, array declaration for which the size is specified with static is described.

[Example 1]

void func(int ary[**static** 10]) {	<-[There is a parameter of function "func" which is array described by keyword static.]
:	

Rule 17.7

[The corresponding errors indicated by Agile+ Relief]

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

[Example 1]

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

Rule 17.8

[The corresponding error indicated by Agile+ Relief]

- None of variable or parameter has been quoted once.

[Example 1]

```
void    func( int arg ){            <-[The value of parameter "arg" has not been quoted even once.]

    arg = 1;
    return;
}
```

Rule 18.1

[The corresponding errors indicated by Agile+ Relief]

- An arithmetic operation performed on a pointer of non-array type.
- Use of a non-zero array pointer as an array.
- Out of the range of the 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.]
:
}
```

[Example 3]

```
Filename: file.c
 1: int   data[10];
      :
10: data[10]=0;               <-[The subscript "10" of the array "data" exceeds the range of
                              the array (array declaration: Line 1 of "file.c").]
```

Rule 18.2

[The corresponding error indicated by 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.]
```

Rule 18.3

[The corresponding error indicated by Agile+ Relief]

- Make magnitude comparisons between the addresses of two different objects.

[Example 1]

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

Rule 18.4

[The corresponding error indicated by Agile$^+$ Relief]

- A pointer arithmetic operation is not in the array indexing form .

[Example 1]

```
int    data[ ]={ 1, 2, 3, 4, 5 };
int    *p = data;
return    *(p+1);                  <-[Pointer arithmetic operation "p+1" is not in the form of array
                                   indexing.]
```

Rule 18.5

[The corresponding error indicated by Agile+ Relief]

- The declaration contains more than 2 levels of indirect pointers.

[Example 1]

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

Rule 18.6

[The corresponding errors indicated by 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 ;
```

Rule 18.7

[The corresponding errors indicated by Agile+ Relief]

- In non-initialized array declaration except the parameters declaration, array length is omitted..
- The number of elements specified for an array is a negative number or *0*.

[Example 1]

```
File : body.c
  #include "head.h"
  int data[ 256 ] ;
File : head.h
  extern int data[ ] ;  <-[The length of an array has been omitted.]
```

[Example 2]

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

Rule 18.8

[The corresponding errors indicated by Agile+ Relief]

- Variable-length array is declared.

[Example 1]

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

Rule 19.1

[The corresponding errors indicated by Agile+ Relief]

- For a union variable, a member of it has been assigned, but another is referenced.

[Example 1]

```
    union {
      int      m1 ;
      char   m2 ;
    } x ;
10: x.m1 = 1 ;
    c = x.m2 ;                          <-[The union member "x.m1" whose value is assigned in line
                                        10 is referenced by another member "x.m2" of the union]
```

Rule 19.2

[The corresponding error indicated by Agile+ Relief]

- A union is used.

[Example 1]

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

Rule 20.1

[The corresponding error indicated by Agile+ Relief]

- Before the line #include, contents exist that are neither preprocessing directives nor comments.

[Example 1]

Filename: file.c

```
1: double   func( void )
  {   return   X; }
  #include "head.h"
```

<-[Statements (in line 1 of "file.c") which are not preprocessing directives exist before this #include line.]

Filename: head.h

```
 extern double X;
```

Rule 20.2

[The corresponding error indicated by Agile+ Relief]

- In the #include-specified file name, there are '(single quotation), \(currency character), "(double quote), /*(slash and asterisk) and multi-bytes characters(Chinese characters, etc).

[Example 1]

```
#include    "abc/*XY*/d.h"
```

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

Rule 20.3

[The corresponding errors indicated by 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 " ".]

Rule 20.4

[The corresponding errors indicated by Agile+ Relief]

- Macro of the same name as keyword.

[Example 1]

```
#define while ( EXP )   for( ; ( EXP ) ; );     <-[The macro name "while" is a keyword.]
#define unless( EXP )   if ( ! ( EXP ) )        /* The reserved word "If" in the substitution character
                                                   string is not pointed out. */
```

Rule 20.5

[The corresponding error indicated by Agile+ Relief]

- A #undef is used.

[Example 1]

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

Rule 20.6

[The corresponding error indicated by Agile+ Relief]

- The argument of the macro function begins with #.

[Example 1]

```
#define    F(b)    b
F(#error "memory error")
```

<-[A preprocessing directive is used in the argument "#error" of a macro function.]

Rule 20.7

[The corresponding error indicated by Agile+ Relief]

- The parameter in the replacement string of the macro is not embraced with parentheses ().
- The replacement string of the macro has not been embraced by () or {}.

[Example 1]

```
#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 ( ).]

[Example2]

```
#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 { }.]

Rule 20.8

[The corresponding error indicated by Agile+ Relief]

- Constants other than 0 and 1 are described in conditional expression of #if and #elif directive.

[Example 1]

```
#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
```

Rule 20.9

[The corresponding errors indicated by Agile+ Relief]

- An unspecified replacement string exists in the #if or #elif statement.
- An #undef is used on the undefined macro.

[Example 1]

/*The macro DEBUG is undefined, or no replacement string has been specified for it*/

#if DEBUG == 1	<-[The replacement string of the macro "DEBUG" in an #if or #elif statement is not specified.]
#elif DEBUG == 2	<-[The replacement string of the macro "DEBUG" in an #if or #elif statement is not specified.]

[Example 2]

/* There are no definitions of macro ABC,XYZ*/

#if ABC	<-["ABC" might have been intended to be appended with defined.]
#elif XYZ	<-["XYZ" might have been intended to be appended with defined.]

[Example 3]

/* There are no definition of macro BBB*/

#undef BBB	<-[#undef is used on the undefined macro "BBB".]

Rule 20.10

[The corresponding error indicated by 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.]
```

Rule 20.11

[The corresponding error indicated by Agile+ Relief]

- The #operator and the ##operator coexist in the macro function.

[Example 1]

```
#define    AAA(x,y)    x###y
```

<-[The operators # and ## are used together in the definition of the macro function "AAA".]

Rule 20.12

[The corresponding error indicated by Agile+ Relief]

- The replacement string of the macro definition contains a "#" or a "##".

[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.]
```

Rule 20.13

[The corresponding error indicated by Agile+ Relief]

- In ANSI specification, some lines have started with the undefined #.

[Example 1]

#asm	<-[The undefined pre-processing command "#asm" cannot be described in ANSI.]

Rule 20.14

[The corresponding error indicated by 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.]

Rule 21.1

[The corresponding errors indicated by 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.]
```

Rule 21.2

[The corresponding error indicated by Agile^{+} Relief]

- An identifier using a name identical to that of the reserved identifier is used in the declaration and definition.

The variable and function using identical names registered under the label [RESERVED_IDENTIFIER] and [RESERVED_LIBRARY_IDENTIFIER] of the identifier file are checked as an object. For more information regarding the registration of an identifier file, see the -F option in the "Command Manual".

[Example 1]

```
int     errno ;                    <-[The name "errno" is the same as an external name used in
                                   the system.]
```

Rule 21.3

[The corresponding error indicated by 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.]
```

Rule 21.4

[The corresponding error indicated by Agile+ Relief]

- File setjmp.h is included by the #include statement.
- The longjmp and macro setjmp functions are used.

[Example 1]

```
#include    <setjmp.h>                    <-[The file "setjmp.h" is included and should be confirmed.]
static    jmp_buf    parse_error;
   :
longjmp( parse_error, 1 );                <-[The function "longjmp" is used.]
```

Rule 21.5

[The corresponding errors indicated by Agile+ Relief]

- File signal.h has been included by the #include statement.
- The following functions and macros are used:
  signal
  SIG_DFL
  SIG_IGN
  SIG_ERR
  SIGABRT
  SIGFPE
  SIGILL
  SIGINT
  SIGSEGV
  SIGTERM

[Example 1]

```
#include   <signal.h>                 <-[The file "signal.h" is included and should be confirmed.]
extern   void   sigint_hnd( void );
:
signal( SIGINT, sigint_hnd );         <-[The function "signal" is used.]
                                      <-[The macro "SIGINT" is used.]
```

Rule 21.6

[The corresponding errors indicated by Agile+ Relief]

- File stdio.h is included by the #include statement.
- The following functions are used:
  fgetpos
  fopen
  ftell
  gets
  perror
  remove
  rename
  ungetc

[Example 1]

```
#include    <stdio.h>                    <-[The file "locale.h" is included and should be confirmed.]
FILE    *fp;
     :
fp = fopen( "abc.txt", "r");             <-[The function "fopen" is used.]
```

Rule 21.7

[The corresponding error indicated by Agile+ Relief]

- The atof, atoi, atol and atoll functions are used.

[Example 1]

```
#include    <stdlib.h>
int    i;
char    *str = "123";
i = atoi( str );                                    <-[The function "atoi" is used]
```

Rule 21.8

[The corresponding error indicated by Agile+ Relief]

- The abort, exit, getenv, and system functions are used.

[Example 1]

```
#include    <stdlib.h>
:
if ( j < 0 ) {
abort( );                                  <-[The function "abort" is used.]
}
```

Rule 21.9

[The corresponding error indicated by Agile+ Relief]

- The bsearch and qsort functions are used.

[Example 1]

```
#include    <stdlib.h>
:
qsort( d ,d_cnt, sizeof( d[0] ),sort_int );     <-[The function "qsort" is used.]
```

Rule 21.10

[The corresponding error indicated by Agile+ Relief]

- The following functions are used:
  clock
  difftime
  mktime
  time
  asctime
  ctime
  gmtime
  localtime
  strftime

[Example 1]

```
#include    <time.h>
:
time_t    now:
time( &now );                              <-[The function "time" is used.]
```

Rule 21.11

[The corresponding errors indicated by Agile+ Relief]

- Standard C library <tgmath.h> is included by the #include statement.

[Example 1]

```
#include    <tgmath.h>                    <-[The file "tgmath.h" is included and should be confirmed.]
```

Rule 21.12

[The corresponding error indicated by Agile+ Relief]

- The following functions and macros are used:
  feclearexcept
  fegetexceptflag
  feraiseexcept
  fesetexceptflag
  fetestexcept
  FE_INEXACT
  FE_DIVBYZERO
  FE_UNDERFLOW
  FE_OVERFLOW
  FE_INVALID
  FE_ALL_EXCEPT

[Example 1]

```
fetestexcept( FE_OVERFLOW );        <-[The function "fetestexcept" is used.]
                                    <-[The macro "FE_OVERFLOW" is used.]
```

Rule 21.13 (It added it by MISRA-C:2012 Amendment 1)

[The corresponding errors indicated by Agile+ Relief]

- The parameters of the function that does the determination and the translation of the character is not unsigned char type. (isalpha, isdigit, and isalnum function, etc.)

[Example 1]

```
unsigned char uc ;
signed     char sc ;
/* It is expressible by the unsigned char type */
if ( isalpha( uc ) )   {
          :
/* There is a possibility including the value not expressible by the unsigned char type */
if ( isalpha( sc ) ) {      <-[The type of the 1st argument sc of the function "isalpha" should be
                               unsigned char. (argument type : signed char )]
          :
/* Value not expressible by unsigned char type */
if ( isalpha( 256 ) ) {     <-[The type of the 1st argument 256 of the function "isalpha" should be
                               unsigned char. (argument type : int )]
        :
```

Rule 21.14 (It added it by MISRA-C:2012 Amendment 1)

[The corresponding errors indicated by Agile+ Relief]

- A "char *" pointer or "char array" type argument to a function like memcmp.

[Example 1]

```
void func ( char* str1 , unsigned char* ustr2 , unsigned char* ustr3 ) {
   char ary1[ 3 ] ;
   char ary2[ 3 ] ;
   if( memcmp( ary1,ary2,3 ) == 0 )   <-[It is not desirable to try to compare string by function
                                         "memcmp".]
         :
   if( memcmp( str1,"ABC",3 ) == 0 ) <-[It is not desirable to try to compare string by function
                                         "memcmp".]
         :
   if( memcmp( str1,ustr2,3 ) == 0 )   <-[It is not desirable to try to compare string by function
                                         "memcmp".]
         :
   if( memcmp( ustr2,ustr3,3 ) == 0 )   /* both arguments are not the string comparisons because of
                                         "Unsigned char *" type, it not points it out. */
         :
}
```

Rule 21.15 (It added it by MISRA-C:2012 Amendment 1)

[The corresponding errors indicated by Agile+ Relief]

- The comparison and the copy are done by memcmp and memcpy function, etc. between objects of a different type.

[Example 1]

```
void TableCheck( int *pATable ,char *pBTable){
    char ICTable[ 12 ] ;
    int   IDTable[ 12 ] ;
          :
    if( memcmp( IDTable , pBTable , 12 ) )   <-[The type of the 1st argument "IDTable" and the 2nd argument "pBTable" of the function "memcmp" are different. (1st argument: int[12], 2nd argument: char *)]

    if( memcmp( pBTable ,"ID" , 2 ) )    /* The confirmation of the truncation is not pointed out. */
```

Rule 21.16 (It added it by MISRA-C:2012 Amendment 1)

[The corresponding error indicated by Agile+ Relief]

- Pointer type to the structure and the union is specified for the memcmp function.

[Example 1]

```
struct STRUCT {
    unsigned char     a;
    unsigned char     b;
    unsigned int      c;
};
int func(struct STRUCT* s1, struct STRUCT* s2)
{
    if( memcmp( (char*)s1, (char*)s2, sizeof( struct STRUCT)) == 0 )
    <-[The function "memcmp" should not be used to compare structures or unions.]
    :
}
```

Rule 21.17 (It added it by MISRA-C:2012 Amendment 1)

[The corresponding error indicated by Agile+ Relief]

- The size of the copy source exceeds the one of the copy target in the strcpy and the strcat function.

[Example 1]

```
char    a[10], b[20];
    :
strcpy( a , b ) ;        <-[The size of the copy source of the copy expression "strcpy(a, b) " might
                         exceed that of the copy target "a" (size of copy target:10, size of copy source:
                         20).]
```

Rule 21.18 (It added it by MISRA-C:2012 Amendment 1)

[The corresponding error indicated by Agile+ Relief]

- The size at the connection destination and the connected size are equal in the strncat function.
- The area specified by the following functions has not been acquired.
  - memchr
  - memcmp
  - memcpy
  - memmove
  - memset
  - strncat
  - strncmp
  - strncpy
  - strxfrm

[Example 1]

```
char a[5];
strncpy( a, "abc", sizeof(a) );
char b[] = "def" ;
strncat( a, b, sizeof(a) );        <-["strncat(a, b, sizeof(a))" might access an address which exceeds the
                                        range of space "a".(space: 5, size: 5)]
```

[Example 2]

```
int   x ;
struct { int m1; int m2 ; } sa ;
        :
memset( &x , 0 , sizeof(sa) ) ;    <-["memset(&x, 0, sizeof(sa))" might access an address which
                                   exceeds the range of space "x".(space:4, size:8)]
```

Rule 21.19 (It added it by MISRA-C:2012 Amendment 1)

[The corresponding error indicated by Agile+ Relief]

- The getenv function, the localeconv function, the setlocale function, and the strerror function are used.

[Example 1]

```
char *path = NULL;
buf = getenv( "PATH" );                <-[The function "getenv" is used.]
```

Rule 21.20 (It added it by MISRA-C:2012 Amendment 1)

[The corresponding error indicated by Agile+ Relief]

- The following functions are used.
  - asctime
  - ctime
  - gmtime
  - localtime
  - localeconv
  - getenv
  - setlocale
  - strerror

[Example 1]

```
const char *res;
res = setlocale( LC_ALL, 0 );                <-[The function "setlocale" is used.]
```

Rule 22.1

[The corresponding error indicated by Agile+ Relief]

- The resource acquired in the standard library function has not been liberated.

[Example 1]

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

Rule 22.2

[The corresponding error indicated by Agile+ Relief]

- Refer to the resource collected by the standard library function.
- It collects it excluding the resource acquired in the standard library function.

[Example 1]

```
    if ( mode == 0 ) {
10:     free( p ) ;
    }
    *p = 0 ;                <-[The variable "p" the resources of which are reclaimed in
                            line 10, might be referenced.]
```

[Example 2]

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

Rule 22.3

[The corresponding errors indicated by Agile+ Relief]

- The file is doubly opened in a different mode.

[Example 1]

```
10: FILE* fp1 = fopen("tmp.txt", "r") ;
      FILE* fp2 = fopen("tmp.txt", "w") ;   <-[File "tmp.txt" maybe has already been opened with "r"
                                              mode in line 10.]
```

Rule 22.4

[The corresponding errors indicated by Agile$^+$ Relief]

- It writes it in the file that opens in the reading mode.

[Example 1]

```
        char str[1024] ;
10: FILE* fp = fopen("tmp.txt", "r") ;
              :
        fprintf( fp, "%s\n", str ) ;        <-[Maybe write into the file "tmp.txt" which is opened with
                                            read-only mode.(file open line : line 10)]
```

Rule 22.5

[The corresponding errors indicated by Agile+ Relief]

- Refer to the file pointer reversely.

[Example 1]

```
FILE* fp = fopen("tmp.txt", "r");
       :
fp->pos = 0 ;              <-["*fp" is an expression of FILE pointer which is back referred.]
```

Rule 22.6

[The corresponding errors indicated by Agile+ Relief]

- Refer to the resource collected by the standard library function.

[Example 1]

```
    if ( mode == 0 ) {
10:     fclose ( fp ) ;
    }
    fprintf( fp, "end" ) ;        <-[The variable "fp" the resources of which are reclaimed in
                                  line 10, might be referenced.]
```

Rule 22.7 (It added it by MISRA-C:2012 Amendment 1)

[The corresponding errors indicated by Agile+ Relief]

- A macro EOF is used.

[Example 1]

```
char c;
c = (char)getchar();
if( EOF != (int)c ) {      <-[The macro "EOF" is used.]
}
```

Rule 22.8 (It added it by MISRA-C:2012 Amendment 1)

[The corresponding errors indicated by Agile^{+} Relief]

- A variable errno is used.

[Example 1]

```
if( 0 == errno )     <-[The variable "errno" is used.]
}
```

Rule 22.9 (It added it by MISRA-C:2012 Amendment 1)

It is the same comment as Rule 22.8, please refer to Rule 22.8 for detail.

Rule 22.10 (It added it by MISRA-C:2012 Amendment 1)

It is the same comment as Rule 22.8, please refer to Rule 22.8 for detail.

3.4 MISRA-C++ V1

For the following Rules, because the Rules for documentation, Agile+ Relief cannot check the behaviors when parsing the source code:

- Rule 0-3-1
- Rule 0-4-1
- Rule 0-4-2
- Rule 0-4-3
- Rule 1-0-2
- Rule 1-0-3
- Rule 2-2-1
- Rule 7-4-1
- Rule 9-6-1
- Rule 15-0-1
- Rule 16-6-1
- Rule 17-0-4

Also, since the following Rule concerns the coding style of the source code, Agile+ Relief cannot check it:

- Rule 0-1-5
- Rule 0-1-6
- Rule 0-1-8
- Rule 2-5-1
- Rule 2-7-2
- Rule 2-7-3
- Rule 2-10-1
- Rule 2-13-5
- Rule 3-1-2
- Rule 3-2-1
- Rule 3-2-2
- Rule 3-9-1
- Rule 4-5-2
- Rule 4-5-3
- Rule 4-10-1
- Rule 4-10-2
- Rule 5-0-3
- Rule 5-0-7

- Rule 5-0-8
- Rule 5-0-9
- Rule 5-0-11
- Rule 5-0-12
- Rule 5-17-1
- Rule 6-4-8
- Rule 6-5-4
- Rule 6-5-6
- Rule 7-1-1
- Rule 7-3-4
- Rule 7-3-5
- Rule 7-4-2
- Rule 7-5-3
- Rule 9-3-1
- Rule 10-1-2
- Rule 10-1-3
- Rule 10-2-1
- Rule 10-3-1
- Rule 10-3-2
- Rule 10-3-3
- Rule 12-1-2
- Rule 12-8-2
- Rule 14-5-1
- Rule 14-6-1
- Rule 14-6-2
- Rule 14-7-1
- Rule 14-7-2
- Rule 14-7-3
- Rule 14-8-1
- Rule 14-8-2
- Rule 15-0-3
- Rule 15-1-3
- Rule 15-3-3
- Rule 15-3-4

Rule 0-1-1

[The corresponding error indicated by Agile^{+} Relief]

- There is an unexecuted statement.

[Example 1]

```
void foo ( ) {
return;
x = y + z;                          <-[This will not be executed.]
}
```

Rule 0-1-2

[The corresponding error indicated by Agile+ Relief]

- Some statements are not executed because of the existence of invalid conditions.

[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 }

Rule 0-1-3

[The corresponding error indicated by Agile+ Relief]

- An unused variable or parameter exists.

[Example 1]

```
void foo ( ){
int   x;                                   <-[The variable "x" is not used.]
return;
}
```

[Example 2]

```
void func (   int   p ,   int   q   ){     <-[The parameter "p" is not used.]
      int n = q;
      return;
}
```

Rule 0-1-4

[The corresponding error indicated by Agile+ Relief]

- None of variable or parameter has been quoted once.

[Example 1]

```
void foo ( ){
   int   x;                <-[The value of variable "x" has not been quoted even once.]
    x = 1;
    return;
}
```

[Example 2]

```
void func( int arg ){      <-[The value of parameter "arg" has not been quoted even once.]

   arg = 1;
   return;
}
```

Rule 0-1-5

Agile+ Relief will not check if this rule has been violated.

Rule 0-1-6

Agile+ Relief will not check if this rule has been violated.

Rule 0-1-7

[The corresponding errors indicated by Agile+ Relief]

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

[Example 1]

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

Rule 0-1-8

Agile[+] Relief will not check if this rule has been violated.

Rule 0-1-9

[The corresponding error indicated by Agile+ Relief]

- An empty statement without meaning exists.

[Example 1]

```
int x ;    ;                          <-[It may be an useless empty statement.]
switch( y ){
case 1: ;                             <-[It may be an useless empty statement.]
```

Rule 0-1-10

[The corresponding error indicated by Agile+ Relief]

- An unused static function exists.

[Example 1]

```
static int func( ){                    <-[The static function "func" is not used.]
    int x = 32;
    return x;
}
/* The function func is not used. */
```

Rule 0-1-11

[The corresponding error indicated by Agile+ Relief]

- An unused parameter exists.

[Example 1]

```
void func(  int  x , int  y  ) {          <-[The parameter "x" is not used.]
        cls( y );
        return;
}
```

Rule 0-1-12

It is the same comment as Rule 0-1-11, please refer to Rule 0-1-11 for detail.

Rule 0-2-1

[The corresponding errors indicated by Agile+ Relief]

- For a union variable, a member of it has been assigned, but another is referenced.

[Example 1]

```
   union {
     int      m1 ;
     char   m2 ;
   } x ;
10: x.m1 = 1 ;
   c = x.m2 ;                          <-[The union member "x.m1" whose value is assigned in line
                                         10 is referenced by another member "x.m2" of the union.]
```

Rule 0-3-1

Agile+ Relief will not check if the rule has been violated.

Rule 0-3-2

[The corresponding errors indicated by Agile+ Relief]

- An abnormal value is returned by the function without having been checked.
- Error code is not determined after the function is called.

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

[Example 1]

```
        char *p;
        p = malloc(sizeof(char)* 100);
19:     *p = '\0':              <-[At line 19, a value may be assigned to the variable *p of 0
                                address, or 0 address may be quoted.]
```

Rule 0-4-1

Agile$^+$ Relief will not check if this rule has been violated.

Rule 0-4-2

Agile+ Relief will not check if this rule has been violated.

Rule 0-4-3

Agile+ Relief will not check if this rule has been violated.

Rule 1-0-1

[The corresponding error indicated by Agile+ Relief]

- There are descriptions not compliant with C++ 2003(ISO/ICE 14882:2003).

[Example 1]

```
int func( int x, );                <-[In ANSI, ". . . " is not permitted to be omitted.]
```

Rule 1-0-2

Agile+ Relief will not check if this rule has been violated.

Rule 1-0-3

Agile+ Relief will not check if this rule has been violated.

Rule 2-2-1

Agile+ Relief will not check if this rule has been violated.

Rule 2-3-1

[The corresponding error indicated by Agile+ Relief]

- A trigraph sequence is used.

[Example 1]

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

Rule 2-5-1

Agile[+] Relief will not check if this rule has been violated.

Rule 2-7-1

[The corresponding error indicated by Agile+ Relief]

- A nested comment exists.

[Example 1]

```
/* if (argc==0)
        /* return -1 */                    <-[A nested comment is used.]
*/
```

Rule 2-7-2

Agile[+] Relief will not check if this rule has been violated.

Rule 2-7-3

Agile+ Relief will not check if this rule has been violated.

Rule 2-10-1

Agile+ Relief will not check if this rule has been violated.

Rule 2-10-2

[The corresponding error indicated by Agile+ Relief]

- An identifier with a name identical to that of another identifier within the external scope has been declared within the internal effective scope.

[Example 1]

```
    Filename: file.cc
10: unsigned char *cp;
    void func( )
    {
        unsigned char *cp;      <-[The declaration of the variable "cp" uses the same name
                                  as the typedef in line 10 of "file.cc", outside the function.]
    }
```

[Example 2]

```
    Filename: file.cc
10: struct s { short a, b; } x;
    void func( )
    {
        struct s { int a, b; } y;      <-[The declaration of the tag "s" uses the same name as "s" in
                                         line 10 of "file.cc", outside the function.]
    }
```

Rule 2-10-3

[The corresponding error indicated by Agile+ Relief]

- Duplicate typedef name.

[Example 1]

Filename: file.cc

```
1:  typedef  char  *cp;
    void  func( )
    {
4:      char  *cp;               <-[The declaration of the variable "cp" uses the same name
                                    as the typedef in line 1 of "file.cc", outside the function.]
        if( data == 0 ){
        typedef char  *cp;       <-[The declaration of the typedef "cp" uses the same name as
                                    the variable in line 4 of "file.cc".]
        }
    }
```

Rule 2-10-4

[The corresponding error indicated by Agile+ Relief]

- Duplicate class name, struct name, union name and enum name.

[Example 1]

```
Filename: file.cc
 1:    class   CLS{
        };
:
        void   func()
       {
          class   CLS{        <-[The declaration of the tag "CLS" uses the same name as
                                 the "CLS" in line 1 of "file.cc", outside the function.]
          };
       }
```

[Example 2]

```
Filename: file.cc
       void   func()
       {
 3:       class   CLS{
          };
       :
          if( data == 0 ){
             class   CLS{     <-[The declaration of the tag "CLS" uses the same name as
                                 the "CLS" in line 3 of "file.cc".]
             };
          }
       }
```

Rule 2-10-5

[The corresponding error indicated by Agile+ Relief]

- Identifiers with the same name have been defined outside and inside the function.

[Example 1]

File name: file.cc

```
1:  typedef char *cp;
    void func ()
    {
            char *cp;        <-[The name of declaration of the variable "cp" is identical to
                               the name of typedef outside the function (line 1 of "file.cc").]
    }
```

[Example 2]

File name: file.cc

```
    void func1()
    {
        char   c;            <-[The name of declaration of the variable "c" is identical to
                               the name of the variable outside the function (line 6 of
                               "file.cc").]
    }
    void func2( ){
6:  extern char   c;
    }
```

Rule 2-10-6

[The corresponding error indicated by Agile^{+} Relief]

- Identifiers with the same name have been defined outside and inside the function.
- A name that is duplicated with the external name used by the system has been used.

[Example 1]

File name: file.cc

```
10:   typedef char *cp;
      void func( ){
            char *cp;                <-[The name of declaration of the variable "cp" is identical to
                                     the name of typedef outside the function (line 10 of "file.cc").]
      }
```

[Example 2]

```
      void *FILE( unsigned int size ) {    <-["FILE" is duplicated with the external name used by the
                                           system.]
            :
      }
```

Rule 2-13-1

[The corresponding error indicated by 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.]

Rule 2-13-2

[The corresponding error indicated by Agile+ Relief]

- An octonary escape sequence has been used.
- A number beginning with 0 exists.

[Example 1]

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

[Example 2]

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

Rule 2-13-3

[The Corresponding errors indicated by Agile+ Relief]

- The unsigned constant is not suffixed by "U".

[Example 1]

```
/* The size of int is 16 bits */
unsigned   int   x = 0xA123 ;          <-[The unsigned value "0xA123" is used without the suffix "U".]
```

Rule 2-13-4

[The corresponding error indicated by Agile+ Relief]

- The lowercase l has been used by integer suffix or floating suffix.

[Example 1]

```
long   x = 32768l ;                <-["l" that may cause error easily has been used by "32768l".]
long   double   y = 3.14l ;        <-["l" that may cause error easily has been used by"3.14l".]
```

Rule 2-13-5

Agile+ Relief will not check if this rule has been violated.

Rule 3-1-1

[The corresponding error indicated by Agile+ Relief]

- A variable or function has been defined in the include file.

[Example 1]

File name: head.h

```
int    x ;                 <-[Space has been allocated in the head file. (Variable “x”)]
int func( )                <-[Space has been allocated in the head file. ( function “func”)]
{
   return x;
}
```

File name: file.cc

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

Rule 3-1-2

Agile+ Relief will not check if this rule has been violated.

Rule 3-1-3

[The corresponding error indicated by Agile+ Relief]

- The array is declared without specifying its length.

[Example 1]

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

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

Rule 3-2-1

Agile[+] Relief will not check if this rule has been violated.

Rule 3-2-2

Agile+ Relief will not check if this rule has been violated.

Rule 3-2-3

[The corresponding error indicated by Agile+ Relief]

- There is an external variable declaration or external function declaration outside of the header file.

[Example 1]

Filename: file.cc

extern int x ;	<-[The external variable "x" is declared outside a header file.]

[Example 2]

Filename: file.cc

int func(int in) ;	<-[The external function "func" is declared outside a header file.]

Rule 3-2-4

[The corresponding error indicated by Agile+ Relief]

- The external variables with the same name of a type have been defined in a file.

[Example 1]

File name: file.cc

```
10:  int   x =1;
        :
     int   x;
```

<-[The variable “x” is also defined at line 10 of “file.cc”. It is recommended to delete the part without initial value.]

Rule 3-3-1

[The corresponding error indicated by Agile+ Relief]

- External variable or external function has been declared outside the head file.
- A function that is not declared has been defined.

[Example 1]

File name: file.cc

```
extern int     x ;                    <-[The external variable “x” has been declared outside the head file.]
```

[Example 2]

File name: file.cc

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

Rule 3-3-2

[The corresponding error indicated by Agile^{+} Relief]

- There is a variable or a function using identical names but having different storage classes.

[Example 1]

Filename: file.cc

```
10:   extern   int   flag;
        :
      static   int   flag;            <-[The variable flag has a different storage class in line 10 of
                                      "file.cc".]
```

Rule 3-4-1

[The corresponding error indicated by Agile+ Relief]

- There is a global static variable used only in one function.

[Example 1]

Filename: file.cc

```
13:     static int   yy ;          <-[The external variable "yy" that can be used in multiple
                                     functions was referenced only in the function "func" in line 23
                                     of "file.cc”. An internal static variable could be used instead.]
           :
23:     void func( int xx )
        {
            /* static variable yy is only used in this function.*/
```

Rule 3-9-1

Agile+ Relief will not check if this rule has been violated.

Rule 3-9-2

[The corresponding error indicated by Agile+ Relief]

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

[Example 1]

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

Rule 3-9-3

[The corresponding errors indicated by Agile+ Relief]

- A member of floating-point type and members of other types exist in a union at the same time.
- Bitwise operation for floating-point or pointer type.
- Cast a pointer of floating-point to pointer of other types.

[Example 1]

```
union   UN1
{
    double   a;
    unsigned   long   b[2];
};
```

<-[There is a member in union "UN1" whose type is different from that of float-type member "a".]

[Example 2]

```
double     d;
x = d << 1 ;
```

<-[In ANSI, the operand "d" of the bit operation "d << 1" must be of an integer type.]

[Example 3]

```
int       *adr1 ;
float     *adr2 ;
adr1 = (int*)adr2 ;
```

<-[The cast expression "(int*)adr2" casts the pointer type to a different pointer type. (cast type: char *, expression: int *)]

Rule 4-5-1

[The corresponding error indicated by Agile+ Relief]

- Operation of bool value or bool type variable has been performed.

[Example 1]

```
int i, data;
     :
i += (data < 1);                     <-[Operation of bool value has been used" i += ( data < 1) " .]
```

Rule 4-5-2

Agile+ Relief will not check if this rule has been violated.

Rule 4-5-3

Agile⁺ Relief will not check if this rule has been violated.

Rule 4-10-1

Agile+ Relief will not check if this rule has been violated.

Rule 4-10-2

Agile+ Relief will not check if this rule has been violated.

Rule 5-0-1

[The corresponding error indicated by Agile+ Relief]

- The ANSI operation order of the expression is indefinite.

[Example 1]

i = j + j++ ; <-[The update timing of "j++" is not defined in ANSI, so the value of "j" cannot be guaranteed.]

i = ++i + j ; <-[The update timing of "++i" is not defined in ANSI, so the value of "i" cannot be guaranteed.]

[Example 2]

word_data = get_byte( ) << 8 | get_byte( ) ;

<-[The execution sequence of the function calls "get_byte" and "get_byte" is not specified in ANSI. The function calls should be made in separate lines.]

Rule 5-0-2

[The Corresponding errors indicated by 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.]

Rule 5-0-3

Agile+ Relief will not check if this rule has been violated.

Rule 5-0-4

[The corresponding error indicated by Agile+ Relief]

- The signed value and unsigned value are compared.

[Example 1]

```
signed   char   sc;
unsigned char   uc;
   :
if ( sc == uc )
```

<-[Unsigned "uc" and signed "sc" are coexisting in the comparison expression "sc == uc".]

Rule 5-0-5

[The corresponding error indicated by Agile+ Relief]

- The result of integer operation (+,-,*,/) was assigned to the floating type, or it is compared with the floating type.

[Example 1]

```
double    d, f ;
int       x, y;
          :
f = x + y;            <-["f = x + y" may not get the correct value. It is recommended
                        to use "f = (double)x + y".]
if ( d < x + y )      <-["d < x + y" may not get the correct value. It is
                        recommended to use "d < (double)x + y".]
```

[Example 2]

```
double    f;
int       x, y;
          :
f = x * y;            <-["f = x * y" cannot get the correct value. Please use "f =
                        (double)x * y".]
if (f < x * y)        <-["f < x * y" cannot get the correct value. Please use "f <
                        (double)x * y".]
```

Rule 5-0-6

[The corresponding errors indicated by Agile+ Relief]

- There is an implicit conversion that may lead to data loss resulting from differences in signs, type size and types.

[Example 1]

```
int     i;
i = 0xffffffffffffffff;
```

<-[“0xffffffffffffffff”that exceeds the range that can be shown by constant cannot be used.]

[Example 2]

```
long data;
short s;
s = data ;
```

<-[The size at the right of assignment expression “s = data” is too large, value may not be assigned correctly. (left : short , right : long)]

[Example 3]

Filename: file.cc

```
10:     void func( signed char sc ) { ~ }
        :
        int i;
        :
        func( i );
```

<-[In the first actual parameter "i" of the function "func" and its correspondent virtual parameter "sc" (Function definition of line 10 of "file.cc"), the size of actual parameter is large, value may not be transferred correctly. (Actual parameter: int, Virtual parameter: signed char)"]

Rule 5-0-7

Agile⁺ Relief will not check if this rule has been violated.

Rule 5-0-8

Agile+ Relief will not check if this rule has been violated.

Rule 5-0-9

Agile⁺ Relief will not check if this rule has been violated.

Rule 5-0-10

[The corresponding errors indicated by 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)".]
```

Rule 5-0-11

Agile⁺ Relief will not check if this rule has been violated.

Rule 5-0-12

Agile+ Relief will not check if this rule has been violated.

Rule 5-0-13

[The corresponding errors indicated by Agile+ Relief]

- There is an assignment expression in the conditional expression.
- The assignment operation and comparing operation coexist.

[Example 1]

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".]

Rule 5-0-14

It is the same comment as Rule 5-0-13, please refer to Rule 5-0-13 for detail.

Rule 5-0-15

[The corresponding error indicated by Agile+ Relief]

- A pointer arithmetic operation is not in the array indexing form .

[Example 1]

```
int    data[ ]={ 1, 2, 3, 4, 5 };
int    *p = data;
return    *(p+1);                    <-[Pointer arithmetic operation "p+1" is not in the form of array indexing.]
```

Rule 5-0-16

[The corresponding errors indicated by Agile+ Relief]

- An arithmetic operation performed on a pointer of non-array type.
- Use of a non-zero array pointer 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.]
 }
```

[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.]
}
```

Rule 5-0-17

[The corresponding error indicated by 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.]
```

Rule 5-0-18

[The corresponding error indicated by Agile+ Relief]

- Make magnitude comparisons between the addresses of two different objects.

[Example 1]

```
char   *p;
long   *q;
if( p < q ) {
```

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

Rule 5-0-19

[The corresponding error indicated by Agile+ Relief]

- The declaration contains more than 2 levels of indirect pointers.

[Example 1]

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

Rule 5-0-20

[The corresponding error indicated by Agile+ Relief]

- The operator of the bit operation is non-integer type.

[Example 1]

```
unsigned int    x;
double   d;
unsigned int    *xp;
x = d << 1 ;     <-[In ANSI, the operator "d" of the bit operation "d << 1" must be integer type.]
x = xp >> 2 ;    <-[In ANSI, the operator "xp" of the bit operation "xp >> 2" must be integer type.]
```

Rule 5-0-21

[The corresponding errors indicated by 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      x, y ;
x =y & 0x30 ;         <-[The operand "y" of the bit operation "y & 0x30" is signed.]
x <<= 3 ;             <-[The operand "x" of the bit operation "x <<= 3" is signed.]
if ( ^x == 0xfe )     <-[The operand "x" of the bit operation "^x" is signed.]
```

Rule 5-2-1

[The corresponding error indicated by 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.]
```

Rule 5-2-2

[The corresponding error indicated by Agile+ Relief]

- The pointer of basic class was cast to the pointer of derived class by non dynamic_cast.

[Example 1]

```
class ANIMAL {
    :
};
class FISH : public ANIMAL {
public:
    void Swim( ) {   }
};
class BIRD : public ANIMAL {
public:
    void Fly( ){   }
};
void func(ANIMAL* a ) {
    FISH* f = (FISH*)a;            <-[dynamic_cast should be used for the cast "(FISH *)a" to
                                   derived class. (after conversion : class FISH*, before
                                   conversion : class ANIMAL*)]
    f->Swim( );
    BIRD* b = static_cast<BIRD*>(a);  <-[dynamic_cast should be used for the cast
                                   "static_cast<BIRD *> (a)" to derived class. (after conversion :
                                   class BIRD*, before conversion : class ANIMAL*)]
    b->Fly( );
}
```

Rule 5-2-3

[The corresponding error indicated by Agile+ Relief]

- dynamic_cast was used.
- The pointer of basic class was cast to the pointer of derived class by non dynamic_cast.

[Example 1]

```
class X {/* X class definition (include virtual function) */};
class Y : public X { /* Y class definition */ };
class Z : public Y { /* Z class definition */ };
void fa(Z* pa)
{
    Y * pb = dynamic_cast < Y* >(pa);     <-[dynamic_cast may not be used in EC++.]
}
```

[Example 2]

```
class ANIMAL {
   :
};
class FISH : public ANIMAL {
public:
    void Swim( ) {   }
};
class BIRD : public ANIMAL {
public:
    void Fly( ){   }
};
void func(ANIMAL* a ) {
   FISH* f = (FISH*)a;                 <-[dynamic_cast should be used for the cast "(FISH *)a" to
                                         derived class. (after conversion : class FISH*, before
                                         conversion : class ANIMAL*)]
   f->Swim( );
   BIRD* b = static_cast<BIRD*>(a);    <-[dynamic_cast should be used for the cast
                                         "static_cast<BIRD *> (a)" to derived class. (after conversion :
                                         class BIRD*, before conversion : class ANIMAL*)]
   b->Fly( );
}
```

Rule 5-2-4

[The corresponding error indicated by Agile+ Relief]

- The cast expression of C style was used.

[Example 1]

```
class A {
    public:
       virtual void func( ){
       }
};
class B : public A {
    public:
       virtual void func( ){
       }
};
         :
A*a = new B;
B*b = (B*)a;                        <-[Please do not use cast"(B*)a" of C style in C++.]
```

Rule 5-2-5

[The corresponding error indicated by 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*".]
```

Rule 5-2-6

[The Corresponding errors indicated by Agile+ Relief]

- One side of the cast expression is of type pointer to a function, the other side is of integer type.

[Example 1]

```
int   (*fp)(void);
int   *ip;
ip = (int *)fp;
```

<-[The cast expression "(int *)fp" is a conversion involving a function pointer type and a non-integer type (cast type: int *, expression type: int (*) (void)).]

Rule 5-2-7

[The corresponding error indicated by Agile+ Relief]

- The pointer type was cast to another pointer type.

[Example 1]

```
int      *adr1 ;
char     *adr2 ;
adr1 = (int*)adr2 ;
```

<-[The cast expression "(int*)adr2"casted the pointer type to another pointer type. (cast type : char *, Expression : int *)]

Rule 5-2-8

[The corresponding error indicated by Agile+ Relief]

- Casting a non-pointer type to a pointer type.

[Example 1]

```
int     adr , *p ;
p = (int *)adr ;                 <-[The cast expression "(int *)adr" casts a non-pointer type to
                                 a pointer type (cast type: int *, expression type: int).]
```

Rule 5-2-9

[The corresponding error indicated by Agile^{+} Relief]

- Casting a pointer type to a non-pointer type.

[Example 1]

```
int      *p , adr ;
adr = (int)p ;                 <-[The cast expression "(int)p " casts a pointer type to a
                               non-pointer type (cast type: int *, expression type: int).]
```

Rule 5-2-10

[The corresponding error indicated by Agile+ Relief]

- Mixed use of increment/decrement operators and other expressions.

[Example 1]

```
i = j++;            <-[The result of "j++" is used in another operation.]
i = j + j++;        <-[The result of "j++" is used in another operation.]
i++, j = 1;         <-[The result of "i++" is used in another operation.]
```

Rule 5-2-11

[The corresponding error indicated by Agile+ Relief]

- && operator was defined in multiple times. Operator, ||Operator.

[Example 1]

```
class A {
    public:
     A* operator&() {
            :
      }
     bool operator&&(const A& rhs ) const {<-[Define the operator && in multiple times may cause
                                              changes in evaluation sequence during operation.]
            :
      }
     bool operator||(const A& rhs ) const {   <-[Define the operator || in multiple times may cause
                                              changes in evaluation sequence during operation.]
            :
      }
} ;
```

Rule 5-2-12

[The corresponding error indicated by Agile+ Relief]

- The array type of derived class was transferred to the pointer of basic class or array type.
- The virtual parameter of array type exists, but the parameter indicating the array size does not exist.

[Example 1]

File name: file.cc

```
    class BASE {
    public:
        int m1;
    } ;
    class DERIVED : public BASE {
    public:
        int m2;
        int m3;
    } ;
10: void func( BASE *pb, int n ) {       //The pointer of basic class was accepted, and the processing
                                         equivalent to the number of arrays was performed
        for( int i = 0 ; i < n ; i++ ) {
            cout << pb[i].m1 << endl ;
        }
    }
     :
    BASE b[5];
    func( b, 5 ) ;
    DERIVED d[5];
    func( d, 5 ) ;                        <-[The array of derived class is passed to the base class
                                         @1.(function: "func"; argument: No. the First one; type of
                                         argument: DERIVED [5]; type of parameter: BASE *;
                                         declaration : line 10 of "file.cc")]
```

[Example 2]

```
void func ( int h[] ){                   <-[The function "func" contains a parameter of type array, but
                                         there are no unsigned parameters describing the array size.]
}
```

Rule 5-3-1

[The corresponding errors indicated by Agile$^+$ Relief]

- A non-boolean value is used as a conditional expression.

[Example 1]

Code	Message
if (**(nerr) && (a + b)**)	<-[The logical operation " (nerr) && (a + b)" uses the result of the operation "(a + b)" as a condition.]
if (**(nerr) || (a << b)**)	<-[The logical operation "(nerr) || (a << b)" uses the result of the operation "(a << b)" as a condition.]

[Example 2]

Filename: file.cc

Code	Message
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.cc").]
:	
if (**!y**)	<-["y" is not a bool value (a non-bool value is assigned in line 3 of "file.cc").]

Rule 5-3-2

[The corresponding error indicated by 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.]
```

Rule 5-3-3

[The corresponding error indicated by Agile+ Relief]

- & operator was defined in multiple times.

[Example 1]

```
class A {
      public:
      A* operator&() {                  <-[Define the operator & in multiple times may cause changes
                                         in evaluation sequence during operation.]
            :
      }
} ;
```

Rule 5-3-4

[The corresponding error indicated by Agile+ Relief]

- There is an update expression, a function call or a volatile variable within the sizeof operator.

[Example 1]

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

Rule 5-8-1

[The corresponding errors indicated by 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 x, y ;	
x = y << 32 ;	<-[The number of bits specified for the shift operation "y << 32" exceeds the type width of the result.]
if (x == (y >> 32))	<-[The number of bits specified for the shift operation "y >> 32" exceeds the type width of the result.]
x <<= 32 ;	<-[The number of bits specified for the shift operation "x <<= 32" exceeds the type width of the result.]

Rule 5-14-1

[The corresponding error indicated by Agile+ Relief]

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

[Example 1]

```
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.]
```

Rule 5-17-1

Agile+ Relief will not check if this rule has been violated.

Rule 5-18-1

[The corresponding error indicated by 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.]

Rule 5-19-1

[The corresponding error indicated by Agile+ Relief]

- The operational result cannot be represented by an unsigned type.

[Example 1]

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

[Example 2]

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

Rule 6-2-1

[The corresponding errors indicated by Agile+ Relief]

- There is an assignment expression in the conditional expression.
- The assignment operation and comparing operation coexist.

[Example 1]

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".]

Rule 6-2-2

[The corresponding error indicated by 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.]
```

Rule 6-2-3

[The corresponding errors indicated by 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.]
```

Rule 6-3-1

[The corresponding errors indicated by Agile+ Relief]

- Statements controlled by for, while, do-while and switch are not embraced by { } .

[Example 1]

while (x > 0)	<-[It might be better to add curly brackets { } to this while statement.]
x--;	

Rule 6-4-1

[The corresponding errors indicated by Agile+ Relief]

- Statements controlled by if, else statement have not been embraced by { }.

[Example 1]

if (x == 0)	<-[It might be better to add curly brackets { } to this if statement.]
x = 10;	

Rule 6-4-2

[The corresponding errors indicated by Agile+ Relief]

- The if and else if statements do not end with an else statement.

[Example 1]

```
if ( x > 0 ) {
    :
} else 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.]

Rule 6-4-3

[The corresponding errors indicated by Agile+ Relief]

- The case label or default label is not contained in the { } after the switch statement.
- The case label has no corresponding break statement.
- No default label has been found in the switch statement.
- A default label is placed before the case label.
- There is a comparing operator, logical operator and a constant in the conditional expression of switch statement.
- There is only one case label in 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;
  }
```

[Example 2]

```
    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++;
   }
```

[Example 3]

```
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.]
        :
 }
```

Rule 6-4-4

[The corresponding errors indicated by 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;
    }
}
```

Rule 6-4-5

[The corresponding error indicated by Agile+ Relief]

- The case label has no corresponding break statement.

[Example 1]

```
    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++;
    }
```

Rule 6-4-6

[The corresponding errors indicated by Agile+ Relief]

- No default label has been found in the switch statement.
- A default label is placed before the case label.

[Example 1]

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

[Example 2]

```
switch ( x ) {
default:                       <-[The label default has been included before the label case
                                 in a switch statement.]
      no++
      break;
case 1 :
      no += x;
}
```

Rule 6-4-7

[The corresponding errors indicated by Agile+ Relief]

- There is a comparing operator, logical operator and a constant in the conditional expression of switch statement.
- There is only one case label in the switch statement.

[Example 1]

```
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.]
{ ~ }
```

[Example 2]

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

Rule 6-4-8

Agile[+] Relief will not check if this rule has been violated.

Rule 6-5-1

[The corresponding error indicated by Agile+ Relief]

- The loop counter of the for-statement is of floating-point type.
- Irrelevant variable exists in the condition to continue the loop.

[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.]
:	
}	

[Example 2]

for(i = 0, flag = 0;	<-[The expression “flag = 0”that is not relevant to the loop control exists in the for statement.]
i < n;	
i++, counter++)	<-[The expression “counter++” that is not relevant to the loop control exists in the for statement.]
{	
:	
}	

Rule 6-5-2

[The corresponding error indicated by Agile+ Relief]

- The loop condition of the for statement is == or !=.

[Example 1]

```
for ( int i = 0 ; i == n ; i++ )          <-[In the loop "for(i=0;i==n;i++)", the loop condition "i==n" may
                                            be wrong.]
{
      :
}
```

Rule 6-5-3

[The corresponding error indicated by Agile+ Relief]

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

[Example 1]

```
Filename: file.cc
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.cc" ).]
      }
```

Rule 6-5-4

Agile+ Relief will not check if this rule has been violated.

Rule 6-5-5

[The corresponding error indicated by Agile+ Relief]

- Irrelevant variable exists in the condition to continue the loop.

[Example 1]

Code	Comment
for(int i = 0, **flag = 0**;	<-[The expression "flag = 0"that is not relevant to the loop control exists in the for statement.]
i < n;	
i++, counter++)	<-[The expression "counter++" that is not relevant to the loop control exists in the for statement.]
{	
:	
}	

Rule 6-5-6

Agile⁺ Relief will not check if this rule has been violated.

Rule 6-6-1

[The corresponding error indicated by Agile+ Relief]

- A goto statement is used.

[Example 1]

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

Rule 6-6-2

It is the same comment as Rule 6-6-1, please refer to Rule 6-6-1 for detail.

Rule 6-6-3

[The corresponding errors indicated by Agile+ Relief]

- A continue statement is used.

[Example 1]

```
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;
  }
```

Rule 6-6-4

[The corresponding error indicated by 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;
    }
}
```

Rule 6-6-5

[The corresponding errors indicated by Agile+ Relief]

- More than one exit in the function.

[Example 1]

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

Rule 7-1-1

Agile+ Relief will not check if this rule has been violated.

Rule 7-1-2

[The corresponding error indicated by Agile+ Relief]

- Un-updated parameters of pointer type or array type.

[Example 1]

```
void func( int p[ ], unsigned int n ) <-[The space indicated by parameter "p" of type "int [ ]" is
                                         not updated and can be qualified with const. ]
{
    int i;
    int num = 0;
    for( i = 0; i < n; i++ ) {
        num += p[ i ];
    }
}
```

Rule 7-2-1

[The corresponding error indicated by Agile+ Relief]

- The type that is irrelevant to enumeration type was compared.

[Example 1]

```
enum COLOR{ RED, GREEN, BLUE } color ;
enum ERROR_CODE{ SUCCESS, ERROR } code ;
int x;
if( color == x )          <-["color" is compared with irrelative type.(enum type : enum COLOR, compared type : int)]
if( color != ERROR )      <-["color" is compared with irrelative type.(enum type : enum COLOR, compared type : enum ERROR_CODE)]
if( color <= 0 )          <-["color" is compared with irrelative type.(enum type : enum COLOR, compared type : int)]
if( color > -2 )          <-["color" is compared with irrelative value.(enum type : enum COLOR, compared value : -2)]
```

Rule 7-3-1

[The corresponding error indicated by Agile+ Relief]

- Name space is not used when declaring or defining a class or function.

[Example 1]

```
//A group class and function
class CLS_A {                    <-[To avoid name conflict, declaration or definition should be
                                  performed in namespace.]
public:
    int x ;
} ;
int   GetData( CLS_A& a ) ;

//B group class and function
class CLS_B {
public:
    long x ;
} ;
long GetData_B( CLS_B& b) {
    return b.x ;
}
```

Rule 7-3-2

[The corresponding error indicated by Agile+ Relief]

- The operator main was used.

[Example 1]

/*This comment is output when main is described under the [CHECK_IDENTIFIER] label of operator file */

if(**main** == ENOENT)   <-[You used the variable "main", please check.]

Rule 7-3-3

[The corresponding error indicated by Agile+ Relief]

- An anonymous name space has been defined in the head file.

[Example 1]

File name: head.h

```
namespace {                <-[The anonymous namespace was defined in the header file "head.h".]
    int x ;
}
```

Rule 7-3-4

Agile[+] Relief will not check if this rule has been violated.

Rule 7-3-5

Agile+ Relief will not check if this rule has been violated.

Rule 7-3-6

[The corresponding error indicated by Agile+ Relief]

- The using command or using declaration was used in the head file.

[Example 1]

File name: header.h

```
namespace NSA {
        void func( ) ;
} ;
using namespace NSA ;          <-["using was used in the head file. (File name: "header.h",
                               Description: "using namespace NSA")]
```

Rule 7-4-1

Agile+ Relief will not check if this rule has been violated.

Rule 7-4-2

Agile+ Relief will not check if this rule has been violated.

Rule 7-4-3

[The corresponding error indicated by Agile+ Relief]

- Assembly language is used.

[Example 1]

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

Rule 7-5-1

[The corresponding errors indicated by Agile+ Relief]

- The address of an automatic variable has been set to a return value of the function.

[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.]
    :
}
```

Rule 7-5-2

[The corresponding errors indicated by Agile+ Relief]

- The address of an automatic variable is assigned to a variable outside the effective scope.

[Example 1]

```
int   *p ;
{
      int    x;
      p = &x ;                  <-[The address of the automatic variable x cannot be used
                                outside the scope.]
}
*p = 0 ;
```

Rule 7-5-3

Agile[+] Relief will not check if this rule has been violated.

Rule 7-5-4

[The corresponding error indicated by Agile+ Relief]

- There is a recursive function.

[Example 1]

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

Rule 8-0-1

[The corresponding error indicated by Agile+ Relief]

- Comma was used in the declaration statement.

[Example 1]

```
int    *a, b;                          <-[Comma was used in the declaration statement.]
enum {
      AA, BB ,CC
} e1, e2 ;                             <-[Comma was used in the declaration statement.]
```

Rule 8-3-1

[The corresponding error indicated by Agile+ Relief]

- The default parameter value of virtual parameters in basic class and derived class are different.

[Example 1]

File name: file.cc

```
   class A {
   public:
3:     virtual void f( int n=1 ) {
       }
   } ;
   class B : public A {
       public:
       void f( int n=2 ) {          <-[The function "f" of class B modified the value of default
                                     parameter of the virtual function "f"(line 3 of "file.cc") of class
                                     A.]
       }
   } ;
```

Rule 8-4-1

[The corresponding errors indicated by Agile+ Relief]

- There is a variable length parameter in the function declaration and definition.

[Example 1]

Code	Error
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 "...".]
}	

Rule 8-4-2

[The corresponding error indicated by Agile+ Relief]

- The parameters of a function declaration and definition have different names.

[Example 1]

Filename: file.cc

Code	Message
int func(**int data**,	<-[The parameter name "data" is inconsistent with the parameter name in the function definition "func" in line 3 of "file.cc".]
int size);	<-[The parameter name "size" is inconsistent with the parameter name in the function definition "func" in line 3 of "file.cc".]

```
3:  int  func( int  size, int  data )
    {
        :
    }
```

Rule 8-4-3

[The corresponding error indicated by 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()
{
}                                        <-[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()
{
        :
   return   0;
        :
   return;                               <-[The return value of a return statement might have been
                                          omitted accidentally.]
}
```

Rule 8-4-4

[The corresponding error indicated by Agile+ Relief]

- Cast a function name to a non-function pointer.

[Example 1]

```
int   x , func( void ) ;
x = reinterpret_cast<int>(func) ;      <-[The expression "x = reinterpret_cast<int>(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.]
        :
}
```

Rule 8-5-1

[The corresponding error indicated by 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.]
```

Rule 8-5-2

[The corresponding error indicated by Agile+ Relief]

- The initializer list and the initialized array/structure/union do not match in the construction.

[Example 1]

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;	
};	
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;	
} ;	
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.]

Rule 8-5-3

[The corresponding error indicated by 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.]
```

Rule 9-3-1

Agile+ Relief will not check if this rule has been violated.

Rule 9-3-2

[The corresponding error indicated by Agile+ Relief]

- The return value is the address of non-const number which has a more strict access limit.

[Example 1]

```
File name: file.cc
    class CLS {
        public :
            int * f ( );
            :
        private :
20:         int data;
    } ;
    int *CLS::f ( )
    {
        return &data ;          <-[The address of the number "data" which has a more strict
                                access limit than the function "f" was returned as a function.
                                (Member declaration: Line 20 of "file.cc")]
    }
```

Rule 9-33

[The corresponding error indicated by Agile+ Relief]

- The member function without const limit and without member variable being updated was defined.

[Example 1]

```
class   CLS {
private:
        int m_val;
        int f( ) {                  <-[Because the member function "f" of class CLS did not
                                    updated the status of object, const limit can be performed.]
                return m_val;
        }
} ;
```

Rule 9-5-1

[The corresponding error indicated by Agile+ Relief]

- A union is used.

[Example 1]

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

Rule 9-6-1

Agile[+] Relief will not check if this rule has been violated.

Rule 9-6-2

[The corresponding error indicated by Agile+ Relief]

- The bitfield is declared with char , short or long type.

[Example 1]

```
struct   STR {
   unsigned   short   m1:2 ;          <-[The bitfield "m1" is declared with type unsigned short. The
                                      declaration should be made with signed int or unsigned int.]
   unsigned   int       m2: 2 ;
   signed      int       m3: 2 ;
};
```

Rule 9-6-3

[The corresponding error indicated by Agile+ Relief]

- A bit field of enum type is declared.

[Example 1]

```
struct STR {
    enum week {SUN, MON, TUE, WED, THU, FRY, SAT};
    week val : 6;        <-[The bitfield "val" is declared with type week. The
                           declaration should be made with signed int or unsigned int.]
};
```

Rule 9-6-4

[The corresponding error indicated by Agile+ Relief]

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

[Example 1]

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

Rule 10-1-1

[The corresponding error indicated by Agile+ Relief]

- Virtual inheritance was used.

[Example 1]

```
class B : public virtual A {                <-[EC++ cannot use virtual inheritance.]
      :
};
```

Rule 10-1-2

Agile+ Relief will not check if this rule has been violated.

Rule 10-1-3

Agile+ Relief will not check if this rule has been violated.

Rule 10-2-1

Agile+ Relief will not check if this rule has been violated.

Rule 10-3-1

Agile+ Relief will not check if this rule has been violated.

Rule 10-3-2

Agile+ Relief will not check if this rule has been violated.

Rule 10-3-3

Agile+ Relief will not check if this rule has been violated.

Rule 11-0-1

[The corresponding error indicated by Agile+ Relief]

- The data member of public was declared in the class.

[Example 1]

```
class CLS {                    <-[public data exists in the class "CLS". Please do not use the
                               public data.]
public :
      CLS( );
      int dataa;
      int datab;
      int datac;
} ;
```

Rule 12-1-1

[The corresponding error indicated by Agile+ Relief]

- The virtual function was called in constructor function or destructor function.

[Example 1]

File name: file.cc

```
    class CLS {
    public:
        CLS( ) {
            m=f( );         <-[The virtual function "f"(Defined in line 6 of "file.cc") was called in the constructor function "CLS".]
        }
6:      virtual int f( ) {
            return 1;
        }
        int m;
    } ;
```

Rule 12-1-2

Agile+ Relief will not check if this rule has been violated.

Rule 12-1-3

[The corresponding error indicated by Agile+ Relief]

- Some classes contain the constructor function of single parameter without specified explicit.

[Example 1]

```
class CLS {                          <-[class CLS has the single-argument constructor.]
public:
    CLS( int x ) {
        m = x ;
    }
private:
    int m ;
} ;
```

Rule 12-8-1

[The corresponding error indicated by Agile+ Relief]

- In the copy constructor function, not all of the data members that contain the basic class have been initialized.

[Example 1]

```
class A {
public :
        A( ) { }
        A( const A & a ) {
            m_a1 = a.m_a1 ;
            m_a2 = a.m_a2 ;
        }
private :
         int m_a1 ;
         int m_a2 ;
} ;
class B : public A {
public :
       //Data member that does not copy the basic class
        B( const B & b ) :
            m_b1( b.m_b1 ) , m_b2( b.m_b2 ) {   <-[The copy constructor function of class B does not
                                                   copy all data members.]
        }
private :
         int m_b1 ;
         int m_b2 ;
} ;
class C : public A {
public :
       // Does not copy m_c2
        C( const C & c ) :
                A( c ) , m_c1( c.m_c1 ) {         <-[The copy constructor function of class C does not
                                                   copy all data members.]
        }
private :
         int m_c1 ;
         int m_c2 ;
} ;
```

Rule 12-8-2

Agile+ Relief will not check if this rule has been violated.

Rule 14-5-1

Agile+ Relief will not check if this rule has been violated.

Rule 14-5-2

[The corresponding error indicated by Agile+ Relief]

- In the class of data member with defined destructor function or pointer type, copy constructor function and copy assignment operator are not defined.

[Example 1]

```
class A {                     <-[class CLS explicitly defined destructor function, but it does
                              not explicitly declared copy constructor function or copy
                              assignment operator.]
public:
      A();
      ~A();
};
```

[Example 2]

```
class B {                     <-[class B contains pointer type data member, but does not
                              explicitly declared copy constructor function or copy
                              assignment operator.]
public:
      B( ) {
          m.create ( );
      }
      ~B( ) {
          m.delete( );
      }
private:
      char* m;
} ;
```

Rule 14-5-3

It is the same comment as Rule 14-5-2, please refer to Rule 14-5-2 for detail.

Rule 14-6-1

Agile+ Relief will not check if this rule has been violated.

Rule 14-6-2

Agile+ Relief will not check if this rule has been violated.

Rule 14-7-1

Agile+ Relief will not check if this rule has been violated.

Rule 14-7-2

Agile+ Relief will not check if this rule has been violated.

Rule 14-7-3

Agile[+] Relief will not check if this rule has been violated.

Rule 14-8-1

Agile+ Relief will not check if this rule has been violated.

Rule 14-8-2

Agile+ Relief will not check if this rule has been violated.

Rule 15-0-1

Agile+ Relief will not check if this rule has been violated.

Rule 15-0-2

[The corresponding error indicated by Agile+ Relief]

- The object of pointer type was thrown.

[Example 1]

```
class Exception {
} ;
void f1( ) {
    Exception* p = new Exception ;
    throw p ;                              <-[The exception "p" should throw a value.]
}
```

Rule 15-0-3

Agile+ Relief will not check if this rule has been violated.

Rule 15-1-1

[The corresponding error indicated by Agile+ Relief]

- In the constructor function or destructor function, the new expression or delete expression was described outside the try block.

[Example 1]

```
class CLS{
public:
     CLS(int m){                        <-[The constructor "CLS" should catch exceptions.]
          m_p = new int[m];
     }
    ~CLS(){                             <-[The destructor "~CLS" should catch exceptions.]
          delete m_p;
    }
private:
    int* m_p;
};
int foo(){
    try{
         CLS cls(-1);
    } catch(...) {.
    }
}
```

Rule 15-1-2

[The corresponding error indicated by Agile+ Relief]

- NULL was thrown.

[Example 1]

```
try {
    throw NULL;                      <-[NULL is thrown.]
} catch ( int e ){
    //Exception processing of integer;
} catch ( char* e ){
    //Exception processing of character pointer;
} catch( int* e ){
    //Exception processing of integer pointer;
}
```

Rule 15-1-3

Agile+ Relief will not check if this rule has been violated.

Rule 15-3-1

It is the same comment as Rule 15-1-1, please refer to Rule 15-1-1 for detail.

Rule 15-3-2

[The corresponding error indicated by Agile+ Relief]

- The catch handler is not ended with catch(...).

[Example 1]

```
void func1( int x ) {
    if(x == 0) {
        throw 0 ;
    }
    Library_x( ) ; //unknown type of throw
}
void func2( ) {
    try {
        func1( 0 ) ;
    }
    catch( int n ) {                    <-[The end of catch handler is not catch(...).]
        // error process
    }
}
```

Rule 15-3-3

Agile[+] Relief will not check if this rule has been violated.

Rule 15-3-4

Agile[+] Relief will not check if this rule has been violated.

Rule 15-3-5

[The corresponding error indicated by Agile+ Relief]

- The parameter during catch can be described by quoting.

[Example 1]

```
void func( ) {
    try {
      f( );
    }
    catch( Except e ) {                    <-[Exception should be catched by quoting.]
        :
    }
}
```

Rule 15-3-6

[The corresponding error indicated by Agile+ Relief]

- The catch handler of basic class comes in the former, and the exception handler of derived class comes in the later.

[Example 1]

File name: file.cc

```
class BICYCLE {
public:
    void run_stop( ) { }   //Run stops
} ;
class MOTORCYCLE : public BICYCLE {
public:
    void engine_stop( ) { } //Engine stops
} ;
        :
 try {
     MOTORCYCLE m;
     throw m;
 }
 catch( BICYCLE& b ) {          <-[The catch handler for base class "class BICYCLE" exists
                                  before the catch handler (in line 18 of "file.cc") for derived
                                  class "class MOTORCYCLE".]
     b.run_stop( ) ;   //Run stops
 }
18: catch( MOTORCYCLE& m ) {
     m.run_stop( ) ;   //Run stops
     m.engine_stop( ) ;//Engine stops
 }
```

Rule 15-3-7

It is the same comment as Rule 15-3-2, please refer to Rule 15-3-2 for detail.

Rule 15-4-1

[The corresponding error indicated by Agile^{+} Relief]

- Exception specification was used.

[Example 1]

```
void f( ) throw(ERROR1) {                <-[The exception specification is used.]
        :
}
```

Rule 15-5-1

It is the same comment as Rule 15-3-1, please refer to Rule 15-3-1 for detail.

Rule 15-5-2

It is the same comment as Rule 15-4-1, please refer to Rule 15-4-1 for detail.

Rule 15-5-3

[The corresponding error indicated by Agile+ Relief]

- The catch handler of the main function is not ended with catch(...).

[Example 1]

```
void main( ) {
    try {
        :
    }
    catch( char c ) {          <-[The end of catch handler of the function "main" is not catch(...).]
        // error process
    }
}
```

Rule 16-0-1

[The corresponding error indicated by Agile+ Relief]

- Before the line #include, contents exist that are neither preprocessing directives nor comments.

[Example 1]

Filename: file.cc

```
1:      double   func()
        {   return   X; }
        #include "head.h"
```

<-[Statements (in line 1 of "file.cc") which are not preprocessing directives exist before this #include line.]

Filename: head.h

```
extern double X;
```

Rule 16-0-2

[The corresponding error indicated by Agile+ Relief]

- #define or #undef statements exist in the block.

[Example 1]

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

Rule 16-0-3

[The corresponding error indicated by Agile+ Relief]

- A #undef is used.

[Example 1]

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

Rule 16-0-4

[The corresponding error indicated by Agile+ Relief]

- Macro function is defined.

[Example 1]

```
#define max( a , b )    ( ( a > b ) ? a : b ) <-[The macro function "max" can be inline function or template
                                               function.]
```

Rule 16-0-5

[The corresponding error indicated by Agile+ Relief]

- The argument of the macro function begins with #.

[Example 1]

```
#define   F(b)   b
F(#error "memory error")          <-[A preprocessing directive is used in the argument "#error" of a macro function.]
```

Rule 16-0-6

[The corresponding error indicated by Agile^{+} Relief]

- The parameter in the replacement string of the macro is not embraced with parentheses ().

[Example 1]

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

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

Rule 16-0-7

[The corresponding errors indicated by Agile+ Relief]

- An unspecified replacement string exists in the #if or #elif statement.
- An #undef is used on the undefined macro.

[Example 1]

/*The macro DEBUG is undefined, or no replacement string has been specified for it*/

#if DEBUG == 1 <-[The replacement string of the macro "DEBUG" in an #if or #elif statement is not specified.]

#elif DEBUG == 2 <-[The replacement string of the macro "DEBUG" in an #if or #elif statement is not specified.]

[Example 2]

/* There are no definitions of macro ABC,XYZ*/

#if ABC <-["ABC" might have been intended to be appended with defined.]

#elif XYZ <-["XYZ" might have been intended to be appended with defined.]

[Example 3]

/* There are no definition of macro BBB*/

#undef BBB <-[#undef is used on the undefined macro "BBB".]

Rule 16-0-8

[The corresponding error indicated by Agile+ Relief]

- In ANSI specification, some lines have started with the undefined #.

[Example 1]

#asm	<-[The undefined pre-processing command "#asm" cannot be described in ANSI.]

Rule 16-1-1

[The corresponding errors indicated by 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.]

Rule 16-1-2

[The corresponding error indicated by 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.]

Rule 16-2-1

[The corresponding error indicated by Agile⁺ Relief]

- Macro function was defined.
- The integer constant has been defined for Macro.

[Example 1]

```
#define max( a , b )   ( ( a > b ) ? a : b ) <-[The macro function "max" can be inline function or template
                                               function.]
```

[Example 2]

```
#define   RED       1        <-[The macro "RED" can be const constant or enumeration
                                constant.]
#define   YELLOW    2        <-[The macro "YELLOW" can be const constant or
                                enumeration constant.]
#define   BLUE      3        <-[The macro "BLUE" can be const constant or enumeration
                                constant.]
```

Rule 16-2-2

It is the same comment as Rule 16-2-1, please refer to Rule 16-2-1 for detail.

Rule 16-2-3

[The corresponding error indicated by Agile+ Relief]

- Multiple inclusions of the same file exist.

[Example 1]

Filename: file.cc

```
11:    #include   "head.h"
       #include   "head.h"          <-[The file "file.h" cannot be included redundantly (location of
                                      redundancy: line 11 in "file.cc").]
```

Filename: head.h

```
    extern   int   X;
```

Rule 16-2-4

[The corresponding error indicated by Agile+ Relief]

- In the #include-specified file name, there are '(single quotation), "(double quote), /*(slash and asterisk) and multi-bytes characters(Chinese characters, etc).

[Example 1]

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

Rule 16-2-5

[The corresponding error indicated by Agile+ Relief]

- In the #include-specified file name, there are \(currency character).

[Example 1]

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

Rule 16-2-6

[The corresponding errors indicated by 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 " ".]

Rule 16-3-1

[The corresponding error indicated by Agile+ Relief]

- The #operator and the ##operator coexist in the macro function.

[Example 1]

```
#define    AAA(x,y)    x###y
```

<-[The operators # and ## are used together in the definition of the macro function "AAA".]

Rule 16-3-2

[The corresponding error indicated by 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.]
```

Rule 16-6-1

Agile+ Relief will not check if this rule has been violated.

Rule 17-0-1

[The corresponding errors indicated by 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.]
```

Rule 17-0-2

[The corresponding error indicated by Agile+ Relief]

- An identifier using a name identical to that of the reserved identifier is used in the declaration and definition.

The variable and function using identical names registered under the label [RESERVED_IDENTIFIER] and [RESERVED_LIBRARY_IDENTIFIER] of the identifier file are checked as an object. For more information regarding the registration of an identifier file, see the -F option in the "Command Manual".

[Example 1]

```
int     errno ;                    <-[The name "errno" is the same as an external name used in
                                     the system.]
```

Rule 17-0-3

[The corresponding error indicated by Agile+ Relief]

- An identifier using a name identical to that of the reserved identifier is used in the declaration and definition.

The variable and function using identical names registered under the label [RESERVED_IDENTIFIER] and [RESERVED_LIBRARY_IDENTIFIER] of the identifier file are checked as an object. For more information regarding the registration of an identifier file, see the -F option in the "Command Manual".

[Example 1]

```
void   *malloc( unsigned   int   size ) {  <-[The name "malloc" is the same as an external name used
                                              in the system.]
           :
}
```

Rule 17-0-4

Agile+ Relief will not check if this rule has been violated.

Rule 17-0-5

[The corresponding error indicated by Agile+ Relief]

- The longjmp and macro setjmp functions are used.

[Example 1]

```
#include    <setjmp.h>
static    jmp_buf    parse_error;
:
longjmp( parse_error, 1 );                    <-[The function "longjmp" is used.]
```

Rule 18-0-1

[The corresponding error indicated by Agile^{+} Relief]

- The following standard C library was imported through #include.
  - assert.h
  - iso646.h
  - setjmp.h
  - stdio.h
  - wchar.h
  - ctype.h
  - limits.h
  - signal.h
  - stdlib.h
  - wctype.h
  - errno.h
  - locale.h
  - stdarg.h
  - string.h
  - float.h
  - math.h
  - stddef.h
  - time.h

[Example 1]

```
#include <locale.h>                    <-[You have included the file "locale.h", please check.]
```

Rule 18-0-2

[The corresponding error indicated by Agile+ Relief]

- The atof, atoi and atoll functions are used.

[Example 1]

```
#include    <stdlib.h>
int    i;
char    *str = "123";
i = atoi( str );                                   <-[The function "atoi" is used.]
```

Rule 18-0-3

[The corresponding error indicated by Agile+ Relief]

- The abort, exit, getenv, and system functions are used.

[Example 1]

```
#include    <stdlib.h>
:
if ( j < 0 ) {
abort( );                                   <-[The function "abort" is used.]
}
```

Rule 18-0-4

[The corresponding error indicated by Agile+ Relief]

- The following functions are used:
  clock
  difftime
  mktime
  time
  asctime
  ctime
  gmtime
  localtime
  strftime

[Example 1]

```
#include    <ctime>
:
time_t    tm:
time( &tm );                                    <-[The function "time" is used.]
```

Rule 18-0-5

[The corresponding error indicated by Agile+ Relief]

- The following functions are used:
  strcpy
  strcmp
  strcat
  strchr
  strspn
  strcspn
  strpbrk
  strrchr
  strstr
  strtok
  strlen

[Example 1]

```
char a[4] = “123”;
char n[4];
strcpy( n, a );          <-[You have used the function “strcpy”, please check.]
```

Rule 18-2-1

[The corresponding error indicated by Agile+ Relief]

- A macro offsetof is used.

[Example 1]

```
#include <cstddef>
struct STR {
        char *sp;
        long dt;
} *strp;
long *dtp = (long *)( (char*)strp + offsetof( struct STR, dt) );
                                        <-[macro "offsetof" is used]
```

Rule 18-4-1

[The corresponding error indicated by 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.]
```

Rule 18-7-1

[The corresponding errors indicated by Agile+ Relief]

- The following functions and macros are used:
  signal
  SIG_DFL
  SIG_IGN
  SIG_ERR
  SIGABRT
  SIGFPE
  SIGILL
  SIGINT
  SIGSEGV
  SIGTERM

[Example 1]

```
extern   void   sigint_hnd();
:
signal( SIGINT, sigint_hnd );            <-[The function "signal" is used.]
                                         <-[The macro "SIGINT" is used.]
```

Rule 19-3-1

[The corresponding error indicated by Agile+ Relief]

- An identifier errno is used.

[Example 1]

```
#include    <errno.h>
:
if( errno == ENOENT ){                      <-[The variable "errno" is used. Please check.]
        :
}
```

Rule 27-0-1

[The corresponding errors indicated by Agile+ Relief]

- File stdio.h or cstdio is included by the #include statement.
- The following functions are used:
  fgetpos
  fopen
  ftell
  gets
  perror
  remove
  rename

[Example 1]

```
#include    <stdio.h>                    <-[The file "locale.h" is included and should be confirmed.]
FILE    *fp;
     :
fp = fopen( "abc.txt", "r");             <-[The function "fopen" is used.]
```

Appendix A: A List of MISRA Rules

A.1 MISRA-C V1 Rule

Rule	Yes or No[2]	Rule	Yes or No[2]	Rule	Yes or No[2]	Rule	Yes or No[2]
1	Y	33	Y	65	Y	97	Y
2[1]	N	34	Y	66	Y	98	Y
3	Y	35	Y	67	Y	99[1]	N
4[1]	X[3]	36	Y	68	Y	100	Y
5	Y	37	Y	69	Y	101	Y
6[1]	N	38	Y	70	Y	102	Y
7	Y	39	Y	71	Y	103	X[5]
8	Y	40	Y	72	Y	104	Y
9	Y	41[1]	N	73	Y	105	Y
10	N	42	Y	74	Y	106	Y
11	Y	43	Y	75	Y	107	Y
12	Y	44	Y	76	Y	108	Y
13	Y	45	Y	77	Y	109	Y[6]
14	Y	46	Y	78	Y	110	Y
15[1]	N	47	Y	79	Y	111	Y
16	Y	48	Y	80	Y	112	Y
17	Y	49	Y	81	Y	113	Y
18	Y	50	Y	82	Y	114	Y
19	Y	51	Y	83	Y	115	Y
20	Y	52	Y	84	Y	116[1]	N
21	Y	53	Y	85	Y	117	X[7]
22	Y	54	Y	86	Y	118	Y
23	Y	55	Y	87	Y	119	Y
24	Y	56	Y	88	Y	120	Y
25	Y	57	Y	89	Y	121	Y
26	Y	58	Y	90	Y	122	Y
27	Y	59	Y	91	Y	123	Y
28	Y	60	Y	92	Y	124	Y
29	Y	61	Y	93	Y[4]	125	Y
30	Y	62	Y	94	Y	126	Y
31	Y	63	Y	95	Y	127	Y
32	Y	64	Y	96	Y		

[1]: Rules required for documentation.

[2]: **Y**: Violation of the MISRA rule can be indicated
N: Violation of the MISRA rule can not be indicated
X: Rules, that during static analysis are difficult to check but that will be checked if possible.

[3]: Violation of the rule will be indicated in the following cases:

- A bitwise shift operation out of the type size.
- Divided by zero.
- Out of the range of the array.
- Address "0" may be referenced.

[4]: macro functions that may result in failures.

[5]: Messages will be indicated for magnitude comparisons of addresses belonging to

objects of different types.

[6]: Messages will be indicated for a union variable in which a member has been assigned a value but an different member is used.

[7]: Messages will be indicated for calling a real-time library function, the argument address of which may be 0.

A.2 MISRA-C V2 Rule

Rule	Yes or No[2]	Rule	Yes or No[2]	Rule	Yes or No[2]	Rule	Yes or No[2]
1.1	Y	8.7	Y	13.4	Y	18.2	Y
1.2	Y	8.8	Y	13.5	Y	18.3	N
1.3[1]	N	8.9	Y	13.6	Y	18.4	Y
1.4[1]	N	8.10	Y	13.7	Y	19.1	Y
1.5[1]	N	8.11	Y	14.1	Y	19.2	Y
2.1	Y	8.12	Y	14.2	Y	19.3	Y
2.2	Y	9.1	Y	14.3	Y	19.4	Y
2.3	Y	9.2	Y	14.4	Y	19.5	Y
2.4	N	9.3	Y	14.5	Y	19.6	Y
3.1[1]	N	10.1	Y	14.6	Y	19.7	Y
3.2[1]	N	10.2	Y	14.7	Y	19.8	Y
3.3[1]	N	10.3	Y	14.8	Y	19.9	Y
3.4[1]	N	10.4	Y	14.9	Y	19.10	Y
3.5[1]	Y	10.5	Y	14.10	Y	19.11	Y
3.6[1]	N	10.6	Y	15.1	Y	19.12	Y
4.1	Y	11.1	Y	15.2	Y	19.13	Y
4.2	Y	11.2	Y	15.3	Y	19.14	Y
5.1	Y	11.3	Y	15.4	Y	19.15	Y
5.2	Y	11.4	Y	15.5	Y	19.16	Y
5.3	Y	11.5	Y	16.1	Y	19.17	Y
5.4	Y	12.1	Y	16.2	Y	20.1	Y
5.5	Y	12.2	Y	16.3	Y	20.2	Y
5.6	Y	12.3	Y	16.4	Y	20.3	X[4]
5.7	Y	12.4	Y	16.5	Y	20.4	Y
6.1	Y	12.5	Y	16.6	Y	20.5	Y
6.2	Y	12.6	Y	16.7	Y	20.6	Y
6.3	Y	12.7	Y	16.8	Y	20.7	Y
6.4	Y	12.8	Y	16.9	Y	20.8	Y
6.5	Y	12.9	Y	16.10	Y	20.9	Y
7.1	Y	12.10	Y	17.1	Y	20.10	Y
8.1	Y	12.11	Y[3]	17.2	Y	20.11	Y
8.2	Y	12.12	Y	17.3	Y	20.12	Y
8.3	Y	12.13	Y	17.4	Y	21.1	X[5]
8.4	Y	13.1	Y	17.5	Y		
8.5	Y	13.2	Y	17.6	Y		
8.6	Y	13.3	Y	18.1	Y		

[1]: Rules required for documentation.

[2]: **Y**: Violation of the MISRA rule can be indicated
N : Violation of the MISRA rule can not be indicated
X : Rules, that during static analysis are difficult to check but that will be checked if possible.

[3]: Messages will be indicated for a negative result from the subtraction of an unsigned constant.

[4]: Messages will be indicated for calling a real-time library function, the argument address of which may be "0".

[5]: Violation of the rule will be indicated in the following cases:

- A bitwise shift operation out of the type size
- Divided by zero

- Out of the range of the array
- Address "0" may be referenced

A.3 MISRA-C V3 Rule

- The rule that there is (*1) mark in the rule column is a rule added by MISRA-C:2012 Amendment 1.

Rule	Yes or No[1]	Rule	Yes or No[1]	Rule	Yes or No[1]	Rule	Yes or No[1]
1.1	Y	8.11	Y	14.3	Y	20.6	Y
1.2	Y	8.12	Y[8]	14.4	Y	20.7	Y
1.3	Y	8.13	Y	15.1	Y	20.8	Y
2.1	Y	8.14	Y	15.2	Y	20.9	Y
2.2	Y	9.1	Y	15.3	Y	20.10	Y
2.3	Y	9.2	Y	15.4	Y	20.11	Y
2.4	Y	9.3	Y	15.5	Y	20.12	Y
2.5	Y	9.4	Y	15.6	Y	20.13	Y
2.6	Y	9.5	Y	15.7	Y	20.14	Y
2.7	Y	10.1	Y	16.1	Y	21.1	Y
3.1	X[2]	10.2	Y	16.2	Y	21.2	Y
3.2	Y	10.3	Y	16.3	Y	21.3	Y
4.1	Y	10.4	Y	16.4	Y	21.4	Y
4.2	Y	10.5	Y	16.5	Y	21.5	Y
5.1	Y	10.6	Y	16.6	Y	21.6	Y
5.2	Y	10.7	Y	16.7	Y	21.7	Y
5.3	Y	10.8	Y	17.1	Y	21.8	Y
5.4	Y	11.1	Y	17.2	Y	21.9	Y
5.5	Y	11.2	Y	17.3	Y	21.10	Y
5.6	Y	11.3	Y	17.4	Y	21.11	Y
5.7	Y	11.4	Y	17.5	Y	21.12	Y
5.8	Y[3]	11.5	Y	17.6	Y	21.13(*1)	Y
5.9	Y	11.6	X[9]	17.7	Y	21.14(*1)	Y
6.1	X[4]	11.7	Y	17.8	Y	21.15(*1)	Y
6.2	Y	11.8	Y	18.1	Y	21.16(*1)	Y[11]
7.1	Y	11.9	Y	18.2	Y	21.17(*1)	Y[12]
7.2	Y	12.1	Y	18.3	Y	21.18(*1)	Y
7.3	Y	12.2	Y	18.4	Y[10]	21.19(*1)	Y
7.4	Y	12.3	Y	18.5	Y	21.20(*1)	Y
8.1	Y	12.4	Y	18.6	Y	22.1	Y
8.2	Y	12.5(*1)	Y	18.7	Y	22.2	Y
8.3	X[5]	13.1	Y	18.8	Y	22.3	Y
8.4	X[6]	13.2	Y	19.1	Y	22.4	Y
8.5	X[7]	13.3	Y	19.2	Y	22.5	Y
8.6	Y	13.4	Y	20.1	Y	22.6	Y
8.7	Y	13.5	Y	20.2	Y	22.7(*1)	Y
8.8	Y	13.6	Y	20.3	Y	22.8(*1)	Y
8.9	Y	14.1	Y	20.4	Y	22.9(*1)	Y
8.10	Y	14.2	Y	20.5	Y	22.10(*1)	Y

[1]: **Y**: Violation of the MISRA rule can be indicated.
X : Rules, that during static analysis are difficult to check but that will be checked if possible.

[2]: The following “/*” and “//” descriptions can not be indicated.

- "//" description in /* comment

- "/*" and "//" descriptions in // comment

[3]: The variable and the function without the external linkage are indicated.

[4]: "_ Bool" type can not be indicated.

[5]: The presence of the type qualifier can not be indicated.

[6]: The presence of variable declaration can not be indicated.

[7]: The declaration in two or more header files can not be indicated.

[8]: When the value of the enumeration constant is unique, it is likely to be indicated.

[9]: The integer constant can not be indicated.

[10]: The arithmetic operation of "++" and "--" are indicated.

[11]: The string or floating type object of comparison in the memcmp function can not be indicated.

[12]: The strcat function and the strcpy function are indicated.

A.4 MISRA-C++ V1 Rule

Rule	Yes or No[2]
0-1-1	Y
0-1-2	Y
0-1-3	X[3]
0-1-4	Y
0-1-5	N
0-1-6	N
0-1-7	Y
0-1-8	N
0-1-9	X[4]
0-1-10	X[5]
0-1-11	X[6]
0-1-12	X[6]
0-2-1	Y
0-3-1[1]	N
0-3-2	Y
0-4-1[1]	N
0-4-2[1]	N
0-4-3[1]	N
1-0-1	X[7]
1-0-2[1]	N
1-0-3[1]	N
2-2-1[1]	N
2-3-1	Y
2-5-1	N
2-7-1	Y
2-7-2	N
2-7-3	N
2-10-1	N
2-10-2	Y
2-10-3	X[8]
2-10-4	X[9]
2-10-5	X[10]
2-10-6	Y
2-13-1	Y
2-13-2	Y
2-13-3	Y
2-13-4	Y
2-13-5	N
3-1-1	Y
3-1-2	N
3-1-3	Y
3-2-1	N
3-2-2	N
3-2-3	Y
3-2-4	X[11]
3-3-1	Y
3-3-2	Y
3-4-1	X[12]

Rule	Yes or No[2]
4-5-3	N
4-10-1	N
4-10-2	N
5-0-1	Y
5-0-2	Y
5-0-3	N
5-0-4	X[13]
5-0-5	X[14]
5-0-6	Y
5-0-7	N
5-0-8	N
5-0-9	N
5-0-10	Y
5-0-11	N
5-0-12	N
5-0-13	Y
5-0-14	Y
5-0-15	Y
5-0-16	Y
5-0-17	Y
5-0-18	X[15]
5-0-19	Y
5-0-20	X[16]
5-0-21	Y
5-2-1	Y
5-2-2	Y
5-2-3	Y
5-2-4	Y
5-2-5	Y
5-2-6	Y
5-2-7	Y
5-2-8	Y
5-2-9	Y
5-2-10	Y
5-2-11	Y
5-2-12	Y
5-3-1	Y
5-3-2	Y
5-3-3	Y
5-3-4	Y
5-8-1	Y
5-14-1	Y
5-17-1	N
5-18-1	Y
5-19-1	Y
6-2-1	Y
6-2-2	Y
6-2-3	Y

Rule	Yes or No[2]
6-4-5	Y
6-4-6	Y
6-4-7	Y
6-4-8	N
6-5-1	Y
6-5-2	Y
6-5-3	Y
6-5-4	N
6-5-5	X[17]
6-5-6	N
6-6-1	Y
6-6-2	Y
6-6-3	Y
6-6-4	Y
6-6-5	Y
7-1-1	N
7-1-2	Y
7-2-1	Y
7-3-1	Y
7-3-2	Y
7-3-3	Y
7-3-4	N
7-3-5	N
7-3-6	Y
7-4-1[1]	N
7-4-2	N
7-4-3	X[18]
7-5-1	Y
7-5-2	Y
7-5-3	N
7-5-4	Y
8-0-1	Y
8-3-1	Y
8-4-1	Y
8-4-2	Y
8-4-3	Y
8-4-4	Y
8-5-1	Y
8-5-2	Y
8-5-3	Y
9-3-1	N
9-3-2	X[19]
9-3-3	Y
9-5-1	Y
9-6-1[1]	N
9-6-2	Y
9-6-3	Y
9-6-4	Y

Rule	Yes or No[2]
10-3-2	N
10-3-3	N
11-0-1	Y
12-1-1	Y
12-1-2	N
12-1-3	Y
12-8-1	Y
12-8-2	N
14-5-1	N
14-5-2	Y
14-5-3	Y
14-6-1	N
14-6-2	N
14-7-1	N
14-7-2	N
14-7-3	N
14-8-1	N
14-8-2	N
15-0-1[1]	N
15-0-2	Y
15-0-3	N
15-1-1	Y
15-1-2	Y
15-1-3	N
15-3-1	X[20]
15-3-2	Y
15-3-3	N
15-3-4	N
15-3-5	Y
15-3-6	Y
15-3-7	Y
15-4-1	Y
15-5-1	Y
15-5-2	Y
15-5-3	Y
16-0-1	Y
16-0-2	Y
16-0-3	Y
16-0-4	Y
16-0-5	Y
16-0-6	Y
16-0-7	Y
16-0-8	Y
16-1-1	Y
16-1-2	Y
16-2-1	X[21]
16-2-2	Y
16-2-3	Y

Rule	Yes or No[2]	Rule	Yes or No[2]	Rule	Yes or No[2]	Rule	Yes or No[2]
3-9-1	N	6-3-1	Y	10-1-1	Y	16-2-4	Y
3-9-2	Y	6-4-1	Y	10-1-2	N	16-2-5	Y
3-9-3	Y	6-4-2	Y	10-1-3	N	16-2-6	Y
4-5-1	Y	6-4-3	Y	10-2-1	N	16-3-1	Y
4-5-2	N	6-4-4	Y	10-3-1	N	16-3-2	Y
16-6-1[1]	N	17-0-4[1]	N	18-0-3	Y	18-4-1	Y
17-0-1	Y	17-0-5	Y	18-0-4	Y	18-7-1	Y
17-0-2	Y	18-0-1	Y	18-0-5	Y	19-3-1	Y
17-0-3	Y	18-0-2	Y	18-2-1	Y	27-0-1	Y

[1]: Rules required for documentation.

[2] **Y**: Violation of the MISRA rule can be indicated
N: Violation of the MISRA rule can not be indicated
X: Rules, that during static analysis are difficult to check but that will be checked if possible.

[3]: Messages will be indicated for unused variable which is neither global variable nor member variable.

[4]: Messages will be indicated for the source code that it maybe executed, while even be deleted, no effect to the function will be produced.

[5]: Messages will be indicated for unused static function.

[6] Messages will be indicated for unused parameter.

[7]: Messages will be indicated for descriptions which is not compliant with C++ 2003(ISO/ICE 14882:2003).

[8]: Messages will be indicated for duplicating typedef name.

[9]: Messages will be indicated for duplicating class name, struct name, union name and enum name.

[10]: Messages will be indicated for defining identifiers with the same name outside and inside the function.

[11]: Messages will be indicated for defining the external variables with the same name of a type in a file.

[12]: Messages will be indicated for using global static variable only in one function.

[13]: Messages will be indicated for comparing signed value and unsigned value.

[14]: Messages will be indicated for assigning the result of integer operation(+,-,*,/)to the floating type or comparing it with the floating type.

[15]: Messages will be indicated if the comparison is made between the addresses of objects with different types.

[16]: Messages will be indicated if the operand of bit operation is non-integer type.

[17]: Messages will be indicated if an irrelevant variable exists in the loop condition of for statement.

[18]: Messages will be indicated if an assemble statement is used.

[19]: Messages will be indicated if the address of non-const member which has a more strict access limit is used as the return value.

[20]: Messages will be indicated if a new expression or a delete expression is written out of the try block in constructor or destructor.

[21]: Messages will be indicated in the following cases:

- Macro function was defined.
- The integer constant has been defined for Macro.