DATE: □□ □□ □□□□

MISRA

✓ 127 guidelines, 93 required, 34 advisory.
✓ size of integer may vary but int16 is always 16 bits.

ISO 9899:1990 — programming-C
C language dependent on compiles
31 characters significance & case sensitivity for external identifiers.
31 = 6 + 25

#define NOP asm("NOR")
Assembly language need to be inline either
a) assembles functions b) C functions c) Macros.

/*__*/, // need not be Mixed.

```
/*
!        ⇒ Allowed.
*/
```

to comment out section of code, #if (0) #ifdef
Constructs with a comment.

All uses of #pragma directive shall be documented & explained - Supporting description to demonstrate that behaviours & implications for application are well documented

Structure of bit fields declared specifically or used (9)
conjunction with ~~bit fie~~ Unions

(2.)

DATE: ☐☐ ☐☐ ☐☐☐☐

Escape Sequences defined in ISO C Standard shall be used.

External Ids — distinct 1st 6 characters

Identifiers in an inner scope shall not use the same name as an identifier in an outer scope — avoids confusion

No reuse of typedef allowed. typedef shall be a unique Id.

A 'tagname' shall be a unique identifier. No tag name shall be reused either to define a different tag or any other purpose: Type definitions is Made in header file, and that header file is included in Multiple Sources, this rule is not violated.

NO identifier in one name Space should have same Spelling as an Identifier in another name Space, with exceptions of Structure & Unions.

```
typedef struct vector { int16 a; int16 b; } vector;
                        ^^                   ^^
```

Rule violation

NO Identifier name should be reused, regardless of Scope, no id should be reused across any files in the System.

Octal constants & Octal escape Sequences shall not be used.

DATE: ☐☐ ☐☐ ☐☐☐☐

8.1. Functions shall have proto declaration and visible at both the func definition & call.

8.3. Each function parameters the type given in the declaration and definition shall be identical, return types also identical.

8.5. No definition of objects/functions in a header file

8.5. Headers should include:
* typedefs
* Macros
* func declaration
* objects

8.6. Func. declared at file scope
8.7. Objects shall be defined at block scope if they are only accessed from within a single function
8.8. An external obj/function declared in 1 & only one file
8.9. and have exactly one external definition; at file scope
8.10. All declaration & definition of functions shall have
8.11 internal linkage unless external linkage is required.

If a variable/function is called elsewhere within the same file, use static. Keyword.
8.12. When array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialization.

DATE: ☐☐☐☐☐☐

9.1. All Auto variables shall have been assigned before being used. (they are not automatically initialised)

All Static variables are initialised to zero by default however explicit declaration, initialisation also allowed.

9.2. Braces shall be used to indicate & match the structure in the non-zero initialisation of arrays & structures.

Note: All elements of array / structure can be initialised (to zero / NULL) by giving explicit initialises for the first element (nested braces need not be used)

9.3.
Use "=" in enum for first Member or explicitly initialise all.
If no explicit initialisation of Members, then C allocates a sequence of integers starting from 0 & increasing by 1.

10.1 & 10.2. Return expression or function arguments shall not be Implicitly converted to another type.
Also,
b/w Signed & unsigned types
b/w Integers & floating types
from Wide to narrow types
of Complex expressions

10.6. "U" shall be applied to all constants of unsigned type.

(5)

DATE: ☐☐ ☐☐ ☐☐☐☐

Signedness of Constant should be explicit.

Pointer types shall be classified as follows:
pointer to object, pointer to function, pointer to void, null pointer constant.

11.3
A cast should not performed b/w a pointer type and an Integral type.

A cast should not be performed b/w a pointer to object type and a different pointer to object type

```
uint8_t *p1;
uint32_t *p2;
p2 = (uint32_t*) p1 // not compatible
```

11.5 A cast shall not be performed that removes any const or volatile qualification from type addressed by a pointer.

```
uint16_t x;
uint16_t * const cpi = &x;
uint16_t * const * pcpi;
const uint16_t **ppci;
uint16 **ppi;
const uint16_t *pci;
volatile uint16_t *pvi;
uint16_t *pi;
pi = cpi; /* yes */
pi = (uint16*) pci // no
```

DATE:

ppi = (uint16_t *) pcpi; /NO/
ppi = (uint16_t *) ppci /NO/

12.1 NO parantheses required for right hand operand of an assignment operator ('=') unless the right hand side itself contains an assignment expression

12.2 Apart from few operators ((), ,, ?:, &&, ||) the order in which sub expressions are evaluated is unspecified & can vary. this means no reliance can be placed on the order of evaluation of sub-expressions

12.3 sizeof() operator shall not be used on expressions that contain side effects.

12.5 Operands of ||, && shall be primary expressions

primary expressions — single identifier or constant (or) a paranthesized expression

If an operand is other than single ~~operand~~ identifier or constant, then it must be paranthesized.

if(x || y || z) // allowed if x, y, z are boolean

12.6 Operands of a logical (&&, ||, !) should be effectively boolean.

(7)

DATE: □□ □□ □□□□

Expressions that are effectively boolean should not be used as operands other than &&, ||, !.

12-7 Bitwise operators (~, <<, >>, &, ^, |) shall not be applied to operands whose underlying type is signed.
because sign bit may be affected

12-8 right hand operand of a shift operator shall lie between 0 and 1 less than width of bits of underlying type of the left hand operand.

```
u8a = (uint8_t)(u8a << 7); // yes
"   "   ("  << 9) // no
u16a = (uint16_t)(uint16_t u8 << 9); // yes
```

12-13 The increment (++), decrement (--) should not be mixed with other operators in an expression

13-1 Assignment operators shall not be used in expressions that yield a boolean value

13-2 tests of a value against 0 should be explicit unless operand is boolean.

13-6 Numeric variables being used within a for loop for iteration counting shall not be modified in the body of the loop

14-1 No unreachable code allowed

14-4, 14-5 goto, continue shall not be used.

14.6 for any iteration statement, there shall be at most one break statement used for loop termination.

DATE: □□ □□ □□□□

14.7 A function shall have a single point of exit at the end of the function

14.8 the statement forming the body of a switch, while, do___while, for, shall be compound statement and must be in braces.

14.10 All if___else if shall be terminated with an else clause
In case of a simple 'if' statement, 'else' statement need not be included.

15.1 the final clause of switch statements shall be "default"

15.2 last statement in every switch clause shall be a break statement.

15.3 A switch expression shall not represent a value that is effectively Boolean.

15.4 switch (x==0) // not correct.

16.1 Recursive function calls cannot be used (directly/indirectly)

16.3 Identifiers shall be given for all parameters in a function prototype declaration & ids in declaration & definition shall be identical

16.7 A pointer parameter in a function prototype should be declared as pointer to const if pointer is not used to modify the addressed object.

DATE: □□ □□ □□□□

The 'const' qualification should be applied to the object pointed to, not to the pointer, since it is the object itself that is being protected.

17.1 Pointer Arithmetic is not allowed

17.5 The declaration of objects should ~~not~~ contain no more than 2 levels of pointer indirection

18.4 Unions shall not be Used.

19.1 #Include statements should only be preceded by other preprocessor directives or comments

19.4 C Macros shall only expand to a braced initialiser, a constant, a parenthesed expression, a type qualifier, a storage class specifier (or) a do-while zero construct.

No language redefinition allowed with #define.

Allowed

```
#define PI 3.14F
#define XSTAL 100000
#define CLOCK (XSTAL/16)
#define PLUS2(X) ((X) + 2)
#define STOR extern /storage class/
#define INIT(value) {(value), 0, 0}
                    /braced initialiser/
#define READ_TIME_U32() \
    do { \
        time_now = (uint32_t) TIMER_HI
5.      time_now = time_now | (uint32_t) TIMER_LO;
```

(10) }while(0)

DATE: □□ □□ □□□□

19.5 NO #defines or #undefs inside code blocks or source file.

19.6 #undef shall not be used

19.7 A function should be used in preference to a function-like Macro.

19.9 Arguments to a function-like Macro shall not contain tokens that look like preprocessing directives

19.11 All Macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives.

#if x < 0 /** assumed to be zero if not defined */.

19.14 Only permissible forms for defined preprocessor.
defined (identifier)
defined identifier

20.4 dynamic heap Memory allocation shall not be used.

20.5 errno shall not be used.

20.6, 20.9 #include <stdio.h> Shall not be used in production code – TBR

21.1 Runtime failures analyzed using
1) static analyzers. 2) dynamic profiling
3) explicit coding checks to handle run-time failures