1 Introduction

1.1 Terms and Definitions

A quick reference with explanations of terms used in this document is provided in Table 1. Note that the definitions are often C programming language specific and might not be true in other contexts.

Term	Definition
allocated storage duration	Object with dynamic storage duration, the storage is created and destroyed upon request.
automatic storage duration	Object with storage duration inside a block only, from the point of declaration to the end of the block.
block	A range of statements enclosed between a pair of braces.
declaration	Statement that introduces an identifier and its type.
identifier	A name given to an entity such as variables, functions, structures, et cetera.
object	A location in memory whose content represent a value.
statement	Instruction to a computer, in C typically a single line expression followed by a semicolon.
static storage duration	Storage duration from start to end of program execution.
storage duration	Determines an objects lifetime. There are three storage durations; allocated, auto, and static.
storage class specifier	Specifies storage duration and linkage of objects. There are five specifiers; auto, extern, register, static, and _Thread_local.
type	Variable classification such as; int, char, double, et cetera, that determines storage size and how the bit pattern stored shall be interpreted.
type qualifier	Adds attributes to types at the point of declaration. There are four type qualifiers; const, restrict, volatile, and _Atomic.

Table 1: Term and definition quick reference

2 Rules

2.1 Initialize Automatic Variables

Variables with automatic storage duration shall be initialized before use.

2.1.1 Reasoning

The value of an automatic variable that is not initialized is undefined. Explicit initialization shall hence be done before using the variable, either when declaring the variable or just before the variable is used for the first time.

Non-Compliant

```
int foo(void)
{
    uint32_t i;
    i++;
    return i;
}
```

Compliant

```
int foo(void)
{
    uint32_t i = 0;
    i++;
    return i;
}
```

2.2 Initialize Constant Variables When Declared

Declaration of variables with const type qualifier shall include initialization.

2.3 Strict Enumeration Initialization

Strategy used for initialization of the members in an enum type shall be one of the following: not specifying any values, specifying all values, specifying only the first value.

2.3.1 Reasoning

Minimizes the risk that a pair of members is assigned the same value by mistake.

Non-Compliant

```
enum nonCompliantEnum
{
    NC_E1 = 1,
    NC_E2,
    NC_E3 = 2
};
```

Compliant

```
enum compliantEnum_1
{
    C1_E1,
    C1_E2,
    C1_E3
};

enum compliantEnum_2
{
    C2_E1 = 1,
    C2_E2,
    C2_E3
};

enum compliantEnum_3
{
    C3_E1 = 1,
    C3_E2 = 3,
    C3_E3 = 4
};
```

2.4 No Usage of Restrict

Do not use the restrict type qualifier.

2.4.1 Reasoning

The keyword restrict is type qualifier that can be added in a object pointer declaration. It provides a hint to the compiler that only this pointer will be used access the object. This will in some situations make it possible for the compiler to generate a more optimized result. The behavior of the code will be undefined if this guarantee is not meet. Using restrict burdens the design of the code to guarantee that the memory areas do not overlap and adds a risk, the restrict type qualifier shall hence not be used.

2.4.2 Examples

Non-Compliant

```
void g(int *restrict a,
       int *restrict b,
       int *restrict x)
{
  *a += *x;
  *b += *x;
}
```

Compliant

```
void f(int *a,
       int *b,
       int *x)
{
  *a += *x;
  *b += *x;
}
```

2.5 Use Unsigned Type For Bit Manipulation

Operands of bitwise operation shall be of unsigned integer type.

2.5.1 Reasoning

The result of bitwise operations on signed integers are implementation-defined.

2.6 Preserve Type Qualifications for Pointers

Rule 1

The declaration of a function that does not take any parameters shall use the void type parameter.

Reasoning

A C function declaration with an empty parameter list is not the same as that the function has no parameters, it is an obsolete way to declare a function without needing to explicitly specify the number, and types of parameters. Using void states explicitly that a function does not takes any parameters, making it possible for the compiler to check for conflicts in the function usage.

Examples

Non-Compliant

```
void f();
```

Compliant

```
void f(void);
```

Rule 2

Use enum to define related constants.

Examples

Non-Compliant

```
#define UP      0
#define RIGHT   1
#define DOWN    2
#define LEFT    3
```

Compliant

```
typedef enum eDirection
{
    UP,
    RIGHT,
    DOWN,
    LEFT
} direction;
```

Reasoning

Defining related constants as enum type, as opposed to a group of preprocessor defines comes with various benefits. Makes it possible to have automated error checks for when the constant is used in the wrong context. Simplifies debugging due to that there will be a symbol for each enum constant in the debugger symbol table.