DCL23-C. Guarantee that mutually visible identifiers are unique

According to subclause 6.2.7 of the C Standard [ISO/IEC 9899:2011],

> *All declarations that refer to the same object or function shall have compatible type; otherwise, the behavior is undefined.*

(See also undefined behavior 15 of Annex J.)

Further, according to subclause 6.4.2.1,

> *Any identifiers that differ in a significant character are different identifiers. If two identifiers differ only in nonsignificant characters, the behavior is undefined.*

(See also undefined behavior 31 of Annex J.)

Identifiers in mutually visible scopes must be deemed unique by the compiler to prevent confusion about which variable or function is being referenced. Implementations can allow additional nonunique characters to be appended to the end of identifiers, making the identifiers appear unique while actually being indistinguishable.

It is reasonable for scopes that are not visible to each other to have duplicate identifiers. For example, two functions can each have a local variable with the same name because their scopes cannot access each other. But a function's local variable names should be distinct from each other as well as from all static variables declared within the function's file (and from all included header files.)

To guarantee that identifiers are unique, the number of significant characters recognized by the most restrictive compiler used must be determined. This assumption must be documented in the code.

The standard defines the following minimum requirements:

- 63 significant initial characters in an internal identifier or a macro name. (Each universal character name or extended source character is considered a single character.)
- 31 significant initial characters in an external identifier. (Each universal character name specifying a short identifier of 0000FFFF or less is considered 6 characters; each universal character name specifying a short identifier of 00010000 or more is considered 10 characters; and each extended source character, if any exist, is considered the same number of characters as the corresponding universal character name.)

Restriction of the significance of an external name to fewer than 255 characters in the standard (considering each universal character name or extended source character as a single character) is an obsolescent feature that is a concession to existing implementations. As a result, it is not necessary to comply with this restriction as long as the identifiers are unique and the assumptions concerning the number of significant characters are documented.

Noncompliant Code Example (Source Character Set)

On implementations that support only the minimum requirements for significant characters required by the standard, this code example is noncompliant because the first 31 characters of the external identifiers are identical:

```
extern int *global_symbol_definition_lookup_table_a;
extern int *global_symbol_definition_lookup_table_b;
```

Compliant Solution (Source Character Set)

In a compliant solution, the significant characters in each identifier must differ:

```
extern int *a_global_symbol_definition_lookup_table;
extern int *b_global_symbol_definition_lookup_table;
```

Noncompliant Code Example (Universal Character Names)

In this noncompliant code example, both external identifiers consist of four universal character names. Because the first three universal character names of each identifier are identical, both identify the same integer array on implementations that support only the minimum requirements for significant characters required by the standard:

```
extern int *\U00010401\U00010401\U00010401\U00010401;
extern int *\U00010401\U00010401\U00010401\U00010402;
```

Compliant Solution (Universal Character Names)

For portability, the first three universal character name combinations used in an identifier must be unique:

```
extern int *\U00010401\U00010401\U00010401\U00010401;
extern int *\U00010402\U00010401\U00010401\U00010401;
```

Risk Assessment

Nonunique identifiers can lead to abnormal program termination, denial-of-service attacks, or unintended information disclosure.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
DCL23-C	Medium	Unlikely	Low	P6	L2

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04		Supported indirectly via MISRA C:2012 Rules 5.1, 5.2, 5.3, 5.4 and 5.5.
Axivion Bauhaus Suite	6.9.0	**CertC-DCL23**	
CodeSonar	5.1p0	**LANG.ID.ND.EXT** **LANG.ID.ND.MM** **LANG.ID.ND.MO** **LANG.ID.ND.NEST** **LANG.ID.ND.SS** **LANG.ID.NU.EXT** **LANG.ID.NU.INT** **LANG.ID.NU.LIBFN** **LANG.ID.NU.TAG** **LANG.ID.NU.TYPE** **LANG.STRUCT.DECL.MGT**	Non-distinct identifiers: external names Non-distinct identifiers: macro/macro Non-distinct identifiers: macro/other Non-distinct identifiers: nested scope Non-distinct identifiers: same scope Non-unique identifiers: external name Non-unique identifiers: internal name Library Function Override Non-unique identifiers: tag Non-unique identifiers: typedef Global variable declared with different types
Compass/ROSE			Can detect some violations of this rule but cannot flag violations involving universal names
Klocwork	2018	**MISRA.IDENT.DISTINCT.C99.2012**	
LDRA tool suite	9.7.1	**17 D** **355 S** **61 X**	Fully implemented
Polyspace Bug Finder	R2019a	CERT C: Rec. DCL23-C	Checks for: • External identifiers not distinct • Identifiers in same scope and namespace not distinct • Macro identifier not distinct • Name for macros and identifiers not distinct Rec. partially covered.
PRQA QA-C	9.5	**0627, 0776, 0777, 0778,** **0779, 0789, 0791, 0793**	Partially implemented
RuleChecker	19.04		Supported indirectly via MISRA C:2012 Rules 5.1, 5.2, 5.3, 5.4 and 5.5.
SonarQube C/C++ Plugin	3.11	**IdentifierLongerThan31**	

Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the CERT website.

Related Guidelines

ISO/IEC TR 24772:2013	Choice of Clear Names [NAI] Identifier Name Reuse [YOW]

MISRA C:2012	Rule 5.1 (required) Rule 5.2 (required) Rule 5.3 (required) Rule 5.4 (required) Rule 5.5 (required)

Bibliography

[ISO/IEC 9899:2011]	Subclause 6.2.7, "Compatible Type and Composite Type" Subclause 6.4.1, "Keywords"