DCL01-C. Do not reuse variable names in subscopes

Do not use the same variable name in two scopes where one scope is contained in another. For example,

- No other variable should share the name of a global variable if the other variable is in a subscope of the global variable.
- A block should not declare a variable with the same name as a variable declared in any block that contains it.

Reusing variable names leads to programmer confusion about which variable is being modified. Additionally, if variable names are reused, generally one or both of the variable names are too generic.

Noncompliant Code Example

This noncompliant code example declares the `msg` identifier at file scope and reuses the same identifier to declare a character array local to the `report_error()` function. The programmer may unintentionally copy the function argument to the locally declared `msg` array within the `report_error()` function. Depending on the programmer's intention, it either fails to initialize the global variable `msg` or allows the local `msg` buffer to overflow by using the global value `msgsize` as a bounds for the local buffer.

```c
#include <stdio.h>
static char msg[100];
static const size_t msgsize = sizeof( msg);

void report_error(const char *str) {
    char msg[80];
    snprintf(msg, msgsize, "Error: %s\n", str);
    /* ... */
}

int main(void) {
    /* ... */
    report_error("some error");
    return 0;
}
```

Compliant Solution

This compliant solution uses different, more descriptive variable names:

```c
#include <stdio.h>
static char message[100];
static const size_t message_size = sizeof( message);

void report_error(const char *str) {
    char msg[80];
    snprintf(msg, sizeof( msg), "Error: %s\n", str);
    /* ... */
}

int main(void) {
    /* ... */
    report_error("some error");
    return 0;
}
```

When the block is small, the danger of reusing variable names is mitigated by the visibility of the immediate declaration. Even in this case, however, variable name reuse is not desirable. In general, the larger the declarative region of an identifier, the more descriptive and verbose should be the name of the identifier.

By using different variable names globally and locally, the compiler forces the developer to be more precise and descriptive with variable names.

Noncompliant Code Example
This noncompliant code example declares two variables with the same identifier, but in slightly different scopes. The scope of the identifier \texttt{i} declared in the \texttt{for} loop's initial clause terminates after the closing curly brace of the \texttt{for} loop. The scope of the identifier \texttt{i} declared in the \texttt{for} loop's compound statement terminates before the closing curly brace. Thus, the inner declaration of \texttt{i} hides the outer declaration of \texttt{i}, which can lead to unintentionally referencing the wrong object.

```c
void f(void) {
    for (int i = 0; i < 10; i++) {
        long i;
        /* ... */
    }
}
```

**Compliant Solution**

This compliant solution uses a unique identifier for the variable declared within the \texttt{for} loop.

```c
void f(void) {
    for (int i = 0; i < 10; i++) {
        long j;
        /* ... */
    }
}
```

**Exceptions**

**DCL01-C-EX1**: A function argument in a function declaration may clash with a variable in a containing scope provided that when the function is defined, the argument has a name that clashes with no variables in any containing scopes.

```c
extern int name;
void f(char *name); /* Declaration: no problem here */
/* ... */
void f(char *arg) { /* Definition: no problem; arg doesn't hide name */
    /* Use arg */
}
```

**DCL01-C-EX2**: A temporary variable within a new scope inside of a macro can override a surrounding identifier.

```c
#define SWAP(type, a, b) do { type tmp = a; a = b; b = tmp; } while(0)
void func(void) {
    int tmp = 100;
    int a = 10, b = 20;
    SWAP(int, a, b); /* Hidden redeclaration of tmp is acceptable */
}
```

**Risk Assessment**

Reusing a variable name in a subscope can lead to unintentionally referencing an incorrect variable.

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCL01-C</td>
<td>Low</td>
<td>Unlikely</td>
<td>Medium</td>
<td>P2</td>
<td>L3</td>
</tr>
</tbody>
</table>

**Automated Detection**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astrée</td>
<td>19.04</td>
<td></td>
<td>Supported indirectly via MISRA C:2012 Rule 5.3.</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-DCL01</td>
<td></td>
</tr>
<tr>
<td>Tool</td>
<td>Version</td>
<td>Rule</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CodeSonar</td>
<td>5.2p0</td>
<td>LANG.ID.ND.NEST</td>
<td>Non-distinct identifiers: nested scope</td>
</tr>
<tr>
<td>Compass/ROSE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECLAIR</td>
<td>1.2</td>
<td>CC2.DCL01</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Klocwork</td>
<td>2018</td>
<td>MISRA.VAR.HIDDEN</td>
<td></td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>131 S</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-DCL01-a</td>
<td>Identifier declared in a local or function prototype scope shall not hide an identifier declared in a global or namespace scope</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CERT_C-DCL01-b</td>
<td>Identifiers declared in an inner local scope should not hide identifiers declared in an outer local scope</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C, Rec. DCL01-C</td>
<td>Checks for variable shadowing (rule fully covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>0795, 0796, 2547, 3334</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>PVS-Studio</td>
<td>7.07</td>
<td>V561, V688, V703, V711</td>
<td></td>
</tr>
<tr>
<td>RuleChecker</td>
<td>19.04</td>
<td></td>
<td>Supported indirectly via MISRA C:2012 Rule 5.3.</td>
</tr>
<tr>
<td>Splint</td>
<td>3.1.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Related Vulnerabilities**

Search for vulnerabilities resulting from the violation of this rule on the [CERT website](https://csrc.nist.gov/).  

**Related Guidelines**

<table>
<thead>
<tr>
<th>Standard</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEI CERT C++ Coding Standard</td>
<td>VOID DCL01-CPP. Do not reuse variable names in subscopes</td>
</tr>
<tr>
<td>MISRA C:2012</td>
<td>Rule 5.3 (required)</td>
</tr>
</tbody>
</table>