PRE07-C. Avoid using repeated question marks

Two consecutive question marks signify the start of a trigraph sequence. According to the C Standard, subclause 5.2.1.1 [ISO/IEC 9899:2011].

![Trigraph sequences table]

Noncompliant Code Example

In this noncompliant code example, `a++` is not executed because the trigraph sequence `??/` is replaced by `\.`, logically putting `a++` on the same line as the comment:

```c
// What is the value of a now??/
a++;  
```

Compliant Solution

This compliant solution eliminates the accidental introduction of the trigraph by separating the question marks:

```c
// What is the value of a now? ?/
a++;  
```

Noncompliant Code Example

This noncompliant code example includes the trigraph sequence `??!`, which is replaced by the character `|`:

```c
size_t i = /* Some initial value */;
if (i > 9000) {
  if (puts("Over 9000!??!") == EOF) {
    /* Handle error */
  }
}
```

This example prints `Over 9000!|` if a C-compliant compiler is used.

Compliant Solution

This compliant solution uses string concatenation to concatenate the two question marks; otherwise, they are interpreted as beginning a trigraph sequence:

```c
size_t i = /* Some initial value */;
/* Assignment of i */
if (i > 9000) {
  if (puts("Over 9000!?"?!") == EOF) {
    /* Handle error */
  }
}
```

This code prints `Over 9000!??!`, as intended.

Risk Assessment
Inadvertent trigraphs can result in unexpected behavior. Some compilers provide options to warn when trigraphs are encountered or to disable trigraph expansion. Use the warning options, and ensure your code compiles cleanly. (See MSC00-C. Compile cleanly at high warning levels.)

<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>PRE07-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>trigraph</td>
<td>Fully checked</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-PRE07</td>
<td></td>
</tr>
<tr>
<td>ECLAIR</td>
<td>1.2</td>
<td>CC2.PRE07</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>GCC</td>
<td>4.3.5</td>
<td></td>
<td>Can detect violation of this recommendation when the -Wtrigraphs flag is used</td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>81 S</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-PRE07-a</td>
<td>Trigraphs shall not be used</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C: Rec. PRE07-C</td>
<td>Checks for use of trigraphs (rec. fully covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>3601</td>
<td>Partially implemented</td>
</tr>
<tr>
<td>RuleChecker</td>
<td>19.04</td>
<td>trigraph</td>
<td>Fully checked</td>
</tr>
<tr>
<td>SonarQube C/C++ Plugin</td>
<td>3.11</td>
<td>TrigraphUsage</td>
<td></td>
</tr>
</tbody>
</table>

**Related Vulnerabilities**

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

**Related Guidelines**

<table>
<thead>
<tr>
<th>SEI CERT C++ Coding Standard</th>
<th>VOID PRE07-CPP. Avoid using repeated question marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISRA C:2012</td>
<td>Rule 4.2 (advisory)</td>
</tr>
</tbody>
</table>

**Bibliography**