ToiletPaper #56

jambit
WHERE INNOVATION WORKS

Undefiniertes Verhalten in C, Objective C und C++

Autor: Andreas Maier / Software Architect / Business Division Banking & Insurance

✘ Problem

Was ist das Ergebnis von `i = ++i + i++`?

Es ist alles möglich: Der Standard stellt keine Anforderungen. Möglicherweise kompiliert das Programm nicht, wird falsch ausgeführt (stürzt ab oder erzeugt unbemerkt falsche Ergebnisse) oder es tut vielleicht genau das, was der Programmierer möchte. **Das genaue Verhalten ist undefiniert!**

Das resultierende Verhalten hängt nicht nur von der spezifischen Hardware, Plattform und dem Compiler ab, sondern auch von den spezifischen Einstellungen (Optimierungsstufe, Debug- / Releaseversion) desselben Compilers. Dies führt zu nicht portierbarem Code und Fehlern, die schwer zu erkennen sind.

✓ Lösung

Um undefiniertes Verhalten zu beseitigen, sollte man das **Konzept des Sequenzpunkts** verstehen. Der Standard in C definiert je einen Sequenzpunkt

1. am Ende jedes vollen Ausdrucks (typischerweise am Semikolon `;`)
2. nach der Auswertung aller Funktionsargumente und vor der Ausführung des Funktionskörpers
3. nach der Auswertung des Ausdrucks `a` und unter Nutzung des eingebauten (nicht überladenen) Operatoren

`a && b`

`a || b`

`a? b : c`

`a , b`

Es gibt zwei Hauptfälle mit undefiniertem Verhalten **zwischen zwei Sequenzpunkten**

1.
```
i = i++ + 1;        // undefined behavior
i = ++i + 1;        // undefined behavior (well-defined in
C++11) ++++i;              // undefined behavior (well-defined
in C++11) f(++i, ++i);       // undefined behavior f(i = -1, i
= -1); // undefined behavior
```

Lösung: Zwischen zwei Sequenzpunkten soll der Wert eines Skalars **höchstens einmal modifiziert** werden.

2.
```
cout << i << i++; // undefined behavior
a[i] = i++;       // undefined behavior
```

Lösung: Zwischen zwei Sequenzpunkten soll nur auf den vorherigen Wert eines modifizierten Skalars **zugegriffen werden, um den zu speichernden Wert zu bestimmen.**

Weitere Aspekte

- **Compiler-Warnungen** aktivieren und ernst nehmen
- Verwendung **statischer Analysetools** (wie Clang, cppcheck usw.), um noch mehr Warnungen zu erhalten