Sequence point

From Wikipedia, the free encyclopedia

A **sequence point** defines any point in a computer program's execution at which it is guaranteed that all side effects of previous evaluations will have been performed, and no side effects from subsequent evaluations have yet been performed. A sequence point is a point in program execution at which all side effects are evaluated before going on to the next step. They are often mentioned in reference to C and C++, because they are a core concept for determining the validity and, if valid, the possible results of expressions. Adding more sequence points is sometimes necessary to make an expression defined and to ensure a single valid order of evaluation.

With C++11, usage of the term sequence point has been replaced by sequencing. There are three possibilities:[1][2][3]

1. An expression's evaluation can be **sequenced before** that of another expression, or equivalently the other expression's evaluation is **sequenced after** that of the first.
2. The expressions' evaluation is **indeterminately sequenced,** meaning one is sequenced before the other, but which is unspecified.
3. The expressions' evaluation is **unsequenced.**

The execution of unsequenced evaluations can overlap, with catastrophic undefined behavior if a write to an object is unsequenced with regard to another access to the same.

Examples of ambiguity

Consider two functions `f()` and `g()`. In C and C++, the `+` operator is not associated with a sequence point, and therefore in the expression `f()+g()` it is possible that either `f()` or `g()` will be executed first. The comma operator introduces a sequence point, and therefore in the code `f(),g()` the order of evaluation is defined: first `f()` is called, and then `g()` is called.

Sequence points also come into play when the same variable is modified more than once within a single expression. An often-cited example is the C expression `i=i++`, which apparently both assigns `i` its previous value and increments `i`. The final value of `i` is ambiguous, because, depending on the order of expression evaluation, the increment may occur before, after, or interleaved with the assignment. The definition of a particular language might specify one of the possible behaviors or simply say the behavior is undefined. In C and C++, evaluating such an expression yields undefined behavior.[4] Other languages, such as C# may define the precedence of the assignment and increment operator in

such a way that the result of the expression `i=i++` is guaranteed.

Sequence points in C and C++

In C[5] and C++,[6] sequence points occur in the following places. (In C++, overloaded operators act like functions, and thus operators that have been overloaded introduce sequence points in the same way as function calls.)

1. Between evaluation of the left and right operands of the && (logical AND), || (logical OR) (as part of short-circuit evaluation), and comma operators. For example, in the expression `*p++ != 0 && *q++ != 0`, all side effects of the sub-expression `*p++ != 0` are completed before any attempt to access `q`.
2. Between the evaluation of the first operand of the ternary "question-mark" operator and the second or third operand. For example, in the expression `a = (*p++) ? (*p++) : 0` there is a sequence point after the first `*p++`, meaning it has already been incremented by the time the second instance is executed.
3. At the end of a full expression. This category includes expression statements (such as the assignment `a=b;`), return statements, the controlling expressions of `if`, `switch`, `while`, or `do-while` statements, and all three expressions in a `for` statement.
4. Before a function is entered in a function call. The order in which the arguments are evaluated is not specified, but this sequence point means that all of their side effects are complete before the function is entered. In the expression `f(i++) + g(j++) + h(k++)`, `f` is called with a parameter of the original value of `i`, but `i` is incremented before entering the body of `f`. Similarly, `j` and `k` are updated before entering `g` and `h` respectively. However, it is not specified in which order `f()`, `g()`, `h()` are executed, nor in which order `i`, `j`, `k` are incremented. Variables `j` and `k` in the body of `f` may or may not have been already incremented. Note that a function call `f(a,b,c)` is not a use of the comma operator and the order of evaluation for `a`, `b`, and `c` is unspecified.
5. At a function return, after the return value is copied into the calling context. (This sequence point is only specified in the C++ standard; it is present only implicitly in C.[7])
6. At the end of an initializer; for example, after the evaluation of `5` in the declaration `int a = 5;`.
7. Between each declarator in each declarator sequence; for example, between the two evaluations of `a++` in `int x = a++, y = a++`.[8] Note that this is not an example of the comma operator.
8. After the action associated with input/output conversion format specifier. For example, in the expression `printf("foo %n %d", &a, 42)`, there is a sequence point after the `%n` is evaluated before printing `42`.

References

1. "ISO/IEC 14882:2011". Retrieved 2012-07-04.

2. "A finer-grained alternative to sequence points (revised) (WG21/N2239 J16/07-0099)". Retrieved 2012-07-05.
3. "Order of evaluation". Retrieved 2015-10-14.
4. Clause 6.5#2 of the C99 specification: "*Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression. Furthermore, the prior value shall be accessed only to determine the value to be stored.*"
5. Annex C of the C99 specification lists the circumstances under which a sequence point may be assumed.
6. The 1998 C++ standard lists sequence points for that language in section 1.9, paragraphs 16–18.
7. C++ standard, ISO 14882:2003, section 1.9, footnote 11.
8. C++ standard, ISO 14882:2003, section 8.3: "*Each init-declarator in a declaration is analyzed separately as if it was in a declaration by itself.*"

- Question 3.8 (http://c-faq.com/expr/seqpoints.html) of the FAQ for comp.lang.c

Categories: C (programming language) | C++ | Programming paradigms

---