WIKIPEDIA

Expression (computer science)

An **expression** in a programming language is a combination of one or more explicit values, constants, variables, operators, and functions that the programming language interprets (according to its particular rules of precedence and of association) and computes to produce ("to return", in a stateful environment) another value. This process, as for mathematical expressions, is called evaluation.

In simple settings, the resulting value is usually one of various primitive types, such as numerical, string, and logical; in more elaborate settings, it can be an arbitrary complex data type.

For example, 2+3 is an arithmetic and programming expression which evaluates to 5. A variable is an expression because it denotes a value in memory, so y+6 is an expression. An example of a relational expression is 4≠4, which evaluates to `false`.[1][2]

Contents

Void as a result type

In C and most C-derived languages, a call to a function with a void return type is a valid expression, of type void.[3] Values of type void cannot be used, so the value of such an expression is always thrown away.

Side effects and elimination

In many programming languages a function, and hence an expression containing a function, may have side effects. An expression with side effects does not normally have the property of referential transparency. In many languages (e.g. C++), expressions may be ended with a semicolon (;) to turn the expression into an expression statement. This asks the implementation to evaluate the expression for its side-effects only and to disregard the result of the expression (e.g. "x+1;") unless it is a part of an expression statement that induces side-effects (e.g. "y=x+1;" or "func1(func2());").

Caveats

Note that the formal notion of a side effect is a change to the abstract state of the running program.

Another class of side effects are changes to the concrete state of the computational system, such as loading data into cache memories. Languages which are often described as "side effect–free" will generally still have concrete side effects which can be exploited, for example, in side-channel attacks.

Furthermore, the elapsed time evaluating an expression (even one with no other apparent side effects), is sometimes essential to the correct operation of a system, as behaviour in time is easily visible from outside the evaluation environment by other parts of the system with which it interacts, and might even be regarded as the *primary* effect such as when performing benchmark testing.

It depends on the particular programming language specification whether an expression with no abstract side effects can legally be eliminated from the execution path by the processing environment in which the expression is evaluated.

See also

- Statement (computer science) (contrast)
- Boolean expression
- Expression (mathematics)
- Evaluation strategy

References

1. Javascript expressions, Mozilla (https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide/Expressions) Accessed July 6, 2009
2. Programming in C (https://www.cs.drexel.edu/~rweaver/COURSES/ISTC-2/TOPICS/expr.html) Accessed July 6, 2009
3. ISO/IEC 9899:1999 (http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf) section 6.3.2.2, accessed August 31, 2009

External links

- This article is based on material taken from Expression (http://foldoc.org/Expression) at the Free On-line Dictionary of Computing prior to 1 November 2008 and incorporated under the "relicensing" terms of the GFDL, version 1.3 or later.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Expression_(computer_science)&oldid=803727099"