Function calls

A call expression calls afunction . The syntax for a call expression is theconcrete path of the function followed by a list of arguments. The list of arguments is a comma-separated list of expressions enclosed in parentheses. (,,...) Examples: cmp::max(a: 5, b: 1) get_info() increment(ref a.b) A call expression evaluates to the value returned by the function. The expression's type is the return type of the function. For example, the expressionis_le(1, 3) evaluates totrue , and its type isbool .

Arguments

The number of arguments passed should match the number of parameters defined in the function signature. The type of each argument should match the type of the corresponding parameter. There are 2 kinds of arguments: Expressions and references. A reference argument is passed to reference parameters, and an expression argument is passed to regular parameters.

Expression arguments

Any expression can be passed to a regular parameter. The type of the expression should match the type of the parameter. Example:cmp::max(a: 5, b: 1) .

Reference arguments

The syntax for a reference argument isref lvalue . The type oflvalue should match the type of the reference parameter. The behavior of passing a reference argument is similar to theassignment .lvalue is passed to the function and then reassigned to when the function returns. Example:increment(ref a.b) . fn main() { let x = 3; let mut y = A { z: 5 }; foo(x, ref y.z); }

Argument evaluation order

The arguments are evaluated from left to right, starting with all the expression arguments, and then the reference arguments. This is important, since the value of a reference argument may change during evaluation of the expression arguments.

Named arguments

Arguments can also be specified explicitly by the parameter name. For exampleinc_3(x: 7) . This is called a "named argument" and it verifies that the name of the parameter is correct, to prevent mistakes. For example, a call toinc_3(y: 7) would not compile - If you use a variable as an argument and its name is the same as the parameter name, a shorthand for writinginc_3(x: x) is to simply writeinc_3(:x) . This lets you write the name only once, but still verifies it. Here too,inc_3(:y) (assuming there is a variabley in the scope) would not compile. With a reference argument, you can writemut_inc_3(ref x:y) , ormut_inc_3(ref :x) .

7.5 Parenthesized expressions 7.7 Method calls