Structs

A struct is a collection of named fields. It is one of the ways to define custom user types. The fields of a struct, called members, can be of any defined type, including user-defined types, like other structs orenums .

Definition

A struct is defined by thestruct keyword. Here are a few examples of struct definitions: * A struct with only core type members: * rust * struct Tree { * height: u16, * number_of_leaves: u32, * } * * A struct with members of user-defined types: *rust * struct Forest { * number_of_trees: u32, * highest_tree: Tree, * lowest_tree: Tree, * } * * A struct with members of the same type: *rust * struct ListNode { * value: u8, * next: Option<Box<ListNode>>, * } *

Usage

A struct is instantiated with theStructName { member1: value1, member2: value2, … } syntax. For example: let mut tree = Tree { height: 6, number_of_leaves: 10000, }; All members must be specified in the instantiation, but their order does not matter. If a member should be initialized with a variable with the same name of the member, instead ofname: name you can simply specifyname . For example: let height = 6; let mut tree = Tree { number_of_leaves: 10000, height, }; It can then be accessed in two ways:

Deconstruction

A struct can be deconstructed with thelet struct_name { member1, member2, … } = struct_instance syntax. For example: let Tree { height, number_of_leaves } = tree; let higher_tree = Tree { height: height + 1, number_of_leaves } You can use different patterns (for example change the order and names of the new variables) in the deconstruction. Seepatterns for more information.

The Dot (.) operator:

Another way to access struct members is with the dot (.) operator. let old_tree_height = tree.height; tree.height = old_tree_height + 1; As Cairo’s memory is immutable behind the scenes, assigning to a member of a struct using the dot (.) operator actually copies the memory of the whole struct. This does not affect the correctness of your program, but it may affect its performance if the struct is big. A more expressive way to mutate a struct similar to the last example is: let old_tree_height = tree.height; tree = Tree { height: old_tree_height + 1, number_of_leaves: tree.number_of_leaves }; This way also lets you guarantee avoiding extra copies of the struct’s memory if you want to change multiple members. For example, this code might copy the struct’s memory twice (but is mostly optimized by the compiler): tree.height = 1; tree.number_of_leaves = 100; While this code has the same logical effect, but guarantees to copy the struct’s memory only once: tree = Tree { height: 1, number_of_leaves: 100 };

4.5 Impl aliases ð§ 4.7 Enums