The Move Book

Vectors are a native way to store collections of elements in Move. They are similar to arrays in other programming languages, but with a few differences. In this section, we introduce the vector type and its operations.

The vector type is written using the vector keyword followed by the type of the elements in angle brackets. The type of the elements can be any valid Move type, including other vectors.

Move has a vector literal syntax that allows you to create vectors using the vector keyword followed by square brackets containing the elements (or no elements for an empty vector).

The vector type is a built-in type in Move, and does not need to be imported from a module. However, vector operations are defined in the std::vector module, and you need to import the module to use them.

The standard library provides methods to manipulate vectors. The following are some of the most commonly used operations:

A vector of non-droppable types cannot be discarded. If you define a vector of types without the drop ability, the vector value cannot be ignored. If the vector is empty, the compiler requires an explicit call to the destroy_empty function.

The destroy_empty function will fail at runtime if you call it on a non-empty vector.

Vector syntax

The vector type is written using the vector keyword followed by the type of the elements in angle brackets. The type of the elements can be any valid Move type, including other vectors.

Move has a vector literal syntax that allows you to create vectors using the vector keyword followed by square brackets containing the elements (or no elements for an empty vector).

```bash // An empty vector of bool elements. let empty: vector = vector[];

// A vector of u8 elements. let v: vector = vector[10, 20, 30];

// A vector of vector elements. let vv: vector> = vector[ vector[10, 20], vector[30, 40] ]; ```

The vector type is a built-in type in Move, and does not need to be imported from a module. However, vector operations are defined in the std::vector module, and you need to import the module to use them.

The standard library provides methods to manipulate vectors. The following are some of the most commonly used operations:

```bash let mut v = vector[10u8, 20, 30];

assert!(v.length() == 3); assert!(!v.is_empty());

v.push_back(40); let last_value = v.pop_back();

assert!(last_value == 40); ```

A vector of non-droppable types cannot be discarded. If you define a vector of types without the drop ability, the vector value cannot be ignored. If the vector is empty, the compiler requires an explicit call to the destroy_empty function.

``bash /// A struct without`drop` ability. public struct NoDrop {}

# [test]

fun test_destroy_empty() { // Initialize a vector of `NoDrop` elements. let v = vector[];

```
// While we know that `v` is empty, we still need to call
// the explicit `destroy_empty` function to discard the vector.
v.destroy_empty();
```

} ```

The destroy_empty function will fail at runtime if you call it on a non-empty vector.

Vector operations

The standard library provides methods to manipulate vectors. The following are some of the most commonly used operations:

```bash let mut v = vector[10u8, 20, 30];

assert!(v.length() == 3); assert!(!v.is_empty());

v.push_back(40); let last_value = v.pop_back();

assert!(last_value == 40); ```

A vector of non-droppable types cannot be discarded. If you define a vector of types without the drop ability, the vector value cannot be ignored. If the vector is empty, the compiler requires an explicit call to the destroy_empty function.

``bash /// A struct withoutdrop` ability. public struct NoDrop {}

# [test]

fun test_destroy_empty() { // Initialize a vector of `NoDrop` elements. let v = vector[];

```
// While we know that `v` is empty, we still need to call
// the explicit `destroy_empty` function to discard the vector.
v.destroy_empty();
```

} ```

The destroy_empty function will fail at runtime if you call it on a non-empty vector.

## Destroying a Vector of non-droppable types

A vector of non-droppable types cannot be discarded. If you define a vector of types without the drop ability, the vector value cannot be ignored. If the vector is empty, the compiler requires an explicit call to the destroy_empty function.

``bash /// A struct withoutdrop` ability. public struct NoDrop {}

# [test]

fun test_destroy_empty() { // Initialize a vector of `NoDrop` elements. let v = vector[];

```
// While we know that `v` is empty, we still need to call
// the explicit `destroy_empty` function to discard the vector.
v.destroy_empty();
```

} ```

The destroy_empty function will fail at runtime if you call it on a non-empty vector.

Further reading
```