# Crate alloc

Since 1.36.0 ·



## The Rust core allocation and collections library

This library provides smart pointers and collections for managing heap-allocated values.

This library, like core, normally doesn't need to be used directly since its contents are re-exported in the `std` crate. Crates that use the `#![no_std]` attribute however will typically not depend on `std`, so they'd use this crate instead.

### Boxed values

The `Box` type is a smart pointer type. There can only be one owner of a `Box`, and the owner can decide to mutate the contents, which live on the heap.

This type can be sent among threads efficiently as the size of a `Box` value is the same as that of a pointer. Tree-like data structures are often built with boxes because each node often has only one owner, the parent.

### Reference counted pointers

The `Rc` type is a non-threadsafe reference-counted pointer type intended for sharing memory within a thread. An `Rc` pointer wraps a type, `T`, and only allows access to `&T`, a shared reference.

This type is useful when inherited mutability (such as using `Box`) is too constraining for an application, and is often paired with the `Cell` or `RefCell` types in order to allow mutation.

### Atomically reference counted pointers

The `Arc` type is the threadsafe equivalent of the `Rc` type. It provides all the same functionality of `Rc`, except it requires that the contained type `T` is shareable. Additionally, `Arc<T>` is itself sendable while `Rc<T>` is not.

This type allows for shared access to the contained data, and is often paired with synchronization primitives such as mutexes to allow mutation of shared resources.

### Collections

Implementations of the most common general purpose data structures are defined in this library. They are re-exported through the standard collections library.

### Heap interfaces

The `alloc` module defines the low-level interface to the default global allocator. It is not compatible with the libc allocator API.

## Modules

| | |
|---|---|
| alloc | Memory allocation APIs |
| borrow | A module for working with borrowed data. |
| boxed | The `Box<T>` type for heap allocation. |
| collections | Collection types. |
| ffi | Utilities related to FFI bindings. |
| fmt | Utilities for formatting and printing `String`s. |
| rc | Single-threaded reference-counting pointers. 'Rc' stands for 'Reference Counted'. |
| slice | Utilities for the slice primitive type. |
| str | Utilities for the `str` primitive type. |
| string | A UTF-8–encoded, growable string. |
| sync | Thread-safe reference-counting pointers. |
| task | Types and Traits for working with asynchronous tasks. |
| vec | A contiguous growable array type with heap-allocated contents, written `Vec<T>`. |
| bstr Experimental | The `ByteStr` and `ByteString` types and trait implementations. |

## Macros

| | |
|---|---|
| format | Creates a `String` using interpolation of runtime expressions. |
| vec | Creates a `Vec` containing the arguments. |