`ffi_pure`

The tracking issue for this feature is: #58329

The `#[ffi_pure]` attribute applies clang's `pure` attribute to foreign functions declarations.

That is, `#[ffi_pure]` functions shall have no effects except for its return value, which shall not change across two consecutive function calls with the same parameters.

Applying the `#[ffi_pure]` attribute to a function that violates these requirements is undefined behavior.

This attribute enables Rust to perform common optimizations, like sub-expression elimination and loop optimizations. Some common examples of pure functions are `strlen` or `memcmp`.

These optimizations are only applicable when the compiler can prove that no program state observable by the `#[ffi_pure]` function has changed between calls of the function, which could alter the result. See also the `#[ffi_const]` attribute, which provides stronger guarantees regarding the allowable behavior of a function, enabling further optimization.

Pitfalls

A `#[ffi_pure]` function can read global memory through the function parameters (e.g. pointers), globals, etc. `#[ffi_pure]` functions are not referentially-transparent, and are therefore more relaxed than `#[ffi_const]` functions.

However, accessing global memory through volatile or atomic reads can violate the requirement that two consecutive function calls shall return the same value.

A `pure` function that returns unit has no effect on the abstract machine's state.

A `#[ffi_pure]` function must not diverge, neither via a side effect (e.g. a call to `abort`) nor by infinite loops.

When translating C headers to Rust FFI, it is worth verifying for which targets the `pure` attribute is enabled in those headers, and using the appropriate `cfg` macros in the Rust side to match those definitions. While the semantics of `pure` are implemented identically by many C and C++ compilers, e.g., clang, GCC, ARM C/C++ compiler, IBM ILE C/C++, etc. they are not necessarily implemented in this way on all of them. It is therefore also worth verifying that the semantics of the C toolchain used to compile the binary being linked against are compatible with those of the `#[ffi_pure]`.