SAFETY IN THE WASM C API

GOALS

- Promote security.
- Simplify bindings to other languages.
- Promote interoperability between implementations.

PHILOSOPHY

- Safe defaults, with unsafe escape hatches
- Express safety in terms of existing API predicates
- Reason about invariants

EXAMPLE (CURRENT API)

```
void wasm_global_set(
  wasm_global_t*,
  const wasm_val_t*
);
```

EXAMPLE (PR #134)

```c
/// Assign a new value to a global variable.
///
/// # Errors
///
/// This function returns an error if the global is immutable
/// or if the new value has a type with a different
/// `wasm_valkind_t` than the global.
own wasm_trap_t* wasm_global_set(
  wasm_store_t*,
  wasm_global_t*,
  const wasm_val_t*
);
```

Existing API: `wasm_global_type`, `wasm_globaltype_mutability`, `wasm_globaltype_content`.

EXAMPLE (PR #134)

```c
/// Similar to `wasm_global_set`, but with undefined behavior
/// instead of reporting errors.
///
/// # Safety
///
/// This function has undefined behavior in response to any
/// errors.
void wasm_global_set_unchecked(
  wasm_global_t*,
  const wasm_val_t*
);
```

PR #134

```
wasm_instance_new
wasm_global_new
wasm_global_set
wasm_table_new
wasm_table_get
wasm_table_set
wasm_table_grow
wasm_func_call
```

UTF-8 (needs a new predicate)

```
wasm_importtype_new
wasm_exporttype_new
```

Memory lifetime (needs new APIs)

```
wasm_memory_data
```

WHAT DO WE MEAN BY "SAFE"?

- Hazards common to all C APIs:
  - bad pointers
  - arrays of wrong length
  - running out of stack space
  - etc.
- Hazards specific to wasm C API:
  - mutating immutable globals
  - wasm type errors
  - exports satisfy imports
  - etc.

DOES PASSING IN ARRAY LENGTHS IMPROVE SAFETY?

```
 own wasm_instance_t* wasm_instance_new(
   wasm_store_t*,
   const wasm_module_t*,
   const wasm_extern_t* const imports[],
+  size_t num_imports,
   own wasm_trap_t** trap
 );
```

- We've seen bugs where the module and the instantiating code get out of sync. 💥
- Bogus values are still possible, but harder to do by accident.
- Bindings generators can reliably get this right.

```
instance = wasm_instance_new(
    store,
    module,
    imports.as_ptr(),
    imports.len(),
    &trap
);
```

WHAT ABOUT PASSING A REFERENCE TO THE WRONG STORE?

- That's an interesting but separatable topic.
- The JS API assumes you only have one store.

BINDINGS TO OTHER LANGUAGES

- It's the C API's job to implement WebAssembly.
- It's the binding code's job to manage pointers, arrays, and lifetimes.

GOALS

- Promote security.
- Simplify bindings to other languages.
- Promote interoperability between implementations.