A borrow of a constant containing interior mutability was attempted.

Erroneous code example:

```
use std::sync::atomic::AtomicUsize;

const A: AtomicUsize = AtomicUsize::new(0);
const B: &'static AtomicUsize = &A;
// error: cannot borrow a constant which may contain interior mutability,
//        create a static instead
```

A `const` represents a constant value that should never change. If one takes a `&` reference to the constant, then one is taking a pointer to some memory location containing the value. Normally this is perfectly fine: most values can't be changed via a shared `&` pointer, but interior mutability would allow it. That is, a constant value could be mutated. On the other hand, a `static` is explicitly a single memory location, which can be mutated at will.

So, in order to solve this error, use statics which are `Sync`:

```
use std::sync::atomic::AtomicUsize;

static A: AtomicUsize = AtomicUsize::new(0);
static B: &'static AtomicUsize = &A; // ok!
```

You can also have this error while using a cell type:

```
use std::cell::Cell;

const A: Cell<usize> = Cell::new(1);
const B: &Cell<usize> = &A;
// error: cannot borrow a constant which may contain interior mutability,
//        create a static instead

// or:
struct C { a: Cell<usize> }

const D: C = C { a: Cell::new(1) };
const E: &Cell<usize> = &D.a; // error

// or:
const F: &C = &D; // error
```

This is because cell types do operations that are not thread-safe. Due to this, they don't implement Sync and thus can't be placed in statics.

However, if you still wish to use these types, you can achieve this by an unsafe wrapper:

```
use std::cell::Cell;
use std::marker::Sync;

struct NotThreadSafe<T> {
    value: Cell<T>,
```

```
}

unsafe impl<T> Sync for NotThreadSafe<T> {}

static A: NotThreadSafe<usize> = NotThreadSafe { value : Cell::new(1) };
static B: &'static NotThreadSafe<usize> = &A; // ok!
```

Remember this solution is unsafe! You will have to ensure that accesses to the cell are synchronized.