A type or lifetime parameter has been declared but is not actually used.

Erroneous code example:

```
enum Foo<T> {
    Bar,
}
```

If the type parameter was included by mistake, this error can be fixed by simply removing the type parameter, as shown below:

```
enum Foo {
    Bar,
}
```

Alternatively, if the type parameter was intentionally inserted, it must be used. A simple fix is shown below:

```
enum Foo<T> {
    Bar(T),
}
```

This error may also commonly be found when working with unsafe code. For example, when using raw pointers one may wish to specify the lifetime for which the pointed-at data is valid. An initial attempt (below) causes this error:

```
struct Foo<'a, T> {
    x: *const T,
}
```

We want to express the constraint that Foo should not outlive `'a`, because the data pointed to by `T` is only valid for that lifetime. The problem is that there are no actual uses of `'a`. It's possible to work around this by adding a PhantomData type to the struct, using it to tell the compiler to act as if the struct contained a borrowed reference `&'a T`:

```
use std::marker::PhantomData;

struct Foo<'a, T: 'a> {
    x: *const T,
    phantom: PhantomData<&'a T>
}
```

PhantomData can also be used to express information about unused type parameters.