The compiler could not infer a type and asked for a type annotation.

Erroneous code example:

```
let x = "hello".chars().rev().collect();
```

This error indicates that type inference did not result in one unique possible type, and extra information is required. In most cases this can be provided by adding a type annotation. Sometimes you need to specify a generic type parameter manually.

A common example is the `collect` method on `Iterator`. It has a generic type parameter with a `FromIterator` bound, which for a `char` iterator is implemented by `Vec` and `String` among others. Consider the following snippet that reverses the characters of a string:

In the first code example, the compiler cannot infer what the type of `x` should be: `Vec<char>` and `String` are both suitable candidates. To specify which type to use, you can use a type annotation on `x`:

```
let x: Vec<char> = "hello".chars().rev().collect();
```

It is not necessary to annotate the full type. Once the ambiguity is resolved, the compiler can infer the rest:

```
let x: Vec<_> = "hello".chars().rev().collect();
```

Another way to provide the compiler with enough information, is to specify the generic type parameter:

```
let x = "hello".chars().rev().collect::<Vec<char>>();
```

Again, you need not specify the full type if the compiler can infer it:

```
let x = "hello".chars().rev().collect::<Vec<_>>();
```

Apart from a method or function with a generic type parameter, this error can occur when a type parameter of a struct or trait cannot be inferred. In that case it is not always possible to use a type annotation, because all candidates have the same return type. For instance:

```
struct Foo<T> {
    num: T,
}

impl<T> Foo<T> {
    fn bar() -> i32 {
        0
    }

    fn baz() {
        let number = Foo::bar();
    }
}
```

This will fail because the compiler does not know which instance of `Foo` to call `bar` on. Change `Foo::bar()` to `Foo::<T>::bar()` to resolve the error.