Implicit Parameters

Principles of Functional Programming

Parametrization with Ordering

There is already a class in the standard library that represents orderings.

```
scala.math.Ordering[T]
```

provides ways to compare elements of type `T`. So instead of parameterizing with the `lt` operation directly, we could parameterize with `Ordering` instead:

```
def msort[T](xs: List[T])(ord: Ordering[T]) =

  def merge(xs: List[T], ys: List[T]) =
    ... if ord.lt(x, y) then ...

  ... merge(msort(fst)(ord), msort(snd)(ord)) ...
```

Ordering Instances:

Calling the new `msort` can be done like this:

```
import math.Ordering

msort(nums)(Ordering.Int)
msort(fruits)(Ordering.String)
```

This makes use of the values `Int` and `String` defined in the `scala.math.Ordering` object, which produce the right orderings on integers and strings.

Aside: Implicit Parameters

Calling `msort` can be done like this:

```
import math.Ordering

msort(nums)(Ordering.Int)
msort(fruits)(Ordering.String)
```

Problem: Passing around `lt` or `ord` values is cumbersome.

Aside: Implicit Parameters

We can avoid this by making `ord` an *implicit parameter*.

```
def msort[T](xs: List[T])(given ord: Ordering[T]) =

  def merge(xs: List[T], ys: List[T]) =
    ... if ord.lt(x, y) then ...

  ... merge(msort(fst), msort(snd)) ...
```

Then calls to `msort` can avoid the ordering parameters:

```
msort(nums)
msort(fruits)
```

The compiler will figure out the right `Ordering` instance to pass based on the demanded type.

Rules for Implicit Parameters

Say, a function takes an implicit parameter of type `T`.

The compiler will search a definition or parameter that

- is marked `given`
- has a type compatible with `T`
- is visible at the point of the function call, or is defined in a companion object associated with `T`.

If there is a single (most specific) definition, it will be taken as actual argument for the implicit parameter.

Otherwise it's an error.

Exercise: Implicit Parameters

Consider the following line of the definition of `msort`:

```
... merge(msort(fst), msort(snd)) ...
```

Which implicit argument is inserted?

```
O       Ordering.Int
O       Ordering.String
O       the "ord" parameter of "msort"
```

Exercise: Implicit Parameters

Consider the following line of the definition of `msort`:

```
    ... merge(msort(fst), msort(snd)) ...
```

Which implicit argument is inserted?

```
O        Ordering.Int
O        Ordering.String
X        the "ord" parameter of "msort"
```