# Enzo

## Grammar

### Mixfix

As enzo is an experiment, as to how small a language kernel can be, an important feature are mixfix operators (which are used to define constructs such as ‘if then else’, or even the normal operators like ‘+’). Mixfix operators allow registering new operators inside of code, that can accept an arbitrary amount of arguments, with ‘name-parts’ interspersed. Parsing them is slightly more complicated than the naive precedence climbing used, when one cares about neither associativity nor complex operators. The following is a general form grammar, that can be used to parse this structure.

```
expr → k(0)
k(p) → closed(p)
     / k(p+1) none(p) k(p+1)
     / r(p)+ k(p+1)
     / k(p+1) l(p)+
     / k(p+1)         -- when all else fails, climb

k(max+1) → ( ident / k(max+2) )+
k(max+2) → :openParen expr :closeParen

ident = :name - (closed  ∪
                 none    ∪
                 prefix  ∪
                 right   ∪
                 postfix ∪
                 left    )

r(p) → prefix(p)
     / k(p+1) right(p)

l(p) → postfix(p)
     / left(p) k(p+1)
```

Here, the terms “right”, “prefix”, “closed” and so on are all registered at runtime of the compiler, as these enumerate the operators. The maximum precedence level in use is determined as the highest level at which operators are registered. Names always resolve to the second *highest* precedence (max+1), as do function calls. Function calls and simple variables are not differentiated, as a variable is practically the same as a function `() → a`. The highest precedence level (max+2) is reserved for parentheses, as they are used for grouping arguments to functions, which can otherwise not be expressed in source code.