lollipop

A functional programming language with support for linear types.

What have we done?

We have designed and implemented a general-purpose functional programming language with support for linear types, and it's called lollipop!

Linear types?

Simply put, a linear type variable is something that has to be used exactly once in the same context it is introduced. They are denoted a bit differently in type declarations:

```
update : iArray -o Elem -> iArray
```

Linear type variables starts with an "i" and the "lollipop" sign from linear logic denotes that the left side variable has to be consumed when the function is applied. This is also what inspired the name of the language!

In practice this means a function is guaranteed neither to duplicate nor ignore its linear arguments and this comes with a couple of nice consequences. For example, the type checker makes sure that we can't lose or duplicate the array reference and this allows destructive updates since we know there is only one reference.

Why?

The purpose of the project is to research how linear types can be implemented in a functional language, and hopefully to act as a platform where developers can get comfortable with this new concept.

How does it work?

Pretty well! The language is complete and heavily inspired by Haskell. Bundled with the language is Sugar, a standard library with common functions.

Have a taste!

```
function takeWhile : (a -> Bool) -> [a] -> [a]
takeWhile _ [] := []
takeWhile f (x:xs)
    := (x:(takeWhile f xs)) when f x
    := []

function max : (a,a) -> a
max (x,y) := x when x > y
    := ty
```

```
sumList := foldr (\x y -> x+y) 0

datatype Cake a := Sweet a | Lie

function fib : Int -> Int
fib 0 := 0
fib 1 := 1
fib n := fib (n-2) +
         fib (n-1)
```