Assignment 2

Task 1: De Bruijn Indices (4 + 5 Points)

Interpreter

Implement an interpreter featuring *addition*, *subtraction*, *let* and *ref*.

Instead of symbols, de Bruijn indices are used to reference the named expressions. Please read section 3.5 from the PLAI book to get an understanding of the topic of this task.

Note: "with" in the referenced PLAI book section has the same semantics as our "let" from the lecture. In the lecture, we therefore renamed the languages in the PLAI book called *WAE to* LAE (L for "let").

Hint 1: You can access elements by index very easily:

```
scala> val l = List(1, 2, 3)
l: List[Int] = List(1, 2, 3)

scala> l(1)
res0: Int = 2
```

Hint 2: You do not need substitution to implement the interpreter

Converter

Implement a function `convert` which converts `LAE` to `LAEDB` expressions.

Hint: `List.indexOf` might come in handy. It finds the position of an element in the `List`.

Task 2: Implementing and using the If0 Expression (4 + 1 Points)

Extending the F1LAE language

Extend the language F1LAE with an `If0` expression:

```
case class If0(test: F1LAE, thenBody: F1LAE, elseBody: F1LAE)
extends F1LAE
```

The new expression type should have the following semantics: If *test* is 0, the *then* part will be evaluated. Otherwise, the *else* part will be evaluated.

Clearly mark all of your additions to the code, otherwise you risk not receiving the full grade!

Using the extended F1LAE language

Use the extended F1LAE language (with `If0` expression) you just implemented to implement a function that computes the n-th Fibonacci number by using recursion.

It is okay if your function only runs in an acceptable time-frame for the first ten Fibonacci numbers.

Task 3: Zip (4 Points)

Analyse what the `zipop` function does and use it to implement the following functions:

- `zipadd` adds two lists as vectors, i.e. produces a list, the elements of which are sums of the corresponding elements of the input lists
- `zip` combines two lists to a list of pairs
- `addMatrix` adds the corresponding elements of two matrices (lists of lists)

If one list is longer than the other, ignore the last elements of the longer list.

Task 4: Multiple Arguments (6 Points)

Here is the substitution-based interpreter for F1LAE that was presented in the class.

Modify the language F1LAE such that it allows functions with an arbitrary number of arguments. This means every function definition and every function application should hold a list of function arguments.

Clearly mark all of your additions to the code, otherwise you risk not receiving the full grade!