Interpreter in Haskell

Implement a simple language interpreter in Haskell

Ankit Kumar
Final Year, M.Tech CSE

About Haskell

High level,

Statically typed,

Strongly typed,

Inferred,

Lazy,

Declarative,

Purely Functional Programming Language

Quick Sorting in C

```c
void quicksort(int *A, int len)
{
  if (len < 2) return;

  int pivot = A[len / 2];

  int i, j;
  for (i = 0, j = len - 1; ; i++, j--)
  {
    while (A[i] < pivot) i++;
    while (A[j] > pivot) j--;

    if (i >= j) break;

    int temp = A[i];
    A[i]     = A[j];
    A[j]     = temp;
  }

  quicksort(A, i);
  quicksort(A + i, len - i);
}
```

Quick Sorting in Haskell

```haskell
qsort [] = []
qsort (x:xs) = qsort [y | y <- xs, y < x] ++ [x] ++ qsort [y | y <- xs, y >= x]
```

Projects Offered

Implement an interpreter for

- A subset of While language : a simple imperative language which only supports integer literals.

```
a := 10 ;
b := 100 ;

if ( a < b ) then
    {
        min := a ;
        max := b
    }
else {
    min := b ;
    max := a
    }
```

Projects Offered

Implement an interpreter for

- BrainF__k : a minimal language with only 8 instructions

Ex. prog in BF

```
+++++ +++++             initialize counter (cell #0) to 10
[                       use loop to set the next four cells to 70/100/30/10
    > +++++ ++              add  7 to cell #1
    > +++++ +++++           add 10 to cell #2
    > +++                   add  3 to cell #3
    > +                     add  1 to cell #4
    <<<< -                  decrement counter (cell #0)
]
> ++ .                  print 'H'
> + .                   print 'e'
+++++ ++ .              print 'l'
.                       print 'l'
+++ .                   print 'o'
> ++ .                  print ' '
<< +++++ +++++ +++++ .  print 'W'
> .                     print 'o'
+++ .                   print 'r'
----- - .               print 'l'
----- --- .             print 'd'
> + .                   print '!'
```

Extensions

Use a dependently typed programming language (coq or Agda) to prove properties about the languages we created.

Summary (What you get to learn)

Basic Functional Programming

Programming with Applicatives and Monads

Type Theory

Type Systems : Type checking, Type inference

Managing a project : text editors, command line, git

If interested, can explore Proof Assistants, and use them to prove correctness of our implementation