The Reference Manual for ƎL

Version 3.141

The ƎL Development Team

March 10, 2016

Contents

Chapter 1

Language Guide

§1.1 types

ƎL provides fundamental types, including `Int` for integers, `Double` for floating-point values, `String` for textual data, and `Bool` for Boolean value. ƎL also provides Tuple and List as described in §1.3, §1.4.

§1.2 constants

You cannot use variable. Alternatively, ƎL provides constant. The value of a constant cannot be changed once it is set in the future. This constraint ensures referential transparency.

§1.3 list

In ƎL, lists are a homogenous data structure. It stores several elements of the same type. List can be written in three different ways (Enumeration, Range , List comprehension). It should be noted that you don't have to declare the list with its type.

§1.4 tuple

ƎL provide tuple type. You can bound some values with tuple and handle as an unit value.

§1.5 function

In ƎL, you must define function to return just a value. Besides, you must declare types of each arguments and type of the return value. Using constants locally, You could declare them in `where` block below the function.

§1.6 control flow

ƎL provide only pattern-mach for control flow. Any conditional balancing you need is constructed with pattern-mach and several recursive call.

Chapter 2

Usage

§2.1 declare constants

2.1.1 declare constant

In this section, we describe the way to declare a constant. Here is the sample code to decleare $n = 2$.

Listing 2.1 declare a constant `n`

,

```
n :: Int
  = 2
```

In the first line, declare type of the constant named `n`. This time `n` is `Int` type. Then, describe the value of `n` in the right hand of ”`=`” in the second line.

2.1.2 declare list

In this section, we describe the way to declare a list. Here is a sample code to declare a list, which holds `Int` datas {`1,2,3,4,5`}.

Listing 2.2 declare a list `listA`

,

```
listA = {1,2,3,4,5}
```

You could declare a list without type declaration. You need only single line to declare a list like that example.

We could also describe same list with Range expression like this.

Listing 2.3 declare a list `listA` with `Range`

,

```
listA = {1...5}
```

§2.2 declare function

Listing 2.4 function $fibo(n)$

,

```
fibo(n) :: Int -> Int
 = 0 [n == 0]
 = 1 [n == 1]
 = fibo(n-1) + fibo(n-2)
```

§2.3 pattern match

§2.4 Input and Output

§2.5 sample code

Here is the sample code, which returns a fibonacci number.

Listing 2.5 sample code

,

```
fibo(n) :: Int -> Int
 = 0 [n == 0]
 = 1 [n == 1]
 = fibo(n-1) + fibo(n-2)
where
 who :: Int
 =5
 hoge :: Int
 =6

m :: Double
=2
```