Manual de Usuario: Lambdish

Diseño de Compiladores

Erick Francisco González Martínez

A01039859

Carlos Daniel Estrada Guerra

A01039919

3 de junio del 2020
Monterrey, Nuevo León México

Tabla de Contenido

Prefacio

Lambdish es un lenguaje funcional pequeño que tiene como objetivo la simplicidad, el rendimiento y la programación estricta. Este lenguaje tiende a ser un enfoque más simple para los lenguajes modernos de programación funcional como Haskell, Racket, etc. Este nuevo enfoque tiene la intención de aplanar la curva de aprendizaje de la programación funcional al proporcionar una manera fácil de usar las características que hacen que la programación funcional sea útil e interesante permitiendo que la propia comunidad muestra apoyo a los nuevos usuarios que deseen contribuir con el código.

Notas de Distribución

Mayor Release (V1.0) a 3 de junio del 2020

- Declaración de Funciones

  Ejemplo:

```
func helloWorld :: bool isHello => [char] (
     if(isHello,
     "HelloWorld",
     [char]
 )
)
```

- Declaración de Funciones Lambda

  Ejemplo:

```
(# num x => num (/(x, 2))
```

- Declaración de Variables

  Ejemplo:

```
num n, char c, bool b
```

- Llamada de Funciones

  Ejemplo:

```
helloWorld(true)
map([1,2,3,4], (# num x => num (/(x, 2)) ))
```

- Declaración de Listas

  Ejemplo:

  - `[num]`
  - `[[char]]`
  - `[[[lists]]]`

- Declaración de Constantes

  Ejemplo:

  - `1`
  - `'A'`
  - `"Hello"`
  - `true`
  - `[[1,2,3],[4,5,6],[7,8,9]]`

- Inicialización de Listas Vacías

  Ejemplo:

  - `helloWorld(true)`
  - `map([1,2,3,4], (# num x => num (/(x, 2)) ))`

- Uso de las Funciones Aritméticas

  Ejemplo:

  - `+(3,2)`
  - `-(3,2)`

    - `/(3,2)`
    - `*(3,2)`
    - `%(2,2)`
- Uso de las Funciones Lógicas

  Ejemplo:
    - `>(3,2)`
    - `<(3,2)`
    - `equal('c','c')`
- Uso de las Funciones Relacionales

  Ejemplo:
    - `!(false)`
    - `and(false,true)`
    - `or(true,false)`

- Uso de las Funciones del Sistema

  Ejemplo:
    - `insert(1,[2,3,4,5])`
    - `append([1,2],[3,4])`
    - `Tail([1,2,3])`
    - `head([1,2,3])`
- Uso de las Funciones Condicionales

  Ejemplo:
    - ```
      if(true,
            "HelloWorld",
            [char]
       )
      ```
- Uso de los Comentarios

  Ejemplo:

  ```
  //Esta es una linea de comentarios
  ```
  ```

Instalación y Licencia de Uso

1. Instalación del lenguaje de Go.
    a. Acceder a la siguiente página y seguir los pasos de instalación para el sistema operativo deseado: https://golang.org/doc/install
    b. Configurar el PATH local para poder usar el comando ***"go"*** en la terminal.
2. Clonar el repositorio o descargar el zip de la siguiente página: https://github.com/Loptt/lambdish-compiler
3. Configurar los variables de ambiente del compilador
    a. Abre una terminal en el sistema operativo deseado
    b. Localiza el repositorio que se descargó
    c. Localizar la carpeta **/cmd**
    d. Entrar a la carpeta
    e. Correr el comando para instalar el compilador: **`go install clamb/clamb.go`**
    f. Correr el comando para poder ejecutar: **`go install rlamb/rlamb.go`**
4. Crear un nuevo archivo con la terminación **".lsh"**
5. Empezar a programar.
6. Compilar bajo el comando: **`clamb <NombredeArchivo.lsh>`**
7. Ejecutar el programa bajo el comando: **`rlamb <NombredeArchivo.obj>`**

Licencia

Quick Start

A continuación se puede visualizar un ejemplo: **HelloWorld.lsh**

```
func helloWorld :: bool isHello => [char] (
      if(isHello,
      "HelloWorld",
      [char]
 )
)
helloWorld(true)
> [H, e, l, l, o, W, o, r, l, d]
```

Para hacer operaciones matemáticas se puede utilizar el siguiente programa para utilizar los operadores:

```
func sum :: num x, num y => num (
      +(x,y)
)
func subtraction :: num x, num y => num (
      -(x,y)
)
func division :: num x, num y => num (
      /(x,y)
)
func multiplication :: num x, num y => num (
      *(x,y)
)
func modulus :: num x, num y => num (
      %(x,y)
)
```

Para hacer operaciones relacionales se puede utilizar el siguiente programa para utilizar los operadores:

```
func greaterThan :: num x, num y => bool (
      <(x,y)
)
func lessThan :: num x, num y => bool (
      >(x,y)
)
func equalOperator :: char x, char y => bool (
      equal(x,y)
)
```

Para hacer operaciones lógicas se puede utilizar el siguiente programa para utilizar los operadores:

```
func andOperator :: bool x, bool y => bool (
      and(x,y)
)
func orOperator :: bool x, bool y => bool (
      or(x,y)
)
func notOperator :: bool x => bool (
      !(x)
)
```

Para hacer utilizar las funciones del sistema se les llama con los siguientes nombres:

```
func insertElement :: num x, [num] y => [num] (
      insert(x,y)
)
//insertElement(1,[2,3])
//>([1,2,3])
func appendLists :: [num] x, [num] y => [num] (
      append(x,y)
)
//appendLists([1],[2,3])
//>([1,2,3])

func getFirstElement :: [num] x => num (
      head(x)
)
//getFirstElement([1,2,3,4,5,6,7,8,9])
//>1

func getRestofTheList :: [num] x => [num] (
      tail(x)
)
//getRestofTheList([1,2,3,4,5,6,7,8,9])
//>[2,3,4,5,6,7,8,9]

func isEmpty :: [num] x => num (
      empty([num])
)
//isEmpty([num])
//>true
```

Para hacer operaciones lambda en las llamadas a las funciones se provee un código base para aplicar una función a los elementos de una lista de números:

```
//Declaración de función como parámetro
func map :: [num] l, (num => num) f => [num] (
      if (empty(l),
            [num],
            insert(
                  f(head(l)),
                  map(tail(l), f)
            )
      )
)
//Llamada a la función lambda
map([1,2,3,4], (# num x => num (/(x, 2)) ))
```

Extensiones

Syntax Highlighter para VSCode: Este lenguaje fue desarrollado en el IDE llamado VSCode por lo que los mismos desarrolladores del lenguaje hicieron una extensión que apoya la legibilidad del código.

Mayor Release (V0.0.3) a 3 de junio del 2020 (NO DISPONIBLE EN VSCode)
https://drive.google.com/file/d/1miwIDM3gYsj_KN_kktl9eOexbBHEyzBY/view?usp=sharing

Referencias

Para mayor referencia acerca de la generación de léxico y sintáctico consultar la siguiente liga: https://github.com/goccmack/gocc

Para mayor referencia acerca de las utilerías extras para proyectos de Go consultar la siguiente liga: https://golang.org/pkg/

Mensajes de los Autores

"El desarrollo de este lenguaje me ha facilitado la comprensión de el paradigma funcional y espero que le sirva a la comunidad desarrolladora como a mí me ha servido. Ha sido un trabajo complejo y arduo para elaborar un proyecto de esta magnitud, espero lo disfruten y podamos cumplir juntos la visión del proyecto."

- Erick González, 2020

`

"Dicen que crear un compilador representa el pináculo de la esencia del estudio de las ciencias computacionales. Después de desarrollar este proyecto, creo que esto es cierto. Este proyecto me ayudó a destapar la caja negra que siempre estaba presente mientras desarrollaba en cualquier lenguaje. Siempre había tenido el sueño de comprender qué sucede tras bambalinas. Lambdish es el producto de este sueño hecho realidad."

- Carlos Estrada, 2020

Para cualquier duda o comentario se adjuntan los correos oficiales de los desarrolladores del proyecto:

Carlos Estrada: estrada.carlosd@gmail.com
Erick González: erick.frank@icloud.com
Link al Proyecto: https://github.com/Loptt/lambdish-compiler