

INSTITUTO TECNOLÓGICO SUPERIOR DE CHICONTEPEC.

Ingeniería en sistemas computacionales.

ASIGNATURA:

Programación lógica y funcional.

ALUMNO:

Flor Hernández Cruz.

DOCENTE:

Ing. Efrén Flores Cruz.

UNIDAD 2:

Modelo de programación funcional.

TRABAJO:

Reporte de unidad 2.

SEMESTRE:

8º

Contenido

INTRODUCCION	3
DESARROLLO	4
CONCLUSION.....	16
ANEXOS	17

INTRODUCCION

El presente documento representa la unidad 2, de la asignatura de programación lógica y funcional, el tema central de esta unidad es: modelo de programación funcional, en este documento se presentan algunos ejercicios realizados durante la unidad, con la ayuda del software de aplicación WinGHCi este software es un apoyo del lenguaje de programación haskell.

Para esta unidad se desarrollaron los siguientes temas:

2.1 Introduccion al modelo de programación funcional

2.1 El tipo de datos

2.2 Funciones

2.3 Intervalos

2.4 Operadores

2.5 Aplicaciones de las listas

2.6 Arboles

2.7 Evaluación perezosa.

Para la realización de las siguientes operaciones se van a usar los siguientes operadores:

- Aritméticos
- Lógicos
- Comparación.

DESARROLLO

HASKELL

Es un lenguaje de programación estandarizado multi-propósito puramente funcional con semánticas no estrictas y fuerte tipificación estática. Su nombre se debe al lógico estadounidense Haskell Curry. En Haskell, "una función es un ciudadano de primera clase" del lenguaje de programación. Como lenguaje de programación funcional, el constructor de controles primario es la función. El lenguaje tiene sus orígenes en las observaciones de Haskell Curry y sus descendientes intelectuales.

En los años 1980 se constituyó un comité cuyo objetivo era crear un lenguaje funcional que reuniera las características de los múltiples lenguajes funcionales de la época, el más notable Miranda, y resolviera la confusión creada por la proliferación de los mismos.

Características

- ❖ Clasifica los entes de un programa en: objetos (constantes y funciones)
- ❖ tipos: cada objeto debe tener un tipo.
- ❖ Dispone de objetos y tipos predefinidos.
- ❖ Permite diversas declaraciones de objetos (monomorfos y polimorfos): funciones constructoras y funciones definidas.
- ❖ Permite diversas declaraciones de tipos (monomorfos y polimorfos): tipos sinónimos y tipos algebraicos
- ❖ Aplica: un sistema de inferencia de tipos basado en el sistema de Hindley-Milner; una estrategia de reducción perezosa.

TIPOS SIMPLES PREDEFINIDOS

En Haskell, y en lo siguiente, "o: t" quiere decir que el objeto "o" es miembro del tipo "t", y "t -> s" es un tipo, específicamente una función, que consume algo de tipo "t" y produce algo de tipo "s". El operador (->) se nida por el derecho, ya que "t -> s -> r" quiere decir "t -> (s -> r)".

El tipo Bool

Los valores con este tipo representan expresiones lógicas cuyo resultado puede ser True o False.

Funciones y operadores

`(&&): Bool -> Bool -> Bool`. Conjunción lógica.

`(||): Bool -> Bool -> Bool`. Disyunción lógica.

`not: Bool -> Bool`. Negación lógica.

`otherwise :: Bool`. Función constante que devuelve el valor True.

El tipo Int

Los valores de este tipo son números enteros de precisión limitada que cubren al menos el intervalo $[-2^{29}, 2^{29} - 1]$ (`[minBound, maxBound]`).

El tipo Integer

Los valores de este tipo son números enteros de precisión ilimitada que tienen las mismas funciones y operadores del tipo Int.

El tipo Float [

Los valores de este tipo son números reales. (2010, 23.4, 5.7)

Funciones y operadores

`(+), (-), (*), (/), (^): Float -> Float -> Float`. Suma, resta, producto, división real y potencia de exponente entero.

`abs, signum, negate: Int -> Int`. Valor absoluto, signo y negación.

`(**): Float -> Float`. Potencia de exponente real

El tipo Double

Los valores de este tipo son números reales, de mayor rango y con aproximaciones más precisas que los de tipo Float.

El tipo Char

Los valores de este tipo son caracteres que se encuentran en una masa de alta complejidad de en una suma de caracteres dados con su alta definición.

Antes de utilizar esta función en hugs debemos utilizar `IMPORT CHAR` antes de nuestro algoritmo.

Tuplas

Los elementos que forman una **tupla** pueden ser del mismo o de distintos tipos. Es un conjunto de componentes relacionados. Por ejemplo: ('a', True, 3)

Listas

Los valores de este tipo son una colección de elementos del mismo tipo. Existen dos constructores para listas:

[Elementos_separados_por_comas], por ejemplo: [1,2,3,4]

(primer_elemento: resto_de_la_lista), por ejemplo: (1:(2:(3:(4: []))))

SUCC

•succ: devuelve el valor siguiente al parámetro introducido

```
Prelude> succ 9
10
Prelude> succ 'a'
'b'
Prelude> succ 'h'
'i'
Prelude> succ 'f'
'g'
Prelude> succ 'r'
's'
Prelude>
```

MIN

min: devuelve el valor mínimo de dos argumentos. devuelve el número más pequeño de dos números introducidos, solo soporta dos parámetros. Se agrega el comando de min seguido de 2 números, al dar enter se va a registrar el número más pequeño.

```
Prelude> min 3 89
3
Prelude> min 90 78
78
Prelude> min 20 600
20
```

MAX

•max: devuelve el valor máximo de dos argumentos. Se agrega el comando de max seguido de 2 números, al dar enter se va a registrar el número más grande.

```
Prelude> max 56 78.5  
78.5
```

```
Prelude> max 10 12  
12
```

```
Prelude>
```

```
Prelude> max 89 677  
677
```

```
Prelude> max 56 83  
83
```

```
Prelude> max 4 (succ 10)  
11
```

```
Prelude> succ (max 8 3)  
9
```

```
Prelude> succ (max 8 (min 24 9.5))  
10.5
```

```
Prelude> max 7 (succ 76)  
77
```

```
Prelude> succ (max 90 67)  
91
```

```
Prelude> succ (max 5 (min 45 23.4))  
24.4
```


LISTAS

Una lista es una estructura de datos que representa un conjunto de datos de un mismo tipo, es muy usada e importante en el lenguaje Haskell.

`length` = longitud de la lista `length nomLista`

`head` = muestra el primer elemento de la lista

`tail` = muestra el cuerpo de la lista, excepto el primer elemento

`last` = devuelve el último elemento de la lista

`init` = muestra todos los elementos de la lista, excepto el último

`reverse` = devuelve la lista de manera inversa

`take x` = muestra los primeros `x` elementos que indiquemos

`take 5 lista` = mostrará los primeros cinco elementos de la lista

`drop x` = quita los primeros `x` elementos que indiquemos

`drop 3 lista` = eliminará los primeros 3 elementos de la lista

`minimum` = devuelve el valor mínimo de una lista

`maximum` = devuelve el valor máximo de una lista

`sum lista` = suma los elementos de la lista

`product lista` = devuelve el producto de todos los elementos de la lista
``elem`` = hace referencia a un elemento de la lista
`8 `elem` lista`

```
Prelude> lista = [1,2,3,4,5]
Prelude> lista
[1,2,3,4,5]
Prelude> lista = ['a','b','c']
Prelude> lista
"abc"
Prelude> lista = ['F','l','o','r']
Prelude> lista
"Flor"
Prelude> lista = ['H','e','r','n','a','n','d','e','z']
Prelude> lista
"Hernandez"
Prelude> lista = ['C','r','u','z']
Prelude> lista
"Cruz"

Prelude> ['h', 'o'] ++ ['l' , 'a']
"hola"
Prelude> [6,7,8] ++ [2,5]
[6,7,8,2,5]
Prelude>

Prelude> 45: [35, 25, 15]
[45,35,25,15]
Prelude>
Prelude> 'H' : "ola mundo"
"Hola mundo"
Prelude> 'F' : "lor Hernandez"
"Flor Hernandez"
```

Listas

Se agrega el comando `let` seguido del nombre de la variable entre corchetes los números separados por comas. El resultado muestra las listas ordenadas.

```
Prelude> let lista = [23, 24, 25]
```

```
Prelude> lista
```

```
[23,24,25]
```

```
Prelude> lista !!0
```

```
23
```

```
Prelude> lista = [[1,2], [3,4]]
```

```
Prelude> lista
```

```
[[1,2],[3,4]]
```

```
Prelude> lista !!0
```

```
23
```

En la siguiente imagen muestra todos los números del rango que se está estableciendo por ejemplo del 2 hasta el 100, pero de 2 en 2.

```
Prelude> let lista= [2, 4 .. 100]
```

```
Prelude> lista
```

```
[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80,82,84,86,88,90,92,94,96,98,100]
```

```
Prelude> let lista=[ x | x <- [1.. 20], x `mod` 2 ==1 ]
```

```
Prelude> lista
```

```
[1,3,5,7,9,11,13,15,17,19]
```

```
Prelude> let lista =[x*10 | x <- [1..20] , x `mod` 2 == 0]
```

```
Prelude> lista
```

```
[20,40,60,80,100,120,140,160,180,200]
```



[illegible]

REPLICATE

Se duplica o replica el mensaje que se pone entre comillas

```
Prelude> replicate 2 "ho"  
["ho", "ho"]
```

LET DUPLA

```
Prelude> let dupla = (1,"dos")  
Prelude> dupla  
(1,"dos")
```

Se muestran los datos de la información seguidas.

```
Prelude> let tripla = (1, "Pedro", 7461134094)  
Prelude> tripla  
(1,"Pedro",7461134094)  
Prelude> let lista = [(1, "dos"), (2, "uno")]  
Prelude> lista  
[(1,"dos"),(2,"uno")]
```

: T

El comando: t se utiliza para ver qué tipo de dato tiene un valor o una función: t
head: t fst

```
Prelude> :t head  
head :: [a] -> a  
Prelude> :t fst  
fst :: (a, b) -> a
```

```
Prelude> lista = [[1,2], [3,4]]
Prelude> lista
[[1,2],[3,4]]
Prelude> lista !!0
[1,2]
Prelude> lista !!0 !!1
2
```

LIS

```
Prelude> lis
[2,4,8]
Prelude> 8 `elem` lis
True
```

```
Prelude> let eje =[(a,b,c)|a<-[1..10], b<-[1..10], c<-[1..10]]
Prelude> eje
[(1,1,1),(1,1,2),(1,1,3),(1,1,4),(1,1,5),(1,1,6),(1,1,7),(1,1,8),(1,1,9),(1,1,10),(1,2,1),(1,2,2),(1,2,3),(1,2,4),(1,2,5),(1,2,6),(1,2,7),(1,2,8),(1,2,9),(1,2,10),(1,3,1),(1,3,2),(1,3,3),(1,3,4),(1,3,5),(1,3,6),(1,3,7),(1,3,8),(1,3,9),(1,3,10),(1,4,1),(1,4,2),(1,4,3),(1,4,4),(1,4,5),(1,4,6),(1,4,7),(1,4,8),(1,4,9),(1,4,10),(1,5,1),(1,5,2),(1,5,3),(1,5,4),(1,5,5),(1,5,6),(1,5,7),(1,5,8),(1,5,9),(1,5,10),(1,6,1),(1,6,2),(1,6,3),(1,6,4),(1,6,5),(1,6,6),(1,6,7),(1,6,8),(1,6,9),(1,6,10),(1,7,1),(1,7,2),(1,7,3),(1,7,4),(1,7,5),(1,7,6),(1,7,7),(1,7,8),(1,7,9),(1,7,10),(1,8,1),(1,8,2),(1,8,3),(1,8,4),(1,8,5),(1,8,6),(1,8,7),(1,8,8),(1,8,9),(1,8,10),(1,9,1),(1,9,2),(1,9,3),(1,9,4),(1,9,5),(1,9,6),(1,9,7),(1,9,8),(1,9,9),(1,9,10),(1,10,1),(1,10,2),(1,10,3),(1,10,4),(1,10,5),(1,10,6),(1,10,7),(1,10,8),(1,10,9),(1,10,10),(2,1,1),(2,1,2),(2,1,3),(2,1,4),(2,1,5),(2,1,6),(2,1,7),(2,1,8),(2,1,9),(2,1,10),(2,2,1),(2,2,2),(2,2,3),(2,2,4),(2,2,5),(2,2,6),(2,2,7),(2,2,8),(2,2,9),(2,2,10),(2,3,1),(2,3,2),(2,3,3),(2,3,4),(2,3,5),(2,3,6),(2,3,7),(2,3,8),(2,3,9),(2,3,10),(2,4,1),(2,4,2),(2,4,3),(2,4,4),(2,4,5),(2,4,6),(2,4,7),(2,4,8),(2,4,9),(2,4,10),(2,5,1),(2,5,2),(2,5,3),(2,5,4),(2,5,5),(2,5,6),(2,5,7),(2,5,8),(2,5,9),(2,5,10),(2,6,1),(2,6,2),(2,6,3),(2,6,4),(2,6,5),(2,6,6),(2,6,7),(2,6,8),(2,6,9),(2,6,10),(2,7,1),(2,7,2),(2,7,3),(2,7,4),(2,7,5),(2,7,6),(2,7,7),(2,7,8),(2,7,9),(2,7,10),(2,8,1),(2,8,2),(2,8,3),(2,8,4),(2,8,5),(2,8,6),(2,8,7),(2,8,8),(2,8,9),(2,8,10),(2,9,1),(2,9,2),(2,9,3),(2,9,4),(2,9,5),(2,9,6),(2,9,7),(2,9,8),(2,9,9),(2,9,10),(2,10,1),(2,10,2),(2,10,3),(2,10,4),(2,10,5),(2,10,6),(2,10,7),(2,10,8),(2,10,9),(2,10,10),(3,1,1),(3,1,2),(3,1,3),(3,1,4),(3,1,5),(3,1,6),(3,1,7),(3,1,8),(3,1,9),(3,1,10),(3,2,1),(3,2,2),(3,2,3),(3,2,4),(3,2,5),(3,2,6),(3,2,7),(3,2,8),(3,2,9),(3,2,10),(3,3,1),(3,3,2),(3,3,3),(3,3,4),(3,3,5),(3,3,6),(3,3,7),(3,3,8),(3,3,9),(3,3,10),(3,4,1),(3,4,2),(3,4,3),(3,4,4),(3,4,5),(3,4,6),(3,4,7),(3,4,8),(3,4,9),(3,4,10),(3,5,1),(3,5,2),(3,5,3),(3,5,4),(3,5,5),(3,5,6),(3,5,7),(3,5,8),(3,5,9),(3,5,10),(3,6,1),(3,6,2),(3,6,3),(3,6,4),(3,6,5),(3,6,6),(3,6,7),(3,6,8),(3,6,9),(3,6,10),(3,7,1),(3,7,2),(3,7,3),(3,7,4),(3,7,5),(3,7,6),(3,7,7),(3,7,8),(3,7,9),(3,7,10),(3,8,1),(3,8,2),(3,8,3),(3,8,4),(3,8,5),(3,8,6),(3,8,7),(3,8,8),(3,8,9),(3,8,10),(3,9,1),(3,9,2),(3,9,3),(3,9,4),(3,9,5),(3,9,6),(3,9,7),(3,9,8),(3,9,9),(3,9,10),(3,10,1),(3,10,2),(3,10,3),(3,10,4),(3,10,5),(3,10,6),(3,10,7),(3,10,8),(3,10,9),(3,10,10),(4,1,1),(4,1,2),(4,1,3),(4,1,4),(4,1,5),(4,1,6),(4,1,7),(4,1,8),(4,1,9),(4,1,10),(4,2,1),(4,2,2),(4,2,3),(4,2,4),(4,2,5),(4,2,6),(4,2,7),(4,2,8),(4,2,9),(4,2,10),(4,3,1),(4,3,2),(4,3,3),(4,3,4),(4,3,5),(4,3,6),(4,3,7),(4,3,8),(4,3,9),(4,3,10),(4,4,1),(4,4,2),(4,4,3),(4,4,4),(4,4,5),(4,4,6),(4,4,7),(4,4,8),(4,4,9),(4,4,10),(4,5,1),(4,5,2),(4,5,3),(4,5,4),(4,5,5),(4,5,6),(4,5,7),(4,5,8),(4,5,9),(4,5,10),(4,6,1),(4,6,2),(4,6,3),(4,6,4),(4,6,5),(4,6,6),(4,6,7),(4,6,8),(4,6,9),(4,6,10),(4,7,1),(4,7,2),(4,7,3),(4,7,4),(4,7,5),(4,7,6),(4,7,7),(4,7,8),(4,7,9),(4,7,10),(4,8,1),(4,8,2),(4,8,3),(4,8,4),(4,8,5),(4,8,6),(4,8,7),(4,8,8),(4,8,9),(4,8,10),(4,9,1),(4,9,2),(4,9,3),(4,9,4),(4,9,5),(4,9,6),(4,9,7),(4,9,8),(4,9,9),(4,9,10),(4,10,1),(4,10,2),(4,10,3),(4,10,4),(4,10,5),(4,10,6),(4,10,7),(4,10,8),(4,10,9),(4,10,10),(5,1,1),(5,1,2),(5,1,3),(5,1,4),(5,1,5),(5,1,6),(5,1,7),(5,1,8),(5,1,9),(5,1,10),(5,2,1),(5,2,2),(5,2,3),(5,2,4),(5,2,5),(5,2,6),(5,2,7),(5,2,8),(5,2,9),(5,2,10),(5,3,1),(5,3,2),(5,3,3),(5,3,4),(5,3,5),(5,3,6),(5,3,7),(5,3,8),(5,3,9),(5,3,10),(5,4,1),(5,4,2),(5,4,3),(5,4,4),(5,4,5),(5,4,6),(5,4,7),(5,4,8),(5,4,9),(5,4,10),(5,5,1),(5,5,2),(5,5,3),(5,5,4),(5,5,5),(5,5,6),(5,5,7),(5,5,8),(5,5,9),(5,5,10),(5,6,1),(5,6,2),(5,6,3),(5,6,4),(5,6,5),(5,6,6),(5,6,7),(5,6,8),(5,6,9),(5,6,10),(5,7,1),(5,7,2),(5,7,3),(5,7,4),(5,7,5),(5,7,6),(5,7,7),(5,7,8),(5,7,9),(5,7,10),(5,8,1),(5,8,2),(5,8,3),(5,8,4),(5,8,5),(5,8,6),(5,8,7),(5,8,8),(5,8,9),(5,8,10),(5,9,1),(5,9,2),(5,9,3),(5,9,4),(5,9,5),(5,9,6),(5,9,7),(5,9,8),(5,9,9),(5,9,10),(5,10,1),(5,10,2),(5,10,3),(5,10,4),(5,10,5),(5,10,6),(5,10,7),(5,10,8),(5,10,9),(5,10,10),(6,1,1),(6,1,2),(6,1,3),(6,1,4),(6,1,5),(6,1,6),(6,1,7),(6,1,8),(6,1,9),(6,1,10),(6,2,1),(6,2,2),(6,2,3),(6,2,4),(6,2,5),(6,2,6),(6,2,7),(6,2,8),(6,2,9),(6,2,10),(6,3,1),(6,3,2),(6,3,3),(6,3,4),(6,3,5),(6,3,6),(6,3,7),(6,3,8),(6,3,9),(6,3,10),(6,4,1),(6,4,2),(6,4,3),(6,4,4),(6,4,5),(6,4,6),(6,4,7),(6,4,8),(6,4,9),(6,4,10),(6,5,1),(6,5,2),(6,5,3),(6,5,4),(6,5,5),(6,5,6),(6,5,7),(6,5,8),(6,5,9),(6,5,10),(6,6,1),(6,6,2),(6,6,3),(6,6,4),(6,6,5),(6,6,6),(6,6,7),(6,6,8),(6,6,9),(6,6,10),(6,7,1),(6,7,2),(6,7,3),(6,7,4),(6,7,5),(6,7,6),(6,7,7),(6,7,8),(6,7,9),(6,7,10),(6,8,1),(6,8,2),(6,8,3),(6,8,4),(6,8,5),(6,8,6),(6,8,7),(6,8,8),(6,8,9),(6,8,10),(6,9,1),(6,9,2),(6,9,3),(6,9,4),(6,9,5),(6,9,6),(6,9,7),(6,9,8),(6,9,9),(6,9,10),(6,10,1),(6,10,2),(6,10,3),(6,10,4),(6,10,5),(6,10,6),(6,10,7),(6,10,8),(6,10,9),(6,10,10),(7,1,1),(7,1,2),(7,1,3),(7,1,4),(7,1,5),(7,1,6),(7,1,7),(7,1,8),(7,1,9),(7,1,10),(7,2,1),(7,2,2),(7,2,3),(7,2,4),(7,2,5),(7,2,6),(7,2,7),(7,2,8),(7,2,9),(7,2,10),(7,3,1),(7,3,2),(7,3,3),(7,3,4),(7,3,5),(7,3,6),(7,3,7),(7,3,8),(7,3,9),(7,3,10),(7,4,1),(7,4,2),(7,4,3),(7,4,4),(7,4,5),(7,4,6),(7,4,7),(7,4,8),(7,4,9),(7,4,10),(7,5,1),(7,5,2),(7,5,3),(7,5,4),(7,5,5),(7,5,6),(7,5,7),(7,5,8),(7,5,9),(7,5,10),(7,6,1),(7,6,2),(7,6,3),(7,6,4),(7,6,5),(7,6,6),(7,6,7),(7,6,8),(7,6,9),(7,6,10),(7,7,1),(7,7,2),(7,7,3),(7,7,4),(7,7,5),(7,7,6),(7,7,7),(7,7,8),(7,7,9),(7,7,10),(7,8,1),(7,8,2),(7,8,3),(7,8,4),(7,8,5),(7,8,6),(7,8,7),(7,8,8),(7,8,9),(7,8,10),(7,9,1),(7,9,2),(7,9,3),(7,9,4),(7,9,5),(7,9,6),(7,9,7),(7,9,8),(7,9,9),(7,9,10),(7,10,1),(7,10,2),(7,10,3),(7,10,4),(7,10,5),(7,10,6),(7,10,7),(7,10,8),(7,10,9),(7,10,10)]
```

```
Prelude> let eje =[(a,b,c)|c<-[1..10], b<-[c], a<-[b]]
Prelude> eje
[(1,1,1),(2,2,2),(3,3,3),(4,4,4),(5,5,5),(6,6,6),(7,7,7),(8,8,8),(9,9,9),(10,10,10)]
Prelude> let eje =[(a,b,c)|c<-[1..10], b<-[c], a<-[b],a^2+b^2==c^2]
Prelude> eje
[]
Prelude> let eje =[(a,b,c)|c<-[1..10], b<-[1..c], a<-[1..b],a^2+b^2==c^2]
Prelude> eje
[(3,4,5),(6,8,10)]
Prelude> let eje =[(a,b,c)|c<-[1..10], b<-[1..c], a<-[1..b],a^2+b^2==c^2, a+b+c==24]
Prelude> eje
[(6,8,10)]
```


CONCLUSION

Durante esta unidad 2 de la asignatura de programación lógica y funcional me pareció muy interesante, debido a que desarrollamos las funcionalidades del lenguaje de programación Haskell en el software de aplicación WinGHCi, este software permite la ejecución de algunas operaciones aritméticas, lógicas, etc.

Esta operación nos permite tener mejor conocimiento y razonamiento de cómo se puede realizar soluciones de algunos problemas lógicos y funcionales.

Haskell es un buen lenguaje de programación muy complejo que nos permite conocer el funcionamiento de diversas operaciones.

ANEXOS

HASKELL WinGHC:

→ **Conceptos**

El entorno HUGS funciona siguiendo el modelo de una calculadora en el que se establece una sesión interactiva entre el ordenador y el usuario.

Una vez arrancado, el sistema muestra un prompt "?" y espera a que el usuario introduzca una expresión (denominada expresión inicial y presione la tecla).

A. Un identificador comienza con una letra del alfabeto seguida, opcionalmente, por una secuencia de caracteres, cada uno de los cuales es, una letra, un dígito, un apóstrofe (') o un subrayado (_).

Los identificadores que representan funciones o variables deben comenzar por letra minúscula (los identificadores que comienzan con letra mayúscula se emplearán como funciones constructoras). Los siguientes son ejemplos de posibles identificadores:

sum f f" intSum elemento_dos

Los siguientes identificadores son palabras reservadas y no pueden utilizarse como nombres de funciones o variables:

case	of	where	let	in	infixl
then	else	Data	type	infix	
infixr	primitive	class	instance	if	

Cuando se trabaja con símbolos de operador es necesario tener en cuenta.

La precedencia: La expresión "2 * 3 + 4" podría interpretarse como "(2 * 3) + 4" o como "2 * (3 + 4)".

infixl 9!!

infixr 9.

infixr 8^

infixl 7*

infix 7 /, 'div', 'rem', 'mod'

infixl 6 +, -

infix 5 \

infixr 5 ++, %

infix 4 ==, /=, <, <=, >=, >

infix 4 'elem', 'notElem'

infixr 3 &&

infixr 2 ||

Tabla de procedencia/asociatividad de operadores.

La asociatividad: La regla anterior resolvía ambigüedades cuando los símbolos de operador tienen distintos valores de procedencia. Sin embargo, la expresión "1-2-3" puede ser tratada como "(1-2)-3" resultando -4 o como "1-(2-3)" resultando 2.

infixl digito ops Para declarar operadores asociativos a la izquierda
infixr digito ops Para declarar operadores asociativos a la derecha
infix digito ops Para declarar

Asociativo a la izquierda: Si la expresión "x-y-z" se toma como "(x-y)-z"

Asociativo a la derecha: Si la expresión "x-y-z" se toma como "x-(y-z)"

No asociativa: Si la expresión "x-y-z" se rechaza como un error sintáctico a operadores no asociativos.

En el estándar prelude el (-) se toma como asociativo a la izquierda por lo que la expresión "1-2-3" se tratará como "(1-2)-3".

La expresión "fx + gy" equivale a "(fx) + (gy)".

La expresión "fx+1", que es tratada como "(fx)+1" en lugar de "f(x+1)".

= Tipos booleanos =

$x \& \& y$ es True si y sólo si x e y son True

$x || y$ es True si y sólo si x ó y ó ambas son True

$\text{not } x$ es el valor opuesto de x (not True = False, not False = True)

= Operadores =

$+$ Suma

$*$ Multiplicación

$-$ resta

\wedge Potencia

negate menos unario (la expresión " $-x$ " se toma como "negate x ")

div división entera " $/"$

rem resto de la división entera siguiendo la ley = $(x \text{ 'div' } y) * y + (x \text{ 'rem' } y) = x$

mod módulo, como rem sólo que el resultado tiene el mismo signo ^{del divisor}

odd devuelve True si el argumento es impar

even devuelve True si el argumento es par

gcd máximo común divisor.

lcm mínimo común múltiplo

abs valor absoluto

Signum devuelve -1, 0 o 1 si el argumento es negativo, cero o positivo

Ejemplos: $3 \wedge 4 = 81$ $7 \text{ 'div' } 3 = 2$ $\text{Even } 23 = \text{False}$

$7 \text{ 'rem' } 3 = 1$ $-7 \text{ 'rem' } 3 = -1$ $7 \text{ 'rem' } -3 = 1$

$7 \text{ 'mod' } 3 = 1$ $-7 \text{ 'mod' } 3 = 2$ $7 \text{ 'mod' } -3 = -2$

$\text{gcd } 3212 = 4$ $\text{abs}(-2) = 2$ $\text{Signum } 12 = 1$

= Flotantes =

Representados por el tipo "Float", los elementos de este tipo pueden ser utilizados para representar fraccionarios así como cantidades muy largas o muy pequeñas.

Ejemplo: $1.0e3$ equivale a 1000.0, mientras que $5.0e-2$ equivale a 0.05

El estándar prelude incluye también múltiples funciones de manipulación de flotantes: π , \exp , \log , $\sqrt{}$, \sin , \cos , \tan , \arcsin , \arccos , \arctan ,

= Caracteres =

Representados por el tipo "Char". Los elementos de este tipo representan caracteres individuales como los que se pueden introducir por teclado. Los valores de tipo carácter se escriben encerrando el valor entre comillas simples, por ejemplo 'a', '0', '.' y 'Z'.

'\|' Barra invertida
'\'' Comilla simple
'\" Comilla doble
'\n' Salto de línea
'\b' or '\BS' (backspace) (espacio atrás)
'\DEL' borrado
'\t' or '\HT' Tabulador
'\a' or '\BEL' Alarma (campana)
'\f' or '\FF' alimentación de papel

= Listas =

El estándar prelude incluye un amplio conjunto de funciones de manejo de listas, por ejemplo:

$\text{length } xs$ devuelve el número de elementos de xs .

$xs ++ ys$ devuelve la lista resultante de concatenar xs e ys .

$\text{concat } xss$ devuelve la lista resultante de concatenar las listas de xss .

$\text{map } f \ xs$ devuelve la lista de valores obtenidos al aplicar la función f a cada uno de los elementos de la lista xs .

Ejemplos $\text{length } [1,3,10] == 3$

$[1,3,10] ++ [2,6,5,7] == [1,3,10,2,6,5,7]$

$\text{concat } [[1],[2,3],[],[4,5,6]] == [1,2,3,4,5,6]$

$\text{map fromEnum } ['H','0','1','a'] == [72,111,108,97]$

$\text{sum } [1..10]$ La notación $[1..10]$ representa la lista de enteros que van de 1 hasta 10, y sum es una función estándar que devuelve la suma de una lista de enteros. El resultado obtenido por el sistema es: $1+2+3+4+5+6+7+8+9+10 = 55$

= Cadenas =
Una cadena es tratada como una lista de caracteres y el tipo "String" se toma como una abreviación de "[char]". Las cadenas pueden ser escritas como consecuencias de caracteres encerradas entre comillas dobles. Todos los códigos de escape utilizados para los caracteres, pueden utilizarse para las cadenas.

Puesto que las cadenas son representadas como listas de caracteres, todas las funciones del standard para listas pueden ser utilizadas también con cadenas.

length "Hola" → 4

"Hola," ++ "amigo" → Hola, amigo

concat ["Super", "cali", "fragi", "listico"] →

Supercali fragi listico

map from Enum "Hola" → [72, 111, 108, 97]

R28/02/2020

let
~~length~~ listas = [(1,2), (3,4), (5,6)]

```
let listas [(1,2),(3,4),(5,6)]
= [(1,2),(3,4),(5,6)]
```

4-Marzo-2020

```
let lis2 = [(1,"A"),(2,"B")]
= [(1,"A"),(2,"B")]
```

listas son en
corchetes

```
let tupla = ("uno",2)
```

```
fst tupla = "uno"
```

```
snd tupla = 2
```

→ muestra el
1º número

→ muestra el último número

Tuplas en
parentesis

```
let nombre ["Juan","Pedro","Olivia"]
```

```
let edad [15,18,25]
```

```
zip nombre edad → Une las 2 listas
```

```
= [( "Juan",15), ( "Pedro",18), ( "Olivia",25)]
```

```
zip [1..] nombre
```

```
[(1,"Juan"),(2,"Pedro"),(3,"Olivia")]
```

```
:+ fst
```

```
fst :: (a,b) → a
```

} muestra el funcionamiento
de fst

```
:+ snd
```

```
snd :: (a,b) → b
```

} funcionamiento de snd

```
Show S  
"S"
```

```
Show True  
"True"
```

```
read "S" + S  
10
```

```
read "[1,2,3]" ++ [2]  
[1,2,3,2]
```

[Lo que queremos
muestra] $x \leftarrow [lista\ filtrar], condición$

$x = [x \mid x \in [1..100], x \bmod 2 == 0]$ muestra los
números pares

let list = $[x \mid x \in [1..100], x \bmod 10 == 0]$ muestra nume
ros de 10 en 10
let list = $[x \mid x \in [1..100], x \bmod 2 == 1]$ Números pares
"Oavo"

$\left\{ \begin{array}{l} \text{Vocal frase} = [vocal \mid vocal \leftarrow frase, vocal \in ['á', 'e', 'i', 'ó', 'u']] \\ \text{Vocal "Mexico"} \\ R = "eio" \end{array} \right.$

$\left\{ \begin{array}{l} \text{vocal frase} = [vocal \mid vocal \leftarrow frase, vocal == 'á'] \\ \text{vocal "árbol"} \\ R = "a" \end{array} \right.$

$\left\{ \begin{array}{l} \text{suma vocal} = \text{sum} [1 \mid x \leftarrow (\text{vocales vocal})] \\ \text{suma "Mexico"} \end{array} \right.$

$\left. \begin{array}{l} \text{Vocal frase} = [vocal \mid vocal \leftarrow frase, vocal == 'á'] \\ \text{suma vocal} = \text{sum} [1 \mid x \leftarrow (\text{vocales vocal})] \\ \text{suma "amaranta"} \\ R \rightarrow 4 \\ R \rightarrow "aaaa" \end{array} \right\}$

let eje = (a,b,c)