



Instituto Tecnológico de Mexicali

Reporte de Practica 2 **“Aprende Haskell en 10 minutos”**

Carrera:

Ingeniería en Sistemas Computacionales

Nombre del Alumno(a):

Martínez Yebra Beatriz Andrea
#13490929

Nombre del profesor(a):

Batista Gaxiola Oscar Ruben

Materia:

Programación Lógica y Funcional

Horario:

9:00 p.m. – 10:00 p.m.

Mexicali Baja California, Jueves 08 de Marzo 2018.

Introducción

El siguiente documento muestra un reporte de practica el cual sirve como evidencia de que se han realizado los ejercicios de la sesión de clase y que se ha comprendido el desarrollo de estos mismos, se mostrara y explicara cada ejercicio de la siguiente página [https://wiki.haskell.org/Learn Haskell in 10 minutes](https://wiki.haskell.org/Learn_Haskell_in_10_minutes) , la cual proporciono el profesor de la clase Batista Gaxiola Oscar Ruben.

En esta práctica se explica brevemente la descripción general del lenguaje Haskell, expresiones simples, consola, datos, definiciones, sintaxis y mucho más con ejemplos para realizar y para aprender Haskell en 10 minutos, los cuales se vieron todos en una clase de una 1 hora.

Contenido

El siguiente índice de contenido muestra los temas desarrollados a lo largo del reporte, si se desea dirigir a uno puede dar click en dicho título para dirigirse a la página donde se encuentra la información deseada.

	N° Página
Introducción	1
Tabla de contenido	2
Tabla de figuras	3
Introducción al reporte	4
a) Objetivos	
b) Fundamento Teórico	
c) Justificación, Aplicación e importancia	
Enunciado de las actividades	5 - 14
1- Simple Expressions	
2- Console	
3- Simple Types	
4- Datos Estructurados	
5- Definiciones de Funciones	
Resultados	15
Conclusiones	16
Bibliografía	17

Tabla de Figuras

La siguiente tabla muestra un listado de la ubicación de capturas de pantallas o imágenes de los ejercicios realizados a lo largo del siguiente reporte.

NOTA:

- Selecciona el título de alguna de captura de pantalla o Imagen para ir a la página donde se encuentra.

TABLA DE DIRECCIONAMIENTO DE CAPTURAS DE PANTALLA/IMÁGENES	
Captura de Pantalla 1	<u>“Operaciones Sencillas”</u>
Captura de Pantalla 2	<u>“Sintaxis y Concatenar”</u>
Captura de Pantalla 3	<u>“Uso de Funciones”</u>
Captura de Pantalla 4	<u>“Saltos de línea”</u>
Captura de Pantalla 5	<u>“Uso de “do”!”</u>
Captura de Pantalla 6	<u>“Lectura”</u>
Captura de Pantalla 7	<u>“Indiferencia en Haskell”</u>
Captura de Pantalla 8	<u>“Declaración en Haskell”</u>
Captura de Pantalla 9	<u>“Clase de tipos”</u>
Captura de Pantalla 10	<u>“Tipo de clase Num”</u>
Captura de Pantalla 11	<u>“Tipo de clase Unit”</u>
Captura de Pantalla 12	<u>“Tipos de datos básicos Listas”</u>
Captura de Pantalla 13	<u>“Cadena de caracteres”</u>
Captura de Pantalla 14	<u>“Numero de valores, uso de Zip”</u>
Captura de Pantalla 15	<u>Listas, Tuplas</u>
Captura de Pantalla 16	<u>“Llamada a main”</u>
Captura de Pantalla 17	<u>“Ejemplo Factorial”</u>

Introducción al reporte

En el siguiente reporte mostraremos practicas realizadas donde utilizando el intérprete “WinGHCi” de Haskell, solucionamos operaciones matemáticas. Como, por ejemplo: $3 * 5$ o $4^2 - 1$. Además, echaremos un vistazo a comandos básicos para cierta sintaxis que se debe llevar a cabo al imprimir algún dato, ya que Haskell suele ser muy delicado con su sintaxis hasta con los espacios que hay que llevar y el acomodo de nuestro código, aunque estemos escribiendo las operaciones correctamente.

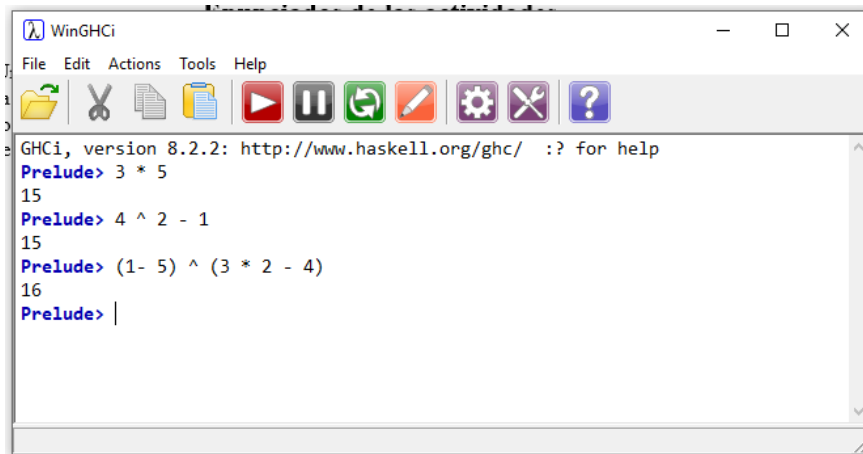
[1] Haskell es un lenguaje de programación para computadoras. Lenguaje polimórficamente tipificado, perezoso, puramente funcional, muy diferente a la mayoría de los otros lenguajes de programación. El lenguaje recibe su nombre en honor a Haskell Brooks Curry, por sus trabajos en lógica matemática que sirvieron como fundamento para el desarrollo de lenguajes funcionales.

“Esta práctica me permitirá pensar más lógicamente a lo que estoy acostumbrada a programar, se supone que al resolver un problema matemático Haskell lo resuelve en una o dos líneas, a lo que he leído antes de comenzar, se necesita un nivel muy alto de conocimiento de matemáticas para comprender más el lenguaje. No soy muy buena con cálculos matemáticos, espero aprender al menos lo básico en cálculos en el lenguaje y que efectivamente pueda dominar al menos con los ejercicios que realizare y la sintaxis básica si necesito realizar algún calculo o imprimir un “Hola Mundo”. Lo explicare con capturas de pantalla en intérprete para realizar los ejercicios nuevamente y comprender mejor que es lo que sucede al estar leyendo.

Enunciados de las actividades

Una vez descargado Haskell, podremos abrir “WinGHCi”, el intérprete de Haskell, al abrirse la ventana veremos Prelude> el cual es “prompt” por default de GHCi. En el comenzaremos los siguientes ejercicios con expresiones matemáticas directamente y obtendremos una respuesta inmediata de bajo de esta.

1- Simple Expressions




The screenshot shows the WinGHCi window with the following content:

```
WinGHCi
File Edit Actions Tools Help
[Icons]
GHCi, version 8.2.2: http://www.haskell.org/ghc/ :? for help
Prelude> 3 * 5
15
Prelude> 4 ^ 2 - 1
15
Prelude> (1- 5) ^ (3 * 2 - 4)
16
Prelude> |
```

En la siguiente captura de pantalla se muestra cómo es que se debe de escribir una operación matemática sencilla y como es que el intérprete de Haskell la resuelve fácil, rápido y sencillamente.

Captura de Pantalla 1. “Operaciones sencillas”



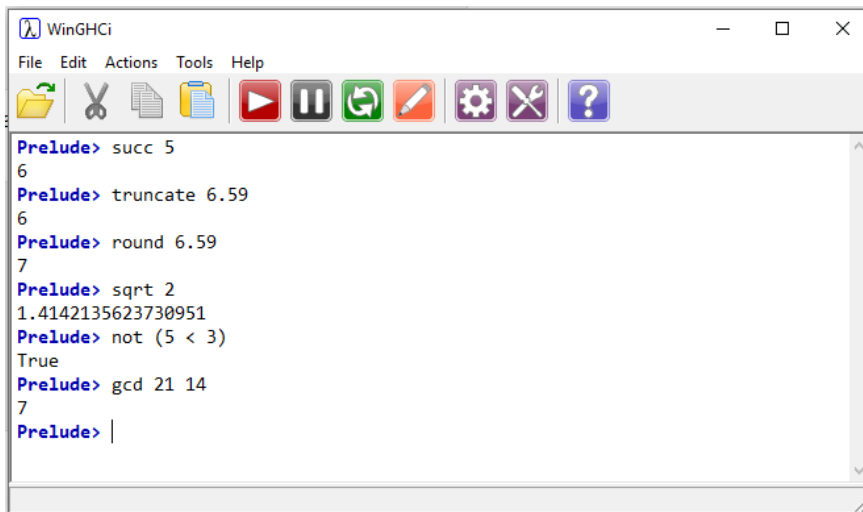
The screenshot shows the WinGHCi window with the following content:

```
WinGHCi
File Edit Actions Tools Help
[Icons]
Prelude> "Hola"
"Hola"
Prelude> "Hola" ++ ", Haskell"
"Hola, Haskell"
Prelude> |
```

En la siguiente captura de pantalla se muestra la sintaxis que se debe usar para escribir una palabra. En la primera línea se escribe una expresión “Hola”, en la segunda línea lo que se está haciendo es escribir un “Hola” y concatenarlo con ++ y la palabra Haskell”

para que al imprimirse se junten las palabras y se vea como lo imprime en la respuesta de la segunda línea.

Captura de Pantalla 2. “Sintaxis para escribir una palabra y concatenar”



```
WinGHCi
File Edit Actions Tools Help
[Icons]
Prelude> succ 5
6
Prelude> truncate 6.59
6
Prelude> round 6.59
7
Prelude> sqrt 2
1.4142135623730951
Prelude> not (5 < 3)
True
Prelude> gcd 21 14
7
Prelude> |
```

En el caso de las funciones, estas se realizan poniendo los argumentos directamente después de la función. Como se ve en la siguiente captura de pantalla no es necesario poner paréntesis.

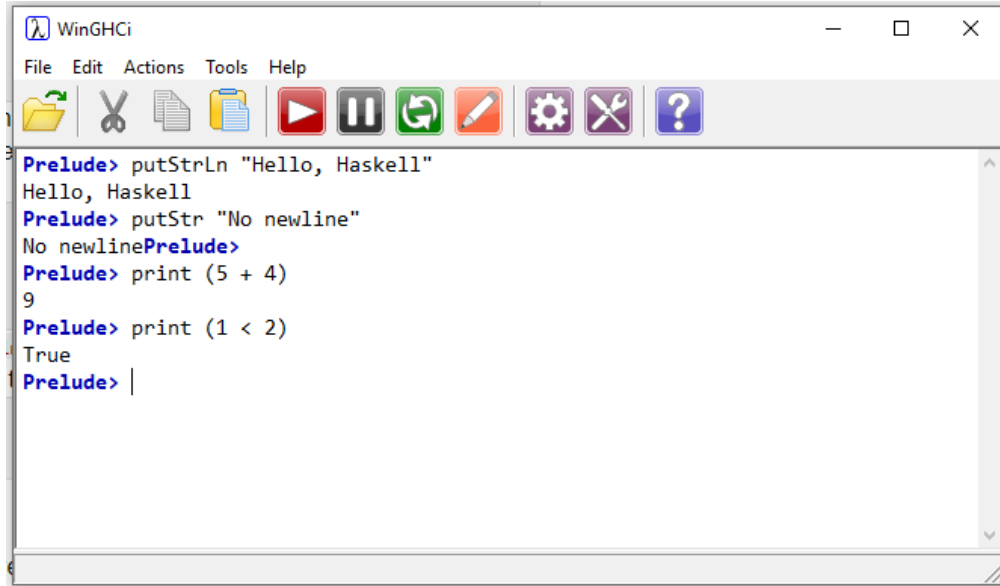
Captura de Pantalla 3. “Uso de Funciones”

Las funciones utilizadas nos muestran:

Funciones	
“succ”	El número siguiente al que escribamos.
“truncate”	El número que se escriba, pero sin decimal.
“round”	El número redondeado si es que lo escribimos con decimal.
“sqrt”	Nos muestra raíz cuadrada.
“not”	Nos muestra True/False dependiendo del tipo de operación que se realice.
“gcd”	Nos muestra la diferencia de numero entre los dos números que escribimos, pero con números pares.

2- Console

Las acciones de E/S pueden usarse para leer y escribir en la consola. Algunos comunes incluyen:

A screenshot of the WinGHCi Haskell interpreter window. The window has a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for file operations (folder, scissors, document), execution (play, pause, refresh), editing (pencil), settings (gear), and help (question mark). The main text area shows the following interaction:

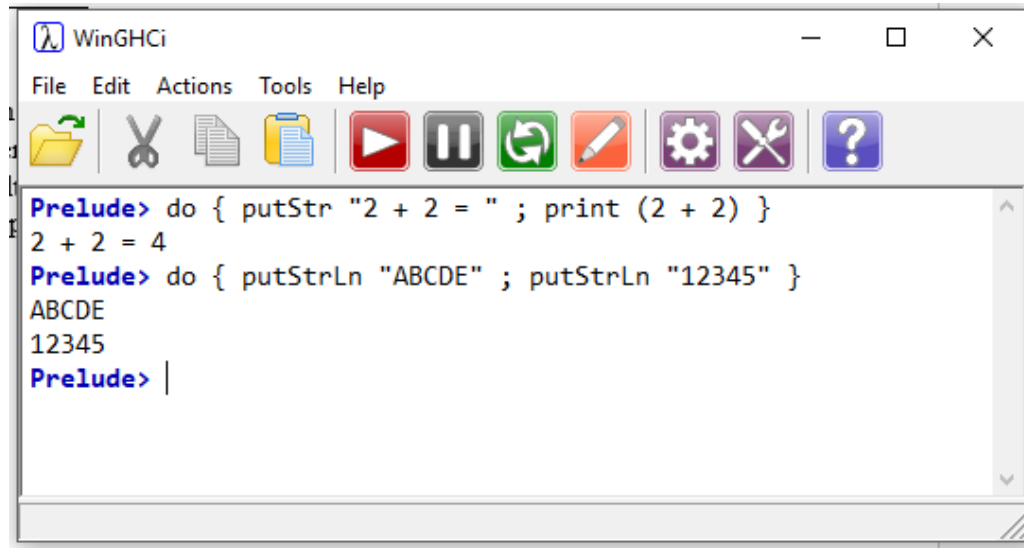
```
Prelude> putStrLn "Hello, Haskell"
Hello, Haskell
Prelude> putStr "No newline"
No newlinePrelude>
Prelude> print (5 + 4)
9
Prelude> print (1 < 2)
True
Prelude> |
```

Captura de pantalla 4. “Saltos de línea”

Como se muestra en la captura de pantalla “**putStrLn**” para escribir e imprimir con salto de línea y “**putStr**” sin salto de línea, como “**print**” solamente para imprimir.

Los “**putStr**” y “**putStrLn**” son funciones de salida de cadena a la terminal. Los “**print**” tienen la función genera cualquier tipo de valor. Si un “**print**” una cadena y tiene comillas a su alrededor. Pues si necesita múltiples acciones de E/S en una expresión, se puede usar “**do**” bloquear. Y las acciones están separadas por punto y coma.

En la siguiente captura de pantalla se muestra cómo es que “do” bloquea la impresión, seguimos de corchetes, la función “**putStr**” que manda a imprimir nuestro “**2 + 2 =**”, después separamos con punto y coma y seguimos con la función “**print**” que nos manda a imprimir el resultado de la suma de la operación matemática (2 + 2), y terminamos cerrando el corchete.

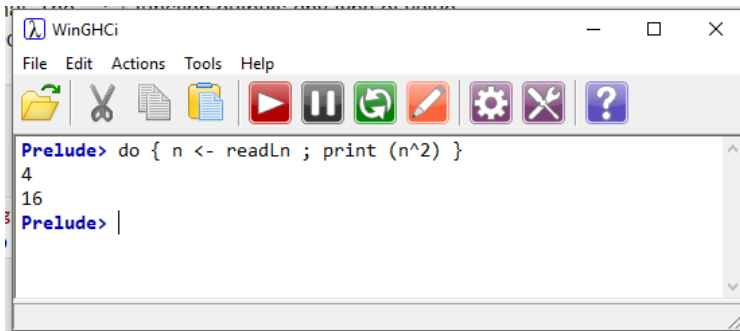


```
Prelude> do { putStr "2 + 2 = " ; print (2 + 2) }
2 + 2 = 4
Prelude> do { putStrLn "ABCDE" ; putStrLn "12345" }
ABCDE
12345
Prelude> |
```

Captura de pantalla 5. “Uso de “do”!”

Ahora, en la segunda línea se muestra nuevamente “do” bloqueando, el corchete abierto, la función “**putStrLn**” para imprimir salto de línea nuestro texto que es “**ABCDE**”, seguido de punto y coma para separar de la siguiente función “**putStrLn**”, que también nos imprime con salto de línea el texto con numeración “**12345**”, seguimos con un cierre de corchetes para que todo quede escrito dentro de nuestro “do” y listo. Nuestro resultado son nuestros dos textos uno de bajo del otro.

La lectura se puede hacer con “**getLine**”, el cual regresa un String. También se hace con un “**readLn**” que devuelve el tipo de valor que se desee. Y el símbolo “<-”, se usa para asignar un nombre al resultado de una acción de E/S.



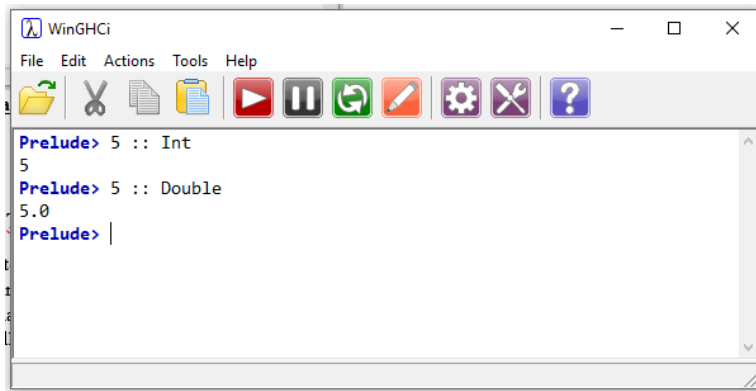
```
WinGHCi
File Edit Actions Tools Help
[Icons]
Prelude> do { n <- readLn ; print (n^2) }
4
16
Prelude> |
```

Como se muestra en la captura de pantalla se puede ver que el número “4” es el número cualquiera que escribimos al escribir toda la función con la operación matemática y el número 16 es el resultado de dicha operación.

Captura de pantalla 6. “Lectura”

3- Simple Types

Hasta el momento, no se ha mencionado ninguna declaración de tipo único. Eso es porque Haskell escribe inferencia. Por lo general, no tiene que declarar tipos a menos que lo desee. Si se quiere declarar tipos, se usa “ :: ” para hacerlo, como se muestra en el ejemplo de la captura de pantalla siguiente.

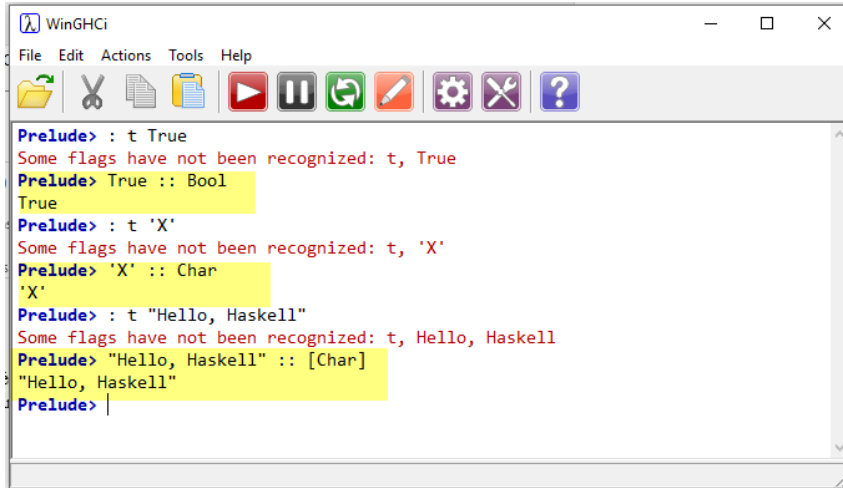


```
WinGHCi
File Edit Actions Tools Help
[Icons]
Prelude> 5 :: Int
5
Prelude> 5 :: Double
5.0
Prelude> |
```

Captura de pantalla 7. “Indiferencia en Haskell”

Los tipos y clases de tipos, siempre comienzan con letras mayúsculas en Haskell. Las variables siempre comienzan con letras minúsculas. Esta es una regla del lenguaje, no una convención de nomenclatura.

También se puede escribir o preguntar en “ghci” qué tipo ha elegido para algo. Esto es útil porque generalmente no tienes que declarar tipos. Como se muestra en la parte sombreada en la siguiente captura de pantalla.



```
WinGHCi
File Edit Actions Tools Help
[Icons]

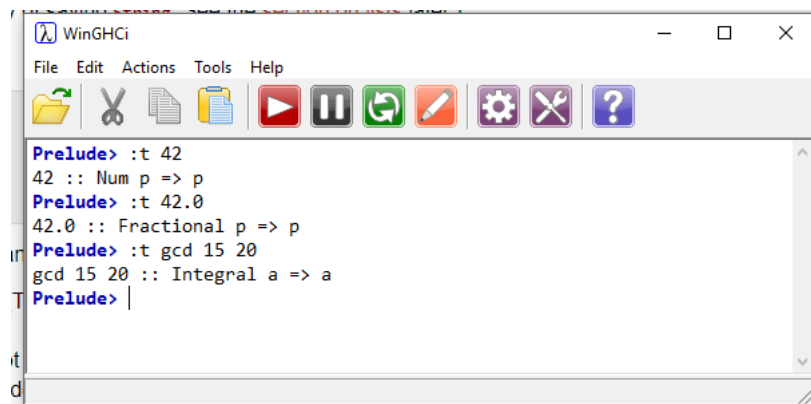
Prelude> :t True
Some flags have not been recognized: t, True
Prelude> True :: Bool
True
Prelude> :t 'X'
Some flags have not been recognized: t, 'X'
Prelude> 'X' :: Char
'X'
Prelude> :t "Hello, Haskell"
Some flags have not been recognized: t, Hello, Haskell
Prelude> "Hello, Haskell" :: [Char]
"Hello, Haskell"
Prelude> |
```

[**Char**] es otra forma de String.

Captura de pantalla 8. “Declaración en Haskell”

Ahora en la siguiente captura de pantalla, estos tipos usan “clase de tipos”, los cuales quieren decir:

- 42 se puede usar como cualquier tipo de número. Por esto se declara 5 ya sea como un “**int**” o una “**double**”.
- 42.0 puede ser cualquier tipo fraccional, pero no un tipo integral.
- gcd 15 20 (donde se llama a la función), puede ser de cualquier tipo integral, pero no de tipo fraccionario.



```
WinGHCi
File Edit Actions Tools Help
[Icons]

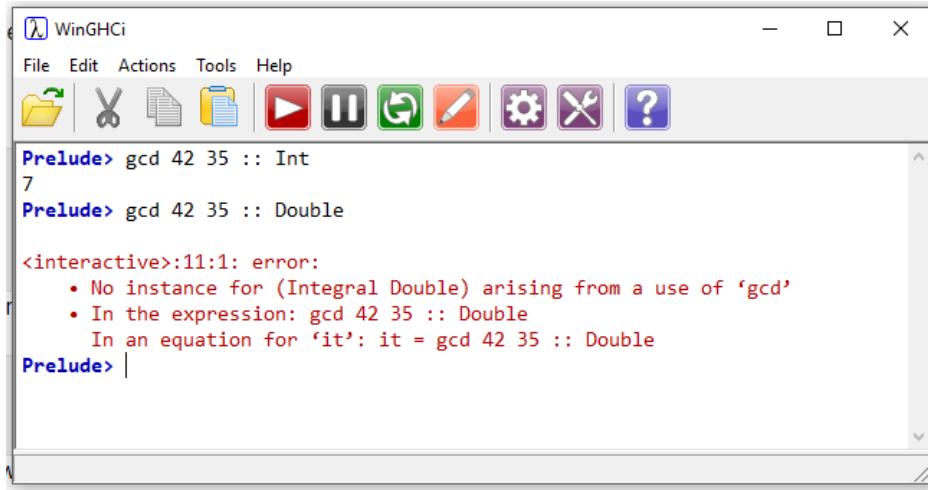
Prelude> :t 42
42 :: Num p => p
Prelude> :t 42.0
42.0 :: Fractional p => p
Prelude> :t gcd 15 20
gcd 15 20 :: Integral a => a
Prelude> |
```

Captura de pantalla 9. “Clase de tipos”

Hay cinco tipos numéricos en el “preludio” de Haskell.

- **Int**: que es un número entero con al menos 30 bits de precisión.
- **Integer**: que es un número entero con la precisión ilimitada.
- **Float**: que es un número de punto flotante de precisión simple.
- **Double**: que es un número de punto flotante de doble precisión.
- **Rational**: que es un número de fracción, sin error de redondeo.

Las cinco son instancias de “**Num**” tipo de clase. Los primeros dos casos de Integral, y los últimos tres son casos fraccionales pidiéndolo todo junto.

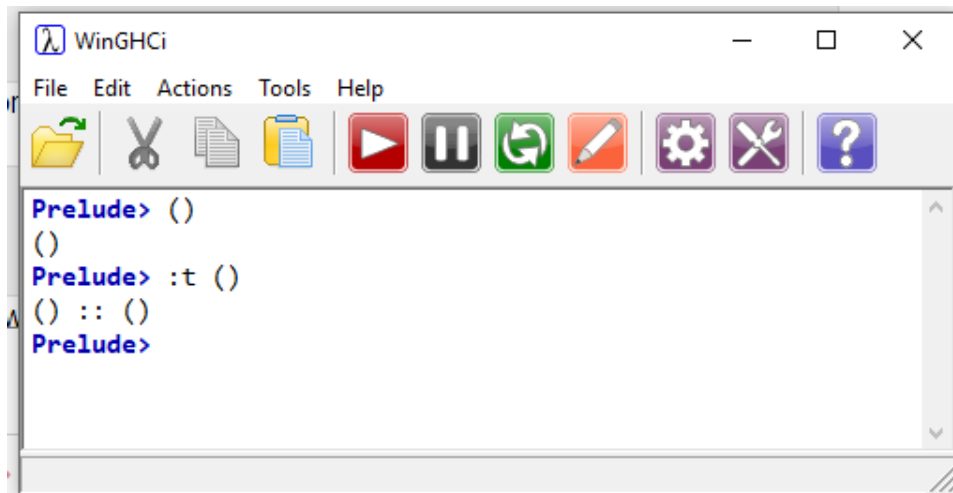


```
WinGHCi
File Edit Actions Tools Help
[Icons]
Prelude> gcd 42 35 :: Int
7
Prelude> gcd 42 35 :: Double
<interactive>:11:1: error:
  • No instance for (Integral Double) arising from a use of 'gcd'
  • In the expression: gcd 42 35 :: Double
    In an equation for 'it': it = gcd 42 35 :: Double
Prelude> |
```

Captura de Pantalla 10. “Tipo de clase Num”

El tipo final es (), pronunciado “**Unit**”. Solo tiene un valor.

Se puede pensar que esto es similar a la palabra clave “**void**” en los lenguajes de la familia C. Puedes regresar () de una acción en E/S si no desea devolver nada.



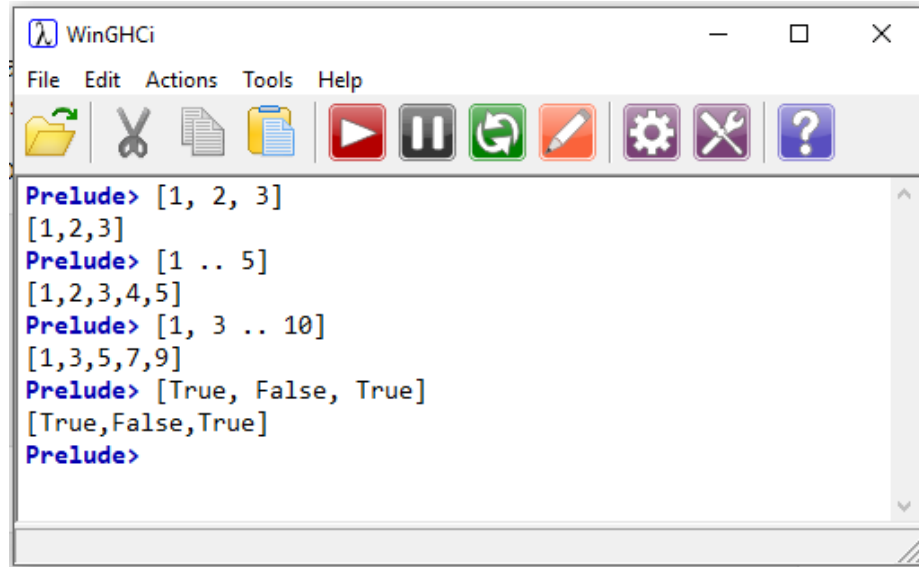
```
WinGHCi
File Edit Actions Tools Help
[Icons]
Prelude> ()
()
Prelude> :t ()
() :: ()
Prelude>
```

Captura de pantalla 11. “Tipo de clase Unit”

4- Datos Estructurados

Los tipos de datos básicos se pueden combinar fácilmente de dos maneras: lista, que van entre [] y tuplas, que entran en ().

Las listas se usan para contener múltiples valores del mismo tipo, como se muestra en la siguiente captura de pantalla.

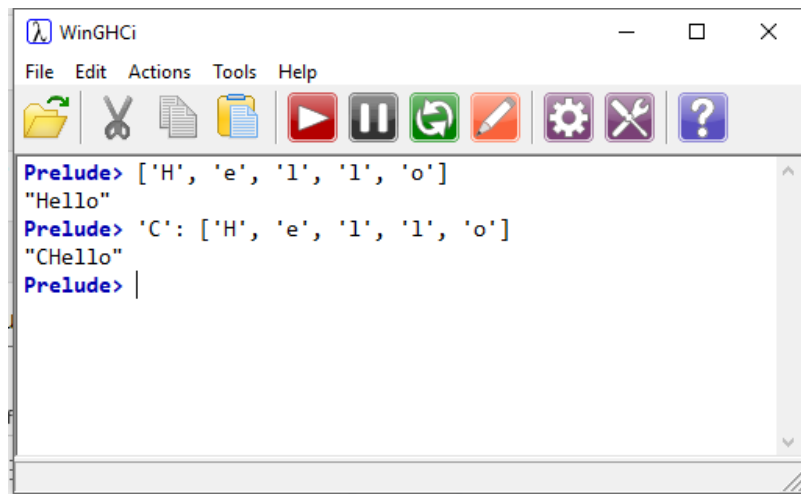


```
WinGHCi
File Edit Actions Tools Help
[Icons]

Prelude> [1, 2, 3]
[1,2,3]
Prelude> [1 .. 5]
[1,2,3,4,5]
Prelude> [1, 3 .. 10]
[1,3,5,7,9]
Prelude> [True, False, True]
[True,False,True]
Prelude>
```

Captura de Pantalla 12. “Tipos de datos básicos Listas”

Las cadenas son listas de caracteres. Los “:” el operador agrega un artículo al principio de una lista.

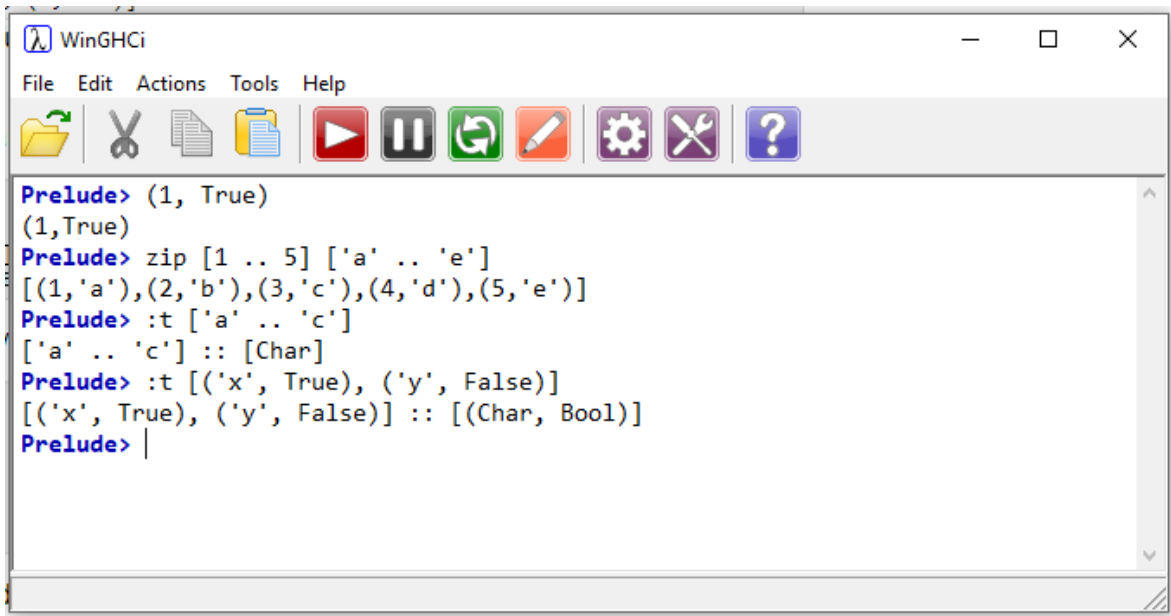


```
WinGHCi
File Edit Actions Tools Help
[Icons]

Prelude> ['H', 'e', 'l', 'l', 'o']
"Hello"
Prelude> 'C': ['H', 'e', 'l', 'l', 'o']
"CHello"
Prelude> |
```

Captura de pantalla 13. “Cadena de caracteres”

Las tuplas tienen un número fijo de valores, que pueden tener diferentes tipos. En el siguiente ejemplo se verá utilizado “**Zip**”, el cual es una función de biblioteca que convierte dos listas en una lista de tuplas. Y los tipos son bastante predecibles.

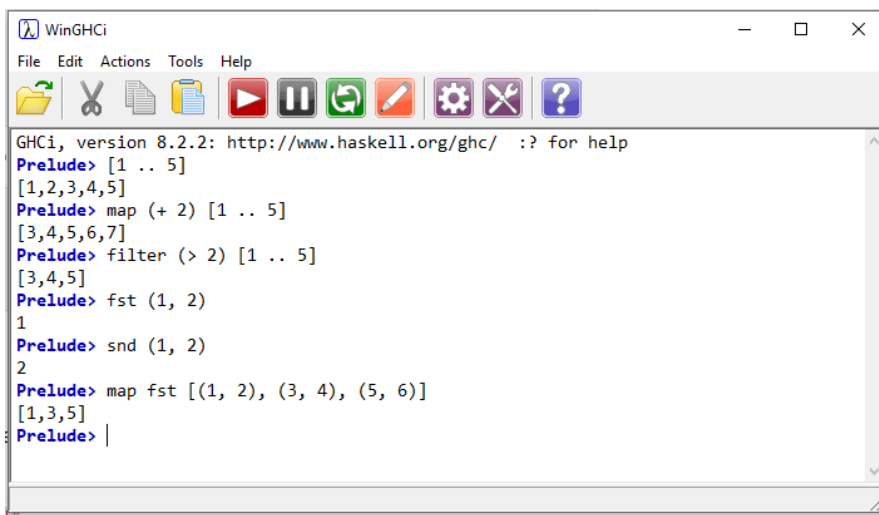


```
WinGHCi
File Edit Actions Tools Help
[Icons]

Prelude> (1, True)
(1,True)
Prelude> zip [1 .. 5] ['a' .. 'e']
[(1,'a'),(2,'b'),(3,'c'),(4,'d'),(5,'e')]
Prelude> :t ['a' .. 'c']
['a' .. 'c'] :: [Char]
Prelude> :t [('x', True), ('y', False)]
[('x', True), ('y', False)] :: [(Char, Bool)]
Prelude> |
```

Captura de pantalla 14. “Numero de valores, uso de Zip”

Las listas se usan mucho en Haskell. Hay varias funciones que hacen cosas mucho más fáciles de lo que son normalmente, hay dos funciones en pares ordenados (Tuplas de dos elementos), como se muestra en la siguiente captura de pantalla.



```
WinGHCi
File Edit Actions Tools Help
[Icons]

GHCi, version 8.2.2: http://www.haskell.org/ghc/ :? for help
Prelude> [1 .. 5]
[1,2,3,4,5]
Prelude> map (+ 2) [1 .. 5]
[3,4,5,6,7]
Prelude> filter (> 2) [1 .. 5]
[3,4,5]
Prelude> fst (1, 2)
1
Prelude> snd (1, 2)
2
Prelude> map fst [(1, 2), (3, 4), (5, 6)]
[1,3,5]
Prelude> |
```

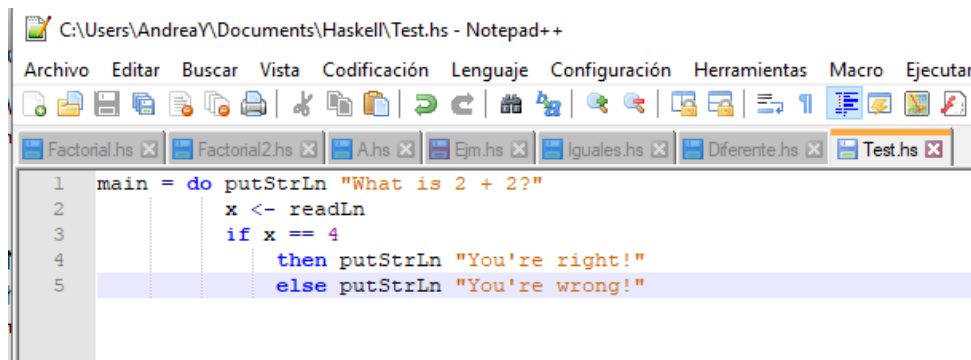
Podemos observar como con “**map**” nos busca los numero después del dos de la lista con los parámetros que le indicamos. Como “**filter**”, filtra que nos muestre los números mayores al que le indicamos, dentro de los parámetros de lista.

Captura de pantalla 15. Listas, Tuplas

“fst” nos muestra el primer número de escritos en tuplas. “snd” nos muestra el siguiente número de los que escribimos en tuplas. Si juntamos las funciones “map” seguido de “fst” y una lista con parámetros y pares de números cualesquiera escritos en tuplas dentro de la lista, estamos pidiendo que nos muestre la búsqueda de cada primer número de pares que escribimos en los parámetros.

5- Definiciones de Funciones

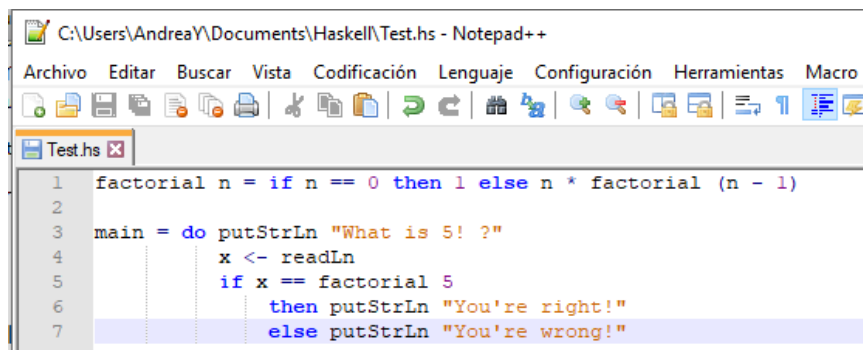
Escribamos una definición de una acción de E/S. Llamada main.



```
C:\Users\AndreaY\Documents\Haskell\Test.hs - Notepad++
Archivo  Editar  Buscar  Vista  Codificación  Lenguaje  Configuración  Herramientas  Macro  Ejecutar
1  main = do putStrLn "What is 2 + 2?"
2          x <- readLn
3          if x == 4
4              then putStrLn "You're right!"
5              else putStrLn "You're wrong!"
```

Captura de Pantalla 16 “Llamada a main”

Ahora, tiene una definición de función, llamada factorial, agregamos un encabezado de modulo, que está en buen estado. Hay una función al igual que las funciones integradas, pero se puede llamar como factorial 5 sin necesidad de paréntesis.



```
C:\Users\AndreaY\Documents\Haskell\Test.hs - Notepad++
Archivo  Editar  Buscar  Vista  Codificación  Lenguaje  Configuración  Herramientas  Macro
1  factorial n = if n == 0 then 1 else n * factorial (n - 1)
2
3  main = do putStrLn "What is 5! ?"
4          x <- readLn
5          if x == factorial 5
6              then putStrLn "You're right!"
7              else putStrLn "You're wrong!"
```

```
C:\Users\AndreaY\Documents\Haskell>ghc --make Test.hs
Linking Test.exe ...

C:\Users\AndreaY\Documents\Haskell>Test
What is 5! ?
120
You're right!
```

Captura de pantalla 17. “Ejemplo Factorial”

Resultados

Desde que comencé el reporte tenía una expectativa completamente diferente del lenguaje porque cuando se vieron los ejemplos (ejercicios) en clase, fue algo rápido y solo los probábamos para ver cómo es que estos funcionaban, se entendían en el momento cuando el profesor los explicaba, pero realmente no es algo que se quedara para después recordarlo. Empecé con dificultad porque el pensar que es un lenguaje donde tienes que tener la mente abierta para las matemáticas para mi es algo tedioso, al compilar tuve algunos problemas porque el lenguaje es muy delicado con cualquier salto de línea, espacio o posición de código, pero después con el paso de las practicas ya no fue tan complicado y lo fui dominando. Cuando comencé con las primeras prácticas, para poder describirlas necesitaba comprender que es lo que se realizaba en código y era satisfactorio ver que a pesar de que es lo básico que debemos saber para comenzar con el lenguaje, no es tan difícil a como pensé al inicio, el verlo, pensarlo y después realizarlo para ver cómo es que se forma el resultado o intercambiar números diferentes a los de los ejemplos para comprobar que sea verdad la instrucción y resultado de la función me facilitaba mucho el trabajo a como avanzaba de ejercicio a ejercicio y cada vez era más comprensible a pesar de que a cada ejercicio se le metía algo más a lo que se iba explicando. Siento que al ver realizado las prácticas y luego explicarlas con captura de pantalla me hizo como estudiar o comprender el doble cada ejercicio y el propósito de esta práctica y reporte tuvo su objetivo, pues no necesite buscar información en alguna otra página o ver algún video hasta el momento.

Conclusiones

Lenguaje de programación funcional que cumple lo que ofrece, hace realmente más fácil la distribución y desarrollo de los códigos para los programas, ya que lo que un programa normal en algún otro lenguaje se hace en 10 líneas de código, Haskell logra realizar el mismo trabajo con el resultado hasta más limpio con tan solo tres líneas, es un lenguaje que no debe ser menospreciado, abarca mucho terreno en sus cortas funciones y validaciones, con esto nos permite tener en rango menos errores, son limpios y fáciles de entender, te cambia la manera de ver la forma de programar porque te hace pensar mucho más funcional matemáticamente hablando.

Bibliografía

[1] Introducción a Haskell. *Basado en un escrito por Simon Peyton Jones.*

Disponible en:

<https://wiki.haskell.org/Es/Introduccion>

Aprende Haskell en 10 minutos. *Una pequeña y rápida descripción de lo que es Haskell.*

Disponible en:

https://wiki.haskell.org/Learn_Haskell_in_10_minutes