Logotipo

Descripción generada automáticamenteImagen que contiene Logotipo

Descripción generada automáticamenteInstituto Politécnico Nacional

Escuela Superior de Computo

Materia:

Paradigmas de Programación

Grupo:

3CM3

Nombre:

Rodriguez Sanchez Daniel

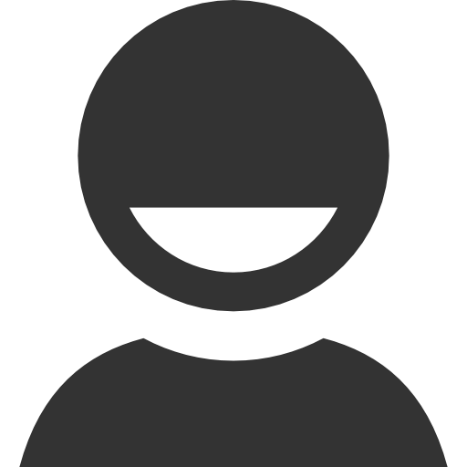
Méndez Pichardo Cesar Daniel

Monroy Mora Arturo

Rubio Haro Diego

Luna Martínez Jesús Alberto Dorian

Titulo:

*“Manual de Usuario”*

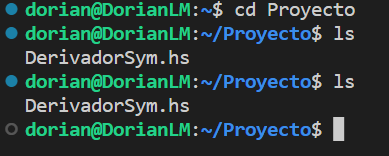
Manual de Usuario

* Forma de compilar el programa

Para compilar un programa en haskell es necesario tener el compilador interactivo de haskell GHCI el cual se ocupa exclusivamente en un entorno Linux (Ubuntu, wsl 2, debian, etc) en este caso lo mostraremos en el entorno WSL con ayuda de visualcode (para mostrar el código pero no es necesario).

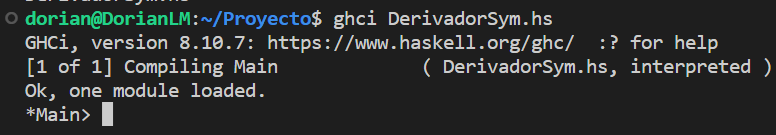
Después de que ya tenemos descargado el compilador GHCI necesitamos el archivo fuente y la dirección en donde esta guardado dicho archivo ya que debemos posicionarnos en la terminal en la ubicación del archivo fuenta para poder realizar la compilación.

Debemos usar el comando cd para viajar entre rutas dentro de la termial y con el comando ls podemos visualizar el contenido de la carpeta en la que estamos situados.



Como se ve en la imagen tenemos la carpeta Proyecto la cual esta alojada en la raíz de nuestro sistema entonces realizamos un cd para movernos a esta y poder encontrar nuestro archivo fuente “DerivadorSym.hs” es importante que el archivo sea .hs ya que esta nos denotara que es un archivo realizado exclusivamente para el lenguaje haskell.

Ahora que ya estamos en la ruta correcte debemos entrar en el modo interactivo de haskell y poder compilar el archivo, esto se realiza con el siguiente comando: ghci “Nombre\_del\_archivo.hs”



Podemos estar seguros que el programa compilo correctamente ya que no salto ningún error el cual puede verse de la siguiente manera:



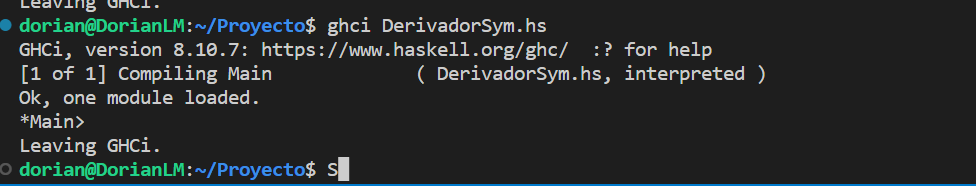
Seguido con los errores que tenga el código por ejemplo:

Texto

Descripción generada automáticamente

Si te saltan uno o mas errores de este tipo es porque se modifico el código y no logra compilarse correctamente, **es de vital importancia que no se modifique el código a menos que tengas el conocimiento adecuado en la materia.**

Para volver a realizar la compilación es necesario salir del modo interativo de haskell para esto debemos teclear CTRL + D y volveremos a la terminal de Linux



* Ejecución y funcionamiento del programa

Tenemos ciertas reglas que simplifican funciones usando unas coicidencias de patrones, dichas simplificaciones lo podemos tomas como si fueran axiomas fundamentales para realizar operaciones que son necesarias para la derivación pero debido a que son muy concurrentes se definen para que al momento de derivar solo se mande a llamar la función de dicho axioma, algunos de estos son:

Simplificaciones para la suma

simplificar (BinExp Sumar (Const a) (Const b)) = Const (a + b) -> La suma común de dos números

simplificar (BinExp Sumar a (Const 0)) = simplificar a -> La suma de un numero con cero

simplificar (BinExp Sumar (Const 0) a) = simplificar a ->La suma de cero a un numero

simplificar (BinExp Sumar a b) = BinExp Sumar (simplificar a) (simplificar b) -> La suma de dos números sin importar el orden

simplificar (BinExp Restar (Const a) (Const b)) = Const (a - b) ->La resta común de dos numeros

simplificar (BinExp Restar a (Const 0)) = simplificar a -> La resta de un numero con cero

simplificar (BinExp Restar (Const 0) a) = simplificar (UnExp Minus a) ->La resta de cero con un numero esto da un numero negativo

simplificar (BinExp Restar a b) = BinExp Restar (simplificar a) (simplificar b)

simplificar (BinExp Pow (Const a) (Const b)) = Const (a \*\* b)

simplificar (BinExp Pow a (Const 0)) = Const 1 ->Un numero elevado a una potencia cero es 1

simplificar (BinExp Pow a (Const 1)) = simplificar a ->Un numero elevado a una potencia 1 es igual al mismo numero

simplificar (BinExp Pow (BinExp Pow a (Const b)) (Const c)) = BinExp Pow (simplificar a) (Const (b \* c))

simplificar (BinExp Pow a b) = BinExp Pow (simplificar a) (simplificar b)

Y todas estas reglas de simplificación que se definieron en el programa se utilizan en la siguiente función polimórfica la cual el usuario es la que tiene que utilizar para poder simplificar las funciones y derivadas correspondientes, la función que se encarga de simplificar hasta lo mas mínimo posible es la siguiente:

Texto

Descripción generada automáticamente

Por ende el empleo de dicha función para el usuario es la siguiente:

Por cada expresión de nuestra función se debe simplificar por ejemplo:

Main = do – Acciones que se realizaran en el main

Printf(simplifyFull (diff(BinExp Pow (Var ‘x’)(Const4))))—impression de pantalla aqui Podemos ver como solo se esta simplificando después de obtenerla

Printf(simplifyFull (diff(simplifyFull((BinExp Mul (Var ‘x’)(BinExp Mul (Const 5)))))))—Aqui estamos simplificando antes de derivar y después de simplificar derivamos de tal manera que después simplificamos la misma derivada hasta su mínima expresión.

Ya compilado el programa podemos realizar el funcionamiento de este, para realizar una derivada de debemos poner el comando *derivar(operación)* dentro del modo interativo de haskell, las operaciones posibles en el derivador son:

derivar(Const a)-> Es la forma de derivar una constante

derivar(Var a)-> Es la forma de derivar una variable

derivar(BinExp sumar a b)-> Es la forma de derivar una función de suma ejemplo x+2 donde a es x y b es 2

derivar (BinExp Restar a b) -> Es la forma de derivar una función de resta x-2 por ejemplo

derivar (BinExp Multiplicar a b) -> Es la forma de derivar una función de multiplicacion

derivar (BinExp Dividir a b) -> Es la forma de derivar una función de derivación

derivar (BinExp Pow (UnExp Sin a) (Const b)) ->Tenemos una regla de la cadena para seno por ejemplo seno(90)(90)

derivar (BinExp Pow a (Const b)) ->Forma de derivar una pontencia

derivar (BinExp Pow a b) = error "derivar potencias, sin parentesis no sera admitida la operacion"

derivar (UnExp Sin a) -> Forma de derivar la función seno

derivar (UnExp Cos a) -> Forma de derivar la función conseno

derivar (UnExp Tan a) -> Forma de derivar la función tangente

derivar (UnExp Sec a) -> Forma de derivar la función secante

derivar (UnExp Log a) -> Forma de derivar la función de un logaritmo natural

derivar (UnExp Sqrt a) -> Forma de derivar una raíz cuadrada

Las funciones que tienen BinExp es porque requieren de dos parámetros para poder realizar la operación y las funciones que tienen UnExp es porque solo es necesario un parámetro para realizar la operación, con estas formulas básicas es posible realizar infinidad de derivadas mas complejas mas adelante unos ejemplos:

1. f(x)= sen(x+2) f'(x)= cos(x+2)

Haskell: derivar (UnExp Sin (BinExp Sumar(Var 'x')(Const 2)))

Texto

Descripción generada automáticamente

1. f(x)= sen(cos(3x^3)) f'(x)= cos(cos(3x^3)) \* (-sen(3x^3) \* 9x^2)

Texto

Descripción generada automáticamente con confianza mediaHaskell: derivar (UnExp Sin (UnExp Cos (BinExp Pow (BinExp Multiplicar (Const 3) (Var 'x')) (Const 3))))

1. f(x)= ln (tan((senx)^2)) f'(x)= 1/tan((senx)^2)) \* sec^2 ((senx)^2) \* 2senxcosx

Texto

Descripción generada automáticamenteHaskell: derivar (UnExp Log(UnExp Tan (BinExp Pow (UnExp Sin (Var 'x')) (Const 2))))