anleon@ucab.edu.ve

Algoritmos y Programación I.

Guía de Ejercicios y Comandos para el uso de Haskell.

Ejercicios, Soluciones (sugeridas), Pruebas Cortas, Comandos, Uso de tu Cuenta Shell y otros tips.

Por Angel Luis León Molina.

Que hacer la primera vez que utilices tu cuenta.

- *Comandos básicos para el shell de UNIX (Aixterm)*
- *Como entrar a Hugs*

Comandos Básicos (UNIX)

Lo más seguro es que te den una cuenta con mayores prioridades que las otras cuentas del resto de los estudiantes de otras facultades, eso es porque estas en ingeniería, además de que vas a necesitar el shell de UNIX, que es algo parecido a utilizar DOS.

Para entrar al sistema lo puedes hacer desde una de las estaciones en el laboratorio de computación de ingeniería (por lo menos hasta el semestre pasado era así [Marzo-Junio 98]) o desde un PC regular utilizando telnet y conectándote a ZEUS u otro servidor de la universidad.

Una vez que hayas entrado al sistema con tu login y password, si tienes una de las cuentas privilegiadas aparecerá algo así:

```
[zeus]:/preg(u otro directorio donde te hallan asignado)/TuLogin >
```

Si aparece algo como esto estamos listos para comenzar, no te preocupes si no te sale un menú como el del resto de los usuarios que veras por allí en los salones de computación, ellos tienen “cuentas restringidas”, la tuya utiliza un shell llamado ‘bash’, tiene ciertas restricciones pero aun puedes hacer travesuras (cuidado con el CAI). Esas cuentas (las restringidas) las obtienen estudiantes de otras carreras una vez que llegan al 4to año, personal de la Universidad (profesores, secretarias) y cuando consigues una beca trabajo, o si trabajas en la *Oficina Web.*

Si te aparece el menú balurdo, ve inmediatamente al CAI (MODULO 6 – PISO 3) para ver que sucede.

Ahora bien que puedes hacer con este shell...? ...todo! Para empezar pide un directorio

```
> ls
```

ls (list): Hace que aparezca un directorio con los archivos pertenecientes a la carpeta donde estas.

Si quieres saber mas sobre este comando tipea:

```
> man ls
```

Ahora si quieres cambiar de directorio lo puedes hacer de este modo (cd -> change dir):

```
> cd ElDirectorio
```

Si quieres volver al directorio anterior...

```
> cd ..
```
(con un espacio antes de los dos puntos)

Comandos en forma de Ejemplo (Nombre: Función del comando)

`>cd.` (change dir: Te devuelve al directorio de partida)

`>mkdir NuevoDirectorio` (make dir: Crea un directorio llamado "NuevoDirectorio")

`>rmdir NuevoDirectorio` (remove dir: Borra el directorio "NuevoDirectorio")

`>rm Archivo.Algo` (remove: Borra el Archivo.Algo)

`>finger`

`>f` (finger: Muestra una lista de los usuarios en línea)

Nota: Si los usuarios son tantos que no caben en la pantalla puedes utilizar el filtro

" | more ", por ejemplo: > finger | more (Esto también sirve con el comando ls)

Ahora al presionar la Barra Espaciadora iras bajando pagina por pagina y al presionar Enter, bajaras línea por línea, o alrevez...

`> finger loginCualquiera` (Te consigue la información de 'loginCualquiera')

`> cp` (copy : Copia archivos)

`> pwd` (pwd: Muestra el nombre del directorio actual)

`> talk loginCualquiera` (sirve para establecer una conversación con "loginCualquiera")

Para contestar a un talk todo lo que hay que hacer es copiar talk y el nombre del login que pide hablar contigo. Supongamos que derrepente aparece el siguiente mensaje en tu monitor...

>

`Talk request from` anleon@ucab.edu.ve

Eso significa que anleon quiere hablar contigo, para responder solo tipea:

`>talk anleon`

Y la pantalla se dividirá en dos. Las letras de la parte inferior provienen del terminal de anleon.

`>mesg n` (message no: Sirve para que nadie te moleste con un talk, veras que está activado cuando al hacer un "finger", porque al lado de tu login aparece un " * " asterisco)

`>mesg y` (message yes: Para que te puedan hablar)

Para re-nombrar directorios:

`> move directorio1/ directorio2/`

`> ls`

../ ./ directorio2/

El directorio "directorio1" ahora se llama "directorio2" gracias al comando *move*, que sirve para mover archivos de un lado a otro.

`> bye`

`> logout` (sirven para terminar la sesión)

Mandando E-mails, Ftp-ando, y Navegando por los Servidores

`> w` (watch: Veras que hacen los otros usarios y para averiguar su localización)

`> w loginCualquiera` (Ves lo que esta haciendo loginCualquiera)

Para mandar E-mails existen muchos programas, uno de mis favoritos es PINE, sin importar lo feo que se ve, hay que admitir que es uno de los mejores sistemas para enviar correos. Para utilizar este bebe lo único que tienes que hacer es esto:

```
> pine
```

Una vez que entres a PINE, tu login va a estar anotado, presiona Enter, coloca tu password (que Despues puedes cambiarlo si gustas.) y después veras un menú que habla por si solo. Por si tienes dudas presiona 'c' (Compose Message) y te aparecerá algo así:

```
To: fulano@servidor.extension

CC: (Aquí colocas otras direcciones de otras personas si quieres)

Subject: Aquí va el asunto.


  Aquí copias tu mensaje y lo envías creo que presionando Ctrl+Z o Ctrl+X , cualquier cosa abajo aparece un menú donde debe decir "¨Z Send" o algo parecido.  La cuestión es facilísimo.
```

Si no te gusta PINE y estas muy apurad@ por mandar ese correo, y no estas en Windows, puedes intentar el siguiente comando...

```
> comp fulano@servidor.extension
```

La máquina te pide los datos correspondientes y al final después que escribas correo presionas Ctrol-C, y ya.

Si no te gusta mandar correos desde el Shell de UNIX tienes la opción de mandarlos desde el famoso *SIMEON* estando en Windows.

Ahora bien, el servidor mas utilizado en la católica es ZEUS, al menos así lo ha sido todo este tiempo, pero eso no quiere decir que no hayan otros servidores, están otros en la universidad llamados:

* TANTALO (IP 200.2.8.26)
* TEMIS (IP 200.2.8.27)
*APOLO(IP200.2.8.1)
* ODIN (IP 200.2.12.250)
* ATENEA (IP 200.2.8.4)
*REA (IP 200.2.8.23)
* CALIOPE (IP 200.2.8.5)
* TALIA (IP 200.2.8.25)

El número al lado de cada “Host” es el “IP” que es su dirección en Internet. En la universidad existen muchos otros hosts, por ejemplo, en la dirección de deportes, en la Biblioteca (IP 200.2.9.162), y en casi todas las oficinas. Muchos de los Hosts restringen el acceso por razones de seguridad. Hace unos años un grupo de Hackers (al parecer de República Dominicana) acabaron con todo y ahora no se puede acceder a ninguno de los Hosts desde afuera de la Universidad. Lo que sí puedes hacer es enviar y recibir correos electrónico, y mandar archivos por medio de FTP. Así que ni sueñes que puedes hackear a la Universidad desde tu casa.

¿Cómo entro a uno de estos Hosts?

Muy fácil, tan solo haces telnet al servidor, metes tu login y tu password y ya estas dentro. Allí luego haces un finger y vez quien esta contigo.

Supongamos que estas en CALIOPE y no hay nadie en ese Host con quien hablar, lo más lógico es que alguien este en ZEUS, para ir a ZEUS solo debes hacer lo siguiente:

```
> telnet ZEUS
```

ó

```
> telnet 200.2.8.100
```

Cualquiera de estas dos formas sirve.

Si quieres Port-Surfear desde tu casa, Windows95 y versiones posteriores poseen TELNET, puedes acceder remotamente a la Universidad Central de Venezuela, el nombre de uno de los servidores es:

sagi1.ucv.edu.ve (IP Number 150.185.84.250)

Tambien puedes ir a neblina.reacciun.ve. Para todos estos debes tener un login y un password, sino no puedes entrar. De todos modos no son nada divertidos, para entretenerte tienes el World Wide Web.

¿Qué es FTP?

FTP significa *File Transfer Protocol*, Protocolo de Transferencia de Archivos... Esto sirve para mandar y traer archivos desde lugares distintos, o simplemente de tu diskette a ZEUS. Esto lo vas a necesitar para mandar tus archivos desde la casa a la Universidad o Viceversa. FTP lo puedes utilizar desde DOS, UNIX o Windows, es muy fácil de manejar. Solo tienes que estar pendiente del tipo de Archivo que mandas ya que

puede ser Binario ó ASCII. Se utiliza Binario para archivos de Imágenes, Sonido o algo que no sea simple, se utiliza ASCII para archivos de texto, o archivos que quieras editar.

Recuerda que para todo esto debes estar conectado a Internet, y que puede ser que esta información este sujeta a cambios para este semestre, ya que el CAI planeaba hacer algunos cambios en su sistema. De todas formas así sabrás como fue todo antes de que llegaras.

Así como haces telnet a otros servidores puedes hacer ftp para mandar y recibir archivos. De esta manera es como la mayoría de las personas que tienen sitios web actualizan sus paginas cada vez que lo necesitan.

Comenzando a trabajar en Haskell.

EMACS

Lo más probable es que te hayan dado un papel que dice "GNU Emacs Reference Card" y tu seguro que no tienes ni idea de que es Emacs. Emacs es un procesador de palabras, algo así como Block de Notas en Windows, pero mejor. Al comienzo vas a odiarlo por que es un problema acostumbrarse al manejo, si no te acostumbras a borrar hacia a atrás con SUPR o DEL es mejor que hagas tu tarea en tu casa y no la hagas en la Universidad.

Para entrar a Emacs, lo único que debes hacer es estar en el shell de UNIX y tipear:

```
> emacs
```

Si quieres ver un archivo en especifico por ejemplo "readme.txt " :

```
> emacs readme.txt
```

Y abrirás Emacs con el archivo readme.txt automáticamente.

Bueno si no te dieron el papelito con los comandos de Emacs aquí van algunos:

Ctrl+X+S	Salvar el Archivo
Ctrl+X + Ctrl+Z	Salir de Emacs
Ctrl+X+F	Encontrar un archivo
Ctrl+D	Borra después del carácter (Muy Util)
Ctrl+X+G	Busca una línea por su número.
Ctrl+K	Kill. Sirve para borrar todo lo que este adelante del cursor en esa línea.

Si quieres cerrar temporalmente emacs presiona Ctrl-Z, después que hayas hecho lo que tenias en mente puedes volver a Emacs tipeando:

```
> fg
```

Si has suspendido mas de una tarea para volver a emacs debes tipear:

```
> fg emacs
```

Esto sirve para todos los programas a excepción de PINE que no puede ser suspendido.

Ahora si no te place trabajar con emacs, tienes otro editor un poco más rudimentario pero un poco menos tedioso a la hora de editar el texto, se llama PICO y se parece mucho a PINE. Para utilizarlo debes estar también en el shell de UNIX y tipear:

```
> pico
```

TIP No.1.

Trata de trabajar con varias ventanas a la vez, para que no tengas que entrar a emacs, salir, entrar a Hugs, salir de Hugs.... es preferible tener un Hugs abierto y un editor de texto abierto, simplemente salvas, y luego lo cargas a ver que tal te quedo la gracia ☺

TIP No.2.

Si piensas tener Hugs en Casa, te recomiendo que trabajes con WinHugs y no con la versión para DOS. Así podrás poner en practica el TIP No.1. Además que el Hugs 1.4 para DOS trae consigo muchos Bugs a la hora de trabajar INPUT / OUTPUT.

HUGS

Es el interprete que vas a utilizar para correr tus SCRIPTS, trabaja con archivos de extensión hs. Para entrar a Hugs desde UNIX:

```
> hugs
```

Los comandos en el shell de Hugs vienen precedidos por ":" ejemplo si quieres salir de Hugs debes tipear:

```
Prelude> :quit
```

ó

```
Prelude> :q
```

Si quieres saber más sobre los comandos de Hugs tipea:

```
Prelude> :?
```

TIP No.3

Para que no tengas que esperar una eternidad cada vez que quieras correr tu programa después de haberlo alterado, no lo cargues por medio de

Prelude> :l programa.hs

Si ya habías hecho eso ahora lo único que tienes que hacer es correrlo cada vez que lo edites y lo salves:

Prelude> :r

Y si estas en WinHugs dale click al icono del muñequito corriendo.

TIP No.4.

Para cargar los archivos no hace falta copiar la extensión del mismo, con tan solo copiar el nombre basta. Eso sí, a la hora de editarlos sea de donde sea debes llamar a tu archivo por nombre y apellido.

Ejemplo:

```
Prelude>:e tentobin.hs
```

ó

```
>emacs tentobin.hs
```

ó

```
Prelude>:l tentobin
```

COSAS QUE EL PROFESOR NUNCA QUISO DECIRNOS POR QUE DEBIAMOS APRENDERLAS FRENTE AL COMPUTADOR... (YO TE AHORRO EL TRABAJO)

Primero es lo primero...

Si estas acostumbrado a programar en forma secuencial, olvídate del Tango... Esto es totalmente distinto, pero se te va a hacer muy fácil ya que es un *lenguaje funcional* . Esto quiere decir que todo lo que vayas a programar no vendrá dado por una serie de pasos, sino por una serie de funciones, que generalmente son funciones matemáticas, lo que hace la cosa más fácil, pero a la vez más abstracto.

Haskell es un lenguaje puramente funcional, y esto trae ventajas. Los Computistas siempre han estudiado algoritmos de ordenamiento. Por ejemplo el algoritmo de ordenamiento "Quick Sort", programado en un lenguaje secuencial toma hasta 4 paginas, en Haskell toma 4 ó 5 líneas. Haskell es utilizado para crear animaciones en 3D debido a la complejidad matemática que conlleva crear movimientos en 3D. Es utilizado para demostrar principios y teoremas matemáticos y tiene 2 grandes ventajas sobre otros lenguajes.

- Listas Infinitas.

Los otros lenguajes utilizan arreglos, que son como listas pero con un número que debe ser especificado de elementos, las listas en Haskell contienen elementos "infinitos"

- Polimorfismo.

Vas a escuchar al profesor hablar de esto pero nunca lo vas a entender hasta que veas un ejemplo. Las funciones pueden ser definidas para trabajar con distintos tipos de elementos.

MODULOS

Cada programa o script que tu creas va a ser un módulo, y un conjunto de módulos formarán un proyecto. En Hugs existen librerías llenas de módulos muy útiles que puedes utilizar para resolver tus problemas con mayor facilidad.

Una vez que comiences a escribir tu script lo primero que debes hacer es definir el nombre de ese módulo. Supongamos que vas a crear un programa que quieras llamar "Test1", debes hacer lo siguiente (en tu editor de texto):

```
module Test1 where

...
...
```

(La primera letra del nombre del módulo debe ir en mayúscula)
y copias tu programa, lo salvas, lo cargas en Hugs y ahora en vez de aparecer

```
Prelude>
```

Aparecerá esto:

```
Test1>
```

La ventaja de los módulos es que pueden importarse, y puedes cargar muchos programas a la vez. Por ejemplo el archivo a continuación es el archivo principal de mi proyecto para el semestre pasado.

```
-- Modulo principal France98

module TIPEAfrance98  where
import AnsiScreen
import Welcome
import Prob1
import Prob2
import Prob3


-- Algunas funciones de pantalla
clear :: IO ()
clear = putStr cls
limpiar = clear
```

Los guiones "- - " significan que a continuación viene un comentario para que otro programador pueda entender que fue lo que se hizo en el script.

Aquí se ha utilizado un comando llamado *import* que hace que al cargar el módulo TIPEAfrance98, también se carguen Welcome.hs , Prob1.hs, Prob2.hs , Prob3.hs.

IMPORTANTE: Cuando vayas a utilizar import debes copiar la primera letra del archivo en mayúsculas, cuando la maquina los va a cargar los busca con la primera letra en mayúscula, por eso recomiendo...

TIP No.5.

Siempre salva los archivos.hs con el mismo nombre que le colocaste al modulo, y siempre coloca la primera letra en mayúscula, así sea para salvarlos o para nombrar el modulo.

Por ejemplo el modulo Test1 debes salvarlo como: *Test1.hs*
Y definir el modulo:

```
module Test1   where
```
Bla bla bla

Imprimir mensajes en pantalla. (Lo que aprendimos por nosotros mismos)

Lo que te estoy enseñando hasta ahora no se debe parecer en lo absoluto a lo que esta enseñando el profesor en estos primeros dias de clase, pero yo creo que esto es lo primero que uno debe aprender. Si alguna vez has trabajado con Qbasic, Pascal o algo parecido, recordaras que para imprimir mensajes en la pantalla se hacia lo siguiente. (El ejemplo típico de Hola Mundo)

CLS

PRINT “Hola Mundo”

END

En Haskell esta es la única parte que se hace un poco mas larga. Haskell necesita saber que vas a utilizar mensajes en pantalla y para eso es recomendable que importes un modulo llamado AnsiScreen, que sirve para dar vida a comandos como CLS (Clear Screen), at (x, y) (que posiciona una serie de caracteres en la pantalla dependiendo de las coordenadas que le des), highlight, etc.

Para trabajar con mensajes en pantalla hay que utilizar el *tipo* IO ()
Que significa INPUT/OUTPUT.

Seguramente ya te enseñaron que son caracteres y que son Strings, pero existen unos caracteres especiales, que hagas lo que hagas siguen apareciendo y no cumplen la función que tu quieres, deben ser estos...

\n Cambia de línea.

\t tabulación.
\’ Escribe apóstrofes
\” Escribe comillas

Lo mas seguro es que hagas esto y te aparezca esto...

```
Prelude> “Aquí viene una linea\nnueva”
Aquí viene una linea\nnueva
```

Y ninguno de los otros funcione, y tu dirás que el profesor y los preparadores están locos... esto es porque debes utilizar el siguiente comando...

putStr

```
Prelude> putStr  "Aquí viene una linea\n nueva"
Aquí viene una línea
Nueva
```

Con el uso de putStr puedes hacer una pantalla de bienvenida, programas interactivos como tests, cuentos y cosas un poco más amenas.

Para utilizar cls, highlight, at, y otros debes haber cargado AnsiScreen (que lo trae Hugs), o haberlo importado en el modulo que creaste.

CLS

Que sirve para limpiar la pantalla. Se usa de la siguiente manera.
En el shell de Hugs:

```
AnsiScreen> putStr cls
```

Y enseguida se borra la pantalla.

En un modulo...

```
    module Borrador where
    import AnsiScreen

-- Aquí viene la función que borra la  pantalla
borraPantalla = putStr cls
```

Como veras aquí la cuestión no tiene secuencia, simplemente defines las funciones y después las utilizas a tu gusto. Para hacer que funcione ese programa lo que

debes hacer es salvarlo como Borrador.hs, lo cargas y una vez que el prompt de Hugs diga "Borrador>" tipeas "borraPantalla", y la pantalla se borra.

getLine

getLine es también una función del *tipo* IO String. Y sirve para recibir Strings, y luego operar con ellos. No trabaja con Int (Enteros) ni con ningún tipo de números.

Ejemplo (Una función que pide el nombre y luego te dice tu nombre):

```
nombrador :: IO ()
nombrador =   putStr "¿Cuál es tu nombre?" >>
              getLine >>= \variable ->
              putStr ("Tu nombre es " ++ variable)
```

Fíjate en el uso de ">>" en cierta forma aquí si existe una secuencia. ">>" Significa que viene otro comando IO, y ">>=" permite entrar una variable.
Los símbolos ">>" & ">>=" se llaman combinadores.

En cuanto al "\" antes de "variable", eso es algo llamado *lambda calculo*, que permite que una variable se defina para ser utilizada más tarde.

El cálculo lambda permite utilizar una función dentro de otra sin darle nombre.

función x = y es lo mismo que decir
función = \x -> y

COMPOSICIÓN DE FUNCIONES. *(Function composition)*

Esto es igual que en matemáticas. Para componer una función cualquiera *f* (x) con *g(x)* uno lo denota así: "f(g(x)) " y lo leemos así: "f de g de x"

En Haskell lo podemos hacer de muchas formas, para empezar a familiarizarte puedes aprender la forma que se parece mas a las matemáticas:

```
f (g x)
```
"f de g de x"

En Haskell eso es lo mismo que decir

```
(f . g) x
```
"g en f de x"

Primero se aplica g y lo que resulte se aplica en f.

COMPOSICION HACIA DELANTE. *(Forward composition)*

Esto puede resultar un poco confuso, porque ahora la composición se lee alrevez.

Lo que antes hacíamos así:

```
f (g x) = (f . g) x
```

Ahora lo hacemos así:

```
(g >.> f) x
```

Ahora se toman los argumentos en el orden contrario al uso de ‘ . ‘, primero se aplica g y lo que resulte se aplica en f. Es lo mismo pero se escribe alrevez. En resumen tenemos que:

```
f (g x) = (f . g) x  = (g >.> f) x
```

DE TODOS MODOS *Forward Composition* NO SIRVE EN HUGS.

OPERATIVIDAD EN HASKELL.

En Haskell llamamos operadores funciones que utilizan dos variables aunque en realidad Haskell solo trabaja con funciones de una variable y después la compone.

Como ejemplo de operadores tenemos ‘+’,’-‘,’/’,’*’, ‘div’, ‘mod’ , ‘++’,’>’,’:’, etc.

La operatividad se lleva a cabo de esta manera
(‘a’ y ‘b’ son variables. ‘op’ es el operador)

(a op) b = a op b
(op a) b = b op a

En pocas palabras Haskell coloca los elementos alrededor del operador, ejemplo:

(4+) 2 = 4 + 2
(+2) 4 = 4 + 2

TIPOS (*Types*)**.**

Los ejemplos anteriores han utilizado caracteres, texto, números, también se pueden utilizar booleanos, etc. Para que las funciones sirvan “debemos” especificar bien, que tipos utilizan. Si vamos a crear una función que sume las edades de distintas personas, debemos especificar que va a trabajar recibiendo enteros y que como resultado va a dar un entero. Ejemplo:

SumaEdades :: Int -> Int

Los Tipos más usados son:

Int (Enteros)
Float (Flotantes ó Decimales)
Char (Caracteres)
String (Listas de Caracteres)
Bool (Booleanos, True, False)
Monads (Monádicos)

TIP No.6.

Cuando estés definiendo una función en Haskell, no hace falta especificar con que tipo trabaja, Haskell lo asumirá inmediatamente. Un error definiendo los tipos que vas a utilizar, puede tenerte horas sentado frente al computador intentando descifrar cual es el error que hace que no corra. **Copia tu programa, si corre, copias `':t función'` en el shell de Hugs y el te dice el tipo que corresponde a la función, quizás corras con la suerte de que tu función sea polimórfica** (que trabaje con cualquier tipo). **Una vez que sepas el tipo lo copias encima de la función o al comienzo del documento.** (Hay practicas en las que dada una función debes descifrar que tipos utiliza)

BOOLEANOS

Los operadores booleanos son aquellos que tanto utilizaste el semestre pasado en *Lógica Computacional* para las demostraciones, y que seguirás usando hasta que te gradúes.

En Haskell se denotan de la siguiente manera:

‘&&’ = ‘y’ ó ‘ ^ ’
‘| |’ = ‘ó’ ó ‘ v ‘
‘not’ = ‘no’ o‘¬’

odd impar
even par

Otros Operadores:

```
>    mayor que / greater than
<    menor que / less than
>=   mayor igual que / greater or equal than
<=   menor igual que / less or equal than
```

```
==   igual (se utiliza para variables, generalmente dentro
           de los guardianes)
/=   diferente que (evita mucho trabajo)

|    guardián (se utiliza para definir condiciones)
```

Ejemplo de guardián:

Función que determina quien es mayor:

```
mayor :: Ord a => a -> a -> a
mayor x y
 |x >= y  = x
 |otherwise    = y
```

(Nota: Para definir el tipo de esta función apliqué el tip No. 6, yo ni siquiera sé que significa Ord ;))

Todo lo referente a Listas, Arboles, Trabajar con Enteros y todas esas cosas es mejor que las aprendas por ti mismo y con la ayuda de el profesor.

De todos modos aquí tienes una guía con una selección de ejercicios resueltos, de parciales y prácticas.

Ejercicios.

1. Dar una definición de la función nAnd

```
nAnd :: Bool -> Bool -> Bool
```

que da como resultado True excepto cuando los argumentos dados son ambos True

```
-- Función que da como resultado True excepto cuando los
-- argumentos dados son ambos True
-- Angel 'Kangaroo Gubatron' León.

module NotAnd__nAnd where

nAnd :: Bool -> Bool -> Bool
nAnd n m = not (n && m)
```

2. Considere la siguiente definición de la función ack, de N x N en N

```
ack (0,X) =    1    Si x=0
               2    Si x=1
               X+2  Si x>1

ack (y+1, 0) = 1
ack (y+1, x+1) = ack (y, ack (y+1,x))
```

a) De una definición en Haskell para la función ack.
b) Muestre la evaluación de la aplicación ack 3 2

```
module Ack__ack__ack2 where

ack :: (Integral a, Integral b) => a -> b -> b
ack2 :: (Integral a, Integral b) => a -> b -> b

-- Primera solución.
ack y x
  |(y==0) && (x==0) =1
  |(y==0) && (x==1) =2
  |(y==0) && (x>1)  =x+2
ack (y+1) 0 = 1
ack (y+1) (x+1) = ack (y) (ack (y+1) (x))


-- Segunda Solución
ack2 y x
  |(y==0) && (x==0) =1
  |(y==0) && (x==1) =2
  |(y==0) && (x>1)  =x+2
ack2 (y) 0 = 1
ack2 (y) (x) = ack (y-1) (ack (y) (x-1))
```

3.- *Definición de la función Fibonacci*
sabiendo que es una función definida
matemáticamente como:
fib (0)=0
fib (1)=1
fib (n)=fib (n-1)+fib (n-2)

```
--------------------Angel León 14/04/98-------------------
---------------------3:22 p.m.-----------------------------
fib :: Int -> Int
fib n = fib2 n 0 1
fib2:: Int -> Int -> Int -> Int
fib2 n a b | n == 0    = a
           | n > 0     = fib2 (n-1) b (a+b)
------------------------------------------------------------
```

* LOS PRIMEROS 10 NUMEROS FIBONACCI SON :

* 1,1,2,3,5,8,13,21,34,55

---

*4.- Defina **suma :: N -> N -> N** suponiendo que la máquina sólo puede sumar de uno en uno. (obviamente, suma m n = m+n, no es la solución a este problema)*

```
-- Función que suma de uno en uno
-- Luis 'G-lipe' Sintjago
-- El tipo se sudó las n...

module Suma_de_Uno_en_Uno__suma
where

suma :: Int -> Int -> Int

suma a 0 = a
suma 0 b = b
suma a 1 = a + 1
suma 1 b = b + 1
suma a b = suma (a-1) (b+1)
```

*5.- Defina **producto :: N -> N -> N,** suponiendo que la máquina sólo sabe sumar.*

```
-- Definición de producto, suponiendo que la maquina solo
--- sabe  sumar.
-- Luis 'G-lipe' SintJago  14/04/98 4:02 p.m.
module Producto__producto where

producto :: Int -> Int -> Int
producto n m
  |n==0      =0
  |m==0      =0
  |otherwise        = m + (producto (n-1) m)
```

*6.- Defina **potencia :: N -> N -> N,** suponiendo que la máquina sólo sabe multiplicar.*

```
-- Función Potencia suponiendo que la máquina sólo sabe
-- multiplicar.
-- Angel 'Kangaroo Gubatron' León.
-- 24/08/98 10:50 AM

module Potencia__potencia where

potencia :: Integer -> Integer -> Integer
```

```
potencia n m
 | m == 0 = 1
 | m == 1 = n
 |otherwise = n * potencia n (m-1)
```

7.- Defina una función que transforme números Decimales a Binarios ***tentobin,*** *y otra que haga lo mismo pero al revés* ***bintoten.***

```
-- Anónimo, todo el mundo se lo copió.
-- Si lo analizas con detenimiento en un papel te darás
-- Cuenta de lo simple que es... parece brujería.
-- Los Rumores dicen que fueron creados por
-- Maritza Martínez

tentobin :: Int -> Int
tentobin y
 |y<2 =y
 |otherwise = mod y 2 + tentobin (div y 2) * 10

-- Hugs puede transformar hasta el 1023, ya que no calcula
-- números mayores de 10 cifras



bintoten :: Int -> Int
bintoten y
 |y<2 =y
 |otherwise = mod y 10 + bintoten (div y 10) * 2
-- Hugs puede transformar hasta el 1111111111, ya que
-- no soporta mas cifras
```

8.- Definir una función ***charToNum :: Char -> Int,*** *que convierta un dígito como '9' a su valor numérico. (El valor de los caracteres no correspondientes a números debe ser 0)*

```
------- ChartoNum ----------------------------
-- Convierte Números dados en forma de Char --
-- a Números Enteros                        --
-- Created by Dioni 'DiUNIX' Escalante      --
-----------------------------------------------

module CharToNum
where

charToNum :: Char -> Int
charToNum ch
 | (ch>='0') && (ch <='9') = fromEnum ch - 48
 |otherwise = 0
-- También hubiese servido utilizando
-- la función 'ord' en vez de 'fromEnum'
```

9.- Definir una función ***romanDigit :: Char -> String,*** *que tome un número (en forma de* ***Char****) y lo lleve a su representación en números Romanos.*
NOTA: Recuerda que la longitud máxima de un ***Char*** *es de 1(uno). Eje:* romanDigit '7' = 'VII'

```
-----------------------------------------------
-- romanDigit que toma un numero en forma de --
-- Char y lo lleve a su representación en    --
-- números Romanos.                          --
-----------------------------------------------
-- Angel Loen. 10:06 a.m. 15/04/98           --
-- Terminado a las 12:13 p.m.                --
-----------------------------------------------
module Roman_Digit__romanDigit where
romanDigit :: Char -> String
chartoNum :: Char -> Int
roman :: Int -> String

romanDigit n = roman (chartoNum n)



-----------------------------------------------
---- se utiliza chartoNum para transformar ----
-- el caracter , a enteros y luego utilizar ---
-- roman para dar el resultado final        ---
-----------------------------------------------

chartoNum ch
 | (ch>='0') && (ch <='9') = fromEnum ch - 48
 |otherwise = 0

----------------------------------------
----  roman transforma  enteros      --
--              a Romanos             --
----------------------------------------

roman n
  |n== 0 = "O"
  |n== 1 = "I"
  |(n>1)&& (n<4) = "I" ++ roman (n-1)
  | (n==5)       = "V"
  | (n==4)       = "I" ++roman (n+1)
  | (n>5)&& (n<9)= "V"++ roman (n-5)
  |n==9          = roman (n-8)++ "X"
   |otherwise = "Solo se puede hasta el nueve, además eso
que escribiste no es un dígito"
```

Si lo entendiste te felicito. Lo que hice fue hacer una función roman que transforma enteros a Romanos, pero el enunciado dice que deben ser números en forma de caracteres, así que le aplico charToNum al carácter, eso me da un entero, y al entero le aplico roman, todo esto es romanDigit. ***"romanDigit n = roman (charToNum n)"****, simple!*

De la siguiente forma lo hizo Mauricio 'Mel' Echezuría.

```
roman:: Char -> String
roman n
 | m == 9               = "IX"
 | m==1                 = "I"
 | m==5                   = "V"
 | m==4                   = "IV"
 | m == 0                 = "O"
 | (m>0) && (m < 4)      = "I" ++ roman (chr (m+47))
 | (m>5) &&  (m <= 8)      = "V" ++ roman h
           where           m=(ord n)-48
                           h=chr (m+43)
```

La ventaja de mi programa es que tiene una función que transforma Enteros a Romanos.

FUNCIONES UTILES

```
ord       Dado un carácter lo transforma en ASCII

reverse (a:x)       Voltea la Lista.

cycle [a]      Repite la lista '[a]' infinitas veces.

unlines   Transforma una [String] a líneas una abajo
          de la otra.

lines     Transforma String -> [String]
```

capitalize Transforma un carácter en minúscula a mayúscula (Pag 204 del Libro

> fst (a,b,..,z) Muestra el primer elemento de la tupla a

> snd (a,b,..,z) Muestra el segundo elemento de la tupla b

Se pueden definir funciones que muestren distintos elementos de las tuplas.

10.- Defina una función que dada una lista de String, calcule el String de menor longitud, distinto del String nulo. (QUIZ)

```
-- Función que dada una lista de String, calcula el
-- String de menor longitud, distinto del String nulo
-- Carlos 'Pinky' Carbonell.

module SmallString where

smallString :: [String] -> String
smallString [] = []
smallString (a:x) = foldr minus "" (a:x)

minus :: String -> String -> String

minus a b
 |(length a < length b) && (a /= "") = a
 |(length b < length a) && (b /= "") = b
 |(length b < length a) && (b == "") = a
 |otherwise = b
```

11.- Defina una función que reciba una lista de enteros, y duplique los enteros pares (si el entero es impar, éste queda igual) (QUIZ)

```
-- evenDuplex duplica solo los números pares
```

```
-- que están en la lista
-- Angel 'Kangaroo' León. Lunes 24/08/98 1:09 PM

module EvenDuplex where

evenDuplex :: (Functor a, Integral b) => a b -> a b
evenDoubler :: Integral a => a -> a


evenDuplex l =  map evenDoubler l

evenDoubler n
 | (even n == True) = 2*n
 | otherwise   = n
```

*12.- Con **List comprehension** defina una función que cuenta los números negativos de una lista. (PARCIAL)*

```
-- Función que cuenta los números negativos de una
-- lista.
-- Angel 'Kangaroo' León. Primer Parcial 30/04/98

module Negative_Counter where

negCounter :: (Ord a, Num a) => [a] -> Int
listMorpher :: (Ord b, Num b, MonadZero a) => a b -> a b


negCounter n = length (listMorpher n)

listMorpher n = [x| x <- n, x < 0]

-- Los tipos no los definí yo, ojo, Recuerda TIP 6, pag. 14
```

*13.- Dados 2 vectores **u** y **v**, crear:*
a) Un programa que sume U + V.
b) *El vector producto punto U • V.*
c) La longitud de un vector cualquiera.
d) La Longitud de su diferencia.
(PARCIAL)

```
-- Vectors
-- Angel 'Kangaroo' León

module Vectors where
addComponents :: (Num c, Num b, Num a) => (a,b,c) ->
(a,b,c) -> (a,b,c)
dotVector :: Num a => (a,a,a) -> (a,a,a) -> a
absVector :: Floating a => (a,a,a) -> a
absDifference :: Floating a => (a,a,a) -> (a,a,a) -> a

addComponents (ux,uy,uz) (vx,vy,vz) =
               (ux + vx, uy + vy, uz+vz)

dotVector (ux,uy,uz) (vx,vy,vz) = (ux*vx)+(uy*vy)+(uz*vz)

absVector (ux, uy, uz) = sqrt ((ux^2)+(uy^2)+(uz^2))

absDifference (ux,uy,uz)(vx,vy,vz) =
               absVector ((ux-vx),(uy-vy),(uz-vz))

-- No importa si terminan de definir una función en la
-- Línea de abajo, como 'addComponents' Hugs la reconoce.
```

13.- Función que remueva al primer elemento de una Lista I que no tiene la propiedad p. (PARCIAL)

filterFirst :: (t->Bool) -> [t] -> [t]

¿Qué hace su función si le entra como argumento una lista cuyos elementos cumplen la propiedad **p** ?

FUNCIÓN UNSHOW

Por Angel 'Kangaroo Gubatron' León.

Existe una función en Haskell llamada "**show**", esta función transforma Int a Strings, ejemplo:

```
Prelude> show 100
"100"
Prelude> show (22+4)
"26"
```

Mientras trabajaba en mi proyecto tuve la necesidad de una función que hiciera lo contrario, así que tuve que crear **unshow**. Espero que te sirva de algo.

```
-- Función Unshow creada por
-- Kangaroo Gubatron.
```

```
-- Dado un String de Enteros, lo convierte
-- a Enteros
-- Ejemplo
-- Unshow> unshow "1998"
-- 1998
-- Unshow> unshow "1998" + unshow "1998"
-- 3996
module Unshow where

import CharToNum

unshow :: [Char] -> Int
transformador :: Functor a => a Char -> a Int
transformer :: Num a => [a] -> a


unshow n = ((transformer. transformador) n)

transformador n = map charToNum n

transformer [] = 0
transformer (lista) = ((head (lista)) * (10 ^ length (tail
(lista))))+(transformer (tail (lista)))
```

TIP No.7

Es recomendable que al escribir tus programas, coloques comentarios acerca de lo que intentas hacer.

Yo conozco al menos 2 formas de insertar comentarios en tus Scripts.

1. La que enseñan en el Libro.

```
-- Esto es un comentario.
-- Y es un fastidio tener que copiar guiones en cada línea
-- Se utiliza principalmente para comentarios pequeños
-- ó  los puedes introducir en una línea que te esté dando
-- problemas (para que deje de funcionar y sea como un
-- comentario más) y no borrarla en caso de que sea otro el
-- error.
```

2.- La que aprendes por ahí...

```
{-
Esta es otra forma de colocar comentarios de una manera más
libre y te evitas estar tipeando  los guioncitos en cada
línea. Esta forma de comentario es mas que todo para decir
que es lo que hace el programa, introducirlo, instrucciones
etc. Fíjate en la sintaxis, debes comenzar con la LLAVE-
GUION, y a partir de la línea siguiente, una vez que
termines el comentario, en la siguiente línea, cierras con
GUION-LLAVE. Así...
```

```
-}
```

De todos modos los ejercicios a continuación tendrán este tipo de comentarios.

Más ejercicios...

14.- Una función que:
a) Chequee si un dígito aparece en un número.
b) Calcule cuantas veces aparece.
c) Me diga el mayor de todos
d) Me diga el menor de todos
e) Me diga el número al revés
f) Si es palíndrome o no.

```
module Digits where
{-
 digitOK. Chequea si un dígito aparece en un número
 se utiliza de la siguiente manera, digitOK x y, busca
 el dígito 'y' en el número 'x'
-}
digitOK :: Integral a => a -> a -> Bool
digitOK 0 0 = True
digitOK 0 _ = False
digitOK n d
 |(mod n 10) == d = True
 |otherwise = digitOK (div n 10) d
-- Calcula cuantas veces aparece
counter :: (Num b, Integral a) => a -> a -> b
counter 1 1 = 1
counter 0 0 = 1
counter 0 _ = 0
counter x y
 |x == y = 1
 |y == mod x 10 = 1 + counter (div x 10) y
 |otherwise = counter (div x 10) y

-- Calcula el mayor dígito en un número

digitLarge :: Int -> Int
digitLarge n
 |(n >= 0) && (n <= 99) = max (mod n 10) (div n 10)
 |otherwise = max (mod n 10) (digitLarge (div n 10))

-- Calcula el menor Dígito
digitSmaller :: Int -> Int
digitSmaller n
 |(n >= 0) && (n <= 99) = min (mod n 10) (div n 10)
 |otherwise = min (mod n 10) (digitSmaller (div n 10))

{-
Estas dos funciones fueron creadas por:
 Luis Felipe 'El G-lipe' SintJago.
 Unix Lab.
-}
-- Escribe el numero al revés.

digitBack :: Int -> Int

digitBack n
```

```
 | (n>=0)&& (n<=9)  = n
 | (n>=10)&& (n<=99)=10* (mod n 10) + (div n 10)
 | (n>=100)&& (n<=999)=100 * (mod n 10) + digitBack (div n 10)
 | (n>=1000)&& (n<=9999)=1000 * (mod n 10) + digitBack (div n 10)
 | (n>=10000)&& (n<=99999)=10000 * (mod n 10)+digitBack (div n 10)
 | (n>=100000)&& (n<=999999)=100000 * (mod n 10)+digitBack (div n 10)
 | (n>=1000000)&& (n<=999999)=
                            1000000 * (mod n 10)+digitBack (div n 10)


-- Función que sabe si es palíndrome

palindrome :: Int -> String
palindrome n
 |digitBack n ==n     ="Si es palindrome"
 |otherwise   = "Falso"
```

15.- Crea un programa que concatene dos listas.

```
{-
Angel León   29/04/98   01:03 PM
Voltea la primera lista, y la concatena con la segunda
sin utilizar reverse
Salió de leche, buscando un programa que concatene dos
listas.
Angel León
-}
module AppendBack where

appendBack :: [a] -> [a] -> [a]
appendBack [ ] [ ]     = [ ]
appendBack [ ] (b:y) = (b:y)
appendBack (a:y) [ ] = (a:y)
appendBack (a:x) (b:y) = appendBack  x (a:b:y)

{-
Ahora que entiendo que fue lo que hice, les explicaré:
toma la cabeza de la primera lista, la concatena con
la otra lista, y luego aplica appendBack a la cola,
con la nueva lista. :) NOTA: Resuelve tú el ejercicio 15
-}
```

16. Crea un programa que transforme una palabra en caracteres ASCII, y otro que sume todos los caracteres.

(Luego prueba que sucede si pruebas con Bill Gates III, pruebas con "BILLGATES", y sumas el 3, debe dar 666, prueba también con "WINDOWS96")

```
{-
     Función que transforma una lista de Caracteres
en ASCCI, y que luego los suma. <ascci>
-}

module ASCCI__transfor_ascci where

transfor :: String -> [Int]
transfor [] = [ord '\NUL']
transfor (n:m) = (ord n:transfor m)

sumList :: [Int] -> Int
sumList [] = 0
sumList (a:x) = a + sumList x

ascci :: [Char] -> Int
ascci s = sumList (transfor s)

-- Si utilizas espacios, se sumaran 32 por cada espacio.
```

Ejemplos.

```
-------------------------------------------------------------------------
-
-- ATO. Tells wich number appears the most and how many times it
--
-- does (It doesn't matter if it's the lowest).
--
--            Angel León 02:44 PM 13/04/1998              --
-------------------------------------------------------------------------
-
module Any_of_the_Three_Occur_ato where
coccur :: Int -> Int -> Int -> Int
eoccur :: Int -> Int -> Int -> Int
ato :: Int -> Int -> Int -> (Int,Int)
-------------------------------------------------------------------------
-
-- commonOccur(coccur) calcula cual es el que aparece con mayor
--
-- frecuencia
--
-------------------------------------------------------------------------

coccur n m p
 |(n==m)    =n
 |(n==p)    =n
 |(m==p)    =m
 |otherwise     =999999999999

-------------------------------------------------------------------------
-
--    eoccur calcula el numero de ocurrencias de números iguales    --
-------------------------------------------------------------------------
```

```
eoccur n m p
 |(n==m) && (n==p)      =3
 |(n==m)           =2
 |(n==p)           =2
 |(m==p)           =2
 |(n/=m) && (m/=p) && (m/=p)  =0

-------------------------------------------------------------------------
-
--  (anyoftheThreeOccur)             ATO                   --
-------------------------------------------------------------------------
-

ato n m p = (coccur n m p,eoccur n m p)


-------------------------------------------------------------------
-- Angel 'Kangaroo Gubatron' León ©, Saturday 01:53 PM 19/04/1998   --
-- lowercase ASCCI  97-122
-- uppercase ASCCI 65-90

-- Función que reconoce que 'case' estas utilizando.




module Cases_cases where
cases :: Char -> String
cases n
 |(fromEnum n>= 97)&&(fromEnum n<=122)     ="That was a lowercase
Character"
 |(fromEnum n>=65)&&(fromEnum n<=90)        ="That was an uppercase
Character"
 |otherwise                    ="That was either a digit or a symbol"



{-
 Tarea
 Función que dada un entero "n" y una lista de tipo "t",
 La transforma en una lista cuyos elementos aparecen n veces
 Para correr este programa tipea 'help'
-}
module Cloner where
import AnsiScreen


-- help

help = información

pause = putStr (at (20,20) "Presiona [ENTER]...") >> getLine >>= \n ->
putStr cls
información =     putStr cls >>
                  putStr (at (10,10) "Este programa dado    un
entero  positivo k") >>
            putStr (at (10,11) "transforma una secuencia [x1,x2] en una
nueva") >>
            putStr (at (10,12) "secuencia [x1...x1k,x2.....x2k]
repitiendo   cada") >>
            putStr (at (10,13) "elemento K veces.") >> pause >>
```

```
            putStr (at (10,10) "Funciona así...") >>
            putStr (at (10,12  "Cloner> cloner 2 [\"DOS\"]")>>
            putStr (at (10,13) "[\"DOS\", \"DOS\"] ") >>
            putStr (at (10,14) "(esta función trabaja con cualquier
tipo)")



-- Esta función repite 1 elemento de tipo t
-- n veces (Servirá como motor para la definición de
-- la función principal)

multiplier :: Num a => a -> b -> [b]
multiplier 0 n = []
multiplier 1 n = [n]
multiplier x n = [n] ++ (multiplier (x -1) (n) )

-- Función principal (Versión recursiva)
cloner :: Num a => a -> [b] -> [b]
cloner 0 (a:x) = [ ]
cloner 1 (a:x) = (a:x)
cloner _ [ ] = [ ]
cloner n (a:x) = (multiplier n a) ++ (cloner n x)



-- Cuenta los negativos de una lista
-- Practica 3

module Cuenta_Negativos_contN where
contN :: [Int] -> Int
contN a
 |a==[]  =0
 |head a<0 = 1 + contN (tail a)
 |otherwise = contN (tail a)

-----------------------------------


-- Dada una lista y un numero
-- me calcula el elemento de la lista
-- cuya posición corresponde con dicho numero
-- Siendo la primera posición el 0
-- Angel 'Kangaroo Gubatron' León

module Element where

element  :: [Float] -> Int -> Float
element [] n = 0
element (x:y)  0= x
element (x:y) 1 = head y
element (x:y) n
 |length (x:y) >= n  = element (drop (n-1) (x:y)) 1
 |length (x:y) < n    = 0

-----------------------------------

-- Definiciones de la función factorial.
-- Cualquiera...

module Función_Factorial__factorial__fac where

factorial :: Int -> Int
```

```
factorial n
 |n==0      =1
 |otherwise = n* factorial (n-1)

fac :: Int -> Int
fac 0 = 1
fac n = n * factorial (n-1)

--------------------------------------
-- Lista 4
-- Una función que calcule el promedio de una Lista
-- Angel 'Kangaroo Gubatron' León.

module Foldaverage where

sumalista :: [Int] -> Int
sumalista [] = 0
sumalista n = (head n) + (sumalista (tail n))

foldaverage :: [Int] -> Float
foldaverage n = div (sumalista n) (length n)

---------------------------------------

-- guía 4
-- Una función que eleve al cuadrado todos los elementos de una lista

foldsquare :: [Int] -> [Int]
foldsquare [] = []
foldsquare (x:xs) = (x*x:foldsquare xs)

-------------------------------------

-- index1, me dice en que posición esta el elemento dado. Si no esta
-- devuelve (-1)
-- Angel León   29/04/98 01:00 PM

module Index1__index1 where


index1 :: (Num b, Eq a) => [a] -> a -> b
member :: Eq a => [a] -> a -> Bool

-- Estos tipos fueron definidos por Hugs después de haber creado
-- las funciones. Yo nunca hubiese llegado a tal definición.

index1 [ ] n = -1
index1 (a:x) n
  | (member (a:x) n == False ) = -1
  | n==a    =1
  | n /= a    = 1 + index1 x n

---------------------------- TIP ------------------------------
-- Algunas veces tu programa puede estar perfecto y no corre.
-- Puede ser que nombraste la función con un nombre que ya existe
-- En este caso no quería correr y tuve que nombrar la función
-- 'index1' ya que 'index' existe.
---------------------------------------------------------------

member [ ] n =False
member (a:x) n
 |n==a    = True
 |n /= a    = member x n
```

```
-------------------------------------

{-
 Función que transforma los caracteres que están en
 mayúscula en un String a minúscula. Es muy útil para programas
 donde el usuario tiene que introducir Strings y tu quieres
 que entren en minúsculas. El usuario los introduce de cualquier
 manera y tu programa los lleva a minúscula para luego trabajar
 con ellos.
-}

-- Angel 'Kangaroo Gubatron' León

module Lower__lower where

lower :: String -> String
lower []=[]
lower (x:xs)
 | x >= 'A' && x <= 'Z' = (chr ( (ord x)+ 32): lower xs)
 | otherwise  = x:lower xs


-- otra forma, solo que da un error...

lower2 :: [Char] -> [Char]
lower2 (a:x)
  | (ord  a >= 65) && (ord a <= 90)= ( chr (ord a + 32) : lower2 x)
  | otherwise                                          = (a:lower2 x)


------------------------------


-------------------------------------------------
-- Luis 'G-lipe' SintJago                       --
-------------------------------------------------
-----Función que calcula el número más grande-----
-------------------------------------------------

module MaxDigit where

maxdigit :: Int -> Int
maxdigit  n
   |(n>=0)&&(n<10)     = n
   |otherwise          = max (mod n 10)  (maxdigit (div n 10))

-- Para entender con mayor claridad este ejercicio
-- imagínate que al usar 'mod' con una división con 10,100,1000, etc.
-- mod es igual al número después del la coma. (En el resultado de la
-- división)
-- La función lo que hace es que va comparando que numero después
-- de la coma es mas grande, y el que es el mas grande de todos
-- viene siendo el dígito más grande del entero.

-- Angel 'Kangaroo Gubatron' León.
-------------------------------------
```

ESTE PROXIMO PROGRAMA ES MUY UTIL PARA MAT. Discretas.

```
-- Máximo Común Divisor
-- From Internet. Useful for those Matematicas Discretas exercises.

module MCD__mcd where
```

```
mcd :: Int -> Int -> Int

mcd a b
 |b==0      =a
 |mod a b > 0     = mcd b (mod a b)
 |mod a b == 0   = b
```

```
-------------------------------------------------------------
-- Member me dice si un elemento pertenece a la lista.
-- Angel León 28/04/98 08:00 PM

module Member_member where

member :: [Float] -> Float -> Bool

member [ ] n =False
member (a:x) n
 |n==a    = True
 |n /= a    = member x n
```

```
-------------------------------------------------------------------------
--
-- maxThreeOccur (mto), Tells who the largest number is, and how many
-- times it appears /Dice cual es el número mas grande y cuantas veces
-- aparece
-- Angel 'Kangaroo Gubatron' León 04:05 PM 13/04/1998
-------------------------------------------------------------------------
-
module MaxThreeOccur__mto where
moccur :: Int -> Int -> Int -> Int
mto :: Int -> Int -> Int -> (Int, Int)


-------------------------------------------------------------------------
--
--     maxOccur (moccur) calcula el numero de ocurrencias del mayor
---     numero                                                        --
-------------------------------------------------------------------------

moccur n m p
 |(n==m) && (n==p)           =3
 |(n==m) && max n p == n     =2
 |(n==p) && max n m == n        =2
 |(m==p) && max n m == m        =2
 |otherwise            =1

-------------------------------------------------------------------------
-- Máximas tres ocurrencias (mto)                                       -
-------------------------------------------------------------------------
mto n m p = (max n (max m p), moccur n m p )
```

```
-------------------------------------------------------------------------
-- Suck my Dick!                                                       --
-------------------------------------------------------------------------

-----------------------------------------------------------
-- Reverse1, pone una lista al revez sin utilizar reverse
-- Angel León , 29/04/98

module Reverse1 where

reverse1 :: [a] -> [a]
reverse1 [] = []
reverse1 (a:x) = reverse1 x  ++  [a]
```

QUICKSORT e INSERTION SORT. (Código en Haskell)

```
module Qsort_Isort where
-- Algoritmo de Quick Sort
-- de orden n.log(n)

qsort :: Ord a => [a] -> [a]
qsort [] = []
qsort [x] = [x]
qsort (x:xs) = (qsort [a | a <- xs, a < x]) ++ (x:qsort [a|a <- xs,
a>= x])

-- Algoritmo de Insertion Sort

isort :: Ord a => [a] -> [a]
isort [] = []
isort (x:xs) = insert x (isort xs)
      where
       insert x [] = [x]
       insert x (b:bs)
         | x<=b   = (x:b:bs)
         | otherwise = (b: insert x bs)
```

ABSTRACT DATA TYPES.

Ejemplos ☺

```
-- PROGRAMA QUE DESCRIBE COMO ES LA TEMPERATURA, EN CADA
-- TEMPORADA CLIMATICA INGLESA!
-- TOMADO DE HASKELL. THE CRAFT OF FUNCTIONAL PROGRAMMING

module Weather where

data Temp = Cold | Hot
            deriving (Show)


data Season = Spring | Summer |  Autumn | Winter
            deriving (Show)

weather :: Season -> Temp
weather Summer = Hot
weather _ = Cold
```

NOTA IMPORTANTE !

Nótese que después de definir 'Temp' debajo de cada uno de los constructores de datos se coloca:
'deriving (Show)', también se puede colocar:
'deriving (Show,Eq,Ord)' esto depende de el uso que le vayas a dar a esos constructores de datos.

En algunas versiones de HUGS es indispensable el uso de:
Deriving (Show, Eq, Ord), por otra parte cuando yo estaba estudiando Haskell, nunca tuve que usarlo, pero ahora que estoy haciendo la Guía (para Uds.) el Hugs que tengo en mi PC me da el siguiente error si no utilizo 'deriving'.

```
ERROR: Cannot find "show" function for:
*** expression : weather Summer
*** of type    : Temp
```

Asi que busqué en el libro, y vi un ejemplo en el que se utilizaba deriving (pag. 257 Binary Trees). Para que tengas una idea de lo que hace DERIVING primero querrás saber para que estamos programando con tipos de datos abstractos. La VENTAJA de *data XXX = Xxx | Xxxx ...* es que cuando estés corriendo el programa no tienes que trabajar con Strings, lo que significa que al fin podrás hacer un programa sin las malditas comillas, y tendrá una apariencia más aceptable.

Por ejemplo para correr ese programa lo único que tienes que hacer es lo siguiente:

```
Hugs session for:
C:\WINHUGS\lib\Prelude.hs
weather.hs
Weather> weather Spring
Cold
Weather> weather Summer
Hot
Weather> weather Autumn
Cold
Weather> weather Winter
Cold
Weather>
```

La única desventaja de esto es que tienes que escribir la primera letra en mayúscula.

"Derivando instancias de Clases (*Deriving instances of Classes*)

- `Eq`, una clase dando igualdad y desigualdad
- Ord, construida en `Eq`, dando y ordenando sobre cimientos de un tipo.
- `Enum`, permitiendo el tipo ser enumerado, y Asi dando `[n..m]` expresiones de estilo sobre el tipo,
- `Show`, permitiendo a los elementos del tipo ser cambiados a forma textual (El que usamos para el ejemplo de weather.hs) y
- `Read`, permitiendo a los items ser leídos de Strings"

(Tomado de La página 245 de"Haskell, The Craft of Functional Programming")

Uno de los mejores usos de los tipos de datos abstractos (Abstract Data Types) son los Arboles. Con Arboles se pueden resolver muchos problemas, sobre todo algoritmos de ordenamiento. Sigamos con los ejemplos...

```
-- Función poda

data Arbol a = Hoja | Rama a (Arbol a) (Arbol a)

-----------------------------------------------------------------------
takeHoja :: Arbol a -> [a]
```

```
-----------------------------------------------------------------------
takeHoja Hoja = []
takeHoja (Rama a Hoja Hoja) = [a]
takeHoja (Rama x iz de) = (takeHoja iz) ++ (takeHoja de)


-----------------------------------------------------------------------
poda' :: Arbol a -> Arbol a
-----------------------------------------------------------------------
poda' Hoja = Hoja
poda' (Rama x Hoja Hoja) = Hoja
poda' (Rama x iz de) = (Rama x (poda' iz) (poda' de))


-- Sigue en la próxima página...



-- Función Final

-----------------------------------------------------------------------
poda :: Arbol a -> (Arbol a,[a])
-----------------------------------------------------------------------
poda Hoja = (Hoja, [])
poda (Rama x iz de) = (poda' (Rama x iz de) , takeHoja (Rama x iz de))
```

Fíjese que al comienzo se define el tipo Arbol, de otro modo jamas serviría la función. Esta definición es muy importante a la hora de empezar el problema, ya que existen muchos tipos de Arboles y tenemos que ver que tipo es el que mas conviene.

```
module MapTree where

data Tree t = Leaf | Node t (Tree t) (Tree t)

mapTree :: (a -> b) -> Tree a -> Tree b
mapTree f Leaf = Leaf
mapTree f (Node x t1 t2) = Node (f x) (mapTree f t1) (mapTree f t2)
```

LA RAZON DE ESTA GUIA.

Esta guía fue creada con la finalidad de que los próximos estudiantes que tengan que lidiar con este lenguaje, no estén tan perdidos como lo estuvimos nosotros.. Siempre hubo un ambiente de incertidumbre (por lo menos en mi grupo) de cómo hacer las cosas, y aprendimos de la forma más difícil. Ahora con esta guía tendrán muchos ejemplos para copiar en sus ordenadores, todos los programas aquí copiados corren perfectamente por que los he probado uno por uno. Si llegas a tener algún problema o alguna pregunta no dudes en preguntarle a tu profesor, preparador (mejor aún), o a mi. Para cualquier pregunta escribe a:

anleon@ucab.edu.ve

Aunque siempre habrán preguntas sin respuestas....
(si la cago...)

Angel "Kangaroo Gubatron" León.
anleon@ucab.edu.ve
ICQ # 14232182

septiembre de 1998

Agradecimientos.

Aunque no saben de la creación de esta guía quiero agradecer a:

- Luis Felipe Sintjago (aunque aveces se encaletaba los problemas, muchas veces me ayudo).
- Dioni Escalante (por todos los truquitos malditos en UNIX).
- Carlos Carbonell (por las copiadas en los quizes).
- Liliana Anzola (por su paciencia, y por esperar a que uno le entregara las prácticas)
- Mauricio Echezuría. (por prestarme su PC todo el semestre pasado)
- Maritza Martínez (por prestarme su libro de Algoritmo).
- Y a todos las personas que se merecen agradecimiento pero que se me olvidaron sus nombres.

INDICE

1998