David Rivera Morales || 320176876

La función par

Básicamente en esta función lo que hice fue implementar una firma un poco diferente a lo visto en el laboratorio ya que utilice IO que son tipos que causan side effects pero con los que puedo imprimir cadenas de caracteres para dar un mensaje más preciso de lo que quiero decir. En cuanto a funciones del preludio ocupe *mod* que es una función que regresa el módulo de una división para verificar que este fuera igual a 0 al evaluar mi número con 2 y así definir si era par o no.

```
par :: Int ->  IO [Char]
par n = if (mod n 2) == 0 then do {return "Es par"} else do {return "No es
par"}
```

La función impar

Esta función es muy similar a la de arriba pero lo único diferente es la comparación que hago del módulo ya que ahora tiene que ser diferente de 0 para que no sea un número par y sea impar.

```
impar :: Int -> IO [Char]
impar n = if (mod n 2) /= 0 then do {return "Es impar"} else do {return "No
es impar"}
```

La función minimo

Dentro de esta función lo que hice fue una firma de 3 int ya que recibe dos números *n* y *m* y regreso un tercer número, aquí lo que hice fue comparar si n es menor o igual para regresar n ya que de esta manera sería menor que m , si no es menor o igual entonces regreso m ya que es menor que n.

```
minimo :: Int -> Int -> Int
minimo n m = if (n <= m) then  n else m
```

La función maximo

Esta función es muy similar a la de arriba pero lo único diferente es que ahora comparo si n es mayor o igual a m para regresar n ya que de esta manera sería mayor que m , si no es mayor o igual entonces regreso m ya que es mayor que n.

```
maximo :: Int -> Int -> Int
maximo n m = if (n >= m) then  n else  m
```

La función absoluto

En esta función lo que hice fue una firma de 2 int ya que recibe un número entero y regreso un número entero, aquí lo que hice fue comparar si n es mayor o igual a 0 para regresar el valor de n pero con un signo positivo ya que de esta manera sería el valor absoluto de n , si no es mayor o igual entonces regreso el valor de -n ya que es menor que 0 y por lo tanto -n donde n es un entero menor a 0 es igual a n.

```
absoluto :: Int -> Int
absoluto n = if n >= 0 then  n else -n
```

La función divE

Esta función es un poco diferente a las demás ya que lo que hice fue una firma de 3 int que recibe *n* y *m* y regresa un tercer número entero. Primero veo si están tratando de dividir con 0 para en ese caso buscar el patrón y regresar un error. Para solucionar por completo el problema de la división entera lo que hice fue utilizar una función auxiliar que se llama *divEAux* que recibe 3 int y regresa un int, lo que hace esta función es que recibe *n* y *m* y un contador que empieza en 0 y va aumentando cada vez que se cumple la condición de que *n* no sea menor a *m* entonces regreso el contador más 1 ya que es la cantidad de veces que se cumplió la condición de que *n* era menor a *m* hasta que ya no se cumpliera y así regresar el resultado de la división entera.Si *n* es menor a *m* entonces regreso el contador ya que es la cantidad de veces que se cumplió la condición de que *n* era menor a *m*.

```
divE 0 m = error "No se puede dividir entre 0"
divE n m = divEAux n m 0

divEAux :: Int -> Int -> Int -> Int
divEAux n m acc = if (n < m) then acc else divEAux (n-m) m (acc+1)
```

La función cabeza y cola

En la función cabeza lo que hice fue una firma un poco diferente a todas las anteriores, ahora lo que recibo es una lista de enteros y regreso un entero. En mi primer caso recibo una lista con un solo elemento y regreso como cabeza ese mismo elemento ya que no hay ningún otro elemento que pueda ser cabeza de la lista, en mi segundo caso recibo una lista con más de un elemento y regreso como cabeza el primer elemento de la lista ya que es el único que puede ser cabeza de la lista.

En la función cola nuevamente recibo una lista de enteros y regreso una lista de enteros pero esta vez es una lista con un elemento menos que la lista que recibo. En mi primer caso recibo una lista con un solo elemento y regreso como cola una lista vacía ya que no hay ningún otro elemento que pueda ser cola de la

lista, en mi segundo caso recibo una lista con más de un elemento y regreso como cola la lista que recibo pero sin el primer elemento ya que es el único que puede ser cola de la lista.

```
cabeza :: [Int] -> Int
cabeza [x] = x
cabeza (x:xs) = x

cola :: [Int] -> [Int]
cola [] = []
cola (x:xs) = xs
```

La función quita

Lo que hice en esta función fue una firma de un entero y una lista de enteros y regreso una lista de enteros con n elementos menos que la lista que recibo. En mi primer caso quiero quitar un elemento de una lista vacía pero no puedo ya que por definición no es posible por lo cual lanzo un error, en el segundo caso recibo 0 y una lista de enteros,regreso la misma porque no estoy quitando ningún elemento.En mi segundo caso recibo un entero y una lista con más de un elemento donde lo que voy a hacer es definir una función constructiva donde voy a ir quitando elementos de la lista hasta que n llegue a 0 (disminuyendo su valor en 1 cada vez que se llama la función) y así regresar la lista con n elementos menos que la lista que recibo.

```
quita :: Int -> [Int] -> [Int]
quita n [] = error "No se puede quitar elementos de una lista vacia"
quita 0 lst = lst
quita n (x:xs) = quita (n-1) xs
```

La función enesimo

Lo que hice en esta función fue una firma de un entero y una lista de enteros que regresa un entero. En mi primer recibo n elemento que quiero regresar pero al ser una lista no puedo saber la ubicación de un elemento que no existe por eso regreso un error , en el segundo caso recibo 0 y una lista de enteros,regreso el primer elemento de la lista ,con el método cabeza que había implementado anteriormente, ya que es el elemento que se encuentra en la posición 0.En mi segundo caso recibo un entero y una lista con más de un elemento en la cual voy a ir quitando elementos de la lista hasta que n llegue a 0 (disminuyendo su valor en 1 cada vez que se llama la función) y así regresar el elemento que se encuentra en la posición n de la lista que recibo.

```
enesimo :: Int -> [Int] -> Int
enesimo n [] = error "No se puede obtener el enesimo elemento de una lista
vacia"
enesimo 0 lst = cabeza lst
enesimo n (x:xs) = enesimo (n-1) xs
```