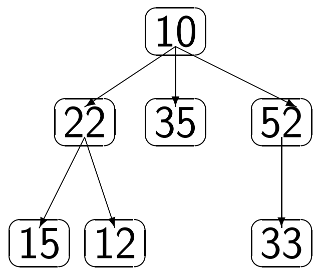
Árboles

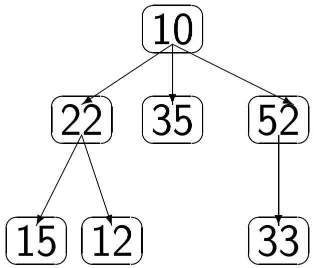
Árboles generales

Un árbol es una estructura no lineal acíclica utilizada para organizar información de forma eficiente.
La definición es recursiva:
Un árbol es una colección de valores $\{v_1, v_2, \ldots v_n\}$ tales que

- ✓ Si $n = 0$ el árbol se dice vacío.
- ✓ En otro caso, existe un valor destacado que se denomina *raíz* (p.e. v_1), y los demás elementos forman parte de colecciones disjuntas que a su vez son árboles. Estos árboles se llaman subárboles del raíz.



Representación en Haskell



$$\textbf{data}\quad \textit{Árbol}\ a\ =\ \textit{Vacío}\ |\ \textit{Nodo}\ \underbrace{a}_{raíz}\ \underbrace{[\textit{Árbol}\ a]}_{hijos}\ \textbf{deriving}\ \textit{Show}$$

```
a1 :: Árbol Integer
a1 = Nodo 10 [a11, a12, a13]
  where
     a11 = Nodo 22 [hoja 15, hoja 12]
     a12 = hoja 35
     a13 = Nodo 52 [hoja 33]

profundidad                :: Árbol a → Integer
profundidad Vacío          = 0
profundidad (Nodo _ [])    = 1
profundidad (Nodo _ xs)    = (+1) . maximum . map profundidad $ xs
```

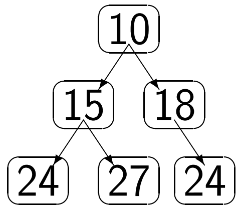
```
hoja   :: a → Árbol a
hoja x = Nodo x []

raíz                :: Árbol a → a
raíz Vacío          = error "raíz de árbol vacío"
raíz (Nodo x _)     = x

tamaño              :: Árbol a → Integer
tamaño Vacío        = 0
tamaño (Nodo _ xs)  = (+1) . sum . map tamaño $ xs
```

Árboles binarios

Un árbol binario es árbol tal que cada nodo tiene como máximo dos subárboles.



$$
\begin{aligned}
&\textbf{data}\,\textit{ÁrbolB}\ a\ = \textit{VacíoB} \\
&\qquad\qquad\quad \mid \textit{NodoB}\ \underbrace{(\textit{ÁrbolB}\ a)}_{\textit{sub izq}}\ \underbrace{a}_{\textit{raíz}}\ \underbrace{(\textit{ÁrbolB}\ a)}_{\textit{sub der}}\ \textbf{deriving}\ \textit{Show}
\end{aligned}
$$

Consideraremos que las tres componentes del constructor $NodoB$ son el subárbol izquierdo, el dato raíz y el subárbol derecho respectivamente.

Si falta un subárbol, se usa $VacíoB$.

$$
\begin{aligned}
&a2\ ::\ \textit{ÁrbolB Integer} \\
&a2\ = \textit{NodoB}\ aI\ 10\ aD \\
&\quad \textbf{where} \\
&\qquad\quad aI \quad = \textit{NodoB}\ aII\ 15\ aID \\
&\qquad\quad aD \quad = \textit{NodoB}\ aDI\ 18\ aDD \\
&\qquad\quad aII \quad = \textit{hojaB}\ 24 \\
&\qquad\quad aID \quad = \textit{hojaB}\ 27 \\
&\qquad\quad aDI \quad = \textit{VacíoB} \\
&\qquad\quad aDD \quad = \textit{hojaB}\ 24
\end{aligned}
$$

$$
\begin{aligned}
&\textit{hojaB}\quad\ \ ::\ a \to \textit{ÁrbolB}\ a \\
&\textit{hojaB}\ x\ = \textit{NodoB}\ \textit{VacíoB}\ x\ \textit{VacíoB}
\end{aligned}
$$

Árboles binarios (II)

$$
\begin{array}{ll}
raízB & :: ÁrbolB\ a\ \rightarrow\ a \\
raízB\ VacíoB & = error\ \text{"raíz de árbol vacío"} \\
raízB\ (NodoB\ \_\ x\ \_) & = x
\end{array}
$$

$$
\begin{array}{ll}
tamañoB & :: ÁrbolB\ a\ \rightarrow\ Integer \\
tamañoB\ VacíoB & = 0 \\
tamañoB\ (NodoB\ i\ r\ d) & = 1\ +\ tamañoB\ i\ +\ tamañoB\ d
\end{array}
$$

$$
\begin{array}{ll}
profundidadB & :: ÁrbolB\ a\ \rightarrow\ Integer \\
profundidadB\ VacíoB & = 0 \\
profundidadB\ (NodoB\ i\ r\ d) & = 1\ +\ max\ (profundidadB\ i)\ (profundidadB\ d)
\end{array}
$$

EJERCICIO: Define funciones para

- ✓ Comprobar si un dato pertenece a un árbol.
- ✓ Contar cuántas veces aparece un dato en un árbol.
- ✓ Sumar todos los nodos de un árbol de números.
- ✓ Calcular el valor máximo almacenado en un árbol.

Da versiones para árboles binarios y generales.

Recorrido de árboles binarios

$$
\begin{array}{ll}
enOrdenB & :: ÁrbolB\ a \rightarrow [a] \\
enOrdenB\ VacíoB & = [\,] \\
enOrdenB\ (NodoB\ i\ r\ d) & = enOrdenB\ i +\!\!+ (r : enOrdenB\ d) \\
preOrdenB & :: ÁrbolB\ a \rightarrow [a] \\
preOrdenB\ VacíoB & = [\,] \\
preOrdenB\ (NodoB\ i\ r\ d) & = (r : preOrdenB\ i) +\!\!+ preOrdenB\ d \\
postOrdenB & :: ÁrbolB\ a \rightarrow [a] \\
postOrdenB\ VacíoB & = [\,] \\
postOrdenB\ (NodoB\ i\ r\ d) & = postOrdenB\ i +\!\!+ postOrdenB\ d +\!\!+ [r]
\end{array}
$$

MAIN> $enOrdenB\ a2$
$[24, 15, 27, 10, 18, 24] :: [Integer]$
MAIN> $preOrdenB\ a2$
$[10, 15, 24, 27, 18, 24] :: [Integer]$
MAIN> $postOrdenB\ a2$
$[24, 27, 15, 24, 18, 10] :: [Integer]$

EJERCICIO: Define los recorridos en pre-orden y post-orden para árboles generales.

La función fmap

La función map solo está predefinida para listas, pero existe una versión sobrecargada predefinida en la siguiente clase:

$$\begin{array}{l} \textbf{class}\ \mathit{Functor}\ m\ \textbf{where} \\ \quad \mathit{fmap}\ ::\ (a \to b) \to m\ a \to m\ b \end{array}$$

Por ejemplo, las listas son una instancia predefinida de esta clase:

$$\begin{array}{l} \textbf{instance}\ \mathit{Functor}\ [\,]\ \textbf{where} \\ \quad \mathit{fmap} = \mathit{map} \end{array}$$

Es posible usar tanto map como $fmap$ con listas.
La función $fmap$ también tiene sentido para árboles binarios:

$$\textbf{instance}\ \mathit{Functor}\ \mathit{ÁrbolB}\ \textbf{where}$$

$$\begin{array}{ll} \mathit{fmap}\ f\ \mathit{VacíoB} & = \mathit{VacíoB} \\ \mathit{fmap}\ f\ (\mathit{NodoB}\ i\ r\ d) & = \mathit{NodoB}\ (\mathit{fmap}\ f\ i)\ (f\ r)\ (\mathit{fmap}\ f\ d) \end{array}$$

O para árboles generales:

$$\textbf{instance}\ \mathit{Functor}\ \mathit{Árbol}\ \textbf{where}$$

$$\begin{array}{ll} \mathit{fmap}\ f\ \mathit{Vacío} & = \mathit{Vacío} \\ \mathit{fmap}\ f\ (\mathit{Nodo}\ x\ xs) & = \mathit{Nodo}\ (f\ x)\ (\mathit{map}\ (\mathit{fmap}\ f)\ xs) \end{array}$$

Una función que duplique los datos enteros almacenados en cualquier funtor:

$$\begin{array}{l} \mathit{duplicar}\ ::\ \mathit{Functor}\ f \Rightarrow f\ \mathit{Integer} \to f\ \mathit{Integer} \\ \mathit{duplicar} = \mathit{fmap}\ (*2) \end{array}$$

Plegado de Árboles

Consideremos

$$
\begin{array}{ll}
sum\acute{A}rbolB & :: \acute{A}rbolB\ Integer \ \rightarrow\ Integer \\
sum\acute{A}rbolB\ Vac\acute{\imath}oB & = 0 \\
sum\acute{A}rbolB\ (NodoB\ i\ r\ d) & = sumar\ (sum\acute{A}rbolB\ i)\ r\ (sum\acute{A}rbolB\ d) \\
\quad \mathbf{where} & \\
\qquad sumar\ x\ y\ z\ =\ x + y + z &
\end{array}
$$

$$
\begin{array}{ll}
enOrdenB & :: \acute{A}rbolB\ a\ \rightarrow\ [a] \\
enOrdenB\ Vac\acute{\imath}oB & = [\,] \\
enOrdenB\ (NodoB\ i\ r\ d) & = concatenar\ (enOrdenB\ i)\ r\ (enOrdenB\ d) \\
\quad \mathbf{where} & \\
\qquad concatenar\ x\ y\ z\ =\ x +\!\!+ [y] +\!\!+ z &
\end{array}
$$

Ambas funciones siguen el esquema:

$$
\begin{array}{ll}
fun\ Vac\acute{\imath}oB & = \boxed{z} \\
fun\ (NodoB\ i\ r\ d) & = \boxed{f}\ (fun\ i)\ r\ (fun\ d)
\end{array}
$$

$$
\begin{array}{ll}
fold\acute{A}rbolB & :: (b\ \rightarrow\ a\ \rightarrow\ b\ \rightarrow\ b)\ \rightarrow\ b\ \rightarrow\ \acute{A}rbolB\ a\ \rightarrow\ b \\
fold\acute{A}rbolB\ f\ z & = fun \\
\quad \mathbf{where} & \\
\qquad fun\ Vac\acute{\imath}oB & = z \\
\qquad fun\ (NodoB\ i\ r\ d) & = f\ (fun\ i)\ r\ (fun\ d)
\end{array}
$$

O equivalentemente, por cumplirse $fold\acute{A}rbolB\ f\ z\ =\ fun$:

$$
\begin{array}{ll}
fold\acute{A}rbolB & :: (b\ \rightarrow\ a\ \rightarrow\ b\ \rightarrow\ b)\ \rightarrow\ b\ \rightarrow\ \acute{A}rbolB\ a\ \rightarrow\ b \\
fold\acute{A}rbolB\ f\ z\ Vac\acute{\imath}oB & = z \\
fold\acute{A}rbolB\ f\ z\ (NodoB\ i\ r\ d) & = f\ (fold\acute{A}rbolB\ f\ z\ i)\ r\ (fold\acute{A}rbolB\ f\ z\ d)
\end{array}
$$

Plegado de Árboles (II)

Recordemos

$$
\begin{array}{ll}
sum\acute{A}rbolB\ \ Vac\acute{\imath}oB & = 0 \\
sum\acute{A}rbolB\ (NodoB\ \ i\ \ r\ \ d) & = sumar\ (sum\acute{A}rbolB\ \ i)\ r\ (sum\acute{A}rbolB\ \ d)\ \textbf{where}\ sumar\ x\ y\ z\ = \ x + y + z
\end{array}
$$

$$
\begin{array}{ll}
enOrdenB\ \ Vac\acute{\imath}oB & = [\,] \\
enOrdenB\ (NodoB\ \ i\ \ r\ \ d) & = concatenar\ (enOrdenB\ \ i)\ r\ (enOrdenB\ \ d)\ \textbf{where}\ concatenar\ x\ y\ z\ = \ x +\!\!+ [y] +\!\!+ z
\end{array}
$$

$$
\begin{array}{l}
fold\acute{A}rbolB \quad :: (b \to a \to b \to b) \to b \to \acute{A}rbolB\ a \to b \\
fold\acute{A}rbolB\ f\ z\ = fun \\
\quad \textbf{where} \\
\qquad fun\ Vac\acute{\imath}oB \qquad\quad = z \\
\qquad fun\ (NodoB\ i\ r\ d)\ = f\ (fun\ i)\ r\ (fun\ d)
\end{array}
$$

Así:

$$
\begin{array}{ll}
sum\acute{A}rbolB\ :: \acute{A}rbolB\ Integer \to Integer & enOrden\ :: \acute{A}rbolB\ a \to [a] \\
sum\acute{A}rbolB\ = fold\acute{A}rbolB\ (\lambda\ x\ y\ z \to x + y + z)\ 0 & enOrden\ = fold\acute{A}rbolB\ (\lambda\ x\ y\ z \to x +\!\!+ [y] +\!\!+ z)\ [\,]
\end{array}
$$

Para definir una función usando $fold\acute{A}rbolB$:

- ✓ Proporcionar el resultado (z) para el árbol vacío.
- ✓ Proporcionar función (f) que calcule el resultado a partir del resultado para el subárbol izquierdo, la raíz y el resultado para el subárbol derecho.

EJERCICIO: Define usando la función de plegado las funciones $tama\tilde{n}oB$, $profundidadB$ y $enOrdenB$

Plegado de Árboles (III)

Consideremos

$$\begin{array}{ll} sum\acute{A}rbol & :: \acute{A}rbol\ Integer \to Integer \\ sum\acute{A}rbol\ Vac\acute{\imath}o & = 0 \\ sum\acute{A}rbol\ (Nodo\ x\ xs) & = sumar\ x\ (map\ sum\acute{A}rbol\ xs) \\ \quad \mathbf{where} & \\ \quad\quad sumar\ n\ ns\ =\ n\ +\ sum\ ns & \end{array}$$

$$\begin{array}{ll} preOrden & :: \acute{A}rbolB\ a \to [a] \\ preOrden\ Vac\acute{\imath}o & = [\,] \\ preOrden\ (Nodo\ x\ xs) & = unir\ x\ (map\ preOrden\ xs) \\ \quad \mathbf{where} & \\ \quad\quad unir\ y\ ys\ =\ y\ :\ concat\ ys & \end{array}$$

Ambas funciones siguen el esquema:

$$\begin{array}{ll} fun\ Vac\acute{\imath}o & = \boxed{z} \\ fun\ (Nodo\ x\ xs) & = \boxed{f}\ x\ (map\ fun\ xs) \end{array}$$

La función $fold\acute{A}rbol$ captura el esquema de cómputo anterior:

$$\begin{array}{ll} fold\acute{A}rbol & :: (a \to [b] \to b) \to b \to \acute{A}rbol\ a \to b \\ fold\acute{A}rbol\ f\ z & = fun \\ \quad \mathbf{where} & \\ \quad\quad fun\ Vac\acute{\imath}o & = z \\ \quad\quad fun\ (Nodo\ x\ xs) & = f\ x\ (map\ fun\ xs) \end{array}$$

Así

$$sum\acute{A}rbol\ =\ fold\acute{A}rbol\ (\lambda\ n\ ns\ \to\ n\ +\ sum\ ns)\ \ 0$$

$$preOrden\ =\ fold\acute{A}rbol\ (\lambda\ y\ ys\ \to\ y\ :\ concat\ ys)\ \ [\,]$$

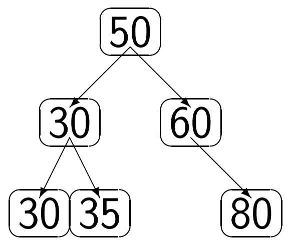
Árboles Binarios de búsqueda

Un árbol binario de búsqueda es un árbol binario tal que

✓ O bien es vacío

✓ O no es vacío y para cualquier nodo se cumple que:

- los elementos del correspondiente subárbol izquierdo son menores o iguales al almacenado en el nodo
- y los elementos del correspondiente subárbol derecho son estrictamente mayores al almacenado en el nodo



El árbol de la figura está ordenado

Árboles Binarios de búsqueda (II)

La siguiente función puede ser utilizada para comprobar si un árbol binario es de búsqueda:

$$
\begin{array}{ll}
es\acute{A}rbolBB & : Ord\ a\ \Rightarrow \acute{A}rbolB\ a\ \rightarrow\ Bool \\
es\acute{A}rbolBB\ Vac\acute{\imath}oB & = True \\
es\acute{A}rbolBB\ (NodoB\ i\ r\ d) & = todos\acute{A}rbolB\ (\leq r)\ i\ \&\& \\
& \quad todos\acute{A}rbolB\ (> r)\ d\ \&\& \\
& \quad es\acute{A}rbolBB\ i\ \&\& \\
& \quad es\acute{A}rbolBB\ d
\end{array}
$$

$$
\begin{array}{ll}
todos\acute{A}rbolB & :: (a\ \rightarrow\ Bool)\ \rightarrow\ \acute{A}rbolB\ a\ \rightarrow\ Bool \\
todos\acute{A}rbolB\ p\ Vac\acute{\imath}oB & = True \\
todos\acute{A}rbolB\ p\ (NodoB\ i\ r\ d) & = p\ r\ \&\& \\
& \quad todos\acute{A}rbolB\ p\ i\ \&\&\ todos\acute{A}rbolB\ p\ d
\end{array}
$$

La función de búsqueda es más eficiente ya que si el dato no coincide con la raíz solo hay que buscar en uno de los subárboles:

$$
\begin{array}{ll}
perteneceBB & ::\ Ord\ a\ \Rightarrow\ a\ \rightarrow \acute{A}rbolB\ a\ \rightarrow\ Bool \\
perteneceBB\ x\ Vac\acute{\imath}oB & = False \\
perteneceBB\ x\ (NodoB\ i\ r\ d) & \\
\quad \mid\ x\ ==\ r & = True \\
\quad \mid\ x\ <\ r & = perteneceBB\ x\ i \\
\quad \mid\ otherwise & = perteneceBB\ x\ d
\end{array}
$$

de modo que como máximo se realizan tantas comparaciones como profundidad tenga el árbol.

Árboles Binarios de búsqueda (III)

Función para insertar un nuevo dato dentro de un árbol de búsqueda, de modo que se obtenga otro árbol de búsqueda:

$$
\begin{array}{ll}
insertarBB & :: \ Ord\ a \ \Rightarrow \ a \rightarrow \acute{A}rbolB\ a \rightarrow \acute{A}rbolB\ a \\
insertarBB\ x\ Vac\acute{\imath}oB & = NodoB\ Vac\acute{\imath}oB\ x\ Vac\acute{\imath}oB \\
insertarBB\ x\ (NodoB\ i\ r\ d) & \\
\quad \mid\ x \leq r & = NodoB\ (insertarBB\ x\ i)\ r\ d \\
\quad \mid\ otherwise & = NodoB\ i\ r\ (insertarBB\ x\ d)
\end{array}
$$

Una propiedad interesante es que si se realiza una visita en orden de un árbol de búsqueda se obtiene una lista ordenada.

Es posible ordenar una lista de datos construyendo un árbol de búsqueda con sus elementos y recorriendo éste en orden:

$$
\begin{array}{l}
listaA\acute{A}rbolBB\ ::\ Ord\ a\ \Rightarrow\ [a]\ \rightarrow\ \acute{A}rbolB\ a \\
listaA\acute{A}rbolBB\ = foldr\ insertarBB\ Vac\acute{\imath}oB \\
treeSort\ ::\ Ord\ a\ \Rightarrow\ [a]\ \rightarrow\ [a] \\
treeSort\ = enOrdenB\ .\ listaA\acute{A}rbolBB
\end{array}
$$

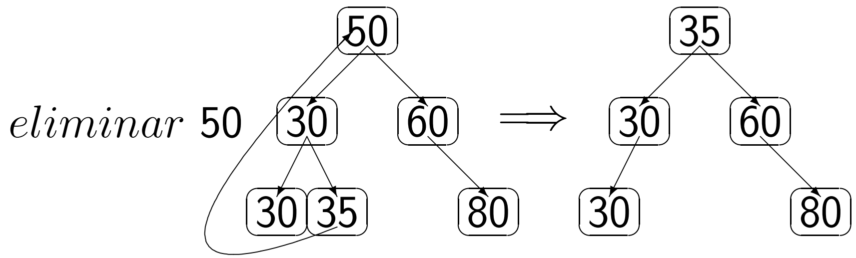
Por ejemplo:

$$
\begin{array}{l}
\textsc{Main}>\ treeSort\ [4,7,1,2,9] \\
[1,2,4,7,9]\ ::\ [Integer]
\end{array}
$$

Árboles Binarios de búsqueda (IV)

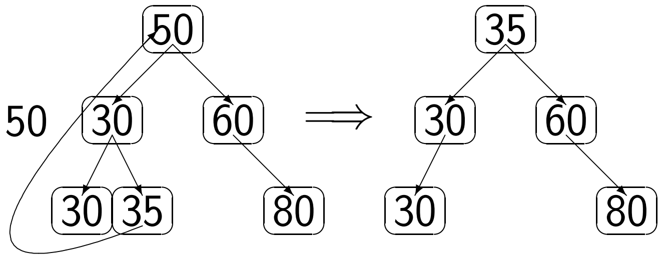
La eliminación de un dato es un poco más complicada: si el nodo a eliminar tiene dos subárboles no se puede dejar un hueco en su lugar.

Una solución consiste en tomar el mayor elemento del subárbol izquierdo del nodo a eliminar y colocar éste en el hueco. De este modo el nuevo árbol seguirá siendo ordenado:



Árboles Binarios de búsqueda (V)

eliminar 50




```
esVacíoB              :: ÁrbolB a → Bool
esVacíoB VacíoB        = True
esVacíoB _             = False

eliminarBB                 :: Ord a ⇒ a → ÁrbolB a → ÁrbolB a
eliminarBB x VacíoB         = VacíoB
eliminarBB x (NodoB i r d)
  | x < r                   = NodoB (eliminarBB x i) r d
  | x > r                   = NodoB i r (eliminarBB x d)
  | esVacíoB i              = d
  | esVacíoB d              = i
  | otherwise               = NodoB i' mi d
  where
     (mi, i')                          = tomaMaxBB i
     tomaMaxBB (NodoB i r VacíoB)      = (r, i)
     tomaMaxBB (NodoB i r d)           = (m, NodoB i r d')
      where
        (m, d') = tomaMaxBB d
```

$tomaMaxBBa$ devuelve un par (ma, a') donde ma es el mayor elemento del árbol de búsqueda a y a' es el árbol que se obtiene al eliminar el elemento ma del árbol a.
El elemento máximo se encuentra profundizando todo lo posible por la derecha en el árbol.

Inducción para árboles binarios

$$
\begin{aligned}
&\textbf{data}\, \textit{ÁrbolB}\ a \ = \textit{VacíoB} \\
&\qquad\qquad\quad \mid \textit{NodoB}\ (\textit{ÁrbolB}\ a)\ a\ (\textit{ÁrbolB}\ a)\ \textbf{deriving}\ \textit{Show}
\end{aligned}
$$

Principio de inducción para valores definidos del tipo ÁrbolB a

$$
\forall x :: \acute{A}rbolB\ \ a \ \ \cdot\ \ P(x) \Leftrightarrow \begin{cases} P(VacíoB) \\ \wedge \\ \forall i, d :: \acute{A}rbolB\ a,\ \forall r :: a\ \cdot \\ \quad P(i) \wedge P(d) \Rightarrow P(NodoB\ i\ r\ d) \end{cases}
$$

Vamos a probar que las funciones

$$
\begin{aligned}
&\textit{sumÁrbolB VacíoB} &&= 0 \\
&\textit{sumÁrbolB}\ (\textit{NodoB}\ i\ r\ d) &&= \textit{sumar}\ (\textit{sumÁrbolB}\ i)\ r\ (\textit{sumÁrbolB}\ d) \\
&\quad \textbf{where} \\
&\qquad \textit{sumar}\ x\ y\ z\ =\ x + y + z
\end{aligned}
$$

$$
\textit{sumÁrbolB}' \ = \ \textit{foldÁrbolB}\ (\lambda\, x\ y\ z\ \rightarrow\ x\ +\ y\ +\ z)\ 0
$$

calculan el mismo resultado:

$$
\forall\, x\ ::\ \textit{ÁrbolB Integer}\ \cdot\ \textit{sumÁrbolB}\ x\ \equiv\ \textit{sumÁrbolB}'\ x
$$