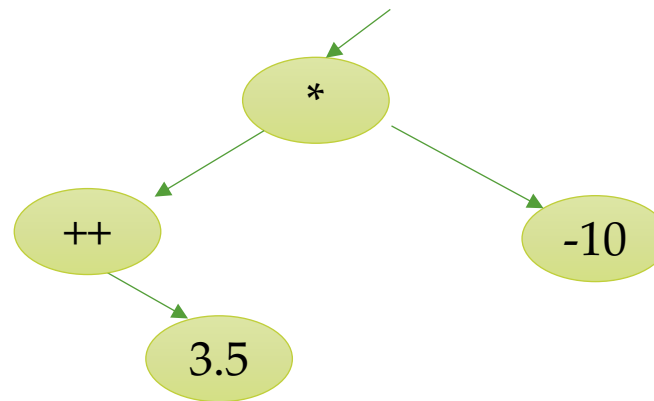


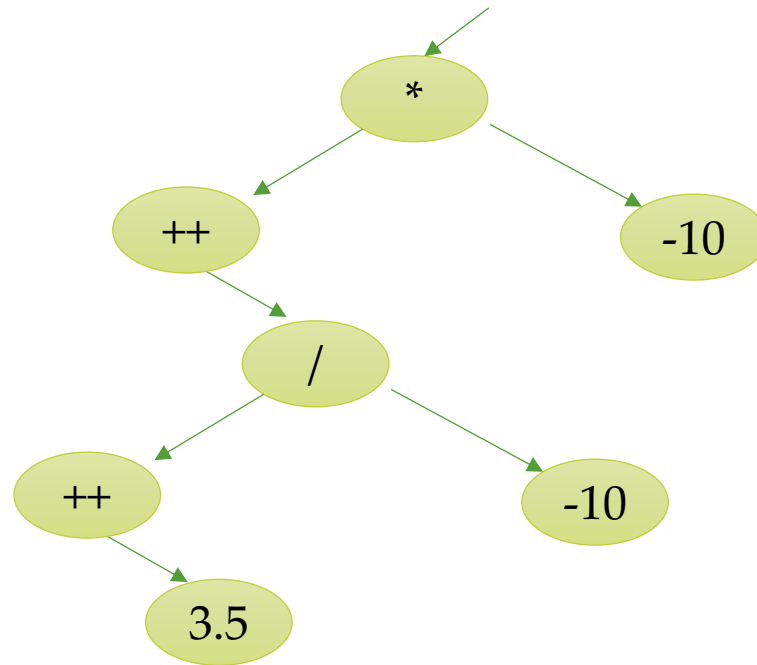
El árbol de expresiones que vimos es especial porque todos los operadores son binarios.

Un árbol de expresiones, pero con operadores unarios/binarios tendría otra forma.

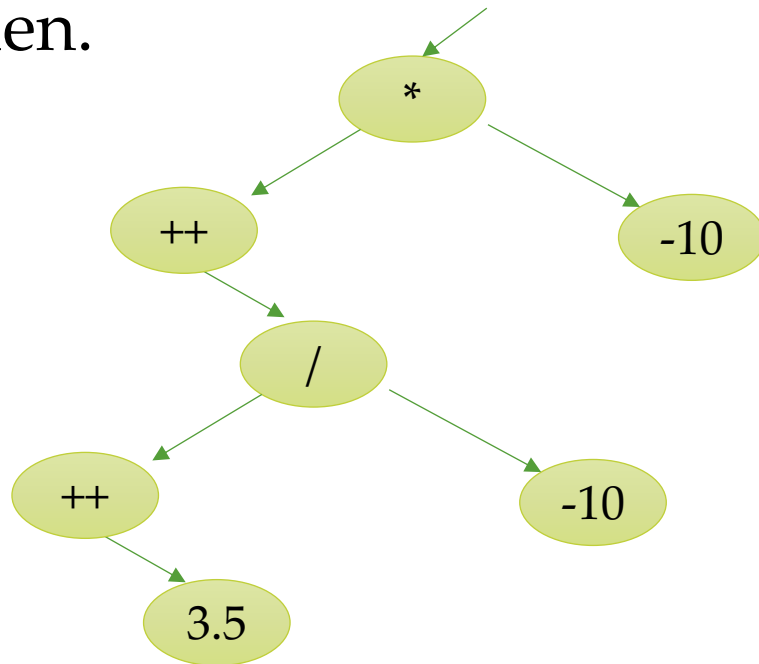
Ej:



Ej:



Si quisiéramos guardar sus datos en un archivo de texto, para luego (en otro momento) reconstruirlo desde el archivo de texto, algún recorrido vendría bien.





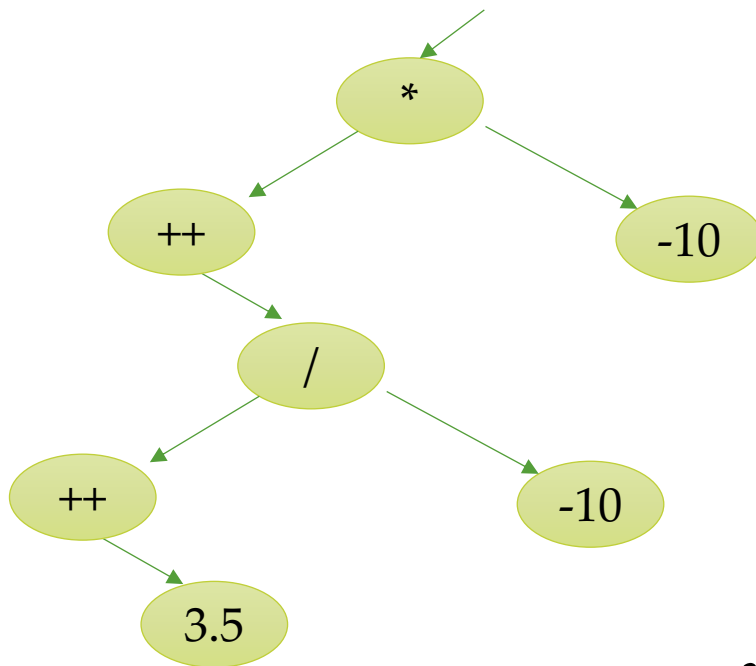
Si quisiéramos guardar sus datos en un archivo para alguno de estos propósitos:

- Reconstruirlo leyéndolo.
- Sin haberlo construido nunca, editar un archivo, escribir la info con su estructura y leerlo leyéndolo

Un archivo de texto es una buena opción.

Se pueden generar convenciones para esto, pero hay una muy utilizada que consiste en guardar los datos “planos”, sin indicar a qué nodo corresponde la info. Claramente esa info está implícita en el archivo.

Almacenar su recorrido, podría servir.



Preorder:

\* ++ / ++ 3.5 -10 -10

Inorder:

++ ++ 3.5 / -10 \* -10

Postorder:

3.5 ++ -10 / ++ -10 \*

Por niveles:

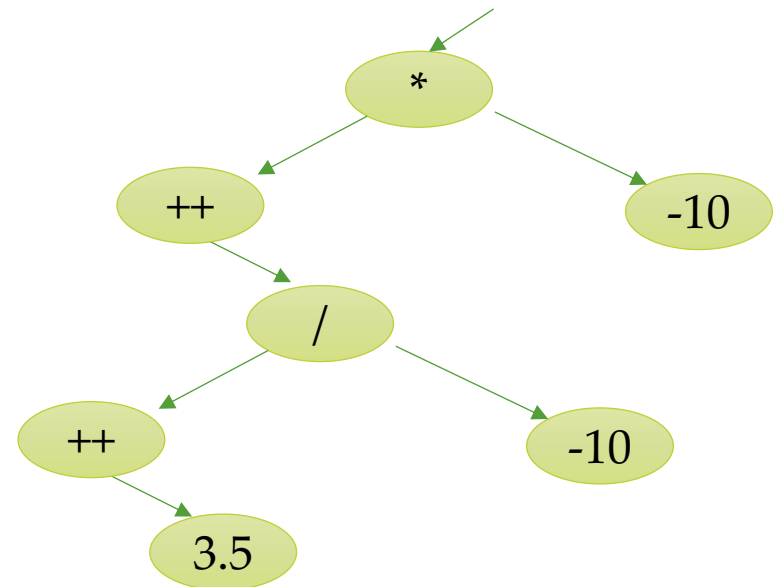
```
      *
    ++   -10
      /
    ++   -10
  3.5
```

Facil de leer para  
personas

¿Cómo armaríamos el árbol si la expresión vienen por niveles?

Por niveles:

\*  
++                  -10  
/   
++    -10  
3.5



Rta: para no desfasarnos

Tenemos que colocar

“dummy” símbolos para

Completar los espacios. Elegir

Algun símbolo “metadato” que no sea parte del lenguaje.

Ej: “?”.

Por niveles:

\*

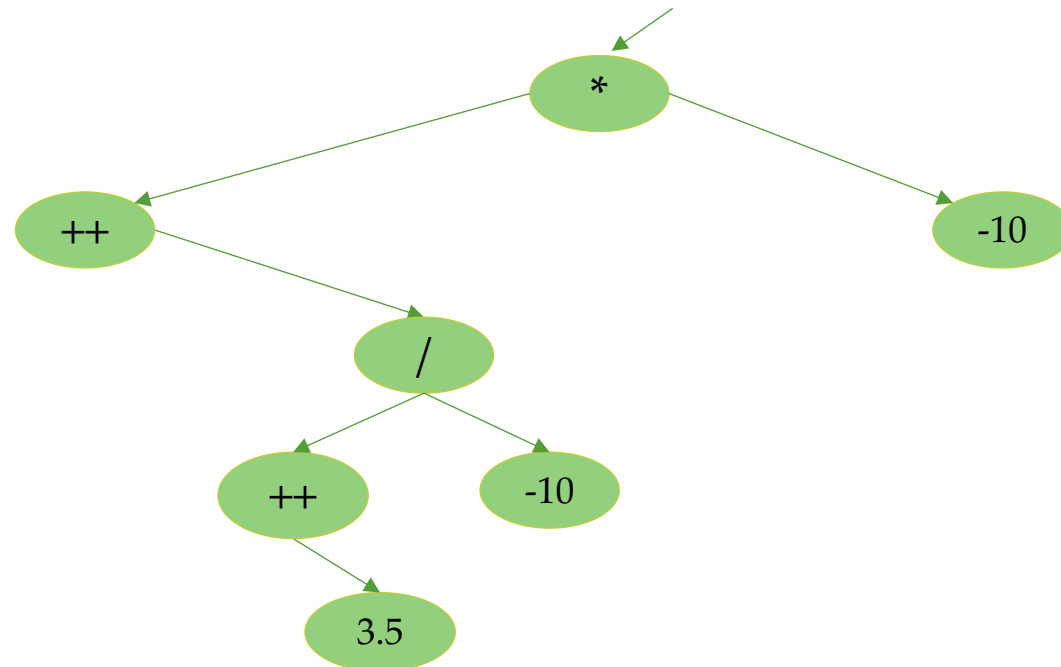
++                      -10

/

++                      -10

3.5

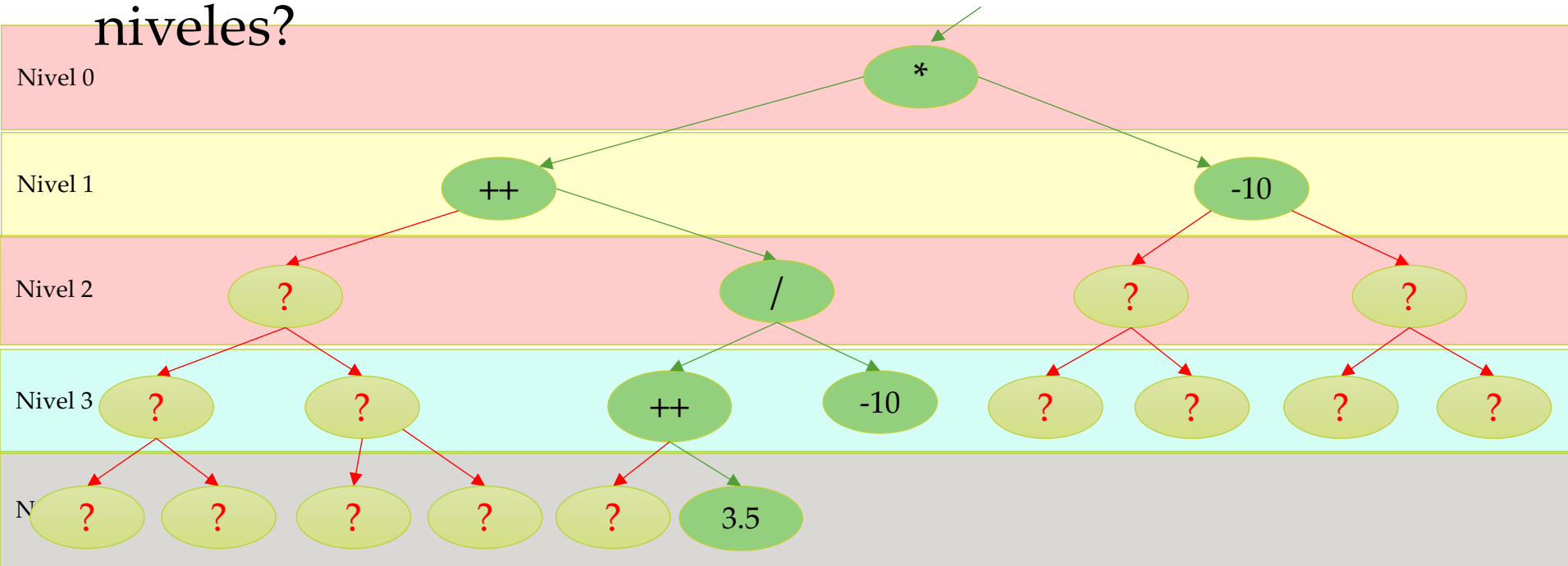
¿Cómo armaríamos el árbol si la expresión vienen por niveles?



Por niveles:

$$\begin{array}{r} ++ \quad -10 \\ / \\ ++ \quad -10 \\ 3.5 \end{array}$$

¿Cómo armaríamos el árbol si la expresión vienen por niveles?



Por niveles con placeholders:

*	++	-10	?	/	?	?	?	?	++	-10	?	?	?	?	?	?	?	3.5
	Nivel 1		Nivel 2		Nivel 3													Nivel 4 (posiblemente incompleto)



¿Cómo armariamos el árbol si la expresión vienen por niveles?

Genero raíz y postergo qué hay que hacer con ella, hasta que llegue el token. Ni siquiera sé si tendrá cero, uno o dos hijos

\$20



Pendientes:

\$20 consumir

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
Saco pendiente y leo token

\$20

\*

Pendientes:  
\$20 consumir

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? ? ? 3.5

Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
 Completo dato. Como no sé si el \$20 tendrá cero, uno o dos hijos  
 (depende del token), pido que cuando llegue el momento se los  
 procese.

\$20

\*

Pendientes:

\$20 izq

\$20 der

↓  
 Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? ? ? 3.5

Nivel 1

Nivel 2

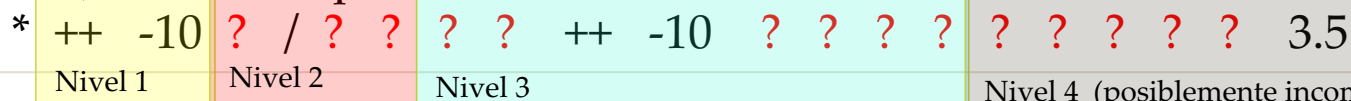
Nivel 3

Nivel 4 (posiblemente incompleto)

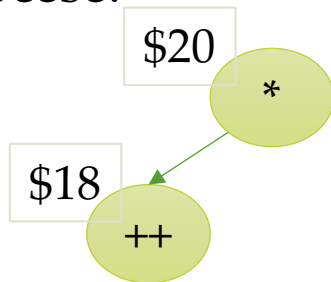
¿Cómo armariamos el árbol si la expresión vienen por niveles?  
Saco pendiente y leo token



Por niveles con placeholders:



¿Cómo armaríamos el árbol si la expresión vienen por niveles?  
 Completo dato. Como no sé si \$18 tendrá cero, uno o dos hijos  
 (depende del token), pido que cuando llegue el momento se los  
 procese.



Pendientes:

\$20 der  
 \$18 izq  
 \$18 der

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? ? ? 3.5

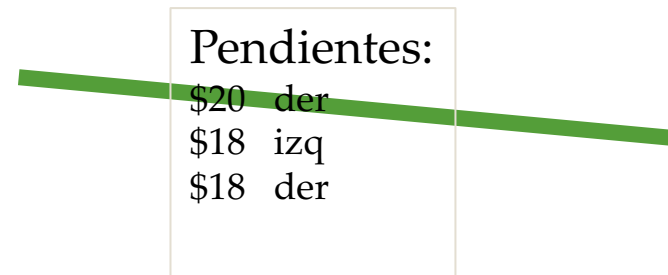
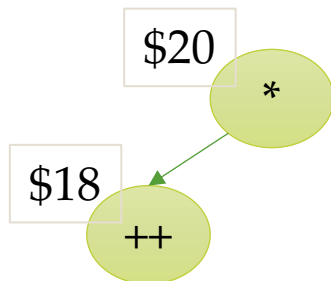
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

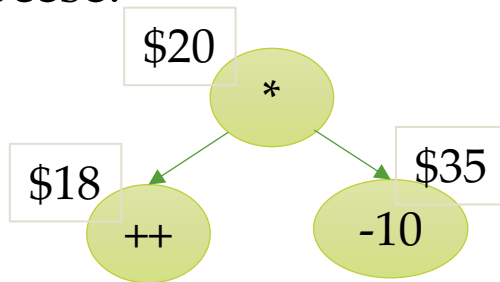
¿Cómo armariamos el árbol si la expresión vienen por niveles?  
Saco pendiente y leo token



Por niveles con placeholders:

*	++	-10	?	/	?	?	?	?	?	++	-10	?	?	?	?	?	?	?	?	?	?	3.5
Nivel 1			Nivel 2			Nivel 3			Nivel 4 (posiblemente incompleto)													

¿Cómo armaríamos el árbol si la expresión vienen por niveles?  
 Completo dato. Como no sé si \$35 tendrá cero, uno o dos hijos  
 (depende del token), pido que cuando llegue el momento se los  
 procese.



Pendientes:

\$18 izq  
 \$18 der  
 \$35 izq  
 \$35 der

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

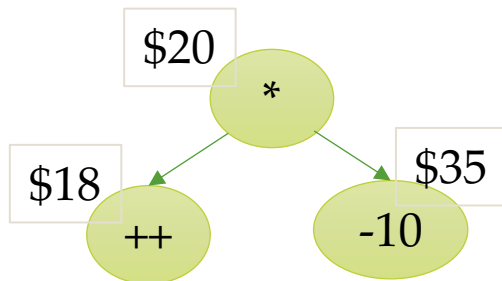
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
Saco pendiente y leo token



Pendientes:

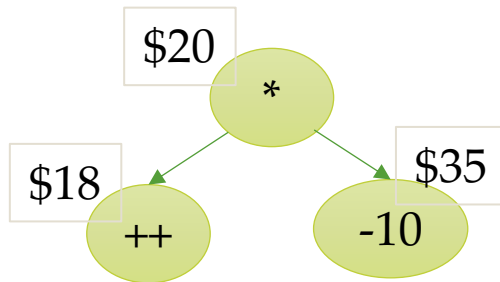
\$18 izq  
\$18 der  
\$35 izq  
\$35 der

Por niveles con placeholders:

*	++	-10	?	/	?	?	?	?	?	++	-10	?	?	?	?	?	?	?	3.5
Nivel 1			Nivel 2			Nivel 3			Nivel 4 (posiblemente incompleto)										



¿Cómo armariamos el árbol si la expresión vienen por niveles?  
 Como es "dummy" no lo pongo en el árbol. Igual, pido pendiente para él.



Pendientes:

\$18 der  
 \$35 izq  
 \$35 der  
 Null consumir  
 Null consumir

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

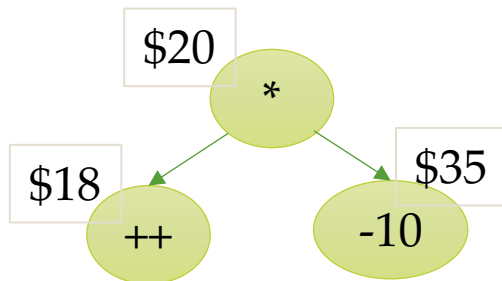
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
Saco pendiente y leo token.



Pendientes:

~~\$18 der~~  
\$35 izq  
\$35 der  
Null consumir  
Null consumir

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

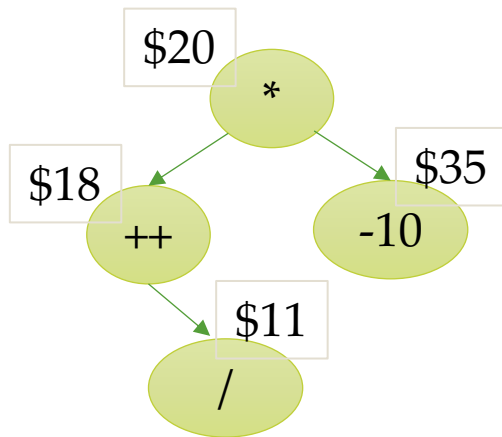
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armaríamos el árbol si la expresión vienen por niveles?  
 Coloco dato. Como no sé si \$11 tendrá cero, uno o dos hijos (depende del token), pido que cuando llegue el momento se los procese.



Pendientes:

\$35 izq  
 \$35 der  
 Null consumir  
 Null consumir  
 \$11 izq  
 \$11 der

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

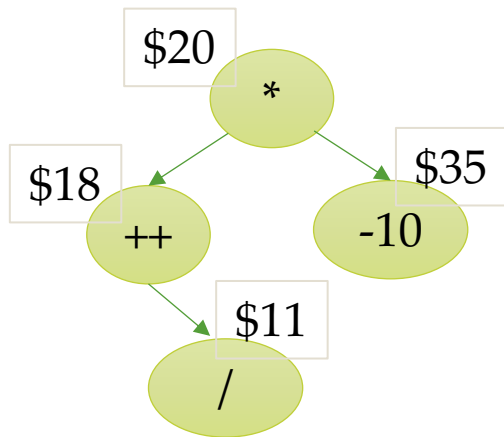
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
Saco pendiente y leo token.



Pendientes:

~~\$35 izq~~  
~~\$35 der~~  
 Null consumir  
 Null consumir  
 \$11 izq  
 \$11 der

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

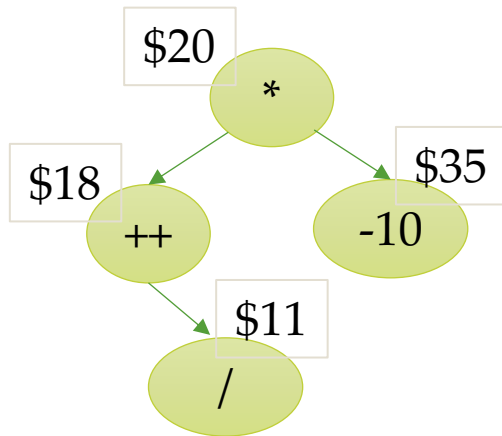
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
 Como es "dummy" no lo pongo en el árbol. Igual, pido pendiente para él.



Pendientes:

\$35 der  
 Null consumir  
 Null consumir  
 \$11 izq  
 \$11 der  
 Null consumir  
 Null consumir

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

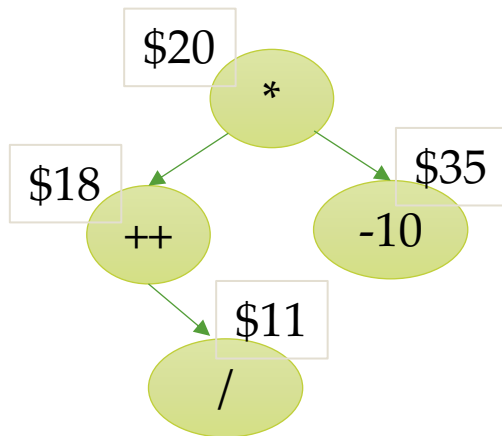
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
Saco pendiente y leo token



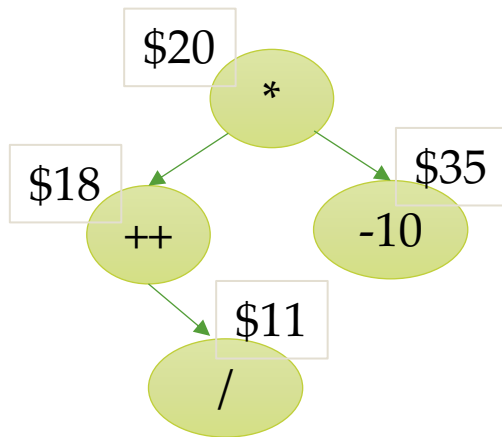
Pendientes:

~~\$35 der~~  
Null consumir  
Null consumir  
\$11 izq  
\$11 der  
Null consumir  
Null consumir

Por niveles con placeholders:

*	++	-10	?	/	?	?	?	?	++	-10	?	?	?	?	?	?	?	?	3.5
Nivel 1			Nivel 2			Nivel 3				Nivel 4 (posiblemente incompleto)									

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
 Como es "dummy" no lo pongo en el árbol. Igual, pido pendiente para él.

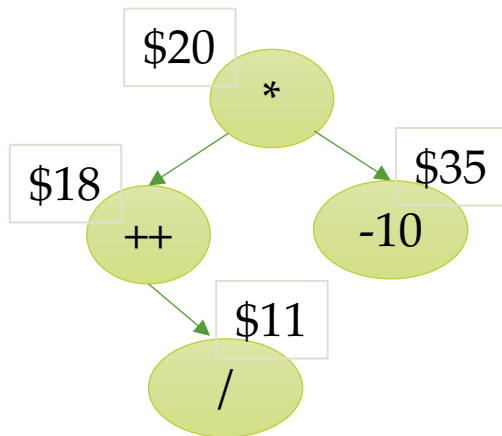


Pendientes:  
 Null consumir  
 Null consumir  
 \$11 izq  
 \$11 der  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir

Por niveles con placeholders:

*	++	-10	?	/	?	?	?	?	++	-10	?	?	?	?	?	?	?	?	?	3.5
	Nivel 1		Nivel 2			Nivel 3										Nivel 4 (posiblemente incompleto)				

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
Saco pendiente y leo token



Pendientes:  
Null consumir  
Null consumir  
\$11 izq  
\$11 der  
Null consumir  
Null consumir  
Null consumir  
Null consumir

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

Nivel 1

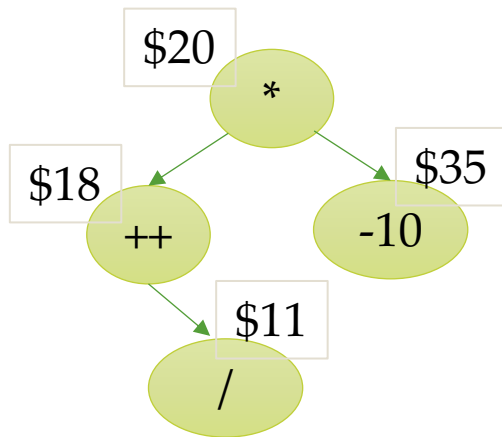
Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)



¿Cómo armariamos el árbol si la expresión vienen por niveles?  
 Como es "dummy" no lo pongo en el árbol. Igual, pido pendiente para él.



Pendientes:  
 Null consumir  
 \$11 izq  
 \$11 der  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

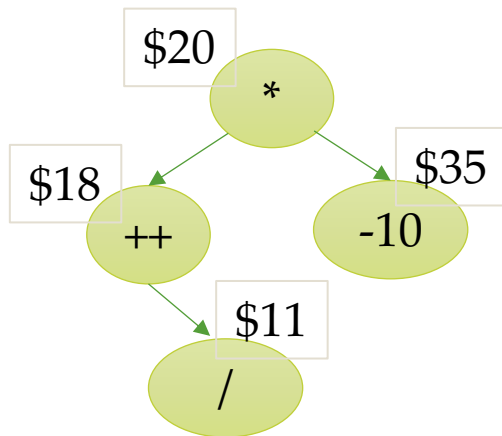
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
Saco pendiente y leo token



Pendientes:

Null consumir  
\$11 izq  
\$11 der  
Null consumir  
Null consumir  
Null consumir  
Null consumir  
Null consumir  
Null consumir

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? ? ? 3.5

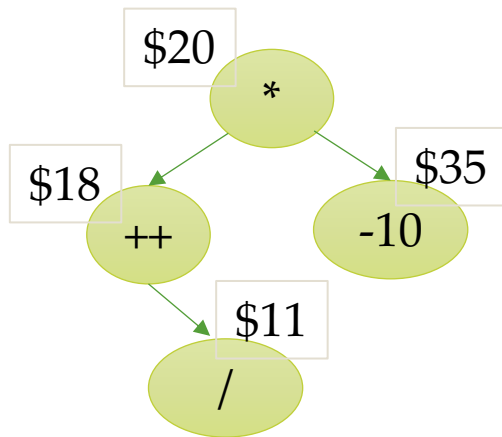
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
 Como es "dummy" no lo pongo en el árbol. Igual, pido pendiente para él.



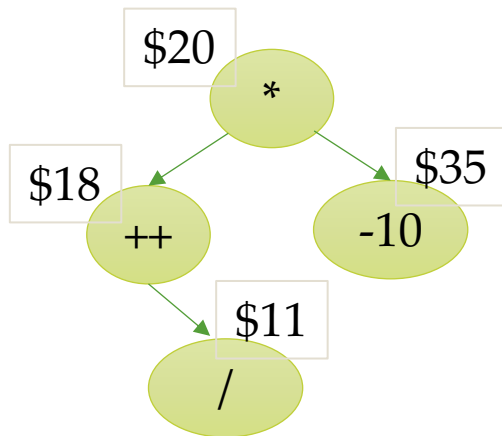
Pendientes:

\$11 izq  
 \$11 der  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir

Por niveles con placeholders:

*	++	-10	?	/	?	?	?	?	++	-10	?	?	?	?	?	?	?	?	3.5
Nivel 1			Nivel 2			Nivel 3			Nivel 4 (posiblemente incompleto)										

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
Saco pendiente de la cola y leo token



Pendientes:

~~\$11 izq~~  
~~\$11 der~~  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

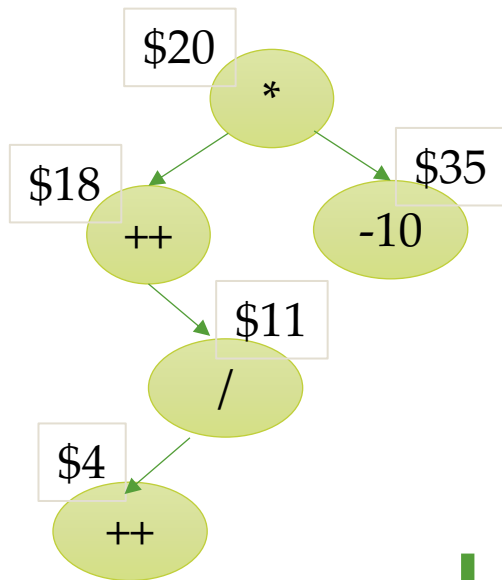
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armaríamos el árbol si la expresión vienen por niveles?  
 Coloco dato. Como no sé si \$4 tendrá cero, uno o dos hijos (depende del token), pido que cuando llegue el momento se los procese.



Pendientes:

\$11 der  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 \$4 izq  
 \$4 der

Por niveles con placeholders:

*	++	-10	?	/	?	?	?	?	++	-10	?	?	?	?	?	?	?	?	?	3.5
---	----	-----	---	---	---	---	---	---	----	-----	---	---	---	---	---	---	---	---	---	-----

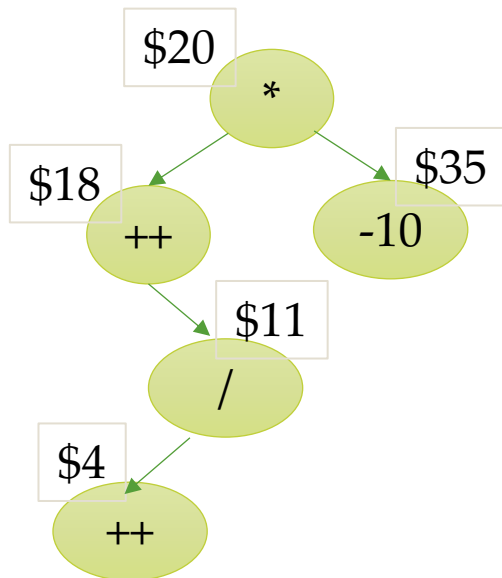
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armariamos el árbol si la expresión vienen por niveles?  
Saco pendiente y leo token



Pendientes:

~~\$11 der~~  
Null consumir  
Null consumir  
Null consumir  
Null consumir  
Null consumir  
Null consumir  
Null consumir  
Null consumir  
\$4 izq  
\$4 der

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

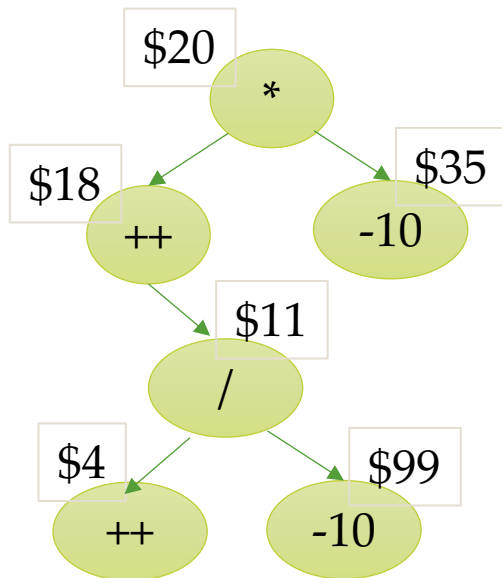
Nivel 1

Nivel 2

Nivel 3

Nivel 4 (posiblemente incompleto)

¿Cómo armaríamos el árbol si la expresión vienen por niveles?  
 Coloco dato. Como no sé si \$99 tendrá cero, uno o dos hijos (depende del token), pido que cuando llegue el momento se los procese.



Pendientes:

Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir  
 Null consumir

\$4 izq

\$4 der

\$99 izq

\$99 der

Por niveles con placeholders:

\* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ? ? ? 3.5

Nivel 1

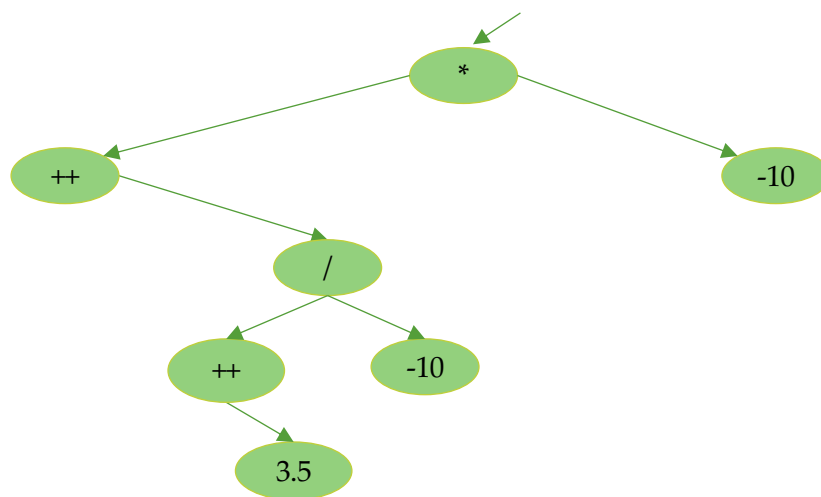
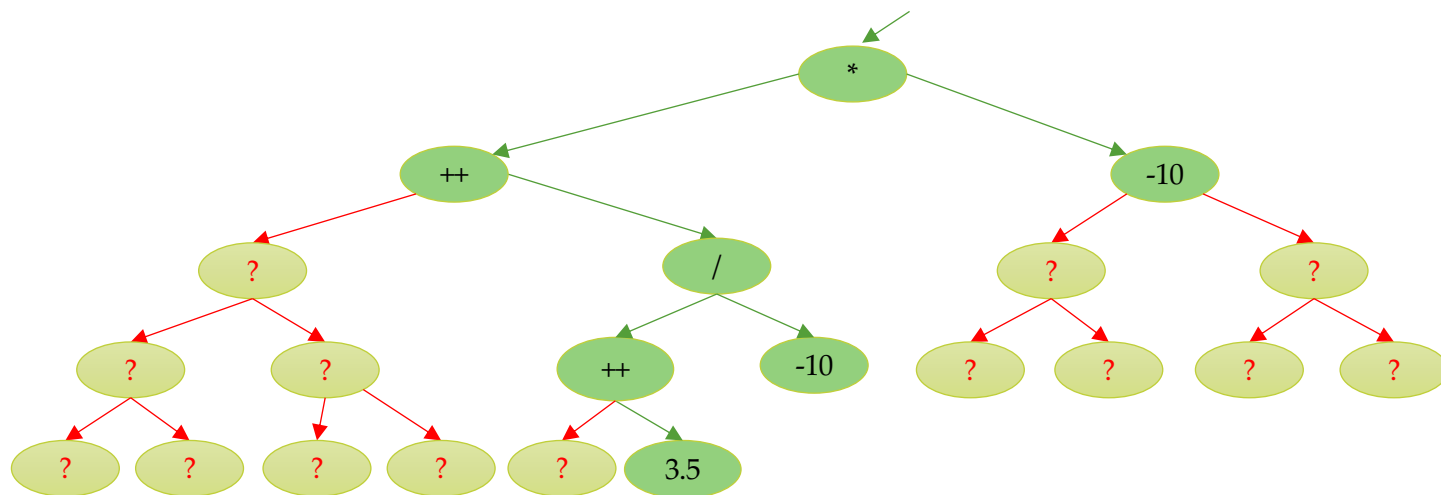
Nivel 2

Nivel 3

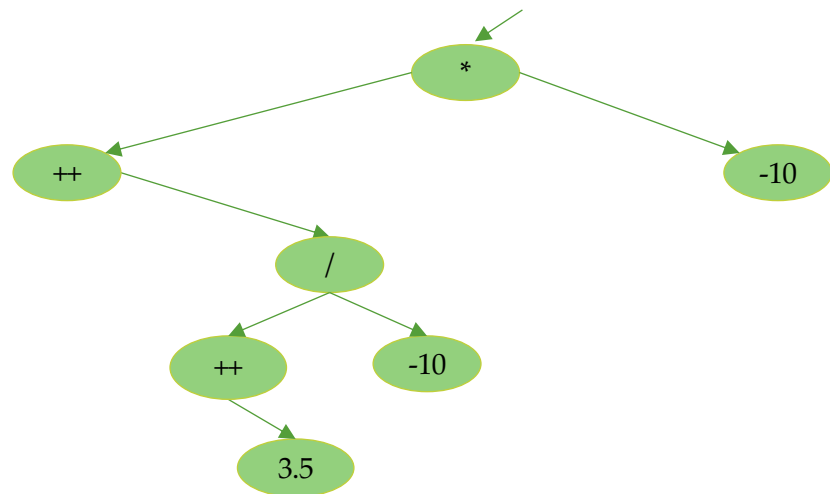
Nivel 4 (posiblemente incompleto)

- 
- Etc etc etc





Este era el árbol:



# TP 5B – Ejer 1

Bajar de Campus BinaryTree  
junto con los .txt en un Proyecto

Probarlo con archivos (en  
carpeta resources)

Datos0\_1

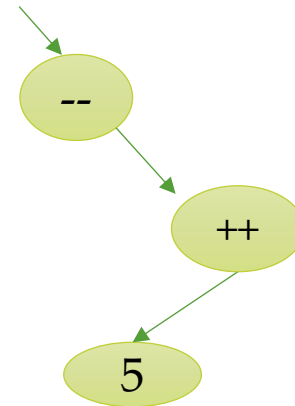
Datos0\_2

Datos0\_3



## Caso de Uso

Ej: -- ? ++ ? ? 5



```
BinaryTree rta = new BinaryTree("data0_1");
```

```
rta.preorder();
```

```
-- ++ 5
```

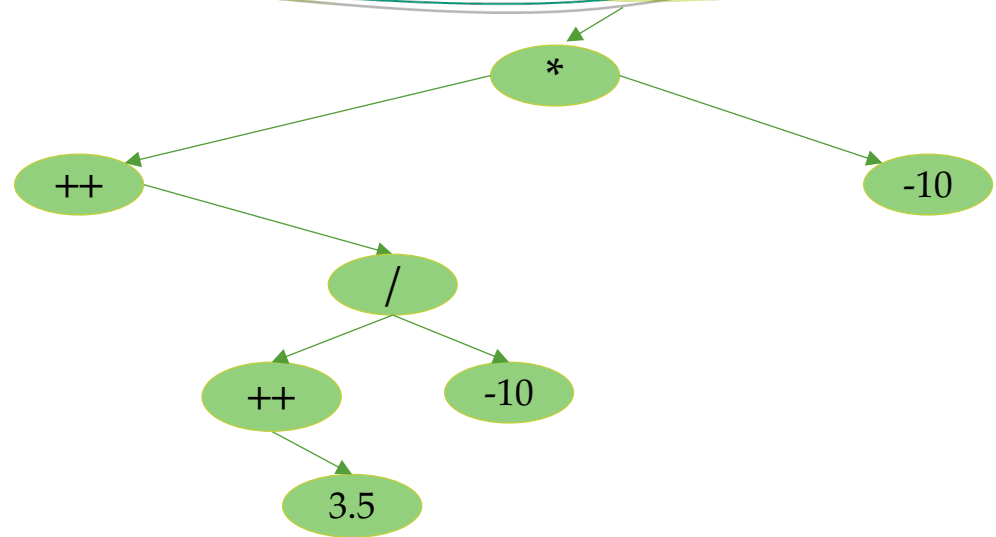
## Caso de Uso

Ej: ?

```
BinaryTree rta = new BinaryTree("data0_2");
```

```
rta.preorder(); //null
```

## Caso de Uso



Ej: \* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ?  
? ? ? 3.5

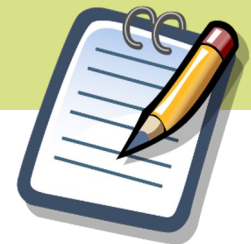
```
BinaryTree rta = new BinaryTree("data0_3");
```

```
rta.preorder();
```

```
* ++ / ++ 3.5 -10 -10
```

# TP 5B – Ejer 2

Agregar la impresion  
`printHierarchy()`

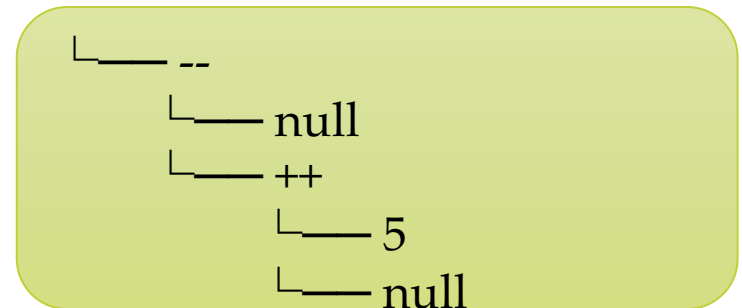
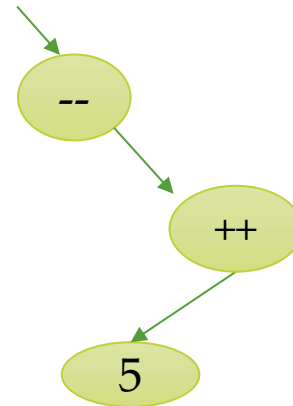


## Caso de Uso

Ej: -- ? ++ ? ? 5

```
BinaryTree rta = new BinaryTree("data0_1");
```

```
rta.printHierarchy();
```





## Caso de Uso

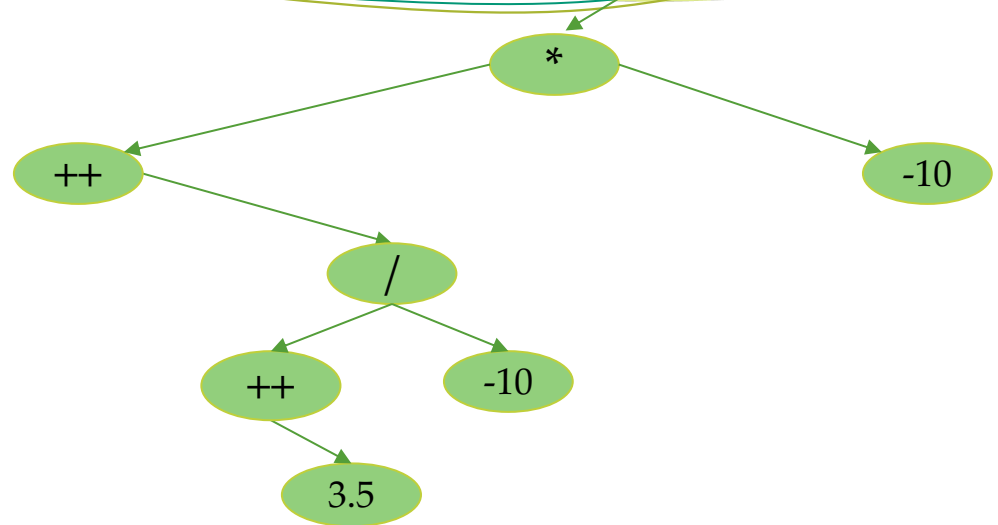
Ej: ?

```
BinaryTree rta = new BinaryTree("data0_2");
```

```
rta.printHierarchy(); //null
```

└── null

## Caso de Uso



Ej: \* ++ -10 ? / ? ? ? ? ++ -10 ? ? ? ? ? ?  
? ? ? 3.5

```
BinaryTree rta = new BinaryTree("data0_3");
```

```
rta.printHierarchy();
```

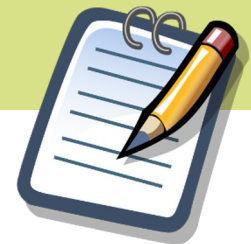
```
└── *
    ├── ++
    │   └── null
    │       └── /
    │           ├── ++
    │               └── null
    │                   └── 3.5
    └── -10
```

## TP 5B – Ejer 3

El comportamiento debe ser el mismo anterior.

Con debugger verificar que el métodos se ejecuta lazy

Cambiar la implementación de BinaryTree para que no reciba un enum como parámetro, sino un método que se evaluará lazy (cuando llegue el momento de saber si el nodo se debe generar como null o no)



# TP 5B – Ejer 3

El comportamiento debe ser  
el mismo anterior.

Con debugger verificar que  
el métodos se ejecuta lazy

Testearlo con  
data\_empty  
data0\_1  
data\_1



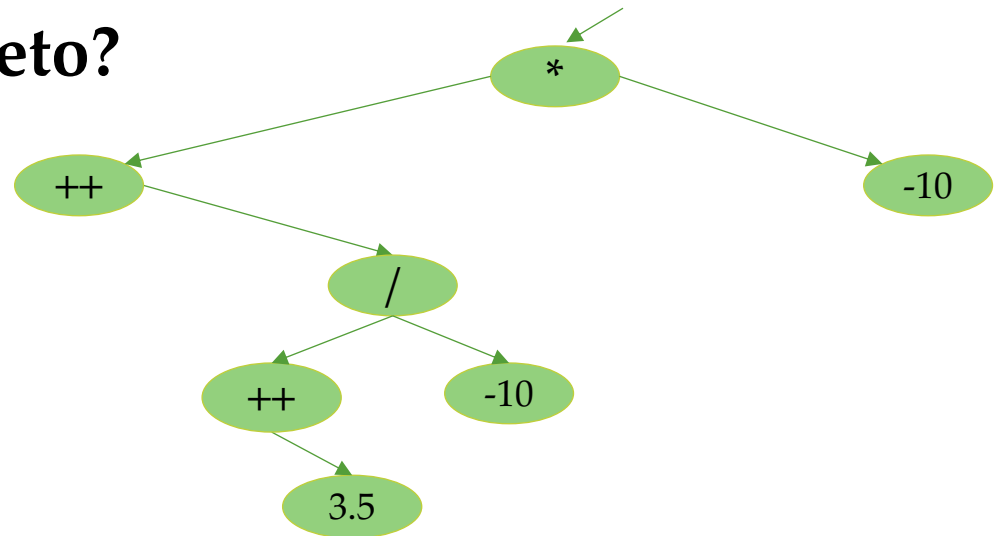
## Definición

Un árbol binario es completo (complete) si todos los niveles, excepto posiblemente el último, tiene todos los nodos posibles y el último nivel tiene los nodos lo más a la izquierda posible.

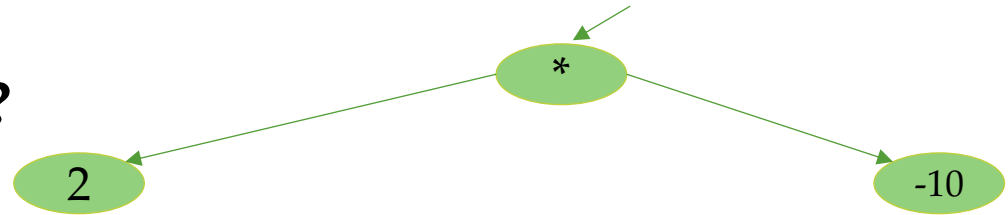
## Definición

Un árbol binario está lleno (full) si todos los niveles, tiene todos los nodos posibles.

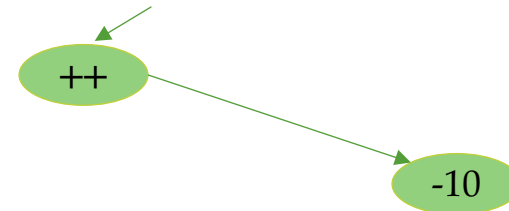
**Este árbol está completo?**  
**Está lleno?**



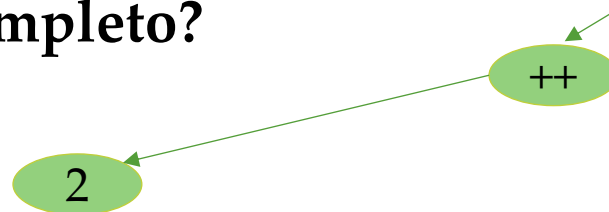
Este árbol está completo?  
Está lleno?




Este árbol está completo?  
Está lleno?



Este árbol está completo?  
Está lleno?





La estrategia de serializar a disco generó un árbol de qué tipo respecto al concepto de completitud/lleno?



## TP 5B – Ejer 4

Agregar el método  
`ToFile("filename")`

Que toma un árbol y lo  
almacena en un archivo con la  
estrategia anterior.

Probarlo leyendo el árbol desde  
`data1` y generar un archivo.  
Verificar que las salidas son las  
mismas excepto quizás espacios  
en blanco.

Como lo podemos verificar?



# TP 5B – Ejer 5

Escribir el método  
boolean equals  
BinaryTree(BinaryTree other)

que detecte equivalencia.

Usarlo para ver si el save()  
estaba OK.



# BinaryTree

## **Altura** (definición)

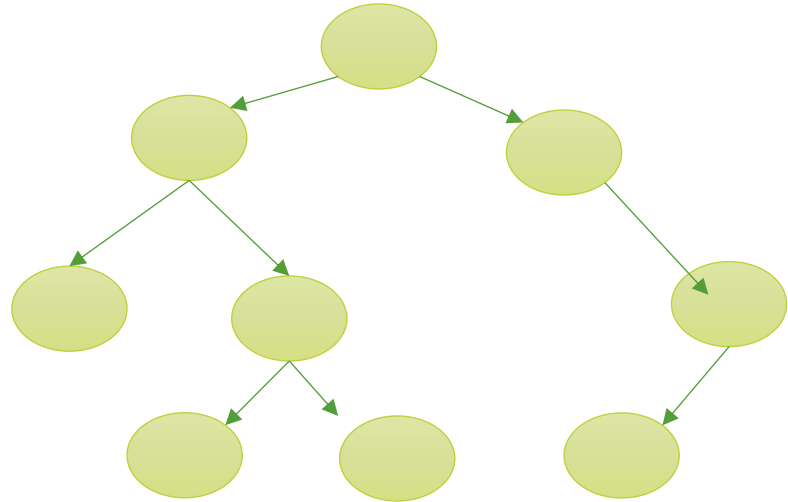
Longitud (cantidad de ejes) del camino más largo desde la raíz hacia las hojas.

Aclaración: un nodo formado solo por una raíz tiene altura 0.

# BinaryTree

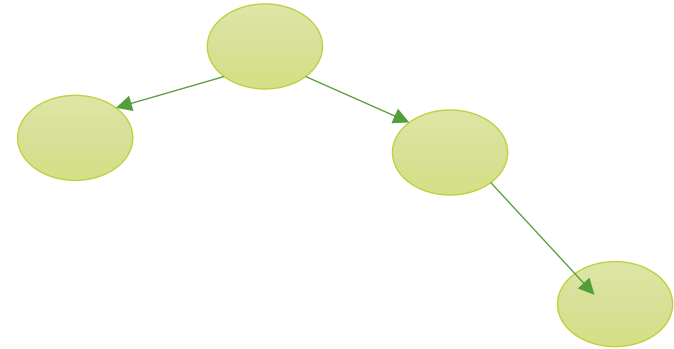
¿Cuál es la altura de este árbol?

Rta 3



¿Cuál es la altura de este árbol?

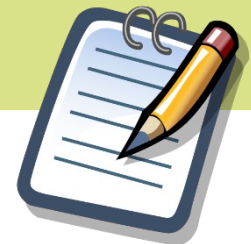
Rta 2



¿Cuál es la altura de un árbol vacío? Poner -1

# TP 5B – Ejer 6

Escribir el método  
`int getHeight()`



# TP 5B – Ejer 7

Escribir la clase  
ParametrizedBinaryTree

Que parametrize el tipo de dato  
de cada Nodo (String, Integer, o  
tipo opaco)

