# Parcialito Proyecto 3 - Tema c

Resolver los ejercicios bstados abajo, en una hora (o menos). El código resultante **no** debe tener memory leaks **ni** accesos (read, write o free) inválidos a la memoria. La entrega es a través de Jaime.

## Ejercicio 1

Modificar el TAD `bst` tal que éste provea una operación nueva cuya signatura es:

```
int bst_height(bst_t bst)
```

Esta operación devuelve la altura del árbol `bst` dado como parámetro. La altura de un árbol es la longitud del camino más largo desde la raíz hasta una hoja (una hoja es aquel nodo cuyo árboles izquierdo y derecho son vacíos). Hay que tener en cuenta que la altura de árbol con un solo nodo es 0, y la de un árbol vacío es -1.

La especificación sería:

$$\begin{aligned} bh <> &= -1 \\ bh <<>, e, <>> &= 0 \\ bh < l, e, r > &= max(bh\ l, bh\ r) \end{aligned}$$

Las PRE y POST de este método nuevo son las siguientes:

**pre** el árbol `bst` es válido.

**post** el resultado es la altura del árbol. Vale que:

`(bst_length(bst) > 0` $\Rightarrow$ `0` $\leq$ `result` $<$ `bst_length(bst))` $\wedge$
`(bst_length(bst) = 0` $\Rightarrow$ `result = -1)`

### Ejemplos

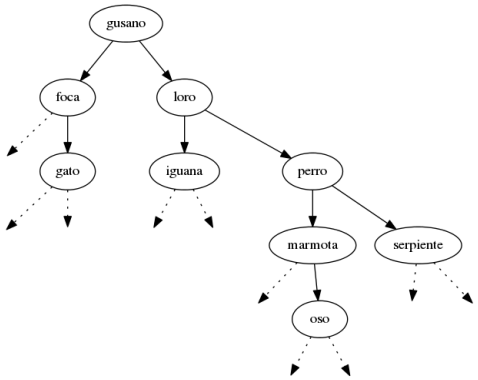


Figura 1: El resultado de llamar a `bst_height` sería 4.

Para la implementación de este método **no** se deben hacer llamadas a otros métodos públicos del TAD bst, pero sí se pueden usar todos los métodos públicos de los otros TADs (es decir, del index, del data, del pair y de la lista).

Notar que para esta implementación van a necesitar escribir una función auxiliar `max` que retorna el máximo entre dos enteros.

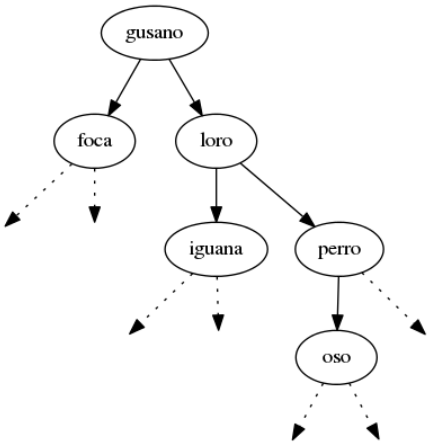


Figura 2: El resultado de llamar a `bst_height` sería 3.

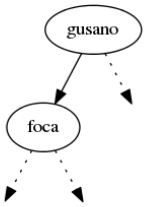


Figura 3: El resultado de llamar a `bst_height` sería 1.


Figura 4: El resultado de llamar a `bst_height` sería 0.

## Ejercicio 2

Crear un archivo `parcialito.c` que provea una función `main` que haga lo siguiente:

- Crear un árbol vacío.
- Imprimir el alto del árbol en su estado actual.
- Agregarle al árbol los siguientes elementos (es **muy** importante poner los 0 del comienzo, cuando corresponde), imprimiendo el alto del bst después de cada llamada (o sea que se imprimirán 13 mensajes):
  - `("18", "dieciocho")`
  - `("10", "diez")`
  - `("50", "cincuenta")`
  - `("05", "cinco")`

- `("15", "quince")`
- `("20", "veinte")`
- `("60", "sesenta")`
- `("02", "dos")`
- `("08", "ocho")`
- `("17", "diecisiete")`
- `("55", "cincuenta y cinco")`
- `("04", "cuatro")`

**Ayuda:** Para crear índices y datos usando cadenas de texto estáticas como las dadas arriba (es decir, strings que no usan memoria dinámica, por ende no hay que liberarlos), pueden hacer lo siguiente:

```
bst = bst_add(bst, index_from_string("03"),
              data_from_string("tres"));
```

Para mostrar el resultado de las llamadas arriba listadas, imprimir **un mensaje por línea**, y usar un fraseo equivalente a:

```
La altura es: 3
```