Parcialito Proyecto 3 - Tema A

Resolver los ejercicios listados 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:

```
unsigned int bst_index_greater_count(bst_t bst, index_t index)
```

Esta operación devuelve la cantidad de nodos del árbol `bst` tal que los índices de esos nodos son mayores estrictos que `index`. La especificación sería:

$$\begin{array}{rcl} bigc <> i & = & 0 \\ bigc < l, e, r > i & | & e \leq i = bigc\ r\ i \\ & | & e > i = 1 + bigc\ l\ i + bigc\ r\ i \end{array}$$

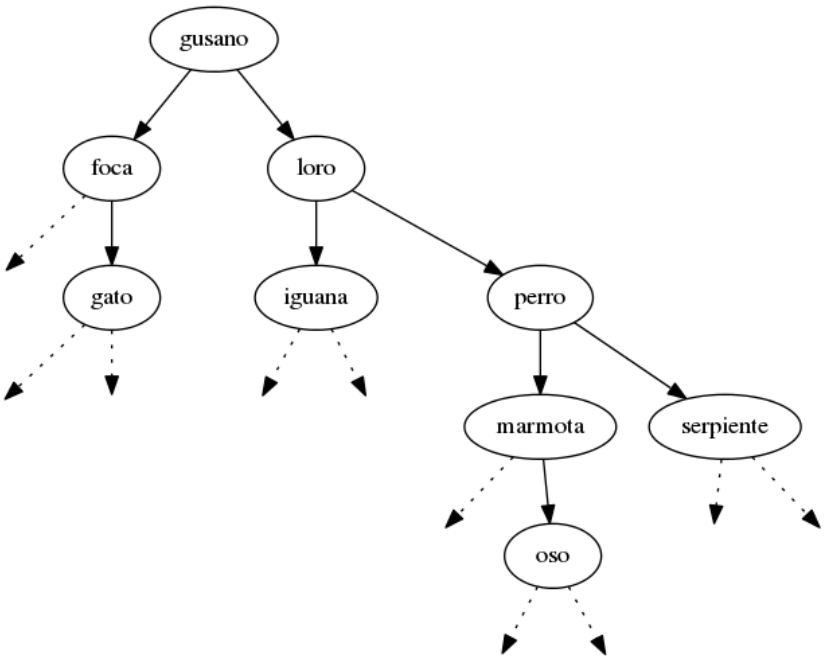
Las PRE y POST de este método nuevo son las siguientes:

pre el árbol `bst` y el índice `index` son válidos. El `index` puede no estar en la lista. El árbol `bst` puede ser vacío.

post el resultado es la cantidad de elementos en el árbol `bst` cuyo índice es mayor que el dado como argumento. Por ende, vale que el resultado es mayor o igual que cero, y menor o igual que el tamaño del árbol. Se deduce que `bst_index_greater_count` retorna 0 para el árbol vacío.

Ejemplo

Supongamos una variable de tipo `bst_t`, de nombre `bst`, que referencia un BST con los siguientes elementos:



Las siguientes llamadas a `bst_index_greater_count` retornan los siguientes resultados:

```
result = bst_index_greater_count(bst, index_from_string("gusano"));  /* result vale 6 */
result = bst_index_greater_count(bst, index_from_string("loro"));    /* result vale 4 */
result = bst_index_greater_count(bst, index_from_string("gato"));    /* result vale 7 */
result = bst_index_greater_count(bst, index_from_string("oso"));     /* result vale 2 */
result = bst_index_greater_count(bst, index_from_string("abeja"));   /* result vale 9*/
result = bst_index_greater_count(bst, index_from_string("zorro"));   /* result vale 0 */
```

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).

Ejercicio 2

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

- Crear un árbol vacío y agregarle los siguientes elementos (es **muy** importante poner los `0` del comienzo, cuando corresponde):
  - `("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"));
  ```

  Llamar al método implementado en el punto 1 tal que se muestre por pantalla la cantidad de nodos cuyo índice es mayor que los siguientes índices:

  - `"18"`
  - `"60"`
  - `"17"`
  - `"04"`
  - `"02"`
  - `"00"`
  - `"99"`

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

  `La cantidad de nodos mayores a 88 es: 0`