# Parcialito Recuperatorio

Resolver los ejercicios listados abajo, en una hora y media (o menos), si no se recupera el proyecto 4, y en dos horas caso contrario. 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 `list` tal que éste provea una operación nueva cuya signatura es:

```
pair_t list_item(list_t list, unsigned int position)
```

Esta operación devuelve una referencia al elemento (par) que está en la lista `list`, en la posición `position`. Las posiciones se enumeran como en C: la primer posición es 0, y la última N-1 (asumiendo que la lista tiene N elementos). La posición pedida puede ser más grande que el tamaño de la lista, en tal caso `list_item` retorna `NULL`.

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

La implementación debe ser iterativa. Las PRE y POST de este método nuevo son las siguientes:

**pre** la lista `list` es un puntero válido.

**post** si la posición pedida es válida, el resultado no es nulo, y es una referencia al par guardado en la lista en esa posición (o sea que **no** es una copia). Si la posición cae "afuera" de la lista, el resultado es nulo.

### Ejemplo

Supongamos una variable de tipo `list_t`, de nombre `list`, que referencia una lista con los siguientes elementos:

```
[("perro", "cuadrupedo que ladra"),
 ("gato", "cuadrupedo que maulla"),
 ("loro", "alumno que habla en clase"),
 ("perro", "el mejor amigo del hombre"),
 ("rinoceronte", "proviene de las palabras griegas rhino, nariz, y kera, cuerno.")]
```

Las siguientes llamadas a `list_item` retornan lo que se muestra abajo:

```
result = list_item(list, 0);
/* result vale ("perro", "cuadrupedo que ladra") */

result = list_item(list, 1);
/* result vale ("gato", "cuadrupedo que maulla") */

result = list_item(list, 4);
/* result value
   ("rinoceronte", "proviene de las palabras griegas rhino, nariz, y kera, cuerno.") */

result = list_item(list, 5);
/* result value NULL */

result = list_item(list_empty(), 0);
/* result vale NULL */
```

## Ejercicio 2

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

1. Crear una lista vacía.
2. Imprimir el resultado de llamar a `list_item` sobre la lista vacía para las posiciones 0 y 10.
3. Agregarle a la lista vacía del punto anterior los siguientes elementos:
   - ("caza", "Accion de cazar")
   - ("guitarra", "Instrumento musical de cuerda")
   - ("casa", "Edificio para habitar")
   - ("cielo", "Esfera aparente azul que rodea la Tierra")
   - ("caza", "Conjunto de animales no domesticados")
   - ("extraordinario", "Fuera del orden o regla natural")
4. Llamar a `list_item` tal que se muestre por pantalla los pares en las posiciones:
   - 0
   - 3
   - 5
   - 6
   - 10

En todos los casos, imprimir un resultado por línea, usando un mensaje análogo a lo que se muestra a continuación:

```
El par en la posición 0 es (caza, Accion de cazar)
```

Si el resultado es nulo, imprimir:

```
El par en la posición 0 es nulo
```

## Ejercicio 3 - Sólo para los que recuperan proyecto 4

Extender el main anterior con la siguiente funcionalidad:

1. Crear un heap vacío con tamaño máximo igual al largo de la lista del punto anterior.
2. Recorrer esta lista, de principio a fin, usando un contador y llamadas a `list_item`. Por cada elemento, crear una arista que tenga:
   - vértice izquierdo con label igual a la posición del elemento
   - vértice derecho con label igual al largo del índice del elemento (ayuda: usar `pair_fst` y `index_length`)
   - peso igual al largo del dato del elemento (ayuda: usar `pair_snd` y `data_length`)

   y agregarla al heap. Al terminar de recorrer la lista, está claro que el heap estará lleno y tendrá tantos elementos como la lista.

   Como referencia, si uno llamara a `heap_dump` del heap debería ver:

```
[0 --15-- 4]
[1 --29-- 8]
[2 --21-- 4]
[3 --40-- 5]
[4 --36-- 4]
[5 --31-- 14]
```

3. Vaciar el heap sacando las aristas una a una. Por cada arista que se saca, imprimir en pantalla:

    - La definición del elemento de la lista que se encuentra en la posición dada por el label del vértice izquierdo de la arista (ayuda: usar `edge_left_vertex`, `vertex_label`, `list_item` y cualquier otro método necesario).

    **OJO** Imprimir una definición por línea.

Lo anterior asegura que se va a imprimir por pantalla todas las "datas" de los elementos de la lista en orden creciente de largo.