Unidad 4: Listas

Solución de los ejercicios 1 y 2, correspondientes a la entrega del 20/3/2020

1. Implementar con control de errores una función no recursiva en C llamada list_tam que devuelva el número de nodos de la lista enlazada proporcionada. Suponga las siguientes estructuras y tipos en C:

```
// En list.h
typedef struct _List List;

// En list.c
typedef struct _Node {
   Element *info;
   struct _Node *next;
};
typedef struct _Node Nodo;

struct _List {
   Node *first;
};
```

Solución:

```
int list_tam(const List *pl) {
   Node *pn;
   int cont=0;

   if(!pl) return -1;   // caso de ERROR

   pn = pl->first;      // nos situamos en el primer nodo de la lista

   // e iteramos sobre todos los nodos de la lista, contándolos
   while (pn != NULL) {
       cont++;
       pn = pn->next;
   }

   // Manera alternativa:
   // for(pn = pl->first; pn !=NULL; pn = pn->next)
   //     cont++;

   return cont;
}
```

2. Implementar recursivamente la función anterior.

Una **función recursiva** es simplemente **una función que se llama a sí misma.** En la presentación del tema de listas en Moodle tenéis un ejemplo con la implementación de la función list_free con ayuda de la función list_free_rec (página 62). Como en esa función, aquí definimos primero una función no recursiva **list_tam** que toma como argumento la **lista** (List *pl) y que usa una función recursiva auxiliar **list_tam_rec** que opera sobre los **nodos** de la lista.

```
int list_tam(const List *pl) {
   if (!pl) return -1;
   return list_tam_rec(pl->first);
}
```

```
int list_tam_rec (Node *pn) {
   if (!pn) return 0;                    // caso base
   return 1 +  list_tam_rec(pn->next);   // llamada recursiva
}
```

La primera llamada a list_tam_rec toma como argumento el primer nodo de la lista, es decir, el nodo pl->first. En sucesivas llamadas, su argumento pn será pl->first->next (el siguiente al primer nodo, es decir, el segundo), pl->first->next->next (el siguiente al segundo, es decir, el tercero), etc. Hasta que eventualmente sea llamada con el último nodo de la lista y luego con su pn->next, que es NULL. En este caso, que es lo que se llama el "caso base de la recursión", retorna 0.

Es decir, la longitud de una lista de 0 nodos es 0. La expresión
1 + list_tam_rec(pn->next),
por otro lado, lo que nos dice es que la longitud de la lista que comienza en el nodo pn es
1 + el tamaño del resto de la lista,
donde "el resto de la lista" es la lista que comienza en pn->next.

El concepto de recursión es fundamental en programación, y es muy importante entenderlo bien porque tiene infinidad de aplicaciones. Tenéis una explicación mucho más detallada de la recursión y de este ejercicio en particular en las Notas sobre recursión que están subidas a Moodle justo bajo este documento.