Funciones y tipos auxiliares

1. Funciones auxiliares

Para la implementación de algunas funciones podrían necesitarse funciones auxiliares.
En `C` no se puede definir una función anidada dentro de otra como se puede hacer en `Pascal`. Por lo tanto la función auxiliar debe definirse fuera de las funciones en las que se va a usar. Además, la declaración debe estar antes de que la función sea usada.
Esto puede llegar a generar un conflicto en el momento del enlazado si en otros archivos también hubiera funciones auxiliares con el mismo encabezado ya que en `C` las funciones de manera predeterminada tienen ámbito global. Para evitar este problema se debe anteponer la palabra reservada `static` a la declaración de la función. Con esto queda como local al archivo. Su ámbito va desde la declaración hasta el fin del archivo.

```
// Declaración de 'auxiliar' usada en 'f' y 'g'.
// Debe preceder a la implementación de las funciones que la usan.
static int auxiliar(int p);
// La implementación se puede hacer junto con la declaración
// o después.

// f y g usan 'auxiliar'
void f() {
  ...
  int a = auxiliar(5);
  ...
}

void g() {
  ...
  int a = auxiliar(8);
  ...
}

// implementación de auxiliar
static int auxiliar(int p) {
   ...

   ...
   return p + ...;
}
```

2. Tipos auxiliares

En cualquier módulo se pueden definir tipos auxiliares cuyo ámbito va desde su definición hasta el fin del archivo. En particular el tipo puede ser un `struct`.

```
  typedef struct {int entero; double real;} TIntDouble;
```

A diferencia de `Pascal`, en `C` un elemento de tipo `struct` se puede pasar como parámetro por valor y ser el resultado de una función:

```
TIntDouble incrementa_duplica(TIntDouble p) {
  p.entero = p.entero + 1;
  p.real = p.real * 2;
```

```
  return p;
}
```

El resultado del siguiente fragmento de código

```
TIntDouble e_r;
e_r.entero = 0;
e_r.real = 1;

TIntDouble res = incrementa_duplica(e_r);
printf("e_r: (%d, %.2f); res: (%d, %.2f)\n", e_r.entero, e_r.real, res.↩
    entero, res.real);
```

es

```
e_r: (0, 1.00); res: (1, 2.00)
```