# Tema B

## Ejercicio 1

Considerar la siguiente asignación múltiple:

```
var x, y, z : Int;
{Pre: x = X, y = Y, z = Z, X ≠ 0}
x, y, z := 2*y + x, z+x, y/x
{Post: x = 2*Y + X, y = Z+X, z = Y/X}
```

Escribir un programa en lenguaje C equivalente usando asignaciones simples teniendo en cuenta:

- Se deben verificar la pre y post condición usando la función `assert()`.
- Los valores iniciales de `x, y, z` deben obtenerse del usuario usando la función `pedirEntero()` definida en el *Proyecto 3*
- Los valores finales de `x, y, z` deben mostrarse por pantalla usando la función `imprimeEntero()` definida en el *Proyecto 3*.

# Ejercicio 2

Programar la función:

```
int multiplo_minimo(int a[], int tam, int k);
```

que dado un arreglo `a[]` con `tam` elementos devuelve el elemento más chico de `a[]` que es múltiplo de `k`. Por ejemplo:

| a[] | tam | k | resultado |
|---|---|---|---|
| [3, 8, **6**, 20, 5] | 5 | 2 | **6** |
| [**3**, 8, 6, 20, 5] | 5 | 3 | **3** |

Si en el arreglo `a[]` no hubiese un elemento múltiplo de `k` la función debe devolver el neutro de la operación `min` para el tipo `int` (usar `<limits.h>`)

Cabe aclarar que `multiplo_minimo` no debe mostrar ningún mensaje por pantalla ni pedir valores al usuario.

En la función `main` se debe solicitar al usuario ingresar un arreglo de longitud `N` (definir a `N` como una constante, el usuario no debe elegir el tamaño del arreglo) y luego se debe pedir el valor `k` (verificar con `assert` que `k≠0` ) y finalmente mostrar el resultado de la función `multiplo_minimo`.

# Ejercicio 3

Hacer un programa que cuente la cantidad de elementos pares (sin el cero), impares y ceros que hay en un arreglo. Para ello programar la siguiente función

```
struct paridad_t contar_pares_impares(int a[], int tam);
```

donde la estructura `struct paridad_t` se define de la siguiente manera:

```
struct paridad_t {
    int n_pares;
    int n_impares;
    int n_ceros;
}
```

La función toma un arreglo `a[]` y su tamaño `tam`, devolviendo una estructura con los tres enteros que respectivamente indican cuántos elementos son pares, cuántos son impares y cuántos son ceros en `a[]`. La función `contar_pares_impares` debe implementarse con un único ciclo y **no debe mostrar mensajes** por pantalla **ni pedir valores al usuario**.

En la función `main` se debe solicitar al usuario ingresar un arreglo de longitud `N` (definir a `N` como una constante, el usuario no debe elegir el tamaño del arreglo) y luego se debe mostrar el resultado de la función por pantalla.

# Ejercicio 4*

Hacer un programa que dado un arreglo de compras de productos calcule el precio total a pagar y la cantidad de kilogramos a llevar. Para ello programar la siguiente función:

```
struct total_t calcular_montos(struct producto_t a[], int tam);
```

donde la estructura `struct producto_t` se define de la siguiente manera:

```
struct producto_t {
    int precio;
    int peso_en_kilos;
};
```

y la estructura `struct total_t` se define como:

```
struct total_t {
    int precio_total;
    int peso_total;
}
```

La función toma un arreglo `a[]` con `tam` elementos de tipo `struct producto_t` y devuelve una estructura con dos números que respectivamente indican el precio a pagar y la cantidad de kilogramos de productos que hay en `a[]`. La función `calcular_montos` debe implementarse con un único ciclo y **no debe mostrar mensajes** por pantalla **ni pedir valores al usuario**.

En la función `main` se debe solicitar al usuario ingresar un arreglo de elementos de tipo `struct producto_t` de longitud `N` (definir a `N` como una constante, el usuario no debe elegir el tamaño del arreglo). Para ello solicitar por cada elemento del arreglo un valor entero y luego otro valor entero. Se puede modificar la función `pedirArreglo()` para facilitar la entrada de datos. Luego se debe mostrar el resultado de la función `calcular_montos` por pantalla.