Tema C

Ejercicio 1

Considerar la siguiente asignación múltiple:

```
var x, y, z: Int;
{Pre: x = X, y = Y, z = Z, Z - Y ≠ 0, X - Y ≠ 0}
x, y := (z - y) / (x - y), z mod x
{Post: x = (Z - Y) / (X - Y), y = Z mod X}
```

Escribir un programa en lenguaje C equivalente usando asignaciones simples. Verificar las pre y post condiciones usando la función `assert()`. Los valores iniciales de **`x`**, **`y`** y **`z`** deben ser ingresados por el usuario y los valores finales se deben mostrar por pantalla.

Ejercicio 2

Vamos a organizar dos números según un booleano. Deben programar la función `organizar()` que tiene el siguiente prototipo:

```
struct organizar_t organizar(bool b, int x, int y) {
    ...
}
```

Y dependiendo del valor de b pone a x en el campo primero o segundo de la estructura organizar_t. Si b es true pone a x en el campo primero y a y en el campo segundo de la estructura `organizar_t`, y si b es false pone a y en el campo primero y a x en el campo segundo.

La estructura `organizar_t` se define como:

```
struct organizar_t {
    int primero;
    int segundo;
};
```

Dentro de la función `organizar()` **no se debe interactuar con el usuario** (no deben pedir valores ni mostrar resultados por pantalla), solo deben realizar los cálculos necesarios para obtener los resultados especificados.

En la función `main()` pedir al usuario tres valores: 2 enteros y 1 booleano e imprimir el resultado de la función `organizar()` pasándole como parámetros los valores ingresados por el usuario.

Ejercicio 3

Dado un arreglo se necesita contar sólo los elementos iguales a un elemento ingresado por el usuario. Deben programar la función:

```
int cantidad(int array[], int tam, int elem) {
    ...
}
```

La función entonces debe contar sólo los elementos iguales a elem. Por ejemplo:

```
cantidad({3, 1, 2, 3, 4}, 5, 3) ↦ 2

cantidad({-1, 2, -3, 4}, 4, -3) ↦ 1

cantidad({12, 1, 21, -6, 45}, 5, 3) ↦ 0

cantidad({}, 0, 5) ↦ 0
```

Asegurar usando `assert()` que el valor de `tam` sea positivo (mayor o igual a cero). La función `cantidad()` **no debe interactuar con el usuario** (no debe pedir valores ni mostrar resultados por pantalla), solo tiene que calcular lo que se especificó y devolver el resultado obtenido.

Programar una función `main()` que solicite un arreglo de tamaño `N` (debe ser una constante definida en el programa) mediante la función `pedirArreglo()` (programada en el *ejercicio 6* del *Proyecto 4*) y un entero que será el elemento a comparar, luego mostrar por pantalla el resultado de `cantidad()` usando como parámetros el arreglo obtenido del usuario, `N y el elemento ingresado por el usuario`.

Ejercicio 4*

Un comerciante para llevar el stock de su negocio tiene un arreglo de productos. Cada producto está descripto en una estructura como la siguiente:

```
struct producto_t {
    int codigo;
    int precio;
    int cantidad;
    };
```

Programar la función `hay_pedido` que dado un codigo de producto, devuelve `true` si hay al menos la cantidad requerida en el stock del comerciante `a[]`.

```
bool hay_cantidad(int codigo, int cantidad_requerida, struct producto_t
a[], int tam) {
    ...
}
```

Además programar la función de entrada de datos:

```
void pedirArreglo(struct producto_t a[], int n_max) {
    ...
}
```

que permite al usuario ingresar los datos de un arreglo de tipo **`struct`** `producto_t`

En la función `main()` deben pedirle al usuario el contenido de un arreglo `a[]` de tamaño constante con elementos del tipo **`struct`** `producto_t`. Luego se le debe pedir un arreglo de tamaño constante con codigos de productos, que se usará en el llamado de `hay_pedido()`. Finalmente se debe mostrar un mensaje indicando si hay stock de los productos solicitados en el arreglo a[].