Parcial 2 - Algoritmos I Taller: Tema H

Ejercicio 1

Considere las siguientes afirmaciones y seleccione la respuesta correcta:

a) Indicar cuál de las comparaciones entre lenguajes imperativos y funcionales es cierta:
   1) La manera de computar de los lenguajes imperativos y funcionales es la misma
   2) En los lenguajes imperativos el resultado de un programa es un estado final al que se llega luego de ejecutar todas las sentencias para lograr el objetivo. A veces también se denomina programación algorítmica. Por el contrario, en los lenguajes funcionales el resultado es una expresión a la cual no se le pueden aplicar más reglas de reducción.
   3) Los lenguajes imperativos tienen variables y los funcionales no.
   4) Ninguna de las anteriores es correcta.

b) Para determinar que un valor es menor o igual a otro en C se usa la expresión:
   1) a => b
   2) a <= b
   3) not (a < b)
   4) a >= b

c) ¿Cuál es la función estándar de C utilizada para leer datos ingresados por pantalla, para almacenarlos en la memoria?
   1) `scanf()`
   2) `log()`
   3) `printf()`
   4) `display()`

d) El comando para poder imprimir el valor de una Variable cada vez que para la ejecución de un programa en GDB es:
   1) print
   2) next
   3) list
   4) Ninguno de los anteriores.

Ejercicio 2

Considere el siguiente código con asignaciones múltiples:

```
var x, y, z : Int;
{Pre: x = X, y = Y, z = Z, X > 0, Z > 0}
if (x ≥ z) →
     x, y, z := y+z, x+z, x+y
□  (x < z)→
    x, y, z := y, z, x
fi
{Pos: (X≥Z ∧ (x=Y+Z ∧ y=X+Z ∧ z=X+Y)) ∨ (X<Z ∧ (x=Y ∧ y=Z ∧ z=X))}
```

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

- Se deben verificar las pre y post condiciones usando la función `assert()`.
- Los valores iniciales de `x, y, z` deben ser ingresados por el usuario y se puede usar la función `pedir_entero` del proyecto 3.
- Los valores finales de `x, y, z` deben mostrarse por pantalla usando la función `imprimir_entero` del proyecto 3.

NOTA: Poner como comentario al menos un ejemplo de ejecución, con los parámetros de entrada y la salida de tu programa (puedes hacer un copiar y pegar de la consola).

Ejercicio 3

Dada la siguiente estructura:

```
struct datos {
    bool es_divisible_por_2;
    int menor_divisible_por_2;
};
```

Programar la función:

```
struct datos hay_divisible(int tam, int a[]);
```

que dado un tamaño de arreglo `tam` y un arreglo `a[]`, devuelve una estructura **`struct datos`**, en el campo `es_divisible_por_2` será **`true`** si en el arreglo `a[]` hay un número que sea divisible por 2 y **`false`** en caso contrario. Pueden asumir que el arreglo tiene al menos 2 elementos (chequear esto con `assert`) .En el campo `menor_divisible_por_2` se deberá retornar el mínimo número divisible por 2 que haya en el arreglo, y INT_MAX en caso de no haber ninguno. La función debe programarse utilizando un solo ciclo.

Por ejemplo:

tam	a[]	resultado variable res	Comentario
3	[7,4,6]	`res.es_divisible_por_2 == true` `res.menor_divisible_por_2 == 4`	En el arreglo **hay** algún número divisible por 2 y **4** es el menor divisible por 2 del arreglo.
3	[9,77,5]	`res.hes_divisible_por_2== false` `res.menor_divisible_por_2 == INT_MAX`	En el arreglo **no hay** ningún número divisible por 2 y **INT_MAX** es el valor que debe devolver en `menor_divisible_por_2`.
4	[1,2,3,4]	`res.es_divisible_por_2 == true` `res.menor_divisible_por_2 == 2`	En el arreglo **hay** algún número divisible por 2 y **2** es el menor divisible por 2 del arreglo.

Cabe aclarar que la función `hay_divisible` 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**.

Finalmente desde la función `main` se debe llamar a la función `hay_divisible y` mostrar el resultado por pantalla.

NOTA: Poner como comentario al menos un ejemplo de ejecución, con los parámetros de entrada y la salida de tu programa (puedes hacer un copiar y pegar de la consola).

Ejercicio 4

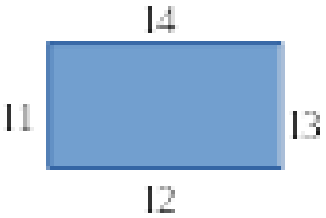
Programar la siguiente función que verificar que tipo de lienzo vas a usar para tu pintura

```
struct tipo_lienzo verificar_lienzo(struct lienzo t);
```

donde las estructuras `lienzo y tipo_lienzo` se definen de la siguiente manera:

```
struct lienzo {
    int l1;
    int l2;
    int l3;
    int l4;
};
```

```
struct tipo_lienzo {
    bool es_figura;
    bool es_paisajes;
    bool ninguno_anteriores;
};
```



La función `verificar_lienzo` toma una `struct lienzo`, y devuelve una `struct tipo_lienzo`, con tres booleanos que respectivamente indican:

- `es_figura` es **`true`** si y sólo si, los cuatro lados l1, l2, l3 y l4 son iguales. Caso contrario es **false**.
- `es_paisajes` es **`true`** si y sólo si, l1 y l3 iguales y l2 y l4 son iguales, pero no son iguales entre sí. Caso contrario es **false**.
- `ninguno_anteriores` es **`true`** si y sólo si, `es_figura` y `es_paisajes` son **false.** Caso contrario es **false**.

En la función `main` se debe solicitar al usuario ingresar los valores de la **`struct`** `lienzo` y luego de llamar a la función `verificar_lienzo` mostrar el resultado por pantalla (los tres booleanos de **`struct`** `tipo_lienzo`).

NOTA: Poner como comentario al menos un ejemplo de ejecución, con los parámetros de entrada y la salida de tu programa (puedes hacer un copiar y pegar de la consola).