Parcial 2 - Algoritmos I Taller: Tema B

Ejercicio 1

Considere las siguientes afirmaciones y seleccione la respuesta correcta:

a) En C podemos decir de los elementos de un arreglo que:
   1) Tienen que ser accedidos en orden secuencial.
   2) Pueden ser accedidos en cualquier orden.
   3) Se puede acceder siempre sólo al primer elemento del arreglo.
   4) Ninguna de las anteriores es correcta.

b) En una estructura en C, podemos decir de sus campos que:
   1) Tienen que ser accedidos en orden secuencial.
   2) Se puede acceder siempre sólo al primer campo de la estructura.
   3) Pueden ser accedidos en cualquier orden.
   4) Ninguna de las anteriores es correcta.

c) En una estructura en C, podemos decir de sus campos que:
   1) Tienen que ser todos del mismo tipo.
   2) Pueden ser todos del mismo tipo.
   3) Tienen que ser todos de distinto tipo.
   4) Ninguna de las anteriores es correcta.

d) Se puede decir de la librería assert.h que:
   1) Se puede usar para controlar Pre y Post condiciones sin alterar la ejecución del programa.
   2) Se puede usar para controlar Pre y Post condiciones abortando el programa.
   3) No se puede usar para controlar predicados.
   4) Ninguna de las anteriores es correcta.

Ejercicio 2

Considere el siguiente código con asignaciones múltiples:

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

Programar la función:

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

que dado un tamaño máximo de arreglo `tam` y un arreglo `a[]`, devuelve una estructura **`struct datos`**, en el campo `esta_ordenado_asc` será **`true`** si el arreglo `a[]` está estrictamente ordenado de manera **ascendente** y **`false`** en caso contrario. Pueden asumir que el arreglo tiene al menos 1 elemento (chequear esto con `assert`). En el campo `maximo` se deberá retornar el máximo del arreglo. La función debe programarse utilizando un solo ciclo.

Por ejemplo:

tam	a[]	resultado variable res	Comentario
3	[2,5,9]	`res.esta_ordenado == true` `res.maximo == 9`	El arreglo **está** ordenado y **9** es el máximo elemento.
3	[2,9,7]	`res.esta_ordenado == false` `res.maximo == 9`	El arreglo **no está** ordenado de manera ascendente y **9** es el máximo elemento.
3	[3,3,4]	`res.esta_ordenado == false` `res.maximo == 4`	El arreglo **no está** ordenado de manera ascendente y **4** es el máximo elemento.

Cabe aclarar que `esta_ordenado_asc` 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 mostrar el resultado de la función `esta_ordenado_asc` 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:

```
struct caract_triangulo averiguar_caract(struct triangulo t);
```

donde las estructuras `triangulo y caract_triangulo` se definen de la siguiente manera:

```
struct triangulo {
    int l1;
    int l2;
    int l3;
};
```

```
struct caract_triangulo {
    bool es_equilatero;
    int  perimetro;
};
```

La función `averiguar_caract` toma una `struct triangulo`, y devuelve una `struct caract_triangulo` con dos campos que respectivamente indican:

- `es_equilatero es` **`true`** si y sólo si, los tres lados l1, l2 y l3 son iguales. Caso contrario es **false**.
- `perimetro es el perímetro del triángulo`.

En la función `main` se debe solicitar al usuario ingresar los valores de la **`struct`** `triangulo` y luego de llamar a la función `averiguar_caract` mostrar el resultado por pantalla (los dos campos de **`struct`** `caract_triangulo`).

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).