Parcial 2 - Algoritmos I Taller: Tema D

Ejercicio 1

Considere las siguientes afirmaciones y seleccione la respuesta correcta:

a) Basado en la siguiente definición, podemos decir,

```
int valor[4];
```

   1) La variable `valor` es igual a 0 hasta su primera asignación.
   2) `valor[4] == valor[0]`.
   3) Los índices del arreglo serán el 0, el 1, el 2 y el 3.
   4) Cuando compilamos obtenemos un `warning`.

b) Cuando escribimos `typedef struct {int ahorro} alcancia;`
   1) El tipo `alcancia` es un sinónimo de `int`.
   2) El tipo `alcancia` es un sinónimo de una estructura que no tiene nombre.
   3) La sentencia está mal escrita porque debe empezar con un #.
   4) Ninguna de las anteriores.

c) *“assert maneja los eventos indeseados o inesperados, para evitar que el programa o sistema se detenga de repente, y sin ellos, las excepciones van a interrumpir la operación normal del programa.”*
   Esta definición es
   1) Correcta. Si el predicado no se cumple, el assert evita que el programa se detenga.
   2) Correcta. Si hacemos algo mal, assert nos tira un warning cuando compilamos
   3) Incorrecta. El manejo de los eventos indeseados los maneja stdlib.h, assert solo sirve para imprimir mensajes, pero no detiene la ejecución.
   4) Ninguna de las anteriores.

d) Que opción describe mejor el comportamiento del siguiente comando,

```
$> gcc hola.c
```

   1) Falla al no tener definido el estándar `c99`
   2) Crea un binario ejecutable llamado `hola`
   3) Crea un binario llamado `a.out`
   4) Ejecuta el código sin crear ningún archivo de salida

Ejercicio 2

Considere el siguiente código con asignaciones múltiples:

```
var x, y, z : Int;
{Pre: x = X, y = Y, z = Z, X > 0}
if (x < y) →
     x, y, z := z, x+2y+3z, x*y
□  (x ≥ y)→
    x, y, z := x+1, 1/x, y / x
fi
{Pos: (X<Y ∧ (x=Z ∧ y=X+2Y+3Z ∧ z=X*Y)) ∨ (X≥Y ∧ (x=X+1 ∧ y=1/X ∧ z=Y/X))}
```

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

- Se deben verificar la pre y la post condición usando la función `assert()`.
- Los valores iniciales de `x, y, z` deben ser ingresados por el usuario.
- 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 particion {
    bool hay_cero;
    int cantidad_de_pares;
    int cantidad_de_impares
};
```

Programar la función:

```
struct particion par_impar(int tam, int a[]);
```

que dado un tamaño de arreglo `tam` y un arreglo `a[]`, devuelve una estructura **`struct particion`**, en el campo `cantidad_de_pares` acumularemos la cantidad de elementos del arreglo que son pares, y `0` en caso de no haber número par. El caso homónimo para números impares, se hará acumulando en `cantidad_de_impares`. Si al menos uno de los valores es igual a cero, entonces la estructura retornada deberá contar en el campo `hay_cero` con valor **`true`**, y **`false`** en caso contrario. El arreglo debe tener al menos 2

elementos, y se debe chequear con `assert`. La función debe programarse utilizando un solo ciclo.

Por ejemplo:

tam	a[]	resultado variable res	Comentario
5	[7,90,4,30,6]	res.hay_cero == false res.cantidad_de_pares == 4 res.cantidad_de_impares == 1	En el arreglo **hay** 4 pares, 1 impar, y no está el cero
2	[0,0]	res.hay_cero == true res.cantidad_de_pares == 0 res.cantidad_de_impares == 0	En el arreglo **no hay** pares ni impares, y **0** es el valor que debe devolver, y true en hay_cero.
6	[0,1,1,1,1,1]	res.hay_cero == true res.cantidad_de_pares == 0 res.cantidad_de_impares == 5	En el arreglo **no hay** pares, pero si impares, asi que **0** es el valor que debe devolver para los pares, y 5 para los impares, y true en hay_cero.

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

En la función `main` se debe

- Definir `N` (el tamaño del arreglo) como una constante. **El usuario no debe elegir el tamaño del arreglo**.
- Solicitar al usuario el ingreso de los `N` valores correspondiente a los elementos del arreglo.

Finalmente desde la función `main` se debe llamar a la función `par_impar` 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

```
struct coordenadas_superpuestas verificar_superposicion(struct
coordenada original, struct coordenada nueva);
```

donde las estructuras **`struct`** `coordenada` y **`struct`** `coordenadas_superpuestas` se definen de la siguiente manera:

```
struct coordenada {
    float ordenada;
    float abscisa;
    float error;
};
```

```
struct coordenadas_superpuestas {
    bool coincide;
    bool contiene;
    bool es_contenida;
};
```

La función `verificar_superposicion` toma dos estructuras de tipo `coordenada`, llamadas `original` y `nueva`, y devuelve una **`struct`** `coordenadas_superpuestas` con tres booleanos que respectivamente indican:

- `coincide` **es `true`** si y sólo si `original.ordenada` es igual a `nueva.ordenada` y `original.abscisa` es igual a `nueva.abscisa`, y `original.error` es igual a `nueva.error`. Caso contrario es **`false`**.
- `contiene` **es `true`** si y sólo si `original.ordenada` es igual a `nueva.ordenada` y `original.abscisa` es igual a `nueva.abscisa`, y `original.error` es mayor a `nueva.error`. Caso contrario es **`false`**.
- `es_contenida` **es `true`** si y sólo si `original.ordenada` es igual a `nueva.ordenada` y `original.abscisa` es igual a `nueva.abscisa`, y `original.error` es menor a `nueva.error`. Caso contrario es **`false`**.

En la función `main` se debe solicitar al usuario ingresar los valores de ambas estructuras **`struct`** `coordenada` y luego de llamar a la función `verificar_superposicion` mostrar el resultado por pantalla (los tres booleanos de **`struct`** `coordenadas_superpuestas`).

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).