Parcial 2 - Algoritmos I Taller: Tema E

Ejercicio 1

Considere las siguientes afirmaciones y seleccione la respuesta correcta:

a) Respecto de la manera de computar en un lenguaje imperativo, podemos decir:
   1) Que es igual a como se computa en Haskell
   2) Que el valor se computa aplicando reglas a una expresión inicial hasta que no se pueden aplicar más reglas, y esa expresión es el resultado.
   3) En los lenguajes imperativos hay expresiones, sentencias y estados. Se ejecutan las sentencias que modifican los estados y las expresiones solo lo consultan.
   4) Ninguna de las anteriores es correcta.

b) ¿Cuál es el operador utilizado para realizar una comparación de igualdad en C?
   1) `=`
   2) `==`
   3) `:=`
   4) `>`

c) ¿Cuál es la función estándar de C utilizada para imprimir en la consola?
   1) `print()`
   2) `log()`
   3) `printf()`
   4) `display()`

d) El comando para poder poner un "break point" en la línea 5 de un programa en GDB es:
   1) display 5
   2) next 5
   3) list 5
   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, Y > X, X > 0}
if (x < y) →
     x, y, z := y, z+y+x, x
□  (x ≥ y)→
    x, y, z := y, z+y, x / y
fi
```

$$\{Pos: (X<Y \land (x=Y \land y=Z+Y+X \land z=X)) \lor (X \geq Y \land (x=Y \land y=Z+Y \land z=X/Y))\}$$

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
- 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

Programar la función:

Dada la siguiente estructura

donde la estructura **`struct`** `datos` se definen de la siguiente manera:

```
struct datos {
    bool esta_ordenado_desc;
    char maximo;
};
```

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

que dado un tamaño máximo de arreglo `tam` y un arreglo `a[]`, devuelve una estructura **`struct datos`**, en el campo `esta_ordenado_desc` será **`true`** si el arreglo `a[]` está ordenado de manera **descendente** y **`false`** en caso contrario. Pueden asumir que el arreglo tiene al menos 2 elementos (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	['c','b','a']	`res.esta_ordenado == true` `res.maximo == 'c'`	El arreglo **está** ordenado y **'c'** es el máximo elemento.
3	['a','b','c']	`res.esta_ordenado == false` `res.maximo == 'c '`	El arreglo **no está** ordenado de manera descendente y **'c'** es el máximo elemento.
3	['a','z','c']	`res.esta_ordenado == false`	El arreglo **no está** ordenado de manera

		`res.maximo == 'z'`	descendente y **'z'** es el máximo elemento.

Cabe aclarar que `esta_ordenado_desc` 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_desc` 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 tipo_triangulo verificar_triangulo(struct triangulo t);
```

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

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

```
struct tipo_triangulo {
    bool es_equilatero;
    bool es_isosceles;
    bool es_escaleno;
};
```

La función `verificar_triangulo` toma una `struct triangulo`, y devuelve una `struct tipo_triangulo` con tres booleanos que respectivamente indican:

- `es_equilatero es true` si y sólo si, los tres lados l1, l2 y l3 son iguales. Caso contrario es **false**.
- `es_isosceles` es `true` si y sólo si, dos de los lados son iguales y uno es distinto que los anteriores. Caso contrario es **false**.

- `es_escaleno` es **`true`** si y sólo si, los 3 lados son distintos. Caso contrario es **false**.

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