Parcial 2 - Algoritmos I Taller: Tema G

Ejercicio 1

Considerar las siguientes afirmaciones y seleccione la respuesta correcta:

a. Utilizando **GDB** para el *debugging*, la opción que me imprime los valores de una expresión en cada acción que realice dentro del debugger es:
    1. `print`
    2. `break`
    3. `step`
    4. `display`
    5. `next`

b. El siguiente fragmento de código:

```C/C++
#include <stdbool.h>
#include <stdio.h>

int main(void) {
    bool v = true;
    printf("%b\n", v);
    return 0;
}
```

    1. Es incorrecto porque `%b` no es un especificador de formato para booleanos.
    2. Imprime un entero representando el valor de verdad del booleano `v`.
    3. Devuelve un error al compilar por usar un formato inválido para `printf`.
    4. El programa no compila porque `void` es un parámetro inválido para la función `main`.
    5. Debe convertirse `v` a entero usando una variable auxiliar antes de imprimirlo por pantalla.

c. Para definir un nuevo tipo en C.
    1. Tengo que usar la cláusula `data`.
    2. Se utiliza la palabra clave `type`.
    3. Debo utilizar `typedef struct`.
    4. Se usa `#define`.
    5. Se tiene que usar la instrucción `typedef`.

d. Al compilar el siguiente programa como se pide en la materia:

```C/C++
int main(void) {
	int n;
```

```
        return 0;
}
```

1. El compilador nos devuelve el siguiente mensaje:

```
Unset
aaaa.c: In function ‘main’:
aaaa.c:2:9: warning: unused variable ‘n’ [-Wunused-variable]
    2 |     int n;
      |         ^
```

2. La siguiente advertencia es mostrada:

```
Unset
aaaa.c: In function ‘main’:
aaaa.c:2:9: warning: ‘n’ is used uninitialized [-Wuninitialized]
    2 |     int n;
      |         ^
```

3. Crea un binario llamado `a.out`.
4. La compilación no genera ningún mensaje de advertencia.
5. Se muestra el siguiente mensaje:

```
Unset
aaaa.c: In function ‘main’:
aaaa.c:3:5: warning: ‘return’ with no value, in function returning non-void
[-Wreturn-type]
    3 |     return;
      |     ^~~~~~
```

Ejercicio 2

Considerar la siguiente código con asignaciones múltiples:

```
Unset
var x, y, z : Int;
{Pre: x = X, y = Y, z = Z, X > 0, X + Y ≠ 0}
if ((x + y) mod 2 = 0) →
    x, y, z := 3 * x, 4 * y, x + y
□ ((x + y) mod 2 ≠ 0) →
    x, y, z := x - x, 6 * y, x - y - 1
fi
```

```
{Pos: y mod 2 = 0, z mod 2 = 0, (x = 3*X ∧ y = 4*Y ∧ z = X+Y) ∨ (x = 3*X ∧ y = 6*Y ∧ z = X-Y-1)}
```

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

Dada la siguiente estructura:

```C/C++
typedef struct {
  int intentos_totales;
  bool esta_autenticado;
  int fallos_consecutivos;
} Autenticacion;
```

programar la función:

```C/C++
Autenticacion autenticar_usuario(char id_usuario, int tam, int passwords[]);
```

que dados, un ID de usuario `id_usuario` de tipo caracter, una cantidad máxima de tamaño de arreglo `tam` y un arreglo de contraseñas `passwords[]` de tipo entero (las contraseñas); devuelva una estructura `Autenticacion`.

La función tiene como objetivo recorrer la lista de *passwords* posibles para el usuario dado, hasta que una de dos condiciones se den (la que ocurra primero):

1. O se llegó a la contraseña correcta para el usuario.
2. O no llegando a dar con la contraseña correcta del usuario, se alcanzó el máximo de 3 intentos fallidos.

Para tal motivo, al terminar la función, la estructura `Autenticacion` deberá contener: en el campo `intentos_totales`, la cantidad de comparaciones realizadas de la lista de contraseñas; el campo `esta_autenticado` será `true` si el usuario logró autenticarse con éxito, `false` en caso contrario; en el campo `fallos_consecutivos` la cantidad de

intentos fallidos hasta terminar, debiendo resetearse a `0` en caso de quedar autenticado el usuario.

Por ejemplo para el usuario de ID `u` y contraseña `123`:

`id`	`tam`	`passwords[]`	`res = autenticar_usuario(id_usuario, tam, passwords)`	Comentario

`'u'`	`5`	`[123, 000, 000, 123, 000]`	`res.intentos_totales == 1`
			`res.esta_autenticado == true`
			`res.fallos_consecutivos == 0`
				El usuario está autenticado luego del primer intento

`'u'`	`5`	`[000, 123, 000, 123, 000]`	`res.intentos_totales == 2`
			`res.esta_autenticado == true`
			`res.fallos_consecutivos == 0`
				El usuario queda autenticado luego del segundo intento, reseteando los intentos fallidos a `0`

`'u'`	`5`	`[000, 000, 000, 123, 000]`	`res.intentos_totales == 3`
			`res.esta_autenticado == false`
			`res.fallos_consecutivos == 3`
				Pues luego de 3 intentos el usuario alcanzó la cantidad máxima de intentos fallidos, queda sin autenticar y ambos contadores en `3`

El ID del usuario con el cual comparar el campo `id_usuario`, la cantidad máxima de intentos, el tamaño máximo del arreglo y la contraseña correcta; deberán ser constantes definidas en el archivo del ejercicio. Se debe pedir al usuario que ingrese la lista de contraseñas a probar.

En la función `main`, se le debe pedir al usuario que ingrese el arreglo de contraseñas con las cuales intentar autenticar al usuario. La misma función `main` deberá luego imprimir:

- En caso de terminar el usuario autenticado, la leyenda: `"¡Autenticación exitosa!"` y el total de intentos realizados.
- En caso de terminar con el usuario sin autenticar, la leyenda: `"Autenticación fallida."` y cuántos intentos fallidos consecutivos hubo.

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

Se establecen los siguientes dos tipos de datos *customizados*. El tipo `Rectangulo`, definido de siguiente manera:

```c
C/C++
typedef struct {
    int ancho;
    int altura;
} Rectangulo;
```

y el tipo `ResultadoRect` definido como:

```C/C++
typedef struct {
    int max_area;
    bool hay_contenido;
} ResultadoRect;
```

Utilizando ambos tipos, definir la función:

```C/C++
ResultadoRect contenido_uno_en_otro(Rectangulo r1, Rectangulo r2);
```

que dados 2 parámetros del tipo `Rectangulo`, `r1` y `r2`, devuelva un resultado de tipo `ResultadoRect` con la siguiente información en sus campos:

- `max_area` debe almacenar el resultado de calcular el área más grande de entre los 2 rectángulos recibidos como argumento.
- `hay_contenido` debe de ser `true` si uno de los dos rectángulos está contenido en el otro, `false` caso contrario.

Para determinar si un rectángulo está contenido o no en otro, el rectángulo contenido debe poseer ambos lados (altura y ancho) menores o iguales al rectángulo que le contiene. Recordemos también que el área de un rectángulo se computa realizando la multiplicación de sus lados.

En la función `main` se debe solicitar al usuario ingresar los valores del tipo `Rectangulo`. Luego de invocar la función `contenido_uno_en_otro`, la misma función `main` es la encargada de mostrar por pantalla los valores guardados en la variable de tipo `ResultadoRect`.

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).*