Final - Algoritmos I Taller

Debés entregar el código de cada ejercicio como archivo adjunto, y pegar el código correspondiente a cada ejercicio abajo de cada pregunta. El archivo debe compilar correctamente y sin warnings.

Para compilar un archivo .c escribir en la terminal en la carpeta donde esta el archivo:
`$> gcc -Wall -Wextra -std=c99 miarchivo.c -o miprograma`
Para ejecutar escribir:
`$> ./miprograma`

Atención: *Los ciclos deben ser implementados sólamente con la instrucción `while`. Además no se permite utilizar instrucciones que rompan la ejecución normal de un programas, como `break` o un `return` ubicado en un lugar que no sea la última línea de la definición de la función. Es decir, se espera que las implementaciones de los programas se asemejan a aquellos programas obtenidos en las derivaciones.*

Ejercicio 1 - Suma de Elementos Impares con Condición

Programar la función

```
int sum_impares(int tam, int a[], int n);
```

que dado un arreglo **a[ ]** de tamaño **tam** y un entero **n** menor o igual que **tam**, devuelve la suma de los primeros **n** elementos que son impares del arreglo. Si existe un **0** entre esos elementos, la función debe retornar **0**. Verificar con **assert** que **n** sea menor o igual a **tam**.

Por ejemplo:

a[]	tam	n	resultado
[3,6,5,8,7]	5	3	**8**
[7,44,8,9,11]	5	1	**7**
[2,3,9,10,0]	5	2	**3**
[1,0,6,3,9]	5	4	**0**

Luego, escribir un programa que solicite el ingreso de un arreglo de enteros y un valor **n**, e imprima por pantalla el resultado de la suma de los primeros **n** elementos impares del arreglo. Si hay un **0**, debe imprimir **0**. El programa debe:

A. Implementar la función `sum_impares`
B. Definir a **N** como una constante.
C. En la función **main**, realizar lo siguiente:
    - Declarar un arreglo de longitud **N**.
    - Solicitar al usuario que ingrese los elementos del arreglo.
    - Pedir al usuario que ingrese un valor **n**.
    - Llamar a la función **sum_impares** para calcular la suma.
    - Imprimir el resultado.

Implementación

Pegar acá el código implementado para este ejercicio. Además el archivo debe ser adjuntado con la entrega en el classroom.

Ejercicio 2 - Estructuras y Tipos Definidos

Programar una estructura, usando **typedef,** llamada **Producto** que contenga los campos:

- **char nombre (que tenga lugar hasta 50 caracteres)**
- **float precio**
- **int cantidad**

Luego, programar la función

```
float costo_iva_consumidor_final(Producto producto, int cant);
```

que recibe un producto, una cantidad a comprar y devuelve el precio del producto por la cantidad de productos a comprar más el 21% del total. En el caso de que la cantidad a comprar sea mayor a la cantidad que existe del producto la función devuelve **-1.**

Luego, escribir un programa que crea un producto y luego, imprima por pantalla el resultados del llamado a la función `costo_iva_consumidor_final`. El programa debe:

A. Implementar la función `costo_iva_consumidor_final`
B. En la función **main**, realizar lo siguiente:
    a. Declarar un **Producto** e inicializarlo con los siguientes valores **nombre:**“oculus3” **precio:** 500000 **cantidad:** 10.
    b. Solicitar al usuario que ingrese cuantos productos quiere comprar.
    c. Llamar a la función `costo_iva_consumidor_final.`
    d. Imprimir el resultado (costo total con iva) si se puede realizar la compra, sino imprimir que no hay suficientes productos en stock.

Implementación

Pegar acá el código implementado para este ejercicio. Además el archivo debe ser adjuntado con la entrega en el classroom.

```
# Ejercicio 2
```