**Ejercitación – Serie M**

**Estructuras, campos de bits, uniones y otros.**

---

Conceptos que se agregan en esta serie: Diferentes maneras de agrupar datos dentro del lenguaje.

Generación de datos compuesta mediante estructuras de datos, utilizando las palabras reservadas del lenguaje **struct** y/o **union**.

Como ya se sabe, está prohibido el uso de variables globales.

---

Ejercicio 1:

1.1.- Definir un tipo de dato (**`struct`**), que permita manejar un trio de valores tipo **`float`** (por ejemplo, un punto en el espacio).

1.2.- Implementar una función que permita el ingreso de este tipo de dato.

1.3.- Mediante la palabra reservada **`typedef`** y la estructura implementada en el punto 2.1, genere el tipo de dato **`punto3d_t`**, que permita manejar puntos en el espacio.

1.4.- Implementar una función que calcule, y retorne, la distancia de un punto con el centro de coordenadas.

1.5.- Implementar una función que calcule, y retorne, la distancia entre dos puntos en el espacio.

Ejercicio 2:

**2.a.**- Para una aplicación de gestión de stock (simplificado), se está trabajando con los siguientes datos.

- Código de producto – número entero de al menos 5 dígitos
- Nombre del producto – texto con máximo 80 caracteres
- Cantidad en existencia – número entero (no debería permitir números negativos)
- Costo del producto – variable que maneja datos con decimales.

Realizar una aplicación que permita ingresar un código e informe:

- Si el elemento existe, informe sus datos relacionados (nombre, cantidad, costo)
- Si el elemento no existe, permita agregar el producto y datos asociados.

Los datos deben ser ingresados en forma ordenada por código de producto (No ordenar el array de datos, sino que el ingreso se debe realizar en forma ordenada).

Debe contar con una función para imprimir el listado de productos entre un rango de códigos de productos, o bien todos los productos.

Tener presente que no puede haber productos con el mismo nombre.

La cantidad máxima de productos está definida en MAX_ELEMENTOS

**2.b.-** Agregar a la aplicación las siguientes funcionalidades:

- Permitir modificar la cantidad y costo de los productos.
- Cada vez que se realiza un cambio, indique el impacto económico de dicho cambio en el stock.
- Agregue una función que me indique la valorización total del stock

Ejercicio 3:

Realizar una función que, utilizando uniones y campos de bits, obtenga los campos característicos de una variable de punto flotante de doble precisión.

La función debe tener la capacidad de devolver estos campos.

Ejercicio 4:

Se posee un sistema el cual está comprometido con el espacio de memoria que utiliza, por lo cual se ha encargado un proceso de reducción del espacio insumido por algunas variables almacenadas en array/s.

La idea es básicamente que variables que hoy ocupan uno o más bytes y requieran solo algunos bits, se los agrupe en un solo bloque de memoria.

Los campos a "comprimir" son:

| Tipo de dato actual | Parámetro | Rango de valores (externos incluidos) | bits a ocupar en la nueva estructura |
|---|---|---|---|
| short | Edad | 18 ~ 65 | 6 |
| short | Antigüedad | 0 ~ 47 | |
| char [4] | Grupo sanguíneo | A+; 0+; B+; AB+;<br>A-; 0-; B-; AB- | |
| int | AptoM1 | Si/No | |
| short | AptoM2 | Si/No | |
| char [40] | Rango | Dirección Ejecutiva<br>Gestión<br>Supervisión<br>Profesionales Especializados<br>Técnicos y Analistas<br>Ejecución y Operación<br>Asistencia y Apoyo<br>Practicantes y Becarios | |
| char | Categoría | A ~ G | |
| char | Sector | A ~ M | |
| int | STN | 1000~1014 | |

Complete la cantidad de bits que requiere para cada parámetro, y defina una estructura de bits que permita almacenar esta información.

En base a lo aquí pedido, se realice:

- Definir la estructura **`datos_t`** que almacene todos los datos antes mencionados.
- Definir una estructura de bit, **`bits_t`**, dimensionada para que no ocupe más de 32 bits, y que tenga capacidad de almacenar todos estos datos.
- Realizar una función que convierta (comprimir) un dato **`datos_t`** en **`bits_t`**
  Si la conversión fue posible, la función debe retornar 0 (cero). En caso contrario, la función debe retornar un valor negativo alusivo al error (el valor debe dar indicios del error ocurrido).
- Realizar una función que convierta (descomprimir) un dato **`bits_t`** en **`datos_t`**
  Si la conversión fue posible, la función debe retornar 0 (cero). En caso contrario, la función debe retornar un valor negativo alusivo al error (el valor debe dar indicios del error ocurrido).
- Realizar una función que reciba un numero de error definido en las funciones anteriores e imprima el significado de dicho error.

---