La práctica consiste en crear un programa que permita extraer unos valores de un archivo .txt y crear con los datos obtenidos un inventario *Tipo Abstracto de Dato* (*TAD*) *Lista Posicional Ordenada,* un catálogo de coches en forma de *diccionario* que contenga *listas* de *tuplas* (pieza, número) y una lista de pedidos que no llega a procesar como un tipo de dato, sino que va procesando cada pedido en simulación.

- ***Instalación y Mecanismo de Ejecución*** :

Para poder ejecutar el programa es necesario disponer de un compilador de Python y el paquete *sys* (para leer los *.txt* de la simulación de la que sacar los datos del catálogo, inventario y pedidos).

El programa se inicia descomprimiendo el fichero *.zip* y ejecutando el archivo *main.py*. Este accede a la ruta donde se encuentra el archivo *.txt* y lo lee. Para su correcta lectura, cada fichero *.txt* debe seguir su propia estructura :

➢ ***Modelos:***

Atributo	**Tipo de Dato**	**Descripción**
Modelo	str	Nombre del modelo al que pertenecen los datos indicados.
Pieza	str	Nombre de la pieza que necesita el modelo.
Número	int	Número de piezas que necesita el modelo.

➢ ***Piezas***

Atributo	**Tipo de Dato**	**Descripción**
Pieza	str	Nombre de la pieza que tenemos en el inventario.
Número	int	Número de piezas que tenemos en el inventario (siempre mayor que 0).

➢ ***Pedidos***

Atributo	**Tipo de Dato**	**Descripción**
Cliente	str	Nombre/Identificador del cliente.
Modelo	str	Nombre del modelo pedido.

[4]*Nicolás García Iglesias – nicolas.garcia.iglesias@udc.es*
Manuel Omil Barreiro – manuel.omil.barreiro@udc.es

Para todos los *.txt* la estructura son los atributos en orden descendente separados únicamente por una coma (,) y cada elemento se escribe en una línea diferente del *.txt* .

Las piezas son leídas por la función *read_parts(path = "piezas.txt")*, los modelos por *read_models(path = "modelos.txt")* y los pedidos por *read_orders(path = "pedidos.txt")* que crean las estructuras de datos necesarias para cada caso.

- ***Fases de desarrollo.***

El programa trata con objetos creados a partir de las clases *Linked Ordered Positional* List y/o *Array Ordered Positional List* (definidas la primera en los archivos *doubly_linked_base.py* y *linked_order_positional_list.py*, y la segunda en *array_ordered_positional_list.py*).

De este modo los modelos se almacenan en un diccionario con estructura :

{ Modelo1 : [(Pieza1, Número1), (Pieza2, Cantidad2) …] ,

Modelo2 : [(Pieza1, Número3), …]…}

Así como las piezas *Linked Order Positional List* o un *Array Ordered Positional List* (programado para ambas opciones):

LOPL/AOPL ([Pieza1, Número1] , [Pieza2, Número2] , …)

Fases de la Simulación (Se ejecutan cada turno)

1) Lee los archivos *piezas.txt*, *modelos.txt* y *orders.txt* con las funciones *read_parts()*, *read_models()* y *read_orders()* respectivamente. Con estas funciones se genera un catálogo de los modelos a la venta y una lista de las piezas disponibles, así como una lista para procesar el control final de la ejecución, atendiendo un pedido por vez.

2) Durante el bucle *while* se llama a la función *procesar_pedidos()*, que comprueba que los modelos de pedidos existan en catálogo y si lo hacen, devuelve el nombre del modelo a ensamblar.

3) La función *ensamblar()* se encarga de la resta y eliminación de las piezas del catálogo a medida que se gasten; siempre y cuando haya suficientes. Para asegurarse de esto, llama a la función *comprobación()* que se encarga de comprobar la disponibilidad de las piezas de un modelo antes de su ensamblado, indicando en caso negativo cuantas faltan y eliminando el modelo del catálogo.

4) Si la resta de las piezas da 0, se llama además a *modelos_dependientes()* que buscará y eliminará del catálogo a todos los modelos que compartan esa pieza agotada.

[4]***Nicolás García Iglesias – nicolas.garcia.iglesias@udc.es***
Manuel Omil Barreiro – manuel.omil.barreiro@udc.es

5) Se generará un mensaje con el resultado de intentar procesar el pedido (dependiendo de la disponibilidad de piezas y de su existencia en el catálogo).

6) Se mostrarán el stock de piezas actualizado tras su ensamblado y el catálogo de pedidos actualizado, ajustado a la disponibilidad de las piezas

---

- ***Errores y Mejoras***

El principal problema de esta práctica ha sido intentar aplicar el uso de un *TAD* y la implementación de ambas *Listas Posicionales Ordenad*as para guardar datos; más específicamente en la manera de guardar pares de datos dentro de las listas y el uso de funciones específicas de sus clases.

Un posible cambio sería alterar la función *modelos_dependientes()* para que elimine del catálogo a los coches que necesiten más piezas de las disponibles (ahora solo se eliminan aquellos coches que necesitan cierta pieza una vez esa pieza llega a 0). Puesto que causa confusión a la hora de hacer un pedido. (Si las piezas no llegan a 0, es necesario que alguien encargue un modelo antes de que sea eliminado del catálogo)

El programa funciona sin tener que hacer modificaciones bien sólo con *AOPLs*, sólo con *LOPLs* o bien con ambos. (Se puede probar comentando una de las líneas del *import* al inicio)
