JUSTIFICACIÓN DE LA PRÁCTICA

- Comerciar -

- CONTEXTO DE LA FUNCIÓN

La función de carácter iterativo pertenece a la clase **Ciudad** y se llama desde el programa principal, usando la instrucción *comerciar* o *co*. Esta función no consta de ninguna función auxiliar asociada a la clase (u otras clases), y tiene como parámetro de entrada una ciudad y un inventario que nunca será modificado.

```
void comerciar(Ciudad& c, const Inventario& inv);
```

Esta función tiene como objetivo general intercambiar productos entre el parámetro implícito y otra ciudad, dándose todos los productos posibles recorriendo el inventario del parámetro implícito. La función basa su funcionamiento únicamente en modificar los inventarios de las ciudades, sin retornar nada al acabar de ejecutarse (función de tipo *void*).

```
/** @brief Comerciar
    Una ciudad le dara a la otra todos los productos que le sobren hasta alcanzar
    si es posible los que la otra necesite, y viceversa
    \pre c es una ciudad válida e inicializada, inv tiene al menos 1 elemento
    \post Se ha realizado el intercambio de productos entre el p.i. y c
*/
void comerciar(Ciudad& c, const Inventario& inv);
```

- PRECONDICIÓN:

La ciudad que forma parte del parámetro de entrada existe en la cuenca del río donde se trabaja en la función principal, con un identificador único asociado. Además, el inventario no está vacío, al menos tiene 1 elemento

- POSTCONDICIÓN:

Según el intercambio de productos entre el parámetro implícito y la Ciudad c se han modificado los inventarios de las ciudades añadiendo o quitando unidades de ciertos productos, pero nunca añadiendo productos al catálogo que no estaban antes de comerciar ni eliminando productos definitivamente del catálogo. Si no comercian las dos ciudades permanecen igual que antes.

En esta función el invariante representa a la garantía de que en esta función siempre se recorrerán productos que existan en el parámetro implícito, que en este caso sería:

it != prods_ciudad.end()

Al empezar esta función se entra en un bucle iniciando el iterator it = prods_ciudad.begin() (representa el primer producto de la ciudad). Si el parámetro implícito no tiene elementos, se cumple que prods_ciudad.begin() == prods_ciudad.end(), es decir, it == prods_ciudad.end(), y por tanto se sale del bucle y directamente no itera y acaba la función. Como toda la funcionalidad de comerciar está en el bucle, directamente no se comercia.

```
void Ciudad::comerciar(Ciudad& c, const Inventario& inv) {
    for (auto it = prods_ciudad.begin(); it != prods_ciudad.end(); ++it) {
        // Asignamos un id al producto que estamos tratando
        int id_prod = it->first;
        // Asignamos si a cada ciudad le falta o le sobra el producto
        int unidades1 = prods_ciudad[id_prod].first - prods_ciudad[id_prod].second;
        int unidades2 = c.consultar_reserva(id_prod) - c.consultar_faltante(id_prod);
        Producto p = inv.devolver_producto(id_prod);

        if (unidades1 > 0 and unidades2 < 0) {
            // Multiplicamos por -1 para obtener el valor absoluto de las unidades
            unidades2 = -1 * unidades2;
            int venta;
            // Si necesita más de las que puede dar, da todas las que puede, sino da todas las que necesita
            if (unidades1 < unidades2) venta = unidades1;
            else venta = unidades2;

            peso_total -= p.consultar_peso() * venta;
            volumen_total -= p.consultar_vol() * venta;
            prods_ciudad[id_prod].first -= venta;

            c.anadir_prod_reserva(p, venta);
        }
        else if (unidades1 < 0 and unidades2 > 0) {
            // Repetimos el proceso anterior pero cambiando unidades1 por unidades2
            unidades1 = -1 * unidades1;
            int compra;
            if (unidades2 < unidades1) compra = unidades2;
            else compra = unidades1;

            peso_total += p.consultar_peso() * compra;
            volumen_total += p.consultar_vol() * compra;
            prods_ciudad[id_prod].first += compra;

            c.quitar_prod_reserva(p, compra);
        }
    }
}
```

Si existe algún producto en el inventario de la ciudad, se hace mínimo una iteración (1 por cada producto distinto del inventario).

Para comerciar hay dos condiciones: Si al parámetro implícito le sobran unidades y a la ciudad c le faltan o si al parámetro implícito le faltan unidades y a la ciudad c le sobran. Para representarlo, se restan las unidades que tiene cada ciudad en reserva menos las que necesita, que dan los productos que les sobran. Si este número es negativo (unidades1 < 0, unidades2 < 0) significa que la ciudad tiene menos productos de los que necesita, y por tanto le faltan productos. Por otro lado, si este número es positivo quiere decir que le sobran unidades. Si da 0, quiere decir que ni le faltan ni le sobran o que la ciudad no tiene ese producto en su inventario, y por tanto no se comercia.

Una vez entrado a la condición cumplida, se multiplica por -1 el parámetro negativo para hacer su valor absoluto, en unidades. Como no se pueden comprar ni vender más unidades de las que hay disponibles, se escoge el parámetro que en valor absoluto sea menor.
Para realizar la compraventa, se incorpora o retira el nuevo peso y volumen de lo comprado o vendido, para actualizar correctamente las ciudades. Se usan los métodos de la clase ciudad *quitar_prod_reserva(const Producto& p, int unidades)* para quitar unidades de un producto p y

anadir_prod_reserva(const Producto& p, int unidades) para realizar la acción contraria a la anterior, añadir unidades de un producto p al inventario de la ciudad correspondiente.
Pasando eso en cada bucle y cubriendo todas las posibilidades, al llegar al final del inventario finaliza el bucle y por tanto finaliza la función, habiendo comerciado correctamente si se diera el caso.

JUSTIFICACIÓN DE LA PRÁCTICA

Hacer viaje

- CONTEXTO DE LA FUNCIÓN

dfsdfds