Justificación de comerciar:

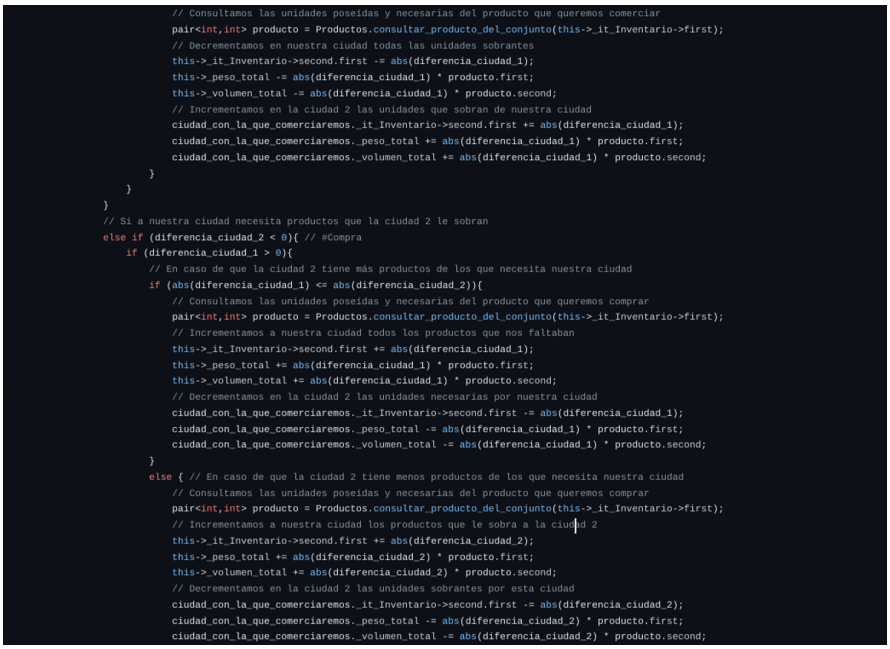
CÓDIGO:

ESPECIFICACIÓN:

```
        /**
         * @brief Realiza comercio con otra ciudad.
         *
         * @param Productos Conjunto de productos disponibles.
         * @param ciudad_con_la_que_comerciaremos Referencia a la ciudad con la que se comerciará.
         *
         * @pre El inventario de la ciudad puede estar vacío o no y debe haber otra ciudad con la que comerciar.
         * @post Se realiza el comercio entre las dos ciudades.
         */
        void comerciar_c(Cjt_Productos& Productos, Ciudad& ciudad_con_la_que_comerciaremos);
```

IMPLEMENTACIÓN:

```
void Ciudad::comerciar_c(Cjt_Productos& Productos, Ciudad& ciudad_con_la_que_comerciaremos){
    // Creamos un "iterator" que apunta al primer elemento del inventario
    this->_it_Inventario = this->_Inventario.begin();
    // Recordemos el inventario
    while (this->_it_Inventario != this->_Inventario.end())
    {
        // Miramos si el producto se encuentra dentro del inventario contrario
        ciudad_con_la_que_comerciaremos._it_Inventario = ciudad_con_la_que_comerciaremos._Inventario.find(this->_it_Inventario->first);
        // En caso afirmativo
        if (ciudad_con_la_que_comerciaremos._it_Inventario != ciudad_con_la_que_comerciaremos._Inventario.end()){
            // Calculamos la diferencia de productos de uno y el otro
            int diferencia_ciudad_1 = this->_it_Inventario->second.second - this->_it_Inventario->second.first;
            int diferencia_ciudad_2 = ciudad_con_la_que_comerciaremos._it_Inventario->second.second - ciudad_con_la_que_comerciaremos._it_Inventario->second.first;
            // En caso de que una ciudad necesite productos que a la otra le sobran
            if (diferencia_ciudad_1 * diferencia_ciudad_2 < 0){
                // Si la ciudad 2 necesita productos que nuestra ciudad le sobran
                if (diferencia_ciudad_1 < 0){ // #Venta
                    if (diferencia_ciudad_2 > 0){
                        // En caso de que tenemos más productos de los que necesita la ciudad 2
                        if (abs(diferencia_ciudad_1) >= abs(diferencia_ciudad_2)){
                            // Consultamos las unidades poseídas y necesarias del producto que queremos comerciar
                            pair<int,int> producto = Productos.consultar_producto_del_conjunto(this->_it_Inventario->first);
                            // Decrementamos en nuestra ciudad las unidades que necesita la ciudad 2
                            this->_it_Inventario->second.first -= abs(diferencia_ciudad_2);
                            this->_peso_total -= abs(diferencia_ciudad_2) * producto.first;
                            this->_volumen_total -= abs(diferencia_ciudad_2) * producto.second;
                            // Incrementamos en la ciudad 2 las unidades que otorgamos la nuestra ciudad
                            ciudad_con_la_que_comerciaremos._it_Inventario->second.first += abs(diferencia_ciudad_2);
                            ciudad_con_la_que_comerciaremos._peso_total += abs(diferencia_ciudad_2) * producto.first;
                            ciudad_con_la_que_comerciaremos._volumen_total += abs(diferencia_ciudad_2) * producto.second;
                        }
                        else { // En caso de que tenemos menos productos de los que necesita la ciudad 2
```



```
                    // Consultamos las unidades poseidas y necesarias del producto que queremos comerciar
                    pair<int,int> producto = Productos.consultar_producto_del_conjunto(this->_it_Inventario->first);
                    // Decrementamos en nuestra ciudad todas las unidades sobrantes
                    this->_it_Inventario->second.first -= abs(diferencia_ciudad_1);
                    this->_peso_total -= abs(diferencia_ciudad_1) * producto.first;
                    this->_volumen_total -= abs(diferencia_ciudad_1) * producto.second;
                    // Incrementamos en la ciudad 2 las unidades que sobran de nuestra ciudad
                    ciudad_con_la_que_comerciaremos._it_Inventario->second.first += abs(diferencia_ciudad_1);
                    ciudad_con_la_que_comerciaremos._peso_total += abs(diferencia_ciudad_1) * producto.first;
                    ciudad_con_la_que_comerciaremos._volumen_total += abs(diferencia_ciudad_1) * producto.second;
                }
            }
        }
        // Si a nuestra ciudad necesita productos que la ciudad 2 le sobran
        else if (diferencia_ciudad_2 < 0){ // #Compra
            if (diferencia_ciudad_1 > 0){
                // En caso de que la ciudad 2 tiene más productos de los que necesita nuestra ciudad
                if (abs(diferencia_ciudad_1) <= abs(diferencia_ciudad_2)){
                    // Consultamos las unidades poseidas y necesarias del producto que queremos comprar
                    pair<int,int> producto = Productos.consultar_producto_del_conjunto(this->_it_Inventario->first);
                    // Incrementamos a nuestra ciudad todos los productos que nos faltaban
                    this->_it_Inventario->second.first += abs(diferencia_ciudad_1);
                    this->_peso_total += abs(diferencia_ciudad_1) * producto.first;
                    this->_volumen_total += abs(diferencia_ciudad_1) * producto.second;
                    // Decrementamos en la ciudad 2 las unidades necesarias por nuestra ciudad
                    ciudad_con_la_que_comerciaremos._it_Inventario->second.first -= abs(diferencia_ciudad_1);
                    ciudad_con_la_que_comerciaremos._peso_total -= abs(diferencia_ciudad_1) * producto.first;
                    ciudad_con_la_que_comerciaremos._volumen_total -= abs(diferencia_ciudad_1) * producto.second;
                }
                else { // En caso de que la ciudad 2 tiene menos productos de los que necesita nuestra ciudad
                    // Consultamos las unidades poseidas y necesarias del producto que queremos comprar
                    pair<int,int> producto = Productos.consultar_producto_del_conjunto(this->_it_Inventario->first);
                    // Incrementamos a nuestra ciudad los productos que le sobra a la ciudad 2
                    this->_it_Inventario->second.first += abs(diferencia_ciudad_2);
                    this->_peso_total += abs(diferencia_ciudad_2) * producto.first;
                    this->_volumen_total += abs(diferencia_ciudad_2) * producto.second;
                    // Decrementamos en la ciudad 2 las unidades sobrantes por esta ciudad
                    ciudad_con_la_que_comerciaremos._it_Inventario->second.first -= abs(diferencia_ciudad_2);
                    ciudad_con_la_que_comerciaremos._peso_total -= abs(diferencia_ciudad_2) * producto.first;
                    ciudad_con_la_que_comerciaremos._volumen_total -= abs(diferencia_ciudad_2) * producto.second;
```

```
                    }
                }
            }
        }
        // Consultamos el siguiente producto
        _it_Inventario++;
    }
}
```

Paso 1: Inicialización:

1) Condición Inicial:

> Al comienzo del bucle while, this→_it_Inventario apunta al primer elemento del inventario de la ciudad (this). Esto significa que estamos considerando el primer producto de la ciudad para el comercio. En este punto, no se han realizado operaciones de comercio, por lo que la condición inicial es verdadera: no se ha intercambiado ningún producto. El producto en cuestión puede o no ser comerciado.

Paso 2: Mantenimiento:

1) Proceso durante cada iteración del bucle while:

> **1.1) Verificación de la presencia del producto:**
>
> > Se verifica si el producto actual en el inventario de this está presente en el inventario de ciudad_con_la_que_comerciaremos utilizando find.
> > Si el producto se encuentra en ambas ciudades, se procede a los cálculos de intercambio.

> **1.2) Cálculo y Ajuste de Intercambio:**
>
> > Se calculan las diferencias entre las unidades deseadas y poseídas de ese producto en ambas ciudades.
> > Dependiendo de las diferencias calculadas, se realizan intercambios de productos entre las dos ciudades, ajustando las cantidades y las métricas de peso y volumen en consecuencia.

> **1.3) Consistencia del Inventario:**
>
> > Cada iteración del bucle asegura que se revisa y se ajusta (si es necesario) la cantidad de un producto específico.

> > La lógica de ajuste garantiza que después de cada iteración, las cantidades de ese producto entre las dos ciudades reflejan un comercio justo basado en sus necesidades y excesos. Al avanzar al siguiente producto (_it_Inventario++), el bucle mantiene su invariante: todos los productos ya procesados están en un estado consistente y correcto.

Paso 3: Invariante + Condición de Terminación:

1) Condición de Terminación:

> La condición de terminación del bucle while es this->_it_Inventario != this→_Inventario.end(). El bucle se ejecuta para cada producto en el inventario de la ciudad this. Una vez que todos los productos han sido procesados, el iterador alcanza end(), lo que detiene el bucle.
>
> **this→Inventario.begin() <= _it_Inventario <= Inventario.end()** **(por def)**

2) Invariante del Bucle:

> La invariante del bucle es que al comienzo de cada iteración, todos los productos procesados hasta ese momento están en un estado consistente y correcto en ambos inventarios, según las reglas de comercio.

3) Estado Final:

> Al final del bucle, todos los productos del inventario de this habrán sido procesados. La invariante junto con la condición de terminación implica que todos los productos en ambos inventarios estarán en un estado final correcto. Como hemos recorrido todo el inventario de this, también hemos modificado el inventario de ciudad_con_la_que_comerciaremos, ya que si las dos ciudades coinciden con el mismo producto, al comerciar, los dos inventarios son modificados simultáneamente.

---

4) Paso 4: Número Finito de Iteraciones:

> La expresión **this→_it_Inventario++** asegura que en cada iteración se avanza al siguiente producto en el inventario de this. Dado que _Inventario tiene un número finito de productos, el número de iteraciones del bucle while también es finito. Cada iteración procesa exactamente un producto y avanza el iterador (**this→_it_Inventario++**), garantizando que el bucle terminará después de **k** iteraciones, donde **k** es el número de productos en el inventario de nuestra ciudad (this).

Conclusión:

> El método **Ciudad::comerciar_c** es correcto