El objetivo de este documento es la justificación de la función iterativa comerciar:

```
void Ciudad::comerciar(Ciudad& city2, vector<Producto>& lista_productos) {
   map<int, amount_products>::iterator it1 = this->InfoProductos.begin();
   map<int, amount_products>::iterator it2 = city2.InfoProductos.begin();

   while(it1 != this->InfoProductos.end() and it2 != city2.InfoProductos.end()) {
      int id_prod1 = it1->first;
      int id_prod2 = it2->first;

      if(id_prod1 == id_prod2) {
         int cantidad1 = this->exceso(id_prod1);
         int cantidad2 = city2.exceso(id_prod2);

         if(cantidad1 < 0 and cantidad2 > 0) {
      //if 'cantidad' is negative, then the city needs products. If 'cantidad' is positive, then the city has an
excess. 'cantidad' is the excess that the city has.

            int cantidad = min(abs(cantidad1), cantidad2);

            this->adquisicion(id_prod1, cantidad, lista_productos[id_prod1 - 1]);
            city2.reduccion(id_prod1, cantidad, lista_productos[id_prod1 - 1]);
         }
         else if(cantidad1 > 0 and cantidad2 < 0) {

            int cantidad = min(abs(cantidad2), cantidad1);

            this->reduccion(id_prod1, cantidad, lista_productos[id_prod1 - 1]);
            city2.adquisicion(id_prod1, cantidad, lista_productos[id_prod1 - 1]);
         }

         ++it1;
         ++it2;
      }
      else if(id_prod1 < id_prod2) {
         ++it1;
      }
      else {
         ++it2;
      }
   }
}
```

INVARIANTE: it1 <= InfoProductos.size() and it2 <= InfoProductos.size(), tal que it1 es el iterador que apunta al inventario de la primera ciudad e it2 apunta al inventario de la segunda ciudad.
it->first es el identificador del producto e it->second es el struct 'amount_products' que contiene la información sobre el producto.

PRECONDICIÓN: Ciudad1 (parámetro implícito) y Ciudad2 existen. (Esta precondición está hecha en el documento program.cc que llama esta función).
Sin tener en cuenta la comprobación previa que se realiza en el program.cc, la precondición sería: TRUE.

POSTCONDICIÓN: Si las ciudades comparten productos en común, entonces habrá intercambio, solo si una ciudad quiere comprar dicho producto y el otro quiere venderlo.

Si este se da el caso, entonces los inventarios de las ciudades en cuestión se modificarán.

INICIALITZACIONS: Inicialmente, ambas ciudades contienen inventarios en forma de mapa, pero no hemos comprobado ningún elemento del mapa, por lo tanto, inicializaremos el iterador it a inventario.begin().

Ambos mapas están ordenados de manera creciente según el identificador del producto (int). Si el mapa está vacío, el iterador va a apuntar a inventario.end(), de manera que no entra dentro del bucle.

Si el mapa no está vacío, entonces va a ser más pequeño que la mida del inventario, así cumpliéndose el invariante.

¿PRECONDICIÓN => INVARIANTE?

Como la precondición técnicamente es TRUE, por lo tanto, se cumplirá el invariante.

CONDICIÓN DE SALIDA (B): Se puede salir del bucle por dos razones: la primera, si el it1 llega a inventario.end(), y la segunda si it2 llega a inventario.end().

En el momento en que una de estas condiciones se cumplan, se saldrá del bucle, dado que es una indicación que una ciudad ya no le quedan productos con quienes comerciar.

CUERPO DEL BUCLE: Primero se comprobará que los identificadores de los productos a quienes apuntan it1 e it2 respectivamente son los mismos, dado que solo se puede comerciar con un mismo producto. Entonces, habrá 3 casos:

1. Caso 1:
   Los identificadores son iguales; it1->first == it2->first.
   Esto nos indica que puede haber comercio entre las ciudades. Seguidamente, es necesario comprobar que una ciudad quiere comprar dicho producto y el otro lo quiere vender.

   a. Caso 1.1:
      Una ciudad tiene exceso de producto y la otra deficiencia. Entonces se dará lugar a un intercambio entre las dos ciudades, así cambiando los respectivos inventarios.

   b. Caso 1.2:
      O ambas ciudades quieren comprar o ambas ciudades quieren vender. Entonces, no se dará lugar un intercambio.

   Al final de este caso, haya o no intercambio, será necesario aumentar ambos iteradores por una unidad.

2. Caso 2:
   Un identificador es más pequeño que el otro: it1->first < it2->first.
   Como los inventarios son mapas y estos están ordenados de orden creciente según it->first, será necesario solo aumentar el iterador it1, dado que ++it1->first > it1->first, y podría darse el caso que ++it1->first == it2->first.
   Esto nos llevaría al primer caso (Caso 1).

3. Caso 3:
   El otro identificador es más pequeño: it1->first > it2->first.
   Igual que el caso anterior, como los mapas están ordenados, será necesario aumentar el iterador it2: it2->first < ++it2->first =? it1->first.
   Si esta condición final se da, entonces volveríamos al Caso 1.

INVARIANTE AND NOT B => POST?

it1 <= InfoProductos.size() and it2 <= InfoProductos.size() and it1 == InfoProductos.end() and it2 == InfoProductos.end() entonces se sale del bucle. Si durante la iteración del bucle ha habido intercambio de productos, entonces se modificarán los inventarios de las ciudades, así cumpliéndose el POST.

Si no ha habido intercambio de productos, entonces no se han modificado los inventarios de las ciudades, así cumpliéndose el POST.

¿ES CIERTO INVARIANTE AL FINAL DE LA FUNCIÓN?

Cuando se llega al final de la función, los iteradores nos quedan de la siguiente manera:

it1 == InfoProductos.end() or it2 == InfoProductos.end()

Si uno apunta al final del inventario, entonces it == InfoProductos.size() e it' < InfoProductos.size(), así cumpliéndose el invariante.

FUNCIÓN DE HITO: La función de hito es la siguiente:

$$f(it) = n(size) - it + 1$$