# Homework 3

**Task 1** (50 points)**:** Implement template utility functions (including variadic templates) and a generic sortable collection class in C++, demonstrating exception handling.

**Requirements:**

1. **Template Utility Functions (Two Namespaces):**

   **a. Namespace `Task_1a`:**

   Implement the following **non-variadic** template functions:

   - `max(a, b)`: Takes two arguments `a` and `b` of the same type `T` and returns the larger.
   - `min(a, b)`: Takes two arguments `a` and `b` of the same type `T` and returns the smaller.
   - `swap(a, b)`: Takes two arguments `a` and `b` of the same type `T` **by reference** and swaps their values.

   **b. Namespace `Task_1b`:**

   Implement `max` and `min` using **variadic templates**. These functions should accept any number of arguments (at least one) of the same type `T` and return the maximum or minimum value among them. *(Set compiler with flag C++11)*

2. **Sortable Collection Class Template:**
   - Create a class template `SortableCollection<T>`.
   - Store elements of type `T` internally (e.g., using `std::vector`).
   - Implement the following member functions:
     - `add(element)`: Adds an element of type `T` to the collection.
     - `getSize()`: Returns the number of elements currently in the collection.
     - `print()`: Prints all elements of the collection.

- `getElement(index)`: Returns the element at the specified `index`. This method should throw a `std::out_of_range` exception if the `index` is invalid (e.g., greater than or equal to the number of elements).
- `sort()`: Sorts the elements currently in the collection in ascending order using a simple algorithm (like Bubble Sort or Selection Sort). **This method MUST use the non-variadic template `Task_1a::swap` function from Requirement 1a.**

3. **Test Cases:** Finish `task_1.cpp` and check `output_1.txt` as sample output

**Task 2** (50 points): Write a C++ program to process hardware product orders using separate files for product information, customer details, and order items. Read product prices from `products.csv`, customer types and names from `customers.csv`, and order items from `orders.csv`. Calculate the final price for each order item based on the customer's type (read from `customers.csv`) and promotional rules, and a standard VAT. Write the results to a comma-separated `result.csv` file.

**Input Files:**

All input files are comma-separated value (CSV) files:

1. **`products.csv`:**
   - Header: `ProductID,ProductName,UnitPrice`
   - Data: `ProductID` (string), `ProductName` (string), `UnitPrice` (floating-point).
2. **`customers.csv`:**
   - Header: `CustomerID,CustomerName,CustomerType`
   - Data: `CustomerID` (string), `CustomerName` (string), `CustomerType` (char: 'A', 'B', or 'C').
3. **`orders.csv`:**
   - Header: `OrderID,CustomerID,ProductID,Quantity,YearsLoyal`
   - Data: `OrderID` (string), `CustomerID` (string), `ProductID` (string), `Quantity` (integer), `YearsLoyal` (integer, may be empty for customers of types A and C).

**Processing Logic:**

Your program should:

1. Read `products.csv` and store `UnitPrice` lookup by `ProductID`.
2. Read `customers.csv` and store `CustomerName` and `CustomerType` lookup by `CustomerID`.
3. Read `orders.csv` line by line. For each order item:
   - Look up the `UnitPrice` using the `ProductID`.
   - Look up the `CustomerName` and `CustomerType` using the CustomerID.
   - Get the `Quantity` and `YearsLoyal` from the order line.
   - Calculate the **Base Amount:** `Quantity * UnitPrice`.
   - Apply the pricing logic based on the retrieved `CustomerType` and YearsLoyal, considering a 10% VAT (0.10) applied *after* any discounts:

     **Type A:** Ordinary customers:

     ```
     Amount = Quantity * Unit price + VAT (10%)
     ```

     **Type B:** Loyal customers:

     ```
     Promotion% =  Min(20% No. of years of loyalty * 2%)
     ```

     (maximum of promotion % is 20%)

     ```
     Amount = (Quantity * Unit price) * (100% – Promotion%) + VAT (10%)
     ```

     **Type C:** Special customers:

     ```
     Amount = (Quantity * Unit price) * 30% + VAT (10%)
     ```

**Output File:**

Write the results to a comma-separated file named result.csv.

- Header: `CustomerID,CustomerName,PriceUnit,Quantity,Amount,Discount,Total`
- **Column Definitions:**

- `CustomerID`: From order.
- `CustomerName`: From customer lookup.
- `PriceUnit`: Original unit price.
- `Quantity`: From order.
- `Amount`: The calculated Base Amount.
- `Discount`: The calculated monetary discount value.
- `Total`: The final calculated total amount including VAT.

- Format all monetary values (`PriceUnit, Amount, Discount, Total`) to two decimal places.

**Test Cases:** Finish `task_2.cpp`. Check `products.csv, customers.csv` and `orders.csv` as sample input and `output_2.csv` as sample output