Project – Milestone 3

Assembly Line – Stations and Line Manager – version 1.2

In this project, you are to code a simulation of an assembly line in three separate milestones.

LEARNING OUTCOMES

Upon successful completion of this project, you will have demonstrated the abilities to

- design and code a composition
- work with vector and queue containers from the Standard Template Library
- work with class variables and functions
- parse a string into tokens
- report and handle exceptions
- move objects from one container to another

SUBMISSION POLICY

Each milestone is to be submitted by its scheduled date. If you cannot complete a milestone by its scheduled date, ask your instructor for an extension. As a general rule, your instructor at their discretion will grant you one extension only for the entire project. A late submission of a milestone with no extension or a submission with a second or third extension will attract a late penalty. The late penalty is 20% of the project mark for each late submission.

In order to be eligible for a credit in this course, you need to submit a workable solution for this project. If you submit all the milestones except the last and have a sufficiently high grade to pass the course with a grade of 0 for the project, your instructor may enter an incomplete (INC) grade for you, in which case you will be required to submit a workable solution before the start of the next semester. If you do not submit a workable solution by the required date, you will fail the course.

Milestone Submission Dates:

1. Milestone 1 – Inventory Item Sets and Project Utilities – July 10 23:59
2. Milestone 2 – Customer Orders – July 19 23:59
3. **Milestone 3 – Stations and Line Manager – August 2 23:59**

PROJECT OVERVIEW

The project simulates an assembly line that fills customer orders from inventory. Each customer order consists of a list of items that need to be filled. The line consists of a set of stations. Each station holds an inventory of items for filling customer orders and a queue of orders to be filled. Each station fills the next order in the queue if that order requests its item and that item is still in stock. A line manager moves the customer orders from station to station until all orders have been processed. Any station that has used all of the items in stock cannot fill any more orders. Orders become incomplete due to a lack of inventory at one or more stations. At the end of all processing, the line manager lists the completed orders and the orders that are incomplete. The figure below illustrates the classes that constitute the simulator and the process of filling orders as they move along the pipeline.

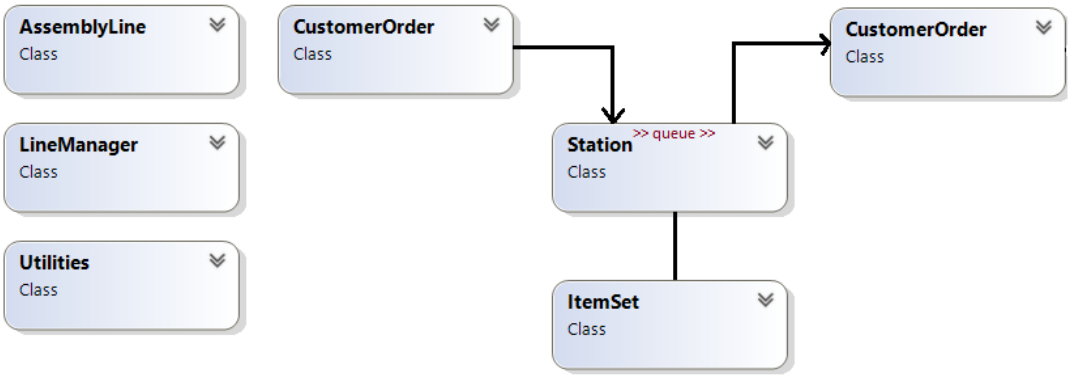


Figure 1 – Simulator Classes

SPECIFICATIONS

Milestone 3 builds the stations of an assembly line, configures the line and builds the line manager. This milestone consists of seven modules:

- **project** (supplied)
- **AssemblyLine** (supplied)
- **Station**
- **LineManager**
- **CustomerOrder** (from Milestone 2)
- **Utilities** (from Milestone 1)
- **ItemSet** (from Milestone 1)

Enclose all your source code within the **sict namespace** and include the necessary guards in each header file. The output from your executable running Visual Studio with the following command line argument should look like

```
Command Line : C:\Users\...\Debug\MS3.exe Inventory.txt CustomerOrders.txt
AssemblyLine.txt

Assembly Line Configuration and Order Processing
================================================

Items in Stock
--------------
CPU          [123456] Quantity 5   Description: Central Processing Unit
Memory       [654321] Quantity 10  Description: Basic Flash Memory
GPU          [456789] Quantity 2   Description: Graphics Processing Unit
SSD          [987654] Quantity 5   Description: Solid State Drive
Power Supply [147852] Quantity 20  Description: Basic AC Power Supply

For Manual Validation: Station 1
 getName(): CPU
 getSerialNumber(): 123456
 getQuantity(): 5
 getName(): CPU
 getSerialNumber(): 123457
 getQuantity(): 4

Customer Orders
---------------
Elliott C.   [Gaming PC]
             CPU
             Memory
             GPU
             GPU
             GPU
             SSD
             Power Supply
Chris S.     [Laptop]
             CPU
             Memory
             SSD
             Power Supply
Mary-Lynn M. [Desktop PC]
             CPU
             Memory
             Power Supply
Chris T.     [Micro Controller]
             GPU
             GPU
             Power Supply
             SSD

Assembly Line Configuration
---------------------------
CPU --> GPU
Memory --> SSD
GPU --> Memory
SSD --> END OF LINE
Power Supply --> CPU

For Manual Validation:
 Power Supply --> CPU
 CPU --> GPU
```

```
 GPU --> Memory
 Memory --> SSD
 SSD --> END OF LINE

Start Processing Customer Orders
--------------------------------
 Filled Elliott C. [Gaming PC][Power Supply][147852]
 --> Elliott C. [Gaming PC] moved from Power Supply to CPU
 Filled Elliott C. [Gaming PC][CPU][123457]
 Filled Chris S. [Laptop][Power Supply][147853]
 --> Elliott C. [Gaming PC] moved from CPU to GPU
 --> Chris S. [Laptop] moved from Power Supply to CPU
 Filled Chris S. [Laptop][CPU][123458]
 Filled Elliott C. [Gaming PC][GPU][456789]
 Filled Elliott C. [Gaming PC][GPU][456790]
 Unable to fill Elliott C. [Gaming PC][GPU][0] out of stock
 Filled Mary-Lynn M. [Desktop PC][Power Supply][147854]
 --> Chris S. [Laptop] moved from CPU to GPU
 --> Elliott C. [Gaming PC] moved from GPU to Memory
 --> Mary-Lynn M. [Desktop PC] moved from Power Supply to CPU
 Filled Mary-Lynn M. [Desktop PC][CPU][123459]
 Filled Elliott C. [Gaming PC][Memory][654321]
 Filled Chris T. [Micro Controller][Power Supply][147855]
 --> Mary-Lynn M. [Desktop PC] moved from CPU to GPU
 --> Elliott C. [Gaming PC] moved from Memory to SSD
 --> Chris S. [Laptop] moved from GPU to Memory
 --> Chris T. [Micro Controller] moved from Power Supply to CPU
 Filled Chris S. [Laptop][Memory][654322]
 Filled Elliott C. [Gaming PC][SSD][987654]
 --> Chris T. [Micro Controller] moved from CPU to GPU
 --> Chris S. [Laptop] moved from Memory to SSD
 --> Mary-Lynn M. [Desktop PC] moved from GPU to Memory
 --> Elliott C. [Gaming PC] moved from SSD to Incomplete Set
 Filled Mary-Lynn M. [Desktop PC][Memory][654323]
 Unable to fill Chris T. [Micro Controller][GPU][0] out of stock
 Unable to fill Chris T. [Micro Controller][GPU][0] out of stock
 Filled Chris S. [Laptop][SSD][987655]
 --> Mary-Lynn M. [Desktop PC] moved from Memory to SSD
 --> Chris T. [Micro Controller] moved from GPU to Memory
 --> Chris S. [Laptop] moved from SSD to Completed Set
 --> Chris T. [Micro Controller] moved from Memory to SSD
 --> Mary-Lynn M. [Desktop PC] moved from SSD to Completed Set
 Filled Chris T. [Micro Controller][SSD][987656]
 --> Chris T. [Micro Controller] moved from SSD to Incomplete Set

Results of Processing Customer Orders
-------------------------------------
COMPLETED ORDERS
Chris S.     [Laptop]
             [123458] CPU - FILLED
             [654322] Memory - FILLED
             [987655] SSD - FILLED
             [147853] Power Supply - FILLED
Mary-Lynn M. [Desktop PC]
             [123459] CPU - FILLED
             [654323] Memory - FILLED
             [147854] Power Supply - FILLED
```

```
INCOMPLETE ORDERS
Elliott C.    [Gaming PC]
              [123457] CPU - FILLED
              [654321] Memory - FILLED
              [456789] GPU - FILLED
              [456790] GPU - FILLED
              [0] GPU - MISSING
              [987654] SSD - FILLED
              [147852] Power Supply - FILLED
Chris T.      [Micro Controller]
              [0] GPU - MISSING
              [0] GPU - MISSING
              [147855] Power Supply - FILLED
              [987656] SSD - FILLED

Assembly Line Configuration and Processing Complete
```

The input for testing your solution is stored in three files: **Inventory.txt** (from Milestone 1), **CustomerOrders.txt** (from Milestone 2) and **AssemblyLine.txt**. The names of these files are specified on the command line as shown in red above. The records in the **AssemblyLine.txt** file for the test problem are

```
Power Supply|CPU
CPU|GPU
GPU|Memory
Memory|SSD
SSD
```

Each record consists of 2 fields delimited by a prescribed char ('|') or a single field. The field pairs identify the connection between one station and the next station along the assembly line:

- Name of the station
- Name of the next station

The single field record identifies the last station on the line.

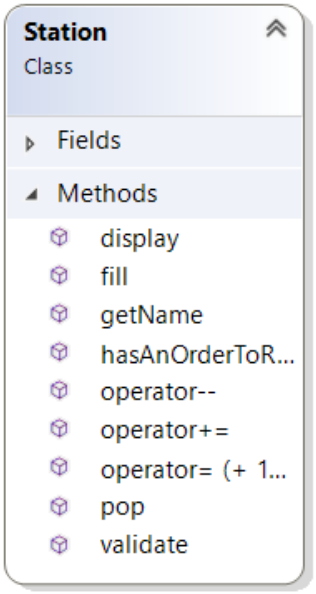
Station Module

The Station module contains all the functionality for filling customer orders with items. Each station that has an order can fill one request at a time for an item from that station. An order can be incomplete due to insufficient items in stock to cover its requests.

Design and code a class named **Station** for managing a set of identical items (with different serial numbers) and processing a queue of customer orders. Each station object is unique and hence neither copyable, moveable, copy-assignable nor move-assignable. Each **Station** contains a **CustomerOrder** queue and an **ItemSet** sub-object. A **Station** object fills a customer request

from the items in its **ItemSet** sub-object. The name of a **Station** object is the name of its **ItemSet** sub_object.

Your class design includes the following public member functions:



- A one-argument constructor that receives a reference to an unmodifiable string and passes that reference to the **ItemSet** sub-object of the current object.
- **`void display(std::ostream& os) const`** – a query that receives a reference to an **`std::ostream`** object **os** and displays the data for its **ItemSet** on **os**.
- **`void fill(std::ostream& os)`** – a modifier that fills the last order in the customer order queue, if there is one. If the queue is empty, this function does nothing.
- **`const std::string& getName() const`** – a forwarding query that returns a reference to the name of the **ItemSet** sub-object.
- **`bool hasAnOrderToRelease() const`** – a query that returns the release state of the current object. This function returns true if the station has filled the item request(s) for the customer order at the front of the queue or the station has no items left; otherwise, it returns false. If there are no orders in the queue, this function returns false.
- **`Station& operator--()`** – a modifier that decrements the number of items in the **ItemSet**, increments the serial number for the next item, and returns a reference to the current object.
- **`Station& operator+=(CustomerOrder&& order)`** – a modifier that receives an rvalue reference to a customer order and moves that order to the back of the station's queue and returns a reference to the current object.
- **`bool pop(CustomerOrder& ready)`** – a modifier that receives an lvalue reference to a customer order, removes the order at the front of the queue and moves it to the calling function through the parameter list. This function returns true if the station filled its part of the order; false otherwise. If there are no orders in the queue, this function returns false.
- **`void validate(std::ostream& os) const`** – a query that reports the state of the ItemSet object in the following format:

```
getName(): ITEM
getSerialNumber(): SERIAL NUMBER
getQuantity(): NUMBER_OF_ITEMS_LEFT
```

A detailed example of the formatting is shown in the output above.

Line Manager Module

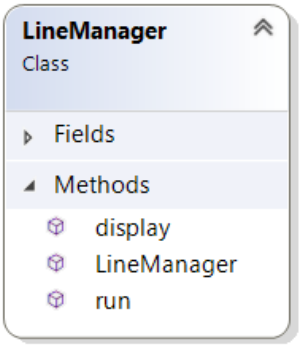
The Line Manager module contains all the functionality for processing customer orders across the entire assembly line. The line manager moves orders along the assembly line one step at a time. At each step, each station fills one order. The manager moves orders that are ready from station to station. Once an order has reached the end of the line, it is either completed or is incomplete. An order can be incomplete due to insufficient items in stock to cover its requests.

Design and code a class named **LineManager** for managing all the stations on the assembly line and processing a vector of customer orders.

Your class design includes the following public member functions:

- A five-argument constructor that receives
  - a reference to an **std::vector** of **Station** addresses,
  - a reference to an **std::vector** of **size_t** objects – each object contains the index in the vector of **Station** addresses that is the index the next station in the assembly line; for example, the vector {2, 3, 1, 5, 0} represents the following set of **Station** connections:
    - station 0 -> station 2
    - station 1 -> station 3
    - station 2 -> station 1
    - station 3 -> station 5
    - station 4 -> station 0
    - for an assembly line 4 -> 0 -> 2 -> 1 -> 3 -> 5
  - a reference to an **std::vector** of **CustomerOrder** objects,
  - the index of the starting station on the assembly line (4 in the example above) and
  - a reference to an **std::ostream** object for displaying output.

  This constructor moves the customer orders to the front of a queue holding the orders waiting to be filled and determines the index of the last station on the line.



- **void display(std::ostream& os) const** – a query that receives a reference to an **std::ostream** object **os** and displays the completed and incomplete orders at the end of the line.
- **bool run(std::ostream& os)** – a modifier that receives a reference to an **std::ostream** object. If there is a customer order on the back of the queue of orders waiting to be filled, this function moves it to the starting station on the line. This function then executes one fill step of the assembly process at each station on the line, by filling the customer order at each station with one item from that station if requested. Once this filling step is done

at each station, this function checks if there is a customer order to be released at each station on the line in the order in which the user has specified. If there is an order to be released, this function releases the order from the station. If the station is not the last station, this function moves the order to the next station. If the station is the last one, this function moves the order to the completed or incomplete set as appropriate. Note that this function executes this step on all the stations in the order in which the user has entered the stations, and not necessarily in the order of their linkage. (This execution order is important for matching the intermediate output generated by this function). This function returns true if all the orders have been processed; false otherwise.

Milestone Submission

To test and demonstrate execution of your program use the same data as shown in the output example above.

Upload your source code to your `matrix` account. Compile and run your code using the latest version of the gcc compiler and make sure that everything works properly.

Then, run the following command from your account: (replace `profname.proflastname` with your professor's Seneca userid)

~profname.proflastname/submit 345XXX_ms3<ENTER>

and follow the instructions. Replace **XXX** with the section letter(s) specified by your instructor.