Project – Milestone 1

Assembly Line - Inventory Items and Project Utilities

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

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.

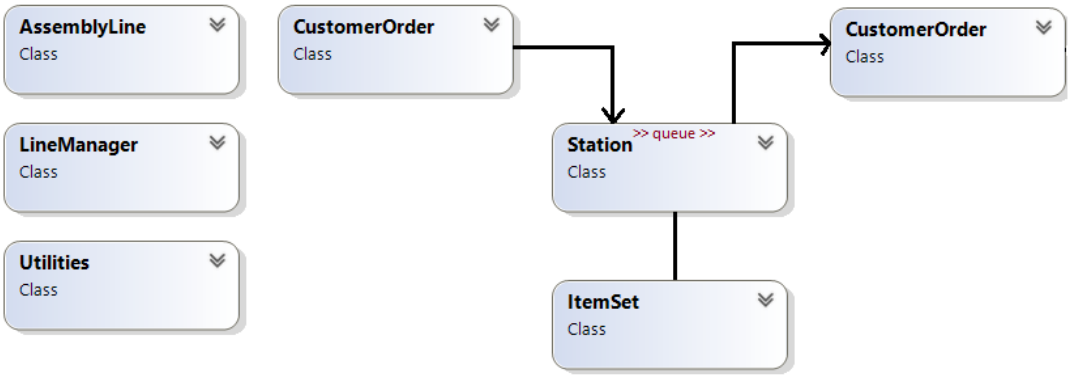


Figure 1 – Simulator Classes

SPECIFICATIONS

Milestone 1 builds the Inventory and the Utilities for the Assembly Line. This milestone consists of four modules:

- **project** (supplied)
- **AssemblyLine** (supplied)
- **Utilities**
- **ItemSet**

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\MS1.exe Inventory.txt

Inventory Assembly
==================

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

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

Inventory Assembly Complete
```

The input for testing your solution is stored in a supplied file. The name of the file is specified on the command line as shown in red above. Its records are

```
CPU|123456|5|Central Processing Unit
Memory|654321|10|Basic Flash Memory
GPU|456789|2|General Processing Unit
SSD|987654|5|Solid State Drive
Power Supply|147852|20|Basic AC Power Supply
```

Each record consists of 4 fields delimited by a prescribed char ('|') for the project. The fields are:

- Name of the inventory item in the item set
- Serial number to be assigned to the item extracted from the set
- Number of items in the item set
- Description of any item in the item set

Utilities Module

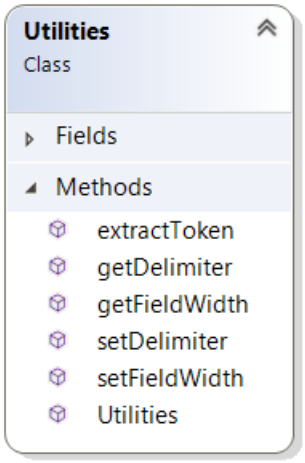
The Utilities module is a support module that contains the functionality that is common across the system. All objects in the system parse string data in the same way, use the same delimiter and report data fields in tabular format.

Design and code a class named **Utilities** for extracting tokens from a string, which consists of a set of fields separated by a specified delimiter and determining a field width that is sufficiently large to accommodate the tokens for a specified field. The field width is to be used to construct the output tables for the project (as shown in the sample output above). Tokens in a string are separated by one delimiter character (see the input file shown in red above).

Your class design includes the following public member functions:

- A default constructor that places the object in a safe empty state and initializes its field width to a size that is less than the possible size of any token.

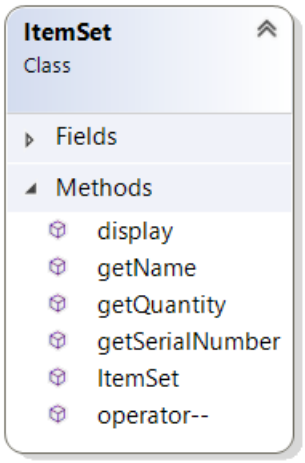
- `const std::string extractToken(const std::string& str, size_t& next_pos)` – a modifier that receives a reference to an unmodifiable string **str** and a reference to a position **next_pos** in that string. This function extracts the next token in the string starting at the position **next_pos**, and ending immediately before the delimiter character. This function compares the size of the extracted token to the field width currently stored in the object and if the size of the token is greater than that width increases that width. This function returns in **next_pos** the position of the next token in the string if one exists. If not, this function returns the position that is beyond the end of the string. This function reports an exception if one delimiter follows another without any token between them.



- `const char getDelimiter() const` – a query that returns the delimiter character
- `size_t getFieldWidth() const` – a query that returns the field width for the current object
- `void setDelimiter(const char d)` – a modifier that set the delimiter character for all object of this class
- `void setFieldWidth(size_t)` – a modifier that set the field width for the current object

ItemSet Module

Design and code a class named **ItemSet** for managing the stock inventory of a particular item. Your class design includes the following public member functions:



- A constructor that receives a reference to an unmodifiable string, extracts 4 tokens from the string, populates the object with those tokens and determines the field width to be used in displaying data for all objects of the class.
- `const std::string& getName() const` – a query that returns a reference to the name of the item
- `const unsigned int getSerialNumber() const` – a query that returns the serial number of the item
- `const unsigned int getQuantity() const` – a query that returns the remaining number of items in the set
- `ItemSet& operator--()` – a prefix decrement operator that reduces the number of items in stock by one and increases the serial number by one. This operator returns a reference to the current object.
- `void display(std::ostream& os, bool full) const` – a query that receives a reference to an **std::ostream** object **os** and a Boolean **full** and inserts the data for the current object into stream **os**. If the Boolean is **false** the data consists of the name of the

items in the set and the next serial number to be assigned. If the Boolean is **true**, the data consists of the name, serial number quantity in stock and the description of the items in the set (as shown above). The field width for the name is just large enough to display the largest name. The field width for the serial number is 5 with '0' fill and the field width for the quantity in stock is 3 left-justified. Fields are separated by a single blank character.

Your design disables copy and move assignment operations and copy construction of the list.

Project – Milestone 2

Assembly Line – Customer Orders

Output:

```
Command Line : C:\Users\...\Debug\MS2.exe Inventory.txt CustomerOrders.txt

Customer Order Assembly
=======================

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

For Manual Validation: Item 1
 getName(): CPU
 getSerialNumber(): 123456
 getQuantity(): 5
 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

For Manual Validation
Chris T.     [Micro Controller]
             GPU
```

```
                GPU
                Power Supply
                SSD
Chris T.        [Micro Controller]
                GPU
                GPU
                Power Supply
                SSD

Chloe           [Flight PC]
                CPU
                GPU
                Power Supply
Mary-Lynn M.    [Desktop PC]
                CPU
                Memory
                Power Supply

For Manual Validation Filling
Mary-Lynn M.    [Desktop PC]
                CPU
                Memory
                Power Supply
 isFilled(): false
 Filled Mary-Lynn M. [Desktop PC][CPU][123457]
 isFilled(): false
 Unable to fill Mary-Lynn M. [Desktop PC][CPU][123457] already filled
 Filled Mary-Lynn M. [Desktop PC][Memory][654321]
 Filled Mary-Lynn M. [Desktop PC][Power Supply][147852]
 isFilled(): true

Customer Order Assembly Complete
```

The input for testing your solution is stored in two files `Inventory.txt` (from Milestone 1) and `CustomerOrders.txt`. The names of these files are specified on the command line as shown in red above. The records in the `CustomerOrders.txt` file are

```
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
```

Each record consists of no less than 3 fields delimited by a prescribed char ('|') for the project. These fields are:

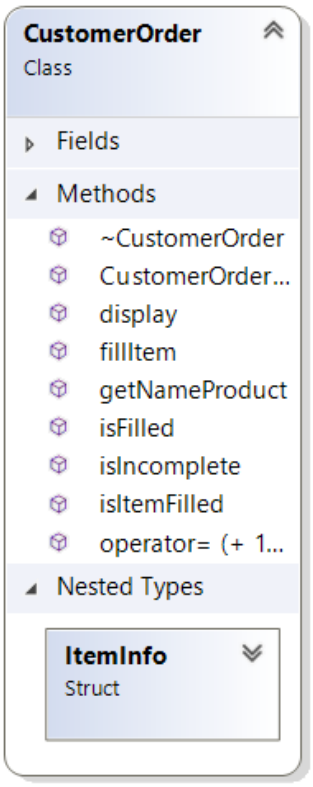
- Name of the customer
- Name of the product being assembled
- Names of the items to be added to the product

Customer Order Module

The Customer Order module contains all the functionality for handling customer orders as they move along the assembly line. As the line manager moves an order along the assembly line, the station where that order currently rests fills a request for an item of that station, if there is any such request. Once the order has reached the end of the line, the order is either completed or incomplete. One cause of incompleteness is the lack of sufficient items in stock at a station.

Design and code a class named **CustomerOrder** for managing and processing customer orders. Customer order objects are unique and hence neither copyable nor copy-assignable. However, they are both movable and move-assignable.

Your class design includes the following public member functions:



- A default constructor that sets the object to a safe empty state.
- A one-argument constructor that receives a reference to an unmodifiable string. This constructor extracts no less than 3 tokens from the string. The first extracted token holds the customer's name. The second token holds the name of the product being assembled. The remaining tokens hold the names of the items to be added to the product throughout the assembly process. This function throws an exception if no items have been requested to be added; that is, there are less than 3 token in the string. Otherwise, this function allocates memory for each item to be added with its fulfillment information. (Hint: one way to hold this information is in array of **ItemInfo** sub-objects nested within the object itself. Information that needs to be tracked includes the name of the item, its serial number, and its filled status.) This constructor determines the field width to be used in displaying customer names for all orders managed by this class.
- A destructor that deallocates any memory that the object has allocated.
- `void fillItem (ItemSet& item, std::ostream& os)` – a modifier that receives a reference to an `ItemSet` object and an `std::ostream` object. This function checks each item request, fills it if the requested item is available and the request has not been filled, reports the filling in the format shown below and decrements the item stock by one:

    Filled CUSTOMER [PRODUCT][ITEM][SERIAL NUMBER]

    If the item request has already been filled or if the item is out of stock, this function displays the corresponding message:

    Unable to fill CUSTOMER [PRODUCT][ITEM][SERIAL NUMBER] already filled

Unable to fill CUSTOMER [PRODUCT][ITEM][SERIAL NUMBER] out of stock

- **`bool isFilled() const`** – a query that searches the list of items requested and returns true if all have been filled; false otherwise.
- **`bool isItemFilled(const std::string& item) const`** – a query that receives the name of an item, search the item request list for that item and returns true if all requests for that item have been filled; false, otherwise. If the item is not in the request list, this function returns true.
- **`std::string getNameProduct() const`** – a query that returns the name of the customer and their product in the following format:

  CUSTOMER [PRODUCT]

- **`void display(std::ostream& os, bool showDetail) const`** – a query that receives a reference to an **std::ostream** object **os** and a Boolean **showDetail** and inserts the data for the current object into stream **os**. If the Boolean is **false** the data consists of the name of the customer, the product being assembled, and the list of items on the order.

  CUSTOMER [PRODUCT]
  ITEM
  ITEM
  ITEM

  Otherwise, the data consists of the name of the customer, the product being assembled, and the list of items with detail information on the order. Details include name of the item, its serial number and its filled status.

  CUSTOMER [PRODUCT]
  [SERIAL NUMBER] ITEM – FILL_STATUS
  [SERIAL NUMBER] ITEM – FILL_STATUS
  [SERIAL NUMBER] ITEM – FILL_STATUS

  All item information is indented exactly the number of spaces required for the longest customer name + 1. See the output above for an example of the formatting.

Project – Milestone 3

Assembly Line – Stations and Line Manager – version 1.2

Output:

```
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

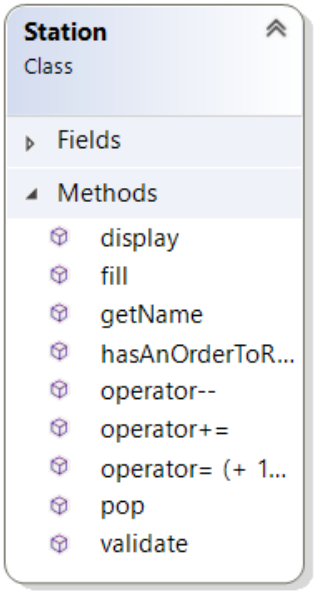
The single field record identifies the last station on the line.

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.