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

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 – March 8 23:59
2. Milestone 2 – Customer Orders – March 22 23:59
3. Milestone 3 – Stations and Line Manager – April 5 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.

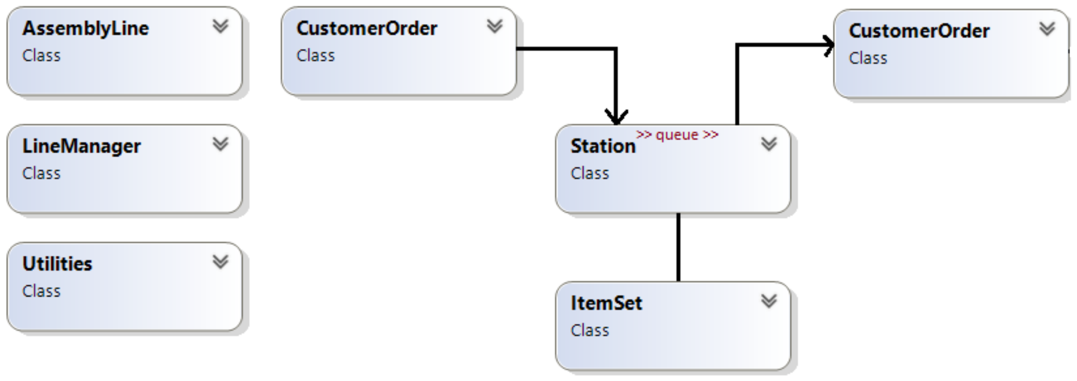


Figure 1 – Simulator Classes

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

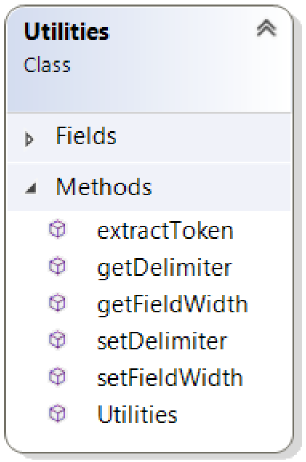
Utilities Module

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:

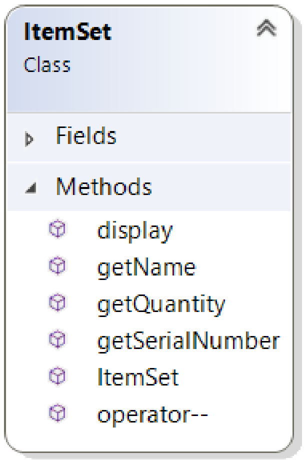
- A default constructor that places the object in a safe empty state and initializes its field width to a size that is less than the possible size of any token.
- `const std::string extractToken(const std::string& str, size_t& next_pos)` – a modifier that receives a reference to an unmodifiable string **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.

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_ms1<ENTER>

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