**Project Overview**

The purpose of this term project is to put your C++ Object Oriented skills to practice by developing a simulated Assembly Line.

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 project is broken down into 3 milestones to help guide you through the process of implementing, debugging and execution of a more complex application than a regular lab/workshop.

**Project Deadlines**

The deadlines for the project's milestones are on Sunday at midnight (see the schedule below).

| **Milestone** | **Date** |
| --- | --- |
| #1 | Nov 17 |
| #2 | Nov 24 |
| #3 | Dec 01 |

All files that you submit (\*.h and \*.cpp) should contain the following comment at the top (with your information filled):

// Name:

// Seneca Student ID:

// Seneca email:

// Date of completion:

//

// I confirm that I am the only author of this file

// and the content was created entirely by me.

**Milestone #3** can have an extension until December 08 2019, with a penalty of 10% for each day that the submission is late. After December 08 2019, the submission process is considered closed.

* submissions received on Dec 02 2019 can receive **max 90%**;
* submissions received on Dec 03 2019 can receive **max 80%**;
* submissions received on Dec 04 2019 can receive **max 70%**;
* submissions received on Dec 05 2019 can receive **max 60%**;
* submissions received on Dec 06 2019 can receive **max 50%**;
* submissions received on Dec 07 2019 can receive **max 50%**;
* submissions received on Dec 08 2019 can receive **max 50%**;

**The project is mandatory!** Students that don't submit a complete working project, cannot pass this course. A project is considered complete if all three milestones have been submited by the deadline and the implementation follows the requirements.

**Milestone 1**

**Utilities and Item classes**

This milestone will implement and test the Utilities and Item classes.

* Utilities class is an object used to support the parsing of input files to setup and configure the assembly line simulation.
* Item class encapsulates the information about a given item on the assembly line that can be filled within a customer order.

The specifications of the classes are defined below.

**Utilities Class**

Parsing string data from input files is performed the same for all objects within the system. The Utilities type provides the basic functionality required for all objects in the assembly line system.

Each Utilities object holds the following information:

* m\_widthField -- specifying the length of the token extracted; used for display purpose later; default value is 1.
* m\_delimiter -- separates the tokens from a given std::string. All Utilities objects **share the same delimiter**.

The member functions of the Utilities class include:

* void setFieldWidth(size\_t) -- sets the field width of the current object to the value of the parameter
* size\_t getFieldWidth() const -- returns the field width of the current object
* const std::string extractToken(const std::string& str, size\_t& next\_pos, bool& more) -- extracts tokens from the first parameter
* This function uses the delimiter to extract the next token from str starting at next\_pos.
* If successful, it returns the extracted token found and sets more to true; false otherwise.
* This function reports an exception if there are two delimiters with no token between them.
* This function updates the current object's m\_widthField data member if its current value is less than the size of the token extracted.
  + **Note:** str represents a single line read from an input file
* static void setDelimiter(const char) -- sets the delimiter for this class to the character received
* const char getDelimiter() const -- returns the delimiter character of the current object.

**Item Class**

An Item object manages a single item on the assembly line.

Each Item holds the following information:

* m\_name – the name of the Item
* m\_description – description of the item
* m\_serialNumber – the next serial number to be assigned on the assembly line
* m\_quantity – the number of current items left in stock
* m\_widthField – the maximum characters to be displayed for a field
* **Note:** All Item objects share the same m\_widthField

The member functions of the Item class include:

* custom constructor
* upon instantiation, an Item object receives a reference to an unmodifiable std::string. This string contains a single record (one line) that has been retrieved from the input file specified by the user.
* the constructor uses an Utilities object (created local in the function) to extract each token from the record and populates the Item object accordingly.
* **Note:** You can assume that a record from the file contains 4 fields separated by a delimiter, in the following order:
  + name of item
  + starting serial number
  + quantity in stock
  + description
* the delimiter of the tokens is a single character, specified by the client and stored into the Utilities object.
* once the constructor has extracted all of the tokens from the record, it updates Item::m\_widthField to the maximum value of Item::m\_widthField and Utilities::m\_widthField.
* **Note:** the display(...) member function uses this field width to align the output across all of the records retrieved from the file.
* const std::string& getName() const - returns the name of the current Item object
* const unsigned int getSerialNumber() – returns the next serial number to be used on the assembly line and increments m\_serialNumber
* const unsigned int getQuantity() – returns the remaining quantity of the current Item object
* void updateQuantity() – substracts 1 from the available quantity; should not go below 0.
* void display(std::ostream& os, bool full) const -- inserts the content of the current object into first parameter.
* if the second parameter is false, this function inserts only the name and serial number in the format: NAME [######]
* if the second parameter if true, this function uses the following format: NAME [######] Quantity: QTY Description: DESCRIPTION
* the NAME and QTY fields will use m\_widthField characters, serial number field will use 6 characters; the DESCRIPTION has no formatting options
* this function will terminate the printed message with an endline

**Tester Module**

The tester module and input files have been added to the repository. Do not modify any of them. See in the sample output below the expected command line.

**Sample Output**

The output for this milestone should look as follows:

Command Line: ms1 Inventory1.dat Inventory2.dat

=======================

= Inventory (summary) =

=======================

CPU [123456]

Memory [654321]

GPU [056789]

SSD [987654]

Power Supply [147852]

=======================

= Inventory (full) =

=======================

CPU [123456] Quantity: 5 Description: Central Processing Unit

Memory [654321] Quantity: 10 Description: Basic Flash Memory

GPU [056789] Quantity: 7 Description: General Porcessing Unit

SSD [987654] Quantity: 5 Description: Solid State Drive

Power Supply [147852] Quantity: 20 Description: Basic AC Power Supply

=======================

= Manual Validation =

=======================

getName(): CPU

getSerialNumber(): 123456

getSerialNumber(): 123457

getQuantity(): 5

getQuantity(): 4

**Submission**

Before the due date, upload to your matrix account the following files:

* Utilities.h
* Utilities.cpp
* Item.h
* Item.cpp

From a command prompt, execute the following command:

~profname.proflastname/submit 305\_ms1

and follow the instructions.

**A successful submission does not guarantee full credit!**

**Milestone #2**

This milestone will implement and test the CustomerOrder class.

The CustomerOrder 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.

**CustomerOrder class**

A CustomerOrder object manages a single order on the assembly line and contains the following information:

* std::string m\_name – the name of the customer (e.g., John, Sara, etc)
* std::string m\_product – the name of the product being assembled (e.g., Desktop, Laptop, etc)
* unsigned int m\_cntItem – a count of the number of items for the customer's order
* ItemInfo\*\* m\_lstItem – a dynamically allocated array of pointers. Each element of the array is a dynamically allocated object of type ItemInfo (see below). **This is the resource** that your class must manage.
* static size\_t m\_widthField – the maximum width of a field, used for display purposes

The member functions of the CustomerOrder class include:

* default constructor
* a custom constructor that takes as a parameter a reference to a string containing a single record from the input file. This constructor uses an Utilities object to extract the tokens and populate the current instance. After the extraction is finished, will update CustomerOrder::m\_widthField if the value stored there is smaller than the value stored in Utilities::m\_widthField.
  + fields in the record are (separated by a delimiter):
    - Customer Name
    - Order Name
    - the list of items making up the order (at least one item)
* a CustomerOrder object should not allow copy operations. The copy constructor should throw an exception if called and the copy operator= should be deleted.
* move constructor. This constructor should "promise" that it doesn't throw exceptions. Use the noexcept keyword in the prototype.
* move assignment operator
* a destructor
* bool getItemFillState(std::string) const – returns the ItemInfo::m\_fillState of the item specified as a parameter. If the item doesn't exist in the order, return true.
* bool getOrderFillState() const – returns true if all the items in the order have been filled; false otherwise
* void fillItem(Item& item, std::ostream&) – fills the item in the current order that corresponds to the item passed into the function (the parameter item represents what is available in the inventory).
  + if item cannot be found in the current order, this function does nothing
  + if item is found, and the inventory contains at least one element, then this function substracts 1 from the inventory and updates ItemInfo::m\_serialNumber and ItemInfo::m\_fillState. Also it prints the message Filled NAME, PRODUCT[ITEM\_NAME].
  + if item is found, and the inventory is empty, then this function prints the message Unable to fill NAME, PRODUCT[ITEM\_NAME].
  + all messages printed should be terminated by an endline
* void display(std::ostream&) const – displays the state of the current object in the format (see the sample output for details)
* CUSTOMER\_NAME - PRODUCT
* [SERIAL] ITEM\_NAME - STATUS
* [SERIAL] ITEM\_NAME - STATUS
* ...
  + SERIAL - a field of width 6
  + ITEM\_NAME - a field of size m\_widthField
  + STATUS is either FILLED or MISSING
  + you will have to use IO manipulators for the output.

Also, add to the header CustomerOrder.h, the following structure:

struct ItemInfo

{

std::string m\_itemName;

unsigned int m\_serialNumber = 0;

bool m\_fillState = false;

ItemInfo(std::string src) : m\_itemName(src) {};

};

**Tester Module**

The tester module and input files have been added to the repository. Do not modify any of them. See in the sample output below the expected command line.

**Sample Output**

The output for this milestone should look as follows:

Command Line: ms2 Inventory1.dat Inventory2.dat CustomerOrders.dat

=======================

= Inventory (summary) =

=======================

CPU [123456]

Memory [654321]

GPU [056789]

SSD [987654]

Power Supply [147852]

=======================

= Inventory (full) =

=======================

CPU [123456] Quantity: 5 Description: Central Processing Unit

Memory [654321] Quantity: 10 Description: Basic Flash Memory

GPU [056789] Quantity: 7 Description: General Porcessing Unit

SSD [987654] Quantity: 5 Description: Solid State Drive

Power Supply [147852] Quantity: 20 Description: Basic AC Power Supply

=======================

= Manual Validation =

=======================

getName(): CPU

getSerialNumber(): 123456

getSerialNumber(): 123457

getQuantity(): 5

getQuantity(): 4

=======================

= Orders =

=======================

Elliott C. - Gaming PC

[000000] CPU - MISSING

[000000] Memory - MISSING

[000000] GPU - MISSING

[000000] GPU - MISSING

[000000] GPU - MISSING

[000000] SSD - MISSING

[000000] Power Supply - MISSING

Chris S. - Laptop

[000000] CPU - MISSING

[000000] Memory - MISSING

[000000] SSD - MISSING

[000000] Power Supply - MISSING

Mary-Lynn M. - Desktop PC

[000000] CPU - MISSING

[000000] Memory - MISSING

[000000] Power Supply - MISSING

Chris T. - Micro Controller

[000000] GPU - MISSING

[000000] GPU - MISSING

[000000] Power Supply - MISSING

[000000] SSD - MISSING

=======================

= Manual Validation =

=======================

CustomerOrders::display(): Chris T. - Micro Controller

[000000] GPU - MISSING

[000000] GPU - MISSING

[000000] Power Supply - MISSING

[000000] SSD - MISSING

CustomerOrders::CustomerOrders(&): ----> ERROR: Cannot make copies.

CustomerOrders::CustomerOrders(&&): Chris T. - Micro Controller

[000000] GPU - MISSING

[000000] GPU - MISSING

[000000] Power Supply - MISSING

[000000] SSD - MISSING

CustomerOrders::CustomerOrders(string): Chloe - Flight PC

[000000] CPU - MISSING

[000000] GPU - MISSING

[000000] Power Supply - MISSING

CustomerOrders::operator=(&&): Mary-Lynn M. - Desktop PC

[000000] CPU - MISSING

[000000] Memory - MISSING

[000000] Power Supply - MISSING

CustomerOrders::fillItem()

getOrderFillState(): MISSING

Filled Mary-Lynn M., Desktop PC[CPU]

getItemFillState("CPU"): FILLED

getOrderFillState(): MISSING

Filled Mary-Lynn M., Desktop PC[CPU]

Filled Mary-Lynn M., Desktop PC[Memory]

Filled Mary-Lynn M., Desktop PC[Power Supply]

getOrderFillState(): FILLED

**Submission**

Before the due date, upload to your matrix account the following files:

* Utilities.h
* Utilities.cpp
* Item.h
* Item.cpp
* CustomerOrder.h
* CustomerOrder.cpp

From a command prompt, execute the following command:

~profname.proflastname/submit 305\_ms2

and follow the instructions.

**A successful submission does not guarantee full credit!**

**Milestone 3**

This milestone will implement and test the Task and LineManager classes.

The Task class is a simulation of a station on the assembly line and contains all the functionality for filling customer orders with items. Each Task is responsible for a particular Item, through inheritance.

The LineManager class is responsible for the execution and movement of CustomerOrders along the assembly line (from start to finish). 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.

**Task Class**

A Task object manages a single Item on the assembly line.

Task inherits from class Item and contains the following additional information:

* m\_orders – is a double ended queue with new CustomerOrders coming in one side and exiting out the other once filled.
* m\_pNextTask – a pointer to the next task on the assembly line

The member functions of the Task class include:

* a custom constructor -- upon instantiation, a Task object receives a reference to an unmodifiable std::string. This string contains a single record (one line) that has been retrieved from the input file specified by the user to be used for Item instantiation.
  + this constructor will also set the m\_pNextTask to a safe state.
* A Task object represents a single location on the assembly line for filling items into customer orders. Therefore, a Task object cannot be copied or moved. You must make sure this capability has been deleted from your Task definition.
* void runProcess(std::ostream&) – runs a single cycle of the assembly line for the current task.
  + If there are CustomerOrders in the queue, it will verify the fill status of the **last** CustomerOrder and fill it if the order contains the Item specified by the current Task.
* bool moveTask() – moves the last CustomerOrder in the queue to the next task on the assembly line if the orders fill state for the current Item is true. Otherwise, the CustomerOrder should remain in the queue for the next processing cycle. If the queue is empty, return false.
* void setNextTask(Task&) – stores the provided Task object's reference into the m\_pNextTask pointer.
* bool getCompleted(CustomerOrder&) – Removes the last CustomerOrder from the queue, places it in the parameter and returns true. If the CustomerOrder queue is empty, return false.
* void validate(std::ostream&) – writes the name of the Item this Task is responsible for into the parameter: ITEM\_NAME --> NEXT\_ITEM\_NAME
  + if m\_pNextTask does not exist it writes: ITEM\_NAME --> END OF LINE.
  + the messages are terminated with an endline
* Task& operator+=(CustomerOrder&&) – Moves the parameter onto the front of the CustomerOrder queue.

**LineManager class**

A LineManager object manages the entire assembly line and contains the following information:

* std::vector<Task\*> AssemblyLine – A container containing all the references of the Task objects on the assembly line
* std::deque<CustomerOrder> ToBeFilled – A queue of customer orders to be filled
* std::deque<CustomerOrder> Completed – A queue of customer orders completed
* unsigned int m\_cntCustomerOrder – The number of CustomerOrder objects the assembly line started with.

The member functions of the LineManager class include:

* custom constructor
  + with the following arguments:
    - a reference to an unmodifiable std::string. This string contains the filename specified by the user to be used for linking the assembly line objects (example: AssemblyLine.dat)
    - a reference to a std::vector<Task\*> that contains the addresses of all the Task objects created for the assembly line
    - a reference to a std::vector<CustomerOrder> that contains all the CustomerOrder objects to be filled
  + the constructor will:
    - read the records from file and setup all the m\_pNextTask references in the Task objects, Linking each Task object together to form the assembly line
    - Move all the CustomerOrder objects onto the front of the ToBeFilled queue
    - Copy all the Task objects into the AssemblyLine container
* bool run(std::ostream&) – this function performs **one** cycle of operations on the assembly line by doing the following:
  + If there are any CustomerOrder objects in the ToBeFilled queue, move the last CustomerOrder object onto the starting point of the AssemblyLine (you have to identify which task is the starting point of your assembly line)
  + Loop through all tasks on the assembly line and run one cycle of the task's process
  + Loop through all tasks on the assembly line and move the CustomerOrder objects down the line.
    - **Hint:** completed orders should be moved into the Completed queue.
  + return true if all customer orders have been filled, otherwise returns false.
* void displayCompleted(std::ostream&) const -- displays all the orders that were completed
* void validateTasks() const -- validates each task on the assembly line

**Tester Module**

The tester module and input files have been added to the repository. Do not modify any of them. See in the sample output below the expected command line.

**Sample Output**

The output for this milestone should look as follows:

Command Line: ms3 Inventory1.dat Inventory2.dat CustomerOrders.dat AssemblyLine.dat

=======================

= Tasks (summary) =

=======================

CPU [123456]

Memory [654321]

GPU [056789]

SSD [987654]

Power Supply [147852]

=======================

= Tasks (full) =

=======================

CPU [123456] Quantity: 5 Description: Central Processing Unit

Memory [654321] Quantity: 10 Description: Basic Flash Memory

GPU [056789] Quantity: 7 Description: General Porcessing Unit

SSD [987654] Quantity: 5 Description: Solid State Drive

Power Supply [147852] Quantity: 20 Description: Basic AC Power Supply

=======================

= Manual Validation =

=======================

getName(): CPU

getSerialNumber(): 123456

getSerialNumber(): 123457

getQuantity(): 5

getQuantity(): 4

=======================

= Orders =

=======================

Elliott C. - Gaming PC

[000000] CPU - MISSING

[000000] Memory - MISSING

[000000] GPU - MISSING

[000000] GPU - MISSING

[000000] GPU - MISSING

[000000] SSD - MISSING

[000000] Power Supply - MISSING

Chris S. - Laptop

[000000] CPU - MISSING

[000000] Memory - MISSING

[000000] SSD - MISSING

[000000] Power Supply - MISSING

Mary-Lynn M. - Desktop PC

[000000] CPU - MISSING

[000000] Memory - MISSING

[000000] Power Supply - MISSING

Chris T. - Micro Controller

[000000] GPU - MISSING

[000000] GPU - MISSING

[000000] Power Supply - MISSING

[000000] SSD - MISSING

=======================

= Validate Tasks =

=======================

CPU --> GPU

Memory --> SSD

GPU --> Memory

SSD --> END OF LINE

Power Supply --> CPU

=======================

= Filling Orders =

=======================

Filled Elliott C., Gaming PC[Power Supply]

Filled Elliott C., Gaming PC[CPU]

Filled Chris S., Laptop[Power Supply]

Filled Chris S., Laptop[CPU]

Filled Elliott C., Gaming PC[GPU]

Filled Elliott C., Gaming PC[GPU]

Filled Elliott C., Gaming PC[GPU]

Filled Mary-Lynn M., Desktop PC[Power Supply]

Filled Mary-Lynn M., Desktop PC[CPU]

Filled Elliott C., Gaming PC[Memory]

Filled Chris T., Micro Controller[Power Supply]

Filled Chris S., Laptop[Memory]

Filled Elliott C., Gaming PC[SSD]

Filled Mary-Lynn M., Desktop PC[Memory]

Filled Chris T., Micro Controller[GPU]

Filled Chris T., Micro Controller[GPU]

Filled Chris S., Laptop[SSD]

Filled Chris T., Micro Controller[SSD]

=======================

= Completed Orders =

=======================

Elliott C. - Gaming PC

[123458] CPU - FILLED

[654321] Memory - FILLED

[056789] GPU - FILLED

[056790] GPU - FILLED

[056791] GPU - FILLED

[987654] SSD - FILLED

[147852] Power Supply - FILLED

Chris S. - Laptop

[123459] CPU - FILLED

[654322] Memory - FILLED

[987655] SSD - FILLED

[147853] Power Supply - FILLED

Mary-Lynn M. - Desktop PC

[123460] CPU - FILLED

[654323] Memory - FILLED

[147854] Power Supply - FILLED

Chris T. - Micro Controller

[056792] GPU - FILLED

[056793] GPU - FILLED

[147855] Power Supply - FILLED

[987656] SSD - FILLED

=======================

= Inventory (full) =

=======================

CPU [123461] Quantity: 1 Description: Central Processing Unit

Memory [654324] Quantity: 7 Description: Basic Flash Memory

GPU [056794] Quantity: 2 Description: General Porcessing Unit

SSD [987657] Quantity: 2 Description: Solid State Drive

Power Supply [147856] Quantity: 16 Description: Basic AC Power Supply

**Submission**

Create a **text** file named reflect.txt. Add any comments you wish to make.

Before the due date, upload to your matrix account the following files:

* Utilities.h
* Utilities.cpp
* Item.h
* Item.cpp
* CustomerOrder.h
* CustomerOrder.cpp
* Task.h
* Task.cpp
* LineManager.h
* LineManager.cpp
* reflect.txt

From a command prompt, execute the following command:

~profname.proflastname/submit 305\_ms3

and follow the instructions.

**A successful submission does not guarantee full credit!**