Assembly Line Design/Implementation Specifications

Data Structures – Term Project

Fall 2022

Table of Contents

Project Overview

This semesters project is to design and implement a simulated assembly line using the data structure techniques you have learned in class. You will be required to write a C++ object oriented program that takes input from a file to build products, passes those products through the assembly line to be manufactured and then register them based on product serial number afterwards.

To help aid you in the development of this project, it has been broken down into three easy milestones each building on the other. Details of the design/implementation requirements for each milestone can be found in the following subsections of this document.

System Design

When completed, your software application will create several Task objects linked together using a single-ended link list representing the assembly line. Each task will have a queue for products being processed and a stack with available inventory. Each task will be connected to the next via a pointer (single ended link list). The output of the assembly line will be two arrays, a backordered products and completed products. All completed products will be registered using the registration module, which consist of a binary tree based on Product serial numbers.

Milestone #1 Details

The goal of the first Milestone for this project will be to design and implement the product structure, task object and container classes that will create the single-ended assembly line link list.

Product & Component Structures

The simulated assembly line requires a product to manage and assemble. The product being simulated is a computer gaming system. Each product will contain a Product Serial Number (S/N) and an array of Items to assembly with it. An item can be one or more of the following:

- CPU
- GPU
- RAM
- Monitor
- Controller
- VR Headset
- AR Glasses

Each item consists of a Name, S/N. Your structures should look as follows in your design:

```
struct Item
{
	std::string Name;
	unsigned int SN;
};

struct Product
{
	unsigned int SN;
	Item Components[8];
};
```

Class Task

The task class is responsible for managing one item on the assembly. As a product is presented to the task it will check to see if the product requires one of its items. If it does, it will pop an item off the inventory stack and update the product with the S/N from the inventory. Each task will consist of the following private member variables and public functionality:

Private:

- `std::string Name` – represents the name of the item the task is responsible for (see list of items in previous section).
- `queue<Product*> ProdQ` – is an STL Queue object that handles the products in and out of the task on the assembly line
- `stack<Item*> Inventory` – is an STL Stack object that handles the inventory available for the task

Public:

- `Task(string)` – a constructor that will set the name of the item the task is responsible for and initialize a random amount of inventory with randomly generated S/N
- `void Run()` – represents the functionality of one iteration of the assembly line execution. This function will
  - Check to see if the next product in the tasks queue requires an item from this station. Remember queues are FIFO so you should be checking the last item in the queue
  - Update product in the queue based on if an item is required and inventory is available
- `Product* MoveProduct()` – this function checks the queue to see if the product at the end of the queue is ready to be pushed to the next task on the assembly line. If yes, the product is popped off the queue and returned to the main program to transfer along the assembly line.
  - *HINT: What should happen to a product that is not fulfilled and the inventory of the task is empty?*
- `void AddProduct(Product*)` – this function pushes a product object onto the tasks product queue.

Class AssemblyLine

The AssemblyLine object is the wrapper class (or container class) that makes a single station on the assembly line. Essentially the AssemblyLine is the data type of your single-ended link list. It should have the follow private and public variables and member functions:

Private:

- `Task* pTask` – The Task object that this part of the AssenblyLine is responsible for
- `AssemblyLine* pNext` -- pointer to the next task on the assembly line

Public:

- `AssemblyLine(Task*)` – a constructor that creates an AssemblyLine object for a given Task and sets the pNext pointer to a safe empty state (i.e. nullptr).

- `bool Push(AssemblyLine*)` – a push function that will add the provided AssemblyLine object to the end of the single-ended link list. The function will return true if the object is successfully added and false if there was a problem.
- `void Run()` – this function is the interface to the Task Run function. Calling it will cause the Task object to run
- `Product* MoveProduct()` – this function is the interface to the Task MoveProduct function. Calling it will cause the Task object to move a product
- `void AddProduct(Product*)` – this function is the interface to the Tasks AddProduct function. Calling it will cause the Task object to add a product to it's product queue

Milestone #1 Deliverables

Although it is important for you to compile and test your objects, there is no required execution/output for this milestone. Simply ZIP up your *.cpp and *.h files and upload the source code to eConestoga. Please ***DO NOT*** upload the entire solution. Penalties will be applied if the entire solution space is uploaded.

Milestone #2 Details

In this milestone you will continue development of your simulated assembly line by working on the main application. The main function will be responsible for reading the input data file (sample provided) and execute the assembly line process. Your main function should provide the following functionality:

- The ability to open and read an ASCII input file until End of File (EOF). Format of the file can be found in APPENDIX A.
- For each Product read from the input file, create a Product object (based on the input details) and add it to the Product queue of the first task on the assembly line
- Execute the simulated assembly line by pushing products along from start to finish. The order of operations is detailed out below.
- When a product is popped off the Assembly Line from the last task it will be assessed to determine if all items were installed or not (HINT: check to see if all components of the product have S/N's). If the Product is incomplete it should be placed in the "backordered" array. If the Product is complete it should be placed in the "Completed Orders" array.
- Display a full list of the products that are on backorder and the ones that have been completed.

The order of operations for the assembly line are as follows:

1. Read an item from the input file and create a product object
2. Add that product object to the first task on the assembly line
3. "Run" each of the assembly line tasks from *back to front*
4. "MoveProducts" along the assembly line from *front to back*
5. Repeat steps 1-4 until
    a. The entire input file has been read, ***AND***
    b. All tasks have empty product queues

Milestone #2 Deliverables

When you are finished, capture an image of the screen with the final output of your program. ZIP up you main.cpp and screenshot files and upload it to eConestoga. Please ***DO NOT*** upload the entire solution. Penalties will be applied if the entire solution space is uploaded.

Milestone #3 Details

Now that your simulated assembly line is functioning correctly we are going to add one more module into the mix. This module will be a registration system and store all S/N's of completed Products that come off the assembly line.

Class Registration

The registration system will consist of a single registration class that will store the S/N's of all completed Products using a binary tree. It will also provide the ability to traverse the tree, in a fast and efficient manner, to determine if a product was registered or not. How you implement the binary tree is up to you (HINT: building some supporting structures/classes may help), but the registration class should have the following interfaces (as a minimum):

- `bool RegisterProduct(unsigned int)` – a function that takes the S/N of the completed Product and registers it by inserting the S/N to the binary tree. If the registration is successful it returns TRUE. If the S/N already exists (i.e. the product has been registered before) the function will return FALSE
- `bool IsRegistered(unsigned int)` – a function that takes the S/N of a product and traverses the binary tree. If the S/N exists the function returns TRUE stating the product is registered, otherwise it returns FALSE

Evaluation

The following is a breakdown of the evaluation for this term project:

Component	What?	Points
Milestone #1	Task and Assembly Line Classes Implemented	12
Milestone #2	Main function and Application Execution	12
Milestone #3	Registration Class and Application Execution	12
	TOTAL POINTS	36

See eConestoga for details about point allocation and due dates of each Milestone.

APPENDIX A: Input File Format

One sample input file has been provided to you, however, one is not enough to test your simulated assembly line. It is your responsibility to generate several inputs files to make sure you have tested all aspects of your system. Things to consider when creating your files:

- Make enough Products to exceed task inventory for testing

- Make Products that have all one of each items
- Make Products that have only one item
- Make Products that have multiple of one item

The format of the input file is a simple ASCII text file as follows:

- A line starting with a '-' represents a new item to be added into a Product
- A line without a '-' represents (a) the start of a new Product, (b) the Serial Number of that new Product
- A Product should always have at least 1 item associated with it. So you should never receive a input with two Product serial numbers back-to-back.

Sample input File:

```
123456
-CPU
-RAM
-VR Headset
654023
-CPU
-GPU
-GPU
-Monitor
```