John Kirschner
07-29-2018
This report is written for the practice problem presented by MModal during the interview process.

Steps Involved with Compilation

The Widget Factory solution is stored and managed with GitHub. The download can be found at this location. The repository can be cloned with git. The command line variant for cloning the repository is “git clone https://github.com/handfistface/WidgetFactory.git”. Once the repository is cloned onto the local machine open the sln file with Microsoft Visual Studios. The project was compiled and tested on Visual Studios 2017. Verify that the correct .NET framework is installed, the program was compiled and tested using .NET 4.5.2. Use Visual Studios to compile the Widget Factory application.

The Debug and Release modes of the application can be run and tested through Visual studios. The binaries are in ./WidgetFactory/bin/Release or ./WidgetFactory/bin/Debug.

Description of Solution

The Widget Factory problem is presented in a vague format. There is a general description of the process involved, but as described, it was left for interpretation.
The program was designed so that the user could write some object definitions and then upload them to the program.
There are two types of files, build order files and widget specification files. The widget specification files define how objects are assembled. The first text in a line details the item that will be constructed, any items following will describe the steps involved in assembly, like what items are assembled next. I have taken it on my own liberty to integrate the ability to specify a number prior to the text, this number indicates how many items in a row that will be assembled before moving onto the next item to assemble. The other type of file is the build order file, this file describes the build order that takes place. I have also taken the liberty to integrate the ability to specify a number prior to the text, this number indicates how many items in a row that will be built. An example of both files can be found in the Documentation folder.
Once the files are chosen then the user may begin construction. For this problem I have allowed the assembly time to be constant between all objects, the estimated time of completion is calculated prior to starting and will be displayed to the user. Arguably one of the most important things about a factory is the output produced in a given time, to argue for the sake of future ideas, the assembly time could easily be integrated into the object definition and that could be used to calculate the estimated time of completion per object with a minimal amount of re-coding.

There are a few quality of life features I added. There is a file dialog that is populated when the user wants to search for a file. The file dialog retains the last known location of the file upload in between application sessions, this is done through an xml configuration file present in the application folder and then AppData folder upon variable change. I have also separated the objects into lists, the lists are components, which are objects that are just outright purchased, and the buildable objects list, which are objects that have requirements to build.

Assumptions Made by Application

Input files must be stored in a UTF-8 encoding, no other encoding is recognized. Input files are also predefined, one definition is through the example provided, and the other allows for a prefixed number prior to the object definition which indicates the quantity required for production. Object construction files must contain at least one component, otherwise the object will not be processed. The current user's AppData folder must be accessible to allow storing persistent session variables. The C: drive must exist for the file dialog to open properly.