GLAB 4

Problem Statement:

The problem we are trying to solve in our program is the batch processing of product inventory control and management in the Hypermarket System. We already designed the user interface flow in the previous Lab. In this lab we add a new functionality to our inventory system by implementing a batch processing system in which takes in all text files containing user names and each user' jobs. The batch processing function will call the system functionalities accordingly and modify the product inventory. The batch processing functionality codes are optimized and kept modular while we also tried to boost the speed and efficiency of the codes.

Design Overview

Our Batch Processing function model has three main components, which are :

1. Batch Processing component
2. Users component
3. Jobs Component

We also made a few changes in the API to support the Batch Processing functions. Our model utilizes 2 particular container classes extensively, which are:

1. Stack class
2. Queue class

The reasons why we split our codes to these components are to make our codes modular and easier to understand

1.Container Class (Stack and Queue)

In our project, we have used STL vector for the stack and queue implementation due to the restrictions of the usage of STL stack and queue implementations. STL vectors were used to utilize convenience of automatic memory allocation and deletion, for fast retrieval of desired items in the stack / queue. Besides, vectors reduce the chances of error in our implementation as one does not have to go through the long tedious process of manually composing a linked list with pointers and related functions which Is prone to error.

For an instance, the vector push_back() function allows the fast and automatic insertion of jobs or users. For the removal of an item in a stack, the pop_back() function is used wheareas for the removal of an item in a queue, the erase() function is used with the begin() function as a parameter. The begin() function retrieves the first item in the vector while the erase function removes the item specified by the parameter.

Users are modeled using stack such that the order of the users who submitted jobs in descending order, with the latest person at the top, i.e. the last person to submit a job at the end of the day. On the other hand, jobs are modeled as queues as the jobs that are submitted first have the top priority. (Jobs submitted first have to be cleared first, for the sake of job efficiency.) This allows the jobs to be listed from the earliest at the top to the latest at the bottom.

2. Jobs Class

Jobs Class basically stores the job information, including the code of the particular job and the required information in completing it. We have split the functionality into 2 main parts, storing the jobs information and executing the jobs after all reading process is done.

Storing jobs functionalities.

Methods	Purposes
bool readjobDetails(ifstream * readFile)	Reads the job code and call the proper storing job functionalities
void job_add(ifstream* readFile)	Reads and stores required information for add product job.
void job_delete(ifstream* readFile)	Reads and stores required information for delete product job.
void job_restock(ifstream* readFile)	Reads and stores required information for restock product job.
void job_dipose(ifstream* readFile)	Reads and stores required information for disposing expired product job.
void job_sale(ifstream* readFile)	Reads and stores required information for selling product job.

Executing Jobs Functionalities

Methods	Purposes
bool job_exec(ifstream * readFile)	Executes the job according to its code and call the proper job execution functionalities
bool exec_add(ListBase<Product>* List)	Executes the add product job according to the stored information.
bool exec_delete(ListBase<Product>* List);	Executes the delete product job according to the stored information.
bool exec_restock(ListBase<Product>* List);	Executes the restock product job according to the stored information.
bool exec_dipose(ListBase<Product>* List);	Executes the dispose expired product job according to the stored information.
bool exec_sale(ListBase<Product>* List);	Executes the sell product job according to the stored information.

3. User Class

The User Class takes in the user's name, the number of jobs the user wants to do and the information of each of the jobs.The job information are then stored into a variable with the type "Jobs" and the variable will then be pushed into a Queue with the type "Jobs".

Basic users functionalities.

Methods	Purposes

string getUser()	Returns the User name
int getNumJobs()	Returns the number of jobs

User Adding functionalities

Methods	Purposes
void addJobs(ifstream* readFile)	Reads in the job information according to the number of jobs and queues it into the job queues

User Executioning functionalities

Methods	Purposes
void execJobs(ListBase<Product> * List)	Executes the jobs according to the job information and code and dequeue it from the queue

User Error functionalities

In addition, whenever there is an error, the function “userLog” will be called and the error will be stored inside a “log.txt”.

Methods	Purposes
void userLog(vector<string> errMsgs)	Writes any errors into txt file

4. Batch Processing Components

The Batch Processing component consisted of BatchJobs class, which has stack of User and number of users. Batch Processing class contains 2 parts, the user adding process and the execution process. The user adding process adds user from the text input and all of the jobs queued by the each user. The execution part pops every user from the stack and execute every user’s job.

Adding user(s) functionalities.

Methods	Purposes
bool addUsers()	Reads user data from file and adds users and its job queues to the stack of user.

Processing user(s)’ command(s) Functionalities

Methods	Purposes
bool startProcess(ListBase<Product> *List)	Pops the user and executes the job(s) queued by each user.

API Component Modification.

We modified the API to support the Batch Processing functionalities. The function “batchProcess” has been added. This will access the “addUser” function in the Batch Jobs Class whenever the batchProcess has been called. The function first creates a new BatchJobs, after that it will call the add User function to add the Users and their jobs. It will then start processing the stack of users by using the “startProcess” function.

Methods	Purposes
bool batchProcess(ListBase<Product>* List)	Calls the addUser function in the BatchJob class and executes the Batch Processing functions

Conclusion

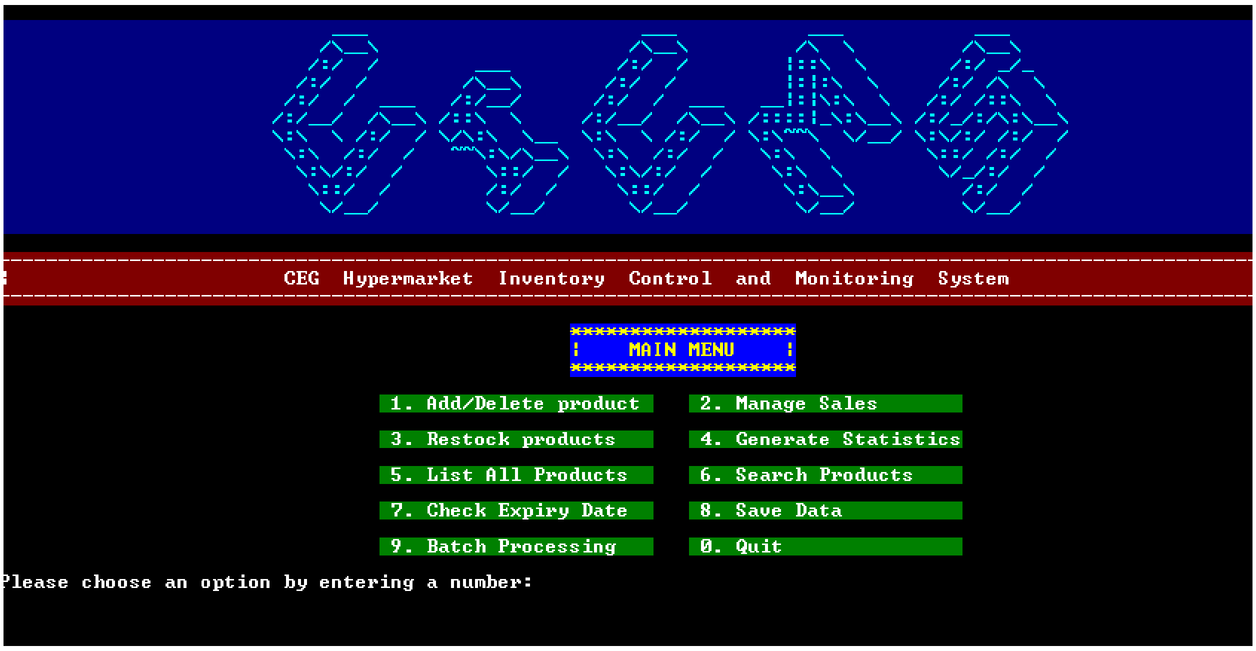
In conclusion, as you have seen in our implementation for the Batch Processing functions in our CICMS system, we utilized Stack and Queue class to store the user and the jobs respectively. There are also some modifications made in the API to support our Batch Processing Functionalities. We optimized the codes few times to ensure the user will get the best user experience with our codes.

Instructions to Use the code:

1. Well just follow the menu and type in the option number you want to use and navigate accordingly. The UI is rather intuitive and self explanatory.

Things we are proud of:

1. Our colouful UI
2. Substring Matching
3. Giving the user a chance to select the product he wants to alter if the query returns many results
4. Validation of most inputs to make sure malicious inputs do not go through our lists.
5. Asking the user to confirm the product details before adding it.



```
CEG  Hypermarket  Inventory  Control  and  Monitoring  System

                 ********************
                 |     MAIN MENU    |
                 ********************

         1. Add/Delete product       2. Manage Sales
         3. Restock products         4. Generate Statistics
         5. List All Products        6. Search Products
         7. Check Expiry Date        8. Save Data
         9. Batch Processing         0. Quit
Please choose an option by entering a number:
```