# SPL 141 Assignment 2

Published - 10/11/2013

Dead line - 24/11/2013 23:59

## 1. Before You Start

1. You must submit all of the assignments in pairs. In addition, you must submit assignment 1 and assignment 2 with the same partner. Therefore please find a partner as soon as possible and **create a submittal group** in the submission system. Once the submission deadline has passed, you will be unable to create submission groups even if you have an approved extension.
2. Read the assignment together with your partner and make sure you understand the tasks. Please do not ask questions before you complete reading the assignment.
3. Write a skeleton of the assignment (i.e. interfaces and class structure).

## 2. The University Coffee Shop – 2nd generation (The Story)

The price reduction helped the University Coffee Shop to stay in the competition with "5 Shekel Coffee" but sales are still a bit low. The owners discovered that the lack of some basic products causes many of the customers to still go to the "5 Shekel Coffee". They requested the help of the department of computer science in order to find a hi-tech solution.

The solution they have chosen was to reduce the labor cost by using an automatic face recognition system so that they will only need a kitchen crew. This means each customer will register the shop by submitting a photo of his face and his favorite product. Each customer will be able to purchase only one product.

The rules for enabling a product are similar to the rules of assignment 1. The sum of ingredient cost + 0.25 NIS (labor cost) + 50% profit. If this number is lower or equal to 5 NIS the product will be included in the menu.

### 2.1 Assignment Goals

The objective of this assignment is to let you design an object-oriented system (class definitions, interaction diagrams), and gain implementation experience in C++ while using classes and standard STL-based data-structures. You will learn how to handle memory in C++ and avoid memory leaks. You will learn to work with 3rd party libraries – OpenCV and POCO – for face detection and logging respectively. The resulting program must operate efficiently as will be explained.

## 3. General Overview

Let us start with a quick overview of the simulation. We have three major kinds of actors in our story: *The University Coffee Shop*, *Customers* and the *Products in the Menu*. In the initial state of the simulation, the Menu is generated using the code you wrote in assignment 1. Customers arrive at the shop, and they can register to the shop, or try to purchase their favorite product. Suppliers can notify the shop of a price change in an ingredient they supply. Your simulation code needs to handle the abovementioned events as will be explained in detail later.

When a customer registers at the coffee shop, he submits a photo of his face. This photo is called a ***photo id*** image. When customers (may be a single customer) enter the shop to perform a purchase, the system automatically takes a picture of them. We will refer to this image as an **input** image. The system should automatically detect faces in the input image and try to identify whether the faces it detected are registered customers or not, using the photo id images. For example, suppose Walter registers to the coffee shop using a photo id image (Figure 1). After some time Walter arrives to the store with his friend (Figure 2). The system should automatically detect the two faces in Figure 2, and check whether they were registered to the shop or not. For each registered face it discovered it should act properly (will be explained in more detail in Section 4 – System Events).

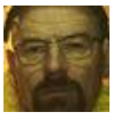

*Figure 1) Walter registers at the coffee shop, and provide the following photo id image.*

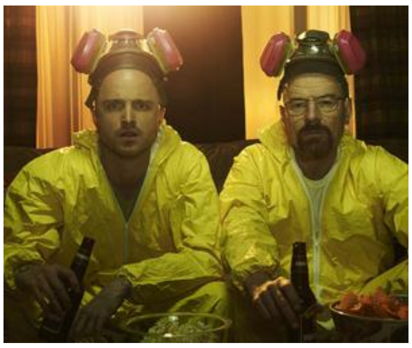

*Figure 2) Walter arrives at the store with a friend*

## 4. System Events

The system supports (at least) three kinds of events: registering a customer, purchase attempt, and ingredient price change.

We will supply three files, whose names will be runtime parameters (and not hard-coded). The files contain the list of *all* products and their ingredients, the list of *all* suppliers and their offered ingredients and prices, and the list of all events that happen during the simulation.

The list of products will be read from the file at the start of the simulation, and will be *fixed* along the simulation. The *menu* will be generated at the beginning of the simulation, but may change according to price change in ingredients.

The events file contains a list of events separated by a newline. Each event should have a different reaction by the program, the actions are as following:

### 4.1. Costumer Registration:

Used to register a new costumer
**Event name**: register
**Command format**:

```
register,<name>,<product_name>,<is_VIP?>
```

Where:

- *<name>* - The name of the costumer. The photo id image for this customer will be in the relative path " faces/<name>/<name>.tiff".
- *<product_name>* - The name of the customer favorite product.
- *<is_VIP>* - A Boolean value (i.e., "0" or "1"), indicating whether the customer is a VIP customer or not.

**What to do**:
Save the details in an appropriate container, for a later use.

### 4.2. Purchase:

A purchase attempt by a customer
**Command name**: purchase
**Command format**:

```
purchase,<customer_image>.tiff
```

Where:

- *< customer_image>*– A name of an input image. It will be located at the path "customers/*<customer_image>*.tiff"

**What to do**:
Use face detection to extract faces in the test image. Then, compare each one of the faces to the photo id images. Supply for each detected registered user his favorite product. VIP customers will get a discount of 20% on products in the menu.

### 4.3. Supplier Price Change

Used to change the price of an ingredient by a supplier
**Command name**: supplier_change
**Command format**:

```
supplier_change,<supplier_name>,<ingredient_name>,<price>
```

Where:

- *<supplier_name>* – The name of the supplier.
- *<ingredient_name>* – The name of the ingredient to change.
- *<price>* – The new price of this ingredient.

**What to do**:
Update the information on the ingredient for that supplier, see how this affects the menu and pricing and act accordingly. This might cause a change in the menu. For example, a price increase in an ingredient may cause a price increase in products that use that ingredient. If the new price is higher than 5 NIS, the product(s) should be removed from the menu. A price decrease may cause a product out of the menu to enter the menu.

### 4.4 End of Simulation

At the end of the simulation (no more events in file) you should compute the total ***revenue*** and ***profit*** of the shop during the simulation. The revenue is the total amount of money the customers paid during the simulation, whereas the profit is just the accumulation of profits made on each sold product (Section 2).

You will also need to create a collage that contains all the images of all the costumers that registered for the coffee shop. Please save the collage images as 'collage.tiff' at your present working directory.

## 5. OpenCV

Step A: Detect a person in an image:

The goal of this step is to detect a person automatically. Automatically means that the user is not allowed to do anything in this phase. The area detected at this stage defines the region of interest (ROI). The algorithm to perform the detection that exists in OpenCV returns a rectangular region of interest.

One way to find the ROI, is first to detect faces in the image and then to approximate where the rest of the bodies lie in the image. Another option is to try to detect the full body as well as the head. To do so, one can use a different classifier.

A classifier is a machine learning technique that searches for a pattern in data according to previous learning examples. For example, a face classifier searches for faces in an entire image and returns for each found face the bounding box around the detected face. A classifier is first trained on many positive and negative examples (i.e., pictures with and without faces). The training results in a model – which is a data structure that contains all the data learned during training. Once trained, the classifier can be executed on new unseen data (i.e., faces).

Fortunately, OpenCV comes with several pre-trained classifiers. One of them is the classical face detector (Viola-jones classifier) – called Cascade Classifier. To use it, you should first load a pre-learned classifier (for example haarcascade_frontalface_alt.xml, which is part of OpenCV) and then execute it on your (gray) image.

Consider the following example:

```
// convert an existing image to gray level only.
// since the classifier was trained on black and white images, it will not work
// on color images. We therefore convert color images to gray-level images
// and then pass the resulting image to the classifier.
cvtColor(objectImg, objectImageGray,CV_BGR2GRAY);
// Load the model before it is used
string face_cascade_name = "haarcascade_frontalface_alt.xml";
CascadeClassifier face_cascade;
face_cascade.load(face_cascade_name);
// use the model to detect faces in the image.
std::vector<Rect> faces;
m_face_cascade.detectMultiScale(objectImageGray, faces, 1.1, 2,
0|CV_HAAR_SCALE_IMAGE);
```

The classifier fills the vector faces with rectangles (cv::Rect) that contain the faces in the picture (See Figure 3). Rec is a data structure representing rectangles using two points (cv::Point). See basic structures in OpenCV. You can follow the links to learn more about cascade classifier and see an example of its use.

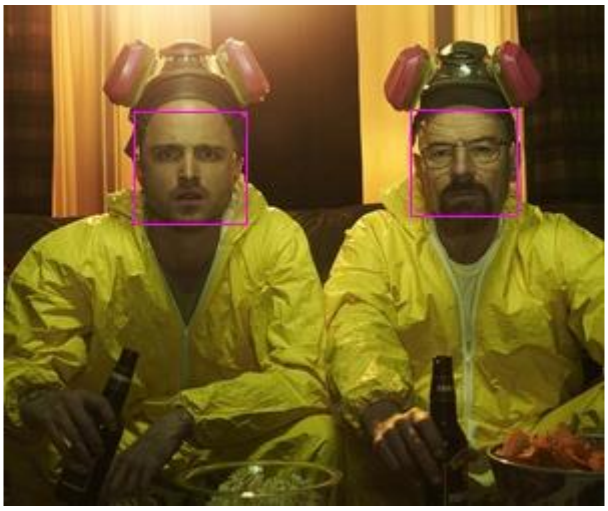
*Figure 3) an example of faces detected using the cascade classifier.*

Step B: Comparing faces:

After extracting the faces from the test image, you need to compare the faces (i.e., the rectangles obtained from the classifier) to the faces of the registered users (photo id images). To implement this you need to compare between the image obtained from the classifier and the image the customer provided at the pixel level. That is, you need to make sure that the images are *identically* the same.

You will use a dataset that we will provide to you. In this dataset the photo id images, were cropped from the input image. You will also have to use the same parameters for the cascade classifier, as shown in the code snippet above. (This will guarantee repeatability of the face cropping).

The directory structure for the images is as follows:

```
-- customers (directory)
        --<customer_image1>.tiff (filename)
        --<customer_image2>.tiff (filename)
        …

-- faces (directory)
        --<name1> (directory)
                -- <name1>.tiff (filename)
        --<name2> (directory)
                -- <name2>.tiff (filename)
        …
```

Step C: Generating a collage.

When the simulation ends, you will need to generate a large picture that is composed of all the faces that were registered during the simulation. You can see an example of such a collage in figure 4. Note that since, the faces are images of different size you will have to resize the face images to generate the collage. The order of faces does not matter.

*Figure 4) A collage generated of 7 registered users*

The assignment page will contain more information about digital images and about OpenCV

## 6. Logger

A logger is a tool that helps you monitor and debug your system. You will make use of a class provided by a 3rd party open source library called POCO. Each time important things happen in your system, you will log it as a message into a file and the console. The POCO Logger (like any standard logger) has a log level, which is used for filtering messages based on their priority. Only messages with a priority equal to or higher than the specified level are passed on. For example, if the level of a logger is set to three (PRIO_ERROR), only messages with priority PRIO_ERROR, PRIO_CRITICAL and PRIO_FATAL will be propagated. The Loggers level will be specified in the configuration file (LOGGER_FILE_PRIORITY and LOGGER_CONSOLE_PRIORITY)

You can use the three lowest priority levels (information, debug and trace) to add your own messages to the logger, which can help you monitor and debug your program.

### 6.1 Logger Messages

The messages you should log with their respective output format are:

1. Error: producers/suppliers/configuration file not found (critical)
   ***<filename> not found***.

2. Customer Registration (notice)
   ***New <VIP> Costumer registered - <name>, Favorite product - <product>.***

   Where:

   - <VIP> - regular/VIP, according to the status of the customer.

3. Supplier price change (notice)
   ***Supplier <supplier_name> changed the price of <ingredient_name>***
   ***Products updated: <updated_num>***

   Where:

   - *<updated_num>* - the number of products in menu that were affected by the change (price updated).

4. Product addition/removal from the menu due to price change (previous item) (warning)
   ***Product <product_name> was <added/removed> from the menu.***

5. Purchase messages (warning)
   ***Costumer <name> purchased <product_name>***

Or if the product isn't sold at the shop

***Costumer <name> failed to purchase <product_name>***

6. End of simulation (warning)
***The total revenue is <revenue>, while the total profit is <profit>***

## 7. Input Files

All The information needed in the simulation will be given in a few configuration files. Unlike assignment 1, where the input file names were hard coded. In this assignment the names of the files will be given to you as command line arguments.

We will execute your assignment from the console with the command:

*SimCoffeeShop* <configuration_file> <products_file> <suppliers_file> <events_file>

1) <**configuration_file**> **-** contains the following lines for setting debug level.

   ```
   LOG_FILE_NAME = <LOG_FILE_NAME>
   LOGGER_FILE_PRIORITY = <LOGGER_FILE_PRIORITY>
   LOGGER_CONSOLE_ PRIORITY = <LOGGER_CONSOLE_ PRIORITY>
   ```

   The first line set the file name for the logger. The second and third lines set the priority for the file and console logger respectively. This priority is given as a number between 1 and 5.

2) <products_file> – This file defines the products that the University coffee shop can make.
   File format:

   ```
   Product_1_Name,Ingredient_1,…,Ingredient_n1
   …
   Product_M_Name,Ingredient_1,…,Ingredient_nm
   ```

   Where:
   - *Product_i_Name* – Name of the i'th product ("cappuccino", "tuna salad" etc…).
   - *Ingredient_i* – The i'th ingredient needed to create the product.

3) <suppliers_file> - This file will define the suppliers, their goods and the prices.
   File format:

   ```
   <Supplier_Name>,<Ingredient_Name>,<Ingredient_Price>
   ```

   Where:
   - *<Supplier_Name>* – The name of the supplier.
   - *<Ingredient_Name>* – The name of the Ingredient ("milk", "cheese", etc…).
   - *<Ingredient_price>* – The price of the ingredient.

4) <events_file>– The simulation file that will include a list of events that happened in the shop. (Section 4).

   You may assume that if an input file exists it is valid and consistent. Each line will be separated with a new line character – '\n'.

## 8. Output

The output will be in 2 separate files:

1) **<**LOG_FILE_NAME **>.log** – This file contains all the log output as defined in the assignment.
2) collage.tiff – A collage of all the registered costumers.

## 9. Implementation Requirements

**Representing customer in the system:**

Though they are many ways to represent a customer in the system, we ask you to use the following implementation.

```
class Customer
{
	public:
			virtual double computeProductPrice(double
			originalPrice)=0;
};
```

From which you will in inherit two classes

a) VipCustomer
b) RegularCustomer

We also ask you to store all the customers in the system using the following class

```
class Customers
{
	private:
			std::vector<Customer *> m_customers;
};
```

Obviously you are allowed to add data members and member function to both classes to suit your needs.

**Memory leak**

This requirement is basic and mandatory. Memory leaks occur when the program fails to release memory that is no longer needed. A memory leak can diminish the performance of the computer by reducing the amount of available memory. Eventually, in the worst case, too much of the available memory may become allocated and all or part of the system or device stops working correctly, the application fails, or the system slows down. To avoid leaks, you need to release the memory when you finish using it. Basically, every time you use ***new***, you must use ***delete***.

You must check your implementation using Valgrind. Valgrind is an easy to use tool that is already installed in the lab on Linux platforms. For easy start read this. Please notice, that the Memcheck in Valgrind is not perfect; it occasionally produces false positives. This may happen when faced with advanced memory management techniques, as memory pool, reference counting, etc., as done in OpenCV and POCO. For this reason, when checking you code for memory leaks, we will look only at the fields ***definitely lost***, and ***indirectly lost*** fields in Valgrind's report. They should be equal to 0.

Please note that displaying an image to screen, while waiting for input from the user *may* cause Valgrind to report a memory leak. To avoid this, when you submit your program for grading, make sure it is not displaying images to screen. Of course, you are allowed to *debug* you code using this utility.

## 10. Submission

Your submission should be in a single zip file called "student1ID-student2ID.zip". The files in the zip should be set in the following structure:

*src/*
*include/*
*bin/*
*makefile*

***src*** directory should include all .cpp files that are used in the assignment.
***include*** directory should include all the header (.h or *.hpp) files that are used in the assignment.
***bin*** directory where the compiled objects (the .o files and the assignment executable) should be placed when compiling your source files (should be empty).
***makefile*** should compile the cpp files into the bin folder and create an executable and place it also in the bin folder (should be named *sim_coffee_shop*).

The *makefile* should properly compile on the department computers, the objects should be compiled with the following flags:

*–Wall –Weffc++ –g*

The submissions must be made in pairs.

After you submit your file to the submission system, re-download the file that you have just submitted, extract the files and check that it compiles.

## 11. Grading

Although you are free to work wherever you please, assignments will be checked and graded on CS Department Lab Computers - so be sure to test your program thoroughly on them before your final submission. "But it worked on my Windows/Linux based home computer" will not earn you any extra points if your code fails to execute at the lab.

## 12. Questions

All questions regarding the assignment should be published in the assignment forum. Please search the forum for similar questions before publishing a new one. Question by mail regarding the assignment will be redirected to /dev/null. Please note that before using the forum you must register here.

## 13. Delays

In case of reserve duty (aka Sherut Miluim), illness, etc. please send an email to the course mail spl141@cs.bgu.ac.il.The mail should include partners name and IDs, explanation and certification for your illness / Miluim. We will not answer other mail in the course mail.

## 14. Course policy about appeals

**על פי מדיניות הקורס אין אפשרות לערער על עבודות**

So please make sure you made all the necessary QA on your work *before* submitting it.