### **1. Supermarket inventory:**

There are several types of products, each having a known, constant, unit price. In the begining, we know the quantity of each product.

We must keep track of the quantity of each product, the amount of money (initially zero), and the list of bills, corresponding to sales. Each bill is a list of items and quantities sold in a single operation, and their total price.

We have sale operations running concurrently, on several threads. Each sale decreases the amounts of available products (corresponding to the sold items), increases the amount of money, and adds a bill to a record of all sales.

From time to time, as well as at the end, an inventory check operation shall be run. It shall check that all the sold products and all the money are justified by the recorded bills.

class Product{ //This class store all information related to a Product

private:

string name; //every product has a name

const double price;//a price which is constant (like is specified in the statement of the problem)

int quantity;//and a quantity

public:

Product();

Product(string name,const double price,int quantity);

string getName();

void setName(string name);

double getPrice();

int getQuantity();

void setQuantity(int quantity);

};

class Sale{ //This class store informations regarding a Sale

private:

Product\* product; //Every Sale has a product which it want to buy

int quantity; //and the required quantity

//For example me as a customer I want to buy 3 apple->Sale(Product(Apple...),3);

public:

Sale(){};

Sale(Product\* prod,int quantity);

void setQuantity(int quantity);

void setProduct(Product\* prod);

int getQuanity();

Product\* getProduct();

friend ostream& operator<<(ostream& os, const Sale& sale){

os <<"SALE "<<sale.product<< " quantity:" << sale.quantity;

return os;

};

};

In the main function I chose a certain number of products and for every product I randomly generated the price, the quantity and the name. All this products will be kept in a local vector call *products.* I also have a local list associated with the precious one which stored mutexes associated with every element, so for the *products[i]* I had an associated mutex *mutexes[i].* There is 2 types of threads *Client thread* and *Inventory thread*. The first one is responsible for doing a Sale and the second one simple make the inventory. The number of thread for both type is provided by the user.

For the Client thread there are two different implementation that differ in the level of concurrency. In both implementation a thread randomly generated the number of operation it will do and for every operation it will choose also randomly the product and the quantity trade.

In the first implementation for every transaction a thread make a lock over the entire list of products and after that do its job.

In the second one I choose a more concurrent approach and at every time a thread do a lock just over the product that it wants to trade right now and do a general lock when it modifies the general sum of many. This approach is not very elegant because it uses more lock but it provides a more concurrent approach.

The inventory thread has just one implementation because it need to do a general lock over the all element of the list in order to check if the transaction were done right.

This program right down important facts about the ran test in the file results.out where you can see different test with the time result.