Project Report: Product Sales Analysis System

Course: Introduction to Programming
Assignment Number: 1
Student Name: Nureddin Can ERDEĞER
Student ID: B231210041
Course Group: C

Objective

The goal of this project is to simulate a trading system for a company selling 20 different products. The program models the products, their sales, expenses, and profit generation, including adjustments for stock, sales price, VAT, and periodic price changes. The program generates monthly and yearly reports based on random sales data.

Code Overview

The program involves creating a product structure with attributes such as product name, stock quantity, purchase price, sales price, VAT rate, etc. The code simulates monthly sales and expenses for a year while adjusting the product price increase frequency and rate based on user input.

1. Product Structure Definition

First, we define a structure product to represent each product in the system:

```
22  // Create a product structure data type that includes members such as name, stock quantity, buying price, sales price, VAT rate, etc.
23  struct product {
24      string nameOfProduct;
25      int stockCount;
26      float buyingPrice;
27      float salePrice;
28      int rateOfVat;
29  };
```

This structure stores:

- nameOfProduct: The name of the product.
- stockCount: The quantity of the product in stock.
- buyingPrice: The price at which the company buys the product.
- salePrice: The price at which the company sells the product (including VAT).
- rateOfVat: The VAT rate applied to the product.

We also define an expenses structure for the monthly fixed expenses:

```
31  // Here we create a structure data type that contains the monthly money. Students were told that the monthly expenses are fixed.
32  struct expenses {
33      float rent;
34      float staffSalaries;
35      float electric;
36      float water;
37      float otherExpenses;
38  };
```

2. Product Information and Random Sale Price Calculation

The products are initially stored in a string companyProducts, containing details about each product, such as name, stock quantity, buying price, and VAT rate. The string is then parsed and the values are transferred to an array of product structures:

```
48      /* Let's assume that the products sold by the company are given in the string firmaurun="mouse,20,51.00,20,microphone,
49         89.50,mobile phone,66.40,desktop computer,45.09,laptop,96.50,…" */
50      string companyProducts =
51          "Mouse,20,51.00,20,"
52          "Microphone,20,89.50,20,"
53          "Mobile Phone,20,66.40,20,"
54          "Desktop Computer,20,45.09,20,"
55          "Laptop,20,96.50,20,"
56          "Keyboard,20,44.50,20,"
57          "Smartwatch,20,35.60,20,"
58          "Headphone,20,30.60,20,"
59          "Television,20,70.80,20,"
60          "Tablet,20,40.40,20,"
61          "Game Console,20,35.80,20,"
62          "Drone,20,24.90,20,"
63          "Printer,20,53.00,20,"
64          "Mousepad,20,10.50,20,"
65          "Webcam,20,14.50,20,"
66          "Vacuum Cleaner,20,14.30,20,"
67          "Monitor,20,35.00,20,"
68          "Speaker,20,25.50,20,"
69          "Dishwasher,20,65.30,20,"
70          "Washing Machine,20,76.50,20,";
```

Using a loop, the string is split by commas to assign values to the relevant members of the product structure.

Next, the sales price for each product is calculated based on its buying price and a random percentage increase between 30% and 50%:

```
101     // Let the selling price be determined randomly as the buying price of the product + (minimum 30% - maximum 50% of the buying price).
102     for (int i = 0; i < 20; ++i) {
103         float randomSaleRate = 30 + (rand() % 21);
104         float priceWithoutVat = products[i].buyingPrice + (products[i].buyingPrice * randomSaleRate / 100.0F); // Price without vat
105         products[i].salePrice = priceWithoutVat * (1 + products[i].rateOfVat / 100.0F); // Price with vat
106     }
```

This formula ensures that the sales price is at least 30% higher than the buying price, with a maximum increase of 50%.

3. User Input for Price Increase Frequency and Rate

The user is prompted to enter the frequency and rate at which product prices will increase. The frequency options are: monthly, every two months, every three months, or every four months. The increase rate is also provided by the user as a percentage:

```
111     /* Buying price: The user must give the buying price increase once a month,
112        once every two months, once every three months, or once every four months. The increase rate must also be given */
113     int frequency;
114     float increaseRate;
115     do {
116         cout << "Please enter the frequency of price increases (1: monthly, 2: every 2 months, 3: every 3 months, 4: every 4 months): ";
117         cin >> frequency;
118         if(frequency < 1 || frequency > 4) {
119             cout << "Invalid frequency! Please enter a value between 1 and 4." << endl;
120         }
121     } while (frequency < 1 || frequency > 4);
122
123     do {
124         cout << "Please enter the rate of price increase (e.g., 5 for 5%): ";
125         cin >> increaseRate;
126         if(increaseRate <= 0 ) {
127             cout << "Invalid rate! Increase rate must be positive!" <<  endl;
128         }
129     } while (increaseRate <= 0);
```

The price increase is applied according to the user's inputs, and the sales price is recalculated for each month.

4. Monthly Sales Simulation and Profit Calculation

Each month, a random percentage between 60% and 80% of the stock is sold. The amount sold is calculated and the corresponding profit is determined by subtracting the cost of goods sold from the revenue:

```
152         for (int i = 0; i < 20; ++i) {
153             // Each product is randomly selected and sold on a monthly basis, with a minimum of 60% and a maximum of 80% of the stock.
154             float randomRate = 60 + (rand() % 21);
155             int soldAmount = products[i].stockCount * (randomRate / 100.0F);
156             yearlySales[i] += soldAmount;
```

The revenue is computed by multiplying the number of units sold by the sales price, and the cost is calculated using the buying price. The VAT is considered in the turnover calculation:

```
158     // Profit and cost
159     float income = soldAmount * products[i].salePrice; // Income
160     float cost = soldAmount * products[i].buyingPrice; // Cost
161     float turnover = income - cost; // Turnover
162     float profit = turnover - turnover * 16.67 / 100;
163     // We added 20% VAT, we subtract it again. We wrote 16.67% because 20% of 100 is 120, 16.67% of 120 is 20.
```

The profit for each product is computed by subtracting the turnover cost from the revenue.

Additionally, the stock is updated by considering both the remaining stock and a randomly calculated restocking amount based on the sold quantity:

```
171     // Stock quantity calculation; 2 * remaining stock + randomly determined as min 70% - max 100% of the sold quantity.
172     products[i].stockCount = (2 * products[i].stockCount) + (soldAmount * (70 + (rand() % 31)) / 100.0F);
```

5. Monthly Report Generation

For each month, the program calculates the total profit, identifies the most and least sold products, and the products with the highest and lowest profit. These statistics are displayed in a monthly report:

```
203     // Monthly report
204     cout << "Monthly Profit: " << monthlyProfit << " USD" << endl;
205     cout << "Most Sold Product: " << products[mostSoldIndex].nameOfProduct
206         << " (" << maxSoldAmount << " units)" << endl;
207     cout << "Least Sold Product: " << products[leastSoldIndex].nameOfProduct
208         << " (" << minSoldAmount << " units)" << endl;
209     cout << "Highest Profit Product: " << products[mostSoldIndex].nameOfProduct
210         << " (" << maxProfit << " USD)" << endl;
211     cout << "Lowest Profit Product: " << products[leastSoldIndex].nameOfProduct
212         << " (" << minProfit << " USD)" << endl;
```

The program also generates an updated list of products with their stock and buying prices for the next month.

6. Year-End Summary

After simulating sales for all 12 months, the program calculates the total yearly profit and identifies the best and worst-performing products. The program then prints the final report, including:

- Total yearly profit
- The most sold product
- The least sold product
- The highest profit-generating product
- The lowest profit-generating product

```
254     cout << "\n--- Year-End Summary ---" << endl;
255     cout << "Total Yearly Profit: " << yearlyProfit << " USD" << endl;
256     cout << "Most Sold Product of the Year: " << products[mostSoldYearlyIndex].nameOfProduct << " (" << yearlySales[mostSoldYearlyIndex] << " units)" << endl;
257     cout << "Least Sold Product of the Year: " << products[leastSoldYearlyIndex].nameOfProduct << " (" << yearlySales[leastSoldYearlyIndex] << " units)" << endl;
258     cout << "Highest Yearly Profit Product: " << products[maxProfitIndex].nameOfProduct << " (" << maxProfit << " USD)" << endl;
259     cout << "Lowest Yearly Profit Product: " << products[minProfitIndex].nameOfProduct << " (" << minProfit << " USD)" << endl;
```

7. Summary and Conclusion

This program simulates the operations of a trading company, including sales, price adjustments, stock management, and profit calculation. By incorporating random elements for sales and stock updates, the program provides a dynamic and realistic model of a company's financial performance over the course of a year.

Conclusion

The project successfully simulates the monthly sales, stock updates, and profit/loss analysis for 20 different products over a one-year period. The generated reports provide valuable insights into product performance, including the most and least sold products, as well as the products that generated the highest and lowest profits.