Julius Ranoa
CSC 121 001 Computer Science I
Homework

Chapter 13 Advanced File and I/O Operations
Programming Challenge Qn. 13 and 14 Corporate Sales Data I/O.

Screenshots.

(1) When *source* provided doesn't refer to an existing file.

```
Loading all records from hwdata.dat...

This program cannot use hwdata.dat
That file, if it exists, will be overwritten.

Please enter the information for the new file.Please enter informa
  Division: North
  |  Enter sales for Quarter 1: 101.45
  |  Enter sales for Quarter 2: 237.42
  |  Enter sales for Quarter 3: 125.76
  |  Enter sales for Quarter 4: 342.45

  Division: South
  |  Enter sales for Quarter 1: 128.64
  |  Enter sales for Quarter 2: 345.76
  |  Enter sales for Quarter 3: 234.87
  |  Enter sales for Quarter 4: 326.5

  Division: East
  |  Enter sales for Quarter 1: 235.98
  |  Enter sales for Quarter 2: 356.87
  |  Enter sales for Quarter 3: 478.89
  |  Enter sales for Quarter 4: 235.78

  Division: West
  |  Enter sales for Quarter 1: 543.32
  |  Enter sales for Quarter 2: 295.56
  |  Enter sales for Quarter 3: 395.23
  |  Enter sales for Quarter 4: 406.32

The file, hwdata.dat, has been saved. Please restart the program.
```

(2) When *source* provided refers to an existing file.

```
Loading all records from hwdata.dat...
Doing analysis. Here are the results.

The total yearly sales per quarter across all divisions:

  Q  Sales
  -  ------------
  1       1009.39
  2       1235.61
  3       1234.75
  4       1311.05

The total yearly sales per division across all quarters:

  Division  Sales
  --------  ------------
  North           807.08
  South          1035.77
  East           1307.52
  West           1640.43

The following are aggregate stats.

  Total Yearly Sales: 4790.80
  Average Yearly Sales: 299.42
  Highest Corporate Quarter: Quarter 4 with $1311.05
  Lowest Corporate Quarter: Quarter 1 with $1009.39

The program is done.
```

main.cpp

```
#include "Driver.h"
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    Driver d;
    string source = "hwdata.dat";

    cout << "Loading all records from " << source << "..." << endl;
    d.deserializeAll(source);
    if (!d.good()) {
        cout << endl;
        cout << "This program cannot use " << source << endl;
        cout << "That file, if it exists, will be overwritten. " << endl << endl;
        cout << "Please enter the information for the new file.";
        d.getInput();
        d.serializeAll(source);
        cout << "The file, " << source << ", has been saved. Please restart the program.";
        return -1;
    }

    d.doCalculations();
    cout << "Doing analysis. Here are the results." << endl << endl;

    cout << "The total yearly sales per quarter across all divisions: " << endl << endl;
    cout << "  Q  Sales " << endl;
    cout << "  -  ------------" << endl;
    cout << fixed << showpoint;
    for (numValuePair nv : d.getTotalQuarterlySales()) {
        cout << "  ";
        cout << setw(1) << setprecision(0) << nv.num << "  ";
        cout << setw(12) << setprecision(2) << right << nv.value << endl;
    }
    cout << endl;

    cout << "The total yearly sales per division across all quarters: " << endl << endl;
    cout << "  Division  Sales " << endl;
    cout << "  --------  ------------" << endl;
    cout << fixed << showpoint;
    for (nameValuePair nv : d.getTotalDivisionSales()) {
        cout << "  ";
        cout << setw(8) << left << nv.name << "  ";
        cout << setw(12) << setprecision(2) << right << nv.value << endl;
    }
    cout << endl;

    cout << "The following are aggregate stats. " << endl << endl;
    cout << "  Total Yearly Sales: " << setprecision(2) << d.getTotalSales() << endl;
    cout << "  Average Yearly Sales: " << setprecision(2) << d.getAverageSales() << endl;
    cout << "  Highest Corporate Quarter: "
         << "Quarter " << d.getHighestCorporateQuarter().num
         << " with $" << d.getHighestCorporateQuarter().value << endl;
    cout << "  Lowest Corporate Quarter: "
     << "Quarter " << d.getLowestCorporateQuarter().num
```

```
        << " with $" << d.getLowestCorporateQuarter().value << endl;
    cout << endl;

    cout << "The program is done." << endl;
    // cout << "For reference, here are all the records saved in " << source << endl << endl;
    // d.printAll();

    return 0;
}
```

CorporateSales.h

```
#ifndef HOMEWORK_CH13_QN13_14_CORPORATESALES_H
#define HOMEWORK_CH13_QN13_14_CORPORATESALES_H

#include <string>
#include <fstream>

class CorporateSales {

private:
    std::string divisionName;
    int quarter;
    double sales;

public:
    CorporateSales() {
        divisionName = "North";
        quarter = 1;
        sales = 0.0;
    }
    CorporateSales(std::string d, int q, double s) {
        divisionName = d;
        quarter = q;
        sales = s;
    }

    // Getters
    std::string getName() const { return divisionName; }
    int getQuarter() const { return quarter; }
    double getSales() const { return sales; }

    // Setters
    void setName(std::string n) { divisionName = n; }
    void setQuarter(int q) { quarter = q; }
    void setSales(double s) { sales = s; }

    void serialize(std::ofstream&);
    void deserialize(std::ifstream&);
    void print();
};


#endif //HOMEWORK_CH13_QN13_14_CORPORATESALES_H
```

CorporateSales.cpp

```cpp
#include "CorporateSales.h"
#include <iostream>
using namespace std;

void CorporateSales::serialize(ofstream &outFile) {
    // Order of Serialization: Quarter > Sales > Name

    outFile.write(reinterpret_cast<char *>(&quarter), sizeof(quarter));
    outFile.write(reinterpret_cast<char *>(&sales), sizeof(sales));

    int l = divisionName.length();
    outFile.write(reinterpret_cast<char *>(&l), sizeof(l));
    outFile.write(divisionName.data(), l);
}


void CorporateSales::deserialize(ifstream &inFile) {
    const int BUFFER_SIZE = 256;
    static char buffer[256];

    inFile.read(reinterpret_cast<char *>(&quarter), sizeof(quarter));
    inFile.read(reinterpret_cast<char *>(&sales), sizeof(sales));

    int l = 0;
    inFile.read(reinterpret_cast<char *>(&l), sizeof(l));
    inFile.read(buffer, l);
    buffer[l] = '\0';

    divisionName = buffer;
}
```

Driver.h

```cpp
#ifndef HOMEWORK_CH13_QN13_14_DRIVER_H
#define HOMEWORK_CH13_QN13_14_DRIVER_H

#include "CorporateSales.h"
#include <vector>

struct numValuePair {
    int num;
    double value;
};

struct nameValuePair {
    std::string name;
    double value;
};

class Driver {

private:
```

```
    bool isGood = false;
    std::vector<CorporateSales> records;
    std::vector<numValuePair> totalQuarterlySales;
    std::vector<nameValuePair> totalDivisionSales;
    double totalSales;
    double averageSales;
    numValuePair lowestCQuarter;
    numValuePair highestCQuarter;

public:
    void serializeAll(std::string filename);
    void deserializeAll(std::string filename);
    bool good() { return isGood; }
    void printAll();

    void randomize();

    void getInput();
    void doCalculations();
    void calcTotalQuarterlySales();
    void calcTotalDivisionSales();
    void calcTotalnAverageSales();
    void findExtrema();

    std::vector<CorporateSales> getRecords() { return records; };
    std::vector<numValuePair> getTotalQuarterlySales() {
        return totalQuarterlySales;
    };
    std::vector<nameValuePair> getTotalDivisionSales() {
        return totalDivisionSales;
    };
    double getTotalSales() { return totalSales; };
    double getAverageSales() { return averageSales; };
    numValuePair getLowestCorporateQuarter() { return lowestCQuarter; };
    numValuePair getHighestCorporateQuarter() { return highestCQuarter; };
};


#endif //HOMEWORK_CH13_QN13_14_DRIVER_H
```

Driver.cpp

```
#include "Driver.h"
#include <iostream>
#include <iomanip>
using namespace std;

void Driver::serializeAll(std::string filename) {
    ofstream file(filename, ios::binary);
    if (file) {
        for (CorporateSales cs : records) {
            cs.serialize(file);
        }
    }
}
```

```
void Driver::deserializeAll(std::string filename) {
    ifstream file(filename, ios::binary);
    if (file) {
        while(file.good() && file.peek() != EOF) {
            CorporateSales cs;
            cs.deserialize(file);
            records.push_back(cs);
        }
        isGood = true;
    } else isGood = false;
}

void Driver::getInput() {
    vector<string> divisions = {
        "North", "South", "East", "West"
    };
    vector<int> quarters = {
        1, 2, 3, 4
    };
    double temp;

    records.clear();
    cout << "Please enter information as prompted." << endl;
    for (string div : divisions) {
        cout << "  Division: " << div << endl;
        for (int q : quarters) {
            cout << "  |  Enter sales for Quarter " << q << ": ";
            cin >> temp;
            records.push_back(CorporateSales(div, q, temp));
        }
        cout << endl;
    }
}

void Driver::printAll() {
    cout << "  "
         << "Division " // 9
         << "Q " // 2
         << "Sales     " << endl; // 10
    cout << "  "
         << "-------- "
         << "- "
         << "----------" << endl;

    cout << fixed << showpoint;
    for (CorporateSales cs : records ) {
        cout << "  ";
        cout << setw(8) << left << cs.getName() << " ";
        cout << setw(1) << cs.getQuarter() << " ";
        cout << setw(10) << setprecision(2) << right << cs.getSales();
        cout << endl;
    }
}

void Driver::randomize() {
    vector<string> divs = { "North", "South", "East", "West" };
    vector<int> qs = {1, 2, 3, 4};
```

```
    srand(time(NULL));
    for (string div : divs) {
        for (int q : qs) {
            records.push_back(
                CorporateSales(div, q, rand() % 1000 + 1)
            );
        }
    }
}

void Driver::doCalculations() {
    calcTotalQuarterlySales();
    calcTotalDivisionSales();
    calcTotalnAverageSales();
    findExtrema();
}

void Driver::calcTotalQuarterlySales() {
    totalQuarterlySales = {
        {1, 0}, {2, 0}, {3, 0}, {4, 0}
    };

    for (CorporateSales cs : records) {
        switch (cs.getQuarter()) {
            case 1:
                totalQuarterlySales[0].value += cs.getSales();
                break;
            case 2:
                totalQuarterlySales[1].value += cs.getSales();
                break;
            case 3:
                totalQuarterlySales[2].value += cs.getSales();
                break;
            case 4:
                totalQuarterlySales[3].value += cs.getSales();
                break;
        }
    }
}

void Driver::calcTotalDivisionSales() {
    totalDivisionSales = {
        {"North", 0}, {"South", 0}, {"East", 0}, {"West", 0}
    };

    for (CorporateSales cs : records) {
        for (int i = 0; i < totalDivisionSales.size(); i++) {
            if (cs.getName().compare(totalDivisionSales[i].name) == 0) {
                totalDivisionSales[i].value += cs.getSales();
                break;
            }
        }
    }
}

void Driver::calcTotalnAverageSales() {
```

```
    totalSales = 0;
    for (CorporateSales cs : records) {
        totalSales += cs.getSales();
    }
    averageSales = totalSales / records.size();
}

void Driver::findExtrema() {
    lowestCQuarter = highestCQuarter = totalQuarterlySales[0];
    for (int i = 1; i < totalQuarterlySales.size(); i++) {
        if (totalQuarterlySales[i].value < lowestCQuarter.value) {
            lowestCQuarter = totalQuarterlySales[i];
        } else if (totalQuarterlySales[i].value > highestCQuarter.value) {
            highestCQuarter = totalQuarterlySales[i];
        }
    }
}
```